# BROKERAGE AND MARKET PLATFORM FOR PERSONAL DATA

## *D3.1 Self-Sovereign Identity Solution First Release*

www.krakenh2020.eu

This project has received funding from the European Union's Horizon 2020 (H2020) research and innovation programme under the Grant Agreement no 871473

# D3.1 Self-Sovereign Identity Solution First Release

| | |
|---|---|
| **Grant agreement** | 871473 |
| **Work Package Leader** | Atos |
| **Author(s)** | Angel Palomares (Atos) |
| **Contributors** | Angel Palomares Pérez (Atos), Juan Carlos Pérez Baún (Atos), Davide Porro (ICERT), Luca Boldrin (ICERT), Giancarlo Degani (ICERT), Christof Rabensteiner (TUG), Rob Holmes (TEX), Silvia Gabrielli (FBK). |
| **Reviewer(s)** | Andreas Abraham (TUG), Tilen Marc (XLAB) |
| **Version** | v1.0 |
| **Due Date** | 31/07/2021 |
| **Submission Date** | 31/08/2021 |
| **Dissemination Level** | Public |

**Release History**

| Version | Date | Description | Released by |
|---|---|---|---|
| v0.1 | 29/04/2021 | Initial version | Sara Diez |
| v0.2 | 03/06/2021 | Distribution of sections | Juan Carlos Pérez Baún |
| v0.3 | 10/06/2021 | Changes on the ToC | Angel Palomares |
| v0.4 | 17/06/2021 | Contribution for Chapter 2 and chapter 3 | Angel Palomares |
| v0.5 | 20/06/2021 | Contribution on KWCT and LIM | Davide Porro |
| v0.6 | 22/06/2021 | Add Chapter 6.1 | Christof Rabensteiner |
| v0.7 | 25/06/2021 | Editor revision | Angel Palomares |
| v0.8 | 28/06/2021 | Complete Chapter 6.1, Add Chapter 6.2.1 and 6.2.2 | Christof Rabensteiner |
| v0.9 | 30/06/2021 | Complete Chapter 6.2 | Christof Rabensteiner |
| v0.10 | 02/07/2021 | Update Sequence 44, Add Intro to Chapter 6 | Christof Rabensteiner |
| v0.11 | 05/07/2021 | Contribution chapter 1 and chapter 7 | Angel Palomares<br>Juan Carlos Pérez Baún |
| v0.12 | 27/07/2021 | Updates on chapter 6, chapter 3 and chapter 4 | Christof Rabensteiner<br>Rob Holmes<br>Giancarlo Degani |
| v0.13 | 11/08/2021 | Update Executive Summary and Conclusion sections. Sanitize sections. | Juan Carlos Pérez Baún |
| v0.14 | 16/08/2021 | Update section 4.1.4 | Angel Palomares |
| v0.15 | 20/08/2021 | Addressing reviewers' comments first iteration | Juan Carlos Pérez Baún<br>Christof Rabensteiner |
| v0.16 | 30/08/2021 | Addressing reviewers' comments | Davide Porro, Luca Boldrin, Romualdo Carbone, Angel Palomares |
| v0.17 | 31/08/2021 | Addressing remaining reviewers' comments. Formatting and sanitizing | Angel Palomares Pérez<br>Juan Carlos Pérez Baun |
| v1.0 | 31/08/2021 | Submitted version | Atos |

# Table of Contents

# List of Figures

# List of Acronyms

| Acronym | Description |
|---|---|
| ACA-Py | Aries Cloud Agent Python |
| AGID | Italian Agency for Digital Identity |
| API | Application Programming Interface |
| CRUD | Create Read Update Delete |
| DID | Decentralised IDentifier |
| EBSI | European Blockchain Service Infrastructure |
| ERS | External Remote Storage |
| ESSIF | European Self Sovereign Identity Framework |
| JNI | Java Native Interface |
| JSON | JavaScript Object Notation |
| JWT | Java Web Token |
| KMS | Key Manager Service |
| KWCT | KRAKEN Web Company Tool |
| LIM | Legal Identity Manager |
| PAdES | Pdf Advanced Electronic Signature |
| REST | Representational State Transfer |
| SDK | Software Development Kit |
| SSI | Self-Sovereign Identity |
| SP | Service Provider |
| TIR | Trusted Issuer Registry |
| TSR | Trusted Schema Registry |
| VC | Verifiable Credential |
| VDR | Verifiable Data Registry |
| VI | Verifiable ID |
| WP | Work Package |

# Executive Summary

The main objective of the Work Package 3 is to provide the tools needed for easing the personal data sharing between different stakeholders and for access management, by following a decentralized and self-sovereign approach. The Self-Sovereign Identity (SSI) prototype provided in this deliverable 3.1 contains the description of the components identified for building the Self-Sovereign Identity (SSI) solution which is one of the three key pillars of the KRAKEN platform. The KRAKEN consortium has considered the compatibility with ESSIF including ESSIF/EBSI services in the "Trusted Framework" components.

The document provides a description of the components identified on the tasks of WP3 related to SSI. T3.1 "Verifiable Credential management tools" produces the components for the verifiable credentials (VCs) management, including the creation and validation of VCs leveraging eIDAS compliance. T3.2 "Universal Ledger Resolver" provides the components for facilitating the integration of the SSI solutions for VC resolution, including the use of several ledgers. Moreover, deals with the Trusted Framework for issuers, schema and DID and revocation. T3.3 "Edge Tools for the Decentralised Identity Solution" provides the components for integrating SSI solutions with the Service Providers (SPs) services, such as a mobile app with the SSI functionalities. T3.4 "SSI Wallet Management" delivers the components assuring the secure cloud-based storage system of protected personal data leveraging cryptographic mechanisms.

The final version of the SSI tools will be released in May 2022 as D3.2 Self sovereign identity solution final release.

# 1 Introduction

## 1.1 Purpose of the document

The aim of this document is to provide an update of the main results obtained in the developments of the tasks: T3.1, T3.2, T3.3 and T3.4. This document firstly introduces the common global design, describing the overall architecture adopted for the Self-Sovereign Identity solution, it outlines the main components and the communication/interactions among them. Secondly, mapped by the different tasks, each component is described in detail, specifying all the sub-modules associated and their specific considerations and constrains.

The work presented in D3.1 is based on the work performed in WP2 "Technical aspects and architecture specifications", specifically in T2.4 "Self-sovereign Identity specifications" (See D2.4 "KRAKEN intermediate technical design" [1]) and T2.2 "Existing platforms architecture extension" (See D2.2 "Intermediate KRAKEN architecture" [2]). WP3 results will be integrated with the KRAKEN marketplace developed in WP5 for the pilot's validation.

## 1.2 Structure of the document

The document comprises several sections mapped to WP3 tasks. The description of the structure and chapters followed in this report is detailed below:

- Chapter 2: SSI Prototype overview gives an introduction and a high-level view of the components implemented during the first iteration and explains how these components interact with each other.
- Chapter 3: T3.1 Verifiable Credential management tools components details the main outcomes and status of the developments performed under the umbrella of the Legal Identity Manager, which will allow obtaining a derived verifiable credential from an eIDAS Identity Provider.
- Chapter 4: T3.2 Universal Ledger Resolver describes the various components that the KRAKEN project is going to put in place to facilitate the integration with the guidelines and services provided by ESSIF.
- Chapter 5: T3.3 Edge Tools for the Decentralised Identity Solution, this chapter defines the components that will be used by the external actors of the SSI solution. The main entry developments are a mobile application (Ledger uSelf Mobile application) and a server REST API (Ledger uSelf Broker).
- Chapter 6: T3.4 SSI Wallet Management outlines the mechanism necessary for managing the SSI key material and the VCs through a Backup system, allowing to backup and synchronise the SSI information, restoring these data on another devices, in the case the end user loses their identity information.

# 2 SSI Prototype overview

The adoption of a Self-Sovereign Identity solution is one of the key pillars of the KRAKEN project. Taking into consideration the software available and the underlaying software selected by the Consortium (see deliverable D2.4 [1]), the following components have been identified:

- **Legal identity Manager (LIM):** to obtain Verifiable Credentials (VCs) derived from an eIDAS authentication. It additionally supports the creation of a (qualified) certificate for digital signature using a Verifiable ID;
- **KRAKEN Web Company Tool (KWCT):** a Web Based Graphical User Interface for the manual issuing/verification of credentials, as a support tool for operating the issuer/verifier services;
- **Ledger uSelf Broker:** to facilitate the integration of a Self-Sovereign Identity solution for the Service Providers (SPs);
- **Ledger uSelf SDK:** a library containing the basic building block for the development of an SSI mobile application;
- **Ledger uSelf Mobile application:** a basic Graphic User Interface (GUI) for the end users to interact with the SSI solution;
- **SSI Wallet Manager:** a mechanism for storing the end user information (VCs and key material) in case he/she lost the device containing their identity information and for facilitating the use of different devices by the same user.

In addition, while the adoption of ESSIF/EBSI[1] services/specifications is not in the scope of KRAKEN, to facilitate the compatibility with ESSIF some components which mimic relevant ESSIF/EBSI services [4] have been considered. The key components are included in the "Trusted Framework" (described in Section 4.1):

- **KRAKEN DID Registry**
- **KRAKEN Trusted Issuer Registry (KTIR)**
- **KRAKEN Revocation Endorsement Registry**
- **KRAKEN Trusted Schema Registry (KTSR)**

Figure 1 shows how all of these components are integrated within the system, and in the following sections of this document, they will be described in detail. Most of the components represented in Figure 1 will be provided during this first SSI release. The DID Registry and the Sidetree plus Ledger components will be released in the second iteration. For addressing DID resolution two approaches have been discussed. On the one hand, this process can be made through the use of a DID Registry which will be a service on top of a blockchain ledger or other trustable storing mechanisms (e.g., encrypted database). On the other hand, the DID resolution process can be implemented leveraging a Sidetree plus Ledger infrastructure, facilitating the use of different ledgers.

A dotted line is depicted between the LIM component and the Trusted Framework. The rationale behind this is the alignment with ESSIF. The LIM, as a legal component, is issuing legal eID VCs based on eIDAS network (relying on country identity providers authorized to issue eID). The LIM, as a trusted issuer, will be registered into the KTIR component. The KTIR contains the issuer information to facilitate the evaluation of VC trustworthiness [4]

[1] https://ec.europa.eu/cefdigital/wiki/display/EBSICOMMUNITY/EBSI+User+Community+-+public+welcome?src=spacemenu

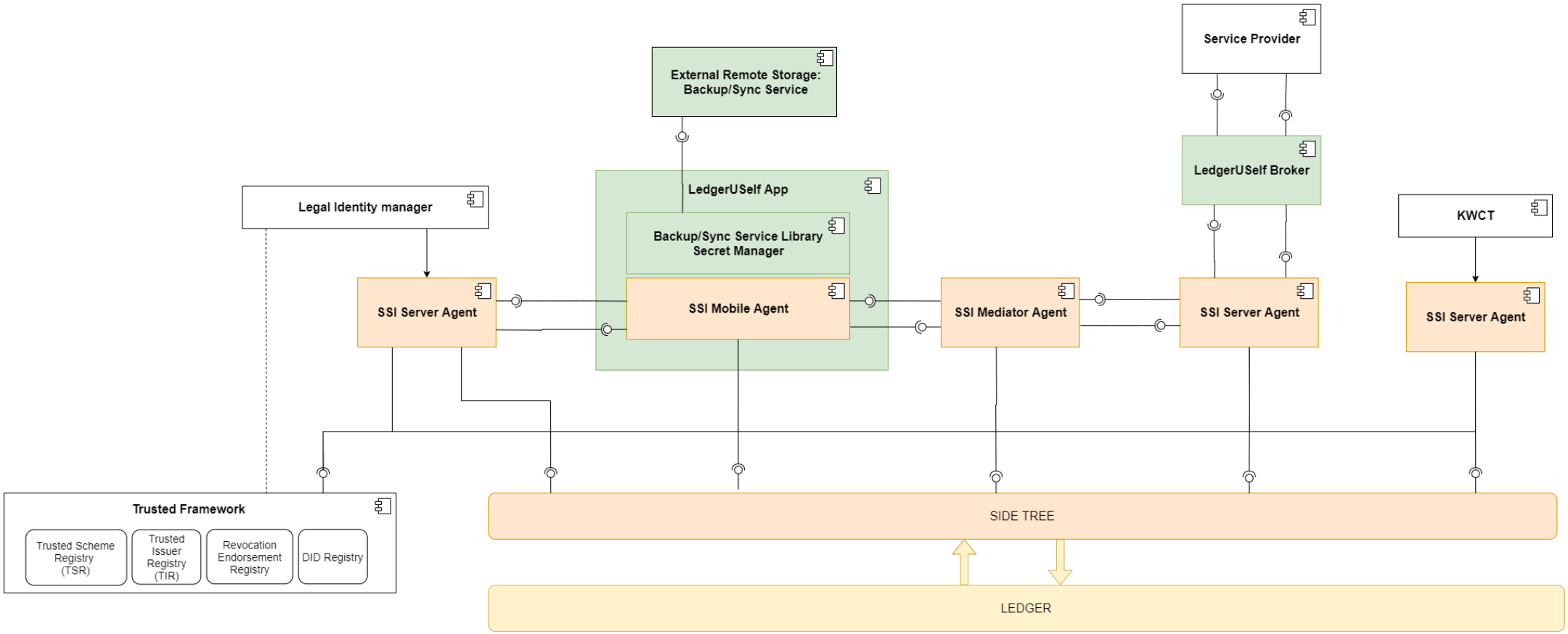


**Figure 1: SSI components overview**

To perform the operations necessary for the SSI solution adoption, the Consortium has identified three different main flows (named protocols [5] [6] [7] in the Hyperledger Aries domain) that the different components involved on this SSI solution needs to implement. The following diagrams show the details about the adaptations of these protocol's implementations by the KRAKEN project.

## 2.1 DID Exchange protocol implementation

The DID Exchange protocol is devoted to allowing the interchange of Decentralised Identifiers (DIDs) between the actors involved in an SSI operation (the sender and the receiver). This operation is the first step of all SSI operations since it is the base of the DID Comm approach taken under the underlaying framework. Further documentation about the details defined by Hyperledger Aries for this protocol can be found on the following link:

https://github.com/hyperledger/aries-rfcs/tree/master/features/0023-did-exchange

The following diagram (Figure 2) shows the implementation of the DID Exchange protocol used within the KRAKEN SSI components.

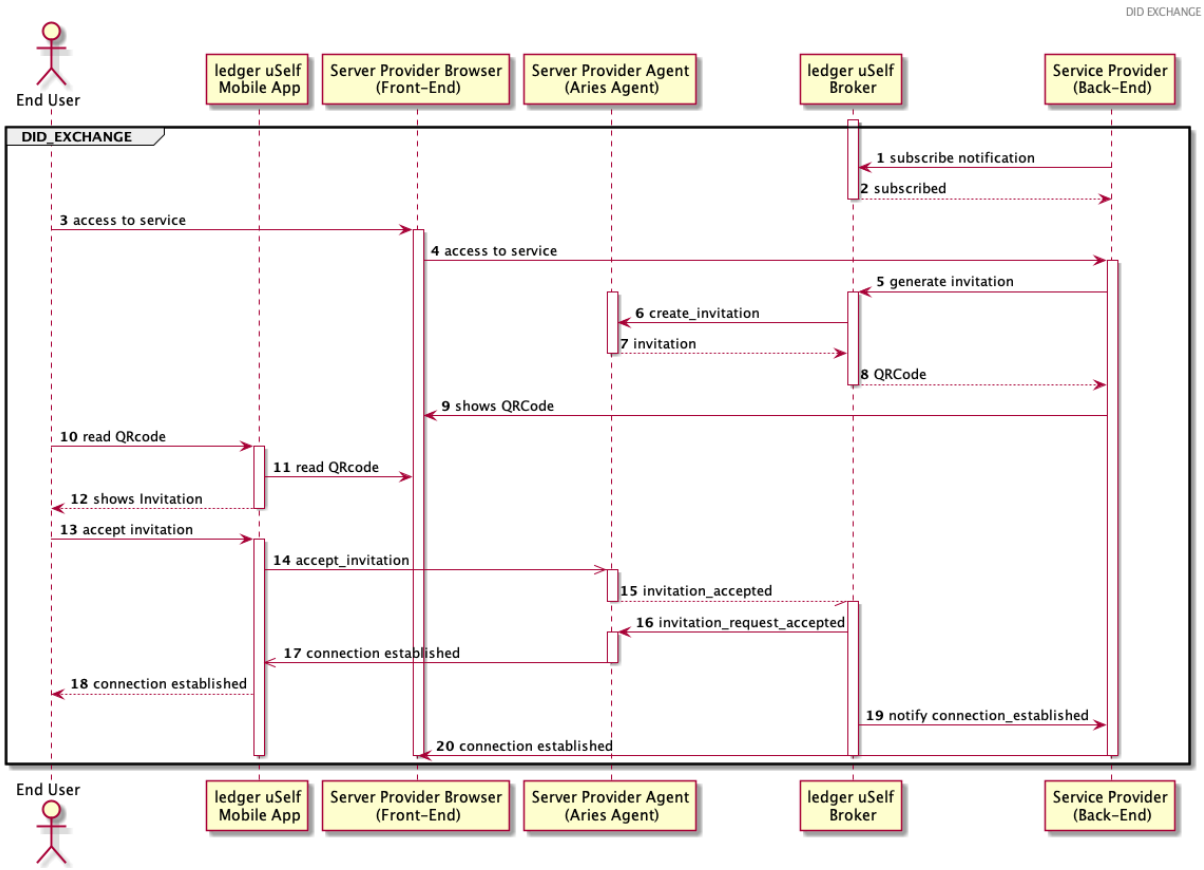


**Figure 2: DID Exchange Aries protocol implementation detail**

Figure 2 shows the details of the DID Exchange protocol, where the Service Provider represents any type of server-side stakeholder using the SSI solution. In KRAKEN the Service Provider can be the Marketplace, a University, a Company, etc. In addition, the end user represents an abstraction of the end user of the service providers, hence the end user could be a Marketplace end user, a student, or an employee, depending on the specific use case where this solution will be applied.

The most relevant components on the previous diagram are the ledger uSelf app and the ledger uSelf Broker, as they are the main interfaces to the SSI solution where the mobile application will be used directly by the end user and the Broker will help and simplify the Service Provider to adopt the SSI solution. Finally, the SSI Agent on the Server Provider represents the Edge Agent of the underlaying framework used by the KRAKEN project. It is worth mentioning that another Agent will be

implemented within the mobile app but simplifying the diagram and for the sake of the reader, it is not included.

This components description is used for this diagram but also can be applied to the following ones.

## 2.2 Issue Credential protocol implementation

The initial use case where Issue Credential protocol can be applied is in the registration to the marketplace. The end user (buyer or seller) enrols into the marketplace, filling in the information needed by the marketplace to issue a credential that will later on be used by other processes.

Further documentation about the details defined by Hyperledger Aries for this protocol can be found on the following link:

https://github.com/hyperledger/aries-rfcs/tree/master/features/0453-issue-credential-v2

The following diagram (Figure 3) shows the implementation of the Issue Credential protocol used within the KRAKEN SSI components. It is important mentioning that this step follows the execution of the DID Exchange protocol (previously described).

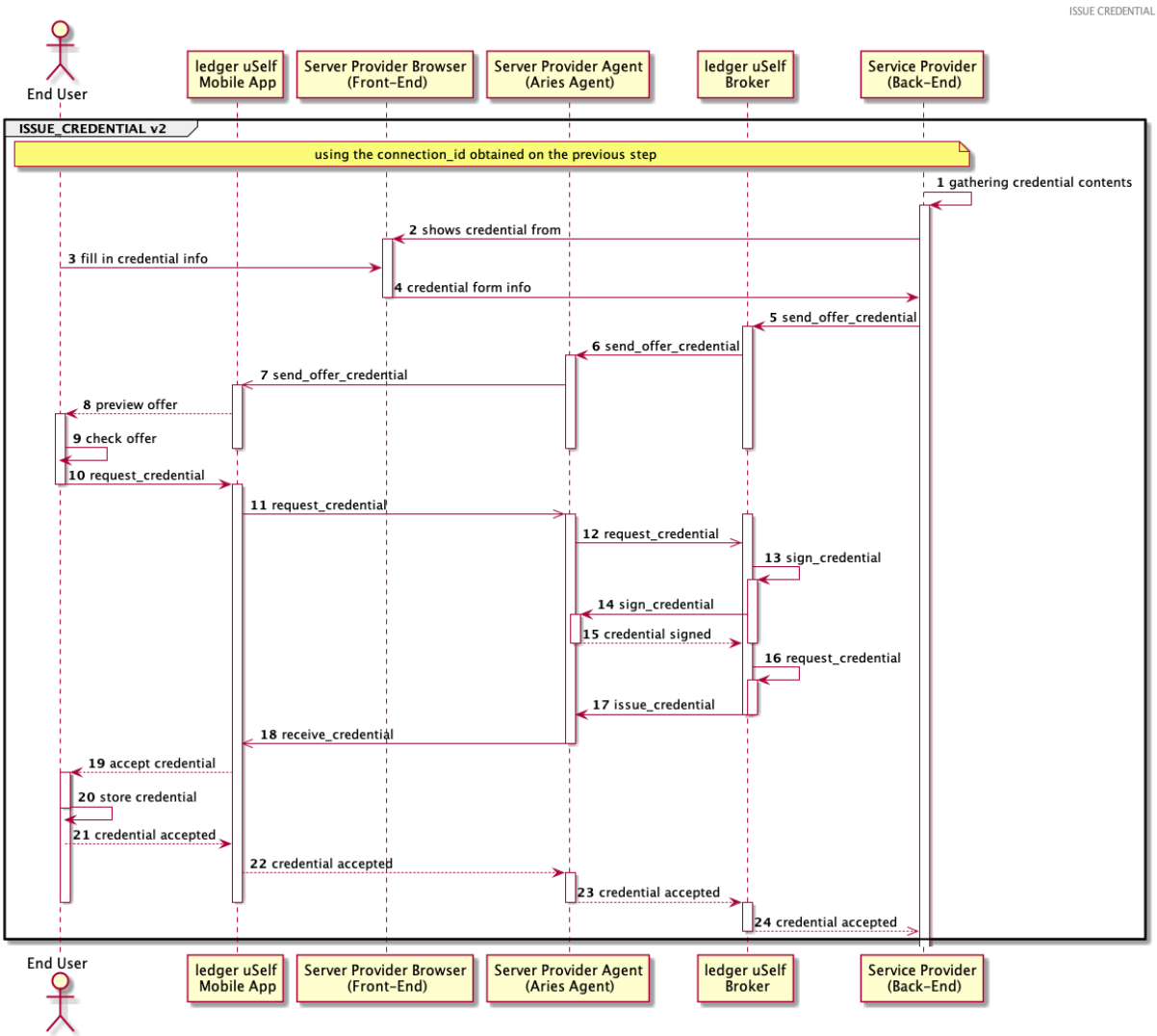


**Figure 3: Issue Credential Aries protocol implementation detail**

## 2.3 Present Proof protocol implementation

The Present Proof protocol establishes the flow between the holder and the verifier needed to deliver a proof of the verifiable credential (VC). Further documentation about the details defined by Hyperledger Aries for this protocol can be found on the following link:

https://github.com/hyperledger/aries-rfcs/tree/master/features/0454-present-proof-v2

The details about the implementation of the protocol can be seen in Figure 4.

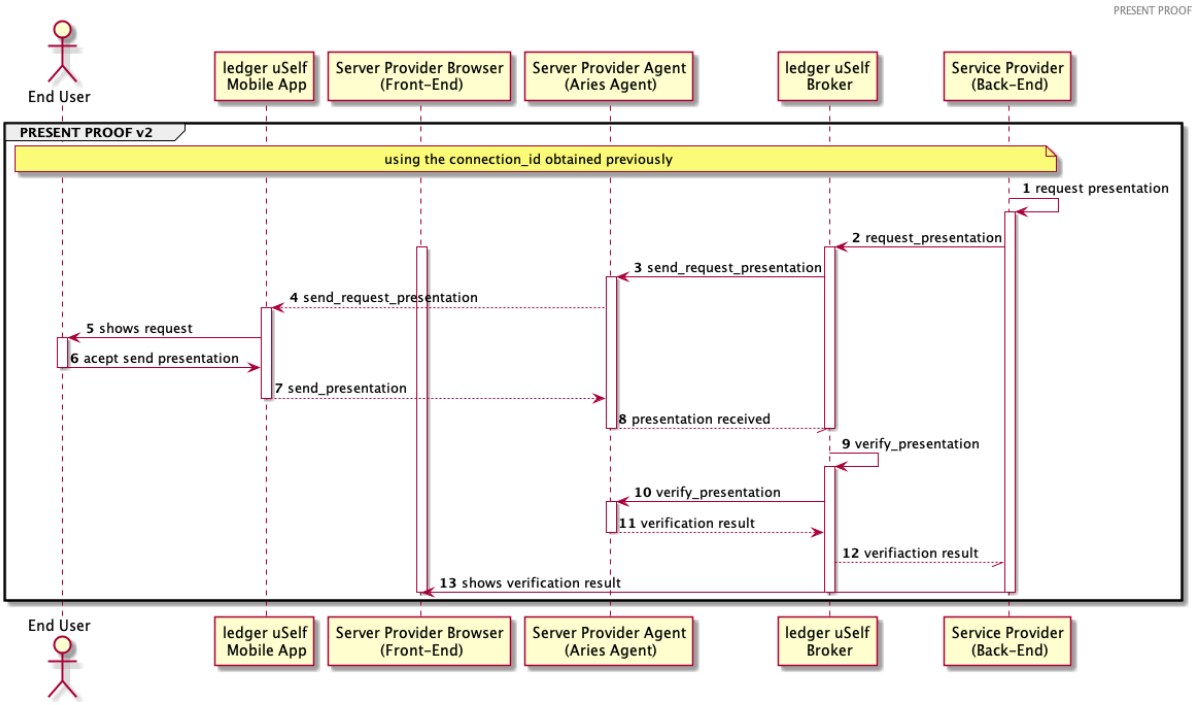


**Figure 4: Present Proof Aries protocol implementation detail**

Once the DID Exchange has been performed, a Service Provider can request the end user to provide a proof of credential, for instance, the Marketplace can request the end user to proof that they work for a company or a company can request the student a proof of the existence of their diplomas.

# 3 T3.1 Verifiable Credential management tools components

The defined WP3 objective "*To leverage the Level of Assurance (LOA) providing the necessary means to derivate credentials from trustable Authorities (eIDAS)*" is covered by the set of tools for VCs management described in the following subsections 3.1 and 3.2.

## 3.1 Legal Identity Manager (LIM)

The Legal Identity Manager (LIM) is a service offered on the web that provides to European Citizens (only natural persons, not legal persons) a verifiable ID (VI) derived from an eIDAS identity assertion. Additionally, it allows European citizens to sign documents with a signature certificate derived from their VI.

### 3.1.1 Description

The LIM acts in two different capacities:

- As an SSI issuer, for issuing a VI to a citizen. To this purpose, the LIM implements an "eIDAS service provider", which connects to the Italian eIDAS eID node. Through the network of eIDAS EU nodes, an EU citizen can authenticate using her national eID mean.
- As an SSI verifier, for allowing a user to sign an arbitrary document by presenting her VI. In this scenario, a user submits a pdf document to be signed, together with her VI. The service validates the VI and then invokes InfoCert remote signature service. This service generates a one-shot signing key pair and a one-shot/time[2] (qualified) certificate and use them to sign the pdf document with PAdES signature (see acronyms' list, the most widely adopted type of digital signature).

The LIM components are:

1. LIM_WebSite: Derived from the KWCT website software;
2. LIM_Backend: Server-side component that acts as an eIDAS SP, it manages the eIDAS authentication phase;
3. LIM_ExpressWebServer: Web server that acts as an HTTP proxy and webhook listener. It is the same software used within KWCT(Section 3.2.1);
4. LIM Database: standard data base;
5. LIM GoRestAgent: standard Aries Go REST agent v0.1.6.

Figure 5 depicts the LIM components, the interactions with the Italian eID node (eIDAS_AGID_Proxy) and InfoCert remote signature service.

[2] A one-shot/time certificate is a x509 signature certificate generated for the purposes of one single signature

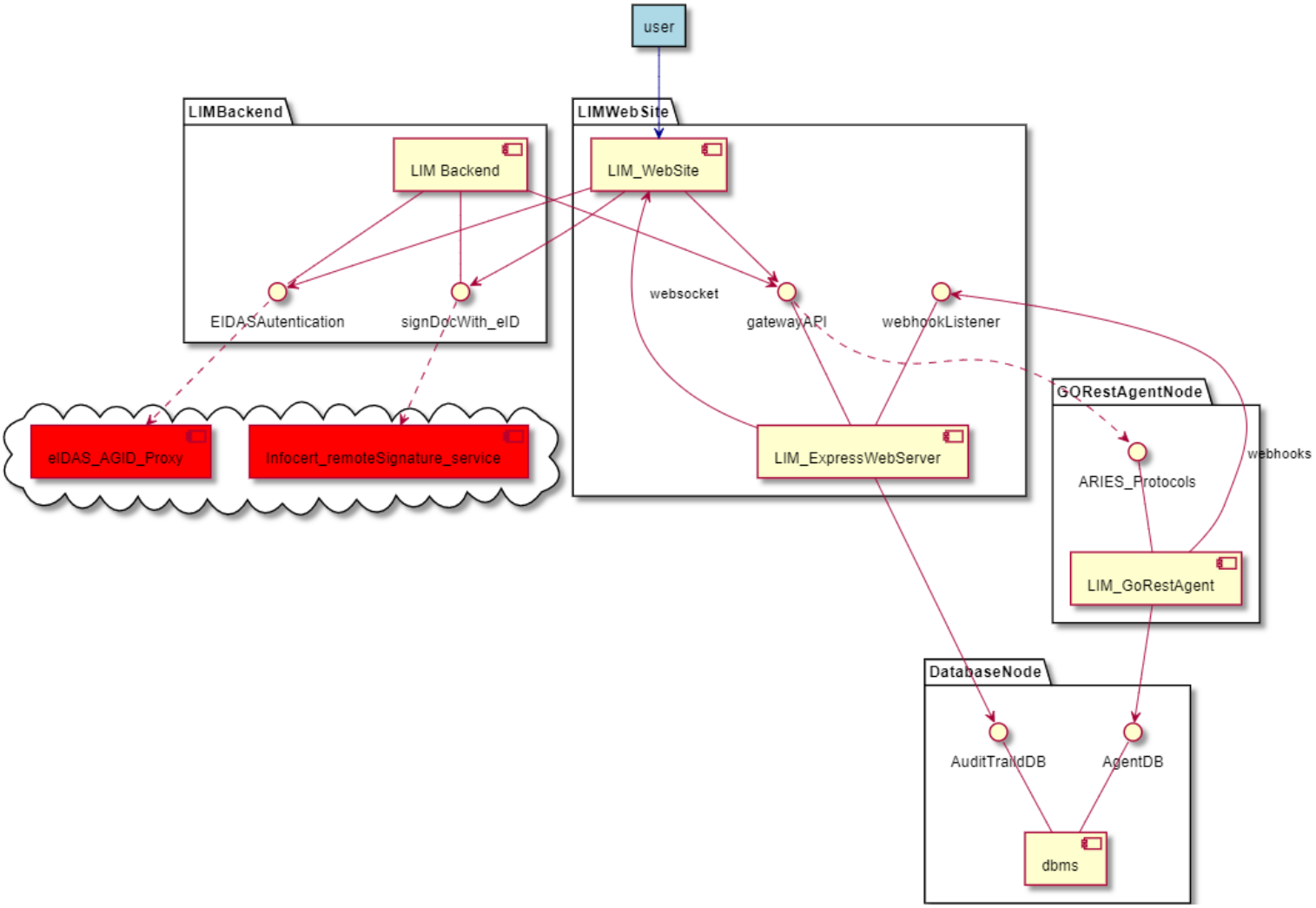


**Figure 5: LIM Components and interactions**

### 3.1.2 VI issuing sequence diagram

The VI issuing is a semi-automatic asynchronous process organized in two main phases:

1. The SSI DID exchange process that enables the LIM menu
2. The user's request of a VI in the UI of the LIM:
   - The eIDAS authentication process;
   - The issuing of the VI verifiable credential by the LIM.

The entire process is described in the sequence diagrams provided in Figure 6, Figure 7 and Figure 8. In the diagrams we used solid arrows to represent REST API calls, dotted arrows to represent webhooks, HTTP redirects or actions of the user on UI, to put in evidence what is synchronous from what is asynchronous.

Figure 6 provides the entire DID exchange main phase 1: It is a standard Aries DID Exchange protocol triggered by the user that visits the LIM landing page to use or setup a DID connection with the LIM agent.

The holder tool is the SSI tool that will contain the eID_VC after the issuing by the LIM, typically it is a mobile app.

The holder agent is the Aries Go REST agent used by the holder tool.

The Aries Go REST agent implementation includes an "AUTOACCEPT" feature, which streamlines the flow.

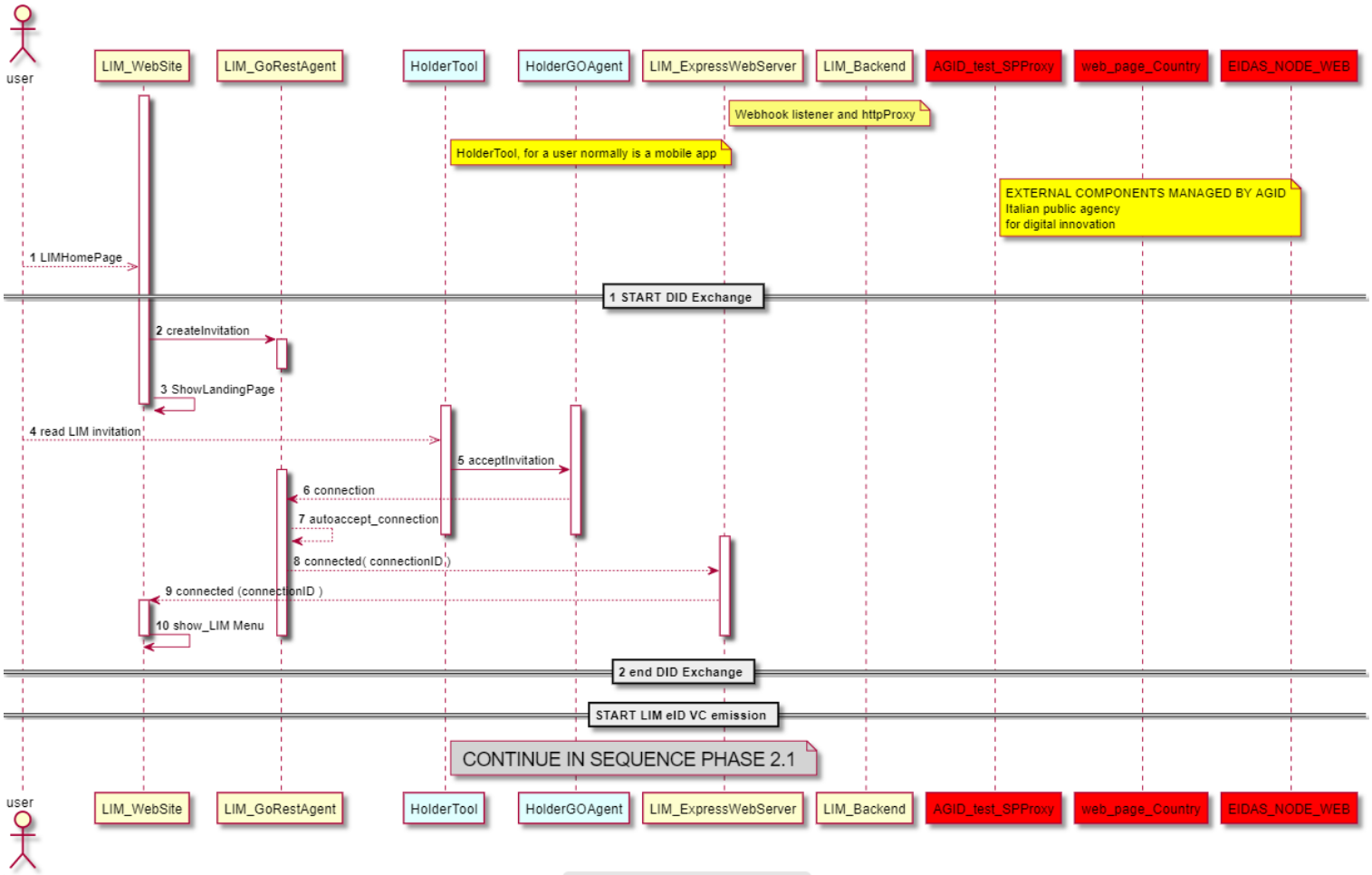


**Figure 6 DID exchange phase**

Figure 7 presents the sub-phase 2.1, which starts when the user chooses the VI functionality. The LIM acts as an eIDAS relying party, communicating with the national eIDAS node, which gives access to the European eIDAS network.

The involved eIDAS node component is the “AGID SP Proxy”.

The red components in the sequence below are external to the LIM, they have been included for completeness of description of the full authentication process, they are:

- AGID SP Proxy: it acts as a proxy of the eIDAS service providers of the different countries. The user chooses on the UI which country he uses to authenticate.
- WebPageCountry: it represents the web page of the country’s eIDAS service provider the user is redirected on after the choice of the country.
  EIDAS_NODE_WEB: it represents the eIDAS node of the chosen country.

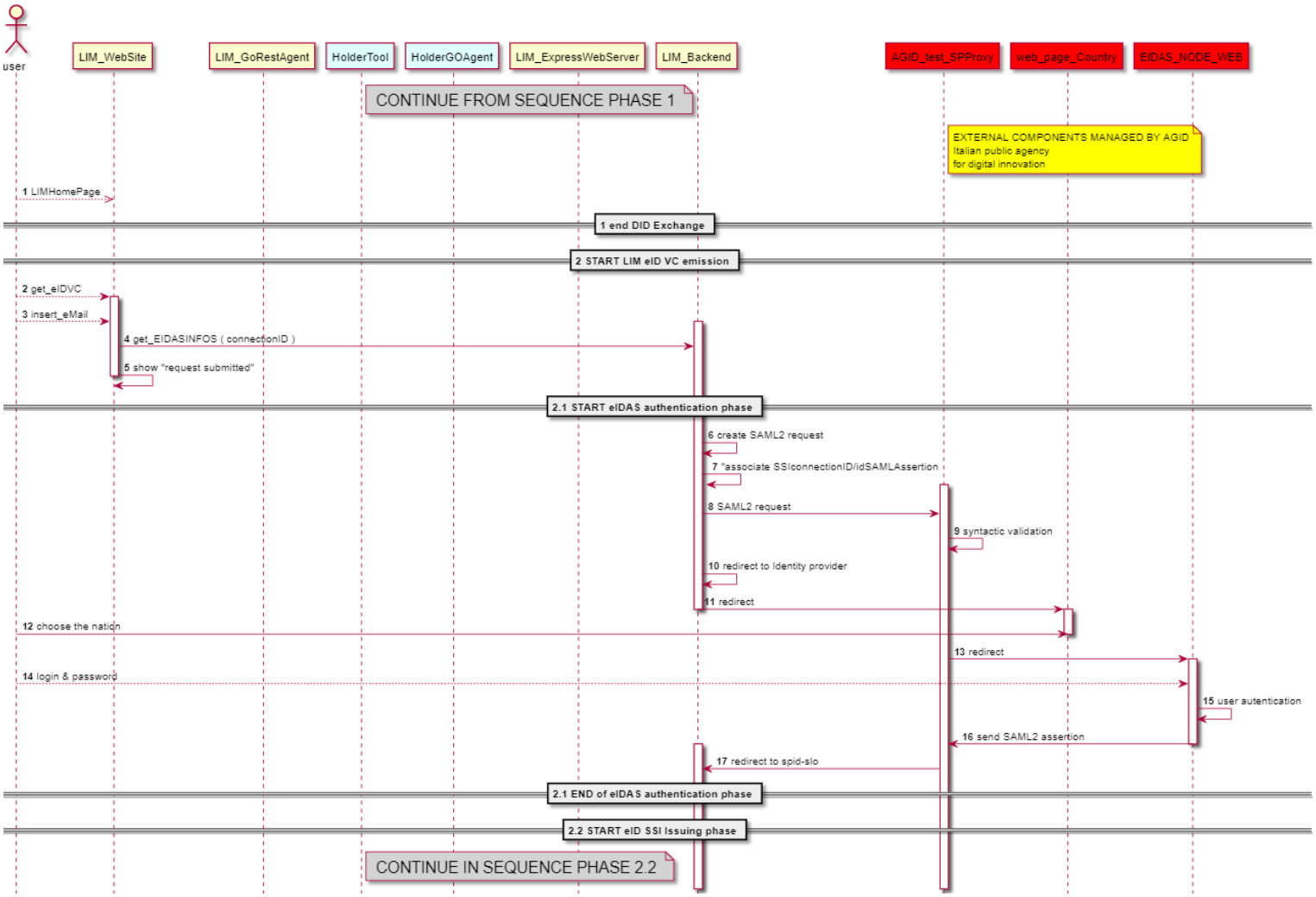


**Figure 7: eIDAS authentication phase**

Finally, Figure 8 shows the sub-phase 2.2. After the reception of the user eIDAS assertion, the LIM produces, signs and delivers the VI. The VC issuing process follows the Aries protocol Issue Credential's flow. The LIM, as an issuer, sends a proposal to the holder, then an agent to agent communication between the LIM's agent and the holder's agent takes place.

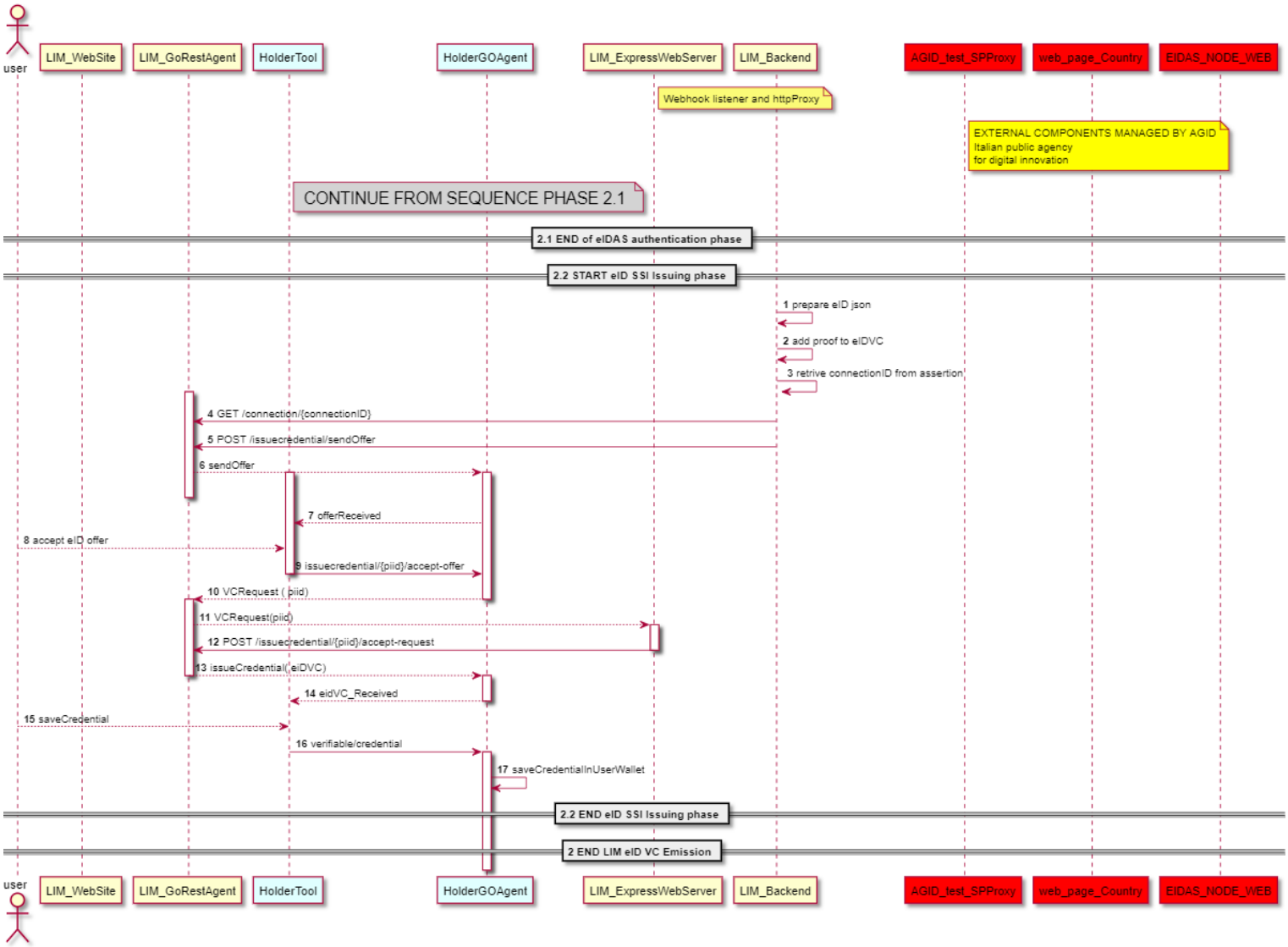


**Figure 8: VI issuing**

### 3.1.3 Interfaces

The components expose the following interfaces:

- LIM_WebSite: a user web interface.
- LIM_Backend: EIDAS_auhentication, SignDocWith_eID
- LIM_ExpressWebServer: gatewayAPI, webhookListener
- LIM_dbms: agentDB, auditTrailDB
- LIM_GoRestAgent: standard Aries_Go agent protocols used in kraken[5][6][7]

### 3.1.4 Deployment

The LIM is deployed in the following 4 docker containers, as represented in Figure 5.

1 LIM_WebSite + LIM_express_webserver;
2 LIM_backend
3 LIM Database
4 LIM GORestAgent

### 3.1.5 Source code

The source code of these components can be found in the following private KRAKEN GitHub repositories:

https://github.com/krakenh2020/LIM_SP_eidas_backend
https://github.com/krakenh2020/KWCT_LIM_httpProxy
https://github.com/krakenh2020/LIM_AngularFrontEnd

### 3.1.6 Baseline technologies and tools

The LIM backed, that acts as an eIDAS relying party, is based on the open source tool "spid-go"[3] provided by the Team per la Trasformazione Digitale - Presidenza del Consiglio dei Ministri (the Digital Transformation Team - Presidency of the Council of Ministers of the Italian Government).

## 3.2 KRAKEN Web Company Tool (KWCT)

The KWCT (former KRAKEN Web Business Logic in D2.4 [1]) is a company "SSI general purpose" tool. Via this tool company users can manually run VC issuing / VC verification processes, when automation is not possible, or to deal with special situations.

The tool includes a login functionality (for the company's employees). Also, an admin role is supported to manage user's permissions.

### 3.2.1 Description

The KWCT operates on an identity wallet associated to a company, not to a specific user.

The tool works using the Aries's Go REST agent implementation; it supports the following Aries protocols:

- DID Exchange
- Issue-Credentials
- Present Proof

Issuing and revocation of credentials are based on the underlying framework functionalities (VC Schemas and VC revocation/endorsement). Since revocation/endorsement functionalities are not available in the current Aries-Go implementation, KRAKEN will provide a custom implementation.

The tool manages three kinds of entities:

- DID connections
- VCs
- Verifiable presentations

The tool manages 3 user roles:

1 External user. The tool implements the usual SSI issuer functionalities for the company. An external user will interact with the company using her wallet, on a conventional SSI flow
2 Admin. This is a special role in the company which grants permissions to company users (role 3).
3 Company user. The typical sequence flow associated to this role is detailed in Figure 10.

Therefore, the tool user interface is organized it 3 main areas:

- Landing page: used by external users to establish the DID connection;
- Admin: includes the admin functionalities for a company super-user. The super-user grant permissions to company users to operate on different functions;

[3] https://github.com/italia/spid-go

- SSI: used by company users that operate SSI protocols (VC issuing/ VC verification). Company users need beforehand to register and authenticate to the system (see Section 3.2.2).

Figure 9 depicts a deployment diagram illustrating the KWCT nodes and their interactions with the agents, each node is deployed as a docker container.

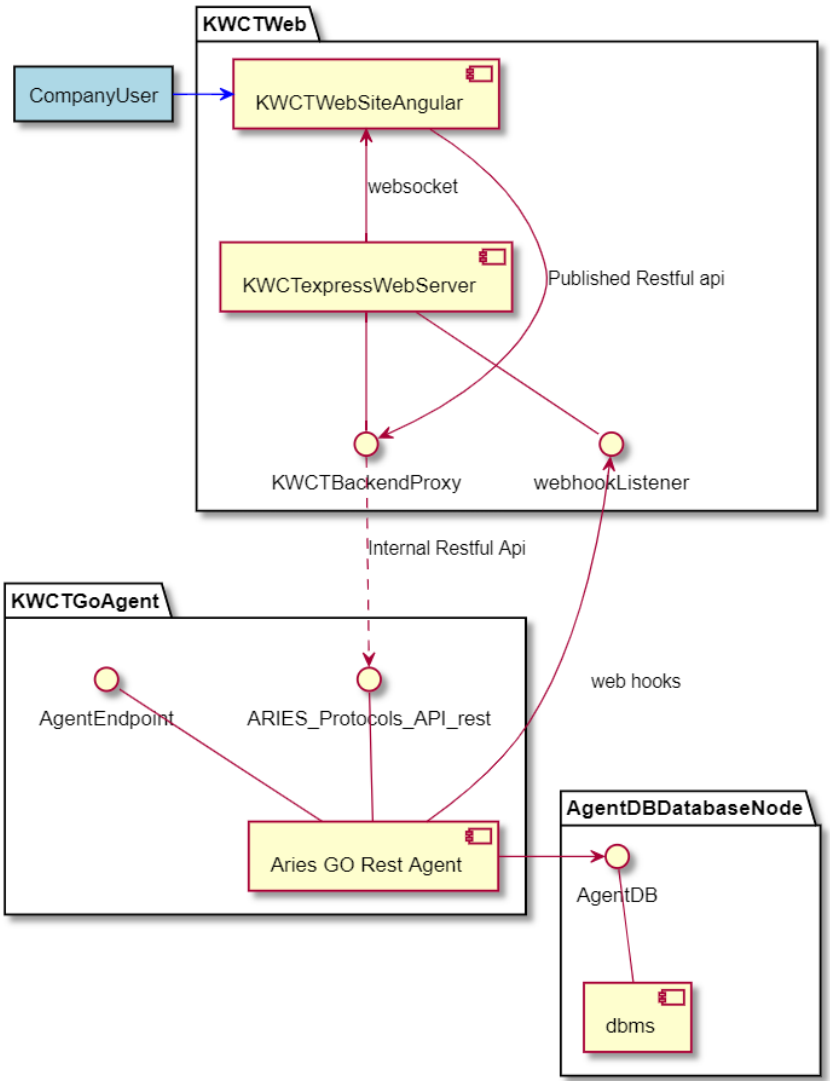


**Figure 9: KWCT node and interactions.**

The software components in Figure 9 are:

- KWC node:
  - KWCWebSite: standard angular 11 web site;
  - KWCexpresswebSite: an express framework based nodejs web server acting as httpproxy;
- AgentNode:
  - Standard Aries Go rest Agent v 0.1.6;
- AgentDBDatabaseNode:
  - Apache CouchDB instance.

### 3.2.2 Company User registration and authentication

External to the company, users or companies can connect to an instance of KWCT using a standard DID Exchange protocol. They, after a DID connection phase, can operate with the KWCT instance using their SSI tools to, for example, require a credential or present a presentation.
The company's user authentication is managed using a Java Web Token-based (JWT) user session and

based on login and password. Company user registration and authentication is described in the sequence diagram shown in Figure 10.

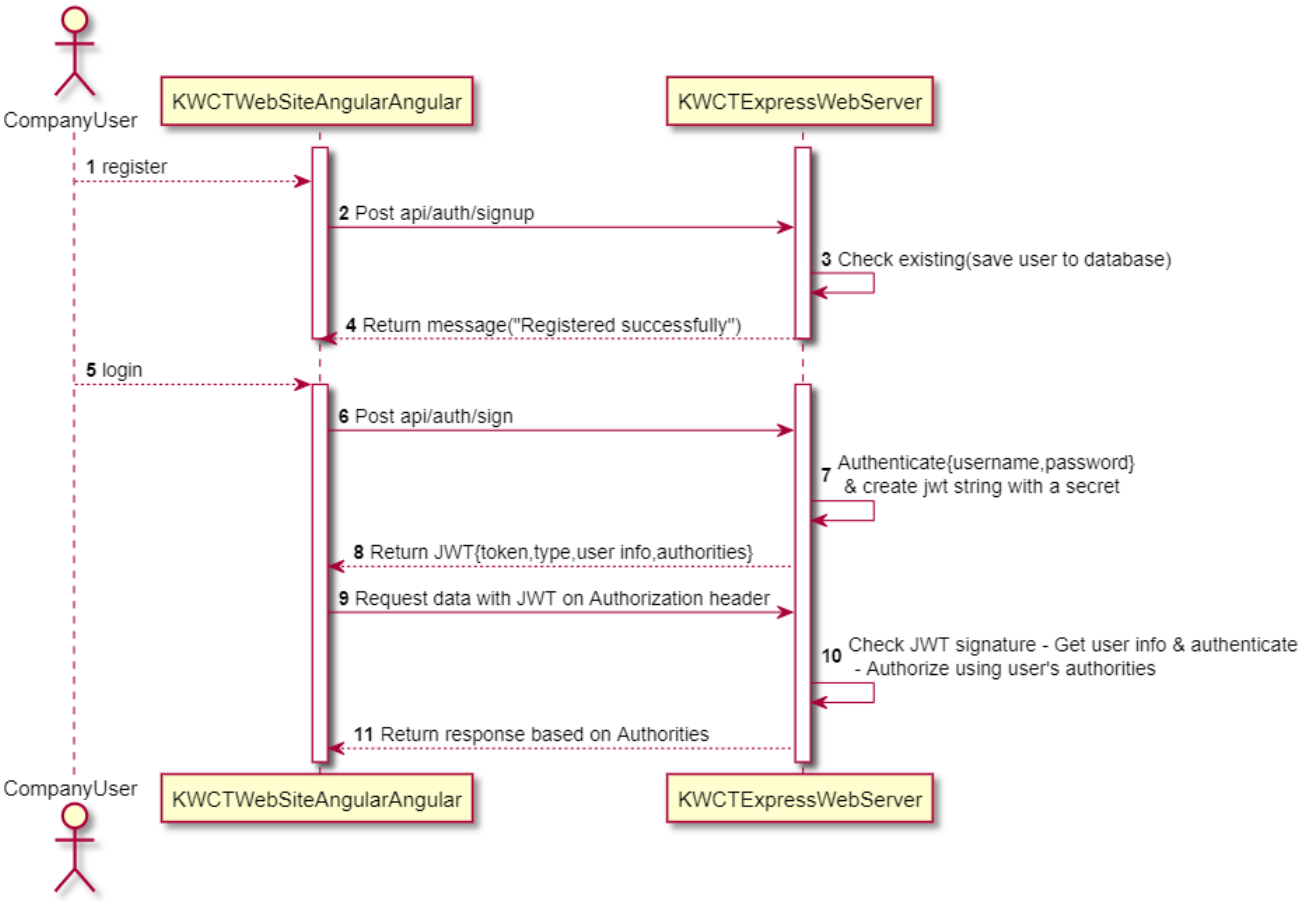


**Figure 10: User registration and authentication**

### 3.2.3 KWCT VC issuing

The VC issuing process implemented by the KWCT is a canonical issuing protocol, driven by a company operator. The sequence is not represented since it would be largely a replication of Section 3.1.2.

### 3.2.4 Interfaces

The components expose the following interfaces:

1. KWC_WebSite: a user web interface.
2. KWC_ExpressWebServer: KWCBackendProxy, webhookListener
3. KWC_dbms: LIM_dbms: agentDB, auditTrailDB
4. KWC_GoRestAgent

The tool is provided as a web interface. The REST API exposed by the KWCT based on the Aries protocols DID Exchange, Issue credentials and Present Proof, is not public and only the LIM front-end uses it.

### 3.2.5 Deployment

The KWCT is deployed in 3 docker containers as shown in Figure 9 (Section 3.2.1).

### 3.2.6 Source code

The source code of these components can be found in the following private KRAKEN GitHub repositories:

https://github.com/krakenh2020/KWCTAngularFrontEnd

https://github.com/krakenh2020/KWCT_LIM_httpProxy

### 3.2.7 Baseline technologies and tools

The KWCT derives from an open-source example provided by Hyperledger community for Cloud Agent Python (ACA-Py) and the Aries Cloud Agent Implementation, based on the Indy Ledger, written in Phyton[4].

[4] https://github.com/hyperledger/aries-acapy-controllers/tree/master/AliceFaberAcmeDemo/controllers/alice-controller

# 4 T3.2 Universal Ledger Resolver

With the aim to cover the objective of "*To abstract and decouple the KRAKEN decentralised identity solution from the different complexities and particularities of the possible ledgers selected to be used as one of the main pillars of the general solution*", a middleware layer has been developed to use several ledgers facilitating the integration of the SSI solution.

## 4.1 Trusted Framework

The design of the Trusted Framework (Figure 1) aims at the alignment with EBSI/ESSIF guidelines and components definition. It comprises the KRAKEN Trusted Issuer Registry (KTIR), the KRAKEN Trusted Schema Registry (KTSR), the KRAKEN Revocation Endorsement Registry and the KRAKEN DID Registry. The listed components are being implemented and will be functional for the SSI second release.

### 4.1.1 KRAKEN Trusted Issuer Registry (KTIR)

The cryptographic proofs included in VCs allow to verify that they are formally correct and have not been tampered with, but they cannot grant that the issuer is trustworthy. The trustworthiness of the issuer must be vouched for an external trusted entity, which acts as a root of trust.

KRAKEN Trusted Issuer Registry implements this root of trust. It is assumed that the registry is maintained by a specific entity which ca be assumed to be trusted. The registry is open for reading to any verifier.

In order not to replicate an effort that is already ongoing in ESSIF, KRAKEN has chosen to keep the design and implementation of the KTIR as simple as possible for the first release.

Efforts have been addressed to cover part of the public APIs of ESSIF v2 TIR component.

#### 4.1.1.1 Description

The KRAKEN TIR's components are:

- **KTIR_AdminWebSite:** A web tool dedicated to KRAKEN administrator users and is used to populate the TIR's_ database. The tools implement the basic security features: JWT session, http-proxy and user login & password (same as KWCT). The tool solicits TIR_Backend through TIR_private_CRUD_api.
- **KTIR_backend:** A server-side component that exposes two REST API:
  - KTIR_Private_CRUD_api: accessible only by the KTIR_AdminWebSite, it provides Create, Read, Update and Delete (CRUD) functionalities on the KTIR_database entities.
  - KTIR_public_GET_Api: published to internet, provides GET methods to be used by SSI verifiers.
- **KTIR database:** Is the container of the configuration info of the KTIR. It is composed by two entities: issuer and authorization (Figure 11). The SchemaID is refereed in Section KRAKEN Trusted Schema Registry (KTSR)

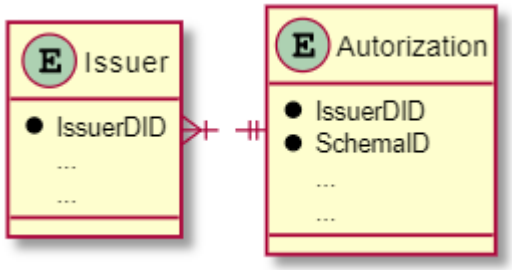


**Figure 11: KTIR database entities**

Figure 12 shows the KTIR components described in this section.

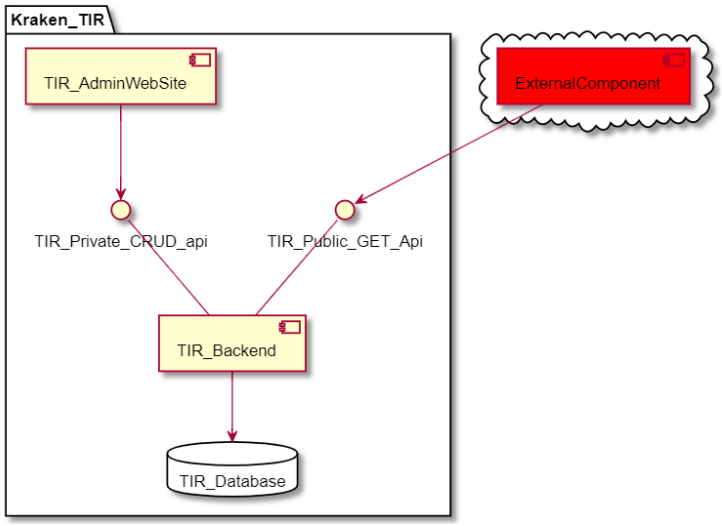


**Figure 12: KTIR components**

#### 4.1.1.2 Interfaces

Figure 13 shows the KTIR interfaces.

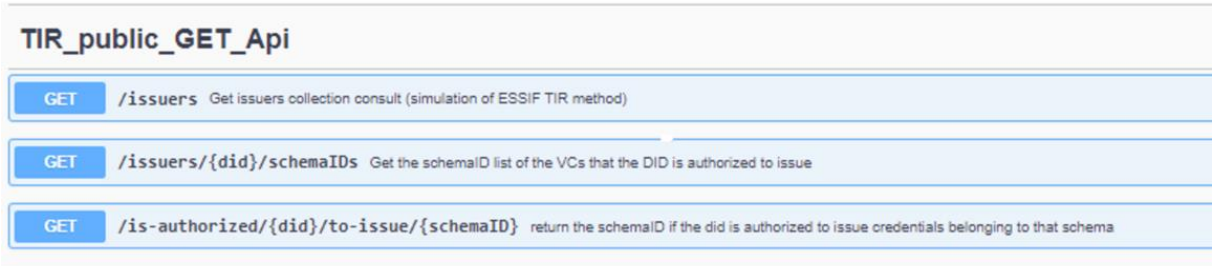


**Figure 13: KTIR interfaces**

#### 4.1.1.3 Deployment

The KTIR tool will be deployed as an only docker container.

#### 4.1.1.4 Source code

The KTIR tool is in the implementation process.

### 4.1.2 KRAKEN Trusted Schema Registry (KTSR)

Verifiable Credential's schema defines the content (set of claims) of the credential belonging to that schema.

The SSI Hyperledger Aries-Go framework doesn't yet provide this functionality, so KRAKEN decided to implement a minimal solution to support the schema definition. Since the full functionality is provided by ESSIF, it makes sense to replicate the implementation.

On this base, schemas will be represented in JSON schema draft-07 format [8] and published as a Git repository.

In the next phase the project will evaluate whether a more complete implementation will be required, also on the base of the expected public availability of ESSIF framework.

#### 4.1.2.1 Description

The KCSR will be implemented as a public Git repository that contains:

- An "index" JSON file that contains the list of the VC schemas used by the system

- One JSON file (aligned to the JSON schema draft-07 specification [8]) for every VC's schema used by a KRAKEN issuer to issue a credential used inside KRAKEN.

Figure 14 presents the structure of the index JSON file containing the list of schemas.

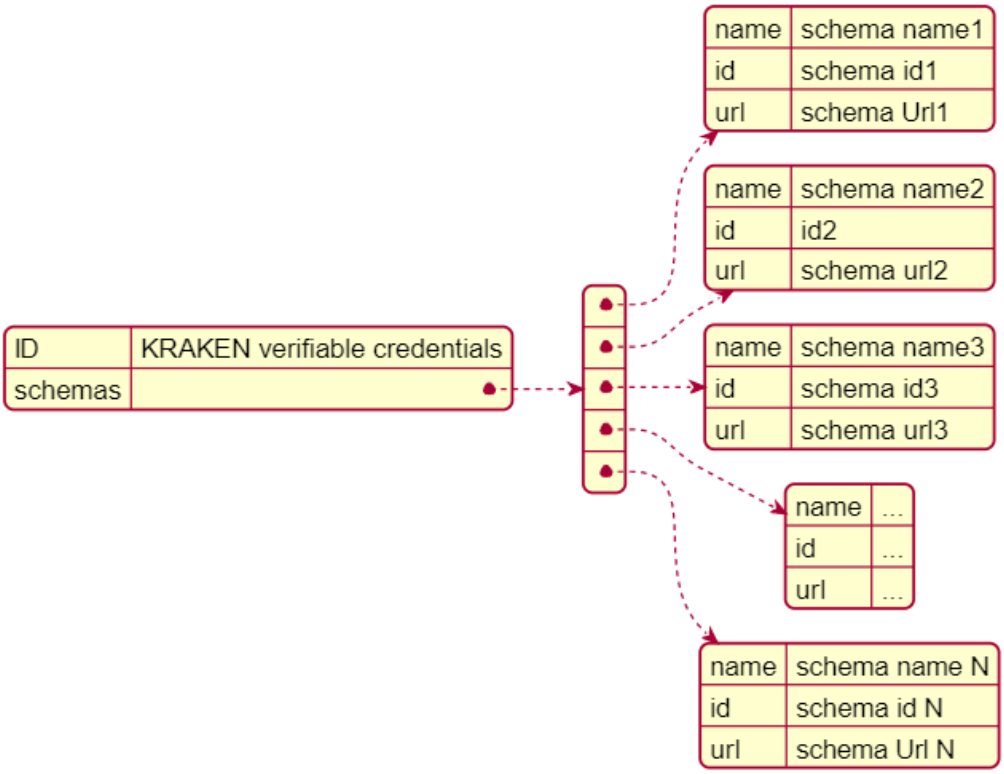


**Figure 14: JSON containing the list of the schemas managed by KRAKEN**

### 4.1.3 KRAKEN Revocation & Endorsement Registry

Figure 1 includes the Revocation Endorsement Registry component, which will be available in the next release of the SSI solution, and will be included in the deliverable D3.2 SSI solution final release due in May 2022.

### 4.1.4 KRAKEN DID Registry

As showed in Figure 1, the KRAKEN DID Registry is a service that will mimic the services provided by ESSIF to be aligned with the guidelines and specifications defined by EBSI/ESSIF until these services are publicly available. The Consortium has been analysing the information available in the ESSIF repository[5] describing the functionality of this service and has identified the minimum set of requirements necessary to comply with the specifications. Among the different operations performed by this component, the most significant interface of this service to be implemented will the one for resolving the identifier by its DID (Figure 15).

[5] https://ec.europa.eu/cefdigital/wiki/display/EBSIDOC/DID+Registry+API

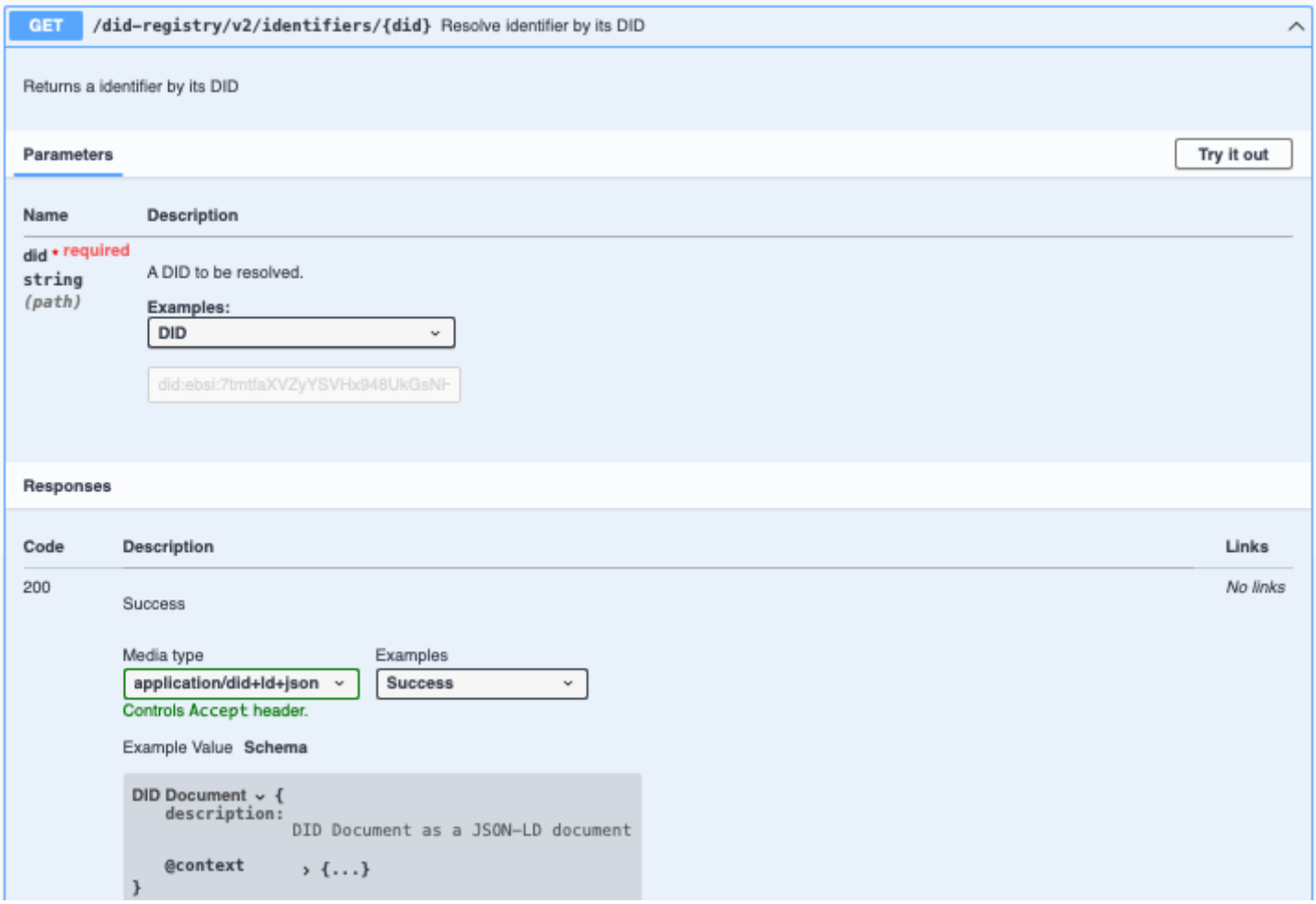


**Figure 15: KRAKEN DID Registry Interface**[6]

The main aim of this method will be to provide access to the DID document associated to a DID stored by ESSIF. An example of a DID Document responded by this service following the ESSIF specifications can be seen in the Figure 16.

6 https://api.preprod.ebsi.eu/docs/?urls.primaryName=DID%20Registry%20API#/DID%20Registry/get-did-registry-v2-identifier

```
{
   "@context":"https://www.w3.org/ns/did/v1",
   "id":"did:example:00001111",
   "authentication":[
      {
         "id":"did:example:00001111#z6MkecaLyHuYWkayBDLw5ihndj3T1m6zKTGqau3A51G7RBf3",
         "type":"Ed25519VerificationKey2018",
         "controller":"did:example:00001111",
         "publicKeyBase58":"AKJP3f7BD6W4iWEQ9jwndVTCBq8ua2Utt8EEjJ6Vxsf"
      },
      {
         "id":"did:example:00001111#_Qq0UL2Fq651Q0Fjd6TvnYE-faHiOpRlPVQcY_-tA4A",
         "type":"JsonWebKey2020",
         "controller":"did:example:00001111",
         "publicKeyJwk":{
            "kty":"OKP",
            "crv":"X25519",
            "x":"pE_mG098rdQjY3MKK2D5SUQ6ZOEW3a6Z6T7Z4SgnzCE"
         }
      }
   ],
   "service":[
      {
         "type":"NameOfService",
         "serviceEndpoint":"https://example.com/servicename/"
      },
      {
         "type":"NameOfService",
         "serviceEndpoint":"https://example.com/servicename/"
      }
   ]
}
```


**Figure 16: DID Document example[7]**

DID Registry will be available in the next release of the SSI solution and will be included in the deliverable D3.2 SSI solution final release due in May 2022.

## 4.2 Sidetree + Ledger

Figure 1 includes the Sidetree + Ledger components. The implementation of the first release of the SSI solution has been focused on the main basic elements and building blocks needed for the integration with the SP. The following options has been considered to be implemented for the Sidetree element:

- Sidetree-Fabric[8], developed by the Aries group of developers
- Sidetree-Ethereum[9]
- ION[10]

Sidetree component will be available in the next release of the SSI solution and will be included in the deliverable D3.2 SSI solution final release due in May 2022.

[7] https://api.preprod.ebsi.eu/docs/?urls.primaryName=DID%20Registry%20API#/DID%20Registry/get-did-registry-v2-identifier

[8] https://github.com/trustbloc/sidetree-fabric

[9] https://github.com/decentralized-identity/sidetree-ethereum

[10] https://github.com/decentralized-identity/ion

# 5 T3.3 Edge Tools for the Decentralised Identity Solution

One of the planned objectives for this task "*To develop the necessary edge tools to facilitate the integration of the KRAKEN SSI solution within the normal data sharing business flow*" is being addressed by implementing tools for integrating the SSI solution with the Service Providers including the marketplace. With this aim a software package has been developed providing a KRAKEN mobile app which includes the SSI functionalities that allow users to manage their own credentials and key material and at the same time facilitating the interaction with the marketplace for sharing data. Figure 17 shows the KRAKEN SSI mobile app components and their main interactions. It shows the different components needed to allow an easy integration of the Self-Sovereign Identity Solution with the different pilots and use cases planned in KRAKEN. It is worth to highlight that some of the SSI solution components will use the underlying framework (Hyperledger Aries), they are represented in light orange. The components that the Consortium will develop are coloured in green in the diagram.

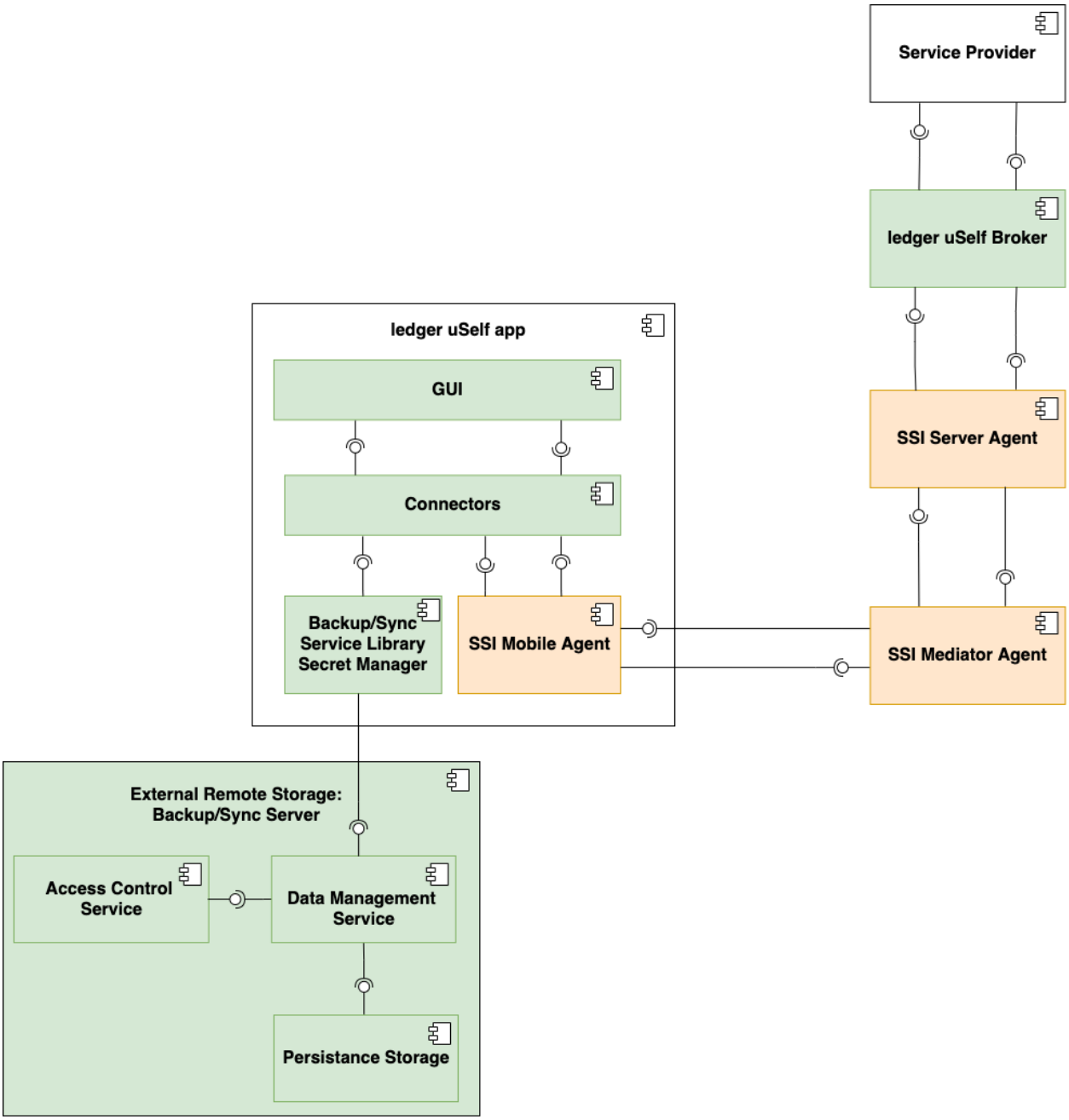

**Figure 17: KRAKEN SSI Mobile app components interaction**

As Figure 17 shows, the main effort in the KRAKEN project has been devoted to the edge components. These components are on the one hand the Ledger uSelf Mobile Application and on the other hand the Ledger uSelf Broker server. The main aim of both is facilitate to the external actors the use and the integration of an SSI solution. The following sections will explain in detail the Ledger uSelf Mobile application, Broker and SDK while the description of the Backup mechanism will be described in the following Section 6.

## 5.1 Ledger uSelf Mobile app

The Ledger uSelf Mobile App is one of the edge components designed and implemented by KRAKEN to allow the integration of an SSI solution to any domain or environment.

### 5.1.1 Description

As part of a SSI solution, the development of a mobile application is one of the key parts, as it will allow the end user to have the whole control over his/her identity data. At the same time, one of the main challenges of the development of this type of solution is to make an easy interface for the end user, making the complexity of the internal functionality transparent.

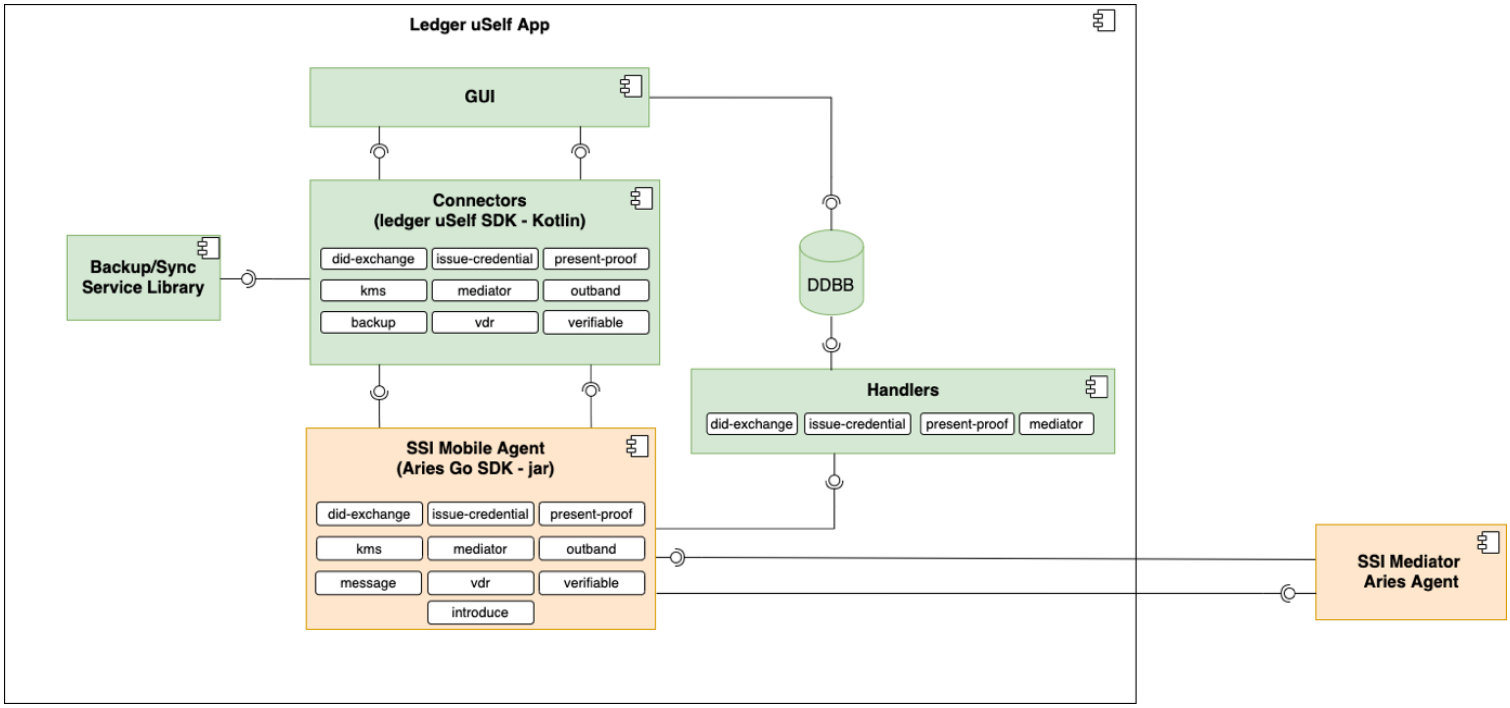


**Figure 18: KRAKEN SSI Mobile app components detail**

Considering these aspects, the development of the mobile application has been split into different layers that allows the developer to provide a GUI to the end user focusing on the usability of the final application. Figure 18 shows this layered approach used in the development of the mobile application, where the bottom part shows the libraries devoted to deal with the complexity of the SSI solution (Ledger uSelf SDK), while at the top part we can see the libraries necessary for the providing the GUI.

### 5.1.2 Interfaces

The mobile application interface has been structured into different tabs for helping on the understanding of the different aspects of the functionality but also for improving the usability of it.

The different available tabs described in the next subsections are the following:

- Settings

- Notifications
- Contacts
- Credentials
- Presentation Proof

#### 5.1.2.1 Settings

The mobile application allows the user to choose between two different configurations:

- Using an external agent: the SSI will be store in an external server;
- Using a local agent: the SSI information will be stored within the mobile.

The Figure 19 shows the screens for both options.

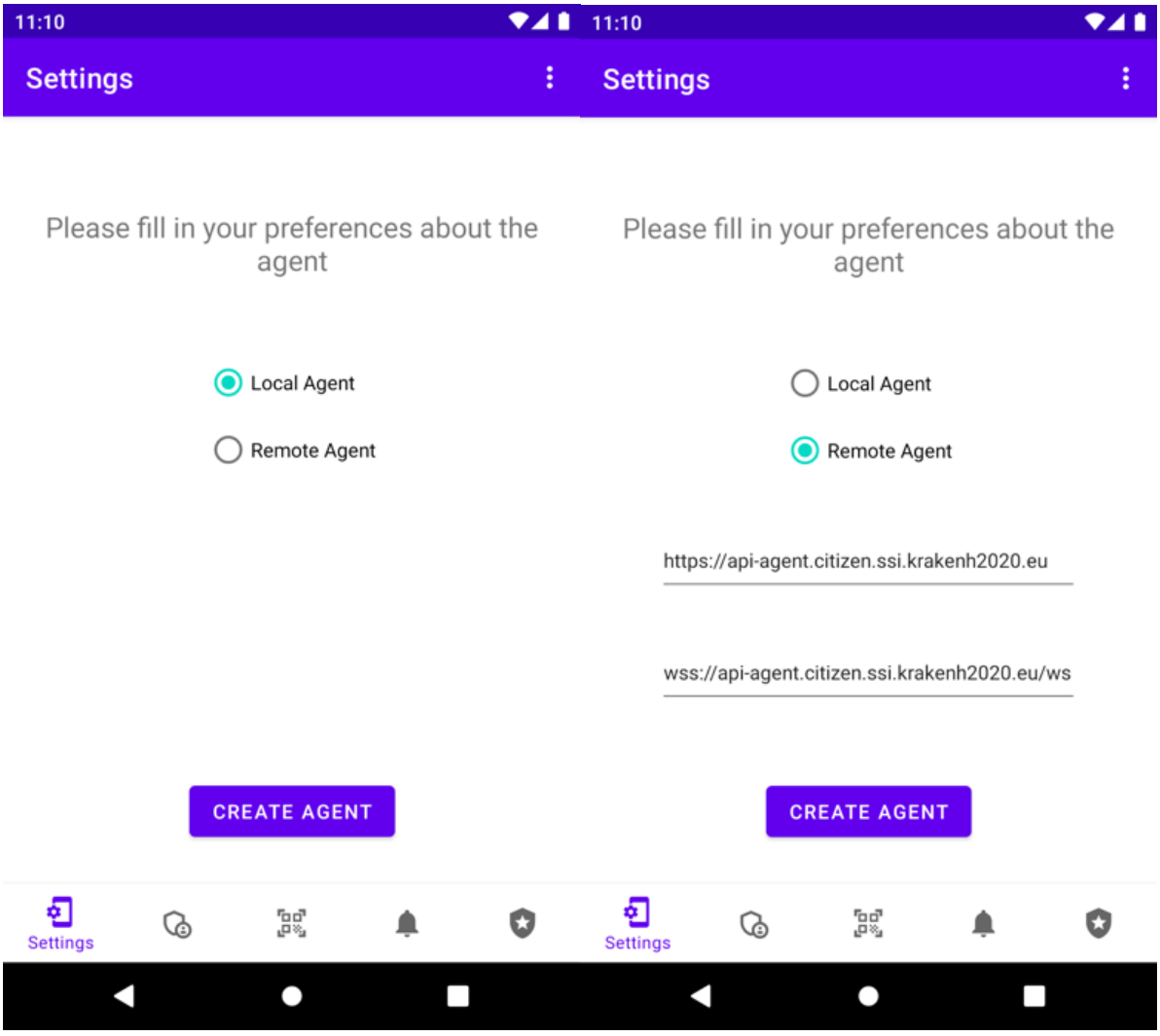


**Figure 19: Settings Tab details**

#### 5.1.2.2 Notifications

The mobile application provides a Notifications tab, where an auditory of the different operations performed by the system can be shown. There are different types of Notifications:

- Related with the contacts notifications;
- Related with the obtention of a credential (issue credential);
- Related with presenting a proof of the credential.

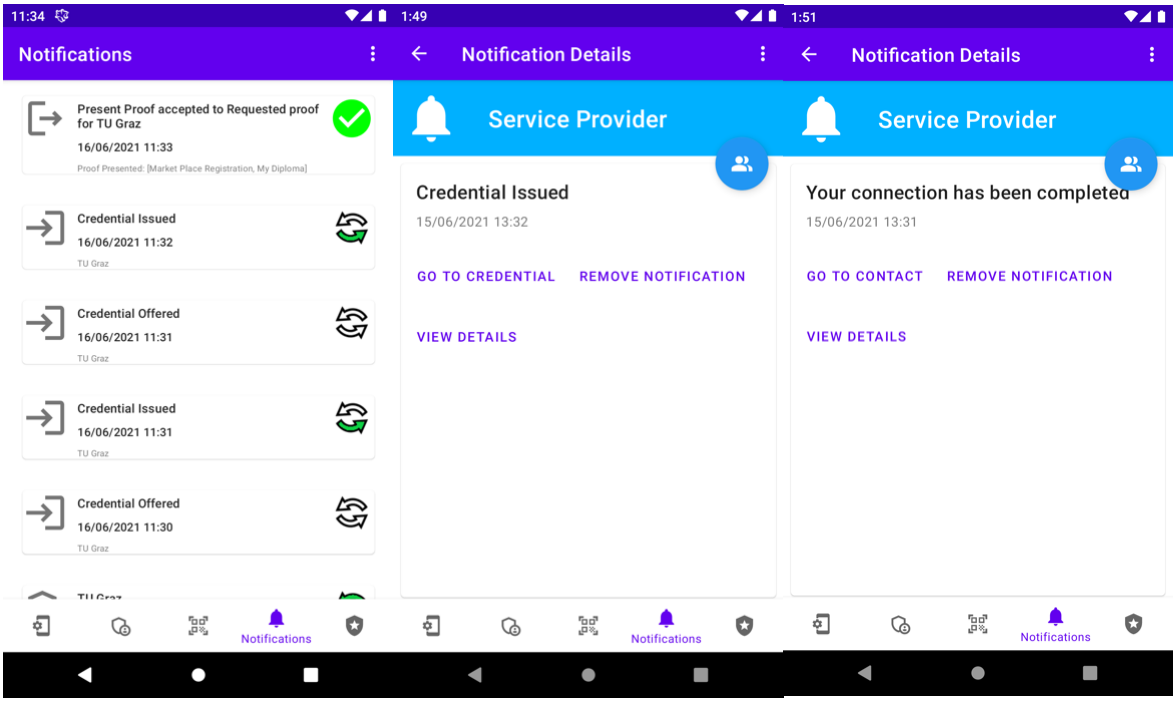


**Figure 20: Notifications Tab details**

Figure 20 shows from left to right the list of the different notifications received into the system (left), the detail of a Verifiable Credential sent to the user (middle) and how a connection was completed (right).

### 5.1.2.3 Contacts

The Contacts Tab is devoted to showing the information of the different contacts (DID Exchange) that the end user has performed. It is worth to highlight that on the different screenshots showed in Figure 21, the status of the specific contact is shown with an icon showing the arrows (in green if the connection has been completed).

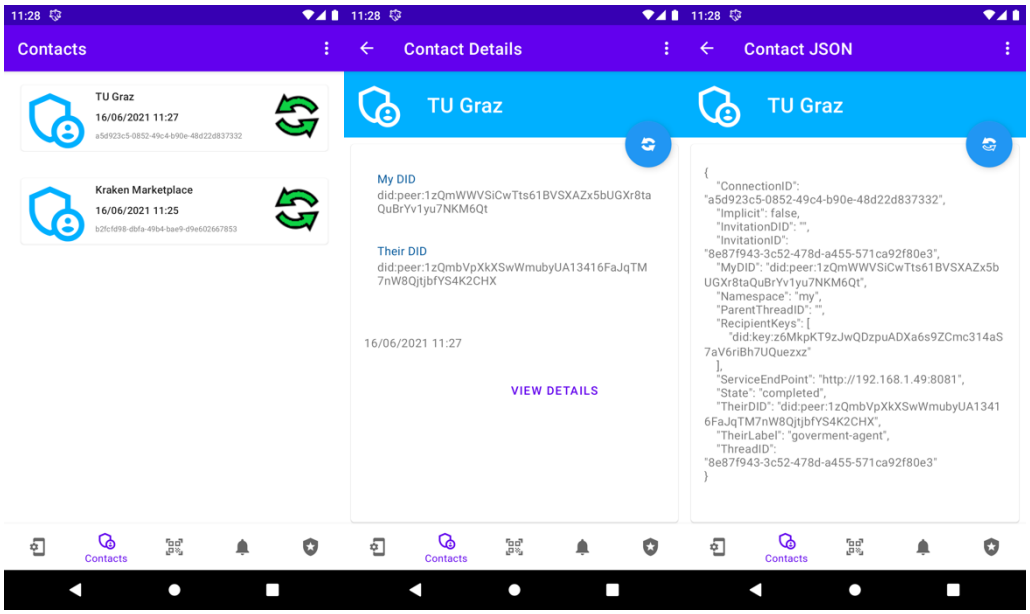


**Figure 21: Contacts Tab details**

Figure 21 shows the list of different connections the end user has made (left), the detail of the main information of this connection (middle) and the detail of the raw data associated with the connection (right).

#### 5.1.2.4 Credentials

All the information related with the obtention and the visualization of different credentials available by the end user can be accessed in this tab. The process of obtaining a verifiable credential from the issuer is performed in two different steps:

- The issuer sends an offer of the credential;
- Once the end user, the holder, accepts the offer, the issuer sends the VC and the end user stores it with a given name.

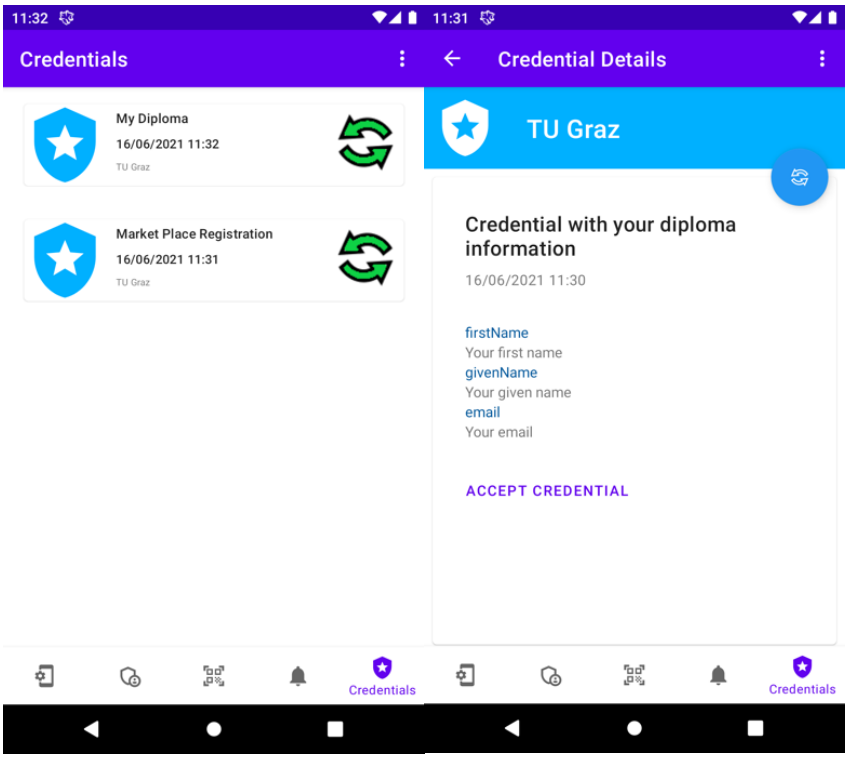


**Figure 22: Credentials Tab details**

Figure 22 shows the list of the different credentials that the end user has available (left), and how the end user sees an offer (right) sent by the issuer. The end user must accept the offer in order to complete the process and to obtain the verifiable credential.

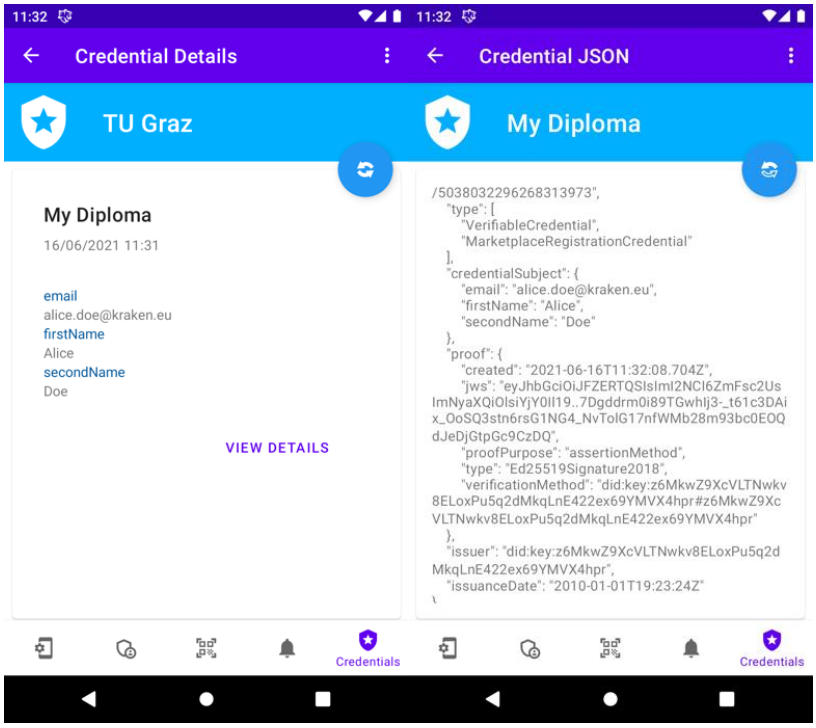


**Figure 23 Credentials Tab details**

Figure 23 shows on the left the final verifiable information and on the right the raw value of the verifiable credential, including the cryptographic material, after clicking the View details link.

### 5.1.2.5 Presentation Proof

In order to present a proof to the verifier, the mobile application will receive a notification with the petition, then the end user needs to select the verifiable credential that he/she wants to provide between the available valid verifiable credentials for this request and then send it to the verifier.

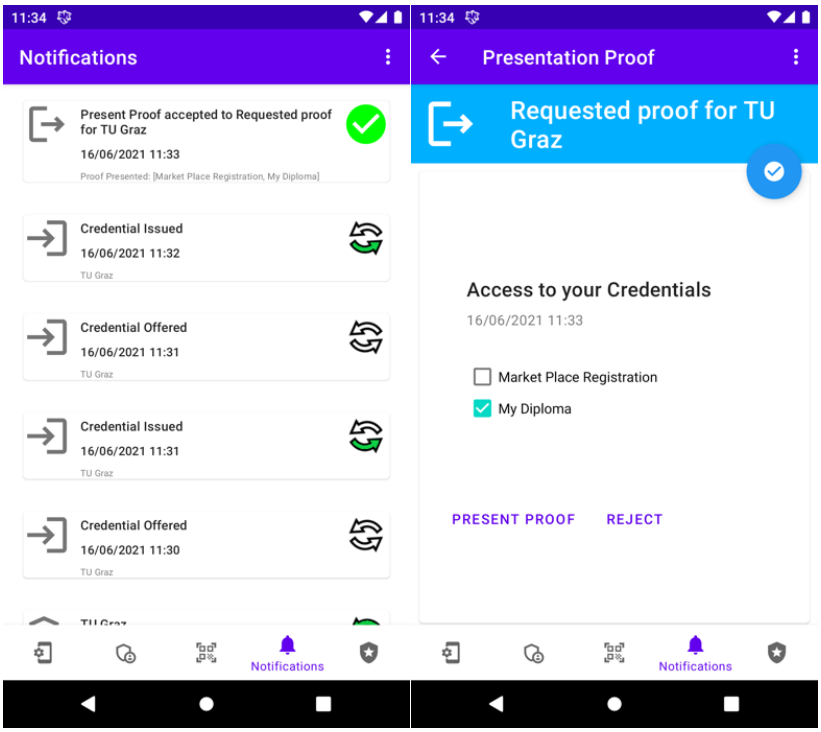


**Figure 24: Presentation Proofs details**

The Figure 24 shows on the left the notification of a proof request of a VC and on the right the VC selected to be sent to the verifier.

### 5.1.3 Deployment

In order to facilitate the use of the system among all the partners involved in the development of the SSI solution, the consortium has created a specific KRAKEN Git repository (accessed by partners), where different scripts allow the deployment of the different components needed. Further information on how to perform the deployment can be found on the following link, where all the parameters and instructions to perform the deployment are described:

https://scm.atosresearch.eu/ari/kraken/ssi-testing-environment

In addition, the following diagram (Figure 25) shows the specific details on the deployment all the associated components necessary for the use of the mobile application.

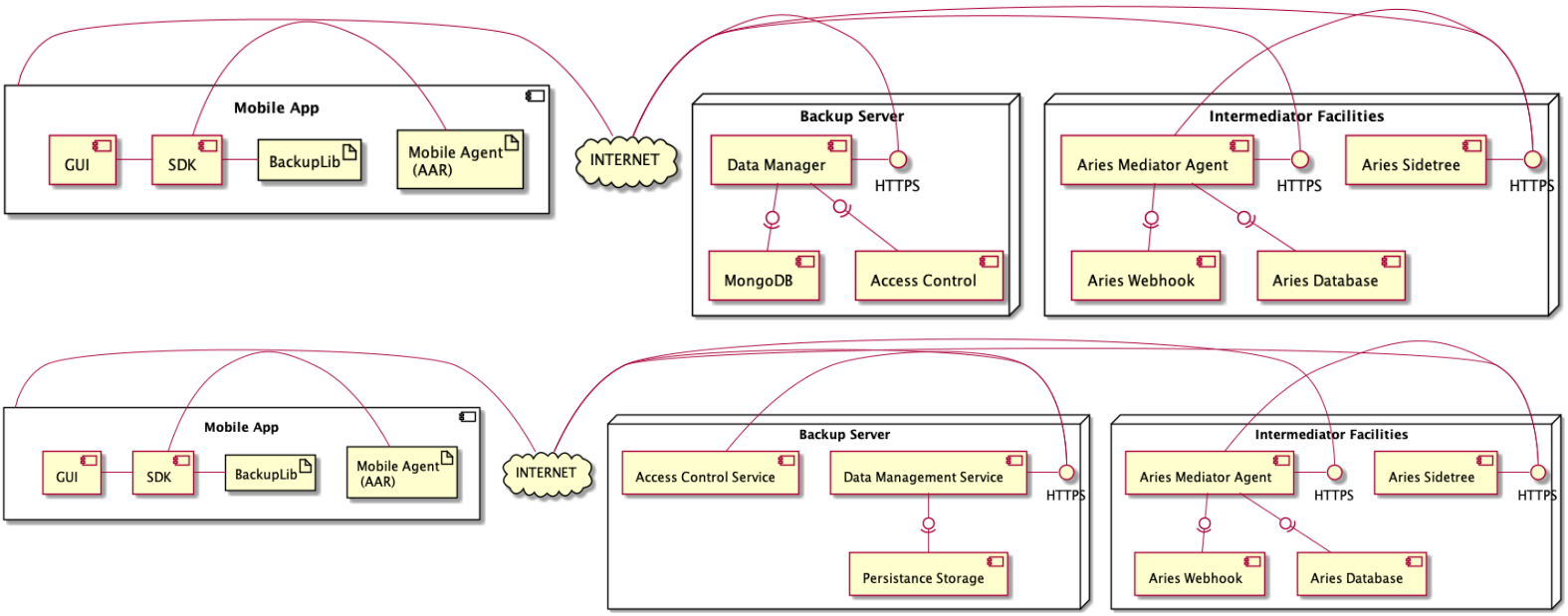


**Figure 25: KRAKEN Ledger uSelf Mobile Deployment details**

It is worth to highlight that the different docker images necessary for the deployment are hosted in a devoted repository for the project that can be found on the following link accessed by the KRAKEN partners:

https://registry.atosresearch.eu:8443/

### 5.1.4 Source code

The source code of this mobile application is available in the Git repository created for the project that can be accessed by the KRAKEN partners using this link:

https://scm.atosresearch.eu/ari/kraken/ssi-ledgeruself-mobile

The information available in the repository explains in detail the different aspects related with the use and development of this application.

## 5.2 Ledger uSelf SDK

The main aim of this component is to provide an easy access for any type of mobile application to the functionality provided by the Hyperledger Aries.

### 5.2.1 Description

As the Hyperledger Aries is developed in Go language, the way to integrate it into a mobile application is using the gomobile[11] functionality. This tool transforms the Go code into a pure C language executable software that using JNI[12] (for Android) and Objective-C[13] (for iOS) can be later invoked from the mobile app. However, this mechanism is using an abstraction method of the parameters passed to the final library. This means that all the request to this library must be transformed into byte string array and formatted into the abstract request format, which makes the whole development with this library tedious and difficult. Thereby the Ledger uSelf SDK is a wrapper over this library to make transparent to the developers the specific issues on the use of this native library. The following diagram in Figure 26 shows how the SDK is wrapping the use of the library provided by the underlaying framework (Hyperledger Aries)

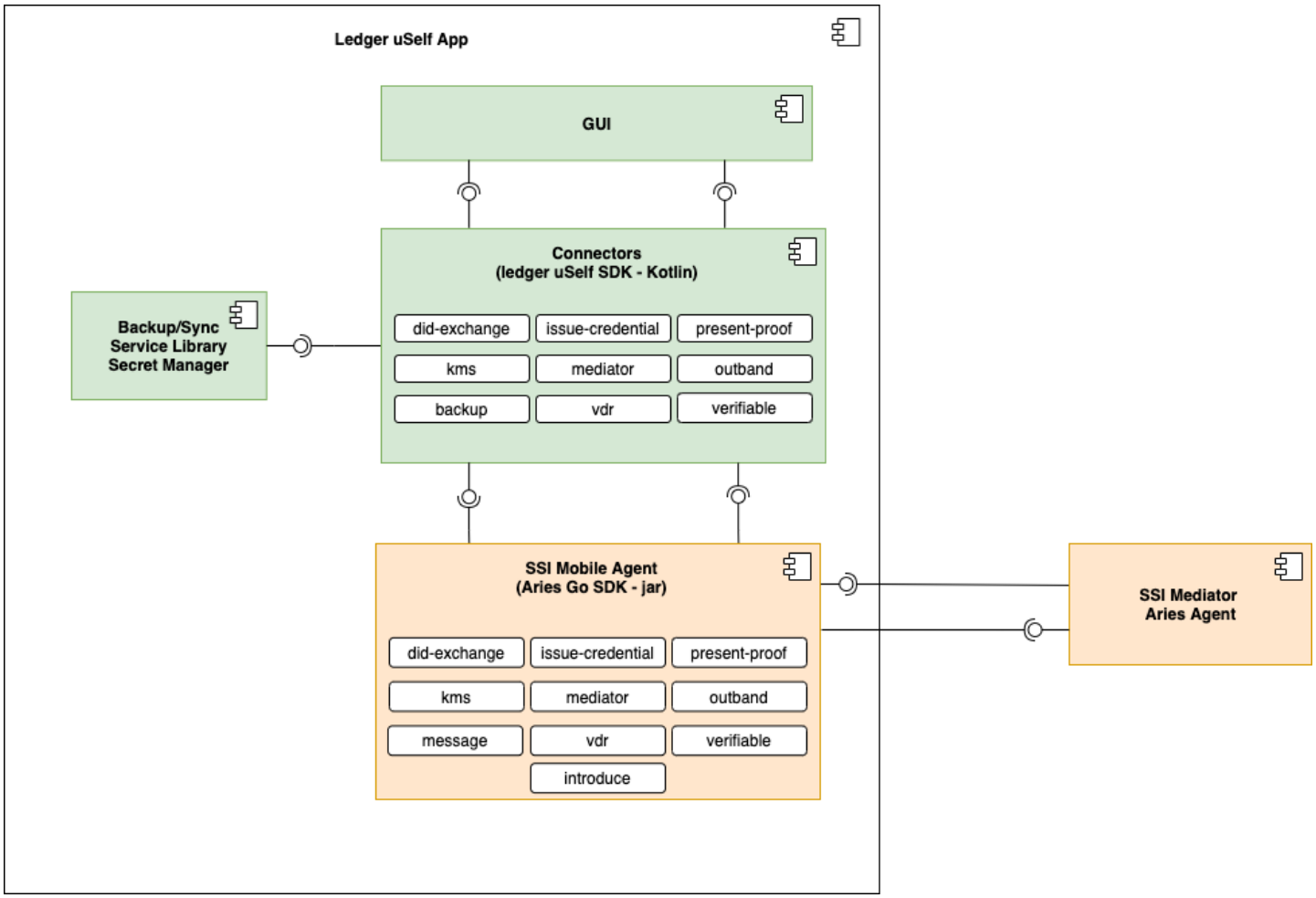


**Figure 26: KRAKEN Ledger uSelf SDK**

[11] https://github.com/golang/mobile
[12] https://es.wikipedia.org/wiki/Java_Native_Interface
[13] https://en.wikipedia.org/wiki/Objective-C

### 5.2.2 Interfaces

This interface has been designed to follow the same entry points provided by Hyperledger Aries Agent. The detailed Swagger[14] file (containing the structure and a detailed description of the API) containing the whole functionality and the parameters description can be found using this link:

https://scm.atosresearch.eu/ari/kraken/ssi-ledgeruself-sdk/-/blob/master/docs/openapi-localhost-8082.yml

In any case, for the sake of the reader, the following images shows the main entry points of the different modules developed: Figure 27 for DID-exchange interface, Figure 28 for Issue-credential interface, Figure 29 for Present-proof interface, Figure 30 for Verifiable interface, Figure 31 for VDR interface and Figure 32 for KMS interface.


**Figure 27: Ledger uSelf SDK – did-exchange interface**

[14] https://swagger.io

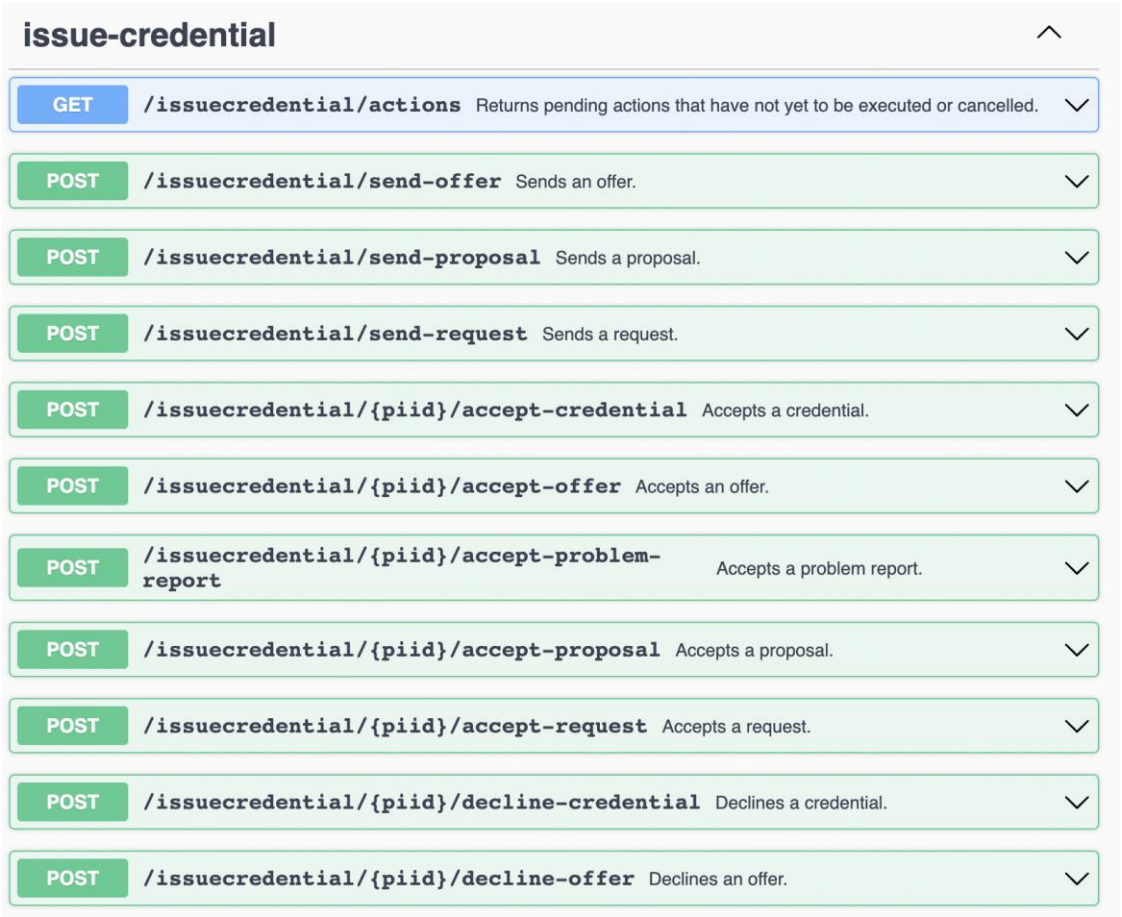


**Figure 28: Ledger uSelf SDK – issue-credential interface**

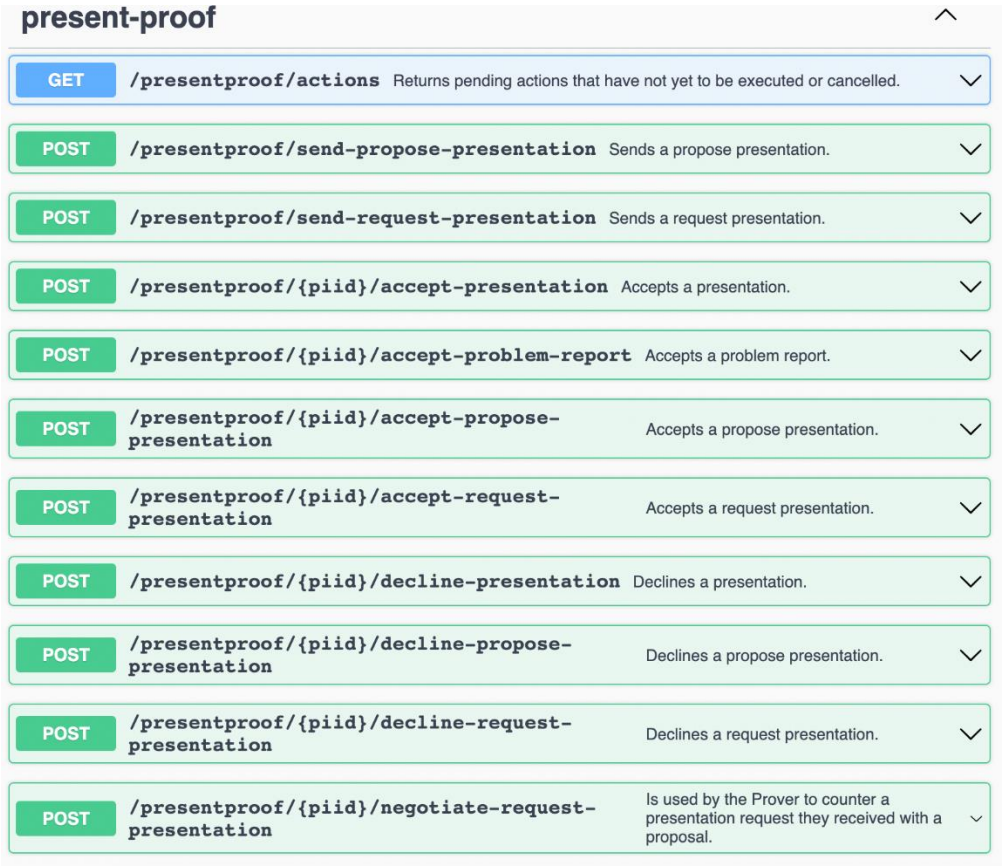

**Figure 29: Ledger uSelf SDK – present-proof interface**


**Figure 30: Ledger uSelf SDK – verifiable interface**

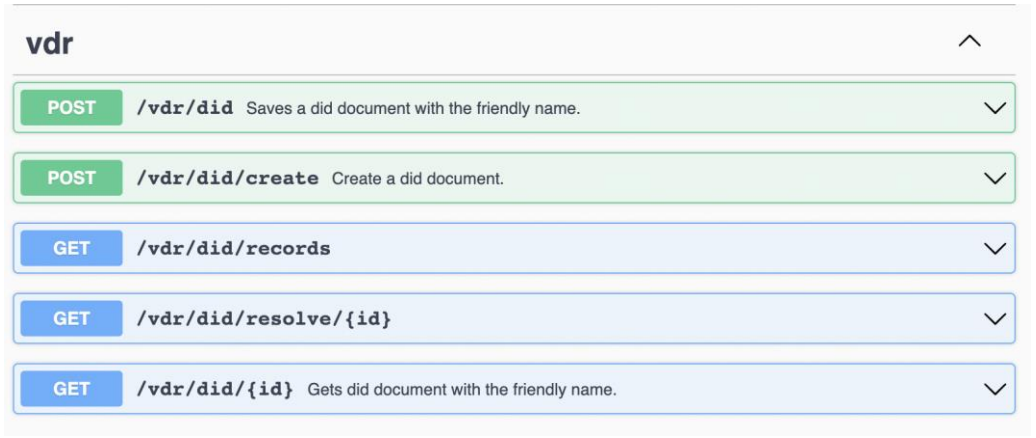


**Figure 31: Ledger uSelf SDK – VDR interface**

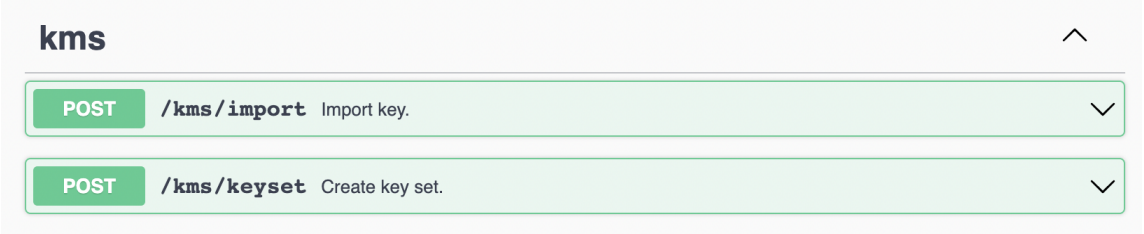


**Figure 32: Ledger uSelf SDK – KMS interface**

### 5.2.3 Deployment

For facilitating the integration of this library on any development, it has been included into the artifact repository of the KRAKEN project and it can be used just adding it as a dependency using the dependency manager selected by the final developer (Maven[15], Gradle[16], etc.). The link to the mentioned repository is:

https://registry.atosresearch.eu:8443/

### 5.2.4 Source code

The development of this library is in the Git KRAKEN project repository that can be accessed using this link:

https://scm.atosresearch.eu/ari/kraken/ssi-ledgeruself-sdk/-/blob/master/docs/openapi-localhost-8082.yml

## 5.3 Ledger uSelf Broker

The main aim of the Ledger uSelf Broker component is to facilitate the use of a Self-Sovereign solution to the SPs. This component will simplify the complex interactions necessary for exchanging the DIDs, issuing VCs or presenting proof, etc. and it will make much easier the adoption of this type of solutions for the SPs.

### 5.3.1 Description

As described in the Figure 1: SSI components overviewand Figure 3, the adoption of an SSI solution is based on complex flows with several back-and-forth requests and calls between the different actors involved. This complexity makes it very difficult for new adopters of this type of solution to integrate it within their services. Thereby, the Ledger uSelf Broker is devoted to facilitating this integration for the final Service Providers, whatever will be the domain or environment they work on.

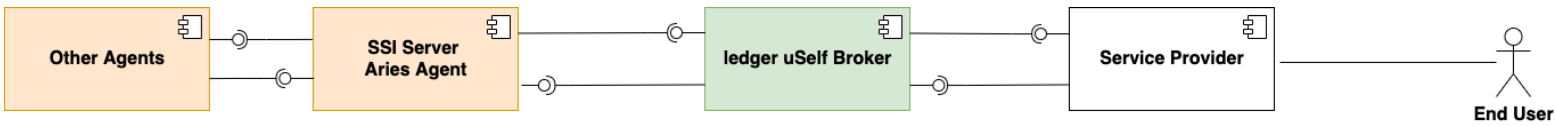


**Figure 33: Ledger uSelf Broker – components design**

The Figure 33 shows the Ledger uSelf Broker is placed between the SSI Server Agent and the SP services in order to mitigate the complexity of adopting an SSI solution. It is worth to mention that even though the Ledger uSelf provides several different entry points for maintenance and management, for the normal use of the SSI Solution only three different requests are required, one for each Hyperledger Aries protocol: DID exchange, issue credential and present proof, hence the final adoption by the SP is much easier.

### 5.3.2 Interfaces

With regards the interfaces that the Ledger uSelf Broker provides, the following images describe all entry points available: Figure 34 for broker use, Figure 35 for Issue credential and KMS, Figure 36 for Present proof and Verifiable and Figure 37 for Webhook notifications. For further details on the different parameters necessary for each request, can be obtained using the Swager file using this link:

[15] https://maven.apache.org/
[16] https://gradle.org/

https://scm.atosresearch.eu/ari/kraken/ssi-ledgeruself-broker/-/blob/master/src/main/resources/api.yml.

As mentioned before, it is worth to highlight that only the entry points describer under the BROKER tag are really necessary for the normal use of this intermediary, while the rest of the entry points are devoted for configuring and managing the server.


**Figure 34: Ledger uSelf Broker – Broker details**

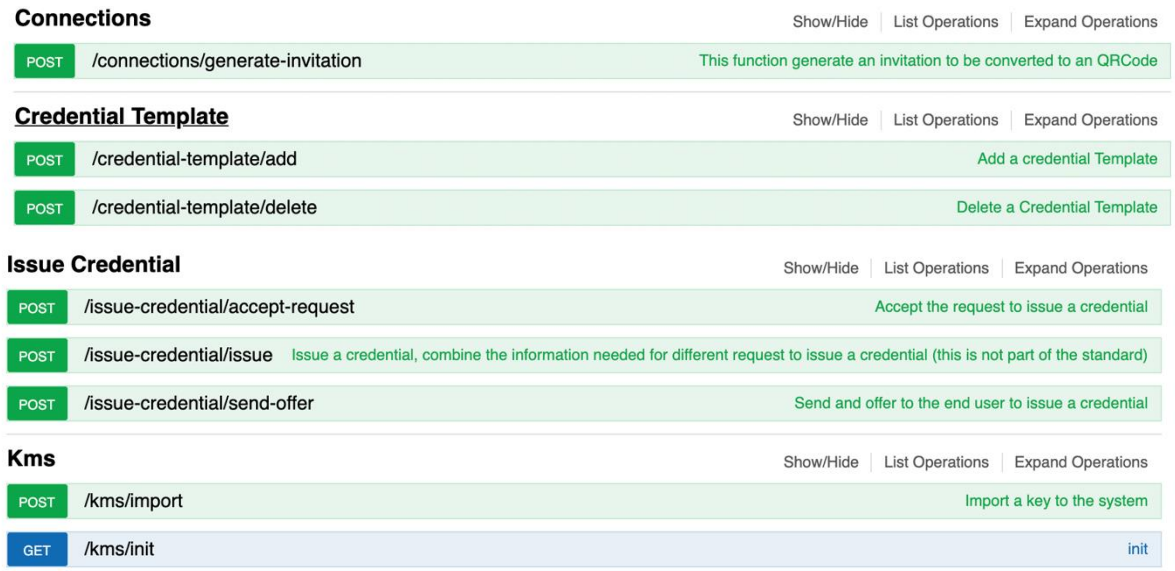


**Figure 35: Ledger uSelf Broker – Issue Credential and KMS details**


**Figure 36: Ledger uSelf Broker – Present Proof and Verifiable interface**

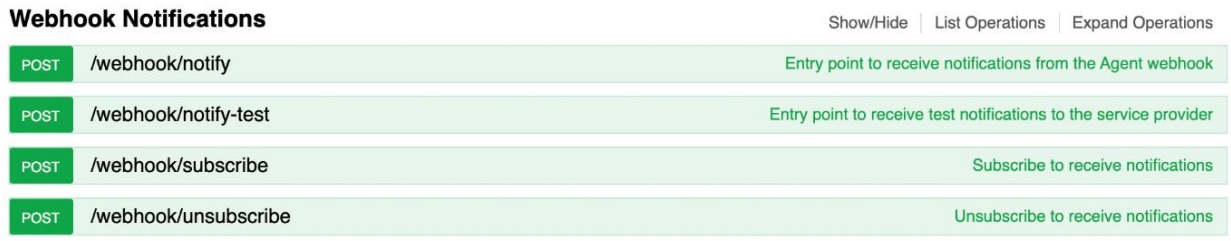


**Figure 37: Ledger uSelf Broker – Webhook Notifications details**

### 5.3.3 Deployment

As mentioned before the Consortium has created a specific Git repository where the scripts for deploying all the different components associated to the use of this tool is available. To access to it you can use this link:

https://scm.atosresearch.eu/ari/kraken/ssi-testing-environment.

In addition, the final deployment diagram necessary is shown in the following Figure 38:

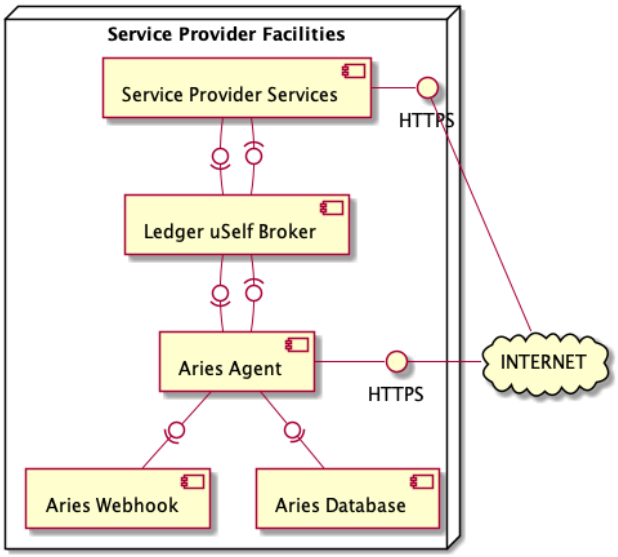


**Figure 38: KRAKEN Ledger uSelf Broker deployment details**

### 5.3.4 Source code

The source code for the Ledger uSelf Broker is available in the following KRAKEN Git repository:

https://scm.atosresearch.eu/ari/kraken/ssi-ledgeruself-broker

This repository contains all the information necessary for the configuration, deployment and use of this component.

# 6 T3.4 SSI Wallet Management

The SSI Wallet Management is carried out by two components: The Secret Manager (Backup/Sync Library) and the External Remote Storage (Backup/Synch Server). The External Remote Storage runs on a remote server and hosts encrypted backups, while the secret manager library is integrated into a mobile application where it enables access to the services of the External Remote Storage. The goal of both components is to keep the user's secrets (e.g., verifiable credentials, cryptographic material) safe (confidentiality) and available. The communication between secret manager and external remote storage takes place via the DID Exchange and the DID Auth Protocol. In order to facilitate the communication, the components implement and run relevant parts of these protocols.

Figure 39 depicts the components developed for the SSI Wallet management on both the client and the server side.

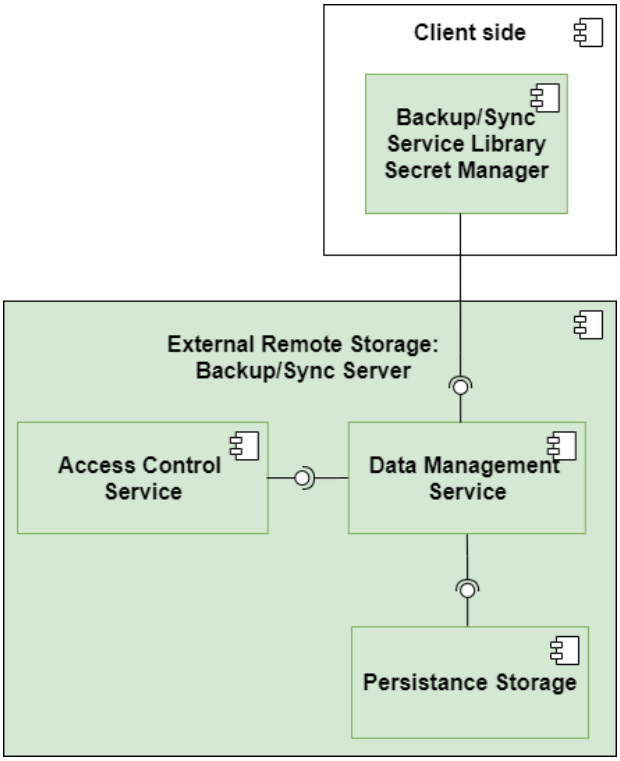


**Figure 39: SSI Wallet Management components design**

## 6.1 External Remote Storage: Back-up Synch Server

### 6.1.1 Description

The external remote storage (ERS) offers users to store and synchronize secrets across multiple devices. These secrets are encrypted from end-to-end, meaning that the data is being encrypted and decrypted on the user's device and only shared with the ERS in an encrypted form. Using proxy re-encryption [3], the ERS is able to aid the synchronization of secrets between different devices without having access to any private encryption key or to the secret.

The ERS is implemented by the following three applications:

- The **Data Management Service (DMS)** is a web application that stores, retrieves and re-encrypts secrets.
- The **Access Control Service (ACS)** handles registration and authentication of devices.

- The **Persistent Storage** stores user secrets in a persistent storage.

### 6.1.2 Interfaces

A client interacts with the External Remote Storage via the following RESTful HTTPS endpoints: The Data Manager Service endpoint at `/dms` and the Access Control Service endpoint at `/did`. The following listing enumerates and describes all operations that the services offer:

#### 6.1.2.1 Access Control Service

The Access Control Service offers a public API via the `/acs` endpoint. A client uses the operations offered on this endpoint to register and authenticate a *pairwise peer DID*[17]. The sequence diagram in Figure 40 shows how a client device registers its DID. Note that the exchange invitation can come from the ACS (in case when users register their first device) or from an already registered device (in case when users register multiple devices that should be synchronized).

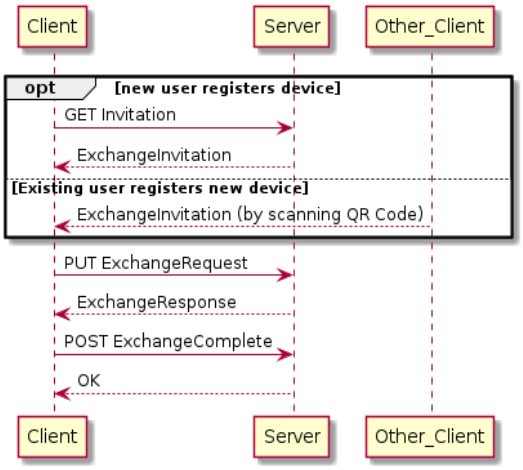


**Figure 40: Sequence Diagram of Client and External Remote Storage while performing the DID Exchange protocol**

The registration follows the DID Exchange Specification [5] using the following operations:

- GET /did/exchange: Request invitation.
  - Header: none
  - Query: none
  - Payload: none
  - Response: QR image with ExchangeInvitation Message
- GET /did/exchange/json: Request an implicit invitation, wrapped in a json message structure.
  - Header: none
  - Query: none
  - Payload: none
  - Response: JSON with QR image of ExchangeInvitation Message
- PUT /did/exchange/request: Send an exchange request.
  - Header: none
  - Query: none
  - Payload: ExchangeRequest Message
  - Response: ExchangeResponse Message
- POST /did/exchange/complete: Finalize DID exchange.
  - Header: none
  - Query: none

[17] Peer DID Method Specification, https:/identity.foundation/peer-did-method-spec/ (on 2021-06-22)

- Payload: ExchangeComplete Message
- Response: 200 (OK) or 400 (BAD_REQUEST)

After the DID of the device has been exchanged, the client proofs ownership of the DID's private key using the DID Auth protocol[18] (Figure 41).

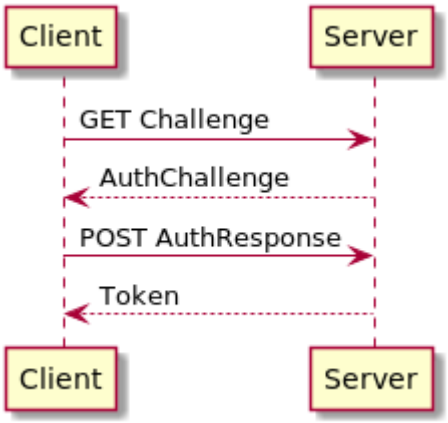


**Figure 41: Sequence Diagram of Client that proofs ownership of DID towards Server via DID Auth**

The ACS offers the following operations to carry out the DID Auth protocol:

- GET /did/auth: get authentication challenge
  - Header: did
  - Query: -
  - Payload: none
  - Response: AuthChallenge Message
- POST /did/auth: send response of authentication challenge, get DIDAuthToken
  - Header: did
  - Query: -
  - Payload: AuthResponse Message
  - Response: DIDAuth JWS Token

#### 6.1.2.2 Data Management Service

The data management service offers its functionality via a private API. Request to this API need to be authenticated via the DIDAuth JWS Token from the DID Auth Protocol. Requests need to be authenticated by putting the DIDAuthToken in the HTTP Request Authorization Header, as follows:

```
GET /dms/register-user HTTP/1.1
HOST ssi-external-remote-storage.iaik.tugraz.at
Authorization: DIDAuthToken <token>
```

The following listing describes the operations that the DMS offers:

- POST /dms/register-device: This endpoint adds the device to a group of devices that share secrets. The endpoint can only be used after the client device was invited from another device. In its response it hands out all public keys of other devices of the same group. The client is expected to issue and upload re-encryption keys for these devices.
  - Header: Authorization with DidAuthToken
  - Query: none

[18] DID Auth, https://github.com/WebOfTrustInfo/rwot6-santabarbara/blob/master/final-documents/did-auth.md (on 2021-06-22)

    - Payload: none
    - Response: UserIdGroup message, which contains the public keys of other devices.

- PUT /dms/reks: Upload a re-encryption key. Necessary for the ERS to be able to re-encrypt data from one device to another device.
    - Header: Authorization with DidAuthToken
    - Query: none
    - Payload: {rencryptionkeyB64, {ownerId, recvrId}}
    - Response: 200 (OK), 400 (BAD_REQUEST), 409 (CONFLICT)
- GET /dms/synchronize: This endpoint is similar to /dms/register-device, but it needs to be invoked by devices that are already registered. The response contains public keys of devices in the same group for which the ERS does not have a re-encryption key. Usually, these other devices joined the group after the client's device. The client is expected to issue and upload re-encryption keys for these devices.
    - Header: Authorization with DidAuthToken
    - Query: none
    - Payload: userIdGroup
    - Response: UserIdGroup message, which contains the public keys of other devices.
- GET /dms/data-root/{userId}: Get a user's folder with. This is the user's root directory for storing files and subdirectories.
    - Header: Authorization with DidAuthToken
    - Query: userId of owner's data root
    - Payload: none
    - Response: Metadata object that describes data root, including its dataId.
- GET /dms/data/{dataId}/children: List files or directories that belong to the folder identified by dataId.
    - Header: Authorization with DidAuthToken
    - Query:
        - dataId, identifies a folder in the ERS.
        - dataType: either 'file' or 'dir'. If you want to list files or folders that belong to the folder identified by dataId.
        - startIndex: optional integer. Return only children with an index greater or equal this number.
        - endIndex: optional integer. Return only children with an index less or equal this nomber.
    - Payload: none
    - Response: List of Metadata of Children
- GET /dms/data/{dataId}: Download the content of a file.
    - Header: Authorization with DidAuthToken
    - Query: dataId, identifies the file.
    - Payload: none
    - Response: metadata and reencrypted file contents, needs to be decrypted by client device.
- PUT /dms/data/{dataId}: Upload a file
    - Header: Authorization with DidAuthToken
    - Query: dataId, identifies the file.
    - Payload: Create Request Message, including metadata and fileContent.
    - Response: 201 (CREATED), 400 (BAD REQUEST), or 500 (INTERNAL SERVER ERROR)

- POST /dms/deregister-user/{userId}: Delete the user and her data.
    - Header: Authorization with DidAuthToken
    - Query: userId, identifies the user
    - Payload: none
    - Response: 200 (OK), 404 (NOT FOUND), 400 (BAD REQUEST)

### 6.1.3 Deployment

The ERS will be deployed at a host in the TU Graz network and will be reachable through the following URL:

https://kraken-ssi-external-remote-storage.iaik.tugraz.at/

Both the Access Control Service and the Data Management Service are web applications that will be deployed in a tomcat container app. While these apps are reachable through the public network, the persistent storage which runs as MongoDB instance on the same host is not reachable through the public network.

### 6.1.4 Source code

The source code of the External Remote Storage is available on GitHub at:

https://github.com/krakenh2020/ssi-backup-library/tree/master/ssi-backup-library-server

## 6.2 Secret Manager: Back-up Synch Service Library

### 6.2.1 Description

The Secret Manager is a library that connects a client application with the External Remote Storage. The tasks of the Secret Manager can be divided into two categories: On the one hand, the Secret Manager handles the interactions with the External Remote Storage, such as registering, authenticating towards the Access Control Service, as well as storing and retrieving secrets from and to the Data Management Service. On the other hand, the secret manager performs all client site cryptographic operations (encrypting, de-crypting secrets, issuing re-encryption keys) and manages the associated key material. The secret manager offers its functionality via an easy-to-use API that relieves a client application from dealing with the complexity of managing key material and interacting with the ERS.

### 6.2.2 Interfaces

The API that the Secret Manager exposes towards a client application consists of the following operations:

- void init()
    - Initialize components in library for usage.
    - If not available: generate re-encryption key pair.
    - If not available: generate client backup authn key pair.
    - If not available: create DID (from backup authn key pair) for communication with External Remote Storage.
- Result registerDevice (ExchangeInvitation invitation)
    - Register this device at the ERS.
    - The invitation Parameter is a JSON Web Signature Token that the client application obtains from the ERS or from another device.
    - The operation returns a Result which tells if the registration succeeded, and, in case it failed, a message explaining the reason.

- void synchronizeKeys()
    - Ensures that the ERS receives the key material for re-encrypting data between devices.
    - Retrieve public re-encryption keys from other devices in the same group.
    - Create re-encryption keys for each public re-encryption key.
    - Upload re-encryption keys to server.
- Result backupVerifiableCredential(String vc)
    - Takes JSON serialized VC as parameter.
    - Encrypt VC with re-encryption public key.
    - Send encrypted VC to server.
- List<String> restoreVerifiableCredentials()
    - Request VC's from server.
    - Decrypts VC's with re-encryption private key.
    - Return list of decrypted VC's to app.

### 6.2.3 Deployment

The Secret Manager Backup / Sync Library is deployed as a maven artifact to the ATOS KRAKEN Nexus Repository at:

https://registry.atosresearch.eu:8443/repository/kraken-maven/

In order to use the library as an artifact in a maven project, copy the following snippet into the project/dependencies section of your pom.xml:

```
<dependency>
  <groupId>eu.krakenh2020</groupId>
  <artifactId>ssi-backup-library-client</artifactId>
  <version>0.0.2-SNAPSHOT</version>
</dependency>
```

In case of using the secret manager in a gradle project, add the following line to your build.gradle file:

```
implementation 'eu.krakenh2020:ssi-backup-library-client:<version>'
```

### 6.2.4 Source code

The source code of the External Remote Storage is available on GitHub at:

https://github.com/krakenh2020/ssi-backup-library/tree/master/ssi-backup-library-client

### 6.2.5 Baseline technologies and tools

The Secret Manager Backup / Sync Library is written in Java 8. Similar to the External Remote Storage, the secret manager relies on cryptographic primitives from the IAIK JCE[19] and parses / serializes JSON with FasterXML Jackson Core library[20]. The zxing[21] image processing library generates and parses the QR Code of the Exchange Invitation.

[19] https://jce.iaik.tugraz.at/
[20] https://github.com/FasterXML/jackson-core
[21] https://github.com/zxing/zxing

# 7 Conclusion

This document is a report of the work performed in T3.1, T3.2, T3.3 and T3.4. The activity so far has focused on the development of the first prototype of the SSI solution which is one of the three main pillars of the KRAKEN platform. This report explains the work carried out to implement the building blocks of the basic SSI functionalities needed for the integration of the SSI solution with the Service Providers (marketplace and university) services. The implementation of the SSI solution is based on the work performed in WP2 where the components have been identified [1] and designed [2].

The components will be integrated with the KRAKEN marketplace and will be used in both pilots, the health and educational pilot. T3.1 develops tools for creating and managing verifiable credentials derived from trustable sources such as the eIDAS network. To this effect, the LIM and the KWCT components are provided. The outcomes of T3.2 such as the Trusted Framework, will facilitate the integration of the SSI solution with the KRAKEN marketplace decoupling and abstracting from the blockchain complexity the KRAKEN data sharing process and also enabling an environment aligned with EBSI/ESSIF. T3.3 builds up different edge tools for the adoption of SSI solutions on multiple devices, leveraging the work developed in T3.2 and T3.4. The implementation of tools such as the Ledger uSelf app for managing the VC and key material and the Ledger uSelf broker will facilitate the adoption of the SSI solution. Finally, T3.4 produces privacy preserving and secure solutions allowing the use of the SSI solution in several devices by creating client and server tools for managing and backup secrets on remote storage.

As results of future work in the context of these tasks (T3.1, T3.2, T3.3 and T3.4) advanced functionalities such as the revocation process and the DID resolution will be developed. Moreover, the current prototypes will be matured and updated according to the needs of both pilots. The final version of the SSI solution will be released in May 2022.

# 8 References

[1] KRAKEN project: D2.4 KRAKEN intermediate technical design, 2020. https://www.krakenh2020.eu/index.php/node/114

[2] KRAKEN project: D2.2 Intermediate KRAKEN architecture, 2020. https://www.krakenh2020.eu/index.php/node/84

[3] Matt Blaze, Gerrit Bleumer, and Martin Strauss. Divertible Protocols and Atomic Proxy Cryptography. In Kaisa Nyberg, editor, Advances in Cryptology - EUROCRYPT '98, International Conference on the Theory and Application of Cryptographic Techniques, Espoo, Finland, May 31 - June 4, 1998, Proceeding, volume 1403 of Lecture Notes in Computer Science, pages 127–144. Springer, 1998.

[4] ESSIF v2: Reference architecture. High-level diagram of key "meta-components" ESSIFv2. https://ec.europa.eu/cefdigital/wiki/pages/viewpage.action?pageId=316802761

[5] Hyperledger Aries RFC 0023: DID Exchange Protocol 1.0. https://github.com/hyperledger/aries-rfcs/tree/main/features/0023-did-exchange

[6] Hyperledger Aries RFC 0453: Issue Credential Protocol 2.0. https://github.com/hyperledger/aries-rfcs/tree/main/features/0453-issue-credential-v2

[7] Hyperledger Aries RFC 0454, Present Proof Protocol 2.0. https://github.com/hyperledger/aries-rfcs/tree/main/features/0454-present-proof-v2

[8] https://json-schema.org/draft-07/json-schema-release-notes.html


Atos

FONDAZIONE BRUNO KESSLER

AIT AUSTRIAN INSTITUTE OF TECHNOLOGY

sic

LYNKEUS
STRATEGY CONSULTING | BLOCKCHAIN & SMART CONTRACTS | DATA ANALYTICS

XLAB

TX

KU LEUVEN CiTiP
CENTRE FOR IT & IP LAW

IAIK TU Graz

InfoCert
TINEXTA GROUP

@KrakenH2020

Kraken H2020

**www.krakenh2020.eu**

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 871473