3

i3-MARKET Trustworthy Design

3.1 Objectives

Today, users' identities and related data are stored in siloes on centralized servers across organizations and are vulnerable to hacking. Repetitive account creation for different applications (e.g., marketplaces) and personal information (often outdated) stored in various services are some other drawbacks of that approach. Distributed/self-sovereign identities supported by decentralized systems come as a solution to those issues and facilitate interoperability, ensuring security and compliance with the privacy regulation.

Figure 3.1 shows the overall architecture of i3-MARKET. We implemented the reference implementation for the identity and access management system of i3-MARKET Backplane based on the self-sovereign identity paradigm.

The idea behind these specifications is to provide self-sovereign identity capabilities, based on distributed identity and Verifiable Credentials concepts, maintaining the most used authentication and authorization flows and standards in this moment, to facilitate the integration of stakeholders' applications and incentivize a wide adoption.

The final implementation of the i3-MARKET IAM system is based on the selected open-source technologies for SSI (Veramo), OIDC (panva/node-oidc-provider), and standard IAM.

The user-centric authentication is provided by the Verifiable Credentials micro service and the OIDC SSI Auth micro service developed using the Veramo framework.

The choice of Veramo as SSI technology has been driven from the maturity and readiness level of the uPort technology with respect to the other state-of-the-art technologies evaluated (Hyperledger Aries, Sidetree) and for the compatibility to the blockchain chosen for i3-MARKET (Hyperledger BESU), and then, after the uPort company announced the launch of this new technology (which makes uPort deprecated), we evaluated it as a very promising framework and decided to adopt it for the final implementation.

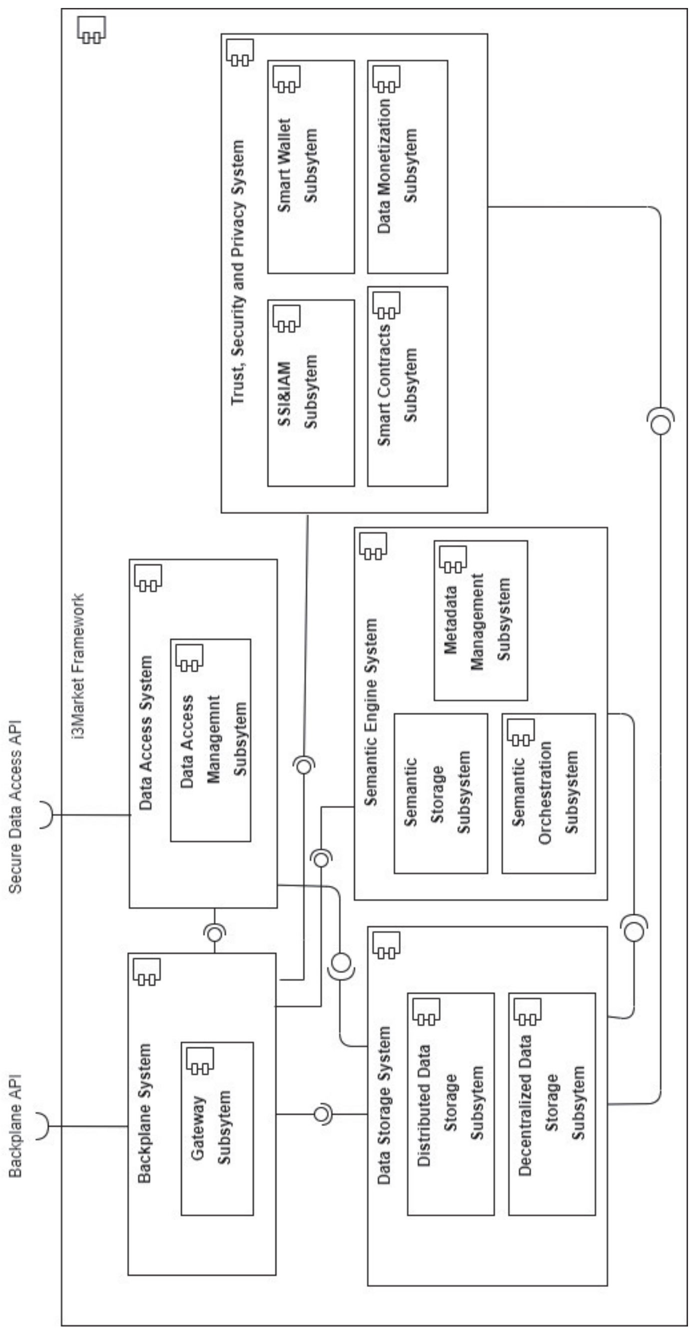

Secure Data Access API
Backplane API
i3Market Framework
Data Access System
Data Access Managemnt Subsytem
Backplane System
Gateway Subsytem
Trust, Security and Privacy System
Smart Wallet Subsytem
Data Monetization Subsytem
SSI&IAM Subsytem
Smart Contracts Subsytem
Semantic Engine System
Metadata Management Subsystem
Semantic Storage Subsystem
Semantic Orchestration Subsystem
Data Storage System
Distributed Data Storage Subsytem
Decentralized Data Storage Subsytem

Figure 3.1 Backplane architecture.

Finally, the user-centric authentication components were integrated with the i3-MARKET Smart Wallet, implementing the pairing flow and modifying the issuing and request of Verifiable Credentials in the registration and login flows.

The following are the high-level capabilities provided by the SSI and IAM subsystems:

1. User-centric authentication: authentication of users based on the self-sovereign identity paradigm.
2. Service-centric authentication: authentication of clients and users based on a standard OIDC/OAuth identity and access management system.

3.1.1 Context

The SSI and IAM subsystem block interacts with the following three building blocks:

- Data storage system: The SSI and IAM subsystem uses the data storage system, in particular the decentralized data storage component, for recording a DID document.
- Backplane system: The SSI and IAM subsystem is used from the Backplane system for authenticating and authorizing users and clients.
- Data access system: The SSI and IAM subsystem is used from the data access system for authenticating and authorizing users and clients.

3.1.2 Building block big picture

The specific SSI and IAM component diagram is shown in Figure 3.2.

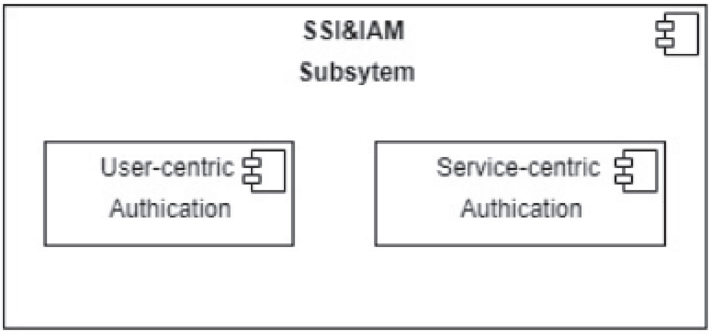


Figure 3.2 SSI and IAM components.

The SSI and IAM subsystem is in charge of providing both "user-centric authentication" and "service-centric authentication" capabilities.

Inside, we can find the following:

- A "user-centric authentication" component, responsible for providing the management of self-sovereign identity based on DID and VC and the compatibility with the OIDC standard.
- A "service-centric authentication" component, responsible for providing authentication and authorization of users and client with standard OIDC/OAuth flows, integrating the user-centric authentication component.

3.2 Technical Requirements

The list of technical requirements was not within an elicitation process to ensure the collection of not only technical needs but also service and application needs from the different stakeholders (actors) as it is described in each table from Table 3.1.

3.2.1 Actors

Table 3.1 Actors of the system.

Name	Description	Labels
Data provider	Actor who receives raw data from data owners and push it to the marketplace	Data provider
Data owner	Actor who generates the data and therefore the ultimate owner of the data. Data owners have to accept data requests to generate contracts, which leads to share the data with data consumers	Data owner
Data consumer	Consumer data shared from data owners has to create data requests through the data discovery in order for data owners to accept them. Data consumers receive only the data they want	Data consumer
Administrator	Manages the marketplace and its users	Administrator

3.2.2 User-centric authentication

The human in the loop, also known as the human-centric approach, is a design consideration that is considered in i3-MARKET. The design consideration and requirements are described in Tables 3.2 and 3.3.

3.2.2.1 Epics

Table 3.2 Epics of user-centric authentication.

Name	Description	Labels
DID management	A decentralized system that enables several key actions by three distinct entities: the controller, the relying party, and the subject. Controllers create and control DIDs, while relying parties rely on DIDs as an identifier for interactions related to the DID subject. The subject is the entity referred to by the DID, which can be anything: a person, an organization, a device, a location, or even a concept. Typically, the subject is also the controller.	Data Consumer Data Marketplace Data Provider Data Owner
Verifiable Credentials management	Verifiable Credential is a tamper-evident credential that has authorship that can be cryptographically verified though a proof. It can be used to share and prove something about the identity of a user.	Data Marketplace Data Consumer Data Provider Data Owner
OIDC client compatibility	• Staying backward compatible with existing OIDC clients and OPs that implement the OIDC specification to reach a broader community. • Adding scopes and validation rules based on VC for OIDC clients to make full use of DIDs. • Not relying on any intermediary such as a traditional centralized public or private OP while still being OIDC-compliant.	Data Marketplace

3.2.2.2 User stories

Table 3.3 User stories of user-centric authentication.

Name	Description	Labels
Create DID	As a subject, I want to create a DID so that I can manage my identity Subject: Data Consumer, Data Provider, and Data Owner	User Story Data Consumer Data Provider Data Owner
Present DID	As a user, I want to present my DID to a relying party so that I can identify myself User: Data Consumer, Data Provider, Data Owner Relying party: Data Marketplace, Data Provider	Data Consumer Data Owner Data Provider User Story
Rotate DID	As a user, I want to change the ownership of my DID so that I can maintain my identity if I change the identity provider	User Story Data Consumer Data Provider Data Owner
Delegate DID	As a user, I want to delegate my DID so that I can make other DIDs able to act on behalf of me	User Story Data Consumer Data Provider Data Owner
Recover DID	As a user, I want to recover my DID so that I can maintain my identity even if I lose my proof of control User: Data Consumer, Data Provider, Data Owner	User Story Data Consumer Data Provider Data Owner
Sign assets	As a user, I want to sign my assets so that I can demonstrate the authenticity of the asset User: Data Consumer, Data Provider, Data Owner	User Story Data Consumer Data Provider Data Owner
Verify asset signature	As a user, I want to verify asset signature so that I can authenticate the asset User: Data Consumer	User Story Data Consumer
Deactivate DID	As a user, I want to deactivate my DID so that I can delete my identity User: Data Consumer, Data Provider, Data Owner	User Story Data Consumer Data Provider Data Owner
Resolve DID	As a data marketplace, I want to resolve DID so that I can retrieve from a DID document the information to authenticate a DID subject and verify data asset signature	Data Marketplace Data Provider User Story
Authenticate DID	As a relying party, I want to authenticate DID so that I can verify the DID ownership Relying Party: Data Marketplace/Data Provider	User Story Data Marketplace Data Provider

Table 3.3 Continued.

Name	Description	Labels
Create Verifiable Credential	As a data marketplace, I want to create a Verifiable Credential so that I can provide a user an attestation of his/her role	User Story Data Marketplace
Issue Verifiable Credential	As a data marketplace, I want to issue a Verifiable Credential so that I can attest something about my users	User Story Data Marketplace
Receive Verifiable Credential	As a user, I want to receive a Verifiable Credential so that I can access the data marketplace	User Story Data Marketplace Data Provider
Store Verifiable Credential	As user, I want to store a Verifiable Credential so that I can use and keep it and use it towards any relying party	User Story Data Consumer Data Provider Data Owner
Request Verifiable Credential	As a data marketplace/data provider, I want to request Verifiable Credentials for the authenticated user so that I can give the right access to my resources	User Story Data Consumer Data Provider Data Owner
Share Verifiable Credential	As a user, I want to share a Verifiable Credential so that I can attest something towards a relying party	User Story Data Consumer Data Provider Data Owner
Verify Verifiable Credential	As a user, I want to receive a Verifiable Credential so that I can access a data marketplace	User Story Data Marketplace Data Provider
Keep track of issued Verifiable Credentials	As an issuer, I want to keep track of issued Verifiable Credentials so that I can monitor and revoke them	User Story Data Marketplace
Revoke Verifiable Credential	As an issuer, I want to revoke a Verifiable Credential so that it cannot be used	User Story Data Marketplace
OIDC authentication	As a relying party (RP), I want to authenticate users based on OIDC standards so that I do not have to change my OIDC clients RP: Data Marketplace, Data Provider	User Story Data Marketplace Data Provider

3.2.3 Service-centric authentication

The design of i3-MARKET also includes service-centric consideration; today, micro services are a trend, but this may change. Thus, the design principles are described in Tables 3.4 and 3.5.

3.2.3.1 Epics

Table 3.4 Epics of service-centric authentication.

Name	Description	Labels
Existing identity provider integration	Run a standard OpenID Connect relaying party (or OAuth2 client) on the Backplane API	Epic Data Marketplace

3.2.3.2 User stories

Table 3.5 User stories of service-centric authentication.

Name	Description	Labels
Existing identity provider authentication	As a data marketplace, I want to authenticate my users using approved external identity providers	User Story Data Marketplace

3.3 Solution Design

3.3.1 User-centric authentication

In order to provide authentication and authorization with distributed identity and Verifiable Credentials, we implemented two Node.js micro services. The Verifiable Credential micro service provides the APIs that implement the core functions to manage Verifiable Credentials, namely issuing, verifying, and revoking Verifiable Credentials, and a utility function. The OIDC SSI Auth micro service provides the API to perform the authorization code flow with PKCE using Verifiable Credentials as a proof method.

To implement the solution, we have chosen Veramo (https://veramo.io/), a framework that replaces the previous implementation of the uPort library, which is deprecated.

Both components (OIDC SSI Auth and Verifiable Credential micro service) integrate the Veramo framework and take advantage of its features to manage DID and Verifiable Credentials in Figure 3.3.

The i3-MARKET network is composed of different data marketplaces running an instance of the i3-MARKET Backplane connector. Each of them has its own OIDC SSI Auth Service and its own Verifiable Credential

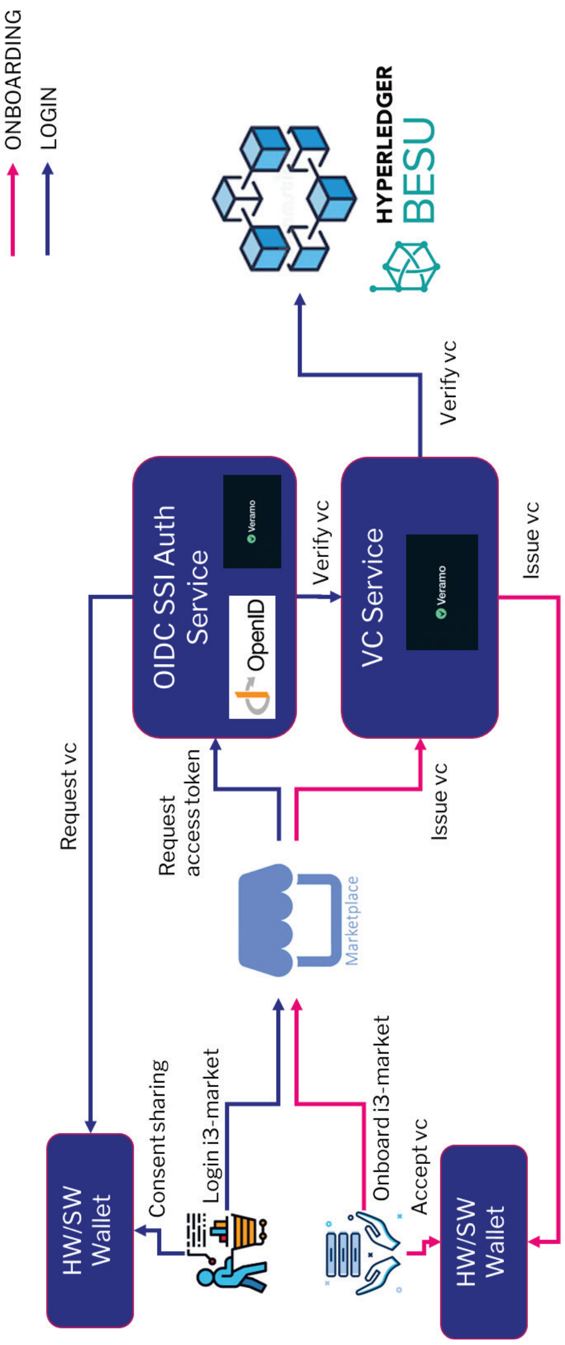


Figure 3.3 OIDC SSI Auth Service architecture.

micro service to generate, verify, and revoke Verifiable Credentials. In relation to the roles of the W3C Recommendations on verifiable credentials (https://www.w3.org/TR/vc-data-model/), the OIDC SSI Auth Service is the verifier, the Verifiable Credential micro service is the issuer (with some extra features), and the user is clearly the holder of his Verifiable Credentials. Each instance of the Verifiable Credential micro service has its own DID (https://www.w3.org/TR/did-core/) and private key used to sign Verifiable Credentials. In this way, each Verifiable Credential has as its issuer the DID of the micro service that generated it. Similarly, for revocation, only the micro service that generated a credential has the privilege to revoke it.

The user saves the Verifiable Credentials in his wallet and gives an explicit consent to share them with the OIDC SSI Auth Service when requested during the authentication phase.

The modules and detailed workflows are presented in the following subsections.

- **DID management:**

DID management is provided by Hyperledger BESU blockchain and the Veramo Ethr-DID library.

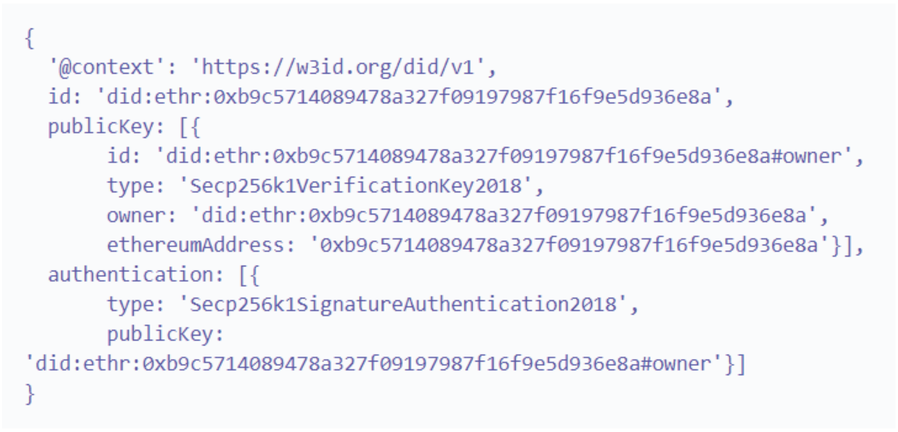
This library conforms to ERC-1056 and is intended to use Ethereum addresses as fully self-managed decentralized identifiers (DIDs).

Ethr-DID provides a scalable identity method for Ethereum addresses, which gives any Ethereum address the ability to collect on-chain and off-chain data.

This particular DID method relies on the Ethr-Did-Registry. The Ethr-DID-Registry is a smart contract that facilitates public key resolution for off-chain (and on-chain) authentication. It also facilitates key rotation, delegate assignment, and revocation to allow third-party signers on a key's behalf, as well as setting and revoking off-chain attribute data. These interactions and events are used in aggregate to form a DID document using the Ethr-Did-Resolver as shown in Figure 3.4.

DID management supports the proposed decentralized identifiers spec from the W3C Credentials Community Group.

This library has been integrated both in OIDC SSI Auth and Verifiable Credentials micro services to resolve and authenticate DID interacting with the user's wallet.



```
{
  '@context': 'https://w3id.org/did/v1',
  id: 'did:ethr:0xb9c5714089478a327f09197987f16f9e5d936e8a',
  publicKey: [{
       id: 'did:ethr:0xb9c5714089478a327f09197987f16f9e5d936e8a#owner',
       type: 'Secp256k1VerificationKey2018',
       owner: 'did:ethr:0xb9c5714089478a327f09197987f16f9e5d936e8a',
       ethereumAddress: '0xb9c5714089478a327f09197987f16f9e5d936e8a'}],
  authentication: [{
       type: 'Secp256k1SignatureAuthentication2018',
       publicKey:
'did:ethr:0xb9c5714089478a327f09197987f16f9e5d936e8a#owner'}]
}
```

Figure 3.4 Example of a DID document resolved.

The library has been used by the Verifiable Credentials micro service to create and manage the distributed identity issuing the credentials while the distributed identities of the users must be created and managed by the user's wallet.

- **Verifiable Credential management:**

For the Verifiable Credential management, the Verifiable Credentials micro service uses the Veramo framework to generate the credentials and call the i3-MARKET Wallet API to provide the credential to the user.

- **Issue a Verifiable Credential:**

The first scenario in which a data marketplace issues a Verifiable Credential to a user is the registration process. In this scenario, the micro service has to authenticate the DID of the user and then issue for this DID a Verifiable Credential that certifies the role of the user, which can be a data consumer, a data provider, or both. The workflow for the registration process is described in Figure 3.5. The entities involved are the following:

- ○ the identity holder, which is the i3-MARKET user;
- ○ the user agent, which is also the client of the OIDC (i3-MARKET data marketplace website);
- ○ the i3-MARKET wallet, which is the wallet in which credentials are stored;
- ○ the Verifiable Credential micro service (i3-MARKET data marketplace instance);

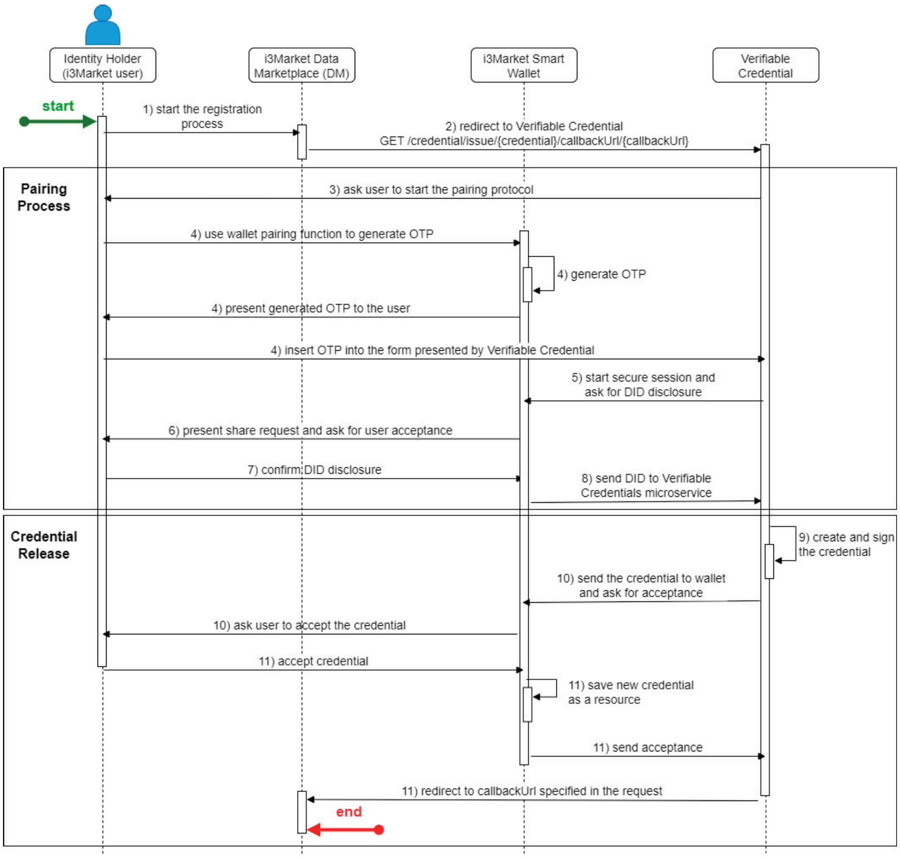

Identity Holder (i3Market user)
i3Market Data Marketplace (DM)
i3Market Smart Wallet
Verifiable Credential
start
1) start the registration process
2) redirect to Verifiable Credential GET /credential/issue/{credential}/callbackUrl/{callbackUrl}
Pairing Process
3) ask user to start the pairing protocol
4) use wallet pairing function to generate OTP
4) generate OTP
4) present generated OTP to the user
4) insert OTP into the form presented by Verifiable Credential
5) start secure session and ask for DID disclosure
6) present share request and ask for user acceptance
7) confirm DID disclosure
8) send DID to Verifiable Credentials microservice
Credential Release
9) create and sign the credential
10) send the credential to wallet and ask for acceptance
10) ask user to accept the credential
11) accept credential
11) save new credential as a resource
11) send acceptance
11) redirect to callbackUrl specified in the request
end

Figure 3.5 User registration flow.

◦ the i3-MARKET data marketplace backend.

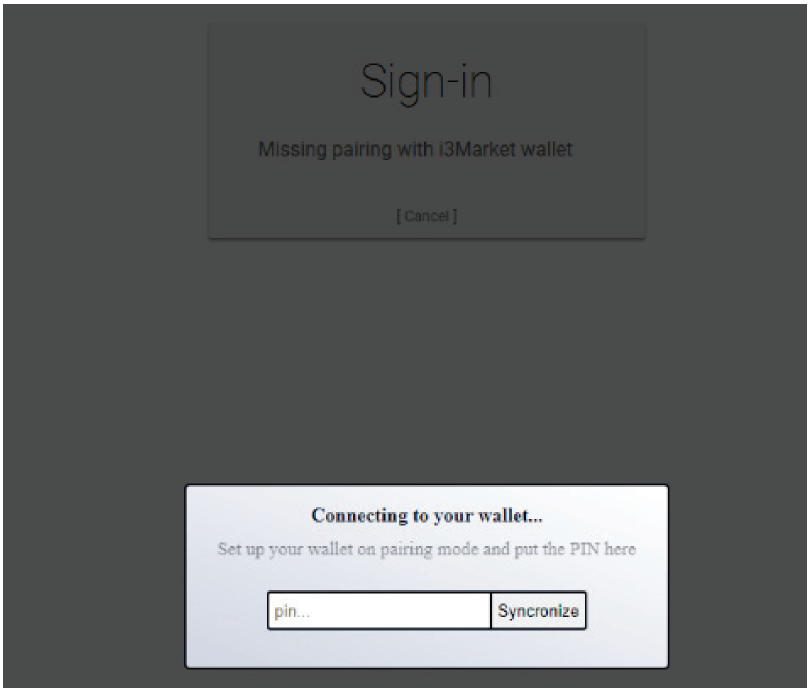
The user registration flow is shown in Figure 3.5. At the beginning, the user enters his registration data, relating to a data consumer user or a data provider. When a user is registered as a data consumer or data provider, he is registered for all the sites of the i3-MARKET network. For this reason, these two Verifiable Credentials are issued exclusively by the i3-MARKET data marketplace entity. These data are entered on registration forms in a dedicated section of the i3-MARKET powered data marketplace.

In order to know for which DID the Verifiable Credential should be issued, the i3-MARKET data marketplace must obtain the DID of the user. This part of the flow involves the *wallet-protocol session* API, which is specifically designed to open a secure connection (“paring”) with the wallet (see deliverables in “Trust, Security and Privacy Solutions for Securing Data Marketplaces” at https://www.i3-market.eu/research-and-technology-library/ for more details), using a generated OTP, to retrieve the DID of the user. In particular, the i3-MARKET data marketplace performs a GET to the *issue* API of the Verifiable Credentials micro service passing as parameters a *callbackUrl* (which indicates the URL where to redirect the user after the issue of the credential) and the *credential* formatted as JSON encoded object.

The i3-MARKET data marketplace initiates this API call, and the Verifiable Credential micro service uses the “pairing” protocol to connect to the wallet and asks for an OTP to connect to the i3-MARKET Smart Wallet.

The user generates a new OTP, using the related wallet function and presents it to the Verifiable Credential to start a secure session (Figure 3.6). Then the Verifiable Credential sends a share request to retrieve user DID.

At this point, the user receives the disclosure request through the wallet and decides whether to accept or not to share the requested identity (DID)


Sign-in

Missing pairing with i3Market wallet

[Cancel]

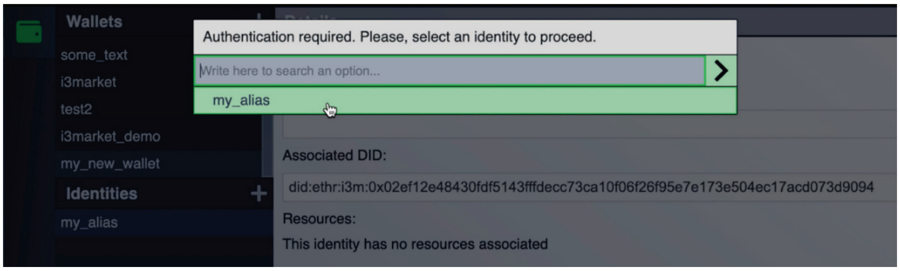
Connecting to your wallet...

Set up your wallet on pairing mode and put the PIN here

pin...

Syncronize

Figure 3.6 OTP request to start the “pairing” process.



Authentication required. Please, select an identity to proceed.

Write here to search an option...

my_alias

Associated DID:

did:ethr:i3m:0x02ef12e48430fdf5143fffdecc73ca10f06f26f95e7e173e504ec17acd073d9094

Resources:

This identity has no resources associated

Figure 3.7 i3-MARKET Smart Wallet request to disclose the DID.

(Figure 3.7). If the user agrees to share the DID, the i3-MARKET Smart Wallet sends the following access token to the Verifiable Credentials micro service via callback. At this point, the Verifiable Credentials service decodes the access token to extract the DID of the user who has authenticated.

The second part of the flow relates to the issue of a Verifiable Credential to certify that the user is a data consumer. At a high level, issuing a Verifiable Credential involves two steps:

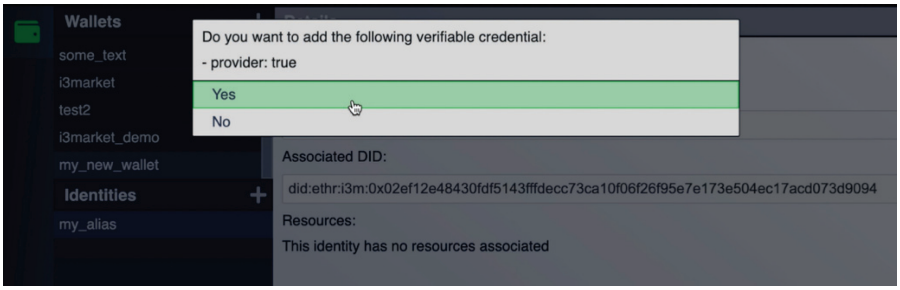
- Cryptographically signing the credential data.
- Sending the signed credential as a JWT (https://datatracker.ietf.org/doc/html/rfc7519) to the i3-MARKET Smart Wallet.

In order to create a Verifiable Credential, the Verifiable Credential micro service performs an internal API call to the POST/credential/issue/{DID} endpoint, communicating the user's DID just retrieved and the credential as form-data in the following format:

```
{
"data_consumer": true
}
```

The i3-MARKET data marketplace may request the issuance of credentials only relating to the registration process, i.e., data consumer and data provider. All other credentials, relating to the purchase of assets or services, can be requested by Data Providers. When the API is called, the Verifiable Credential micro service performs the Veramo *createVerifiableCredential* function, provided by the DID agent of the Veramo Core library as shown in Figure 3.8.

When the i3-MARKET Smart Wallet receives the credential, it verifies its signature. Each signed message has an "iss" attribute that contains a DID of the issuer. To resolve the public key of the message, a DID-resolver is used. The Veramo DID agent currently supports many DID methods, such as



Do you want to add the following verifiable credential:
- provider: true

Associated DID:
did:ethr:i3m:0x02ef12e48430fdf5143fffdecc73ca10f06f26f95e7e173e504ec17acd073d9094
Resources:
This identity has no resources associated

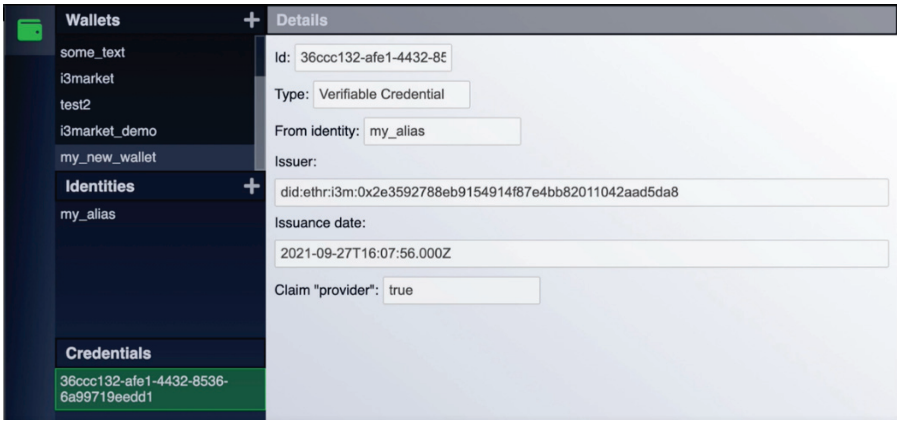
Figure 3.8 Verifiable Credential acceptance.

"did:ethr" (based on ERC-1056), "did:web" (in conjunction with blockchain-based DID, it can bootstrap trust using a web domain's existing reputation), and "did:key" (self-certifying DID method, which does not require any external utility such as blockchain). More details about supported DID methods can be found in Veramo documentation.

Being Hyperledger Besu the reference blockchain, the users' DID is "did:ethr".

After the signature verification, the wallet asks the user to accept the credential. When the user accepts the credential, it will be saved in his wallet and then be present and visible in the resources tab, which contains the list of the credentials registered in the app as shown in Figure 3.9.

At this point, the user will be redirected to the *callbackUrl*, previously specified.



Id: 36ccc132-afe1-4432-85
Type: Verifiable Credential
From identity: my_alias
Issuer:
did:ethr:i3m:0x2e3592788eb9154914f87e4bb82011042aad5da8
Issuance date:
2021-09-27T16:07:56.000Z
Claim "provider": true

Figure 3.9 Credentials list.

Once a user has saved some credentials in his wallet, he can disclose them in authentication requests, in order to certify that he holds the credentials needed to access resources or services.

- **Revoke a Verifiable Credential:**

As part of the process of working with Verifiable Credentials, it is not only necessary to issue credentials, but sometimes it is also necessary to revoke them. The ability to revoke a credential when it is no longer valid is a core function in a Verifiable Credential ecosystem. For example, suppose an i3-MARKET data provider issues a credential to access a service, and a data consumer violates the terms of use. The data provider determines that the user has violated the terms of use and, consequently, wants to suspend access to the service. In this way, the status of the Verifiable Credential needs to be changed and the next time a relying party checks the status, they will be able to see that the user is no longer valid and consequently not authorized to access the service. In order to satisfy this requirement, an API to revoke credentials has been implemented and the workflow to revoke a credential is described with Figure 3.10.

At the beginning of the flow, the data provider calls the API of the Verifiable Credentials micro service (1) communicating that a specific credential belonging to a user must be marked as revoked. The Verifiable Credential to be revoked is passed through the body parameter in the form {“JWT”: “eyJhbGc ...”}.

As an implementation choice, it was decided that only those who issued a credential are allowed to revoke it. To satisfy this requirement, a check

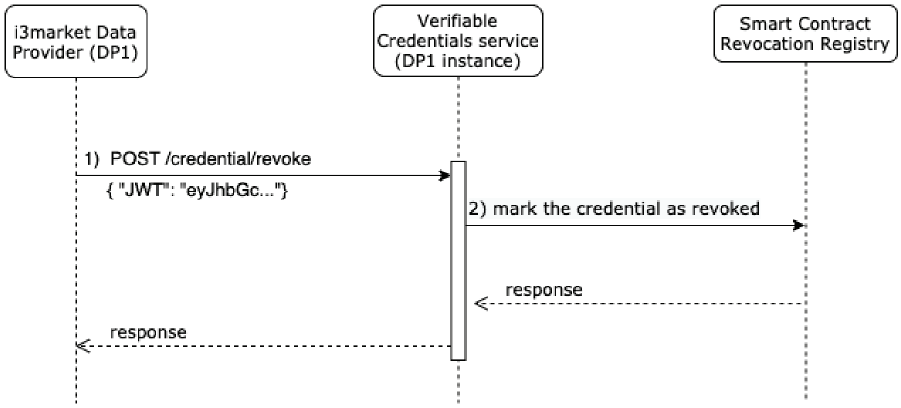

i3market Data Provider (DP1)
Verifiable Credentials service (DP1 instance)
Smart Contract Revocation Registry
1) POST /credential/revoke
{ "JWT": "eyJhbGc..."}
2) mark the credential as revoked
response
response

Figure 3.10 Revoke Verifiable Credential flow.

is made, if the issuer of the credential is the address of the issuer, then it proceeds; otherwise, it immediately blocks the flow. When the Verifiable Credentials micro service receives the API call, it writes the credential hash through a transaction in a smart contract named RevocationRegistry (2), in order to keep track of the action performed. Since the i3-MARKET blockchain is an Ethereum-type blockchain, the smart contract is written in solidity and its code is the following:

```
1  pragma solidity ^0.5.8;
2
3  contract RevocationRegistry {
4
5      mapping(bytes32 => mapping(address => uint)) private revocations;
6
7      function revoke(bytes32 digest) public {
8          require (revocations[digest][msg.sender] == 0);
9          revocations[digest][msg.sender] = block.number;
10         emit Revoked(msg.sender, digest);
11     }
12
13     function revoked(address issuer, bytes32 digest) public view returns (uint) {
14         return revocations[digest][issuer];
15     }
16
17     event Revoked(address issuer, bytes32 digest);
18 }
```

The smart contract RevocationRegistry provides two functions:

- a public function to revoke a credential;
- a public function to check if a credential is in the revocation list, i.e., it has been revoked.

The revoke function takes as input a string of 32 characters and writes a record associating it with the sender of the transaction, i.e., the address commits the line. In order to always have 32 characters, the credential before being marked on the smart contract is processed by a SHA-3 cryptographic hash algorithm and the 32-character digest is written on the smart contract.

The data structure of the smart contract is a private array of digest-address associations, named revocations (line 5). Whenever a credential is added to the register, it is mapped via the credential digest and the issuer of the transaction, i.e., the message sender. On that mapping, the block number is written, i.e., the transaction counter ID.

As an implementation choice, it was decided that only the service that issued a Verifiable Credential can revoke it. This is to prevent third parties from revoking Verifiable Credentials that they have not issued. In fact, it is reasonable that only the provider who grants access to the service can eventually revoke it.

As it is possible to notice from the smart contract code, another requirement to be able to add a Verifiable Credential in the smart contract is the fact that it is not already present in the register (line 8), i.e., in the corresponding mapping, there is not a block number indicating which transaction added the credential. If it has not already been added, then it is possible to write it (line 9). When the transaction is successfully added, an event is emitted (line 11), which communicates the issuer of the transaction and the digest of the newly added credential in the register (line 17).

A possible problem is the fact that this smart contract trusts that what is written to the registry is actually a valid digest of a credential in JWT format. In this implementation, there is no kind of access control list that allows only some addresses to write in the smart contract. In fact, once a smart contract is deployed in blockchain, its public methods can be called from any valid address. It is therefore possible that any address can call these methods and write non-consistent information to the register. This problem can be solved with a list of trusted issuers of transactions. In fact, it is possible to consider an issuer as trusted if it also implements the correct cryptographic hash algorithms on the Verifiable Credential before writing it to the register.

- **Verify a Verifiable Credential:**

The verification is the process of evaluation of a Verifiable Credential, in order to determine whether it is authentic and timely valid for the issuer or the presenter. This process includes the following checks:

- the credential conforms to the specification;
- the proof method is satisfied, i.e., the cryptographic mechanism used to prove that the information in a Verifiable Credential was not tampered;
- the credential is not marked as revoked in the smart contract registry.

The Veramo credential library provides the methods for the first two checks, while for the third it is necessary to implement a call to the smart contract registry. The flow for verifying a credential is described in Figure 3.11.

In the implemented solution, in step (1) the data provider calls the Verifiable Credentials micro service, specifying the credential in JWT format to be verified in the request body. Since verifying the presence of a Verifiable Credential on the registry is an operation that does not change the status of the credential, this can be done by any instance of the micro service. In step

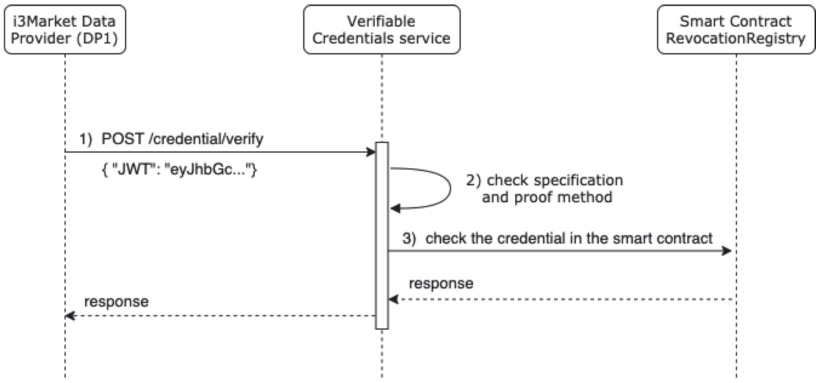

i3Market Data Provider (DP1)
Verifiable Credentials service
Smart Contract RevocationRegistry
1) POST /credential/verify
{ "JWT": "eyJhbGc..."}
2) check specification and proof method
3) check the credential in the smart contract
response
response

Figure 3.11 Verify Verifiable Credential flow.

(2), the Verifiable Credentials micro service checks that the issuer is valid and that the credential is in a format that complies with the data provider's specifications. If there is no problem with the credential, then it computes the hash of the credential using a SHA-3 cryptographic hash algorithms, which produces a 32-character digest. Then, in step (3), the Verifiable Credential micro service calls the "revoked" method of the smart contract registry, specifying the issuer of the Verifiable Credential and the 32-character digest.

The credential issuer is specified in the JWT and since only the issuer of a credential has the permissions to revoke it, it is sufficient to check that only his address, associated with the credential, is not present in the register. As it is possible to see in the solidity code of the smart contract, detailed in the previous section, this method returns the block number when it was revoked by the "issuer", or 0 if it was not. In this way, it is possible to know if the credential has been revoked or not. This information is then returned as a response to the data provider, who will decide for himself what the next steps will be, for example requesting the issuance of a new valid credential or informing the user that he can no longer request access to that data or service. This API is used in the integration of the OIDC identity provider. In fact, to authenticate a user on the basis of the revealed credentials, a further check on the registry is necessary to ensure that the credential is not revoked.

- **OIDC compatibility:**

The use of Verifiable Credentials allows the distributed and decentralized management of users. In particular, users can use Verifiable Credentials issued as a certificate to obtain a token necessary to access specific services or protected resources within the marketplace. In order to retrieve the Verifiable Credentials and use them in an authorization process, a certified open-source Open ID Connect provider (https://github.com/panva/node-oidc-provider) has been enhanced with wallet API library. In this way, users can be authenticated and authorized based on the Verifiable Credentials they hold as shown in Figure 3.12.

In step (1), the user wants to access a resource or service in the marketplace. The resources and services are made available by data providers, who expect to receive a valid access token and ID token, with the necessary scope to access the resource or service. So, the first thing a data provider website does is to initialize the authentication flow (2). The authentication with authorization code flow + PKCE is done through an OAuth 2.0 SDK (https://github.com/IdentityModel/oidc-client-js), which first generates

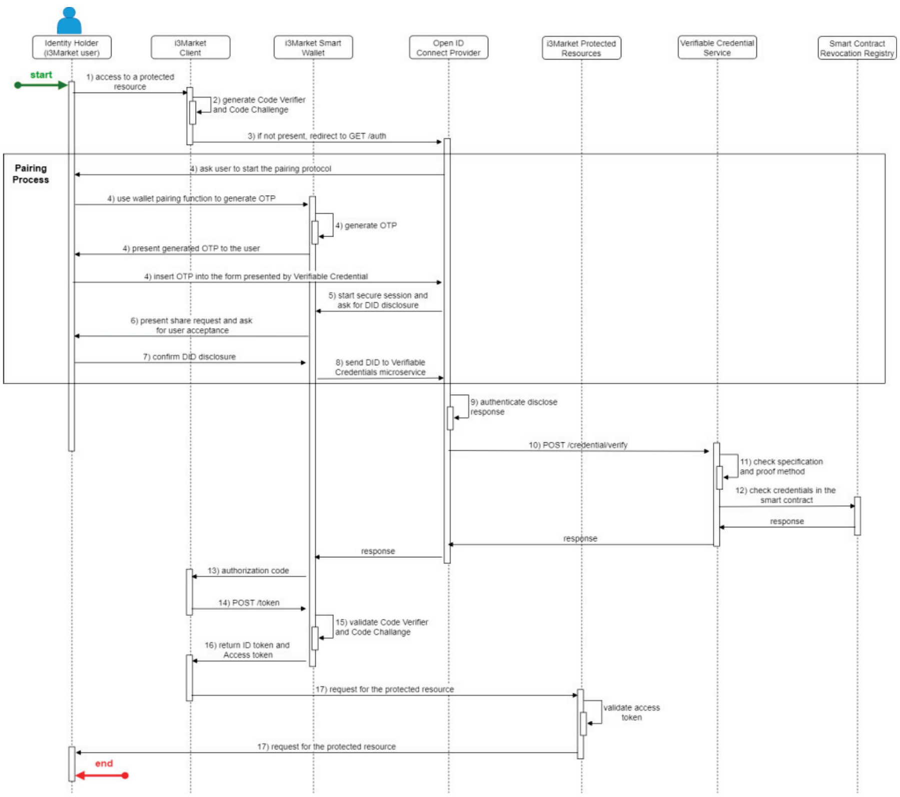

Identity Holder (i3Market user)
i3Market Client
i3Market Smart Wallet
Open ID Connect Provider
i3Market Protected Resources
Verifiable Credential Service
Smart Contract Revocation Registry
start
1) access to a protected resource
2) generate Code Verifier and Code Challenge
3) if not present, redirect to GET /auth
Pairing Process
4) ask user to start the pairing protocol
4) use wallet pairing function to generate OTP
4) generate OTP
4) present generated OTP to the user
4) insert OTP into the form presented by Verifiable Credential
5) start secure session and ask for DID disclosure
6) present share request and ask for user acceptance
7) confirm DID disclosure
8) send DID to Verifiable Credentials microservice
9) authenticate disclose response
10) POST /credential/verify
11) check specification and proof method
12) check credentials in the smart contract
response
response
response
13) authorization code
14) POST /token
15) validate Code Verifier and Code Challange
16) return ID token and Access token
17) request for the protected resource
validate access token
17) request for the protected resource
end

Figure 3.12 Authorization flow.

a code verifier and a code challenge. Specifically, the OAuth 2.0 SDK creates a cryptographically random code verifier and from this generates a code challenge. After that, the authorization code + PKCE flow is initialized with the first call/authorize. The main difference of this Open ID Connect provider compared to traditional ones is that it requires the disclosure of Verifiable Credentials. To specify the credentials to be revealed, the scope field is used.

The Open ID Connect provider has the following static scopes:

- openid: Mandatory for the Open ID Connect standard. It returns the sub-field of the ID token, and its value is the user DID.
- profile: It adds information about the user profile into the ID token.
- vc: It adds the field verifiable claims into the ID token. Useful when the relying party wants to check any information about the verifiable claims asked.

Compared to the standard scope of Open ID Connect, the scopes added are vc and vce. On the other hand, the standard email scope, which returns the user's email, is not present.

There are two different types of scopes:

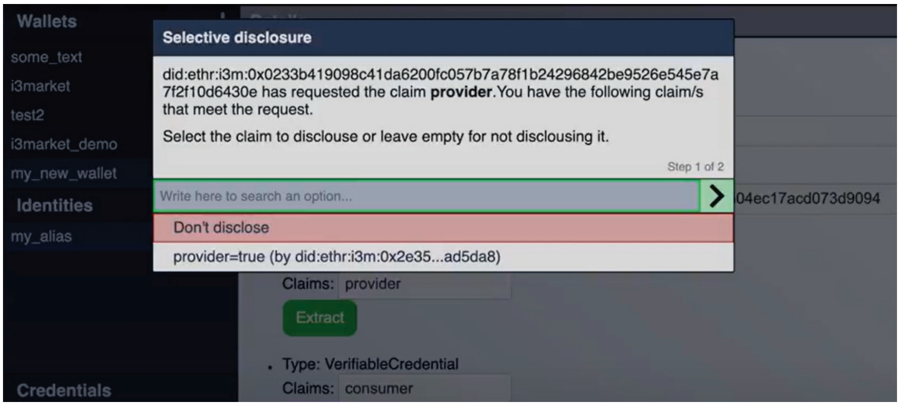
- vc:vc name: It asks the user for the specific verifiable claim vc name. It is optional; so the user can decide whether to append it or not. If the issuer of the verifiable claim is not trusted, it will be added into untrusted verifiable claims array of the ID token. These arrays are described at the end of this section.
- vce:vc name: It asks the user for the essential verifiable claim with name: vc name. It is mandatory; so if the user does not provide it or the issuer is untrusted, the login and consent process will be cancelled.

After specifying in the scope field which credentials need to be disclosed, the OAuth SDK initializes the authentication process, performing the call to the /authorize endpoint (3).

The Open ID Connect provider performs a selective disclosure request (5), using the Veramo libraries, ask to pair i3-MARKET Smart Wallet, using "pairing" protocol (4).

At this point, a notification will appear on the wallet with the authentication request (Figure 3.13), specifying the credentials that must be revealed (6).

After the disclosure of the required credentials, a callback to the Open ID Connect provider (9) follows. The Open ID Connect checks if all the



Selective disclosure

did:ethr:i3m:0x0233b419098c41da6200fc057b7a78f1b24296842be9526e545e7a7f2f10d6430e has requested the claim **provider**.You have the following claim/s that meet the request.

Select the claim to disclose or leave empty for not disclousing it.

Step 1 of 2

Write here to search an option...

Don't disclose

provider=true (by did:ethr:i3m:0x2e35...ad5da8)

04ec17acd073d9094

Claims: provider

Type: VerifiableCredential

Claims: consumer

Figure 3.13 Disclosure of the data provider credential.

required Verifiable Credentials are present and if the Verifiable Credential issuer is trusted (10). Subsequently, it remains to check that the credentials are valid and not revoked. In particular, the credentials are sent in JWT format through the verify (11) API, which checks that they have not expired, calculates the hash, and checks if they are present in the Revocation Registry (12). It then returns the array of revoked or invalid credentials as a response. If the response array is empty, then all credentials are valid. If all credentials are valid, the Open ID Connect provider returns the authorization code to the OAuth SDK of the i3-MARKET Data Provider Client (13). The Data Provider SDK performs POST /token (14). The code verifier and code challenge are checked (15) and the ID token and the access token are returned to the Data Provider website (16). Now that the Data Provider website has a valid access token, it can get the resource (17). When the authorization and authentication process finishes, two tokens are returned: access token and ID token. Through the ID token, it is possible to know which of the revealed Verifiable Credentials are verified (trusted) or not (untrusted).

3.4 Diagrams

The following diagrams describe the processes involving the components of the SSI and IAM subsystem.

The diagrams assume that the user created and controlled with his crypto wallet a distributed identity using Ethereum DID management.

3.4.1 Identity authentication

The process in Figure 3.14 describes how a self-sovereign identity is authenticated as managed by a crypto wallet using Ethereum DID management.

The user-centric authentication component create a challengeRequest to retrieve the user's DID and then check the challengeResponse (signed by user's wallet) to verify if the user controls the DID retrieving the corresponding DID document.

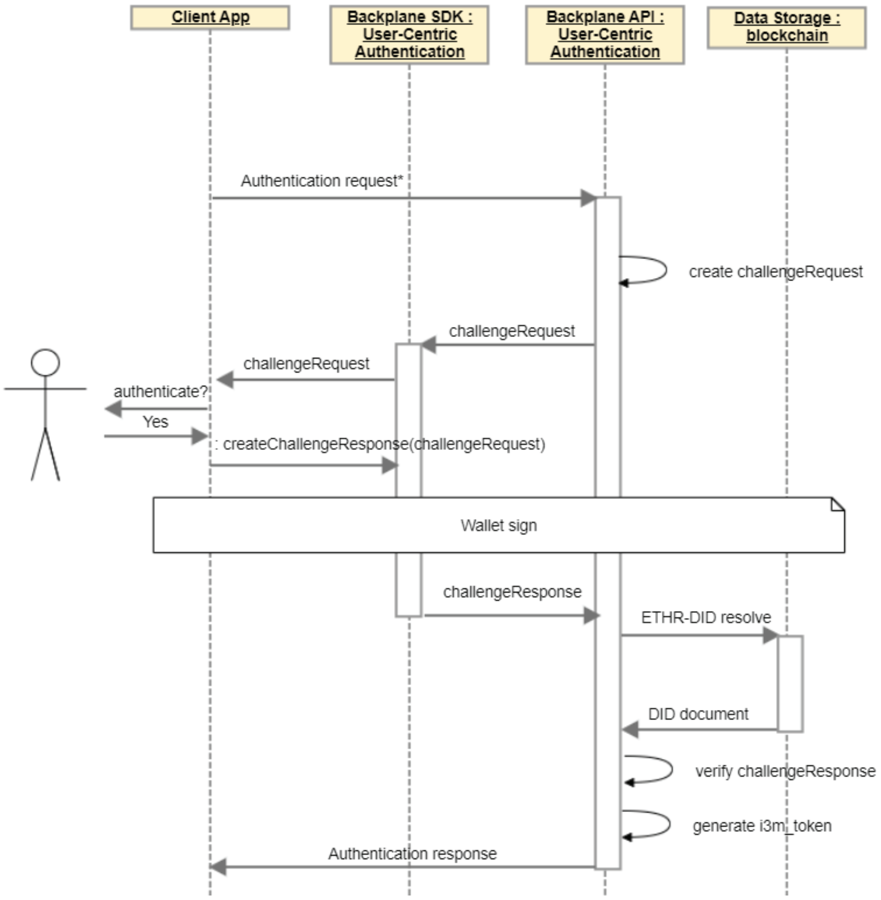


Figure 3.14 Identity authentication process.

3.4.2 User registration

The process illustrated in Figure 3.15 describes how a client application can register a self-sovereign identity as i3-MARKET user issuing a Verifiable Credential attesting his role.

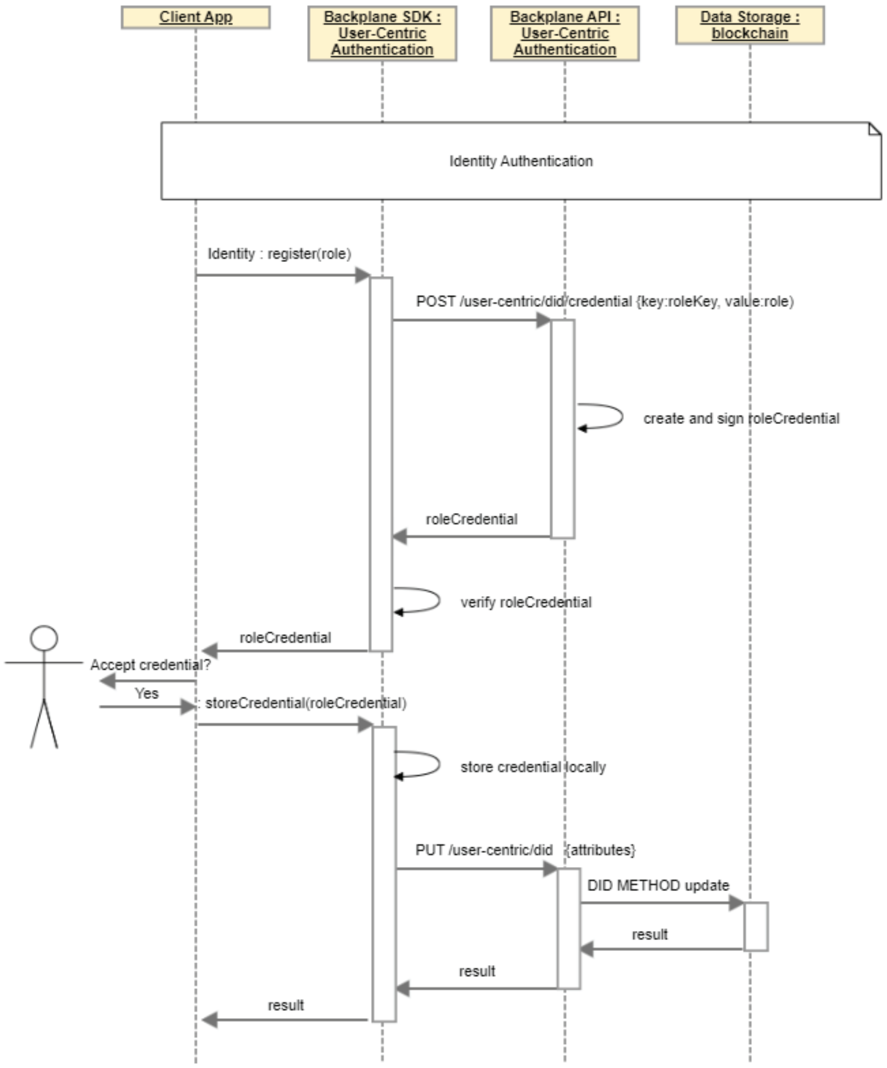

Client App
Backplane SDK : User-Centric Authentication
Backplane API : User-Centric Authentication
Data Storage : blockchain
Identity Authentication
Identity : register(role)
POST /user-centric/did/credential {key:roleKey, value:role}
create and sign roleCredential
roleCredential
verify roleCredential
roleCredential
Accept credential?
Yes
: storeCredential(roleCredential)
store credential locally
PUT /user-centric/did {attributes}
DID METHOD update
result
result
result

Figure 3.15 User registration process.

After DID authentication and the verification of additional information disclosed by the user, the client app issues a Verifiable Credential for that DID, which attest the role of the user (data consumer, data provider, or both). User's wallet stores the credential locally and update the DID document.

3.4.3 OIDC authorization (authentication code + PKCE)

The following process (Figure 3.16) represents how a client application can be authorized by an i3-MARKET user to access a protected API and obtain information about the user using a standard OpenID Connect Authentication code flow with PKCE.

User-centric authentication component is integrated in a standard OIDC IAM as federated identity provider.

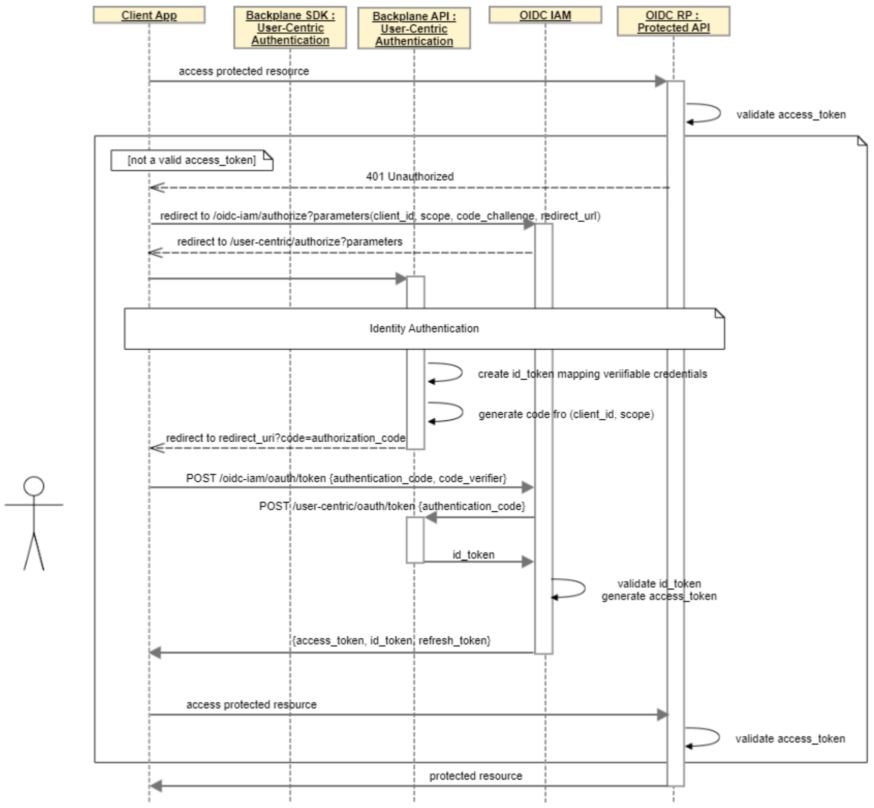

Client App
Backplane SDK : User-Centric Authentication
Backplane API : User-Centric Authentication
OIDC IAM
OIDC RP : Protected API
access protected resource
validate access_token
[not a valid access_token]
401 Unauthorized
redirect to /oidc-iam/authorize?parameters(client_id, scope, code_challenge, redirect_url)
redirect to /user-centric/authorize?parameters
Identity Authentication
create id_token mapping veriifiable credentials
generate code fro (client_id, scope)
redirect to redirect_uri?code=authorization_code
POST /oidc-iam/oauth/token {authentication_code, code_verifier}
POST /user-centric/oauth/token {authentication_code}
id_token
validate id_token
generate access_token
{access_token, id_token, refresh_token}
access protected resource
validate access_token
protected resource

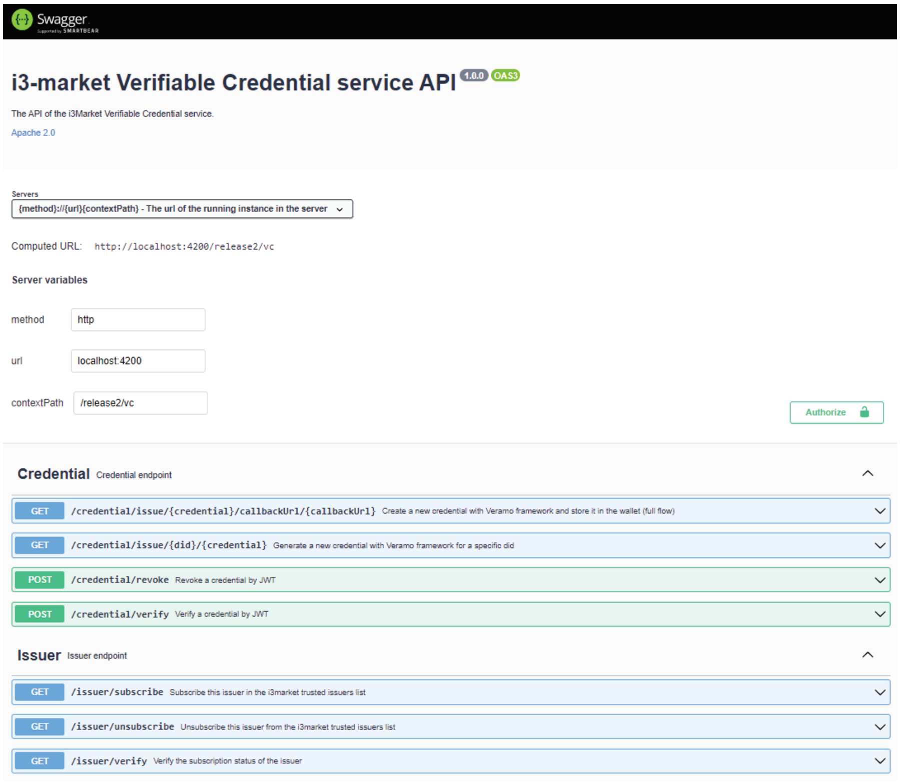
Figure 3.16 OIDC authorization process.

When the client application tries to call a protected resource without a valid access_token, it is redirected to the OIDC IAM authorization endpoint and then to user-centric authentication authorization endpoint showing the login page.

When the user logs in with a wallet, an id_token is created with the DID and the VC associated to the requested scopes and an authentication code is provided to the client to call the token endpoint and receive a valid access_token, a refresh token, and the id_token.

3.5 Interfaces

The interfaces of the final version of Verifiable Credential micro service and OIDC SSI Auth micro service, composing the user-centric authentication component, are presented in Figures 3.17 and 3.18.


i3-market Verifiable Credential service API 1.0.0 OAS3

The API of the i3Market Verifiable Credential service.

Apache 2.0

Servers

{method}://{url}{contextPath} - The url of the running instance in the server

Computed URL: `http://localhost:4200/release2/vc`

Server variables

method: http

url: localhost:4200

contextPath: /release2/vc

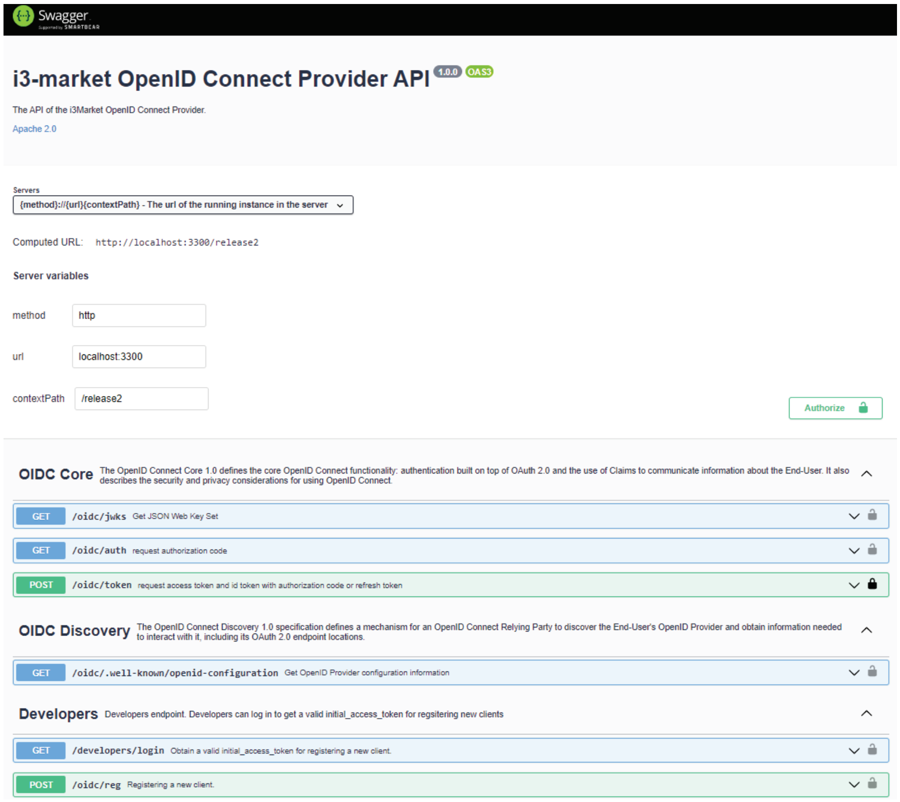
Credential — Credential endpoint

- GET `/credential/issue/{credential}/callbackUrl/{callbackUrl}` Create a new credential with Veramo framework and store it in the wallet (full flow)
- GET `/credential/issue/{did}/{credential}` Generate a new credential with Veramo framework for a specific did
- POST `/credential/revoke` Revoke a credential by JWT
- POST `/credential/verify` Verify a credential by JWT

Issuer — Issuer endpoint

- GET `/issuer/subscribe` Subscribe this issuer in the i3market trusted issuers list
- GET `/issuer/unsubscribe` Unsubscribe this issuer from the i3market trusted issuers list
- GET `/issuer/verify` Verify the subscription status of the issuer

Figure 3.17 Verifiable Credential micro service specification.



Swagger

i3-market OpenID Connect Provider API 1.0.0 OAS3

The API of the i3Market OpenID Connect Provider.

Apache 2.0

Servers

{method}://{url}{contextPath} - The url of the running instance in the server

Computed URL: `http://localhost:3300/release2`

Server variables

method: http

url: localhost:3300

contextPath: /release2

Authorize

OIDC Core The OpenID Connect Core 1.0 defines the core OpenID Connect functionality: authentication built on top of OAuth 2.0 and the use of Claims to communicate information about the End-User. It also describes the security and privacy considerations for using OpenID Connect.

- GET `/oidc/jwks` Get JSON Web Key Set
- GET `/oidc/auth` request authorization code
- POST `/oidc/token` request access token and id token with authorization code or refresh token

OIDC Discovery The OpenID Connect Discovery 1.0 specification defines a mechanism for an OpenID Connect Relying Party to discover the End-User's OpenID Provider and obtain information needed to interact with it, including its OAuth 2.0 endpoint locations.

- GET `/oidc/.well-known/openid-configuration` Get OpenID Provider configuration information

Developers Developers endpoint. Developers can log in to get a valid initial_access_token for regsitering new clients

- GET `/developers/login` Obtain a valid initial_access_token for registering a new client.
- POST `/oidc/reg` Registering a new client.

Figure 3.18 OIDC SSI Auth micro service specification.

3.6 Background Technologies

3.6.1 JSON Web Token (JWT)

The JSON Web Token (JWT) is an open standard (RFC 7519) that defines a schema in JSON format for exchanging information between various services. The generated token can be signed (with a secret key that only those who generate the token know) using the HMAC algorithm or using a pair of keys (public/private) using the RSA or ECDSA standards. JWTs are widely used to authenticate requests in Web Services and OAuth 2.0 authentication mechanisms where the client sends an authentication request to the server and the server generates a signed token and returns it to the client who, from that moment on, will use to authenticate subsequent requests. The structure of the token consists of three fundamental parts:

- Header
- Payload
- Signature

The header contains two main information: the type of token (in this case valued to JWT because it is a JSON Web Token) and the type of encryption algorithm used.

```
{
"alg": "HS256",
"typ": "JWT"
}
```

The payload contains the interchange information. It is possible to categorize them into three blocks:

- Registered parameters: They are predefined properties that indicate information about the token (issuer, audience, expiration, issued at, and subject).
- Private parameters: Here, it is possible to enter new fields, such as verifiable claims, having full extensibility, thanks to the JSON structure.
- Public parameters: They refer to parameters defined in the IANA JSON Web Token Registry, and they can be compiled at will by paying attention to the content that is entered to avoid conflicts with the registered and private parameters.

```
{
"iss": "app_name",
"name": "Mario Rossi",
"iat": 1540890704,
exp": 1540918800,
"user": {
"profile": "editor"
}
}
```

The token is generated by encoding the header and payload in base 64 and joining the two results by separating them by a “.”, and then the algorithm indicated in the header is applied to the string obtained using a secret key. It is possible to verify and unpack a JWT online using the official website.

Fortunately, it is not necessary to re-implement the encryption logic; there are many libraries to generate JWT depending on the programming language. Security is guaranteed by the fact that the token is signed with a server-side secret key; so if it is corrupted or modified by an external agent, it will not pass validation.

3.6.2 OpenID Connect (OIDC)

OpenID Connect 1.0 (https://openid.net/connect/) is a simple identity layer on top of the OAuth 2.0 protocol. It allows clients to verify the identity of the end-user based on the authentication performed by an authorization server, as well as to obtain basic profile information about the end-user in an interoperable and REST-like manner.

OpenID Connect allows clients of all types, including Web-based, mobile, and JavaScript clients, to request and receive information about authenticated sessions and end-users. The specification suite is extensible, allowing participants to use optional features such as encryption of identity data, discovery of OpenID providers, and session management, when it makes sense for them.

3.6.3 Decentralized identity (DID)

Decentralized identifiers (DIDs) are a new type of identifier that enables verifiable, decentralized digital identity. A DID identifies any subject (e.g., a person, organization, thing, data model, abstract entity, etc.) that the controller of the DID decides that it identifies. In contrast to typical, federated identifiers, DIDs have been designed so that they may be decoupled from centralized registries, identity providers, and certificate authorities. Specifically, while other parties might be used to help enable the discovery of information related to a DID, the design enables the controller of a DID to prove control over it without requiring permission from any other party. DIDs are URIs that associate a DID subject with a DID document allowing trustable interactions associated with that subject.

3.6.4 Self-sovereign identity and blockchain

Today, users' identities and related data are stored in siloes on centralized servers across organizations and are vulnerable to hacking. Repetitive account creation for different applications (e.g., marketplaces), and personal information (often outdated) stored in various services are the disadvantages of that approach.

Self-sovereign identities supported by decentralized systems come as a solution for the following issues:

- Identity and personal data are stored with the user.
- Claims and attestations can be issued and verified between users and trusted parties.
- Users selectively grants access to data.
- Data only needs to be verified a single time.

Blockchain technology, proving decentralization, immutability, and cryptographic security allow the creation of credentials that could be issued and verified without the need of a central certification authority and could be owned by the end-users and directly shared with third parties without involving the credential issuer.

3.6.5 Verifiable Credentials (VC)

As in the physical world, a credential is a set of information that identifies an entity. In particular, the information represents:

- Information related to identifying the subject of the credential (for example, a photo, name, or identification number).
- Information related to the issuing authority (for example, a city government, national agency, or certification body).
- Information related to the type of credential this is (for example, a Dutch passport, an American driving license, or a health insurance card).
- Information related to specific attributes or properties being asserted by the issuing authority about the subject (for example, nationality, the classes of vehicle entitled to drive, or date of birth).
- Evidence related to how the credential was derived.
- Information related to constraints on the credential (for example, expiration date or terms of use).

A Verifiable Credential can represent all of the same information that a physical credential represents (Figure 3.19). The addition of technologies, such as digital signatures, makes Verifiable Credentials more tamper-evident and more trustworthy than their physical counterparts.

Holders of Verifiable Credentials can generate verifiable presentations and then share these verifiable presentations with verifiers to prove they possess Verifiable Credentials with certain characteristics.

Both Verifiable Credentials and verifiable presentations can be transmitted rapidly, making them more convenient than their physical counterparts when trying to establish trust at a distance.

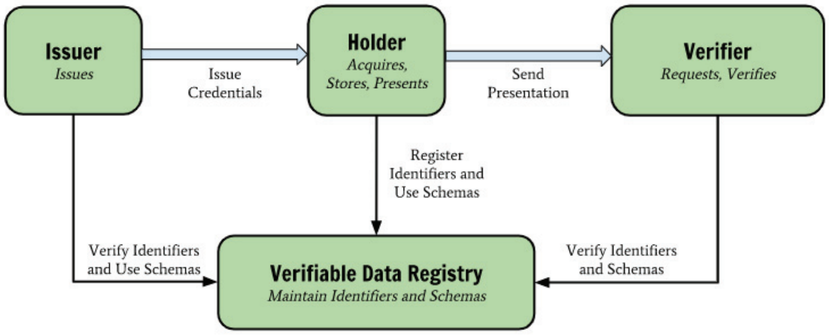

Issuer
Issues
Issue Credentials
Holder
Acquires, Stores, Presents
Send Presentation
Verifier
Requests, Verifies
Register Identifiers and Use Schemas
Verify Identifiers and Use Schemas
Verifiable Data Registry
Maintain Identifiers and Schemas
Verify Identifiers and Schemas

Figure 3.19 Verifiable Credentials model.

Verifiable Credentials are useful in a self-sovereign identity ecosystem because they assert information about the user to whom a credential is issued and can be directly verified by any third-party by involving the issuer. In the context of the project, users are asked to disclose Verifiable Credentials, which attest particular attributes issued by a specific data marketplace. The certified attributes and permissions are used to obtain an OAuth access token that allows the use of Backplane services or access to resources in the marketplace. Verifiable Credentials are therefore used as a proof method in the authorization flow. If a credential is valid, it means that the user is authorized to access a resource or service that requires the holding of that credential. For this reason, a service that generates Verifiable Credentials is necessary. Once a Verifiable Credential is saved by users in their wallets, anyone who receives the Verifiable Credential and has access to the DID of the users can then confirm that the Verifiable Credential has been issued by a trusted server and has not been revoked for some reason.

To implement the solution, we have chosen to use Veramo, a framework that replaces the previous implementation of the uPort library, which is deprecated.

Both components (OIDC SSI Auth and Verifiable Credential micro service) integrate the Veramo framework and take advantage of its features to manage DID and Verifiable Credentials.