POLITECNICO DI TORINO

Corso di Laurea in Ingegneria Informatica

Tesi di Laurea

Evolution of Digital Identity in Europe: Experimenting with the eIDAS 2.0 Framework and the EU Digital Identity Wallet

Relatore
Prof.ssa Diana Gratiela Berbecaru

Alessandro Vanella

2024-2025

To my family, who
supported me all these years

Summary

Over the past decade, the Self-Sovereign Identity model has gained prominence and is expected to become a key element in the authentication and identification processes of European citizens when accessing digital services. Driven by the recent eIDAS 2.0 Regulation, which updates and expands the first eIDAS Regulation by incorporating new concepts and principles aligned with the modern digital landscape, a new framework is being developed to ensure the secure use of digital identity in Europe. This framework places particular emphasis on privacy, user control over personal data and interoperability across different national systems.

This thesis work analyses both the the innovations introduced by eIDAS 2.0 and the SSI model, comparing it with previous approaches and highlighting its strengths and challenges. Then, it explores the EU Digital Identity Wallet infrastructure and the Architecture and Reference Framework (ARF) on which its implementation is based, outlining the role of each component within the ecosystem and its use cases. Particular attention is given to the security protocols employed for the wallet's two main tasks, credential issuance request and credential sharing, highlighting the mechanisms that ensure the integrity and confidentiality of communications between the involved entities.

Finally, to provide a deeper analysis of the flow of both tasks and demonstrate how the credential issuance process can be reproduced and tested in a controlled environment, all the services involved will be deployed locally and configured to enable their interaction; an instance of the EUDI Wallet, a local Service Provider and a pair of eIDAS nodes will be set up on local devices, simulating a real-world environment for authentication and credential exchange. This experimental setup will allow an evaluation of the dynamics of interaction between the entities involved, demonstrating the technical feasibility of the framework and verifying its compliance with the security, privacy and interoperability principles promoted by eIDAS 2.0.

Contents

Chapter 1

Introduction

1.1 The Evolution of Digital Identity

In the last decade, the concept of digital identity has become a hot topic, evolving along with a rapid digital transformation. People's reliance on online services has grown significantly, increasing the need to have more secure and efficient authentication mechanisms that simplify daily operations for individuals, businesses and public administrations. From accessing government portals for tax declarations or healthcare services to performing online financial operations and even trusting e-commerce platforms, digital identity is the key element in securing digital interactions within the system and ensuring that everyone is able to use it. Prior to the establishment of a unified European framework, digital identity systems were highly fragmented: countries developed their own approach based on their national regulations, technological infrastructure and needs, leading to significant differences in security standards and authentication mechanisms between each other. Some countries introduced centralized national identity systems, while others relied on private identity providers or sector-specific solutions; this lack of harmonization made it difficult for citizens to use their digital identity outside national borders, strictly limiting the possibility of cross-border authentication when a citizen needs to access a service from a foreign country.

Before 2014, the lack of a common regulatory foundation prevented European citizens from authenticating seamlessly across borders. A user attempting to access a public service in another Member State had to navigate complex procedures depending on the service, often requiring physical documents or country-specific authentication mechanisms. The European Union addressed these challenges with the introduction of the EU Regulation 910/2014, named eIDAS, which established a legal and technical framework to facilitate cross-border authentication and the recognition of national electronic identification (eID) schemes and trust services.

With the entry into force of eIDAS, Member States were required to recognize each other's notified eID schemes, allowing citizens to use their national digital credentials to access foreign online services. Additionally, the regulation introduced legally binding Trust Services, such as electronic signatures, seals and timestamps, to enhance the security and integrity of digital transactions and establishing new standards for these electronic operations. However, the impact of eIDAS has not been uniform across the whole Europe: while some countries proactively adopted new eID schemes compatible with the regulation, others merely complied with the minimum requirement, recognizing foreign eIDs without changing their own national systems, and leading to an incomplete implementation of eIDAS principles. In addition, over the years, the limitations of the eIDAS Regulation have become evident, as it has struggled to adapt to the constant evolution of digital technologies. First, the Regulation is primarily focused on the public sector, leaving private sector adoption optional, which significantly limits its usability in everyday digital interactions. Furthermore, while eIDAS does not rely on a single centralized authority, it still follows a federated identity model, where identity providers and national authorities retain control over user credentials. This structure does not grant individuals full control over their digital identity, despite providing a higher degree of interoperability than previous national approaches.

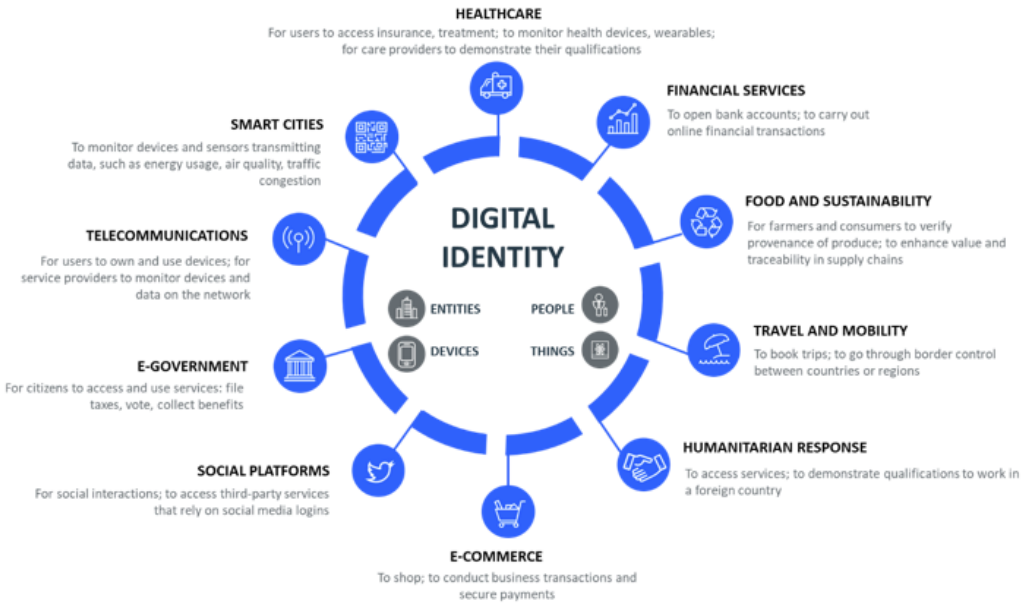

Figure 1.1. The whole spectrum of services that individuals can interact with, by identifying themselves using their own Digital Identity.[1]

These limitations have led to the need for a more advanced identity framework that can incorporate the principles of the more recent Self-Sovereign Identity model. Unlike federated identity systems, Self-Sovereign Identity shifts control entirely to individuals, allowing them to manage directly their credentials and share their identity attributes without relying on an external authority. In a Self-Sovereign Identity system, credentials are issued by trusted entities but are stored directly in a digital wallet held by the user, who can selectively disclose only the necessary attributes when proving their identity to a third party. This approach enhances privacy, security and user autonomy, reducing dependency on centralized identity providers while ensuring cryptographic verification mechanisms to prevent fraud or misuse.

The Self-Sovereign Identity model lays the foundation for the eIDAS 2.0 Regulation, entering into force in 2024, a revision of the whole first Regulation to adapt it to the current digital landscape, aiming at a single European digital identity system to achieve the previous goal of full interoperability between Member States. Aside from that, eIDAS 2.0 introduces new trust services, and extends obligations and rights also to the private sector, involving more individuals and companies in the new environment.

Lastly, eIDAS 2.0 promotes the use of the European Digital Identity Wallet. This application, that will be available to every European citizen over the next years, lets users manage and share their digital credentials, both online and offline, identifying themselves remotely and/or in proximity as it would be done using the corresponding physical identity document. While still under development, supported by the feedback of private companies that work to improve wallet interactions within specific use cases, a technical document named Architecture and Reference Framework is also drafted, which includes specifications and guidelines for Member States to build a wallet that can be compatible with the EU Digital Identity Wallet infrastructure, promoting interoperability with other models built on the same framework.

1.2 Thesis objectives

This thesis is structured into two main phases: a theoretical study of the evolution of the concept of Digital Identity over the last 10 years, and a practical analysis of the EU Digital Identity Wallet

functionalities, in a test environment.

First phase gives a detailed review of the prevailing identity models during the time frame taken into consideration and an analysis of both eIDAS Regulations, notifying differences, improvements and challenges that they have introduced in their related technological scenario. Special attention is given to eIDAS 2.0, the current Regulation, on which the European Commission relies to achieve the European Digital Decade targets for 2030. In order to make it possible that 100% of European citizens can have access to at least one digital identity system by 2030[2], the European Commission, in collaboration with eIDAS expert groups and private associations, began developing the EU Digital Identity Wallet: this framework allows Member States to build their own Wallet solution tailored to their national infrastructures and needs while ensuring interoperability with services offered by other countries. The structure of the application prototype is then studied, alongside the Architecture and Reference Framework, a set of common standards and technical specifications that must be used for the implementation of a national Wallet solution by Member States in order to preserve interoperability with other solutions provided by other Members.

The second phase of the thesis investigates how the EU Digital Identity Wallet facilitates secure identity verification through two primary processes: verifiable credential issuance and verifiable credential presentation. The study aims to analyse the execution flow behind these processes, focusing on the protocols that ensure the integrity and confidentiality of identity exchanges. To gain deeper insights into the inner workings of the Wallet, a local test environment was created, allowing for a detailed examination of both flows across every component involved in the two processes. Specifically, the EU Digital Identity Wallet interacts with a local service provider that allows the issuance of credentials from a fictitious country and redirects the request across an eIDAS node to a local demo identity provider. The goal is to assess whether the proposed architecture and protocols adequately address security concerns and to explore the potential integration of new identity providers within the eIDAS 2.0 framework.

A brief description of the content of each chapter is here reported:

- Chapter 2 provides an overview of key concepts and past solutions that have become stepping stones for the current digital landscape. It begins with an introduction to digital identity, explaining its core components, attributes, and different representations in online environments. Following this, digital identity models are compared with each other, highlighting the differences and challenges they bring to the scene. Then, eIDAS Regulation is introduced as a way to approach some of these challenges, describing its key components and limitations, and showing also how it affected the digital identity integration across Europe.

- Chapter 3 explores the decentralized identity model and the Self-Sovereign Identity concept, focusing on how it made inadequate the first eIDAS Regulation to the current digital scenario. As a consequence, in this chapter is also presented eIDAS 2.0, comparing it to the old Regulation and pointing out the innovations that it brings: particular attention is paid to the European Digital Identity Wallet, as a means to implement a solution that incorporates both the Self-Sovereign Identity model and eIDAS 2.0 principles. Its architecture is described, following the guidelines defined in the Architecture and Reference Framework. Additionally, it analyses real-world use cases, illustrating how the Wallet is designed to interact with external entities to provide a secure management of user credentials.

- Chapter 4 details the experimental analysis that is part of the thesis, specifically examining the execution flow of Verifiable Credential issuance and presentation within the local test environment. The analysis includes the requests and responses exchanged between the involved components (wallet, service provider, eIDAS node, identity provider), the formats of transmitted data and security properties behind the interactions.

- Chapter 5 shows how to reproduce, locally, the test environment presented in the thesis, for further testing or developing. First, it explains how to get a working basic configuration of both the service provider and the eIDAS node (which also includes the identity provider), then describes how to make each component interact with the others, creating the exact flow presented in Chapter 4.

- Chapter 6 concludes the thesis work, summarizing how some of the most common attacks are prevented by using the wallet prototype, and which challenges are now solved and which ones must be addressed in the future, to achieve the European Digital Decade targets by 2030.

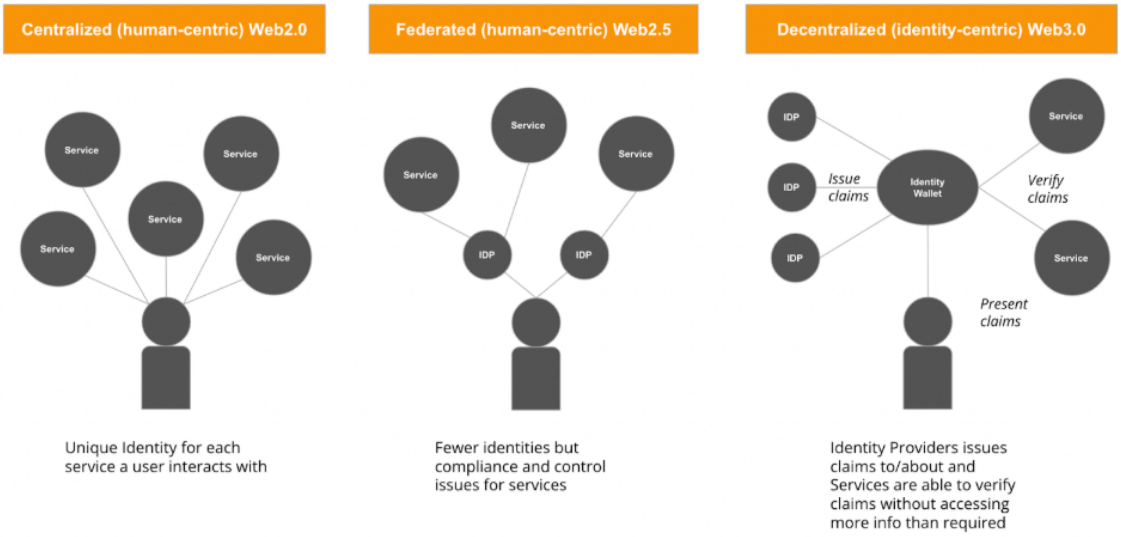

Figure 1.2. Evolution of the identity model.[3]

Chapter 2

Background and Key Concepts

2.1 The Digital Identity concept

Digital Identity is a wide topic that addresses individuals, companies and even devices and software components. With the most generic definition, "it is a one-to-one relationship between an entity and their digital presence"[4]. In other words, a digital identity must be capable of uniquely identifying the entity it represents during any online interaction.

The elements that compose the set of data within a digital identity are named attributes. Attributes may vary, depending on the entity they represent and the context for which they are needed. For humans, common attributes could be:

- Personal information such as name, birth date, country of residence.
- Personalization data, for example an IP address or the GPS location of the individual.
- Credentials that can be used for online authentication, such as passwords, biometric data or cryptographic keys.
- Online tracers that describe user's online behaviour through browser history, cookies, and other interactions across the Internet.

As mentioned above, digital identity can also be applied to other entities outside of human beings. Figure 2.1 shows some of the possible identifiers used as attributes to determine the digital identities of software, hardware, and organizational environments.

2.1.1 Personally Identifiable Information

Referring to human digital identities, attributes can be either inherent to the user or user-generated. While some of them are not enough to uniquely represent an individual on their own, a combination of them can achieve that result. Personally Identifiable Information (PII) represents the subset of digital identity attributes that can be used to identify a person, directly or indirectly, making it a key element in privacy management and data protection. Personally Identifiable Information can be generally classified into two categories:

- Sensitive PII: each one of these attributes is sufficient to uniquely identify a human by itself. National identification numbers (such as social security number or passport number), biometric data or even medical records, fall into this category. Due to their potential, these attributes require strong security measures.
- Non-sensitive PII: they can contribute to identification only if combined with other data. Examples include IP addresses, family name, city of residence or nationality. Usually, most of them are publicly available, because they do not represent a privacy risk if alone.

Nonetheless, the distinction between sensitive and non-sensitive PII is somehow flexible: data that are non-sensitive in some context may be crucial in another, if correlated with other attributes. Combination of ZIP code, birth date and surname, together, can highly reduce the anonymity in a dataset, making possible the identification. In order to mitigate the impact of data breaches and privacy risks, digital identity management systems need to implement data minimization, anonymization and access control techniques: the General Data Protection Regulation [5] (GDPR), adopted by the European Parliament and Council of the European Union in 2016, enforces legal requirements on the processing of personal data to enhance user privacy.

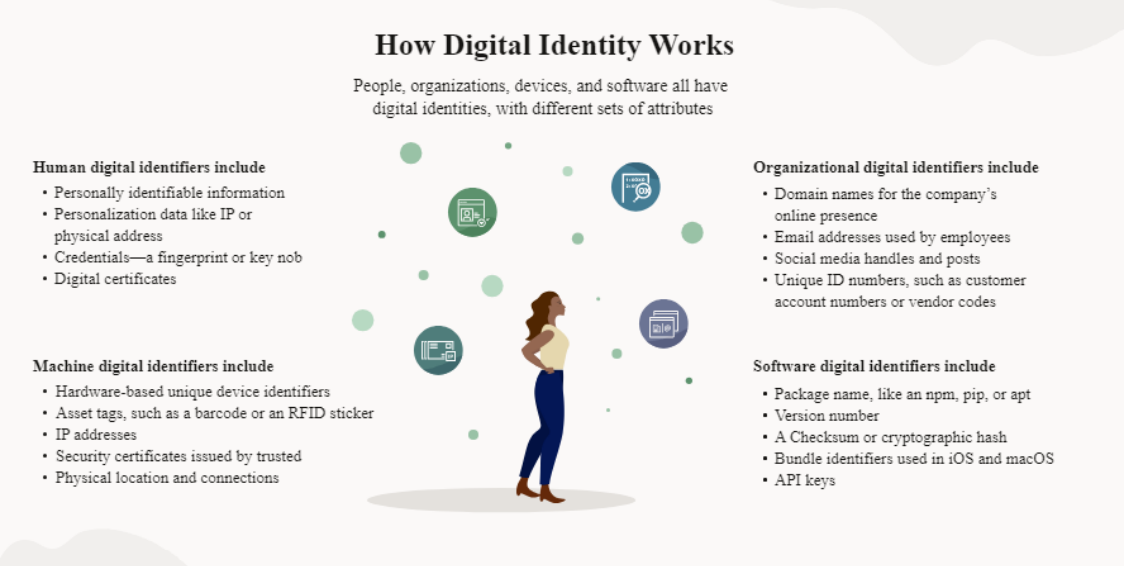

Figure 2.1. The variety of digital identities used for humans, companies, software and devices. [6]

2.1.2 Digital Identity in online services

Although some forms of digital identity can be used, like traditional physical identification methods, to perform in person verification, the most important feature they bring is the possibility of facilitating remote access to services, reducing the time of administrative processes. Many critical services rely on digital identity to verify user identities and enable secure interactions, including:

- Banking and financial services, providing robust security features for the identification process while reducing the burden of Know Your Customer (KYC) and Anti-Money Laundering (AML) procedures that can be automated.
- Healthcare, so that patients can access their personal medical records and manage prescriptions remotely.
- E-commerce platforms, ensuring safe transactions for customers with fraud prevention mechanisms, and enforcing age restriction if necessary.
- E-government services, allowing access for citizens that want to perform operations such as tax declaration or electronic vote, reducing the bureaucracy behind these procedures and increasing accessibility.

Along with the growth of digital interactions over the years, especially when it comes to crucial services, the risk of identity-related cyberattacks increases at the same rate. Phishing attacks carried out by leveraging social engineering, credential theft, session hijacking, or brute-force attacks are some possibilities, all aiming to get control of the victim's sensitive data, potentially causing breaches, frauds and account takeovers.

In order to mitigate these risks, services and digital identity systems in general must rely on Identity and Access Management (IAM) frameworks, which ensure that only legitimate users can access services, successfully identifying and verifying their claim before granting them access to protected resources, and avoiding impersonation from malicious actors. In addition, IAM can enforce policies and regulations behind the identification process, managing user privileges within systems that have complex hierarchies, allowing individuals to access only the resources for which they are entitled. In the next section, the key components of IAM's structured approach to secure identity verification and access control are described.

2.2 Identification, authentication and authorization

Interacting with an on-line service presents new challenges that must be addressed to ensure that the communication is performed in a secure way. In particular, it is mandatory to:

- Identifying the user.
- Authenticate the user.
- Providing the right authorization to the user.

The three steps are strictly interconnected, forming the core of Identity and Access Management systems. The lack of any of these, or their weak implementation, can make the system vulnerable to the attacks mentioned in the previous section, introducing severe security flaws.

Identification is the first step in the process: user makes a claim about themselves, providing an identifier which is unique to the system. Usernames, e-mail addresses, passport number or a MAC address (for devices) are common identifiers that can be recognized within specific systems. Although it is bound to just one entity, an identifier is not sufficient to prove who the claimant is. For example, knowing the e-mail address of another user should not be enough to impersonate them and perform a login procedure, getting access to their account.

Hence, the second step is required, that ensures that the entity claiming the identity is truly who they say they are. This authentication procedure is performed by requesting a specific authentication factor, namely authenticator, from the claimant that is somehow unique to them. Authenticators are categorized as:

- "Something you know", such as a personal PIN, password, or a security question.
- "Something you have", a security token, smart card, an OTP (one-time password) obtained out of band or cryptographic keys.
- "Something you are", referring to biometric data (fingerprints and facial recognition).

More than one authenticator can be requested in order to perform authentication. Multi-Factor Authentication (MFA) strengthens security behind the whole process, making it a lot harder for malicious actors to compromise accounts that are protected by even just two factors (2FA). According to Microsoft [7], 99.9% of its compromised accounts during 2023 did not have MFA enabled, with their systems receiving more than a thousand password brute force attack per second. Because of that, while MFA was initially used just for sensitive services, over the years more and more companies have started integrating it in their systems; recent statistics gathered by a JumpCloud survey [8] show that 87% of companies with more than 10,000 employees are already using MFA, while the adoption rate of smaller companies is just 34% but it is increasing every year, denoting the impact of this security measure. Larger businesses are even making it mandatory for users: Google Cloud plans to have 100% of their accounts protected by MFA by the end of 2025 [9], setting an enforcement timeline depending on the account type (personal, enterprise or reseller accounts).

The user proves control over the authenticator bound to the claim they made by interacting with a Verifier, either an entity within the service they are trying to access or a separate one,

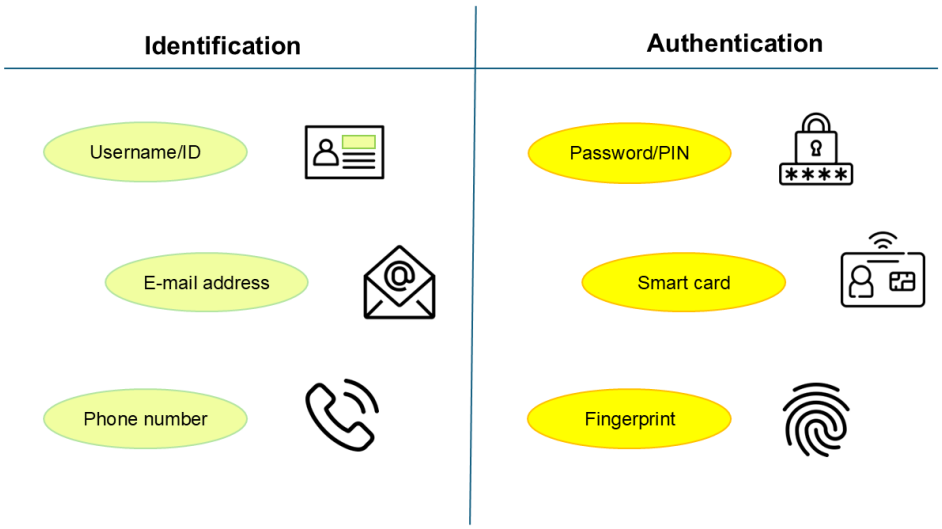

Figure 2.2. Examples of information that can be provided to allow identification and authentication.

commissioned by the Service Provider. The way the Verifier can confirm if the proof received from the user is valid or not heavily depends on the digital identity model used in the system, which will be addressed later in the chapter.

After the user has authenticated themselves, the system must determine which resources can be accessed by them, and which action they can perform; authorization ensures this, following the Principle of Least Privilege (PoLP): users are only given the minimum level of permissions needed to complete the task they are requesting. Access control models define the characteristic used to categorize users inside a system to give them the right privileges:

- Attribute-Based Access Control (ABAC) assigns permissions depending on the value of specific user attributes (age restriction or location).
- Role-Based Access Control (RBAC) grants access based on the role assigned within the system (admin, employee).
- Policy-Based Access Control (PBAC) uses a set of rules to grant privileges to users.

The effectiveness of authentication and access control mechanisms heavily depends on how identities are managed, stored, and shared within a system. Different models have been developed over the years to address these challenges, each of them with its own advantages and drawbacks in terms of security, usability and user control over personal data. The next section explores these identity models, analysing how they influence authentication and authorization processes, and setting the stage for the evolution towards more decentralized identity frameworks.

2.3 Previous digital identity models

Year after year, multiple models have been developed and improved to face the difficulties emerged during those times. Even if different models can have very distinct implementations and approaches, some entities play a fixed role regardless of the model adopted. Therefore, before introducing the most prevalent models, these primary entities are presented:

- User: the owner of a digital identity, who wants to access online services and provides the necessary elements to perform the authentication procedure.

- Service Provider (SP): online platform or application that is in charge of providing specific online services to users. SP must be capable of carrying out identification and authentication, or delegate another entity to handle those processes.
- Identity Provider (IdP): the entity which acts on behalf of SP(s) to perform identification and authentication for users. Providing such a crucial task, a trust relationship must be built between the IdP and the related SP(s).

Interactions between these entities and their responsibilities in the system depends on the model. The following subsections will explore two of the most globally used digital identity models, analysing their structure and adoption, and making some security considerations for both.

2.4 Centralized Identity Model

Most individuals own dozens of online accounts, each one bound to a different service: this is a good representation of the so-called centralized model, in which Service Providers manage all data about users, storing them in their own database. Considering a scenario in which the user Bob wants to use a new e-commerce platform, it is possible to describe which steps are followed to perform the task in a centralized system:

1. Bob registers a new account on the e-commerce website. It will require an identification factor (typically a username), and some personal data, such as his name and e-mail address. In addition, he sets a password to be allowed to authenticate himself on the first login and future sessions.
2. Bob's data are stored in a local database, which is accessible only by the platform administrators. Passwords, instead of being stored in plaintext, are hashed, providing a layer of security.
3. Once the registration is completed, Bob can now access the website using the previously created username and password. In background, credentials provided by Bob are verified against the corresponding pair username-hashed password that is stored in the local database. If they match, Bob is granted access to the service.

Now, suppose that Bob wants to access his personal bank account to verify that he has enough money to perform a purchase on the e-commerce platform. Then, he needs to perform again another login procedure, on the bank website, using a different set of credentials, that are matched against Bob's related data within the bank database. Basically, for each service that is accessed by Bob, different credentials are needed, and each service can only accept as a proof for authentication something that is directly stored in its database. Although the model implementation is very simple, it also leads to some severe issues in terms of security:

- The database represents the single point of failure in the system, making it a honeypot for attackers. Due to an accidental or intentional disclosure of data stored within the database, malicious individuals can gain access to a large amount of user credentials.
- Absence of control by the user over shared data: there is no way to know how data will be stored and processed by each Service Provider, neither what happens if an account gets deleted. The Cambridge Analytica data scandal [10] that emerged in 2018 is a demonstration of this issue, involving the unauthorized collection by Cambridge Analytica of data coming from over 87 million Facebook user profiles for political advertising, without explicit user consent.
- Users have one account for each service, and they should choose different credentials for each one. Considering the multitude of online services that are provided and requested daily, this requires a large amount of usernames and passwords for each user. According to an annual survey taken by NordPass [11] on over 1,500 NordPass users, the number of accounts per user is drastically increasing, jumping from around 80 in February 2020 to over 200 accounts per individual in 2024, including both workplace and personal accounts.

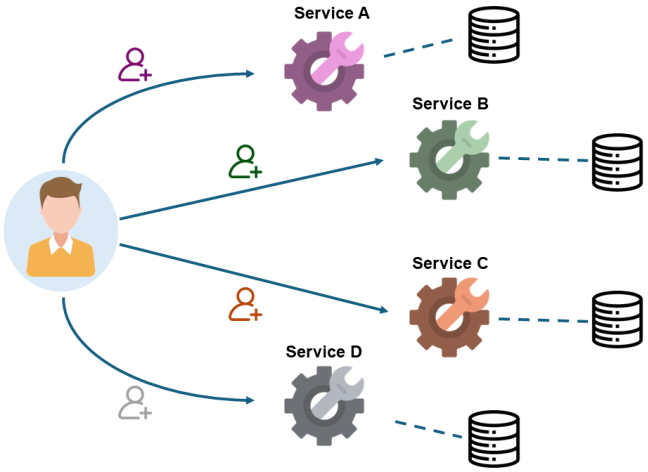

Figure 2.3. Centralized identity model: different services require a distinct user account, and user data are stored and managed inside the service databases.

In conclusion, a centralized approach puts full responsibility in service providers behaviour, which are capable of authenticating users and storing their associated data, acting as an Identity Provider. In return, the model causes fragmentation, since services are independent from each other, and requires users to manage a variety of accounts for every platform, with the risk of discouraging the use of different credentials for each one.

2.5 Federated Identity model

To simplify the user experience and address the weaknesses of the centralized approach, the federated identity model emerged. This model allows users to authenticate once to a single trusted Identity Provider to be granted access to multiple services without needing to log in again for each one. This capability is commonly known as Single Sign-On (SSO), significantly reducing the number of credentials an individual needs to manage and easily implemented on various websites through simple buttons (for example, the common 'Login with Google' button). The model relies upon a trust relationship between the service provider and the identity provider: in particular, the latter performs the authentication procedure on behalf of the former when users request access to the service resources.

This is a common flow regarding a Federated Identity Authentication procedure leveraging SSO:

1. Bob requests the access to an e-commerce platform through its website.
2. The Service Provider redirects the user to the Identity Provider, which is responsible for authenticating the user. The IdP offers an authentication interface that can be used by Bob to provide his credentials.
3. Bob provides his credentials to the IdP, which verifies the claim based on data previously presented by Bob in the registration process to that IdP.
4. If the authentication is successful, the Identity Provider returns to Bob a digitally signed token containing the outcome of the procedure, together with identity information, then redirects the user to the service again.
5. Bob returns the token to the SP. If the token is valid for the previous request, Bob is granted access to the service resources, based on his associated permissions on the platform.

The same Identity Provider can act on behalf of multiple services, just as a service can trust various IdPs. Following the example presented to describe the centralized model flow, if Bob wants to access his bank account too, and the same Identity Provider is delegated to perform authentication for both the e-commerce and bank services, he can simply use the same credentials, benefiting from the SSO functionality facilitated by the trusted IdP. The flow, schematized in Figure 2.4, shows two elements that need further discussion, since they make the model trustworthy: the relationship between Service and Identity Providers, and the assertion (or token) that is shared between the three involved entities during the process.

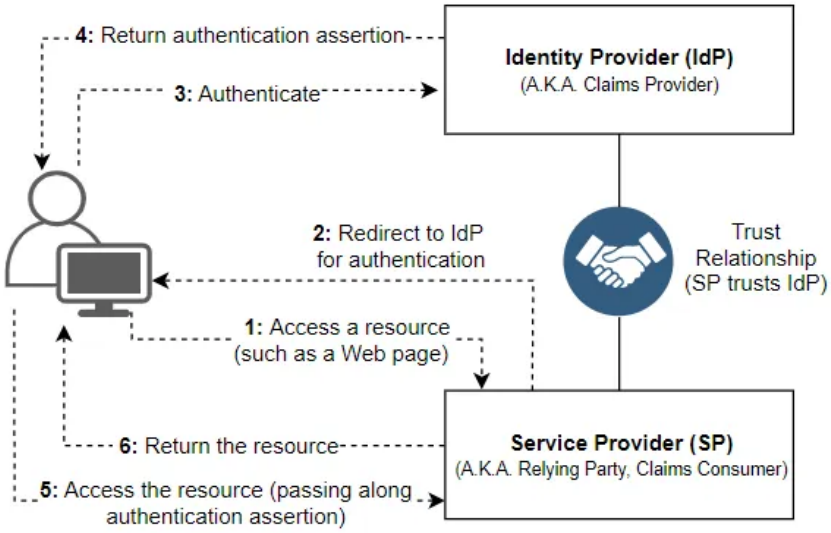

Figure 2.4. A simple schema of the federated model authentication flow. [12]

2.5.1 Trust relationship and evidences

Service Providers can choose which Identity Providers can perform authentication on their behalf. Because of that, there s the need to establish a trust relationship between the two of them, with which the SP accepts to rely upon the IdP capabilities of verifying user identities and to emit a trustworthy evidence of the outcome. This agreement is performed by mutually exchanging a structured set of information named metadata, either by exposing it on a publicly known endpoint on the service or sending it by using an alternative channel. JSON and XML are the most used formats for metadata, depending on the protocol that is used during the exchange, and will be addressed later, taking a closer look to their structure. The following is an example of an IdP metadata [13], in XML format, for Security Assertion Markup Language (SAML) 2.0 protocol:

```
<EntityDescriptor
    ID="_c066524f-ba36-49d5-9dfa-ae14e13c1392"
    entityID="https://idp.identityserver"
    validUntil="2022-07-20T09:48:54Z"
    cacheDuration="PT15M"
    xmlns="urn:oasis:names:tc:SAML:2.0:metadata"
    xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion">

    <IDPSSODescriptor WantAuthnRequestsSigned="true"
        protocolSupportEnumeration="urn:oasis:names:tc:SAML:2.0:protocol">
        <SingleSignOnService
            Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect"
            Location="https://idp.identityserver/saml/sso" />
```

```
        <SingleSignOnService
            Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST"
            Location="https://idp.identityserver/saml/sso" />
        <SingleSignOnService
            Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Artifact"
            Location="https://idp.identityserver/saml/sso" />

        <SingleLogoutService
            Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect"
            Location="https://idp.identityserver/saml/slo" />
        <SingleLogoutService
            Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST"
            Location="https://idp.identityserver/saml/slo" />
        <SingleLogoutService
            Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Artifact"
            Location="https://idp.identityserver/saml/slo" />

        <ArtifactResolutionService
            Binding="urn:oasis:names:tc:SAML:2.0:bindings:SOAP"
            Location="https://idp.identityserver/saml/ars" index="0" />

        <NameIDFormat>[...]:nameid-format:unspecified</NameIDFormat>
        <NameIDFormat>[...]:nameid-format:transient</NameIDFormat>
        <NameIDFormat>[...]:nameid-format:persistent</NameIDFormat>
        <NameIDFormat>[...]:nameid-format:emailAddress</NameIDFormat>

        <KeyDescriptor use="signing">
            <KeyInfo
                xmlns="http://www.w3.org/2000/09/xmldsig#">
                <X509Data>
                    <X509Certificate>[...]</X509Certificate>
                </X509Data>
            </KeyInfo>
        </KeyDescriptor>
    </IDPSSODescriptor>

    <Organization>
        <OrganizationName xml:lang="en-GB">Example</OrganizationName>
        <OrganizationDisplayName xml:lang="en-GB">Example
            Org</OrganizationDisplayName>
        <OrganizationURL
            xml:lang="en-GB">https://example.com/</OrganizationURL>
    </Organization>

    <ContactPerson contactType="technical">
        <Company>Example</Company>
        <GivenName>bob</GivenName>
        <SurName>smith</SurName>
        <EmailAddress>bob@example.com</EmailAddress>
    </ContactPerson>

</EntityDescriptor>
```

The document is divided into three main sections. The first includes information peculiar to the IdP, such as the *ID* and *entityID* fields that act as unique identifiers for both metadata and provider, accompanied by the expiration date of the document. The next section illustrates which service-specific endpoints should be used by other entities to interact with the IdP, in particular the one that receives the authentication requests. Then, the allowed type of identifiers that

can represent users are listed, followed by a valid X.509 digital certificate of the IdP, including the public key, which will be used by the SP to verify the evidence received from the user. More metadata, as the one listed at the end of the document above, can be included in the document, providing additional information. Also, the document itself can be digitally signed and the signature stored inside it, providing a way for the Service Provider to verify metadata integrity. Service Provider's metadata follows the same layout of the previous one, but listing its own endpoints and sharing also the key that will be used for encryption by the IdP.

The object that practically enforces the trust relationship is the evidence generated by the IdP and shared with the SP through the user. This digitally signed piece of information works as a proof of authentication and user attributes, providing authenticity, integrity and non-repudiation thanks to the Identity Provider digital signature, which can be verified by the Service Provider using the public key attached to the previously exchanged metadata. In addition, associating issued tokens to a timestamp or a session it is possible to prevent token replay attacks. Unlike a cookie-based authentication mechanism, in which some data are stored server-side to keep alive the session of the authenticated user, tokens are naturally stateless, allowing more scalability and flexibility.

2.5.2 SAML assertions

In Security Assertion Markup Language (SAML), the evidence is an assertion, i.e. a message in XML format, generated by the IdP in response to a request previously received from the SP. Here is an example of a basic SAML response [14]:

```
<samlp:Response xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
    xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion" ID="[...]"
    Version="2.0" IssueInstant="2014-07-17T01:01:48Z"
    Destination="http://sp.example.com/demo1/index.php?acs"
    InResponseTo="[...]">
  <saml:Issuer>http://idp.example.com/metadata.php</saml:Issuer>
  <samlp:Status>
    <samlp:StatusCode Value="urn:oasis:names:tc:SAML:2.0:status:Success"/>
  </samlp:Status>
  <saml:Assertion xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:xs="http://www.w3.org/2001/XMLSchema" ID="[...]" Version="2.0"
      IssueInstant="2014-07-17T01:01:48Z">
    <saml:Issuer>http://idp.example.com/metadata.php</saml:Issuer>
    <saml:Subject>
      <saml:NameID SPNameQualifier="http://sp.example.com/demo1/metadata.php"
          Format="[...]:nameid-format:transient">[...]</saml:NameID>
      <saml:SubjectConfirmation
          Method="urn:oasis:names:tc:SAML:2.0:cm:bearer">
        <saml:SubjectConfirmationData NotOnOrAfter="2024-01-18T06:21:48Z"
            Recipient="http://sp.example.com/demo1/index.php?acs"
            InResponseTo="[...]"/>
      </saml:SubjectConfirmation>
    </saml:Subject>
    <saml:Conditions NotBefore="2014-07-17T01:01:18Z"
        NotOnOrAfter="2024-01-18T06:21:48Z">
      <saml:AudienceRestriction>
        <saml:Audience>http://sp.example.com/demo1/metadata.php</saml:Audience>
      </saml:AudienceRestriction>
    </saml:Conditions>
    <saml:AuthnStatement AuthnInstant="2014-07-17T01:01:48Z"
        SessionNotOnOrAfter="2024-07-17T09:01:48Z"
        SessionIndex="_be9967abd904ddcae3c0eb4189adbe3f71e327cf93">
      <saml:AuthnContext>
        <saml:AuthnContextClassRef>[...]:Password</saml:AuthnContextClassRef>
```

```
      </saml:AuthnContext>
    </saml:AuthnStatement>
    <saml:AttributeStatement>
      <saml:Attribute Name="uid"
          NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:basic">
        <saml:AttributeValue xsi:type="xs:string">test</saml:AttributeValue>
      </saml:Attribute>
      <saml:Attribute Name="mail"
          NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:basic">
        <saml:AttributeValue
            xsi:type="xs:string">test@example.com</saml:AttributeValue>
      </saml:Attribute>
        [...]
    </saml:AttributeStatement>
  </saml:Assertion>
</samlp:Response>
```

The assertion is composed of different blocks:

- *'samlp:Response'*: contains the rest of the message. It states information about the message, without taking into consideration what is inside, such as a unique ID, the SAML version used (in this case, 2.0), the timestamp when the response was issued, the recipient of this response and a reference ID to the request from the SP. It can be considered like the envelope of a letter, in which is specified the address of the recipient along with the date of dispatch.
- *'saml:Issuer'*: declares who is the issuer of this assertion, in this case the IdP.
- *'samlp:Status'*: shows whether the authentication procedure at the IdP was successful or not.
- *'saml:Assertion'*: this is the main block of the message. It contains information about the recipient and the issuer of the assertion, making a difference between the entity who generated the assertion itself and the entity that handled the complete response (which is the one referred to in the previous Issuer block). In addition, it defines constraints on the validity of the assertion (*'saml:Conditions'* block) and provides an ID and a duration for the session (*'saml:AuthnStatement'* block), in which the user will be authenticated, specifying also which authentication method was used. At the end of this element, in the *'saml:Attribute'* block, are listed user-specific attributes and values.

Since it contains sensitive user information, the integrity and confidentiality of the assertion must be preserved, as well as the authenticity of the whole message. Because of that, the assertion block must be encrypted, using the public key of the SP obtained through the metadata exchange, and the IdP should digitally sign the response using its own private key. Information needed to provide these properties to the communication are all attached to the SAML messages, either to the request or the response, such as public certificates, encrypted data and signatures.

2.5.3 OAuth 2.0 and OIDC: access token and ID token

Other protocols that allow a secure transmission of the evidence across the parties are the combination of OAuth 2.0 and OpenID Connect. OAuth, used primarily as an authorization framework, can issue an access token that allows an application to access on the user's behalf to personal data, while OpenID Connect (OIDC), an identity layer built on top of OAuth 2.0, focuses on authorization and introduces an additional token, named ID Token, containing information about the user identity claims.

Both tokens could be opaque, but they are often formatted as JSON [15] (JavaScript Object Notation) objects. JSON format plays an important role in digital identity management, thanks to the fact that it contains human-readable data and its structure is compact and flexible. Each

object is a list of pairs consisting of a key string and a value that can assume different types (numbers, arrays, even nested objects); similar to a Python dictionary, it allows reference to the value inside a pair through the corresponding key. This is an example of JSON ID token followed by the description of the mandatory claims that must be included:

```
{
  "iss": "http://my-domain.auth0.com",
  "sub": "auth0|123456",
  "aud": "my_client_id",
  "exp": 1311281970,
  "iat": 1311280970,
  "name": "Jane Doe",
  "given_name": "Jane",
  "family_name": "Doe",
  "gender": "female",
  "birthdate": "0000-10-31",
  "email": "janedoe@example.com",
  "address": {
     "street_address": "1234 Hollywood Blvd.",
     "locality": "Los Angeles",
     "region": "CA",
     "postal_code": "90210",
     "country": "United States of America"},
}
```

- *'iss'* represents the URL of the entity that generated the token, referred as the Issuer.
- *'sub'* uniquely identifies the subject of the token (i.e. the user) with a code.
- *'aud'* contains the identifiers of the recipients for which the token is issued (i.e. one or more Service Providers).
- *'exp'* shows the expiration time of the token. If a Service Provider receives an expired token, it must not accept it. The validity time of the ID token is not related to the expiration time of the authenticated session for that user.
- *'iat'* displays the time at which the token was issued.

The other claims included in the previous token are completely optional and, in this case, provide information about the authenticated user. The *address* field is an example of aggregated claim, containing various nested claims about the location in which the user lives. Further claims can be added to the token, either taken from the OIDC specification [16] or customized ones that are fitting for the specific scenario.

Referring instead to access tokens, this is a successfully generated token response:

```
{
  "access_token":"MTQ0NjJkZmQ5OTM2NDE1ZTZjNGZmZjI3",
  "token_type":"Bearer",
  "expires_in":3600,
  "refresh_token":"IwOGYzYTlmM2YxOTQ5MGE3YmNmMDFkNTVk",
  "scope":"create"
}
```

- *'access_token'* contains the value of the issued token.
- *'token_type'* defines how the token must be used by the client. Usually, this is set as *'Bearer'*, meaning that the token can be used as it is, but other token types such as *mac* require the issuer to provide additional attributes to the user.

- *'expires_in'* determines the duration, in seconds, of the token.
- *'refresh_token'* contains a value that can be provided back to the authorization server from the client to obtain a new access token when the previous one expires.
- *'scope'* states the scope of the issued token. Optional unless the request included this parameter too.

While only the first two parameters are mandatory to be included, the expiration time of the token must be shared with the client by some means, hence it is usually provided inside the token.

Both JSON objects contain sensitive information: while the ID token directly includes personal data that can be targeted by malicious actors, a stolen access token could be used to gain access to protected resources until its expiration. To offer the same protections that are provided by the SAML protocol, the JSON object is included into a JSON Web Token [17] (JWT) instead of being sent as it is. A JWT is defined as a structure composed by three different base64url-encoded [18] parts, including:

- An header that carries information about the token type and the algorithm used to secure the token.
- A payload that includes the JSON object itself.
- A signature over the header and the payload, using the algorithm specified in the header.

Parts are concatenated to each other, separated by a period, composing a *header.payload.signature* string. A JWT is able to ensure integrity and authenticity for the JSON object thanks to the digital signature performed by the IdP. In addition, to preserve the confidentiality of token storing sensitive data, JSON Web Encryption [19] (JWE) can be used to encrypt the payload: in this case, issuer should first sign the token and only after encrypt it including the signature, avoiding privacy issues for the signer and ensuring that the signature cannot be removed from the message without invalidating it.

2.6 Challenges in the presented scenario

The identity models discussed previously, namely the Centralized and Federated approaches, while widely adopted, presented significant challenges that hindered the creation of a truly seamless and secure digital identity ecosystem, particularly in a cross-border context.

The Centralized model, characterized by each Service Provider managing its own user database, suffered from several critical drawbacks. Firstly, it created numerous single points of failure; each SP's database became a potential honeypot for attackers, with breaches potentially exposing large volumes of user credentials and personal data. Secondly, users faced a severe lack of control over their information, with no transparency or influence on how different SPs manage or potentially share their data, raising significant privacy concerns. Thirdly, the model resulted in poor usability due to credential fatigue, requiring users to create and manage distinct usernames and passwords for every online service.

The Federated model overcame the credential fatigue introduced by the centralized model by enabling SSO through trusted IdPs. However, it introduced its own set of challenges; establishing and managing the necessary trust relationships between potentially numerous SPs and IdPs, typically via metadata exchange, added significant complexity to the scene. While reducing credential duplication, users became dependent on the security and trustworthiness of their chosen IdPs, shifting the element of trust to another entity rather than eliminating the reliance on third parties. The protocols employed in the model, such as SAML 2.0 and OAuth 2.0/OIDC, while standardizing interactions, require careful implementation to ensure the security of assertions and tokens, posing implementation barriers for some SPs.

Most importantly, both models, as implemented predominantly within national or sector-specific silos, failed to provide a unified solution for cross-border authentication and identification

within the diverse legal and technical landscape of the European Union. The lack of mutual recognition and interoperability between different national identity systems created significant friction for citizens and businesses accessing services outside their home country.

2.7 The eIDAS Regulation

All the issues just exposed have the same common denominator: until that time, every state had complete freedom in defining its own identification system and how to manage Trust services, leading to a regulatory fragmentation that made challenging the interactions between a citizen and a service of a different country.

With the goal of addressing the limits of the old model, on 23 July 2014 the European Parliament published the electronic IDentification, Authentication and Trust Services Regulation [20] (eIDAS, EU Regulation 910/2014), repealing the Electronic Signatures Directive (1999/93/EC) and becoming mandatory for every European country starting from 1 July 2016. The new Regulation aimed to establish a common system that is valid for every Member State to manage digital identities for both companies and individuals, and introduced new standards to provide an higher grade of security and interoperability in digital operations, such as signatures and seals.

Figure 2.5. Trust services defined by the eIDAS Regulation. In the middle, the European Trust Mark.[21]

Main points of interest in the EU Regulation 910/2014 are reported below:

- Creation of a new trusted pan-European framework for digital identities, promoting interoperability between Member States and can be secure and convenient to use for citizens, companies and public administration services, respecting their privacy. Every Member State must accept and recognize eID systems notified by other countries, but only for online public services. The responsibility of making their own notified system reliable and secure is left to Member States, and it is not mandatory to have a national eID system.
- Definition of 'Trusted Services' as electronic services capable of:
  - creating, verifying and validating electronic signatures, seals and timestamps;
  - managing digital certificates for website authentication;
  - managing the e-delivery service.

  These services are offered by Trust Service Providers (TSP), either public or private entities that ensure authenticity and security properties over digital transactions. The Regulation also introduces Qualified Trust Service Providers (QTSP), a set of TSPs that is certified by a national authority and which services have legal value across the whole Europe. QTSPs are

supervised and subject to periodic checks by a dedicated authority, and they are included in the official EU Trusted List, publicly available.

- Creation of new sets of standards that must be applied when providing trust services:
  - for electronic signatures (ETSI EN 319 411-1/2 and ETSI EN 319 421/422), defining three levels of signatures:
    - simple electronic signature;
    - Advanced Electronic Signature (AES);
    - Qualified Electronic Signature (QES);
  - for electronic seals (ETSI EN 319 412-2/3), applicable to legal entities;
  - for timestamping (RFC 3161 and ETSI EN 319 421);
  - for e-delivery (ETSI EN 319 521/522), providing integrity and giving a legal proof about sending and receiving messages;
  - for authenticating websites (ETSI EN 319 412-4) using qualified certificates.

2.7.1 The eIDAS-Node infrastructure

To overcome the previous identity system fragmentation, a new architecture is developed, promoting interoperability between Member States. This structure, called eIDAS node, is a decentralized but interconnected system that enables the mutual recognition of national eID schemes across the EU. Its primary purpose is to allow citizens and businesses to authenticate themselves in foreign online services using their own nationally issued electronic identity. A node requires a national implementation by each Member State willing to use this system. During the years, the node structure changed, reflecting the technological evolution, Member State feedbacks and standards updates. However, a high-level view of an eIDAS node consists of:

- A Connector component, which manages requests from a service provider within the same country and forwards them to other eIDAS nodes.
- A Proxy-Service component, that receives requests from other eIDAS nodes and forwards them to an Identity Provider.

Each component has two internal elements, a Generic part and a Specific one. The first component, common to every node, allows communication with other nodes, managing eIDAS authentication requests and responses, and retrieving SAML metadata from the communicating part, by using the eIDAS protocol. The Specific one, instead, is implemented individually by each Member State to interact with Service and Identity Providers. Both parts of the same component must be executed on the same server to be effective.

Central to the eIDAS framework is the concept of the Minimum Data Set (MDS) for natural and legal persons. The eIDAS technical specifications mandate the support for a core set of attributes designed to provide trustworthy identification data across borders. This set includes four mandatory attributes: *FamilyName*, *FirstName*, *DateOfBirth*, and *PersonIdentifier*. Additionally, four optional attributes (*BirthName*, *PlaceOfBirth*, *CurrentAddress* and *Gender*) can be requested, although their provision depends on the national eID scheme and user consent. The *PersonIdentifier* is guaranteed to be unique across the EU for identifying the person in a specific transaction, but its characteristics presented operational difficulties. For example, it is not necessarily persistent for the same individual over time, nor is it guaranteed to be derived from a stable national identifier; it could even be pseudonymous or change depending on the Identity Provider used for authentication within a Member State (as observed, for instance, with Italy's SPID system where different IdPs can lead to different *PersonIdentifiers* for the same citizen). This lack of guaranteed persistence and uniqueness to the person poses significant challenges for Service Providers requiring long-term user relationship management or linking cross-border identities to existing national user profiles. For instance, practical implementations [22] aiming to link eIDAS identities to university records required leveraging stable national identifiers (like the

Italian fiscal code) within custom application logic (via AP Connectors, which are logical components designed to retrieve additional attributes from external Attribute Providers, interfacing with the university backend), as the standard eIDAS PersonIdentifier alone was insufficient for reliable matching. Conversely, for simpler services like temporary Wi-Fi access, the mandatory MDS attributes were sometimes perceived as excessive or overly sensitive; in a validation study [23] involving 23 participants testing a prototype eIDAS-enabled Wi-Fi service alongside a login service at Politecnico di Torino, some of them expressed concern over sharing their full MDS for Wi-Fi, contributing to a notable percentage (22% "No", 25% "Not sure" in the survey responses) questioning its usefulness in that specific context.

In order to manage security risk related to identification mechanisms, eIDAS introduces three Levels of Assurance (LoA). These levels (Low, Substantial, High) determine the degree of robustness of both the credential issuance and the authentication procedure employed by the national eID scheme: Member States must map their scheme to the corresponding LoA [24], making the system transparent for other countries. Service Providers in one Member State must accept an eID from another Member State if they meet the LoA required by the service, but at least including LoA Substantial and High.

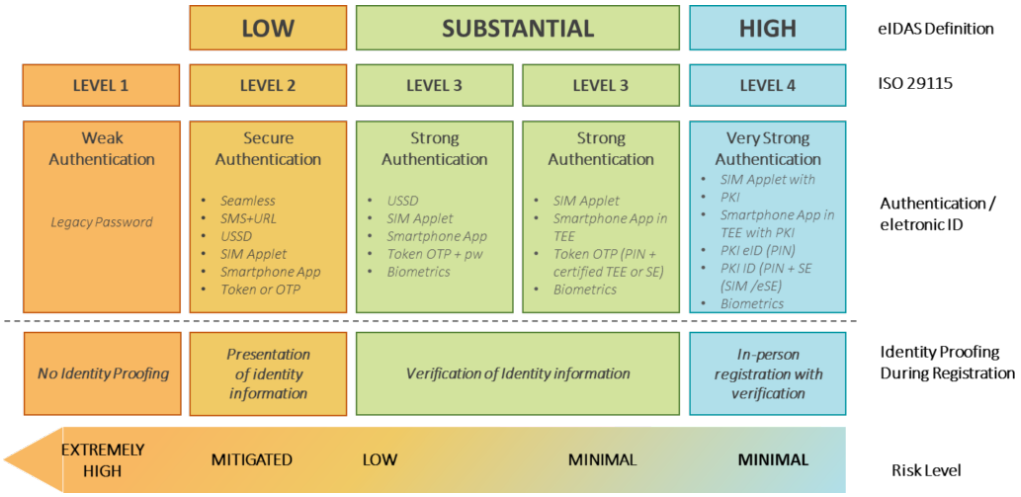

Figure 2.6. Comparison between the three Levels of Assurance defined by eIDAS. [25]

The communication between eIDAS nodes primarily leverages the SAML 2.0 Web Browser SSO Profile for the secure exchange of authentication requests and responses, including the user attributes. The trust within this decentralized network is established not through a central authority, but via bilateral agreements between Member States, together with the verification of eIDAS metadata. Each Member State acts as a trust anchor for its own node and distributes metadata containing the necessary public key certificates, in order to allow other nodes to verify the authenticity and integrity of received SAML messages and to encrypt attributes for confidentiality during the exchange.

Assuming that a citizen of a given Member State wants to access a public service of a foreign country:

1. User requests access to the online service.
2. The service provider redirects the user to the specific eIDAS Connector of that foreign country. This will be defined as Receiving Country, since it is the one that needs user authentication data from the other country, opposed to the Sender, that will manage the authentication procedure.
3. The Connector creates a request to the eIDAS Proxy-Service of the country with which the user has expressed his will to continue the authentication process (during this step or the previous one).

4. The Proxy-Service converts the eIDAS Request in a format compatible with the Identity Provider of the Sending Country.
5. The user authenticates themselves, then user data are returned to the Proxy-Service, where an eIDAS Response is generated based on these data.
6. The receiving country's Connector translates the response to be compatible with the Service Provider.
7. If the authentication is successful, the service provider grants the user access the service.

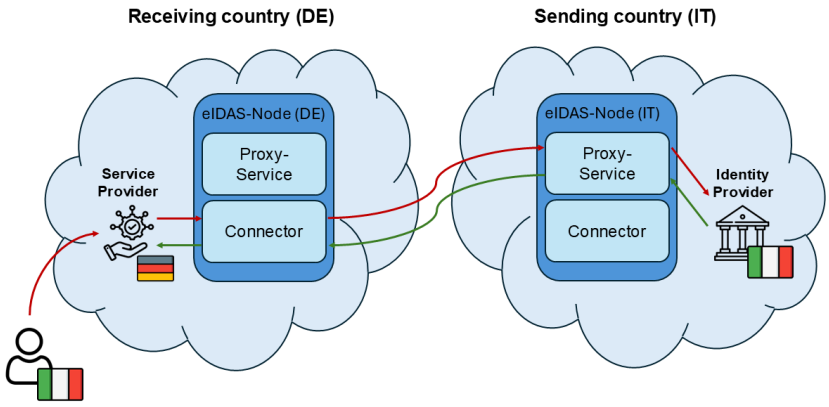

Figure 2.7. High-level flow of eIDAS-Node components involved in the cross-border authentication process.

2.7.2 eIDAS results and expectations

Member States started implementing new systems at their own pace, depending on cultural, technological and organizational differences. The first big improvement is about the introduction of AES and QES, that significantly reduced processing times for administrative procedures, and of the EU Trusted List, which contributed to make electronic transactions more secure. In addition, multiple Member States implemented their own eID schema and notified it to other countries: as reported by the European Commission [26], which is in charge of publishing eID systems, if prior to the eIDAS Regulation only three countries had a national eID schema, at the end of 2020 the amount of notified schemes is 19, from 14 different Member States. The Figure 2.8 shows some relevant information about how, in the past years, the Regulation had an impact on making a big step forward to the current scenario. Although involving only 14 of the 27 Member States might seem a poor result, the whole picture into which this Regulation is inserted should be considered:

- Only 60% of European citizens had access to a notified eID scheme.
- Cross-border authentication is only a small fraction of the authentication scenarios, as most interactions are still limited within national borders.
- The lack of awareness of European citizens about the possibilities offered by the eIDAS network reduced the usage of the infrastructure.
- To benefit from eIDAS, Service Providers must support eIDAS authentication.

Due to this, eIDAS did not achieve its goal of creating a widely adopted digital identity framework across Europe. The private sector was completely excluded from the Regulation, leaving small and

medium-sized enterprises without a standardized mechanism to integrate electronic identification systems into their services. Moreover, the crucial citizen needs of having more control over their personal data and a seamless user experience were not adequately addressed, reducing the adoption of the system. Despite its limitations, eIDAS 1.0 highlighted key areas for future improvements, serving as a crucial starting point for new developments that aim to create a more user-centric and scalable solution, towards a unique solution for the whole Europe.

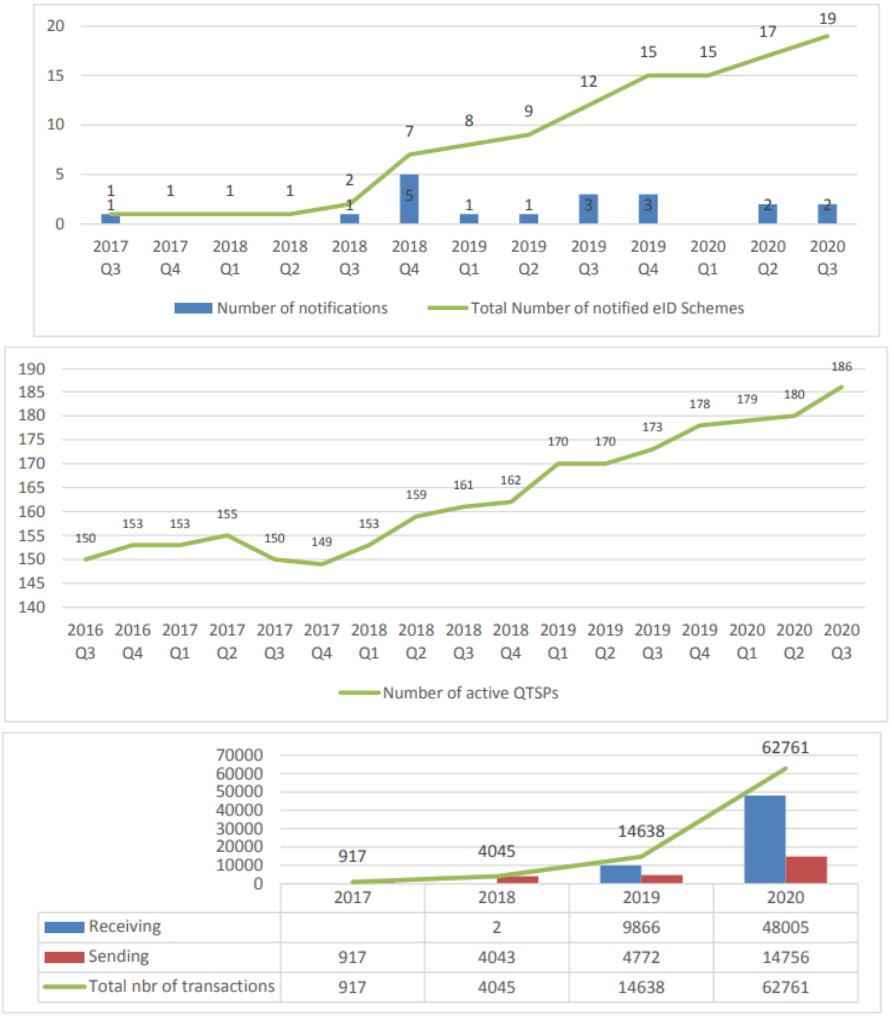

Figure 2.8. From top to bottom, the growth of: notified eID schemes, active QTSPs and number of eID cross-border authentications over time, under the first eIDAS Regulation.[26]

Chapter 3

The eIDAS 2.0 Regulation and EUDI Wallet

3.1 The Decentralized Identity model

During the years following the publication of the first eIDAS Regulation, as a response to the inherent limitations of traditional digital identity systems, the decentralized identity model emerged. Unlike its predecessors, this model redistributes control over identity data, reducing the burden on centralized authorities (whether a single Identity Provider or a federation of providers), addressing the risk of having a single high-value point of failure that can become the target of cyberattacks, and eliminating the need for users to establish a trust relationship with a third-party service that is responsible for managing and securely transmitting their claims.

In a decentralized identity model, identity management is structured around three key roles: the issuer, the holder and the verifier. The issuer, as with the previously described models, is the entity responsible for creating and digitally signing credentials (referred to as a signed and cryptographically secure document, not as traditional login credentials such as usernames and passwords), asserting claims about an individual, organization or device. Credentials are then stored and managed by the Holder, typically using an appropriate application, who can present them to third-party services, exercising full control over them. The third role, the verifier, is the entity that can assess the validity and authenticity of the presented credentials, by validating their cryptographic signature.

3.1.1 Decentralized Identifiers, Distributed Ledger Technologies and Verifiable Credentials

To enhance trust in an environment lacking a central authority, new elements must be introduced. Decentralized IDentifiers (DIDs) are globally unique identifiers which enable entities within the identity ecosystem to securely reference each other, retrieving associated cryptographic keys or other metadata required to verify digital signatures or interact securely, without intermediaries. DIDs are created and managed directly by users and do not inherently include any personal information, allowing the same individual to have multiple identifiers, each intended for a different purpose or context. A DID is a standard structure defined by the World Wide Web Consortium (W3C) [27], split into three different parts separated by colons: *did*:<*did_method*>:<*did_method_specific_ID*> .

- The prefix *did* is used to determine that the structure is a decentralized identifier.
- The *did_method* defines which specification is followed to create and manage DIDs.
- The *did_method_specific_ID* represents the identifier. It must be exclusive within the selected method, thus making the whole DID globally unique.

DID acts as a pointer to a DID document expressed in JSON or JSON-LD [28] format, which contains metadata, public keys and other information that can be used to initiate a secure communication with the subject. The document can be retrieved by resolving the identifier, following the process defined by the specified DID method.

While DIDs can be included in X.509 certificates or within HTTP resources, they are typically anchored to Distributed Ledger Technologies (DLTs), tamper-resistant decentralized databases shared across a peer-to-peer network: every time a DID is created or revoked, the transaction is recorded in the shared registries, ensuring transparency on operations over the recorded data. In blockchains, the most common example of DLT, records are organized into blocks, each containing a set of transactions, a timestamp and a cryptographic hash of the previous block. The chain that is created by adding new blocks (from which the structure takes its name) makes sure that any link cannot be altered without invalidating all the subsequent ones, thus providing an ordered, immutable and verifiable structure. To maintain data integrity and synchronization among all network participants, cryptographic and consensus algorithms are employed, managing potential conflicts when concurrent blocks are generated and guaranteeing a "trustless" environment. In particular, the latter are essential to ensure that participants in the blockchain can agree on the validity of new blocks before they are added to the chain, preventing malicious transactions from being accepted into the structure.

The final key element of the model is the Verifiable Credential [29] (VC), also standardized by W3C, which represents a cryptographically secure digital document stating specific claims about the subject in a machine-readable structure. A Verifiable Credential includes:

- Metadata that define the issuer of the VC and state information about the format and validity of the attestation.
- A set of claims about the subject, each one expressed as a pair attribute-value.
- A cryptographic proof, typically a digital signature made by the issuer, that can be used by a third party to verify the integrity and authenticity of the document without direct interaction with the signer.

When an issuer generates a VC, it digitally signs the credential using a private key whose corresponding public key is published in the issuer's DID document, linked to its DID on the blockchain. During online interactions, the holder may need to share one or more VCs to a third-party verifier: this is done by creating a Verifiable Presentation, a structured container that provides, along with the VCs that the holder wants to share, metadata about the holder and the presentation itself, and the digital signature generated by the holder using their private key which also proves the explicit consent to share the involved data. Receiving a Verifiable Presentation, the verifier can independently retrieve the holder's public key through their DID document, then validate the digital signature, and repeat the same procedure for issuer's signature. Figure 3.1 describes those interactions, highlighting two aspects that distinguish this model from the previous ones: first, in a decentralized system, the credential issuer does not need to know anything about the verifier, since it will never interact directly with it and there is no trust relationship established between the two entities; in addition, neither the verifier nor the issuer has any knowledge about how the credentials are stored and managed, because this responsibility now belongs entirely to the holder.

3.1.2 Digital wallets

In order to store and manage Verifiable Credentials directly and initiate presentations to a third-party service, holders can use specialized applications known as digital wallets. In the decentralized identity model, the wallet acts as the interface between the holder and the whole identity ecosystem, receiving Verifiable Credentials from trusted issuers, linking them to Decentralized Identifiers and generating Verifiable Presentations.

Digital wallets are considered highly secure due to several core features:

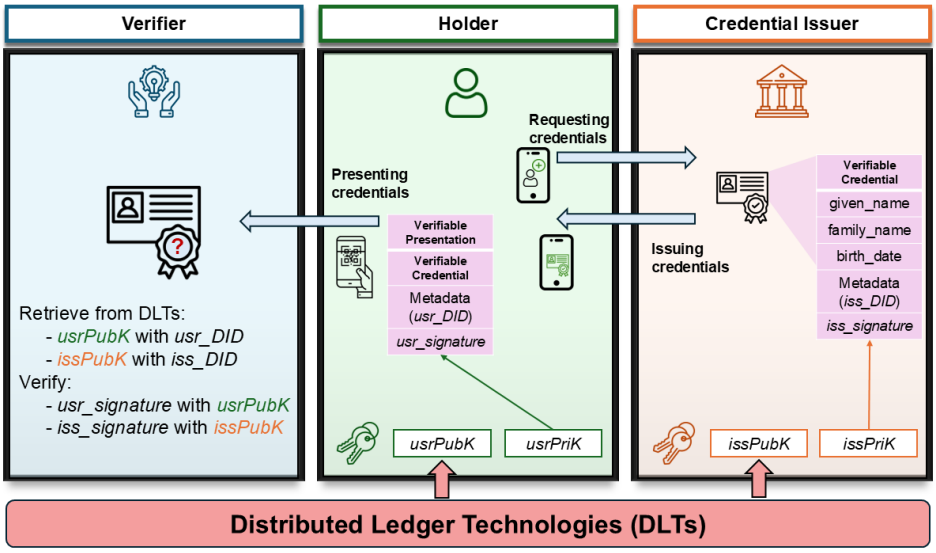

Figure 3.1. Interaction during issuance of a Verifiable Credential and the share of a Verifiable Presentation between the involved entities in a decentralized model.

- They can be used to generate and securely store private keys, associated to the user, inside a trusted environment within the device. Those keys can be used to perform digital signatures or decrypt data.
- They offer protection from unauthorized access by providing a local authentication method, such as PINs or biometric identification.
- Stored credential and metadata are encrypted, as well as any transmission to third parties that is carried establishing secure communication channels.
- User has fully control over their data: when presenting a credential, they are aware of which attributes are shared and why they are requested from a verifier. Presentations cannot be initiated without user's consent.

The deployment of a digital wallet poses several challenges, either technical or human-centred. First of all, it must provide a good balance between strong security mechanisms and a seamless user experience: the management of keys and credentials should not be overwhelming for the individual, thus the application interfaces must be designed to be intuitive and exhaustive. Studies conducted on existing digital wallet prototypes [30] show how different interface designs can support or interfere with users' decision making when using these applications. The authors tested multiple wallet UI approaches allowing the participants to interact with them, experimenting with features such as detailed data visualizations, interactive selectors for shared attributes and privacy warnings (for example, a sensitivity score that determines how much a claim is sensitive, or an alert message). The findings show that while users generally appreciate being informed about what they are sharing, many still tend to accept data disclosure requests when motivated by convenience or reward, particularly under time pressure or in scenarios with perceived benefits. A well-designed interface must be capable of providing users with the necessary information to take a privacy-aware decision when sharing a credential or a set of claims, guiding them in understanding the implications of their choices without being too invasive.

Another problem is the interoperability in the whole ecosystem: wallets must be compatible with different issuers and verification systems that might belong to different jurisdictions. Moreover, a wide range of credential types must be supported, along with the formats in which each credential can be instantiated; digital wallets have to adapt to different legal frameworks and use cases to accomplish their tasks.

Finally, the last challenge concerns mobile threats, significantly increased in recent years, and exploiting insecure network connections, weaknesses in mobile operating systems or social

engineering techniques. Empirical studies, such as the one conducted in the city of Chennai (India) [31], identify how the most common threats, such as weak PINs, phishing emails, fake access points and the use of public Wi-Fi networks can strongly influence user perception about digital wallets security, discouraging their adoption. Also, the case in which the device storing the credentials gets lost or stolen must be addressed, due to the absence of a central authority that can restore the access to issued credentials.

3.1.3 The Self-Sovereign Identity paradigm

Built on the core concepts of the decentralized identity model, the Self-Sovereign Identity (SSI) paradigm emphasizes the autonomy of individuals of managing, storing and sharing their data, becoming themselves the ultimate authority over the control and use of their credentials. The term became popular when Christopher Allen, founder of Blockchain Commons and co-author of both the IETF TLS and W3C DID standards, introduced it in his publication 'The Path to Self-Sovereign Identity' [32] on his personal blog in 2016.

In the article, Allen traces the evolution of digital identity models, critically examining their structural limitations. He describes federated identity systems as an "oligarchy", where instead of entrusting identity management to a single, centralized authority, users rely on a limited group of powerful entities. Although federated identity solutions reduce the number of distinct identities an individual must maintain across services, they still inherently require the user to place trust in these selected providers, which continue to hold considerable control over user data. Moving beyond the federated approach, Allen introduces the concept of user-centric identity, in which individuals or administrative entities exercise greater control over their digital identities, managing them independently across multiple authorities without the need for formal federation agreements between service providers. However, Allen highlights that even user-centric models fall short of delivering full sovereignty: despite they can offer users more direct control over their credentials, the dependency on providers for crucial operation such as issuance and revocation of credentials strictly limits individuals' autonomy.

According to Allen, the concept of Self-Sovereign Identity represents a step beyond user-centric identity. The fundamental idea behind SSI is to enable users to have control not only on the storages of their own identity data, but also on issuing, revoking and sharing credentials without the need of external authorities or intermediaries. In the same article he also describes which are, in his opinion, the ten principles of the paradigm:

- Existence: digital identity is just a limited representation of a real entity or individual. Hence, SSI must enable users to create digital structures that corresponds to their existence.
- Control: individuals should be able to determine how, when and with whom their identity information is shared.
- Access: a user must be capable of retrieving their own identity data without intermediaries.
- Transparency: Identity systems and algorithms that provide security features over user data must be open-source and well known, ensuring that individuals can understand clearly how their data are managed.
- Persistence: identities should be long-lived, remaining valid and consistent even as individual claims or attributes evolve or expire.
- Portability: identities should be transportable across different systems, avoiding technical or jurisdictional barriers in which a user could incur if their data were held by a singular third-party entity.
- Interoperability: SSI must promote a wide usage of digital identities, overcoming limits caused by national boundaries or the lack of applications across the whole spectrum of online services.
- Consent: any use of an individual's identity must be explicitly allowed by the user themselves.

- Minimalization: shared data must be sufficient to accomplish the task for which it is requested, but without providing more information than it should, safeguarding user privacy.
- Protection: identity systems must always prioritize user security and privacy against their needs.

Although the terms "decentralized identity" and "self-sovereign identity" are often used interchangeably, a subtle distinction can be made between the two. While decentralized identity refers more broadly to an architectural model in which identity data is no longer stored or controlled by centralized entities, SSI extends this vision by explicitly defining the principles that empower individuals with full ownership and control over their digital identities. Moreover, in the SSI model, the issuer exercises control over credentials only during the issuance process: after that, revocation, storage, disclosure and other procedures are delegated to the user. In essence, SSI can be understood as a specific implementation of the decentralized identity model, guided by a strong emphasis on individual autonomy, privacy and consent.

3.2 The eIDAS 2.0 Regulation

Despite significant progress in creating interoperability for cross-border authentication, the first eIDAS Regulation exhibited clear limitations when confronted with the growing adoption of decentralized identity principles, particularly with the Self-Sovereign Identity model. The original eIDAS relied on a federated identity system, which requires citizens to trust identity providers and national authorities with control over their credentials and personal data, resulting in a lack of adaptability to the current digital environment.

Given these limitations, the European Commission recognized the necessity to evolve towards a model integrating SSI principles, promoting user autonomy, data minimization and transparency. Consequently, the proposal for revision known as eIDAS 2.0 (2024/1183) [33] was presented by the European Commission in June 2021 officially starting the corresponding legislative process, moving towards a decentralized paradigm where users have direct control over their digital identities, aligning the regulatory framework with current technological and privacy requirements. The final text was voted and confirmed by the European Parliament in February 2024, followed by the formal adoption by the Council, while the Regulation entered into force on 20 May 2024, as illustrated in Figure 3.2.

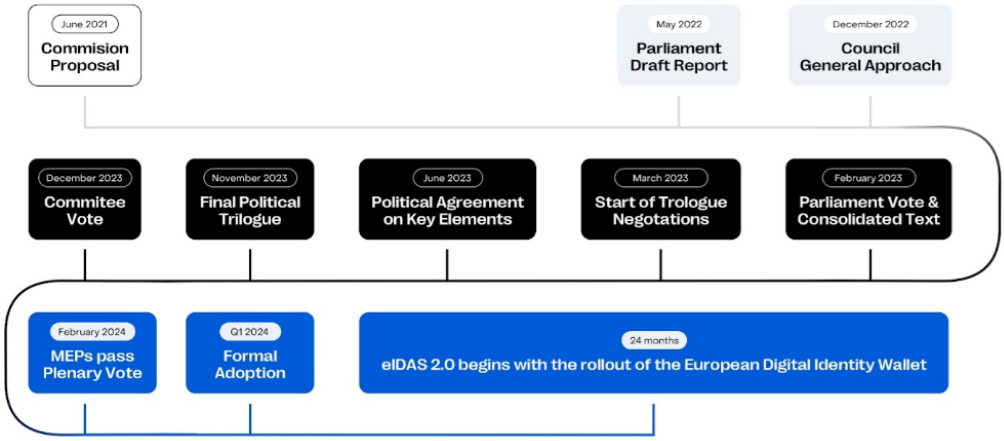

Figure 3.2. Phases of the eIDAS 2.0 drafting, publication and adoption. [34]

Compared to the previous version, eIDAS 2.0 expands its scope, introducing new standards and enhancing user privacy, under the principles of the General Data Protection Regulation: the goal is to accomplish the plan for which the previous Regulation was written: creating a new, unified European framework for digital identities. In the next subsections, each of these innovations will be discussed in detail.

3.2.1 New Trust Services and Qualified Electronic Attestation of Attributes

The first eIDAS Regulation defined the concept of trust service and introduced a set of standards for electronic operations, as explained in the previous chapter. eIDAS 2.0, based on the digital transformation that occurred over the years, introduces four new qualified trust services:

- Qualified Electronic Archiving, which provides a secure way to store any kind of electronic data with legal value, guaranteeing their availability and reliability over time.
- Management of remote electronic signature creation devices or remote electronic seal creation devices, allowing trusted service providers to manage the creation of seals and signatures remotely on behalf of users on their request, storing private keys in secured and certified Hardware Security Modules (HSMs).
- Qualified Electronic Ledgers, used to register data in a verifiable and immutable way, even for trusted business operations and public records.
- Qualified Electronic Attestation of Attributes (QEAAs), that allows the issuance of tamper-proof digital documents about user attributes, verifying them against trusted sources.

These new qualified trust services, along with those already defined in the first eIDAS Regulation, now extend explicitly to the private sector under eIDAS 2.0. This expansion means that private organizations, such as the ones involved in the finance, healthcare and transportation sector, must recognize and integrate these services into their operations, enhancing the reliability and interoperability of digital interactions across both public and private domains.

The introduction of QEAAs represents one of the most impactful innovations, directly complementing the Self-Sovereign Identity model embraced in eIDAS 2.0. By issuing tamper-proof digital attestations about specific user attributes (professional qualifications, licenses, citizenship), QEAAs enable holders to present verifiable information in digital interactions. In addition, to align with the data minimization goal of eIDAS 2.0, QEAAs supports selective disclosure: when presenting their credential with third parties, users can choose which attributes to share, limiting the disclosure only to necessary information. As determined in the first Regulation, Qualified Trust Service Providers, supervised and periodically audited by national authorities, are responsible for the issuance of these attestations, ensuring compliance with security and privacy standards.

3.2.2 Expansion to the private sector

Mutual recognition of national eID schemes, as well as the already mentioned implementation of trust services, were designed to be mandatory only for the public sector, strictly limiting the applicability of eIDAS principles. With the new Regulation, private entities are legally required to recognize the same schemes, allowing users to authenticate to private services using a compliant national eID. This provides advantages for both service providers and individuals.

A large number of services can now leverage national digital identity systems to streamline the verification of their customer, no longer needing to build their own authentication system or rely on third-party checks. These services embrace different sectors, ranging from energy to healthcare, but also including telecommunications, commercial, and travel services. For businesses, it represents a scalable solution that allows, together with the other principles of eIDAS 2.0, to move towards a less fragmented environment. For individuals, it means having access to an increasing number of services using a single trustworthy identity credential, reducing the amount of accounts to manage, and ensuring a secure and seamless user experience.

3.2.3 Mapping of eIDAS 2.0 Minimum Data Set

Another key contribution introduced by eIDAS 2.0 is the redefinition and enforcement of a Minimum Data Set [35] (MDS) for the identification of natural and legal persons in cross-border digital interactions.

For natural persons, the mandatory set includes the same four core attributes that were mandatory for the first eIDAS Regulation: current family name, current first names, date of birth, and a unique person identifier. These attributes are considered the minimum legal and technical requirement to perform identity verification under eIDAS 2.0. Optional attributes may be added by Member States if legally permitted and available in their national systems. Each attribute follows strict encoding rules and naming conventions compatible with SAML 2.0 specifications, enabling seamless integration with identity systems and Relying Parties across Europe.

Attributes for Natural Persons	Attributes for Legal Persons
FamilyName	LegalName
FirstName	LegalPersonIdentifier
DateOfBirth	LegalAddress
PersonIdentifier	VATRegistration
BirthName	TaxReference
PlaceOfBirth	BusinessCodes
CurrentAddress	Legal Entity Identifier (LEI)
Gender	Economic Operator Registration and Identification (EORI)
Nationality	System for Exchange of Excise Data (SEED)
CountryOfBirth	Standard Industrial Classification (SIC)
TownOfBirth	LegalPhoneNumber
CountryOfResidence	LegalEmailAddress
PhoneNumber	
EmailAddress	

Figure 3.3. List of attributes defined by eIDAS. Subtitled ones are mandatory attributes for the corresponding type of person (natural or legal).

The specification goes even further, providing guidance for any form of representation, including situations where a natural person represents another natural person, or where a legal person represents another legal or natural person. To represent such scenarios in SAML assertions, the MDS must include two sets of attributes, one related to the represented person, the other to the representative one.

3.3 Implementation of eIDAS 2.0 principles

In order to integrate SSI principles and resolving the fragmentation caused by the lack of a coordinated implementation of services in the previous models. The Regulation also introduces two fundamental elements, strictly related to each other: the European Digital Identity Wallet (EUDI Wallet) and the associated Architecture and Reference Framework (ARF) [36]. By 2026 (two years after the entry in force of the Regulation), Member States are mandated to provide their

citizens with at least one digital wallet solution that complies with eIDAS 2.0 standards: this can be done leveraging the ARF and the EUDI Wallet as cornerstones, establishing a unified, interoperable and user-centric digital identity framework. However, as for the current European plans, digital documents are not intended to substitute their physical counterpart for identification: due to different cultures and legislations that could affect the usage of the digital identity system, both will coexist, and European citizens should not be penalized for not using a digital wallet for offline or online interactions.

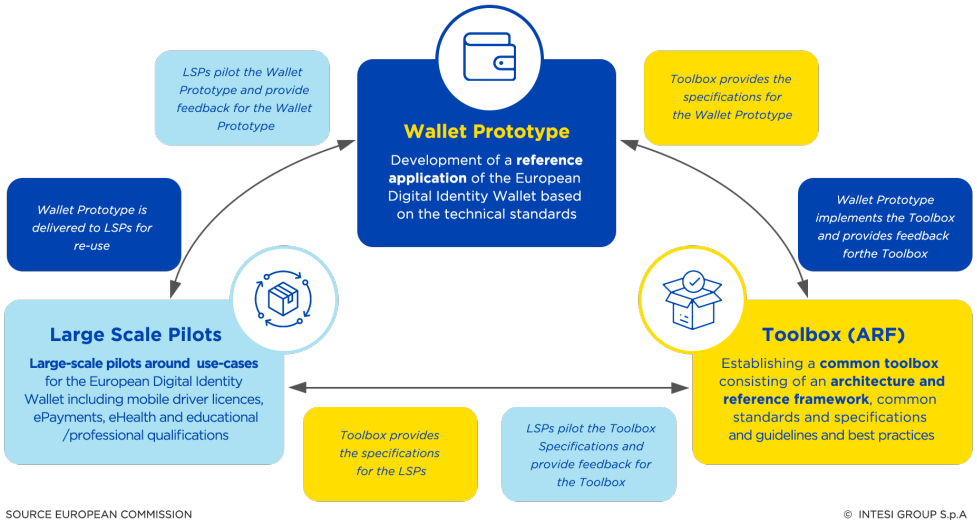

Figure 3.4. Relationship between the EUDI Wallet prototype, the Architecture and Reference Framework and the LSP projects.

3.3.1 European Digital Identity Wallet and Large Scale Pilots

The European Digital Identity Wallet is a secure digital application designed to enable European citizens to store, manage, and share digital identity credentials, both online and offline. It represents the practical implementation of the Self-Sovereign Identity principles within the eIDAS 2.0 framework, allowing European citizens to maintain direct and exclusive control over their personal data. The prototype of the application, continuously updated and reviewed, is open-source, available on the Official GitHub Organization of the European Digital Identity project [37]. To validate the functionality of the EUDI Wallet, the European Commission launched four Large-Scale Pilot (LSP) programs, officially starting in April 2023. These pilots include more than 350 between public and private entities across 26 different Member States, grouped into consortia, testing various features of the wallet implementation. Currently, the four LSPs are focused on showcasing different real-world use cases of the application, providing their feedback to the developers.

- The EU Digital Identity Wallet Consortium (EWC) aims to leverage the benefits of the European digital identity system in the travel sector, by implementing Digital Travel Credentials (DTCs). European travellers should be able to present their DTCs for identification and authentication across borders, with high security standards: EWC deals with the format of the credential, defining which attributes and structure can fit adequately this scenario, along with addressing both remote and in proximity verification of the documents, making DTCs very flexible and easy to use.
- Potential addresses critical domains of the digital identity usage: e-government services, banking, telecommunications, healthcare, electronic signatures and mobile driving licenses (mDLs). The goal is to streamline operation that otherwise would require time-consuming and complex procedures.

- NOBID focuses primarily on utilizing the EUDI Wallet as a way to authorize payments for goods and services. Simplifying the user experience, enhancing the security behind payment processes with a strong authentication procedure and involving the wallet to request user's consent before starting the transaction are a necessary step to increase trust in digital financial interactions.
- Digital Credentials for Europe (DC4EU) is the pilot responsible for making it easier and safer for citizens to manage and share digital credentials related to education and social security across Europe. Specifically, this pilot tests how the EUDI Wallet can securely store digital versions of documents such as educational diplomas, professional certifications, and proof of eligibility for social benefits. This will simplify processes like enrolling in universities, validating professional qualifications when moving countries, or accessing social security services, creating a more consistent and transparent system.

These projects are scheduled to end during 2025, after collecting an extensive amount of feedback and data that will be used to refine the structure of the application and the related ARF.

3.3.2 The Architecture and Reference Framework

The Architecture and Reference Framework [36], published by the European Commission in collaboration with the eIDAS Expert Group, defines common standards, principles and guidelines for the design and implementation of digital wallets, aligned with the Regulation. Member States that deploy their own digital wallet application on the basis of the ARF will ensure that their solution will be interoperable with other ones deployed on the same principles, along and with services, issuers and verifiers. In particular, the document serves as a reference for the implementation of the EUDI Wallet, which can be used as a prototype for national solutions.

The document is organized into sections, each addressing specific aspects relevant to the ecosystem. Three macro-sections can be distinguished within the ARF: the first describes accurately its scope and structure, introducing recurring terms; the second one delves deeper into analysing the trust model between the involved entities; the third is dedicated to credential issuance and presentation and security implementations. While the third will be addressed in the next chapter as the analysis of the issuance and presentation flows will highlight the security features implemented, the first two are described in the following subsections.

Definitions, use cases and overview of the infrastructure

The ARF begins clearly defining its scope and defining the roles of the Large-Scale Pilots and the EUDI Wallet in the proposed environment, determining also some key real-world use cases for which the documentation and the EUDI Wallet are suited. The core usage of the application, the basis for every secure online interaction, is to enable wallet holders to access online services from both private and public sectors, allowing authentication at the highest assurance level (LoA High), through data selectively disclosed by the user. Other use cases involve a wide spectrum of sectors, addressed by the four LSPs; giving access to health data remotely and allowing electronic prescriptions to patients, sharing a mobile Driving License (mDL) to prove driving privileges to both humans and machines, or even demonstrating their own rights and obligations presenting PDA1 or Electronic Health Insurance Card (EHIC). The ARF takes care of services that were heavily impacted by the fragmentation resulting from centralized and federated approaches, as well as by differing national legislations, addressing both online (remote) and proximity scenarios.

Following the use cases section, a substantial list of definitions is provided, regarding terms that will be frequently used in the ARF or that lacked a precise meaning in the context of digital identity, along with an explanation about the role of the main entities in the ecosystem. Here are reported the most important ones, useful for understanding the rest of the document.

- Person Identification Data (PID): a structured set of data that can uniquely identify a natural or legal person. This is the first document that must be issued to a digital wallet,

since the issuance of (Q)EAA relies upon the user identification procedure through their PID. Comparing them with physical documents, PID is the digital representation of a national ID card, while (Q)EAA are referred to a variety of attestation, including driving licenses, health ID cards or educational certificates, or documents with lower criticality. PID lifetime is bound to timestamps for both issuance and expiration: between these two dates, the digital document is considered valid, and can be shared with other entities to obtain access to services or request new documents; after the expiration date or a premature revocation by its Provider, a new PID must be requested.

- Attestation: a signed set of attributes either in the mdoc format specified in ISO/IEC 18013-5:2021 [38] or in SD-JWT format [39]. With reference to previously mentioned documents, this could be a PID, a QEAA or EAA.
- Issuer: a PID or (Q)EAA Provider, which signs those documents before issuing them to users that requested them.
- Authentic Source: trusted public or private repository or system recognized by the Member State that contains attributes of legal and natural persons, and can be questioned by Issuers to verify the authenticity of user claims. If allowed by the Member State, and if complying with eIDAS requirements, Authentic Sources can act as (Q)EAA Providers.
- Wallet Solution: a developed application offered by the Member State or private organizations to the citizens of that country. It must allow users to receive, store and present attestations to prove their identity, and to create Qualified Electronic Signatures and Seals.
- Wallet Provider: develops, distributes and provides support for its Wallet Solution, making sure it adheres with eIDAS 2.0 principles.
- Wallet Provider Trusted List: provided by the Member State, includes all the Wallet Providers (and their related public information) that deployed a Wallet Solution adhering to eIDAS 2.0 standards, thus officially recognized as a national digital identity solution.
- Wallet Instance: the installed application, used to receive, store or share any type of document on the user's device. Directly under the control of the user, through an intuitive UI.
- Relying Parties (RPs): provide services to which users may want to access that require user authentication, carried out by presenting a valid PID or (Q)EAA. The (mutual) authentication process can be performed by the Wallet Instance, using the Web Browser eventually as intermediary. Every RP must establish and communicate to its Member State which attributes it is intended to request for user authentication, making sure that the Wallet, during a transaction, can minimize shared data.

The Figure 3.5 shows how the entities just mentioned interact with each other to accomplish the main tasks of the EUDI Wallet, on user request. In particular, for the issuance and the presentation of Verifiable Credentials, respectively involving PID/(Q)EAA Providers and Relying Parties, are specified the protocols that support a secure communication with the Wallet Instance: both OpenID4VCI and OpenID4VP will be analysed in detail in the next chapter, contextualized to the local implementation of the issuance and presentation flows.

In addition, two more components, linked to each other, are presented in Figure 3.5, which are crucial for making the application trustworthy:

- The Wallet Secure Cryptographic Application (WSCA), that manages the keys associated to a specific Wallet Instance and the cryptographic operations that involve them. WSCA interfaces with the Wallet Secure Cryptographic Device (WSCD), a secure environment to store keys and execute critical functions. A WSCD could be integrated into the user's device, or be an external hardware component owned by the user (smart card), or even a remote device implemented by the Wallet Provider (HSM).
- The Wallet Provider backend, that manages the maintenance and initialization of Wallet Instances, issuing and eventually updating for each one:

– A Wallet Trust Evidence (WTE), information object containing a public key, paired with a private key stored by a certified WSCA/WSCD. When the user requests the issuance of a document, the Provider asks the WSCA to generate a new key pair before issuing the document in such a way that the new private key is protected by that WSCA, providing the same security level as the WTE key.
– A Wallet Instance Attestation (WIA), that contains the information needed for both Providers and Relying Parties to verify that the Wallet Instance is still valid and not revoked or suspended.

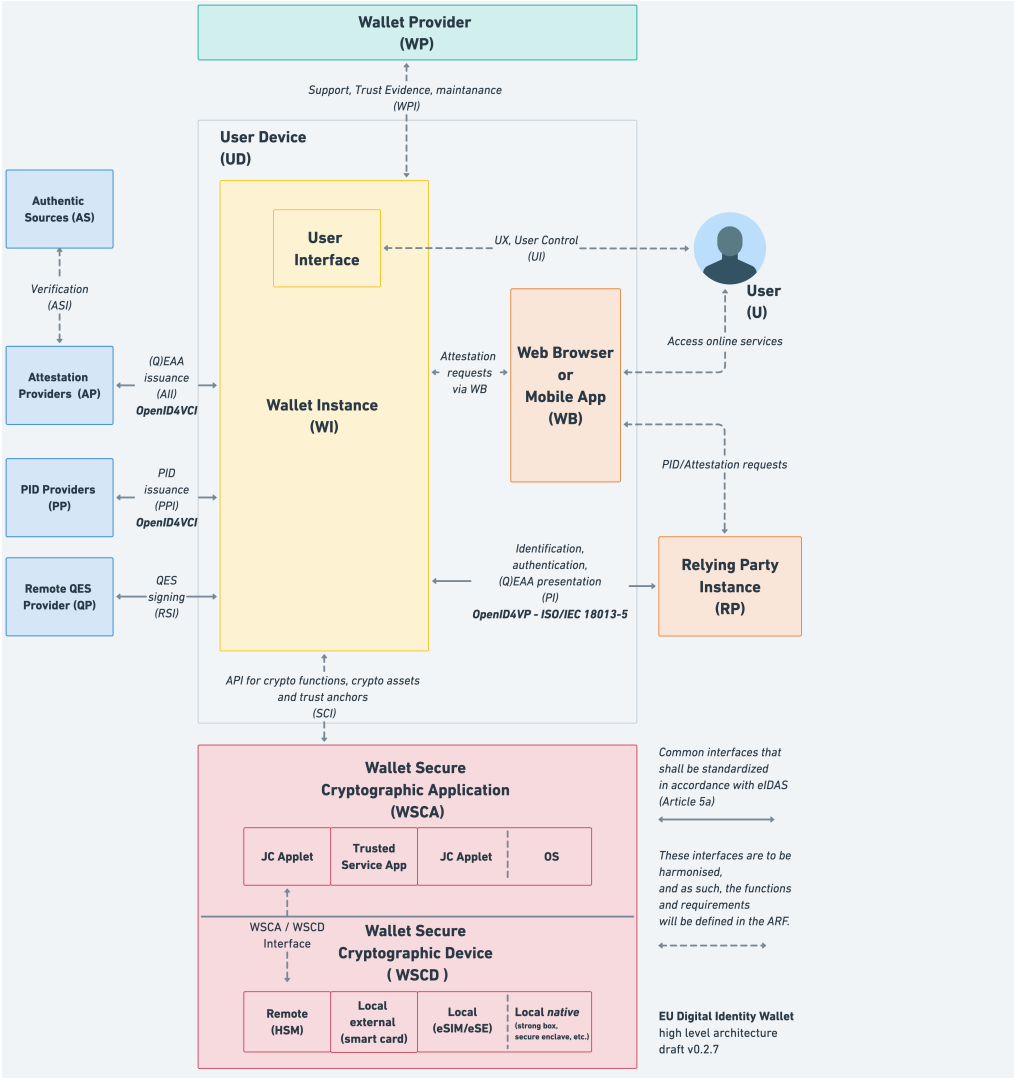

Figure 3.5. High-level overview of the EUDI Wallet infrastructure, including protocols and data formats used during the interactions between entities.[36]

Wallet lifecycle and Trust

After the development process is finished, a set of states can be used to represent the current condition of the Wallet Solution. The first one is the Candidate state, in which the application is subject to tests and validation procedures, in order to get a certification and be approved by the Member State. If certified, the application can be published and offered to citizens: its state changes to Valid, and it can be added to the Wallet Provider Trusted List. From now on, the Solution's state will not change unless a critical issue is detected in the application: in this case, the Member State can suspend its usage and block the distribution until the issue is fixed, shifting

the state to Suspended. When the problem is solved, the state can change again to Valid, resuming all the Wallet functionalities. The Member State could also choose not to support that solution any more, completely removing it from the EUDI Wallet Solutions offered for its citizens, which changes the state to Withdrawn permanently. All transitions between states are summarized in Figure 3.6.

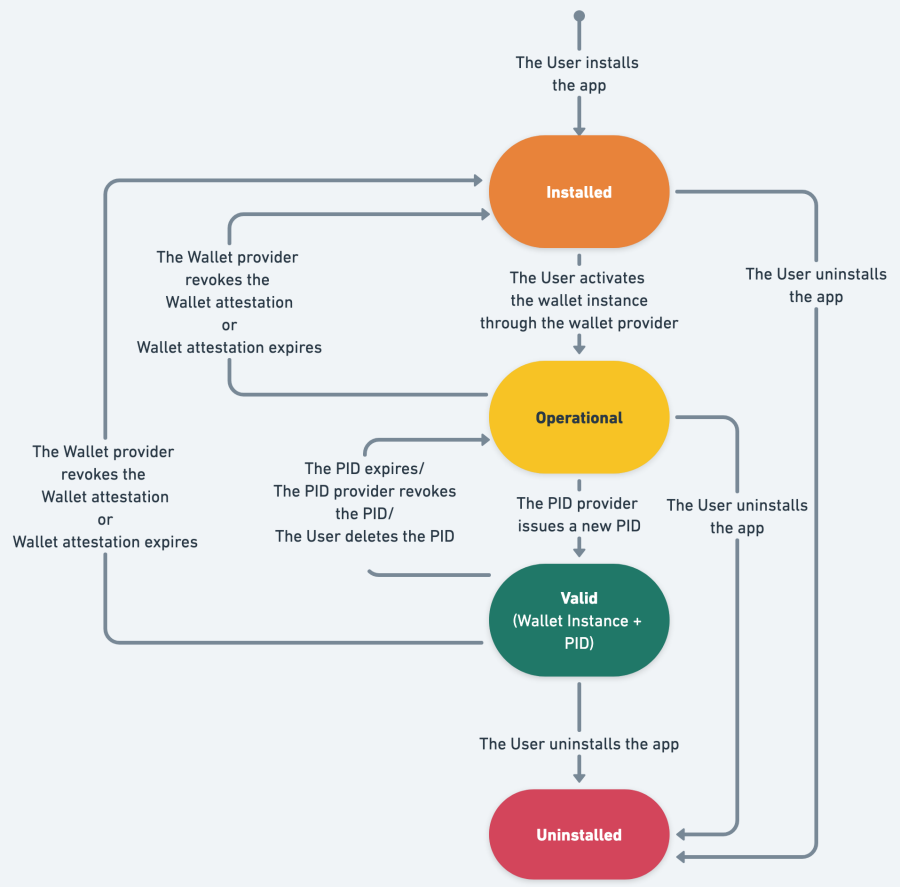

Figure 3.6. Interactions between the Wallet Instance and services.[36]

When users install a Wallet Solution, they come into possession of a new Wallet Instance on their device. The activation process of the instance will start, performing mutual authentication with the Wallet Provider, which is the responsible for this procedure. First, the provider retrieves data about the user device, such as the supported communication technologies and the Wallet Secure Cryptographic Device chosen to store the wallet related cryptographic keys; in the absence of a WSCD, a remote HSM will be assigned to the device. Then, the user is required to configure at least one authentication method to protect the access to the Wallet Instance, for example a PIN, biometrics or even a mechanism integrated within the WSCD: this will ensure that only authorized users can access the application, or initiate a Verifiable Presentation to third parties.

As a last step before becoming fully operational, a Wallet Instance Attestation and a Wallet Trust Evidence are issued from the Wallet Provider to the instance. Besides containing metadata that can be used by other entities to check whether the instance has been revoked, the issued public key, paired with a private key stored in the WSCD, is used during both credential issuance and presentation as a proof of possession of its counterpart. For example, during the issuance of a PID document:

1. The Wallet Instance includes the WIA in the issuance request sent to the PID Provider. The attestation is signed by the Wallet Provider and contains the WTE public key, along with metadata describing the wallet security characteristics.

2. The Provider downloads the Wallet Provider Trusted List, checking for a match against the requesting Wallet Instance. After that, it verifies the signature over the WIA by using the public key retrieved from the Trusted List, ensuring that the PID Provider is authentic and in a valid state. Finally, it performs a cryptographic challenge, with the goal of proving whether the Wallet Instance possesses the private key corresponding to the public key included in the WIA.

3. If the challenge is successful, trust is established between the two parties: the wallet generates a new asymmetric key pair, and sends the public key to the PID Provider. The key will be included in the issued credential, and will be used in the future to prove the ownership of that credential.

Recovery and risk management

The end of the ARF defines some guidelines to follow in case of device loss or theft, as well as pointing out what happens to user data after uninstallation. For the first case, to protect their personal information, users will need to have a registered account at the Wallet Provider to request the revocation of the instance, which can be performed after an authentication procedure. The registration can be performed by using pseudonyms, as it will not require the user to insert their personal attributes. Instead, if the user wants to uninstall the application, no additional steps are required: the WSCA corresponding to the instance will automatically delete all sensitive data and cryptographic keys, ensuring that no personal information or credential material remains accessible on the device. This procedure prevents any misuse of the Wallet Instance in case the device is later resold, transferred or compromised.

Chapter 4

Issuance and Presentation flows: Study and Analysis

The objective of this chapter is to analyse in detail the credential issuance and presentation flows supported by the EUDI Wallet ecosystem, using a local environment that replicates the architecture proposed under the eIDAS 2.0 framework. By reconstructing the infrastructure with custom-configured components, the experiment allows direct observation of interactions between the Wallet, Service Providers, Identity Providers, and eIDAS nodes. This makes it possible to verify the actual compliance of the implementation with the standards and protocols presented in the thesis, especially in terms of security, privacy, and interoperability.

Using the downloadable EUDI Wallet demo application, it is possible to request the issuance of various types of document from different countries, then store them in the Wallet and allow the user to present them to Verifiers or Service Providers. Basically, the process can be summarized with 4 steps:

1. By interacting with the Wallet application, the user chooses which document they want to request. If no documents are stored inside the Wallet, user must first request a PID document.
2. Through a web page, the user can now interact with a Service Provider and choose which country will be reached to issue the document.
3. After finishing a country-specific flow, a web page containing issued user attributes is displayed.
4. The user can authorize the receipt of the document, which will be stored in the Wallet application and displayed through the main user interface.

The implementation of the third step may differ from country to country, depending on how they are configured inside the Service Provider code. Analysing the current choices, *'FormEU'* allows the user to fill a form individually, customizing the values of the attributes: since this option does not provide any security feature, as well as not retrieving any of user data from a trusted entity that manages identities, this will be considered out of scope for the study.

Instead, other available country options are more interesting for this thesis work. Choices such as *'nodeEU'* or real countries like Portugal, Czechia, Netherlands and Luxembourg lead the user directly to the Identity Provider through the eIDAS architecture, starting from the node Connector configured by the developers of the Service Provider and interacting with the Proxy Service of the Sending Country (the one to which the user is requesting the issuance of the digital document). The other option is implemented for Estonia: instead, the Service Provider redirects the user to the Estonian test Service Provider, then the user is asked to choose the country to which they are requesting the issuance of the document, with the consequent redirection of the request through the eIDAS components. Even though this might seem a redundant step, it allows the user to interact with a wider choice of Sending Countries, each one configured inside the Estonian provider web application.

4.1 Overview of the chosen approach

For this study, the goal is to analyse used protocols, exchanged data formats and flows that are executed when requesting the issuance of a PID document and when sharing stored documents to a third party. Since the demo application relies on external services, which internal structure is not public, the whole environment will be recreated by using local services, providing the opportunity to deepen services implementation and configuration, and testing the interoperability of the environment adding a new country to the list of supported ones, including the related eIDAS node and Identity Provider.

To achieve this while not interfering with the realism of the experiment, an approach similar to the one used by Estonia is deployed for the thesis work. The original EUDI Wallet demo has been slightly modified to interact with custom-configured local components. Specifically, the Service Provider originally used by the demo application has been replaced by a locally hosted service, implemented in Python, which allows direct control over requests, responses, and the exchanged data formats. Furthermore, to fully replicate the real-world scenario involving cross-border authentication through the eIDAS architecture, a pair of local eIDAS nodes has been configured to simulate the behaviour of a Receiving Country interacting with an additional European Member State, named Utopia, complete with its own demo Identity Provider for user authentication. An overview of the components involved in the process and their interactions is presented in Figure 4.1.

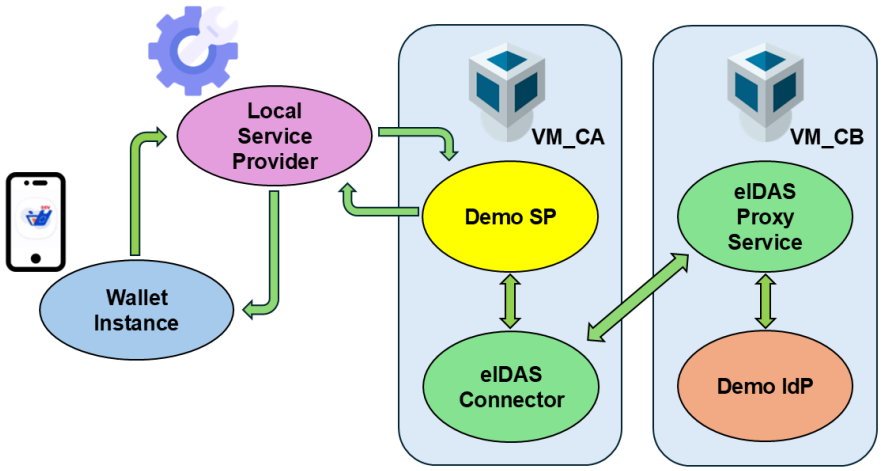

Figure 4.1. High-level interactions between the entities that are part of the tested local flow. Each component, compared to the original open source version, has been modified to allow communication with the others.

Each service used in this experimental setup has been developed or configured starting from already existing open-source implementations, freely available online, which initially lacked the capability of interacting with external components. The original versions provided basic functionality suitable for isolated testing but did not support communication between different services or integration within a local environment. Additionally, configured services will be hosted separately on independent machines, simulating the communication typical in the current distributed infrastructure that is promoted by the eIDAS 2.0 Regulation, and avoiding the trivialization of the model which might be encountered having all services hosted on the same machine. All modifications performed to enable seamless interoperability among these services are thoroughly documented in the following chapter, providing detailed guidelines for replicating and understanding each implementation step.

4.2 OpenID for Verifiable Credential Issuance

Before going into the details of the local implementation, the main protocol that allows authentication and authorization to be performed in the process must be presented. OpenID for Verifiable Credential Issuance (OpenID4VCI) draft 13 [40] is employed to manage the communication between the Wallet and the Credential Issuer, ensuring integrity, confidentiality and authentication throughout the process. OpenID4VCI acts as an extension of OpenID Connect: while with OIDC the user authenticates themselves and receives an ID token to prove their claim to a third-party Service Provider, with OpenID4VCI after the user authentication procedure is completed the Wallet receives a Verifiable Credential, that can be shared later with a Verifier using other protocols to as a proof of a claim made to access a service. The protocol supports two types of issuance methods, each one suited for different use cases; in the following subsections their usual flow is described, then compared with the flow of the local implementation that is subject of the study.

4.2.1 Pre-Authorization Code Flow

The Pre-Authorization Code Flow allows a third-party to start the issuance process of a document in advance, already knowing which attributes (and their related values) of the user must be included in the credential and provided that the user has previously authenticated themselves on the Issuer's platform. For example, assuming the user wants to request a mobile Driving License (mDL):

1. The user performs the login on the Issuer website, then requests the digital document.
2. The issuer generates a pre-authorized code, that is valid only for this specific transaction, and a QR code that includes the code and other data that identifies the issuer and the request.
3. With their Wallet application, the user can scan the received QR code. By doing that, information is decoded and used to send a request to the Authorization Server to get an access token.
4. After having validated the pre-authorized code, the Authorization server sends back to the Wallet the access token.
5. The Wallet uses the access token to retrieve the credential. Usually this credential request also includes the signature over a *c_nonce* previously sent from the issuer along with the access token.

Essentially, the procedure relies on the fact that the user previously completed an authentication procedure, in this case with the issuer website. The temporary validity and one-time usability of the pre-authorized code allow to avoid replay attacks, while a proof attached to the access token can be used by the issuer to confirm that the Wallet is the same entity that started the authorization request.

Using either the original EUDI Wallet demo version or the one modified to interact with the local environment, a Pre-Authorization Code flow can be simulated by accessing, respectively, the EUDI Service Provider or *https://host_IP_address:5000/credential_offer*. Submitting a document format (*sd-jwt vc* or *mdoc*) and type (PID, mDL, Loyalty...), the user can now input their personal details manually in a form. Then, a QR code and a transaction code will be displayed on the web page: if the user scans the code with their application, the issuance of a document which includes those personal information is started, requiring the transaction code as an additional security measure to complete the procedure. Although self-compiling a PID document is not intended to be a real-world application, this shows the concept of the pre-authorized transaction, in which the document can be generated by a third party, such as a company, and then redeemed by the user through the Wallet, giving them access to the QR and transaction code.

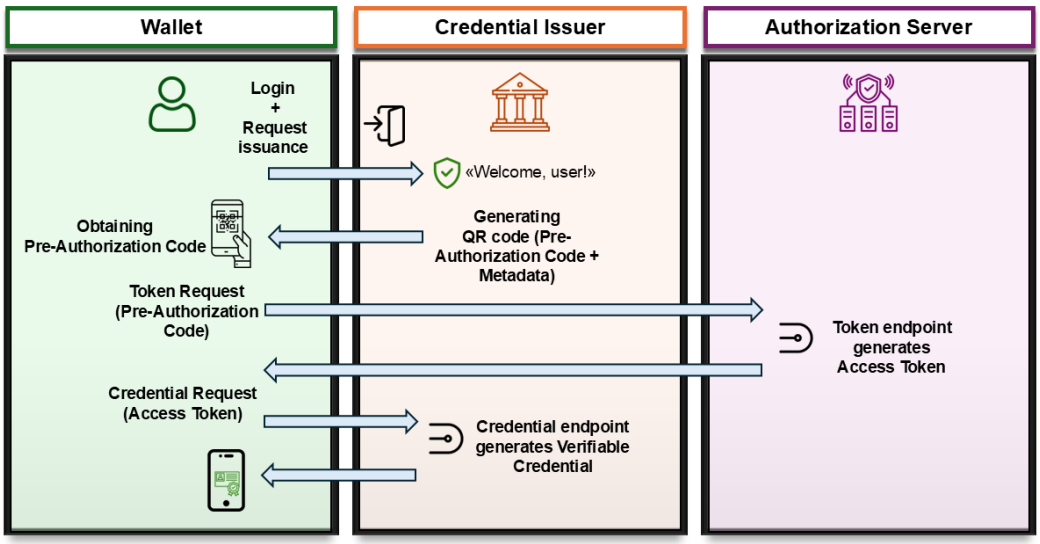

Figure 4.2. Interactions between entities in a Pre-Authorization Code flow.

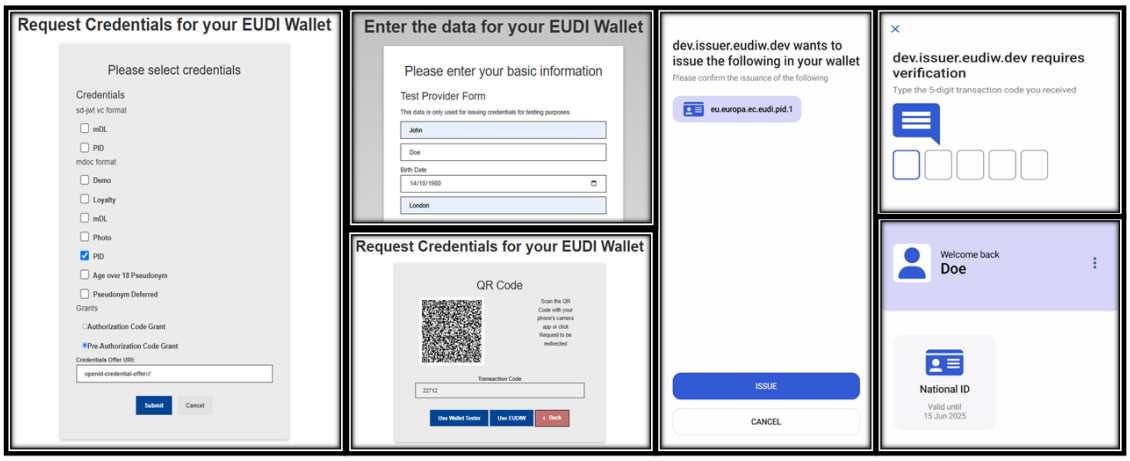

Figure 4.3. Pre-Authorization Code flow offered by the EUDI Wallet. Screenshots show how the user can request a document navigating through the web page of the Service Provider.

4.2.2 Authorization Code Flow

Unlike the Pre-Authorization Code Flow, Authorization Code Flow requires the Wallet to actively initiate the flow and allows the issuance of Verifiable Credentials only after the authentication process has taken place. Because of that, authentication is integrated during the issuance procedure, making use of a trusted third party (the Identity Provider) to verify user's identity. As done before, considering the user is requesting a mDL:

1. Starting from the Wallet application, an Authorization Request is generated, which parameters depend on the exact implementation of the protocol.

2. Through the web browser, the request is forwarded to the Authorization Server. The user must authenticate themselves by choosing an Identity Provider.

3. After the successful authentication, the Authorization Server generates an authorization code, sending it back to the Wallet.

4. A token request including the received authorization code is sent from the Wallet to the Token Endpoint of the Authorization Server. After having validated the code, an access token is generated and forwarded to the Wallet.

5. The Wallet uses the access token to retrieve the credential from the Credential Endpoint at the Credential Issuer.

The Authorization Code Flow is the one used to perform the issuance of credentials through the local environment: the choice is based on the stand-alone nature of that flow compared to the Pre-Authorization one, which instead assumes that the authentication procedure is executed beforehand by a third-party service. In addition, all the steps included in the Pre-Authorization Code flow are a subset of the second flow except for the starting interaction, not bringing any exclusive feature that can be analysed.

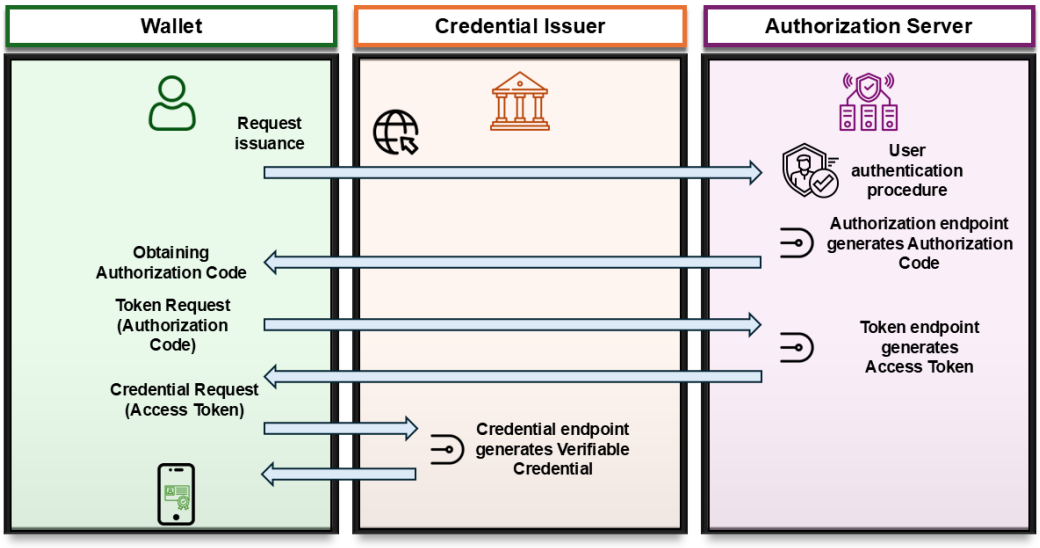

Figure 4.4. Interactions between entities in a Authorization Code flow.

For the purpose of the thesis, the implemented local Service Provider is considered as both an Authorization Server and a Credential Issuer. First, it manages the request received from the Wallet, redirecting the user to the delegated eIDAS infrastructure that manages the cross-border authentication mechanism. Upon successful authentication, receiving user personal data from the eIDAS network, the Service Provider acts as the Authorization Server, generating the authorization code and then the access token. The Wallet will use the access token to send a credential request to the Service Provider Credential Endpoint, that will process and validate it before creating and delivering the Verifiable Credential to the application, acting as the Credential Issuer.

4.3 Credential issuance flow using the local implementation of the Service Provider and eIDAS Node

The complete flow in Figure 4.5 shows how the Wallet, the local Service Provider and the eIDAS node interact with each other, during the issuance request started by the Wallet to add a new credential to its storage. Each step will be thoroughly analysed in the following subsections, highlighting the importance of the parameters contained within the requests and responses and how they impact the security of communication between the involved entities.

4.3.1 First steps in the application

In the case of the proposed Wallet Solution, on first use it will be necessary to set a 6-digit PIN, which will serve both as a security measure to protect access to the application and as an authorization mechanism for sensitive actions, such as credential sharing with third-party services. Then, the main UI of the app will load. In the first log-in, only three options will be available:

- National ID: to add more documents to their own Wallet, a National ID is needed. The thesis configuration allows the issuance of a National ID, hence this is the option that will be used for the scope.
- Load Sample Documents: it allows to load and store a pre-determined sample National ID on the Wallet, just for the sake of showing how a new document is displayed into the application. This process does not interact with any other service, and does not involve any security protocol.
- Scan QR: user can share their stored documents by scanning the QR code of the service that requests for it. Since credential verification is as important as the issuance process, this is also tested in the thesis work to confirm that credential issued with the local configuration can be shared and verified correctly by other services. This option can also be used to start a Pre-Authorized Code Flow, explained in the previous section.

The user can independently start the issuance process by clicking "National ID" (Figure 4.6) from the Wallet UI, and will be redirected to the web page on which the Service Provider is reachable. Once the National ID (from now on, referred as PID) is successfully issued and securely stored within the Wallet, the application can use its verified attributes as a trusted basis to request the issuance of additional, more specialized documents supported by the Wallet, such as a mobile Driving License or a Photo ID.

4.3.2 Collecting local Service Provider metadata

The client seeks to obtain information about the local Service Provider (the first step in 4.5): this occurs through two consecutive HTTP GET requests, reflecting the duality of the entity, once a TLS 1.3 connection has been established, to already known endpoints defined in the mobile application environment. The response to the first one includes information about the Verifiable Credentials supported, in JSON format (the *host_IP_address placeholder is used instead of the local IP address of the device*. Once set the local environment, metadata are obtainable by sending a request to https://host_IP_address:5000/.well-known/openid-credential-issuer .


```
{
  "batch_credential_endpoint": "https://192.168.1.56:5000/batch_credential",
  "credential_configurations_supported": {
    "demo": {[...]},
    "eu.europa.ec.eudi.hiid_mdoc": {[...]},
    "eu.europa.ec.eudi.iban_mdoc": {[...]},
    "eu.europa.ec.eudi.loyalty_mdoc": {[...]},
    "eu.europa.ec.eudi.mdl_jwt_vc_json": {{[...]},
```

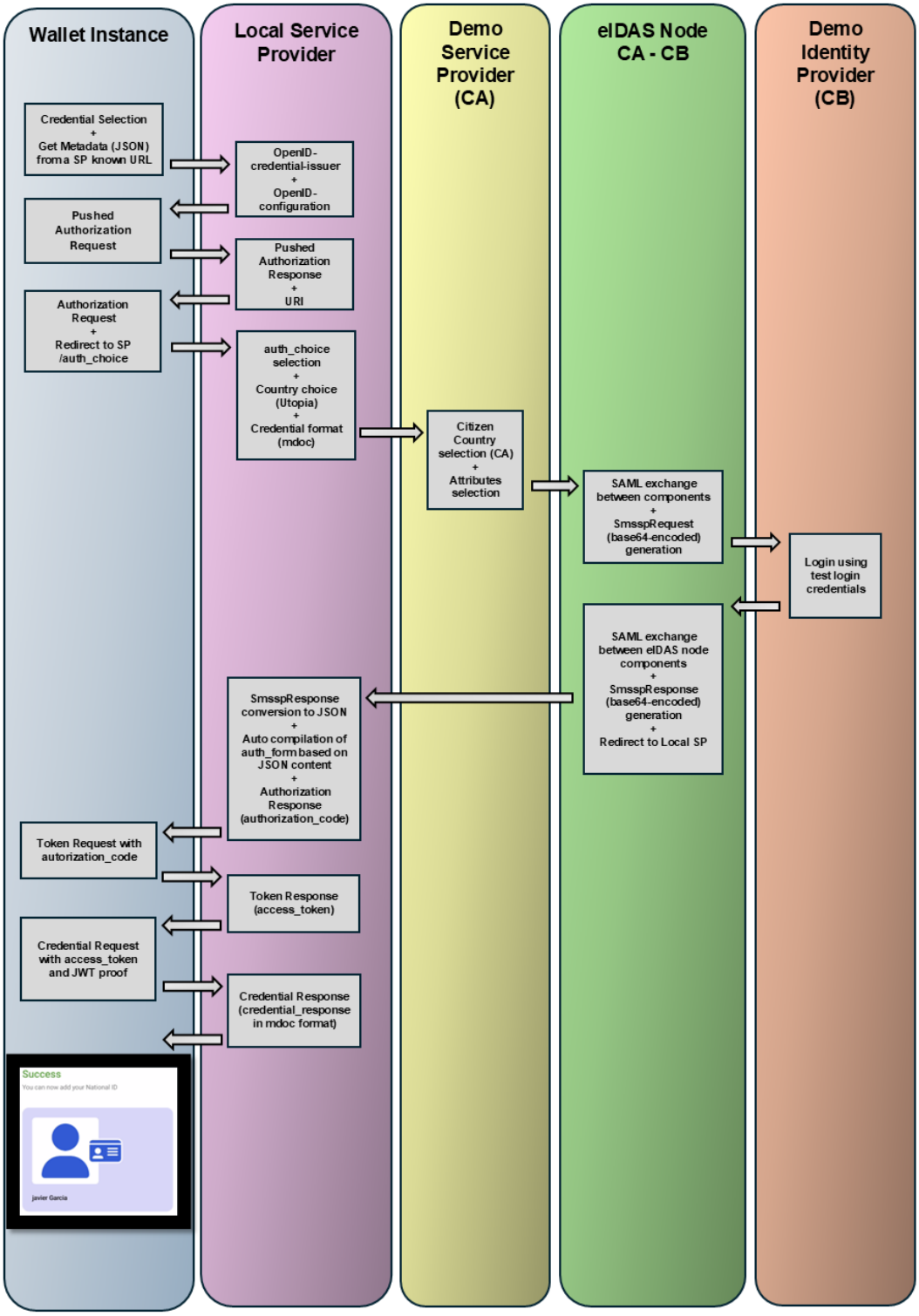

Figure 4.5. Issuance flow, with the local implementation of a Service Provider, two different eIDAS nodes and a demo Identity Provider.

```
"eu.europa.ec.eudi.mdl_mdoc": {[...]},
"eu.europa.ec.eudi.msisdn_mdoc": {[...]},
"eu.europa.ec.eudi.photoid": {[...]},
"eu.europa.ec.eudi.pid_jwt_vc_json": {[...]},
"eu.europa.ec.eudi.pid_mdoc": {
```

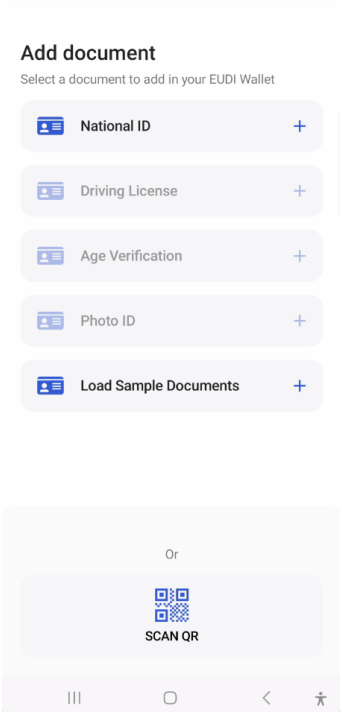

Figure 4.6. Main interface of the EUDI Wallet.

```
"claims": {
  "eu.europa.ec.eudi.pid.1": {
    [...]
    "age_birth_year": {
      "display": [
        {
          "locale": "en",
          "name": "Age Year of Birth"
        }
      ],
      "mandatory": false,
      "value_type": "uint"
    },
    [...]
    "age_over_18": {
      "display": [
        {
          "locale": "en",
          "name": "Age Over 18"
        }
      ],
      "mandatory": true,
      "source": "issuer"
    },
    [...]
    "expiry_date": {
      "display": [
        {
          "locale": "en",
          "name": "Expiry Date"
        }
```

```
  ],
  "mandatory": true,
  "source": "issuer"
},
"family_name": {
  "display": [
    {
      "locale": "en",
      "name": "Family Name(s)"
    }
  ],
  "mandatory": true,
  "source": "user",
  "value_type": "string"
},
"family_name_birth": {
  "display": [
    {
      "locale": "en",
      "name": "Birth Family Name(s)"
    }
  ],
  "mandatory": false,
  "source": "user",
  "value_type": "string"
},
"given_name": {
  "display": [
    {
      "locale": "en",
      "name": "Given Name(s)"
    }
  ],
  "mandatory": true,
  "source": "user",
  "value_type": "string"
},
"given_name_birth": {[...]},
"issuance_date": {
  "display": [
    {
      "locale": "en",
      "name": "Issuance Date"
    }
  ],
  "mandatory": true,
  "source": "issuer"
},
"issuing_authority": {
  "display": [
    {
      "locale": "en",
      "name": "Issuance Authority"
    }
  ],
  "mandatory": true,
  "source": "issuer"
```

```
            },
            [...]
        },
        "credential_alg_values_supported": [-7],
        "credential_crv_values_supported": [1],
        "credential_signing_alg_values_supported": [
          "ES256"
        ],
        "cryptographic_binding_methods_supported": [
          "jwk",
          "cose_key"
        ],
        [...]
        "doctype": "eu.europa.ec.eudi.pid.1",
        "format": "mso_mdoc",
        "policy": {
          "batch_size": 50,
          "one_time_use": true
        },
        "proof_types_supported": {
          "cwt": {
            "proof_alg_values_supported": [-7],
            "proof_crv_values_supported": [1],
            "proof_signing_alg_values_supported": [
              "ES256"
            ]
          },
          "jwt": {
            "proof_signing_alg_values_supported": [
              "ES256"
            ]
          }
        },
        "scope": "eu.europa.ec.eudi.pid.1"
      },
      "eu.europa.ec.eudi.por_mdoc": {[...]},
      "eu.europa.ec.eudi.pseudonym_over18_mdoc": {[...]},
      "eu.europa.ec.eudi.pseudonym_over18_mdoc_deferred_endpoint": {[...]},
      "eu.europa.ec.eudi.tax_mdoc": {[...]},
      "org.iso.18013.5.1.reservation_mdoc": {[...]}
    }
    "credential_endpoint": "https://host_IP_address:5000/credential",
    "credential_issuer": "https://host_IP_address:5000",
    "deferred_credential_endpoint":
        "https://host_IP_address:5000/deferred_credential",
    "notification_endpoint": "https://host_IP_address:5000/notification"
  }
```


Analysing its content, it is possible to find:

- A list of endpoint, including:
  - *credential_endpoint*, that manages the issuance of credentials as Verifiable Credentials, based on the information obtained from the Identity Provider.
  - *batch_credential_endpoint*, used to request multiple credentials at once, without repeating the whole procedure for each different one. The Batch Credential Request that is sent to this endpoint (instead of reaching the *credential_endpoint*) must include a different proof for each credential requested.

  - *deferred_credential_endpoint*, used if the Credential Issuer was not able to provide the issuance of the requested credential. Instead, a generated *transaction_id* is sent to the Wallet, that can be used later to retrieve Verifiable Credentials when they are ready to be issued.
  - *notification_endpoint*, that receives notifications from the Wallet about the outcome of the request.

- The *credential_issuer* field, which is the URL identifier of the Issuer, containing scheme, host and port number (if necessary).

- All the supported credentials for the Credential Issuer, each identified by a unique code (such as *eu.europa.ec.eudi.mdl_mdoc* for mobile Driving License in *mdoc* format) that is needed to initiate an issuance request from the Wallet. The structure of different credentials may slightly vary, using specific parameters. Since a PID is requested in this case, only the claims within the *eu.europa.ec.eudi.pid_mdoc* field will be analysed:

  - A list of attributes defines which ones are recognized and can be issued with the chosen type of document. For each attribute, one or more of these parameters can be specified:
    * *display*: provides a brief description of the attribute.
    * *mandatory*: determines if the attribute is optional for that document.
    * *source*: specifies a value depending on whether the attribute is directly provided (or confirmed) by the user (*user* value) or generated (or verified) by the Credential Issuer (*issuer* value). Attributes derived from the others, such as *age_over_18*, are considered as generated by the Issuer.
    * *value_type*: states the expected format for the value of the attribute.
  - A *format* parameter that specifies the format of the credential.
  - A set of supported cryptographic algorithms and security methods:
    * *credential_signing_alg_values_supported* defines the algorithm used to sign the issued credential.
    * *cryptographic_binding_methods_supported* declares how the Verifiable Credential are cryptographically bound to the user that owns the document.
    * *proof_types_supported* describes which key proofs can be used by the Wallet when sending the Credential Request along with the access token. For each proof, also the related supported algorithm are listed. For a PID in *mdoc* format, both JSON Web Token (*jwt*) and CBOR Web Token [41] (*ctw*) are supported as proofs.
    * *scope* uniquely identifies the resource being requested, to the Authorization Server.

While the first request treated the Service Provider as a Credential Issuer, with the second one (https://host_IP_address:5000/.well-known/openid-configuration) information about the Authorization Server is demanded. Again, a JSON is received:


```
{
  "authorization_endpoint": "https://host_IP_address:5000/authorizationV3",
  "backchannel_logout_session_required": true,
  "backchannel_logout_supported": true,
  "claims_parameter_supported": true,
  "code_challenge_methods_supported": [
    "S256"
  ],
  "credential_endpoint": "https://host_IP_address:5000/credential",
  "end_session_endpoint": "https://host_IP_address:5000/session",
  "frontchannel_logout_session_required": true,
  "frontchannel_logout_supported": true,
  "grant_types_supported": [
    "authorization_code",
    "implicit",
    "urn:ietf:params:oauth:grant-type:jwt-bearer",
```

```
  "refresh_token"
],
"id_token_signing_alg_values_supported": [
  "RS256",
  "RS384",
  "RS512",
  "ES256",
  "ES384",
  "ES512",
  "PS256",
  "PS384",
  "PS512",
  "HS256",
  "HS384",
  "HS512"
],
"introspection_endpoint": "https://host_IP_address:5000/introspection",
"issuer": "https://host_IP_address:5000",
"jwks_uri": "https://host_IP_address:5000/static/jwks.json",
"pushed_authorization_request_endpoint":
    "https://host_IP_address:5000/pushed_authorizationv2",
"registration_endpoint": "https://host_IP_address:5000/registration",
"request_object_signing_alg_values_supported": [
  "RS256",
  "RS384",
  "RS512",
  "ES256",
  "ES384",
  "ES512",
  "HS256",
  "HS384",
  "HS512",
  "PS256",
  "PS384",
  "PS512"
],
"request_parameter_supported": true,
"request_uri_parameter_supported": true,
"require_request_uri_registration": false,
"response_modes_supported": [
  "query",
  "fragment",
  "form_post"
],
"response_types_supported": [
  "code"
],
"scopes_supported": [
  "openid"
],
"subject_types_supported": [
  "public",
  "pairwise"
],
"token_endpoint": "https://host_IP_address:5000/token",
"token_endpoint_auth_methods_supported": [
  "public"
```

```
  ],
  "userinfo_endpoint": "https://host_IP_address:5000/userinfo",
  "userinfo_signing_alg_values_supported": [
    "RS256",
    "RS384",
    "RS512",
    "ES256",
    "ES384",
    "ES512",
    "PS256",
    "PS384",
    "PS512",
    "HS256",
    "HS384",
    "HS512"
  ],
  "version": "3.0"
}
```


This response introduces:

- The server identifier (*issuer*) and OpenID Connect provider version (*version*).
- A set of endpoints typical for Authorization Servers:
  - *authorization_endpoint*, to which the client sends the Authorization Request, initiating the flow.
  - *credential_endpoint*, duplicated from the previous JSON, due to the fact that Authorization Server and Credential Issuer can be two different entities.
  - *end_session_endpoint*, that allows the client to terminate a user session, invalidating the proofs already exchanged, and revoking granted authorizations.
  - *introspection_endpoint*, which can send metadata about an access token on demand, including parameters about its current validity (*active*) and type (*tokenType*), generation and expiration timestamps (*iat* and *exp*) or the identifier of the entity that requested it (*client_id*).
  - *pushed_authorization_request_endpoint*, reached instead of the *authorization_endpoint* from the Wallet through a Pushed Authorization Request [42] (PAR). In the proposed flow, a PAR is preferred instead of the Authorization Request, due to the additional security features that provides to the communication.
  - *registration_endpoint*, that enables clients to register themselves to the Authorization Server dynamically.
  - *token_endpoint*, which issues the access token in exchange for the previously generated authorization code.
  - *userinfo_endpoint*, which can return the user claims upon receiving a valid access token.
- A list of algorithms supported by the Authorization Server to sign ID tokens or request objects coming from the client, and supported Proof Key for Code Exchange [43] (PKCE) code challenge methods (*code_challenge_methods_supported*).
- *jwks_uri*, an URL where the public keys are published to allow the verification of the Authorization Server signature.
- *grant_types_supported*, that specifies which elements are needed for the client to request an access code from the Authorization Server.
- *response_modes_supported*, listing the possible shapes in which the authorization response is returned to the client.

- *response_types_supported* that states, in this case, that only the Authorization Code flow is supported, having *code* as the only value.
- *subject_types_supported*, which determines how the user is represented within the ID token. This Authorization Server allows to have a unique subject identifier (*sub*) per client (*pairwise* value) preventing different services from linking the user's activities without explicit consent. By default, the same identifier is used for all clients.
- How the Wallet can send the parameters attached to the authorization request. In this case, rather than sending the parameter inside the request, it is possible to provide a URI that includes those parameters (*"request_uri_parameter_supported": true*).
- Logout mechanisms, supporting both front-channel and back-channel logouts by using the session ID.

4.3.3 Pushed Authorization Request and Authorization request

Once retrieved all the necessary information to extablish a secure communication with the local Service Provider, the Wallet generates a Pushed Authorization Request (PAR) to the *pushed_authorization_request*. PAR is a particular Authorization Request that involves a backchannel, thus it does not expose the authorization parameters in the browser. This can provide protection against tampering over the information included in the request. The PAR, in this flow, includes the following parameters:

- `'client_id': 'wallet-dev',`

  The identifier of the Wallet Instance.

- `'response_type': 'code',`

  States that an authorization code is requested, to be later exchanged for an access token. The *response_type* value must be consistent with the *response_types_supported* included in the previous metadata.

- `'redirect_uri': 'eu.europa.ec.euidi://authorization',`

  URI where the Wallet will be redirected when receiving the authorization response.

- `'scope': 'eu.europa.ec.eudi.pid.1',`

  Identifier of the requested credential, according to the metadata received from the Credential Issuer.

- `'state': '0RnuOaVPIMSFg_ZIBwjUVzp6RDwU3-BOeRcRzIis7c8',`

  Randomly generated value that provides protection against Cross-Site Request Forgery (CSRF). This value will be bound to the request; at the end of the flow, the local Service Provider will return the value of 'state', allowing the Wallet to prove that the response is bound uniquely to its request.

- `'code_challenge': '3bdkv7KpsjS0qXaZXFwAXCgR0ibPF78gDWxpXIXxZ3I',`
  `'code_challenge_method': 'S256'`

  Protection against replay attacks using PKCE. The Wallet generates a high-entropy string named *code_verifier*, then hashes it using the chosen *code_challenge_method* and sends it to the local Service Provider. Later in the flow, the Service Provider will get access to the *code_verifier* to prove that the *code_challenge* was derived from it.

Upon receiving the request, the Service Provider generates a session ID for the user. Then, it returns a response:

```
{
    'expires_in': 3600,
    'request_uri': 'urn:uuid:1f27bdf1-ede0-49f4-b948-2274553e8788'
}
```

The URI, valid for 1 hour (3600 seconds), is uniquely associated to the previous PAR and ensures that the original parameters previously sent to the Authorization Server cannot be altered. Despite PAR, the Wallet must still generate the Authorization Request for the *authorization_endpoint*, redirecting the user to the web page of the local Service Provider with an URL composed of previously exchanged information:

```
https://[...]]:5000/authorization?redirect_uri=eu.europa.ec.euidi:
//authorization&response_type=code&scope=eu.europa.ec.eudi.pid.1
&client_id=wallet-dev&request_uri=
urn:uuid:1f27bdf1-ede0-49f4-b948-2274553e8788'
```

Again, any modification to the URL will make the connection fail, being strictly bound to the *request_uri* and the other parameters.

4.3.4 Choosing the PID Provider

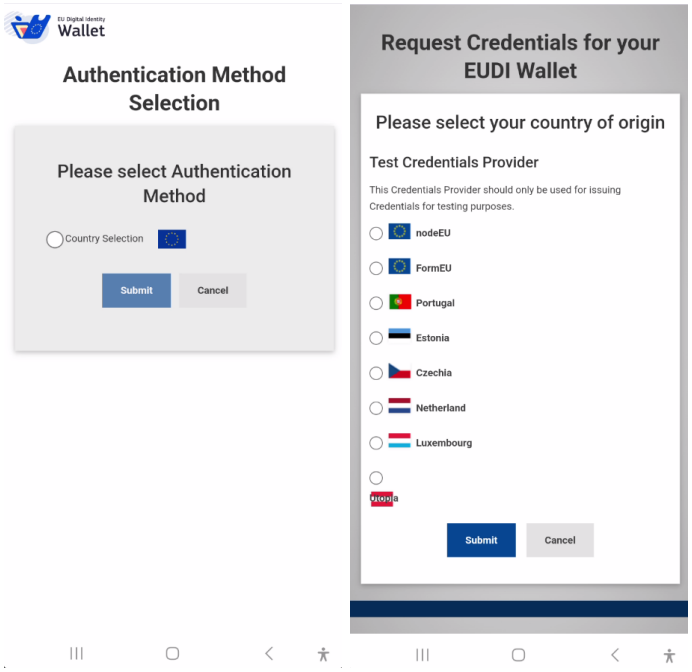

Figure 4.7. Web interfaces guiding the user to choose the PID Issuer country.

User is now presented with a web page as the one on the left screenshot in Figure 4.7. Click the only available option 'Country Selection' to be redirected to the web page showed on the right screenshot of the same Figure. Here, the user can choose the country from which the PID document will be issued (that can be different from the one that performs authentication). Countries listed here have their own interaction with services that are excluded from the thesis work, except for 'FormEU', which allows user to create its own credential filling a form of attributes. However, this is just for testing purposes, as those credentials are unsigned and not verified or usable by no means, except to test if the storage mechanism of the Wallet Instance works, without bothering about the authentication procedure. By choosing the 'Utopia' country, the flow will continue, and the user will be redirected to the demo Service Provider included in the eIDAS node setup.

In the local Service Provider configuration, the Utopia country is defined to support only PID documents in *mdoc* format, hence the Wallet will send a POST Request specifying the following:

```
authorization_details=["openid",%20
{"credential_configuration_id":%20"eu.europa.ec.eudi.pid_mdoc"}]
```

Please note that, depending on the country configuration in the local Service Provider files, the *authorization_details* structure may change. For example, if the local Service Provider was designed to interact directly with an eIDAS node, as it is designed for the 'nodeEU' country in the original demo version of the EUDI Wallet, instead of using *"openid"* as a value it would have been used *"eidasnode"*.

4.3.5 Demo Service Provider and eIDAS node flow

After selecting Utopia as the issuing country, a new web page will be opened, displaying the Demo Service Provider interface as shown in the Figure 4.8 that acts as an intermediary between the Wallet and the eIDAS infrastructure. From the parameters listed in this page, the request that will be sent through the eIDAS node to the demo Identity Provider is built:

- SP Country and Citizen Country: they define the roles of each country involved in the SAML exchange for the authentication process. As mentioned earlier in the chapter, the SP Country is the fictitious country CA (corresponding to Utopia, that will issue the credential), while Citizen Country represents the one that performs user authentication (thus the one for which the user is supposed to own a physical document), in this case delegated to the fictitious country CB.
- SP Return URL: set by default to the URL that redirects the flow back to the Service Provider after having received the document in base64-encoded format on the specific Connector component of the node.
- Name Identifiers: used in SAML protocol to identify users. 'Persistent' generates an identifier that will be used even for future sessions of the same user, while 'Transient' generates it each session.
- Level of Assurance (LoA): scaling from E to A, determines the degree of security behind the authentication procedure, issuance and management of the digital document. A low LoA may be rejected in the process, hence selecting 'A' is strongly advised.
- Requested core attributes: allows the selection of which user attributes will be retrieved from the Identity Provider, upon successful authentication.

After clicking 'Submit', a new page will be loaded, showing a preview of the JSON that is part of the Authentication Request. For example, leaving the previous 'Requested core attributes' as their default selection, the JSON should include:

```
{
   "authentication_request" : {
      "attribute_list" : [ {
         "type" : "requested_attribute",
         "name" : "BirthName",
         "required" : false
      }, {
         "type" : "requested_attribute",
         "name" : "CountryOfBirth",
         "required" : false
      }, {
         "type" : "requested_attribute",
         "name" : "CountryOfResidence",
         "required" : false
      }, {
         "type" : "requested_attribute",
```

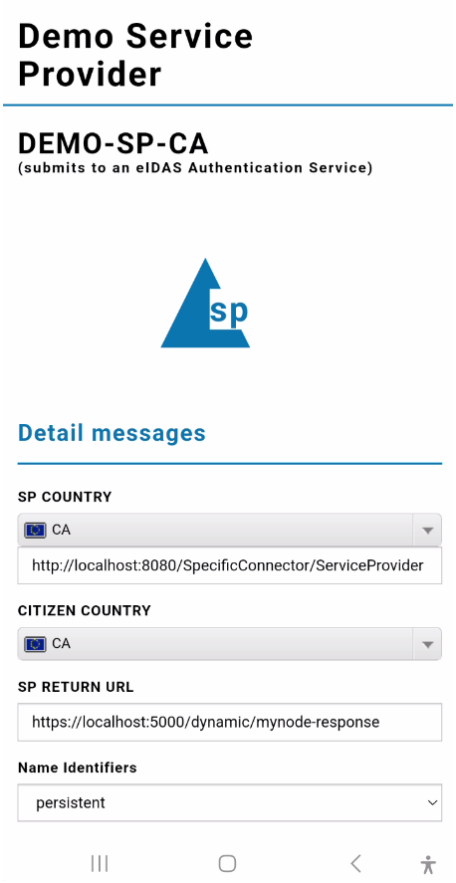

Figure 4.8. Demo Service Provider interface, configured with the eIDAS node.

```
   "name" : "CurrentAddress",
   "required" : false
}, {
   "type" : "requested_attribute",
   "name" : "FamilyName",
   "required" : true
}, {
   "type" : "requested_attribute",
   "name" : "FirstName",
   "required" : true
}, {
   "type" : "requested_attribute",
   "name" : "DateOfBirth",
   "required" : true
}, {
   "type" : "requested_attribute",
   "name" : "EmailAddress",
   "required" : false
}, {
   "type" : "requested_attribute",
   "name" : "Gender",
   "required" : false
}, {
   "type" : "requested_attribute",
   "name" : "Nationality",
```

```
            "required" : false
         }, {
            "type" : "requested_attribute",
            "name" : "PersonIdentifier",
            "required" : true
         },
         [...], {
            "type" : "requested_attribute",
            "name" : "eJusticeNaturalPersonRole",
            "required" : false
         } ],
         "requested_authentication_context" : {
            "comparison" : "minimum",
            "context_class" : [ "A" ]
         },
         "citizen_country" : "CB",
         "created_on" : [...],
         "force_authentication" : true,
         "id" : "4244a49a-30b7-4480-8b92-88554eff4f7d",
         "provider_name" : "DEMO-SP-CA",
         "requester_id" : "http://eidas.eu/EidasNode/RequesterId_CA",
         "serviceUrl" : "https://[...]/dynamic/mynode-response",
         "sp_type" : "public",
         "version" : "1"
      }
   }
```


The authentication request contains:

- A list of attributes, either mandatory or optional, according to the eIDAS minimum data set for Natural Persons mentioned in the chapter related to eIDAS 2.0.
- *requested_authentication_context*, which specifies how the LoA level of the other country is used to determine if the security level is acceptable. In this case, *"comparison" : "minimum"* means that any LoA that is equal or greater than the one requested is enough to perform the exchange.
- Data referred to the involved entities, the user and the process: sender, receiver and request are uniquely identified, and *"force_authentication" : true* establishes that, even if there is an already active session for the user, the authentication procedure must be performed, preventing sessions reuse.

The request is then sent through the eIDAS infrastructure, starting from the node of the CA country and reaching the CB Identity provider. The whole flow, shown in detail in the Figure 4.9, is also commented here:

1. First, the Service Provider forwards the request (MSRequest) to the specific Connector of CA. This component acts as an interface between the SP and the eIDAS network.
2. The specific Connector translates the received request into a LightRequest, a lightweight format that allows to manage internal communications between the specific and generic parts of the same node, without processing an entire SAML Request. A unique LightToken is generated, as a reference to the LightRequest that is stored by the specific Connector.
3. The generic part of the CA Connector receives the LightRequest and converts it to a SAML Authentication Request, following the standards for the SAML protocol used in the eIDAS architecture, encrypting the content and signing the message. This request can be forwarded cross-border to CB Proxy Service, after performing mutual authentication.

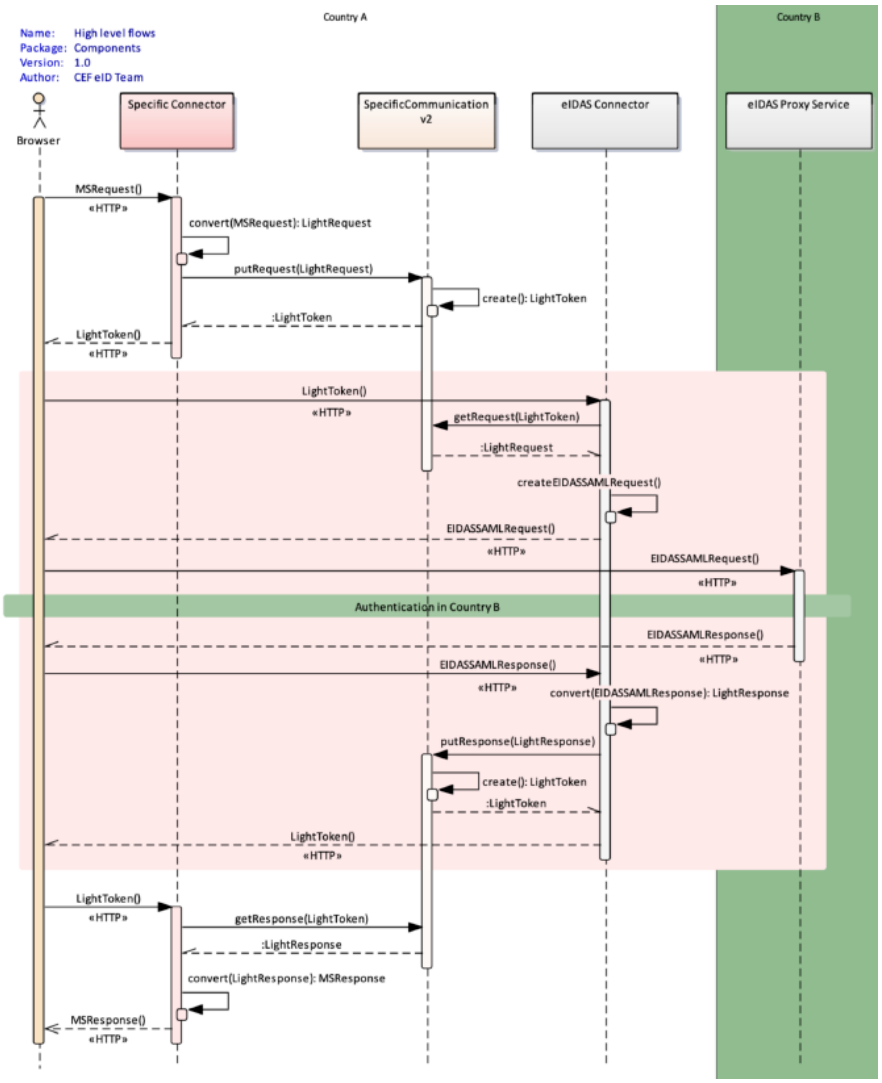

Figure 4.9. The eIDAS flow from the point of view of the Receiving Country.[44]

4. The generic part of CB Proxy Service performs some validation checks over the request coming from the CA Connector. In particular, first it verifies the digital signature of the Connector against the public certificate that was previously exchanged to establish the connection, then checks if the requested LoA is compatible with the Member State policies and if the requested attributes are a valid subset of the eIDAS Attribute Set (Figure 4.10), either for a Legal or Natural Person.

5. Mirroring the role of the specific Connector, the specific Proxy Service acts as the intermediary between the rest of the node and the local Identity Providers. Receiving the SAML Request, it translates it to a format that can be accepted by the IdP. In this case, the request sent to the Identity Provider is base64-encoded SmsspRequest, and its content is similar to the starting request, with the exception of the attributes that were excluded from the request (the ones with *"required" : false*).

6. The user now loads the demo Identity Provider web interface, shown in Figure 4.11, and can perform login using the following test credentials:

   ```
   username: xavi
   password: creus
   ```

   Upon login, user personal data that were asked in the SmsspRequest are displayed on the

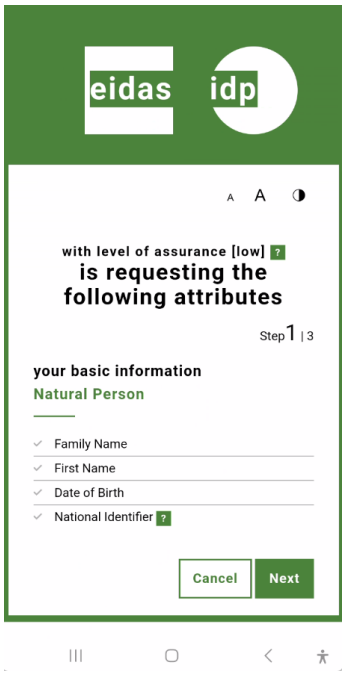

Figure 4.10. Validation of attributes at the Proxy Service.

screen, notifying that the authentication procedure was performed correctly and that the initial request was preserved during the exchange.

Figure 4.11. Login screen of the demo IdP.

7. The IdP generates an SmsspResponse, base64-encoded. The rest of the flow will reflect how the request was generated and forwarded, but following the opposite direction: first, the specific Proxy Service generates a LightResponse starting from the response obtained from the IdP, then the generic Proxy Service converts it in a SAML Response that is sent to the

generic Connector of the CA country; upon successful validation, it translates the message to a LightResponse, which can be processed by the specific Connector in CA. If decoded, the received SmsspResponse is:


```
{
   "response" : {
       "attribute_list" : [ {
           "type" : "string_list",
           "name" : "FamilyName",
           "values" : [ {
               "value" : "Garcia"
           } ]
       }, {
           "type" : "string_list",
           "name" : "FirstName",
           "values" : [ {
               "value" : "javier"
           } ]
       }, {
           "type" : "date",
           "name" : "DateOfBirth",
           "value" : "1964-12-31"
       }, {
           "type" : "string_list",
           "name" : "PersonIdentifier",
           "values" : [ {
               "value" : "CB/CA/12345"
           } ]
        } ],
       "authentication_context_class" : "B",
       "created_on" : "[...]",
       "id" : "4b1c02ca-8b42-475b-a261-e4a3749c2203",
       "inresponse_to" : "5312729a-5d81-4bc6-b4f2-435fe9fff50b",
       "issuer" : "CB",
       "name_id" : "unspecified",
       "status" : {
           "status_code" : "success"
       },
       "subject" : "31D457357",
       "version" : "1"
   }
}
```


Where *inresponse_to* points the identifier of the SmsspRequest, *status* confirms that the procedure was successful, and the *subject* value represent an identifier that is used within the IdP.

8. The CA Service Provider forwards the response to the local Service Provider, as a base64-encoded SmsspResponse, containing all the information that were initially requested.

4.3.6 Authorization Response and Token Exchange

The user is now redirected to the local Service Provider. Here, the SmsspResponse received by the eIDAS specific Connector (last component of the chain in the eIDAS infrastructure) is processed and converted in JSON format, and used to automatically populate a dynamic form. This form contains all the attributes retrieved from the SmsspResponse, along with attributes that can be directly inferred from data in the response (*age_over_18*, *estimated_issuance_date*) or which can only be added by a Credential Issuer to create a PID document (*issuing_authority*,

estimated_expiry_date). For example, starting from the information obtained from the response above, it is possible to extract the following JSON:

```
{
   'family_name': 'Garcia',
   'given_name': 'javier',
   'birth_date': '1964-12-31',
   'estimated_issuance_date': '2025-02-17',
   'estimated_expiry_date': '2025-05-18',
   'issuing_country': 'UT',
   'issuing_authority': 'Utopia ID Authority',
   'age_over_18': True
}
```

The form is shown in Figure 4.12. Now, clicking the 'Authorize' button, the previous authorization process will continue, verifying if the user is entitled to receive those credentials and store them in the Wallet.

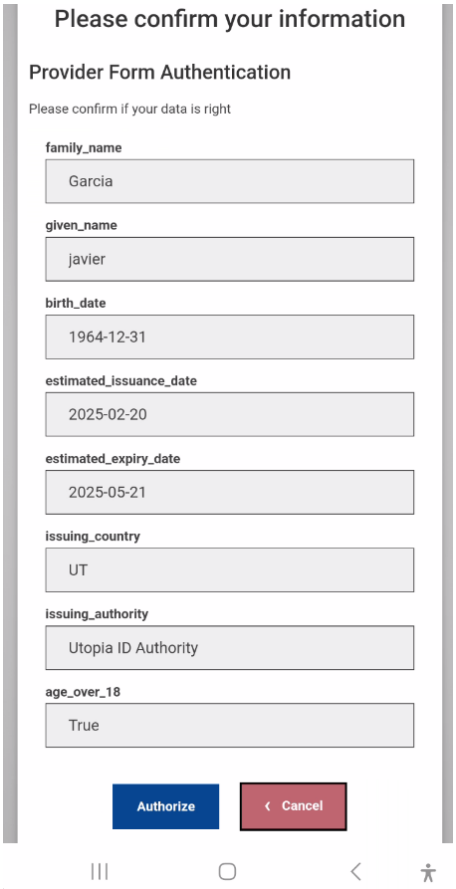

Figure 4.12. Auto filled form after receiving user data in JSON format.

Upon receiving the form submission, the local Service Provider provides the Authorization Response to the Wallet:

```
Code: [...],
State: ORnuOaVPIMSFg_ZIBwjUVzp6RDwU3-BOeRcRzIis7c8
```

The response is bound to the same *state* value that was included from the Wallet within the PAR. A deep link is detected on the user's mobile device, opening the Wallet application.

Including the received authorization code, the Wallet sends a POST request to the *token_endpoint* of the local Service Provider:

```
'client_id': 'wallet-dev',
'grant_type': 'authorization_code',
'code': [...],
'redirect_uri': 'eu.europa.ec.euidi://authorization',
'code_verifier': 'dmxnDv-pMBldjpgIOsgGkJeM_JmN1tMMyqW-T4TBwos'
```

The *code* parameter is the authorization code just obtained, and the *grant_type* specifies that the Authorization Code flow is used for this transaction. The *code_verifier*, from which was derived the *code_challenge* sent to the local Service Provider with the PAR, is used to compute again the challenge (in this flow using SHA256) on the Service Provider side to verify if both codes match. This ensures that, if an attacker intercepts an authorization code issued by the Authorization Server, the code cannot be reused, because the *code_verifier* is stored only in the client that started the Authorization Request.

From the *token_endpoint*, if the PKCE verification succeeded and the authorization code is valid, the client will now receive the *access_token*:

```
{
    "token_type": "Bearer",
    "scope": "eu.europa.ec.eudi.pid.1",
    "access_token": [...],
    "expires_in": 3600,
    "refresh_token": "[...]",
    "id_token": "[...]"
}
```

The *access_token* value is a signed JWT (the three part-structure *header.payload.signature* can be seen in the token value), that will be used to make the final Credential Request. The *refresh_token* allows the user to obtain a new *access_token* without the need of performing authentication again, having a longer expiration time. The *id_token* is a signed JWT related to the user's identity information, which is not used for granting access to resources. These parameters included in the decoded id_token:

- *'iss'*: Credential Issuer's URL (the local SP).
- *'sub'*: a unique identifier for the user, also included in the *access_token*.
- *'sid'*: a unique identifier for the session, also included in the *access_token*.
- *'aud'*: determines who can use the token (in this case, it states the *client_id*, also included in the *access_token*.
- *'acr'*: Authentication Context Class Reference, the level of assurance behind the authentication procedure.
- *'iat'*: date and time for which the token is issued.
- *'exp'*: date and time for which the token expires.
- *'jti'*: unique identifier (JWT ID) used to prevent replay attacks, a different one is also included in the *access_token*.
- *'nonce'*: provides a randomly generated code to the client, that must be used to retrieve the credential.

4.3.7 Credential Request and Response

In the end, a Credential Request is sent to the *credential_endpoint*. The request includes the *access_token* received beforehand, the format for the requested credential (*mso_mdoc*) and a proof of possession of a private key to which the Verifiable Credential will be bound to, according to what the local Service Provider included in the *credential_configurations_supported* within its

metadata. In this case, JWT is used as the *proof_type*, containing in the payload the *client_id* (*iss*), the Credential Issuer identifier (*aud*), the timestamp at which the key proof is generated (*iat*) and the *nonce* received from the Authorization Server. The header, instead, contains:

- *'alg'*, which specifies the algorithm used to perform the digital signature over the JWT.
- *'typ'*, determines the key proof type used, in relation to the used protocol (*openid4vci-proof+jwt*.
- *'jwk'*, that includes the key material used to verify the signature. Nested values specify that an Elliptic Curve key is used (*"kty":"EC"*) with the P-256 Elliptic Curve (*"crv":"P-256"*), including also the base64url-encoded value of the x and y coordinates.

The signature and the *nonce* must be validated by the Credential Issuer, ensuring that the client sending the request owns the private key corresponding to the key material attached to the JWT header and confirming that the *access_token* is sent by the same entity that received the corresponding Token Request. Finally, a Credential Response including the Verifiable Credential is issued, according to the requested format, following the ISO 18013-5 standard. The response includes a base64url-encoded *credential*, together with a *c_nonce* that must be provided in subsequent Credential Requests in the same session before it expires and a new one is generated. The Wallet receives the document, notifying with two consecutive interfaces that the process is successful and the overview of the document, as in Figure 4.13.

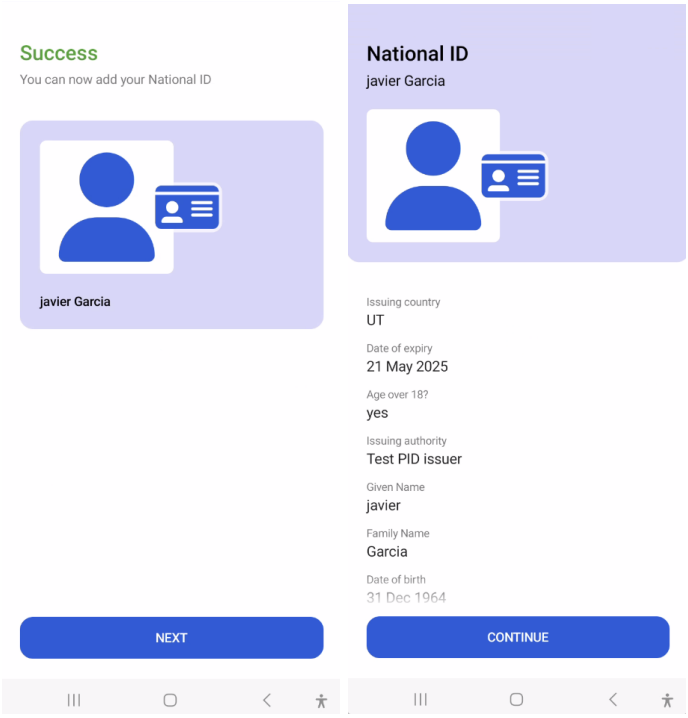

Figure 4.13. These interfaces point out the end of the issuance process.

4.4 Sharing credentials stored within the Wallet

Once a Verifiable Credential has been securely issued and stored in the Wallet, it must also be possible to share it with external Verifiers or Service Providers when required. A Verifiable Presentation must ensure authentication for the client who is presenting the document, and integrity for the data included in the credential. In addition, Unlike physical document verification which typically involves exposing the entire credential, the digital approach allows users to disclose only

the necessary attributes requested by the third party. This feature, known as Selective Disclosure, is crucial in modern digital identity environments as it enhances privacy and data minimization principles.

4.4.1 OpenID for Verifiable Presentation

To accomplish these tasks, OpenID for Verifiable Presentation [45] (OpenID4VP) draft 24 is used to present Verifiable Credentials in a secure manner. OpenID4VP is the complementary protocol of OpenID4VCI, and allows two different flows, depending on the entity that initiates it.

In the Same-Device flow, the user interacts directly with the Verifier using the device where the Wallet application is installed. The Verifier generates an Authorization Request and sends it to the Wallet, containing a *presentation_definition*, which specifies the requested credential and the required attributes to be included. After processing the request, authenticating the user and obtaining their consent, it presents to the Verifier the Authorization Response, including the Verifiable Presentation.

The Cross-Device flow, instead, is initiated by the use of a QR code or deep link presented to the user by the Verifier, containing a Request URI from which the Wallet can obtain the request object which contains the Authorization Request parameters defined by the Verifier. The request object is structured as the previous Authorization Request, including the *presentation_definition*. The flow proceeds as the Same-Device one, gathering the user consent and authenticating them, then sending to the Verifier the Verifiable Presentation.

4.4.2 Testing the presentation of stored Verifiable Credentials

To test how the credentials are shared with services outside of the EUDI Wallet Solution, it is possible to use a Web Verifier provided by the developers of the original EUDI Wallet demo. This test will be performed by navigating to the Web Verifier by using a browser on a different device, thus proceeding with the Cross-Device flow. Through the web page, displayed in Figure 4.14, it is possible to choose between different types of documents and formats, and, in addition, which attributes to request. To test if the Selective Disclosure is correctly provided, after selecting the PID as a *mso_mdoc* formatted document, only some of the attributes included in the document will be requested, in this case the Family Name of the user and whether their age is more than 18.

At the end of the page, the request will be shown:


```
{
  "type": "vp_token",
  "presentation_definition": {
    "id": "1e998093-660c-4ed4-a939-d2982386d32f",
    "input_descriptors": [
      {
        "id": "eu.europa.ec.eudi.pid.1",
        "name": "Person Identification Data (PID)",
        "purpose": "",
        "format": {
          "mso_mdoc": {
            "alg": [
              "ES256",
              "ES384",
              "ES512"
            ]
          }
        },
        "constraints": {
          "fields": [
```

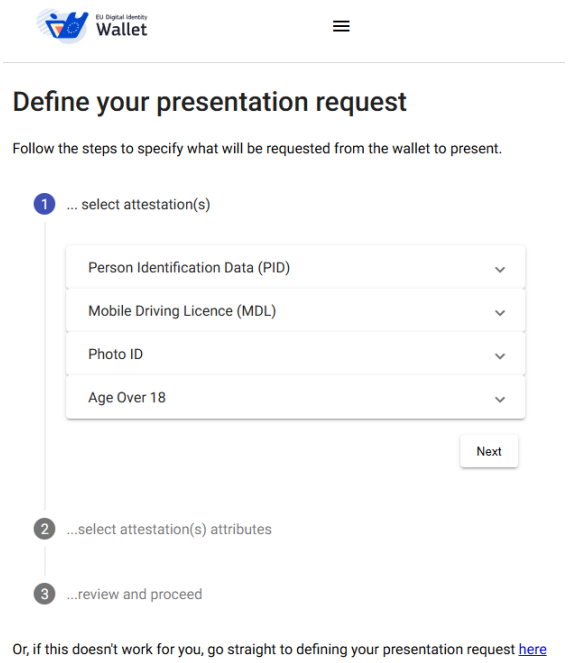

Figure 4.14. Main interface of the Web Verifier.

```
          {
            "path": [
              "$['eu.europa.ec.eudi.pid.1']['family_name']"
            ],
            "intent_to_retain": false
          },
          {
            "path": [
              "$['eu.europa.ec.eudi.pid.1']['age_over_18']"
            ],
            "intent_to_retain": false
          }
        ]
      }
    }
  ]
 },
 "nonce": "a71ae4fe-6ab9-443e-b14a-fe4cdda9a719"
}
```

The structure contains the following parameters:

- *type* points out what the Verifier expects to receive from the Wallet. The *vp_token* is the parameter that will be included in the Authorization Response, containing the Verifiable Presentation.
- The *presentation_definition* defines the requested credential. Inside *input_descriptors*, both the identifier and a human-readable name of the document (*id* and *name*) are specified, along with the chosen format and the cryptographic algorithms supported for the digital signature. Right after, in the *constraints* field, are listed the requested attributes: for each one, the *intent_to_retain* boolean value determines if the Verifier wants to store the attribute after the verification.

- A random *nonce* ensures that the Verifiable Presentation cannot be used to perform replay attacks.

To send the request, a QR code is generated on the web page, ready to be scanned by the Wallet. From the mobile application, this can be done by clicking on the three dots on the top-right side of the main UI and selecting the option "Scan a QR code". Then, as shown on the left screen in Figure 4.15, the Wallet receives the request, displaying the requested attributes and the stored documents that can be shared. Locally, the Wallet requests user consent to allow the transaction, performing authentication through the PIN that was set on the first login. Upon successful authentication, the Authorization response is sent back to the Verifier:

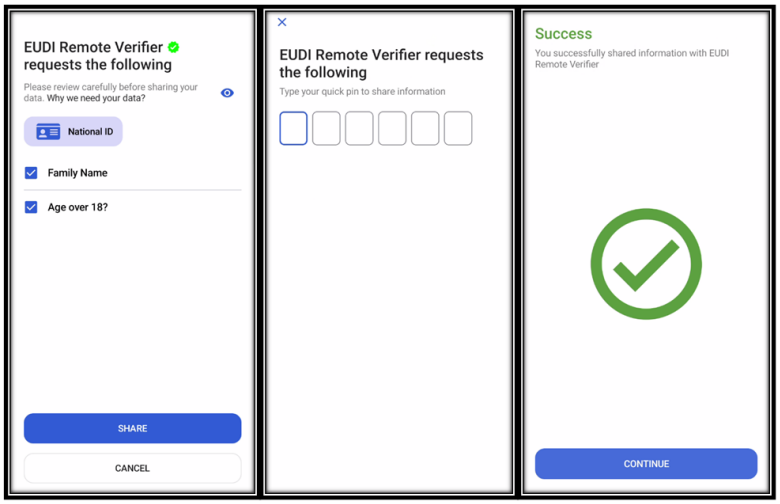

Figure 4.15. The Wallet flow of interfaces during the Verifiable Presentation. Selective Disclosure and user local authentication are shown.

```
{
    "vp_token": [...],
    "presentation_submission": {
        "id": "82e922a6-e696-4c77-b85c-6226df1c3748",
        "definition_id": "1e998093-660c-4ed4-a939-d2982386d32f",
        "descriptor_map": [
            {
                "id": "eu.europa.ec.eudi.pid.1",
                "format": "mso_mdoc",
                "path": "$"
            }
        ]
    }
}
```

While *presentation_submission* reports the response and the request identifiers (respectively with *id* and *definition_id*) along with the credential format, the *vp_token* is a CBOR-encoded structure that contains the Verifiable Presentation. If decoded:

- The *docType* field specifies that the presented credential is a PID , following the format *eu.europa.ec.eudi.pid.1*, with the version set to 1.0.
- The *issuerSigned* field contains two sections: the first, *nameSpaces*, lists the requested attributes as a pair *elementIdentifier* and *elementValue* associated to a *DigestID* identifier and a *random* value. The second, *issuerAuth*, exposes the signature of the Credential

Issuer over the credential itself and the hash of every attribute included in the credential, accompanied by a number to bind the hash to a corresponding *DigestID*.

- The *issuerAuth* section includes the signature of the Credential Issuer over the hashes computed on user attribute values, and additional information of the Credential Issuer from which the credential was issued.
- The *deviceSigned* field contains the Wallet signature over the *vp_token* and also including the nonce that was previously provided by the Verifier in the Authorization Request.

To accept the presentation, the Verifier must prove the validity of the response, and both the authenticity and the integrity of the data. First, it proves the signature of the Credential Issuer, retrieving the public key *deviceKey* stored in *issuerAuth*. Then, to ensure the correctness of the Selective Disclosure implementation, the Verifier must check if the hashes of the requested attributes that are included in the response are valid. For example, this in an extract of a partially decoded *vp_token*:

```
"issuerSigned": {
    "nameSpaces": {
        "eu.europa.ec.eudi.pid.1": [
            {
                "decoded_value": {
                    "random": [...],
                    "digestID": 5,
                    "elementValue": "Garcia",
                    "elementIdentifier": "family_name"
                }
            },
            [...]],
    }
    "valueDigests": {
        "eu.europa.ec.eudi.pid.1": {
        "0": [...],
        "1": [...],
        "2": [...],
        "3": [...],
        "4": [...],
        "5": [...],
        [...]
        }
    "digestAlgorithm": "SHA-256"
},
```

To verify the integrity of the *family_name* claim, *elementValue* is concatenated with the *random*, which acts as salt, then the hash is computed using the *digestAlgorithm* over it. Since the *digestID* of the attribute is "5", the computed value must match the value corresponding to the key "5" under *valueDigests*. Doing this, only the claims that are intentionally shared can be verified. In the end, additional values such as the *validityInfo*, which states the timestamps for both the issuance and the expiration of the document, the presence of *nonce* that was provided within the request and the signature performed with the private key corresponding to the public one stated under the *deviceSigned* field can ensure, respectively, the actual validity of the document, protection against replay attacks and that the credential is being used by its rightful owner, since during the issuance it was bound to that key pair.

Chapter 5

Programmer manual

5.1 Basic configuration of the thesis local environment

5.1.1 Components overview

In order to test and run the setup used in this thesis work, all the main components must be installed on devices that are part of the same network, but not necessarily all on the same device. Each component is listed below:

- The EUDI Wallet demo application, available for download from the EUDI Wallet's GitHub repository. The application must be installed on a suitable device, and its requirements are listed on the same GitHub page. As an alternative, the same application could be run by using a suitable emulator available on Android Studio, which will be needed anyway to build the application after being edited.
- The service provider (*eudi-srv-web-issuing-eudiw-py*) that acts as an intermediary between the wallet and the eIDAS node. The original service, which must be configured properly according to the rest of the setup, is also available within another branch of the EUDI Wallet repository.
- Two instances of the eIDAS node, version 2.8.2, that also includes a demo IdP and a demo SP. As mentioned during the flow analysis, the current setup allows one to request the issuance of a document sending the request from a fictitious country named CA to the Identity Provider of CB, which is the country of origin of the user.

To emulate a common deployment scenario and for compatibility reasons, each installation is performed on a different device. The absence of ownership of a dedicated domain to establish a chain of trust makes it not feasible to obtain a valid certificate from a Certificate Authority. Hence, the following guide will propose the use of self-signed certificates for testing purposes, adding them to the trusted list of the concerned devices. It is discouraged to apply this approach for production environment.

5.1.2 Configuring locally eudi-srv-web-issuing-eudiw-py

The *eudi-srv-web-issuing-eudiw-py* project must be cloned from the respective GitHub repository [46]. In the presented scenario, the service provider is installed on a Windows machine, but a similar set of instructions can be followed for any Linux system, if requirements are met.

Pre-requisites:

- Python version 3.9 [47] or 3.10 [48] , due to their compatibility with the project dependencies.

- An OpenSSL version [49] (3.4.0 used here) to generate a self-signed certificate, as it will be explained below.
- The GitHub project *eudi-srv-web-issuing-eudiw-py* in a local folder. From now on, the path to that folder will be mentioned as *root_folder*. For example, using the command prompt or Git Bash, this can be achieved with the following commands:

```
cd root_folder
git clone
https://github.com/eu-digital-identity-wallet/
eudi-srv-web-issuing-eudiw-py.git
```

  Inside *root_folder*, there is the 'eudi-srv-web-issuing-eudiw-py' folder which contains the project (it should be possible to see folders like 'app', 'flask_session' and 'api_docs').

Flask installation and dependencies setup

Flask is the web application framework required to run the service. Any version that is 2.3 or higher is suitable for this project.

1. Navigate to the 'eudi-srv-web-issuing-eudiw-py' folder:

```
cd root_folder/eudi-srv-web-issuing-eudiw-py
```

2. Create the .venv folder, and activate the virtual environment:

```
python -m venv .venv
. .venv\Scripts\Activate
```

3. Now, Flask can be installed by running:

```
pip install Flask
```

4. And dependencies can be installed:

```
python -m pip install --upgrade pip
pip install -r app/requirements.txt
```

   As of the current *requirements.txt* version, the following packages of the modules will be missing:

   - 'pycose'
   - 'pymdoccbor'

   The issue can be fixed by downloading the missing folders from the respective pyCOSE and pyMDOC-CBOR repositories and placing them in their corresponding folders, inside /.venv/Lib/site-package .

5. Rename the *__config_secrets.py* file stored in eudi-srv-web-issuing-eudiw-py/app/app_config as *config_secrets.py*:

```
cp app/app_config/__config_secrets.py
    app/app_config/config_secrets.py
```

The Flask service is now correctly installed, and running the command

```
flask --app app run --debug
```

The service will be executed in debug mode, on *http://localhost:5000*.

Service configuration

Next steps will allow the service to run on HTTPS, on a chosen address. From now on, this address will be the IP address of the hosting machine of the Service Provider, and will be mentioned as *host_IP_address*.

1. Download the file

   ```
   https://github.com/eu-digital-identity-wallet/
   eudi-srv-web-issuing-eudiw-py/blob/main/api\_docs/
   test\_tokens/IACA-token/PIDIssuerCAUT01.pem.gz
   ```

   and extract its content, named *PIDIssuerCAUT01.pem*, which is a valid IACA certificate for the fictitious country Utopia (UT). Move the *.pem* file into a new folder in root_folder/eudi-srv-web-issuing-eudiw-py/app/certs .

2. Create a new file *localhost.conf*, that will be used to generate the service certificate in a later step. An example could be:

   ```
   [req]
   default_bits = 2048
   distinguished_name = dn
   req_extensions = req_ext
   x509_extensions = v3_req
   prompt = no

   [dn]
   countryName = UT
   stateOrProvinceName = Utopia
   localityName = Localhost
   organizationName = My Organization
   commonName = host_IP_address

   [req_ext]
   subjectAltName = @alt_names

   [v3_req]
   subjectAltName = @alt_names

   [alt_names]
   IP.1 = 127.0.0.1
   IP.2 = host_IP_address
   ```

3. In app/app_config/config_service.py find where *service_url* and *trusted_CAs_path* are defined and edit them in the following way:

   ```
   service_url = os.getenv("SERVICE_URL","http://host_IP_address:5000/")
   trusted_CAs_path =
       "root_folder/eudi-srv-web-issuing-eudiw-py/app/certs"
   ```

4. In app/metadata_config/metadata_config.json find the following parameters, modifying the addresses as shown:

   ```
   "credential_issuer": "https://host_IP_address:5000",
   "credential_endpoint": "https://host_IP_address:5000/credential",
   "batch_credential_endpoint":
       "https://host_IP_address:5000/batch_credential",
   "notification_endpoint": "https://host_IP_address:5000/notification",
   "deferred_credential_endpoint":
       "https://host_IP_address:5000/deferred_credential",
   ```

5. Same procedure should be applied to the file app/metadata_config/openid_configuration.json :

```
"jwks_uri": "https://host_IP_address:5000/static/jwks.json",
...
"issuer": "https://host_IP_address:5000",
"registration_endpoint": "https://host_IP_address:5000/registration",
"introspection_endpoint":
    "https://host_IP_address:5000/introspection",
"authorization_endpoint":
    "https://host_IP_address:5000/authorizationV3",
"token_endpoint": "https://host_IP_address:5000/token",
"userinfo_endpoint": "https://host_IP_address:5000/userinfo",
"end_session_endpoint": "https://host_IP_address:5000/session",
"pushed_authorization_request_endpoint":
    "https://host_IP_address:5000/pushed_authorizationv2",
"credential_endpoint": "https://host_IP_address:5000/credential"
```

   Running the Flask service now, the service should now be hosted, without using HTTPS as a protocol, on *https://host_IP_address:5000* . URLs showing up at that page and referring to other pages of the same application should now point to *host_IP_address:5000/[...]* instead of *localhost:5000/[...]* .

6. Create a new local Certification Authority that will be able to sign the certificate of the local service. To generate a private key *ca.key* and the related certificate *ca.pem*:

```
openssl genrsa -out ca.key 2048

openssl req -x509 -new -nodes -key ca.key -sha256 -days 3650 -out
    ca.pem -subj "/C=IT/ST=Italy/L=Localhost/O=My Local CA/OU=IT
    Department/CN=My Local CA" -config
    "path\to\OpenSSL\bin\openssl.cnf"
```

7. Generate a new key for the service. Use that key, together with the *localhost.conf* file, to make a Certificate Signing Request and have the local CA sign the request by using its own certificate:

```
openssl genrsa -out server.key 2048

openssl req -new -key server.key -out server.csr -config
    localhost.conf

openssl x509 -req -in server.csr -CA ca.pem -CAkey ca.key
    -CAcreateserial -out server.crt -days 730 -sha256 -extensions
    req_ext -extfile localhost.conf
```

8. Move *ca.pem* into the *trusted_CAs_path* previously defined. Move *server.crt* and *server.key* into a new folder root_folder/eudi-srv-web-issuing-eudiw-py/server_cert . To let the browser trust the server certificate, it is important to add manually to the browser's list of trusted certificates the *ca.pem* file. Please note that some browsers do not accept *.pem* format, so a conversion to *.crt* may be needed.

9. From now on, the service can be executed in debug mode on HTTPS, at the chosen IP address on port 5000, using the following commands:

```
set REQUESTS_CA_BUNDLE=root_folder\eudi-srv-web-issuing-eudiw-py\app
\certs\ca.pem

cd root_folder\eudi-srv-web-issuing-eudiw-py
```

```
.venv\Scripts\Activate

flask --app app run
    --cert="root_folder\eudi-srv-web-issuing-eudiw-py\server_cert
\server.crt"
    --key="root_folder\eudi-srv-web-issuing-eudiw-py\server_cert
\server.key" --host=0.0.0.0 --debug
```

To verify that everything was done correctly, open the browser for which the certificate is now considered trusted and navigate to *https://host_IP_address:5000*, where a web page with some URLs should be loaded. Each URL uses HTTPS as protocol.

5.1.3 Installing and configuring the EUDI Wallet

Since the local service provider is now configured, the next step is to set up the environment needed to build and edit the wallet application, in order to make it interact with the service. The procedure will change depending on whether the service certificate was self-signed/signed by the local Certification Authority or if it was created by a certified entity. This guide will consider the first scenario.

Pre-requisites:

- The most recent stable version of Android Studio installed on the device.
- The GitHub project *eudi-app-android-wallet-ui* in a local folder. Using the command prompt in the chosen directory:

  ```
  git clone https://github.com/eu-digital-identity-wallet
  /eudi-app-android-wallet-ui.git
  ```

- An Android device, either physical or emulated, capable of running the wallet application (API level 26+). Android Studio provides different emulated devices for testing needs. For the thesis work, a physical device was preferred.

Building the application to interact with the service provider

In Android Studio, open the project directory *eudi-app-android-wallet-ui*. Android Studio will build the application automatically. Once the process ends, switch the current 'Build Variant' to *devDebug* from the menu 'Build' → 'Select Build'. The building process will start again, since the default Build Variant was set to *demoDebug*, a limited version of the application that misses some important features.

1. In the 'Project' menu, change the view to 'Project Files'. Open eudi-app-android-wallet-ui/network-logic/src/main/res/xml/network_security_config.xml and modify its content as follows:

   ```
   <network-security-config>
       <base-config cleartextTrafficPermitted="true">
           <trust-anchors>
               <certificates src="system" />
               <certificates src="user" />
           </trust-anchors>
       </base-config>
   </network-security-config>
   ```

   In this way, if the service certificate is signed by a trusted entity, communication with the application will not be blocked. Skipping this step will cause the Wallet to generate a generic error when the user tries to open a connection with the local Service Provider.

2. Open eudi-app-android-wallet-ui/core-logic/src/dev/java/eu/europa/ec /corelogic/config/-ConfigWalletCoreImpl.kt and modify the *VCI_ISSUER_URL* value to the URL on which the service provider is reachable, i.e. *https://host_IP_address:5000*.

3. Open the *build.gradle.kts* file of the *core_logic* module, and add the following dependencies:

```
implementation(libs.ktor.android)
implementation(libs.ktor.logging)
```

4. Create a new *ProvideKtorHttpClient.kt* file inside the folder src/main/java/eu/europa/ec/-corelogic/config , with the following code:

```
import android.annotation.SuppressLint
import io.ktor.client.HttpClient
import io.ktor.client.engine.android.Android
import io.ktor.client.plugins.logging.Logging
import java.security.SecureRandom
import javax.net.ssl.HostnameVerifier
import javax.net.ssl.SSLContext
import javax.net.ssl.TrustManager
import javax.net.ssl.X509TrustManager
import javax.security.cert.CertificateException

object ProvideKtorHttpClient {

    @SuppressLint("TrustAllX509TrustManager",
        "CustomX509TrustManager")
    fun client(): HttpClient {
        val trustAllCerts = arrayOf<TrustManager>(
            object : X509TrustManager {
                @Throws(CertificateException::class)
                override fun checkClientTrusted(
                    chain: Array<java.security.cert.X509Certificate>,
                    authType: String
                ) {
                }

                @Throws(CertificateException::class)
                override fun checkServerTrusted(
                    chain: Array<java.security.cert.X509Certificate>,
                    authType: String
                ) {
                }

                override fun getAcceptedIssuers():
                    Array<java.security.cert.X509Certificate> {
                    return arrayOf()
                }
            }
        )

        return HttpClient(Android) {
            install(Logging)
            engine {
                requestConfig
                sslManager = { httpsURLConnection ->
                    httpsURLConnection.sslSocketFactory =
                        SSLContext.getInstance("TLS").apply {
                        init(null, trustAllCerts, SecureRandom())
```

```
                    }.socketFactory
                    httpsURLConnection.hostnameVerifier =
                        HostnameVerifier { _, _ -> true }
                }
            }
        }
    }

}
```

5. Modify the *provideEudiWallet* function declared in the src/main/java/eu/europa/ec/corelogic/config/id/LogicCoreModule.kt file, as it follows:

```
@Single
fun provideEudiWallet(
context: Context,
walletCoreConfig: WalletCoreConfig,
walletCoreLogController: WalletCoreLogController
): EudiWallet = EudiWallet(context, walletCoreConfig.config) {
    withLogger(walletCoreLogController)
    // Custom HttpClient
    withKtorHttpClientFactory {
        ProvideKtorHttpClient.client()
    }
}
```

6. As a last step before installing the application on the connected mobile device or emulator, add to that device trusted list of certificates the one of the Certification Authority that signed the local service provider configured in the last section. Import the certificate on the device (using the Android Studio emulator it is possible to drag and drop it on the screen). Open the device's Settings app → Security & privacy → More security settings → Install a certificate. The certificate should appear under the 'User certificates' section. Now, launching the option Run from the Android Studio interface, the installation process will start on the mobile device.

If everything is done correctly, after opening the wallet application (the icon should have a red mention 'DEV') and setting a new access PIN, tapping on 'Add a new document' will redirect the user to a web page, asking to choose a Credential Provider (just one option should be available).

5.1.4 The eIDAS node setup

In the thesis work, the eIDAS node was installed and configured on an Ubuntu 20 virtual machine hosted on the device on which the local service provider is running. This made it possible to have different machines interact with each other during the flow, to recreate a more realistic scenario since eIDAS nodes used in production are those owned and developed by Member States, independent from the wallet application and the service provider. Because of this setup, it is crucial to have the virtual machine and the host on the same network. To run the eIDAS node, an application server is needed. The node supports:

- Apache Tomcat
- WildFly
- WebLogic
- WebSphere Liberty Profile

For each application server, a different country will be configured, because each service uses a different port (8080 for Tomcat). For example, running the application through Tomcat, the fictitious Country A ('CA') node will be used, meaning that both specific Connector and specific Proxy-Service components will be part of that country infrastructure. In a real scenario, node components implemented by different Member States need to interact with each other, so configuring two or more application servers or having two instances of the node on different machines would be ideal.

Pre-requisites:

- Oracle VM VirtualBox 7.0 and a virtual machine with Ubuntu 20 or Ubuntu 21 installed. From the virtual machine Settings in the Oracle VirtualBox Manager, in the Network tab, select 'Bridged Adapter' and the adapter used by the hosting machine. The following prerequisites are to be referred to the virtual machine system, not to the host machine.
- Java SDK 11 installed. In the thesis work, the folder *jdk-11.0.1* (the one that stores Java files and folders, such as 'bin', 'conf', 'lib') is located in /opt , which will be the path where all the project related applications and files are stored.
- At least 4 GB of memory, 8 GB of disk space and 2 CPU cores

Installing Tomcat

Tomcat [50] is the chosen application server for the thesis work. The following steps assume that users already have an Apache Tomcat version downloaded on their device. The thesis work uses Apache Tomcat 9.0.85, which compressed version *apache-tomcat-9.0.85.tar.gz* can be found in the Apache Tomcat archive.

1. Extract the content of *apache-tomcat-9.0.85.tar.gz* and move it to the folder that will be used for Tomcat and the eIDAS node configuration:

   ```
   sudo mkdir -p /opt/tomcat && sudo tar -xvzf
       apache-tomcat-9.0.85.tar.gz -C /opt/tomcat
   ```

2. Set a new environment variable *CATALINA_HOME*, that is used to locate the *apache-tomcat-9.0.85* folder:

   ```
   nano ~/.bashrc
   ```

   Add at the end of the file:

   ```
   export CATALINA_HOME=/opt/tomcat/apache-tomcat-9.0.85
   ```

   And, after saving the file, run the following command to apply the new variable:

   ```
   source ~/.bashrc
   ```

3. Create a script *setenv.sh* inside $CATALINA_HOME/bin containing the command stated below. This will be executed at Tomcat startup to run additional commands (some of them will be added when configuring the node), without editing the main startup file. In this case, the variable *JAVA_HOME* is set only when Tomcat is starting, to avoid conflict with other Java versions in the system.

   ```
   export JAVA_HOME=/opt/jdk-11.0.1
   cmd
   ```

4. Verify that Tomcat is now successfully installed, running, from the command line inside the *$CATALINA_HOME* folder:

   ```
   bin/catalina.sh run
   ```

   Which will start the application server, opening a log console. Wait for it to complete the deployment (a mention in the log similar to 'Server startup in [XXX] seconds'). After that, open a browser and navigate to *http://localhost:8080* , where the Apache Tomcat homepage should load.

Installing Bouncy Castle

Bouncy Castle [51] is the security provider that allows the node to perform cryptographic operations such as signing, encrypting and decrypting. Because of that, it is mandatory to have it (or any equivalent security provider) installed and configured properly to avoid any conflict with Ignite [52], which runs on top of Tomcat. The version employed in the tested environment is 1.80, downloadable as the *bcprov-jdk18on-1.80.jar* file from the Bouncy Castle official website.

1. At /opt path, create a new *Bouncy Castle* folder and move the *bcprov-jdk18on-1.80.jar* file into it.

2. Open the *java.security* file stored at */opt/jdk-11.0.1/conf/security* and search for the section in which security providers are listed. It should look like the following:

```
# List of providers and their preference orders (see above):
#
security.provider.1=SUN
security.provider.2=SunRsaSign
security.provider.3=SunEC
security.provider.4=SunJSSE
[...]
```

   Right after the last element, add:

```
security.provider.N=org.bouncycastle.jce.provider.
BouncyCastleProvider
```

   Substituting the character *N* after *security.provider.* with the next number of the sequence in the providers list.

3. Edit the *setenv.sh* file defined in the previous section:

```
export JAVA_HOME=/opt/jdk-11.0.1

export JAVA_OPTS="--module-path
    /opt/BouncyCastle/bcprov-jdk18on-1.80.jar --add-modules
    org.bouncycastle.provider --add-opens
    org.bouncycastle.provider/org.bouncycastle.jcajce.provider.
asymmetric.x509=ALL-UNNAMED"
cmd
```

Deploying the basic eIDAS node configuration

1. Navigate to the eIDAS node version 2.8.2 download page and download the *eIDAS-node-2.8.2.zip* file. Extract only *EIDAS-Binaries-Tomcat-2.8.2.zip* and from the latter extract the *TOMCAT* folder, which contains *config.zip* and a list of *.war* files, one for each node component (SP, ProxyService, Connector, SpecificProxyService, SpecificConnector, IdP).

2. Move the content of *config.zip* in a new folder named *eidas-config* located at /opt . Now the following paths should exist:

```
/opt/eidas-config/tomcat/connector/eidas.xml
/opt/eidas-config/tomcat/idp/idp.properties
/opt/eidas-config/tomcat/specificConnector/specificConnector.xml
```

   The *eidas-config* folder now contains all the configuration files for each node component.

3. Set a new environment variable for each component of the node, pointing to the corresponding folder in /opt/eidas-config/tomcat , right under the definition of *CATALINA_HOME*:

```
export EIDAS_CONNECTOR_CONFIG_REPOSITORY=/opt/eidas-config/
tomcat/connector

export EIDAS_PROXY_CONFIG_REPOSITORY=/opt/eidas-config/tomcat/proxy

export SPECIFIC_CONNECTOR_CONFIG_REPOSITORY=
/opt/eidas-config/tomcat/specificConnector

export SPECIFIC_PROXY_SERVICE_CONFIG_REPOSITORY=
/opt/eidas-config/tomcat/specificProxyService

export SP_CONFIG_REPOSITORY=/opt/eidas-config/tomcat/sp

export IDP_CONFIG_REPOSITORY=/opt/eidas-config/tomcat/idp
```

4. Move every *.war* file extracted previously into /opt/tomcat/apache-tomcat-9.0.85/webapps so that, at the next Tomcat boot, each module will be deployed.
5. Launch again Tomcat from its folder, with:

   ```
   bin/catalina.sh run
   ```

   And wait for each module to be deployed. After, navigate to *http://localhost:8080/SP* and verify that the page loads. To check if every component is successfully loaded, simulate a request from the web page:

   (a) Select for both 'SP Country' and 'Citizen Country' the country CA, then click 'Submit' at the bottom of the page
   (b) A JSON associated to the request just sent will be shown. Click 'Submit' again.
   (c) Request is forwarded through the node components, until it reaches the specific Proxy Service. Click 'Next' to be redirected to the demo Identity Provider.
   (d) Complete the login procedure using the test credentials: 'xavi' as username and 'creus' as password. After submitting, a summary of the user attributes will be shown as an SMSSPResponse.
   (e) Traffic will be redirected to the Service Provider component again, where it is possible to display received data, finishing the demo flow.

Creating the second eIDAS node infrastructure

At this point, shut down the virtual machine and, from the Oracle VM VirtualBox Manager interface, select the virtual machine. From the top-left options, click on 'Machine' → 'Clone...' to open a new tab. Adjust name, path and additional options as preferred, but choose 'Generate new MAC addresses for all network adapters' for the MAC Address Policy option. This will create a second machine that can be used to configure a valid node for the second country involved in the process, CB. After booting the second machine, make sure the IP address is different from the first one, or change it from the 'Settings' menu within the Ubuntu system. From now on, the guide will refer either to the first or the second VM by using the country for which the node is configured (CA for the former, CB for the latter).

5.2 Making the components interact with each other

As of the current setup, possible interactions between the services are defined with green arrows in the Figure 5.1. While the Wallet can communicate with the local Service Provider due to the acceptance of its public certificate by the user's device system, the last cannot communicate with the node yet. To achieve this, a new country, named Utopia, needs to be added to the list of available countries in the local Service Provider, redirecting the user to the demo Service Provider hosted on the virtual machine together with the eIDAS node.

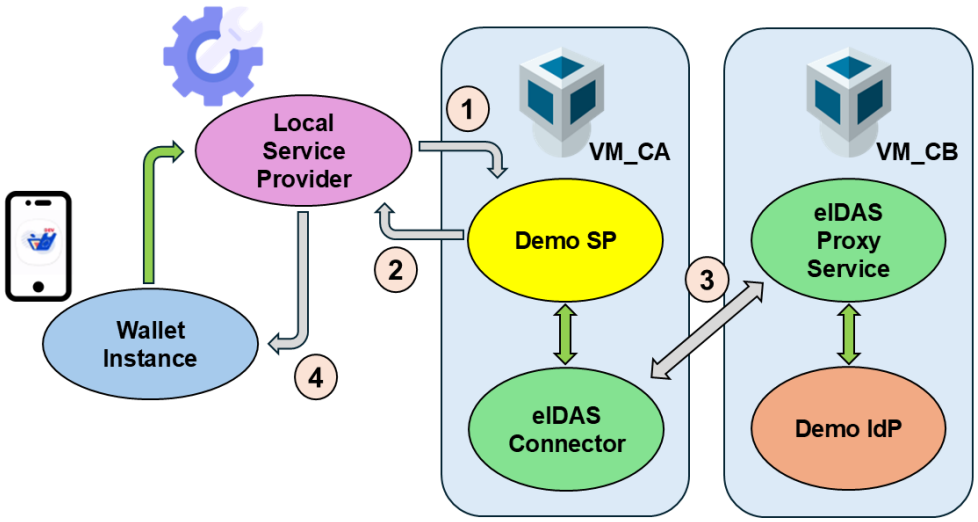

Figure 5.1. Temporary flow. Grey arrows highlight flows that are implemented in this section.

5.2.1 Interactions between the local Service Provider and the eIDAS node

In the eudi-srv-web-issuing-eudiw-py/app/app_config folder, open *config_countries.py*. Inside the *ConfCountries* class there is the list of supported countries. Append at the end of the list the following configuration for the fictitious Utopia country:

```
"UT": {
    "name": "Utopia",
    "pid_url": "http://eIDAS_node_CA_IP:8080/SP",
    "pid_mdoc_privkey": "root_folder/eudi-srv-web-issuing-eudiw-py/app/
    private/ds_token/PID-DS-0002.pid-ds-0002.key.pem",
    "pid_mdoc_privkey_passwd": b"pid-ds-0002",
    "pid_mdoc_cert": "root_folder/eudi-srv-web-issuing-eudiw-py/app/
    private/ds_token/PID-DS-0002.cert.der",
    "supported_credentials": [
        "eu.europa.ec.eudi.pid_mdoc"
    ],
    "connection_type": "openid",
    "oidc_auth": {
        "base_url": "http://eIDAS_node_host_IP:8080/SP",
        "redirect_uri": "http://eIDAS_node_host_IP:8080/SP",
        "scope": "openid",
        "state": "stateRandomValue",
        "response_type": "code",
        "client_id": "utopia_client_id",
    },
    "attribute_request": {
        "url": "https://local_SP_host_IP:5000",
        "headers": "",
        "custom_modifiers": "",
    },
    "dynamic_R2": cfgserv.service_url + "form_R2",
},
```

This configuration will allow, when the user requests a PID from the Wallet through the Utopia selection, to be redirected to *http://eIDAS_node_CA_IP:8080/SP*, keeping the OpenID parameters that were been negotiated before with the Wallet. In addition, this sets also the *dynamic_R2* parameter, which will be used to return to the dynamic form in the later steps of the flow. Open the app/route_dynamic.py file, and search for the definition of the *dynamic_R1* function. The goal of this function is to redirect the user after selecting the credential issuer country (Figure 4.7). Add a new basic configuration, specific for the Utopia country:

```
[...]
elif cfgcountries.supported_countries[country]["connection_type"] == "openid":

    if country == "UT":
        redirect_url = cfgcountries.supported_countries[country]
        ["oidc_auth"]["redirect_uri"]
        return redirect(redirect_url)
[...]
```

Now, the eIDAS node must be configured to:

- Have its components interact with each other by using the virtual machine IP address, rather than localhost, since the service must be provided to an external device
- Redirect the user back to the local Service Provider after completing the eIDAS flow, preserving user data collected at the Identity Provider.

In this subsection, the *eIDAS_node_host_IP* will refer to the IP address of the virtual machine on which the file that is going to be edited is stored, while setups that are specific for only one of the VMs (VM_CA or VM_CB) will use either *eIDAS_node_CA_IP* or *eIDAS_node_CB_IP* .

Open both virtual machines and navigate to the opt/eidas-config/tomcat folder, which contains the configuration files of node components.

- Open connector/eidas.xml and change from *localhost* to *eIDAS_node_host_IP* the value of the following parameters:
  - *connector.assertion.url*
  - *connector.metadata.url*
  - *service1.metadata.url* for VM_CA, *service2.metadata.url* for VM_CB
  - *security.header.CSP.report.uri*
  - *specific.connector.response.receiver*
- Open connector/metadata/MetadataFetcher_Connector.properties and append at the end of *metadata.location.whitelist*:

  ```
  http://eIDAS_node_CA_IP:8080/EidasNodeProxy/ServiceMetadata;
  http://eIDAS_node_CB_IP:8080/EidasNodeProxy/ServiceMetadata;
  ```

  To allow the Connector to retrieve Proxy Service metadata during the communication.
- Open proxy/eidas.xml and change from localhost to *eIDAS_node_host_IP* the value of the following parameters:
  - *service.metadata.url*
  - *ssos.serviceMetadataGeneratorIDP.redirect.location*
  - *ssos.serviceMetadataGeneratorIDP.post.location*
  - *security.header.CSP.report.uri*
  - *specific.proxyservice.request.receiver*

  In VM_CB, set the value of *service.countrycode* and *metadata.node.country* to CB.

- Open proxy/metadata/MetadataFetcher_Service.properties and append at the end of *metadata.location.whitelist*:

  ```
  http://eIDAS_node_CA_IP:8080/EidasNodeConnector/ConnectorMetadata;
  http://eIDAS_node_CB_IP:8080/EidasNodeConnector/ConnectorMetadata;
  ```

- Open specificConnector/specificConnector.xml and change the value of the following parameter from localhost to *eIDAS_node_host_IP*:

  - *specific.connector.request.url*

  In addition, it is possible to change the value of *issuer.name* in order to change which issuer will be listed inside the SmsspResponse.

- Open specificProxyService/specificProxyService.xml and change the value of the following parameter from *localhost* to *eIDAS_node_host_IP*:

  - *specific.proxyservice.idp.response.service.url*
  - *specific.proxyservice.response.url*

- Open sp/sp.properties and change the following parameters (note that the last URL refers to the local Service Provider):

  ```
  country1.url=http://eIDAS_node_CA_IP:8080/
  EidasNodeConnector/ServiceProvider
  country2.url=http://eIDAS_node_CB_IP:8080/
  EidasNodeConnector/ServiceProvider
  sp.return=https://host_IP_address:5000/dynamic/mynode-response
  ```

  In addition, within the file on VM_CB change also:

  ```
  provider.name=DEMO-SP-CB
  requester.id=http://eidas.eu/EidasNode/RequesterId_CB
  ```

  These changes will allow to display the right URL values in the fields shown at Figure 4.8, selecting the component to which the issuance request will be sent and the return URL to redirect the user back to the local Service Provider with the user data obtained from the Identity Provider.

These steps enables the flow corresponding to grey arrows 1, 2 and 3 in Figure 5.1. This can be confirmed by starting a request from the Wallet application: the issuance should continue uninterrupted through the local Service Provider until reaching the node SP, where the user can select CA as SP country and CB as citizen country and send the request to the CB Identity Provider. The flow will stop when the response sent back from the IdP through the eIDAS node components tries to reach *https://host_IP_address:5000/dynamic/mynode-response*, since it is not implemented yet.

5.2.2 Interactions between the local Service Provider and the Wallet

The local Service Provider now tries to receive the SMSSPResponse as a payload, through a POST request to *dynamic/mynode-response*. This route can only be reached by the demo SP Provider that is called for the Utopia country, hence it will be customized to rework the response of that specific flow.

Open the 'eudi-srv-web-issuing-eudiw-py/app/route_dynamic.py' file. Here are defined all the routes that use the prefix /dynamic, including also the page shown in Figure 4.7 listing the countries. The new route */mynode-response* must be able to:

1. Extract the SMSSPResponse parameter from the POST Request and decode it, since it is base64 encoded, obtaining a JSON string.

2. Convert the JSON string into a Python dictionary
3. Map attribute values in a format that is compatible with the form template
4. Render the form template

The presented route will achieve these goals:

```
@dynamic.route("/mynode-response", methods=["POST"])
def mynode_response():

    try:
        smssp_response_base64 = request.form.get("SMSSPResponse")

        if not smssp_response_base64:
            return jsonify({"error": "Missing SMSSPResponse parameter"}), 400

        decoded_response =
            base64.b64decode(smssp_response_base64).decode("utf-8")

        response_json = json.loads(decoded_response)

        form_data = {
            "family_name": response_json["response"]["attribute_list"]
            [0]["values"][0]["value"],
            "given_name": response_json["response"]["attribute_list"]
            [1]["values"][0]["value"],
            "birth_date": response_json["response"]["attribute_list"]
            [2]["value"],
            # "pid": response_json["response"]["attribute_list"]
            [3]["values"][0]["value"],
            "estimated_issuance_date": datetime.now().strftime("%Y-%m-%d"),
            "estimated_expiry_date": (datetime.now() +
                timedelta(days=90)).strftime("%Y-%m-%d"),
            "issuing_country": "UT",
            "issuing_authority": "Utopia ID Authority",
            "age_over_18": True if int(response_json["response"]
            ["attribute_list"][2]["value"][:4]) < datetime.now().year - 18
                else False
        }

        user_id = "UT" + "." + generate_unique_id() + "#" + str(form_data)

        return render_template("dynamic/form_authorize.html",
            presentation_data=form_data, user_id=user_id,
            redirect_url=cfgserv.service_url + "dynamic/redirect_wallet")

    except Exception as e:
        return jsonify({"error": str(e)}), 500
```

After this, modify the eudi-srv-web-issuing-eudiw-py/app/templates/dynamic/form_authorize.html to reflect how the plain dictionary is generated:

```
[...]
    <form id="selectCountryForm" method="post" action="{{redirect_url}}"
        accept-charset="UTF-8"
        enctype="application/x-www-form-urlencoded;charset=UTF-8"
        autocapitalize="off" spellcheck="false">
        <h3>Provider Form Authentication</h3>
```

```
        <p>Please confirm if your data is right</p>
        <div id="eidasCountries">
            {% for name, value in presentation_data.items() %}
                <div class="form-group">
                    <div class="col-md-12 col-xs-12">
                        <label class="control-label">{{ name }}</label>

                        {% if name == "portrait" %}
                            <img style="width: 50%;" class="img-thumbnail"
                                src="{{ value }}">
                        {% elif name == "driving_privileges" and value is
                            iterable and value is not string %}
                            <h5>{{ name }}</h5>
                            {% for privilege in value %}
                                <div class="col-md-12 col-xs-12"
                                    style="background-color: rgb(213, 213, 213);
                                    margin-bottom: 5%; outline: auto;">
                                    {% for sub_name, sub_value in
                                        privilege.items() %}
                                        <label class="control-label">{{ sub_name
                                            }}</label>
                                        <input type="text" class="form-control"
                                            placeholder="{{ sub_value }}"
                                            name="{{ sub_name }}" value="{{
                                            sub_value }}" readonly>
                                    {% endfor %}
                                </div>
                            {% endfor %}
                        {% else %}
                            <input type="text" class="form-control"
                                placeholder="{{ value }}" name="{{ name }}"
                                value="{{ value }}" readonly>
                        {% endif %}
                    </div>
                </div>
            {% endfor %}

            <div class="col-md-12 col-xs-12" id="hidden_elems">
                <div class="col" >
                    <input type="hidden" value="{{ user_id }}" name="user_id">
                </div>
            </div>
        </div>
        <div class="clearfix"></div>

        <span class="more-submit">
            <input type="submit" name="proceed" accesskey="S"
                value="Authorize" class="btn btn-primary" title="Submit"
                style="margin-right: 5%;"/><a href="/auth_choice" class="btn
                btn-back">Cancel</a>
        </span>
    </form>
[...]
```

In the end, modify the route */dynamic_R2*, which is accessed to generate the credential response in CBOR format. It must be capable of processing the string created by concatenating the issuer identifier, the unique user identifier and the dictionary containing user information from the */mynode-response* route.

```
@dynamic.route("/dynamic_R2", methods=["GET", "POST"])
def dynamic_R2():
    json_request = request.json

    (v, l) = validate_mandatory_args(json_request, ["user_id",
        "credential_requests"])

    if not v:
        print("ERROR: Missing fields in JSON request")
        return {
            "error": "invalid_credential_request",
            "error_description": "missing fields in json",
        }

    user = json_request.get("user_id", "")

    try:
        country, rest = user.split(".", 1)
    except ValueError:
        print(f"ERROR: user not valid - {user}")
        return jsonify({"error": "Invalid user format"}), 400

    user_id_true, sep, ut_form = rest.partition("#")

    if ut_form:
        try:
            ut_form = ut_form.replace("'", '"').replace("True",
                "true").replace("False", "false")
            ut_form = json.loads(ut_form)
        except json.JSONDecodeError as e:
            print(f"ERROR parsing ut_form: {ut_form} - {e}")
            ut_form = None
    else:
        ut_form = None

    credential_request = json_request["credential_requests"]

    session["country"] = country
    session["version"] = cfgserv.current_version
    session["route"] = "/dynamic/form_R2"

    data = dynamic_R2_data_collect(
        country=country, user_id=user_id_true, ut_form=ut_form
    )

    if "error" in data:
        return data

    credential_response = credentialCreation(
        credential_request=credential_request, data=data, country=country
    )
    return credential_response
```

Chapter 6

Results and Observations

6.1 Security Analysis: Threats and Mitigations

Throughout this work, various interactions involving the issuance and presentation of credentials using a local setup that replicates the EUDI Wallet have been carefully analysed. One of the key goals of this analysis was to verify whether the implemented flows effectively embody the security and privacy principles defined by both eIDAS 2.0 and the SSI model. In this section, common attacks are reported, highlighting the countermeasures that provide protection against them.

6.1.1 Man-In-The-Middle Attacks

A Man-In-The-Middle (MITM) attack occurs when an adversary intercepts and possibly modifies the messages exchanged between two parties without their knowledge. Given the multiple interactions within the EUDI Wallet ecosystem (between Wallet, Service Provider, eIDAS nodes and Identity Providers), the attacker could alter messages between any of the entities, compromising the communication.

The implemented setup provides effective protection against MITM attacks by relying extensively on secure communication protocols. Specifically, all interactions involving sensitive data exchanges (such as between the Wallet and the Authorization Server, or between the eIDAS nodes themselves) are strictly carried over TLS 1.3, ensuring end-to-end encryption, message integrity and endpoint authentication. With TLS, each entity involved in the communication proves its identity, preventing the attacker from impersonating any component within the architecture.

In addition, security measures against MITM attacks are enhanced through digital signatures: credentials, tokens and authorization responses are digitally signed by their respective issuers, and these signatures can be independently verified by the receiving entities, making the undetected manipulation unfeasible.

6.1.2 Replay Attacks

Replay attacks occur when an attacker intercepts legitimate communication between entities and subsequently retransmits it to get unauthorized access to the resources requested by the victim. In digital identity systems, an attacker could potentially reuse intercepted tokens, authorization codes or Verifiable Presentations, bypassing the authentication mechanisms.

The local implementation provides strong protection against replay attacks through various embedded security mechanisms. One primary countermeasure is the use of unique, single-use and short-lived authorization codes issued by the Authorization Server: once used, these codes are immediately invalidated, thus preventing subsequent reuse. Additionally, the PKCE mechanism ensures the authorization code can only be redeemed by the original requesting Wallet, since the

code_verifier is securely stored on client-side: if the wrong code is sent to the Authorization Server, the computation will not match the *code_challenge*, resulting in the invalidation of the procedure.

Within the credential presentation flow, the OpenID4VP protocol enforces additional protective measures. The Wallet creates a cryptographic binding between the Verifiable Presentation and two parameters included in the original Authorization Request: the wallet identifier (*client_id*) and a freshly generated *nonce* provided by the Verifier itself. The *nonce* is a random value, unique for each session, allowing the Verifier to verify that the Verifiable Presentation corresponds exclusively to that specific authorization request. Upon receiving the response, the Verifier checks that the presented credentials are correctly linked to the *client_id* and *nonce* values originally transmitted. If multiple Verifiable Presentations are present within a single response, every *nonce* value must match consistently; otherwise, the response is rejected.

6.1.3 Credential Theft and Unauthorized Usage

Credential theft involves unauthorized access or misuse of digital credentials or cryptographic material. Once stolen, credentials or associated cryptographic keys might allow an attacker to impersonate a legitimate holder, accessing sensitive resources and services without proper authorization.

The EUDI Wallet significantly reduces this threat through several layers of protection. First, credentials are securely stored within a trusted and encrypted storage on the user's device, and are strictly bound to the device. Access to these credentials requires local authentication, such as the 6-digit PIN set by the user or biometric authentication, ensuring that only the legitimate holder can initiate a presentation procedure or a credential request. Moreover, all issued credentials are cryptographically bound to the holder's private key which is stored securely in the Wallet. Any attempt to use the credentials externally, without possessing the corresponding private key, would fail, as the cryptographic proof required for credential presentation would be impossible to generate.

6.1.4 Cross-Site Request Forgery

In a Cross-Site Request Forgery (CSRF) attack, an attacker tricks the authenticated user into submitting requests without their knowledge or consent, leading to unauthorized actions being performed on the behalf of the user.

Within the described implementation, CSRF protection is explicitly provided through the use of the *state* parameter included within the authorization request. The Wallet generates a random *state* value during the request initiation, which is subsequently returned within the response by the Authorization Server. The wallet verifies that the returned *state* matches the original value issued, confirming that the response corresponds uniquely to a legitimate request initiated by the holder. Any mismatch immediately results in the rejection of the response.

6.1.5 Malicious Credential Issuer or Verifier

Compromised or malicious Credential Issuers and Verifiers can pose a serious threat, as these entities might attempt to issue fraudulent credentials, modify claims or collect unauthorized user data. Malicious Verifiers might also try to misuse presented credentials beyond their intended purpose or attempt to retain unauthorized user data, thus violating data minimization principles.

In the local environment, credentials are digitally signed by the Credential Issuer using a private key whose public counterpart is known and verifiable, and shared in the issuer metadata. Any credential manipulation by a malicious issuer would invalidate the digital signature, allowing immediate detection by both the holder and Verifiers. Furthermore, Selective Disclosure ensures that users provide only the minimal set of data required by Verifiers, preventing unnecessary exposure of personal attributes. The explicit consent step required within the Wallet, along with the possibility of choosing which document to use and having a detailed visualization of attributes, supports users in the process of recognizing and declining interactions with suspicious entities.

6.2 Relationship with eIDAS 2.0 goals

Outside of security features, which are an important requirement in the eIDAS 2.0 Regulation, it is necessary to evaluate also how other principles are covered by the current EUDI Wallet implementation.

By replicating locally the interactions between different countries via the implemented eIDAS infrastructure and integrating it with the flow for credential issuance and verification, the result is a working environment that simulates, with its own limitations, a real-world interoperability scenario. Moreover, the use of standardized credential formats and secure communication protocols such as SAML and OpenID4VCI/OpenID4VP demonstrate how a credential issued by a specific country can be recognized and validated by another.

From the perspective of the Self-Sovereign identity, the EUDI Wallet satisfies its main pillars: users always have control over their credentials; Verifiable Credentials are generated on-the-fly and stored only on user devices, in a secure storage along with cryptographic material. During presentation, holders are aware of which entity is requesting their credentials, as well as which attributes are required or optional, and explicit consent is mandatory to proceed with the transaction. In addition, thanks to intuitive and effective interfaces, the application seamlessly guides the user through the issuance and presentation processes. However, it should be noted that, to avoid confusion about the issuance flows that involve multiple redirections, some usability improvements could be implemented, for example making clearer with which entity the user is interacting at any time during the flow.

Lastly, the test environment demonstrates that the EUDI Wallet prototype can be used as a base application on which Member States can build their own Wallet Solution without worrying about interoperability issues. The application modularity allows to add more features, such as support for other document types, stronger authentication procedures or the integration of trust services, making it ideal to overcome the scalability problem of previous solutions.

The results obtained through the local setup confirm not only the technical feasibility of the credential issuance and presentation flows, but also their alignment with the broader goals outlined by the European Commission in the Digital Decade policy programme [2]: according to this strategy, by 2030 the totality of EU citizens should have access to a digital identity solution that is universally recognized and trusted across Member States, finally overcoming the fragmentation that afflicted previous digital identity systems.

Chapter 7

Conclusion

In this thesis, the main objective was the practical analysis and evaluation of the new European Digital Identity (EUDI) Wallet within a local experimental setup. The purpose was twofold: to highlight the enhanced security mechanisms provided by digital wallets, in particular the EUDI Wallet, and to demonstrate the potential interoperability and ease of integration for Member States adopting the eIDAS 2.0 framework.

The analysis carried out on the credential issuance and presentation flows highlighted how security and user protection are the core design principles of the EUDI Wallet ecosystem: mechanisms such as selective disclosure, cryptographic verification of credentials, protection against replay attacks through nonces, and decentralized management of identity data substantially enhance user privacy and data security, mitigating some of the risks typical of traditional identity systems. At the same time, the experiment acts as a practical demonstration of how easily Member States can integrate the solution into their existing digital identity infrastructure. Within the thesis, a realistic scenario was created, simulating the issuance of Verifiable Credentials with two fictitious Member States. Although carried out in a controlled local environment, this scenario effectively demonstrated the inherent flexibility and interoperability of the EUDI Wallet architecture, suggesting that the EUDI Wallet framework can promote easier integration and cooperation among different Member States, significantly reducing complexity and costs compared to traditional centralized or federated approaches.

Nevertheless, several limitations emerged during the research that should be explicitly acknowledged. The work relied on adapting pre-existing services rather than creating them from scratch, limiting the possibilities of modification and preventing some architectural optimizations that could have been introduced on the local Service Provider. Also, the local demonstration leveraged simplified demonstration nodes, including a demo Identity Provider and a demo Service Provider: while valuable for proving the feasibility of interactions, they might differ from real-world services for complexity and security. Lastly, the credentials supported in the local flow were limited to the issuance and presentation of the Person Identification Data document. Extending the setup to accommodate additional document types, such as diplomas, mobile driving licenses or other QEAAs would have required extensive reconfigurations, particularly involving the Identity Provider and Credential Issuer components. Future developments of this work could try to realize a more complete scenario: for example, the Wallet and the SP could be modified to interact with real national services, such as the Italian SPID demo portal or other national identity providers, thus elevating the realism of the demonstration. Furthermore, future research could aim at extending the current local configuration to support the issuance and verification of a broader range of credential types, testing the flexibility of the solution.

In conclusion, this thesis demonstrated the considerable potential of the EUDI Wallet and decentralized identity systems in enhancing security, interoperability and user control. Despite the identified limitations, the practical results and insights gained through the local implementation provide a solid foundation for further exploration and expansion of digital identity solutions aligned with eIDAS 2.0 objectives.

Bibliography

[1] J. Dawson, C. Duda, "How digital identity can improve lives in a post-COVID-19 world", January 14, 2021, https://www.weforum.org/stories/2021/01/davos-agenda-digital-identity-frameworks/

[2] European Commission, "Europe's Digital Decade." https://digital-strategy.ec.europa.eu/en/policies/europes-digital-decade

[3] P. Alfheim, "The Evolving Role of Identity - So what's next?", March 21, 2022, https://www.indykite.com/blogs/evolving-role-of-identity-whats-next

[4] BeyondTrust Corporation, "Digital Identity." https://www.beyondtrust.com/resources/glossary/digital-identity

[5] European Commission, "General Data Protection Regulation", April 27, 2016, https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX%3A02016R0679-20160504

[6] L. Garey, "What Is Digital Identity?", September 19, 2024, https://www.oracle.com/it/security/identity-management/digital-identity/#digital-identity-explained

[7] A. Weinert, "2023 identity security trends and solutions from Microsoft", 2023, https://www.microsoft.com/en-us/security/blog/2023/01/26/2023-identity-security-trends-and-solutions-from-microsoft/

[8] H. Ozsahan, "2025 Multi-Factor Authentication (MFA) Statistics Trends to Know", January 3, 2025, https://jumpcloud.com/blog/multi-factor-authentication-statistics

[9] Google, "Multi-factor authentication requirement for Google Cloud", 2025, https://cloud.google.com/docs/authentication/mfa-requirement

[10] Federal Trade Commission, "Cambridge Analytica, LLC, In the Matter of", December 18, 2019, https://www.ftc.gov/legal-library/browse/cases-proceedings/182-3107-cambridge-analytica-llc-matter

[11] K. Viezelyte, "Juggling security: How many passwords does the average person have in 2024?", April 24, 2024, https://nordpass.com/blog/how-many-passwords-does-average-person-have/

[12] Ravi, "Fundamentals of Federated Identity Authentication", December 28, 2019, https://medium.com/demystifying-security/fundamentals-of-federated-identity-authentication-bf6581cb250f

[13] Rock Solid Knowledge, "SAML2P Documentation." https://docs.identityserver.com/saml2p/protocol/examples/idp-metadata/

[14] OneLogin, "SAML Developer Tools." https://www.samltool.com/generic_sso_res.php

[15] E. T. Bray, "The JavaScript Object Notation (JSON) Data Interchange Format." RFC-8259, December 2017, DOI 10.17487/RFC8259

[16] N. Sakimura, J. Bradley, M. B. Jones, Breno de Medeiros and C. Mortimore, "Openid connect core 1.0 incorporating errata set 2." https://openid.net/specs/openid-connect-core-1_0.html#StandardClaims

[17] J. B. M. Jones and N. Sakimura, "JSON Web Token (JWT)." RFC-7519, May 2015, DOI 10.17487/RFC7519

[18] S. Josefsson, "The Base16, Base32, and Base64 Data Encodings." RFC-4648, October 2006, DOI 10.17487/RFC4648

[19] M. Jones and J. Hildebrand, "JSON Web Encryption (JWE)." RFC-7516, May 2015, DOI 10.17487/RFC7516

[20] European Parliament and Council of European Union, "Regulation (eu) no 910/2014 on electronic identification and trust services for electronic transactions in the internal market

and repealing directive 1999/93/ec", 2014, https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=uriserv%3AOJ.L_.2014.257.01.0073.01.ENG

[21] Utimaco, "eIDAS Compliance." https://utimaco.com/compliance/compliance-standardization/eidas-compliance

[22] 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, DOI 10.1109/ACCESS.2021.3115853

[23] D. G. 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, DOI 10.1109/ACCESS.2020.3007998

[24] International Organization for Standardization, "ISO/IEC 29115:2013", April 2013, https://www.iso.org/standard/45138.html

[25] eHAction, "Common eID Approach for Health in the EU - Information paper for eHN", March 24, 2021, http://ehaction.eu/wp-content/uploads/2021/06/eHAction-D8.2.4-Common-eID-Approach-for-Health-in-the-EU-_-for-adoption_19th-eHN.pdf

[26] European Commission, "Report from the commission to the european parliament and the council on the evaluation of regulation (eu) no 910/2014 on electronic identification and trust services for electronic transactions in the internal market (eidas)", 2021, https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX%3A52021SC0130&qid=1645449908028

[27] M. Sporny, D. Longley, M. Sabadello, D. Reed, O. Steele and C. Allen, "Decentralized Identifiers (DIDs) v1.0", July 19, 2022, https://www.w3.org/TR/2022/REC-did-core-20220719/

[28] M. Sporny, D. Longley, G. Kellogg, M. Lanthaler, P. Champin and N. Lindstrom, "JSON-LD 1.1", July 16, 2020, https://www.w3.org/TR/json-ld11/

[29] M. Sporny, D. Longley and D. Chadwick, "Verifiable Credentials Data Model v1.1", March 3, 2022, https://www.w3.org/TR/vc-data-model/

[30] M. Teuschel, D. Pöhn, M. Grabatin, F. Dietz, W. Hommel and F. Alt, "Don't Annoy Me With Privacy Decisions! - Designing Privacy-Preserving User Interfaces", November 20, 2023, DOI 10.1109/ACCESS.2023.3334908

[31] B. Barackath and A. Anis Akthar Sulthana Banu, "The Impact of Digital Wallets Threats and Safety Measures on the Level of Usage - A Study with Reference to Chennai", Turkish Journal of Computer and Mathematics Education, vol. 12, no. 6, 2021, pp. 95–100

[32] C. Allen, "The Path To Self-Sovereign Identity", April 26, 2016, https://www.lifewithalacrity.com/article/the-path-to-self-soverereign-identity/

[33] European Parliament and Council of European Union, "Proposal for a regulation of the european parliament and of the council amending regulation (eu) no 910/2014 as regards establishing a framework for a european digital identity", 2024, https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX%3A32024R1183&qid=1716986198888

[34] K. Martin, "eIDAS 2.0: Your Complete Guide to the EU's Plan to Revolutionise Business." https://www.truvity.com/blog/eidas-2-your-complete-guide-to-the-eu-plan-to-revolutionise-business

[35] eIDAS eID Technical Subgroup, "eIDAS SAML Attribute Profile version 1.4.1", September 9, 2024, https://ec.europa.eu/digital-building-blocks/sites/display/DIGITAL/eIDAS+eID+Profile?preview=/467109280/817168536/eIDAS%20Interoperability%20Architecture%20v.1.4.1_final.pdf

[36] European Commission and eIDAS Expert Group, "The Common Union Toolbox for a Coordinated Approach Toward a European Digital Identity Framework", February 10, 2023

[37] European Commission and eIDAS Expert Group, https://github.com/eu-digital-identity-wallet/.github/blob/main/profile/reference-implementation.md

[38] International Organization for Standardization, "ISO/IEC 18013-5:2021", September 2021, https://www.iso.org/standard/69084.html

[39] D. Fett, K. Yasuda and B. Cambpell, "Selective Disclosure for JWTs (SD-JWT)", March 1, 2025, https://datatracker.ietf.org/doc/html/draft-ietf-oauth-selective-disclosure-jwt

[40] OpenID for Verifiable Credential Issuance - draft 15, https://openid.net/specs/openid-4-verifiable-credential-issuance-1_0.html

[41] M. Jones, E. Wahlstroem, S. Erdtman and H. Tschofenig, "CBOR Web Token (CWT)." RFC-8392, May 2018, DOI 10.17487/RFC8392
[42] T. Lodderstedt, B. Campbell, N. Sakimura, D. Tonge and F. Skokan, "OAuth 2.0 Pushed Authorization Requests." RFC-9126, September 2021, DOI 10.17487/RFC9126
[43] J. B. N. Sakimura, Ed. and N. Agarwal, "Proof Key for Code Exchange by OAuth Public Clients." RFC-7636, September 2015, DOI 10.17487/RFC7636
[44] European Commission, "eIDAS-Node National IdP and SP Integration Guide v2.8." https://ec.europa.eu/digital-building-blocks/sites/display/DIGITAL/eIDAS-Node+version+2.8.2?preview=/823951453/823951465/eIDAS-Node%20National%20IdP%20and%20SP%20Integration%20Guide%20v2.8.pdf
[45] OpenID for Verifiable Credential Issuance - draft 24, https://openid.net/specs/openid-4-verifiable-presentations-1_0.html
[46] eIDAS Expert Group, https://github.com/eu-digital-identity-wallet/eudi-srv-web-issuing-eudiw-py/tree/main
[47] Python Software Foundation, https://docs.python.org/3.9/
[48] Python Software Foundation, https://docs.python.org/3.10/
[49] The OpenSSL project, http://www.openssl.org/
[50] The Apache Software Foundation, https://tomcat.apache.org/tomcat-9.0-doc/index.html
[51] Legion of the Bouncy Castle Inc., https://www.bouncycastle.org/documentation/
[52] The Apache Software Foundation, https://ignite.apache.org/docs/latest/