

PoSeID-on

Protection and Control of Secured Information
by Means of a Privacy Enhanced Dashboard

GRANT AGREEMENT NUMBER: 786713
H2020-DS-2016-2017/DS-08-2017

Deliverable 3.1

PoSeID-on blockchain - Interim implementation



Disclaimer

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 786713. This document has been prepared for the European Commission. However, it reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

PoSeID-on blockchain - Interim implementation

Project Title:	PoSeID-on
-----------------------	------------------











Deliverable Number:	D3.1
Grant Agreement number:	786713 H2020-DS-2016-2017/ DS-08-2017
Funding Scheme:	HORIZON 2020
Project co-ordinator name:	Francesco Paolo Schiavo – MEF (Italian Ministry of Economy and Finance)
Title of Deliverable:	PoSeID-on blockchain – Interim implementation
WP contributing to the Deliverable:	WP3 Blockchain and Smart contracts
Deliverable type	R – Report
Dissemination level	PU - Public
Partner(s)/Author(s):	I. Gutierrez, S. Anguita, C. Cortés (TECN), J. van Rooij (JIBE), R. Manzo, B. Intonti, N. Bucello (ACN), P. Silva, R. Casaleiro, M. Curado (UC), M. P. Verzillo, D. Reccia (ELEX)

History:			
Ver.	Comments	Date	Author
0.0	Draft version	28/01/2019	C. Cortés (TECN)
0.1	TOC	04/02/2019	I. Gutiérrez (TECN)
0.2	Information added	23/07/2019	all
0.3	Final version for internal review	25/07/2019	all
1.0	Final version ready for delivery	30/07/2019	all

Project funded by the European Commission Horizon 2020 - The EU Framework Programme for Research and Innovation.

The PoSeID-on Consortium consists of following partners identified in the table provided next:

Table 1 - Consortium Partners

Logo	Name	Country
	MEF – Ministero dell'Economia e delle Finanze	Italy
	ACN - Accenture S.p.A.	Italy
	PNO – PNO Innovation	Belgium
	ELEX – e-Lex Studio Legale	Italy
	TECN – Fundación Tecnalia Research & Innovation	Spain
	SAN – Ayuntamiento de Santander	Spain
	SOFT - Softeam	France
	UC – Universidade de Coimbra	Portugal
	JIBE – SMARTFEEDZ B.V.	Holland
	MITA – Malta Information Technology Agency	Malta

Executive Summary

This deliverable builds on the results of the initial version of the low level-design and implementation for the Blockchain and Smart Contracts Module developed at Task 3.1 ("Privacy ensuring Permissioned BC implementation"), Task 3.2 ("Private data managing smart contracts implementation"), Task 3.3 ("BC Client implementation") and Task 3.4 ("API Gateway implementation"), providing a description of the core mechanisms that will support the privacy protection and control of secured information of the users in PoSeID-on, based on distributed ledger technology.

The goal of Blockchain and Smart Contracts design and implementation is to provide PoSeID-on with a solution that enhances the privacy protection and the security of the way Data Subjects are providing and managing the permissions to access their personal data. These mechanisms will rely in a distributed ledger technology supported by a permissioned Blockchain network and a series of secure Smart Contracts that will be elaborated for enabling privacy of the personal identifiable information and data of the end-users.

As it is shown in the state-of-the-art, Blockchain techniques are being applied in different governments. So that, it exists the necessity to accomplish with the GDPR, specially with art.17, where PoSeID-on has ended up with a solution using burnable pseudo-identities, as well as the use of the relationship between transports keys and the Blockchain keys.

The requisites to be covered during the components' design and development undertaken inside the WP3 are translated into a section explaining the detailed development design that arises from the WP2 output architecture. The flows and interactions from the requisites' actions (list, grant, revoke, request and check) will be further explained and analyzed there.

To permit the actions and notifications present in the requisites, a correct key management and user anonym system definition are mandatory. To fulfill that need, consequent sections will be created in this document.

This set of definitions, designs and interactions will be later developed, and that development will lead in a prototype also detailed in this document.

This document will also provide a final glimpse of the integration works affecting the Blockchain and DP API created components, broaden and detailed in the WP5 appropriate document.

The necessary network infrastructure and services that will be constructed to constitute the permissioned blockchain and the platform APIs, and that work will be reflected in this document.



TABLE OF CONTENTS

Executive Summary.....	5
PoSeID-on Terminology/Glossary	9
1. Introduction.....	11
1.1. Scope and Relation with the PoSeID-on Workplan.....	11
1.2. Structure of the Document	11
2. Analysis of System Requirements Coverage	12
3. State-of-the-art in Blockchain technology for eGovernment services.....	16
4. Architecture and Detailed Design	18
4.1. Architecture Overview	18
4.2. Design.....	21
4.2.1. Cloud-based Permissioned Blockchain	22
4.2.2. Account Manager	25
4.2.3. Smart Contract Basics	34
4.2.4. Blockchain Client Flows	36
4.2.5. Data Processor API	46
4.3. Development status	48
4.4. Security remark	50
5. Initial Prototype of Blockchain Functionality	51
5.1. Smart Contracts and Blockchain functionalities.....	51
5.1.1. Smart Contracts.....	51
5.1.2. Blockchain Functionalities	53
5.1.3. Key Management	56
5.1.4. Blockchain Client.....	60
6. Integration in PoSeID-on architecture	61
6.1. Packaging.....	61
6.2. Configuration	62
6.3. Communication	62
6.4. Data Storage	64
6.5. Example of the Integrated Use Case between two pilots.....	64
7. Innovation summary	65
8. Conclusion.....	66
9. References	67
Annex I. Blockchain API formal documentation	69

LIST OF FIGURES

Figure 1 - Main modules of the PoSeID-on system	19
Figure 2 - Detail of the modules - Technical overview of the system.....	20
Figure 3 - Final components in the Blockchain architecture first design	21
Figure 4 - Quorum Logical Architecture Diagram	24
Figure 5 - Flow throw the Burnable Pseudo-Identities Solution Implementation	26
Figure 6 - Request ID & Request Payload.....	27
Figure 7 - Account generation simplified flow diagram.....	30
Figure 8 - Account recovery flow diagram.	32
Figure 9 - Steps to be followed from the ledger transaction flow according to official documentation [29] guidelines.	33
Figure 10 - PoSeID-on General Architecture	37
Figure 11 - BC API authentication diagram	39
Figure 12 - New Data Subject registration diagram	40
Figure 13 - Data Subject account recovery/regeneration diagram.....	41
Figure 14 - Access permissions granted by Data Subject diagram.....	42
Figure 15 - Grant access permissions to Data Processor diagram.....	43
Figure 16 - Revoke access permissions to Data Processor diagram	44
Figure 17 - Request access permissions to Data Processor diagram.....	45
Figure 18 - Batch permissions interaction to Data Processor diagram.....	46
Figure 19 - Communication between PoSeID-on and the Data Processor API	48
Figure 20 - Internal Permission Model and Flow.....	52
Figure 21 - Multiple account generation algorithm for Burnable Identities and Key rotation support [33]	59
Figure 22 - HDwallet creation example [34].....	60
Figure 23 - JSON-RPC message exchange example	60
Figure 24 - Default Blockchain Client Implementation for Standard Quorum Implementation ...	61
Figure 25 - Blockchain Client Implementation for PoSeID-on platform.....	61
Figure 26 - Data Processor's Blockchain Module.....	63
Figure 27 - Integration with Message bus diagram.....	63
Figure 28 - Integrated Use Case between two pilots	64



LIST OF TABLES

Table 1 - Personal data protection requirements.....	13
Table 2 - BC Technology applied in some regions	18
Table 3 - Development status	49
Table 4 - Legend.....	49
Table 5 - Permission Status / Description	52
Table 6 - BC Functionalities.....	53
Table 7 - Request PII Permission.....	53
Table 8 - Grant PII Permission	54
Table 9 - Revoke PII Permission	54
Table 10 - Check PII Permission	55
Table 11 - PII Access Notification	55
Table 12 - Other non-related functionalities.....	56

PoSeID-on Terminology/Glossary

This section complements the list of acronyms already provided, presenting short definitions and/or descriptions of selected terms used throughout this deliverable. While some of these terms are generic (e.g. blockchain, GDPR), most are specific from the PoSeID-on project and extensively used in the technical discussions along this document.

- **Administrator** – person or persons responsible for configuring, operating and maintaining the PoSeID-on platform.
- **Advanced Message Queuing Protocol (AMQP)** - is an open standard application layer protocol for message-oriented middleware.
- **Anonymization** – the process of removing personal identifiers that may lead to a Data Subject being identified.
- **Blockchain** - distributed digital ledger of cryptographically signed transactions that are grouped into blocks.
- **Blockchain API** – an interface that abstract all blockchain operations into a high-level application programming interface.
- **Burnable pseudo-identities** – pool of pseudo-identities of Data Subjects that can be erased on user request in order to unlink PII or PII metadata from its Data Subject.
- **Cryptographic Hash Function** – a one-way (i.e., irreversible) function that takes as input a variable length bit stream and produces a fixed-size random-like output. Cryptographic hash functions are also collision resistant, i.e., they are constructed so that it is computationally infeasible to find any two distinct inputs that map to the same output.
- **Data Processor (DP)** – an entity that processes or stores PII data and that can also determine the purposes, conditions and means for processing PII data.
- **Data Processor API** – an API made available by Data Processors to interface with their PII database, and that is also used by Data Processors to access the PoSeID-on platform.
- **Data Subject** – a natural person that represents the primary target of GDPR.
- **eIDAS (electronic IDentification, Authentication and trust Services)** - is an EU regulation and a set of standards on electronic identification and trust services for electronic transactions in the European Single Market.
- **GDPR – General Data Protection Regulation**, (EU) 2016/679 - is a regulation in EU law on data protection and privacy for all individuals within the European Union (EU) and the European Economic Area (EEA).
- **Message bus** – a combination of a common data model, a common command set, and a messaging infrastructure to allow different PoSeID-on components to communicate through a shared set of interfaces.
- **NoiPa** - NoiPA is the IT system provided by the Ministry of Economy and Finance to manage the processes of processing, settlement and consultation of salaries of staff of the Public Administration.

Through the NoiPA portal, public employees can consult a series of documents regarding their job position.

- **Permissioned Blockchain** - a blockchain where every node and every user must be granted permissions to utilize the system. Permissions are generally assigned by an administrator.
- **Personal Identifiable Information (PII)** – information related to a Data Subject, that can be used to directly or indirectly identify the person.
- **PII metadata** – a set of data that describes and gives information about PII data. Depending on circumstances, metadata may be privacy-sensitive and require the same privacy protection mechanisms that are applied to PII.
- **PII Value** – an instantiation of a PII Type, such as 'John', 'Smith', 'June 5th, 1959', or even an entire document, such as a signed contract, etc.
- **Pseudonym** – an alternate name for a Data Subject so that he/she cannot be directly identified. Identification can still be done with the use of additional data that, typically, is kept outside the main system.
- **Risk Management Module** – a PoSeID-on component that assesses and manages privacy risks by analysing transactions information.
- **Smart Contract** – code (functions) and associated data (state) that automatically execute the terms of a contract, without third parties' intervention, and that is deployed using cryptographically signed transactions on a blockchain network.
- **Web-Based Dashboard** – a web application that gives Data Subjects access to the PoSeID-on platform.

1. Introduction

This deliverable will include an initial version of the PoSeID-on Blockchain network design, including the implementation of the Private Data management Smart Contracts and the Blockchain client that allows to access the network. Finally, the Gateway Module design is included to provide cloud support for the services that use the PoSeID-on platform.

1.1. Scope and Relation with the PoSeID-on Workplan

This report is the first deliverable of Work Package WP3 ("Blockchain and Smart Contracts"). It has been produced within the scope of Task 3.1 ("Privacy ensuring Permissioned BC implementation"), Task 3.2 ("Private data managing smart contracts implementation"), Task 3.3 ("BC Client implementation") and Task 3.4 ("API Gateway implementation", now referred as "Data Processor API"), and its goals are fourfold:

- The development of the **permissioned blockchain network** as a distributed and secure means for personal data transactions;
- The development of the **smart contracts** that ensure privacy of personal data transactions in the blockchain. The smart contracts will be self-executing code in the blockchain that can be verified by the participants in the blockchain to enable the enforcement of user (data subject) defined permissions over specific data, whenever data controllers or data process would ask access to them;
- The development of the **blockchain client** that enables the user control and get informed on the status of transactions in the blockchain. This blockchain client will be integrated with the general Web-based Dashboard that provides additional features to the end-user such as risk assessment and personal data identification;
- The development of **APIs** for accessing to the cloud where users' wallets and optionally personal data are saved.

This deliverable will include the description of an initial version of the PoSeID-on blockchain network, including the initial smart contracts and blockchain client implementation, as well as the Data Processor API Module.

1.2. Structure of the Document

The rest of this deliverable is structured as follows:

Section 2 provides an analysis of the requirements coverage that must have been done at the end of the WP3 works. This is an introduction to the objectives to cover during the WP3 that will allow readers less familiar with the PoSeID-on project to understand the rationale behind the identified requirements. It also presents a set of guidelines, agreed between the project partners, that has been considered in the definition of the architecture. That architecture guidelines will conduct the design and implementation decisions affecting the Blockchain and API Gateway modules addressed in this Work Package.

Section 3 provides the state-of-the-art in Blockchain technology for eGovernment services. It begins with a general definition regarding Blockchain technology, giving the main characteristics and relating the Blockchain technology chosen, continue listing some different ways in which

governments have been applied this technology and finalize with examples applied along different regions.

Section 4 provides firstly and Architecture overview and follows describing the final design chosen, detailing the Cloud-based permissioned Blockchain, the account manager (mentioning the importance of the GDPR for this module), the smart contracts, the Blockchain client flows and the Data Processor API. It ends up explaining the development status and describing the security remarks.

Section 5 provides the description of the Initial Prototype of Blockchain Functionality, describing deeply about Smart Contracts and Blockchain functionalities, Key Management plan and Blockchain Client.

Section 6 provides a general description of the integration for the obtained components (containers and images) in the overall PoSeID-on architecture. It will enclose the packaging, configuration, storage and communication activities. These integration works, affecting the Blockchain and DP API, broadening and detailing will be done in the appropriate document of the WP5.

Section 7 provides the innovation key points achieved during the WP3 development and their alignment with the overall PoSeID-on scientific objectives. This is a translation of the work done inside the WP3 tasks, that will result on the Blockchain and the DP API software modules.

Section 8 provides the conclusion of the work done during the making of this deliverable. Further works aligned with the deliverable are envisioned.

Section 9 provides the references that are needed to follow the document explanations. They contain the knowledge of external systems, technologies and research.

Annex I provides the formal documentation to interact with the Blockchain API from any external component, and subsequently, with the Blockchain platform and Smart Contract functions themselves.

2. Analysis of System Requirements Coverage

In this section, the system requirements coverage is analyzed. The input for the section is the "requisites" table developed at the WP2. The requirements affecting the Blockchain network components, the Smart Contract coding and the Data Processor API have been extracted from that table to work over them. Extracting those items will help to focus the work done in this deliverable.

The requirements affecting WP3 developments, in any of the forms of caller or destination interfaces and modules, are gathered in the list below these lines:

- List access permissions granted by Data Subject
- Grant access permission to Data Processor
- Revoke access permission to Data Processor
- Request access permissions to Data Processor
- Check access permission by Data Processor
- Notify access to permission
- Blockchain Events (Transactions validated/denied)

The first three are motivated by Web Dashboard actions, the next three are motivated by the DP API, and the last of them is composed by events launched from actions taken inside the Blockchain platform that might be notified to the RMM or to the PDA. The events are sent to create operation logs regarding the risk and personal data usage analysis.

The requisites to be covered during the components' design and development undertaken inside the WP3 are translated into a section explaining the detailed development design that arises from the WP2 output architecture. The flows and interactions from the requisites' actions (list, grant, revoke, request and check) will be further explained and analyzed there.

To permit the actions and notifications present in the requisites, a correct key management and user anonym system definition are mandatory. To fulfill that need, consequent sections will be created in this document.

This set of definitions, designs and interactions will be later developed, and that development will lead in a prototype. This step will prove the accomplishment of the target system requisites.

It is also important not to forget the strong union between PoSeID-on and the GDPR. As a result of that, the project requisites are in line with the GDPR. The Table 1 studies the linkage between regulation provisions and the functional recommendations embraced by the WP3.

Table 1 - Personal data protection requirements

GDPR provision		GDPR reference	PoSeID-on Recommendation
Lawfulness of collected personal data	<p><i>Lawfulness</i> implies having legitimate grounds for collecting and using the personal data, including having the unambiguously given consent of the individual whose personal data are being processed and not using the data in ways that have unjustified adverse effects on the individuals concerned, and each purpose is consented separately unless it is appropriate to merge them.</p> <p>Sensitive data concerning a person's race, political opinions, religion, sexuality, genetic info and other biometrics should be prohibited by default unless consent is explicitly given and processing is necessary.</p>	Art. 5, 6	<p>The consent should be recorded and a clear record of the agreement of each data subject should be kept. An explicit consent for sensitive data should be requested.</p> <p>Functional mechanisms should be also set up for consent withdrawal, implying the capability to locate and remove the personal data during the process and also from backups and archives (even in cloud).</p>
Adequateness, relevance and proportionality	<p>For the <i>purpose limitation</i> principle, collected data should be adequate and relevant to the objectives of the system of collected personal data. Therefore, for the <i>data minimization</i> principle, only the minimum amount of personal data</p>	Art. 5	<p>The collection of personal data must be limited to what is directly relevant and necessary to accomplish the PoSeID-on platform specified purpose. The purpose has to be legitimate, and it has to be specified and made explicit before collecting personal data.</p>

GDPR provision		GDPR reference	PoSeID-on Recommendation
	which is needed to achieve the specific PoSeID-on platform purpose must be collected, used and retained.		
<i>Accuracy of the collected personal data</i>	Data has to be the right value, it has to precisely represent the value in consistent form and it must be up-to-date.	Art. 5	Procedures to keep data up-to-date should be implemented, such as the final user validation of data. Functional mechanisms should be in place to check, edit and extend stored data, with various controls concerning secure and reliable identification, authentication, access, validation, etc. These mechanisms may also affect backup and archives copies.
<i>Storage limitation</i>	Personal data must be retained only as long as is necessary to fulfil the declared purpose. It must be erased or effectively anonymised as soon as it is not anymore needed for the given purpose.	Art. 5	Functional mechanisms should be able to erase specific stored data, with various controls concerning secure and reliable identification, authentication, access, validation, etc. These mechanisms may also affect backup and archives copies.
<i>Transparency and openness</i>	Being transparent about the purpose to use the data.	Art. 12	The PoSeID-on platform should provide appropriate information to individuals to exercise their rights, to data controllers to evaluate their processors, and to Data Protection Authorities to monitor according to responsibilities.
<i>Individual rights</i>	Individuals must have the possibility of effectively and conveniently exercise their rights to access and rectify as well as to block and erase their personal data and to obtain a usable portable electronic copy of their personal data. Further, they have the right to withdraw given consent with effect for the future.	Art. 15, 16, 17, 18, 19, 20, 21	Secure and reliable identification, authentication and data access should be ensured. A withdrawing form should be available in the platform. A mechanism should be implemented to identify the specific data that is to be blocked or restricted. Extracted data should be limited to the identified and authenticated person concerned and communicated securely (e.g. encrypted). All these mechanisms may also affect backup and archives copies.
<i>Automatic processing</i>	Subjects have the right to insist that key decisions arising from automatic processing of their personal data are manually reviewed/reconsidered.	Art. 22	Mechanisms allowing such a manual review should be implemented.

GDPR provision		GDPR reference	PoSeID-on Recommendation
Accountability	Data controllers and processors should be able to demonstrate the compliance with privacy and data protection principles and legal requirements.	Art. 24	Examples of accountability measures are related to tracking of personal data access and of communications with external systems, documenting and recording all processing activities, mapping data flow. Moreover, appropriate data breaches reporting, response, assessment and information security should be developed.
Data security	Data security addresses integrity, confidentiality and availability concerns.	Art. 25	From a privacy and data protection perspective, they require a set of rules to be applied to limit access to personal data only to authorized people, and to ensure that the data is trustworthy and accurate. Therefore, data should be kept secure applying Privacy Enhancing Technologies (e.g. encryption, pseudonymization, anonymization, identity and access management), preventing accidental disclosure of personal data, securing communications with external stakeholders (such as for instance external systems).

The security is another important topic in every software development. Focusing on maintaining the system security, a requirements' guideline has been defined to be followed during the WP3 works and discussions. These requirements will be included in the software development although they aren't explicitly mentioned. Further information can be read in the Security remark section.

Security requirements:

- 1) Identification and password management
 - a) User access
 - i) The users must be authenticated to the Web Dashboard by an eID scheme under the corresponding eIDAS identity provider implementation;
 - ii) Any stored key must be stored securely and encrypted;
 - iii) After authentication, the users will receive an authentication token, e.g. carried as a cookie (SECURE and HTTPONLY).
 - b) Administrative access
 - i) Any administrative access should be requested by secure channel (SSH).
- 2) Authorization and role management
 - a) On every module, control of user access rights must be provided.
- 3) Cryptography and key management

- a) The system needs to hold security functions to generate and manage internal private keys and the corresponding certificates.
- 4) Network and data security
 - a) It is necessary to consider the following
 - i) Web Application Firewall;
 - ii) Intrusion Detection Services;
 - iii) VPN and/or TLS;
 - iv) Whitelist access controls.
 - b) Inside the DLT network it is appropriate to use a Reverse Proxy to protect incoming DLT connections.

3. State-of-the-art in Blockchain technology for eGovernment services

Blockchain is one form of implementation of distrusted ledger technology and combines a series of cryptographic algorithms like public-key-cryptography (PKC), elliptic-curve (EC) digital signatures and hashing which together guarantee the following key characteristics:

- **Decentralized network:** There is no central authority and no central data storage, i.e. no single point of trust, vulnerability or failure.
- **Trustlessness:** A blockchain does not require trust in any authority or every participant.
- **Consensus-based network:** A process allows participants to come to an agreement over what (e.g. the validity of a transaction) is true or false.
- **Transparent and traceable transactions:** all transactions in a blockchain are visible and verifiable to participants with rights.
- **Immutable transactions:** transactions and blocks added to the blockchain are technically impossible to manipulate or modify.
- **Security:** assets in the blockchain are cryptographically secured, and due to its decentralized nature, there is no single point of failure being Denial of Service resistant by design.
- **Pseudonymous and anonymity:** implementations in block chain vary a lot in this respect, most chains allow implementing pseudonyms for id nodes in the network, but anonymity levels tend to be low due to traceability in transactions. The validity of all transactions is available to everyone on the network

Public blockchains are implementations of the distributed ledger where the data inside the blockchain (transactions) is open to the public and everyone can take part as a node while Private blockchains also called permissioned[1] blockchains operate inside a previously defined network of participants.

As it has been mentioned in the D2.2, section 7.1, Quorum[2] is the selected Blockchain base platform to be used in PoSeID-on, complemented with Smart Contracts and using Constellations.

Due to the key inherent properties of the technology (previously described), Blockchain, one of the most significant innovations in data gathering and processing to appear in a long time, has

captured the attention of government administrators in the European Union[3]. In fact, Blockchain has been applied in government services in many ways:

- **Securing and sharing important data and records:** Verification of the records and sharing of data of various kinds:
 - *Identity:* In Switzerland, the city of Zug, in 2017 used the first publicly verified blockchain-based identity credential to residents[4] which they have subsequently used for e-voting[5] and renting e-bikes[6].
 - *Title/asset registrations:* It has been applied in Africa[7] and India[8], developing countries looking to fight corruption by local officials who could “steal” land by altering paper-based records. In Sweden, has been carried out the first successful test transaction of a fully blockchain-based transfer of title[9]. In the UK, HM Land Registry is testing blockchain in its bid “to become the world’s leading land registry for speed, simplicity and an open approach to data”[10].
 - *Healthcare:* In Estonia KSI Blockchain technology is being used to ensure data integrity and mitigate internal threats to the data[11]. In Sweden, there is an initiative to develop a national blockchain for health records to give citizens more control of their data[12].
 - *Educational certification:* The University of Nicosia issues academic certificates that can be verified online via a blockchain[13]. In Malta, the government is teaming up with a startup to build a prototype system to do the same[14]. A consortium of Malaysian universities is building a blockchain-based platform to combat fake degrees,[15] while a French startup is looking to use a blockchain network for the issuance and sharing of university and other degrees[16].
 - *E-Voting:* Related blockchain-based e-voting projects are underway in areas as far flung as West Virginia[17] and Moscow[18]. It is still a controversial topic within both political and scientific circles. Nevertheless, Blockchain-based e-voting solutions address almost all the security concerns, like privacy of voters, integrity, verification and non-repudiation of votes, and transparency of counting[19].
- **Monitoring and regulating markets:** Government regulates and monitors markets to protect consumers, make sure markets remain viable and ensure that laws are adhered to, as one of the key tasks. Singapore carried out a proof of concept related to this point with several banks[20].
- **Improving transactions, processes and transparency in public and private-sector markets:** The ways in which governments transact and interact with citizens and companies directly, particularly in complex settings with multiple stakeholders and high transaction volumes, using Blockchain for improving.
- **Efficiency:** Blockchain can help increase efficiency and reduce costs in government operations.
- **From dreams to reality:** Blockchain can increase efficiencies, reduce costs and improve security along the way.

The world is witnessing rapid adoption of Blockchain technology. Some numbers collected until 2018 are the following[21]:

- \$1.1 Billion Invested by private sector in 2016 alone.
- 600 New companies active in Blockchain in 2018
- \$290 Billion value expected market value in 2019
- Leading governments along the world exploring Blockchain Technology (Table 2):

Table 2 - BC Technology applied in some regions

REGION	BLOCKCHAIN TECHNOLOGY APPLIED
UNITED STATES (DE)	Company incorporation records stored on Blockchain (Delaware)
SINGAPORE	Invoicing on Blockchain
ESTONIA	E-Citizen records, e-payment keys and medical records secured on Blockchain
SWEDEN	Real-estate transactions on Blockchain
UK	Blockchain used to monitor the distribution of grants
GHANA	Land registry in Blockchain
CHINA[22]	Chancheng government applies blockchain technology to solve problems of identity, credit and information disclosure

In many regions are aligned with the advantages that Blockchain technology offers[23] to the public sector, not just to governments but also to the citizens. So that, it can be seen that it is important to find a solution in the way of how to use Blockchain technology following the recommendations of the GDPR (PoSeID-on solution detailed in Section 4).

4. Architecture and Detailed Design

It is going to be detailed both, the architecture and the design, as a detailed explanation along this section.

4.1. Architecture Overview

The low-level design carried out by this work package will be split in two steps. The first one will propose an initial solution of the Blockchain system and the Blockchain API that will open services and decentralized permission management to other modules. The second step will make an evaluation and redesign of the developed components.

This deliverable will formalize the works done to obtain the first version of the Blockchain system and the API Gateway. The second step is out of the scope of this deliverable and it will be achieved at the Deliverable 3.2 "PoSeID-on blockchain - Final Implementation".

The access to the data is restricted to outsiders, but also to insiders without permissions. The data subject can also hide their PII (Personal Identifiable Information) trace, performing the right to erasure (established in the GDPR, art. 17[24]) as the only actor fit to recover their uniquely owned keyset. The Smart Contracts rule the permissions enabling them to be made low level and granular. This fact, joint to the Account Manager module, ensures the privacy and lack of linkage for everyone once they deleted their account, that represents the last footprint.

The Blockchain system components and developments have been formulated in a way that each component is modular, reusable and easily modifiable without affecting other components in the system. This development philosophy allows to pivot from one solution to another in an agile way, according to the needs of the project.

Each separate component of the Blockchain Module represents a functionality entry point for the system, that has to be connected to another one in order to provide the platform with full functionality.

The created architecture is based on the works performed during the WP2 analysis and design. But that architecture has been modelled, fitted and focused on the solution, which translates into the modular scheme shown in Figure 1.

Figure 1 shows the main modules of the system in a comprehensive manner. Each block acts independently in development environments, and it can be tested as it is.

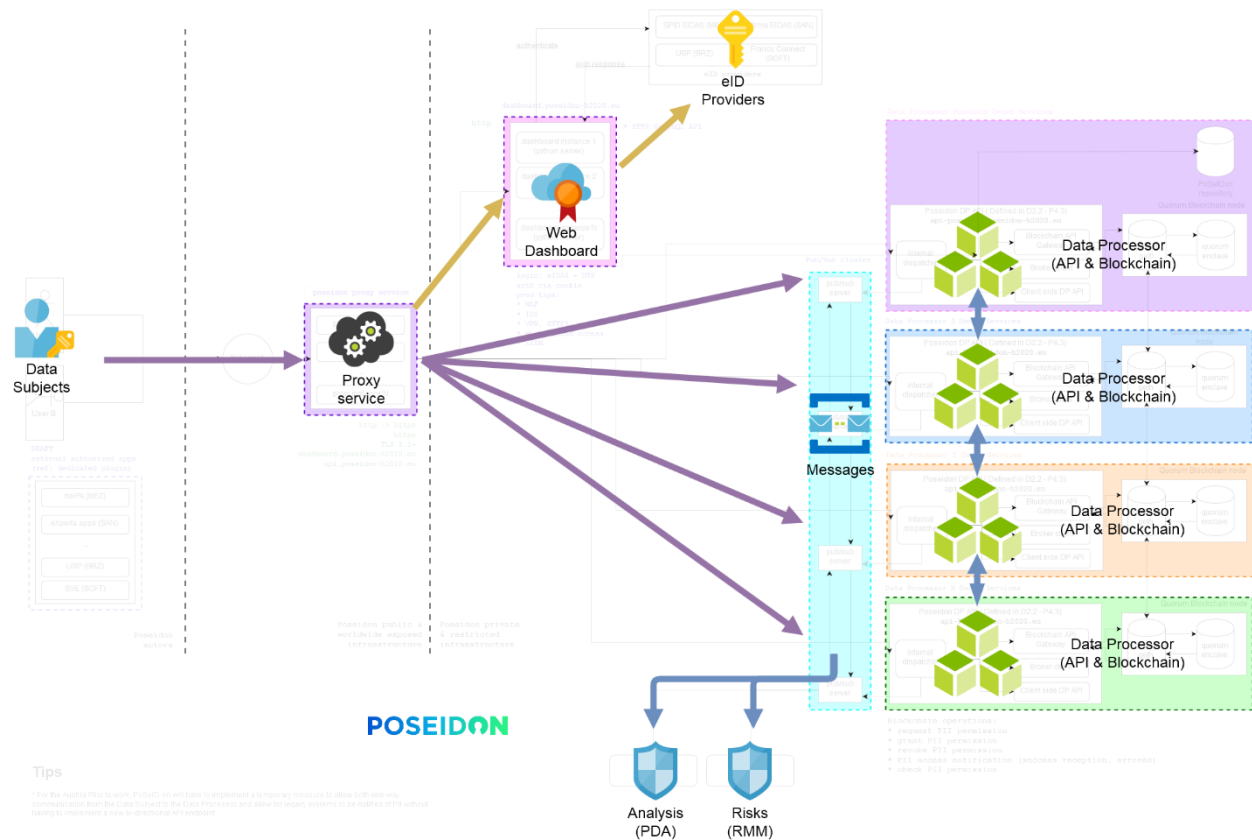
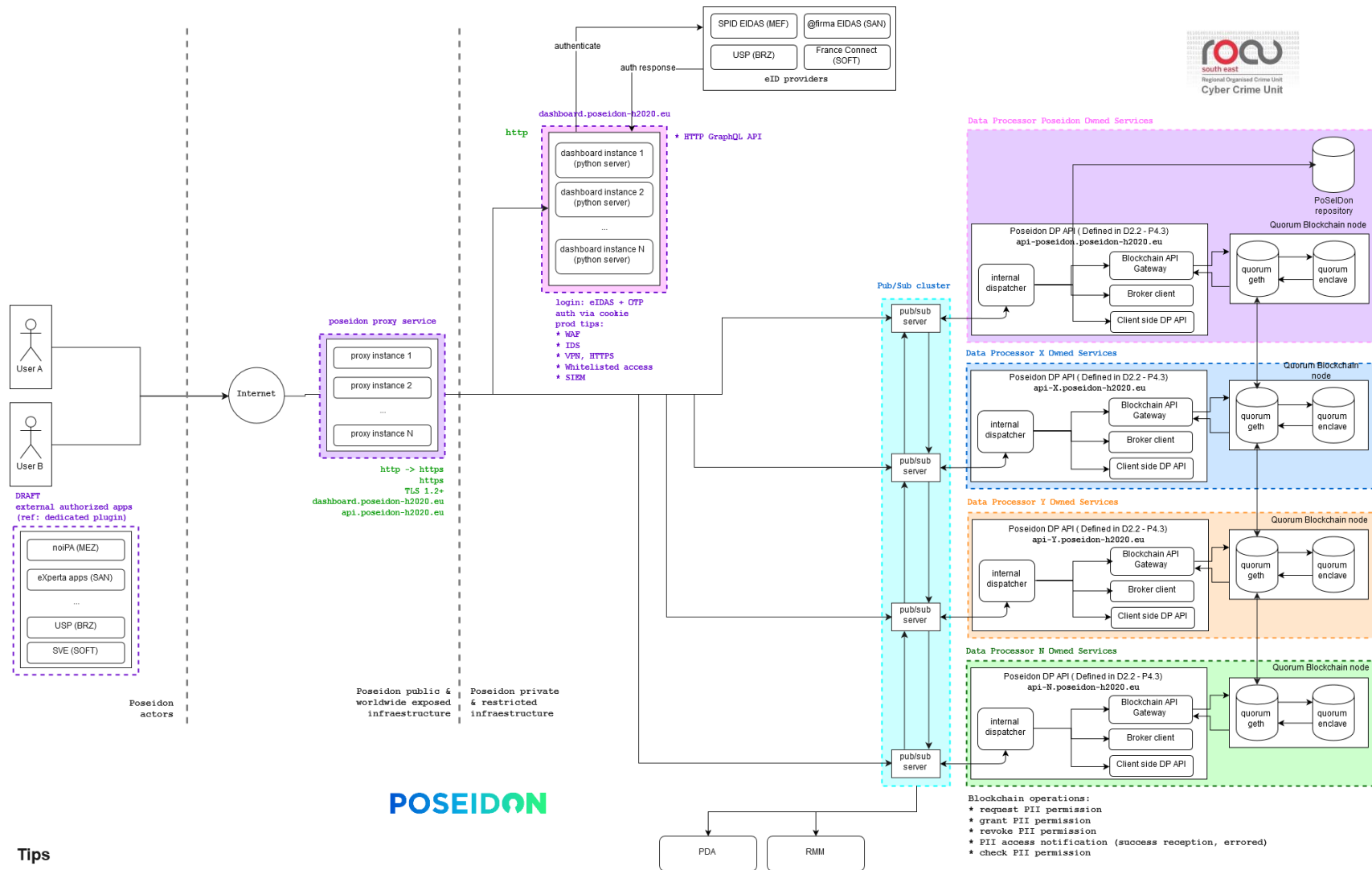


Figure 1 - Main modules of the PoSeID-on system

Figure 2 explodes each block in a technical overview of the system. It shows that the Blockchain part is closely connected to the Data Processor systems, but it has interactions with other actors in the system. For example, the key management is a key feature of the system to identify Data Subjects and Data Processors. Since both type of users' needs to interact with the Blockchain, the key management is also a key feature for Blockchain, and a key section inside this document.

The block connection arrows shall indicate the connections to be performed during the integration phase.



POSEIDON

Figure 2 - Detail of the modules - Technical overview of the system

4.2. Design

The PoSeID-on distributed ledger will be shared between a selected group of participants. The common point is that every participant needs to provide services for their users (the Data Subjects). Each organization will own its own peer. But from a technical, high level overview everything related to Blockchain is limited to two main components:

- Blockchain Communication Module as Blockchain API
- Distributed ledger nodes

These two main components will be later exploded and detailed in this design section. The section will be targeted from a technical, implementation and deployment focused, component-based point of view, as it is founded in the previous design deliverable D2.2, where the Blockchain module design was broadly studied and worked.

The Blockchain-related architecture first design has been adapted to better fit the PoSeID-on overall deployment needs. Its final state has the components shown in Figure 3 (they will be described in the following sections):

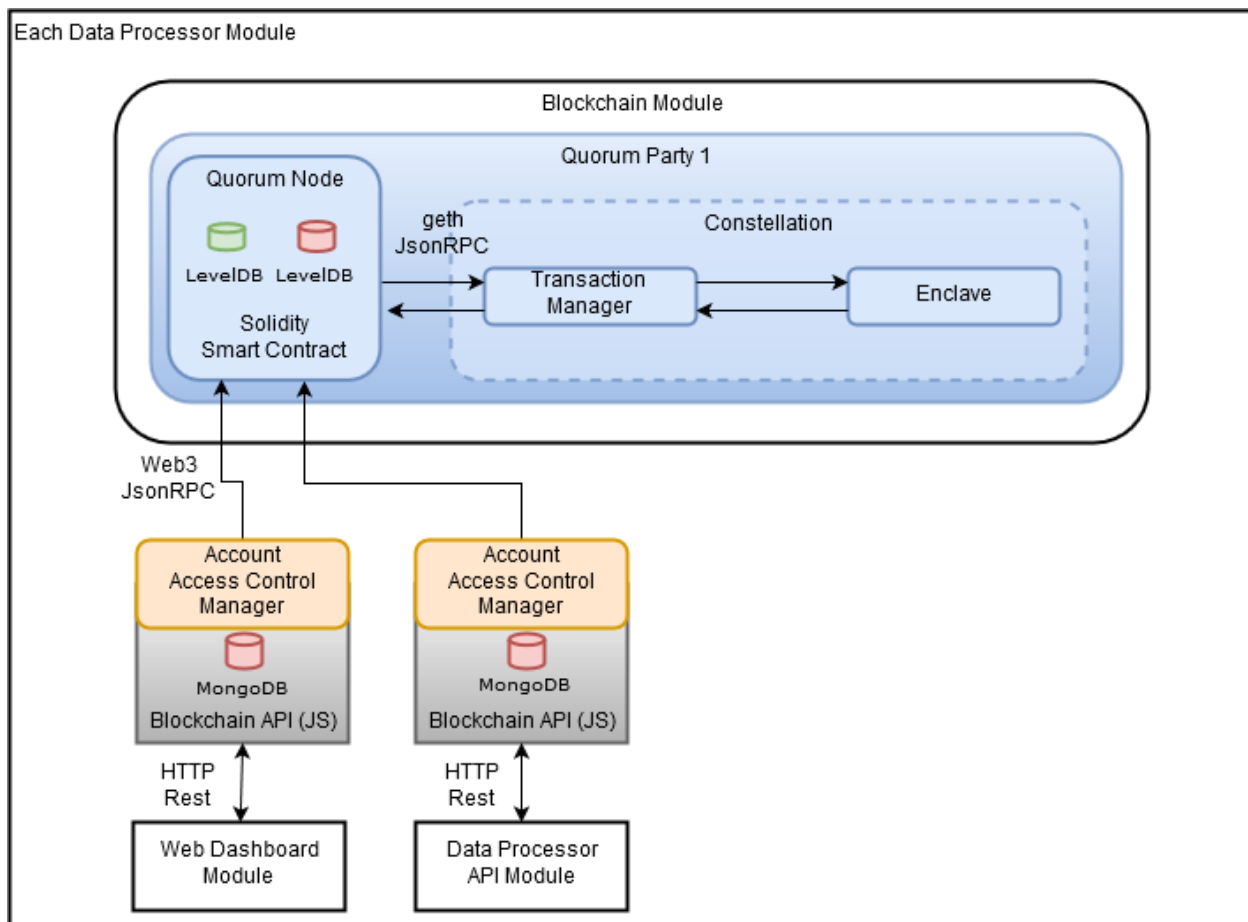


Figure 3 - Final components in the Blockchain architecture first design

4.2.1. Cloud-based Permissioned Blockchain

The Blockchain Network is considered a key component inside PoSeID-on because the continuously growing list of records that will be stored in an immutable way will keep the solution to allow the platform users to manage their permissions. The records kept by the nodes of this network are the output from the transactions to be linked inside the Blockchain blocks. In this solution, the data quality is maintained by massive database replication and computational trust instead of keeping single data copies produced by separate program executions that make the origin to be trusted.

The rest of the PoSeID-on platform involved components have been designed considering the decentralized and distributed character of the ledger created by this component. This decision is the driver of presenting a decentralized system governed under consensus that is reached by all participants of PoSeID-on in a democratic manner. No entity has greater control over the others and all entities are at par. It gives this solution the desired trust degree for an innovative research project.

The PoSeID-on blockchain network is designed as a permissioned network with private access ledger in which only duly authorized members can interact with each other. A permissioned blockchain is a specific blockchain type that has been adopted for the following reasons:

- **Privacy** – Only actors with view rights for specific transaction payloads can access to a specific piece of data. A permissionless blockchain would have been ideal as a shared database if everyone could read everything but being the BC permissionless no single user controls who can perform write operations. In the PoSeID-on use case scenarios, the transactions are not visible to anyone that is not authorized.
- **Scalability** – It is possible to adopt a specific consensus model. This prevents other permissionless oriented mechanisms, like Proof of Work, from burning a heavy amount of computational cycles. The ultimate result is scalability compared to a public blockchain network.
- **Traceability** – The selected type of blockchain allows to have a certainty over the current set of user permissions. These have been recorded by their interactions with PoSeID-on when they manage their own personal data. This capacity allows transparency in all the entities involved in the personal data access until the user decides to perform the so known “right of erasure”.
- **Immutability** – The adoption of a permissioned blockchain enables trust on data controllers/processors involved in the system. This is a well desired capacity because the data subject needs to be provided with functionality from them, but also needs a (sometimes disappeared) feel of protection.
- **Access Control** – The access to the data is restricted to outsiders, but also to insiders without permissions. The data subject can also hide their PII trace, performing the right to erasure (established in the GDPR, art. 17) as the only actor fit to recover their uniquely owned keyset. The Smart Contracts rule the permissions enabling them to be made low level and granular. This fact, joint to the Account Manager module, ensures the privacy and lack of linkage for everyone once they deleted their account, that represents the last footprint.

In this solution, all the Data Processors involved will record the PII Permissions they handle in their different business processes and services. This information will be used to provide services for the Data Subjects. The record of PII will be specially designed to comply with:

- The GDPR regulations, in force in the member countries of the European Union.
- The data protection regulations and specific implementations associated with the countries of the participating companies.

Blockchain network components

The explanation about how the PoSeID-on platform's distributed network has been deployed, based on the design proposed in the general architecture deliverable D2.2, is as follows (its different components will be enumerated).

The Blockchain network deployment, as any other distributed system may need, requires an ordered group of steps to successfully execute the process. For PoSeID-on, the deployment steps to be followed are defined below:

1. **Define the number of initial nodes to deploy:** According to previous agreements, each Data Processor (DP) will own and manage its own node, so for PoSeID-on first production release, this means that will have 4 nodes.
2. **Provision the hardware and virtual machines necessary for the deployment of the services.** No matter what vendor or provided is behind the service, all participants of PoSeID-on Blockchain need to have at least a fully working instance in order to deploy the Quorum Ledger node.
3. **Configure the node with a Digital Identity and boot-time parameters.** Nodes will require to share the genesis block configuration, usually encoded and shared as genesis.json file between ledger partners and parties. This genesis.json will have all required configuration details in order to deploy new nodes on the ledger. Apart from this, there are other several parameters that needs to be agreed upon boot and shared between partners. Such details are:
 - a) Ledger Network ID
 - b) List of IDs from allowed nodes (also known as list of permissioned nodes)
 - c) Gas value
 - d) Gas Price value

The selected blockchain to be applied to PoSeID-on, as it has been mentioned before, is Quorum as a permissioned chain. Quorum has the following components (shown in Figure 4):[25]

- **Quorum Node** (modified Geth Client): It is intentionally designed to be a lightweight fork of geth in order that it can continue to take advantage of the R&D that is taking place within the ever growing Ethereum community. To that end, Quorum will be updated in-line with future geth releases.
- **Constellation:** As it was mentioned in D2.2, it is a general-purpose system for submitting information in a secure way. It is comparable to a network of MTA (Message Transfer Agents) where messages are encrypted with PGP. It is not blockchain-specific and is potentially applicable in many other types of applications where you want individually-sealed message exchange within a network of counterparties. The Constellation module consists of two sub-modules:

- *Constellation/Tessera - Transaction Manager*: "Quorum's Transaction Manager is responsible for Transaction privacy. It stores and allows access to encrypted transaction data, exchanges encrypted payloads with other participant's Transaction Managers but does not have access to any sensitive private keys. It utilizes the Enclave for cryptographic functionality (although the Enclave can optionally be hosted by the Transaction Manager itself.) The Transaction Manager is restful/stateless and can be load balanced easily".
- *Constellation/Tessera - Enclave*: "Distributed Ledger protocols typically leverage cryptographic techniques for transaction authenticity, participant authentication, and historical data preservation (i.e. through a chain of cryptographically hashed data.). In order to achieve a separation of concerns, as well as to provide performance improvements through parallelization of certain crypto-operations, much of the cryptographic work including symmetric key generation and data encryption/decryption is delegated to the Enclave. The Enclave works hand in hand with the Transaction Manager to strengthen privacy by managing the encryption/decryption in an isolated way. It holds private keys and is essentially a "virtual HSM" isolated from other components."

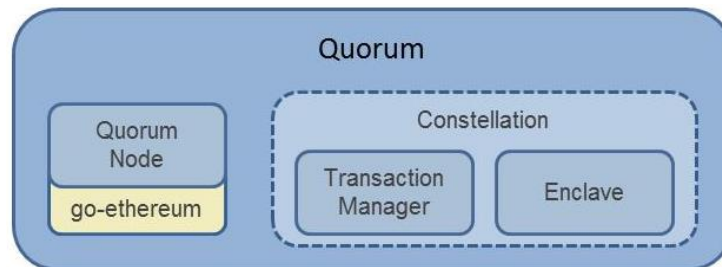


Figure 4 - Quorum Logical Architecture Diagram

- **Ledger and state DBs**: Only one distributed ledger is shared and replicated in each Quorum Node of each Blockchain Module of each Data Processor. The public state DB is also replicated in every Quorum Node.
- **Account manager**: External user accounts in Ethereum-based blockchain network are basically composed of an address and its state.
- **Clients**: The Client represents the entity that acts on behalf of an end-user (Data Subject in case of PoSeID-on).
- **Smart contracts**: Smart Contracts capabilities will be implemented by building as many contracts as needed for the target use cases in PoSeID-on.

In order to manage Quorum API, it has two methods already defined, as well as two calls.

On the one hand, Quorum API has the following methods: [26]

- **eth.sendTransaction**: To support private transactions in Quorum.
- **eth.sendRawPrivateTransaction**: To support sending raw transactions in Quorum.

On the other hand, in addition to the JSON-RPC provided by Ethereum, Quorum exposes two API calls:

- **eth_storageRoot**: Returns the storage root of given address (Contract/Account etc).
- **eth_getQuorumPayload**: Returns the unencrypted payload from Tessera/constellation.
- **eth_sendTransactionAsync**: Sends a transaction to the network asynchronously.

4.2.2. Account Manager

The account manager oversees the PoSeID-on platform user identities. This means that the purpose of this module is to ensure linkage between user known accounts and anonymous Blockchain accounts. The connection is dependent on how PoSeID-on user accounts are managed and how they can interact with an external module, as it is the Blockchain platform.

To achieve that goal, the management of user Blockchain accounts (identities) is a key point. Within this project, the Burnable Pseudo-Identities Solution has been designed.

4.2.2.1. *The importance of GDPR Art. 17 for the Account Manager module*

As explained in previous sections, the platform allows the Data Subjects to stop using PoSeID-on, and thus, activate the right to be forgotten (right of erasure as established in the GDPR, art. 17). When this action takes place, Data Processors must “forget” any data related to that specific user. As Blockchain keeps an immutable, traceable, forever growing record of the actions taken place in the network a special effort has been made to achieve this need.

From a technical point of view, the team has done research to find the best way to unlink data from the system without breaking the block record and without rejecting the Blockchain original philosophy, that would have ended in a misuse of the technology.

The solution has embraced the fact that if the system can't access the data and if it could be possible, the atomic accessible data has no sense, the data is useless. For example, if a Data Processor could view an address, but it can't be linked to a person or an account, no Data Subject data can be inferred. In other words, the dissociation between the single data and the data subject leads to an anonymization of the data itself. To achieve that goal, the management of user Blockchain accounts is a key point.

4.2.2.2. *The Burnable Pseudo-Identities Solution Design*

External user accounts in Ethereum-based blockchain networks are basically composed of an address and its state. The address has 20 bytes. Any person signing a request with a private key associated to a public one whose hash ends with the 20 bytes that match an address can act in the name of the corresponding account.

Focusing on the development, each write request that arrives to the ledger is composed by a unique address and the requested data. Since the Identity Data (external account address) is stored in the ledger in every user interaction (e.g. change a permission status in the PoSeID-on system) it is not possible to use a single user ID, identity or any kind of eIDAS (electronic Identification, Authentication and trust Services) information, because it would result on breaking the GDPR compliance because the PII about that user would be traceable. The aim of the proposed solution is to allow user interactions in a way that can be:

- Traceable over the time while the user is using PoSeID-on services.
- Compliant with GDPR and `right to be forgotten` when the user requests it.

As only known Data Processors will be the parties maintaining Blockchain nodes and the Data Subjects will be previously authenticated on the Web Dashboard by PoSeID-on with a legit identity (view eID Provider Module from the architecture), the access control strategy is to operate account permissions privately by each Data Processor. The account management will act as later

explained at section **iError! No se encuentra el origen de la referencia.** (Identity Management), where the eID Provider is meant to facilitate user information to manage its account after a correct identification.

This section will explain how the connection between the Data Subjects or Data Processors and their Blockchain identities has been implemented based on the design proposed.

4.2.2.3. The Burnable Pseudo-Identities Solution Implementation

In short, the Burnable Pseudo-Identities Solution is a mechanism to create a pool of pseudo-identities for each Data Subject user that can be erased by request. When the pseudo-identities are deleted by the Data Subject, the link between two given permissions does not exist; a permission history over the time can't be created; the Data Controller owning the data stored on the ledger is not known; and the Data Subject addresses are forgotten by PoSeID-on.

The proposed flow was described at D2.2 as shown in Figure 5.

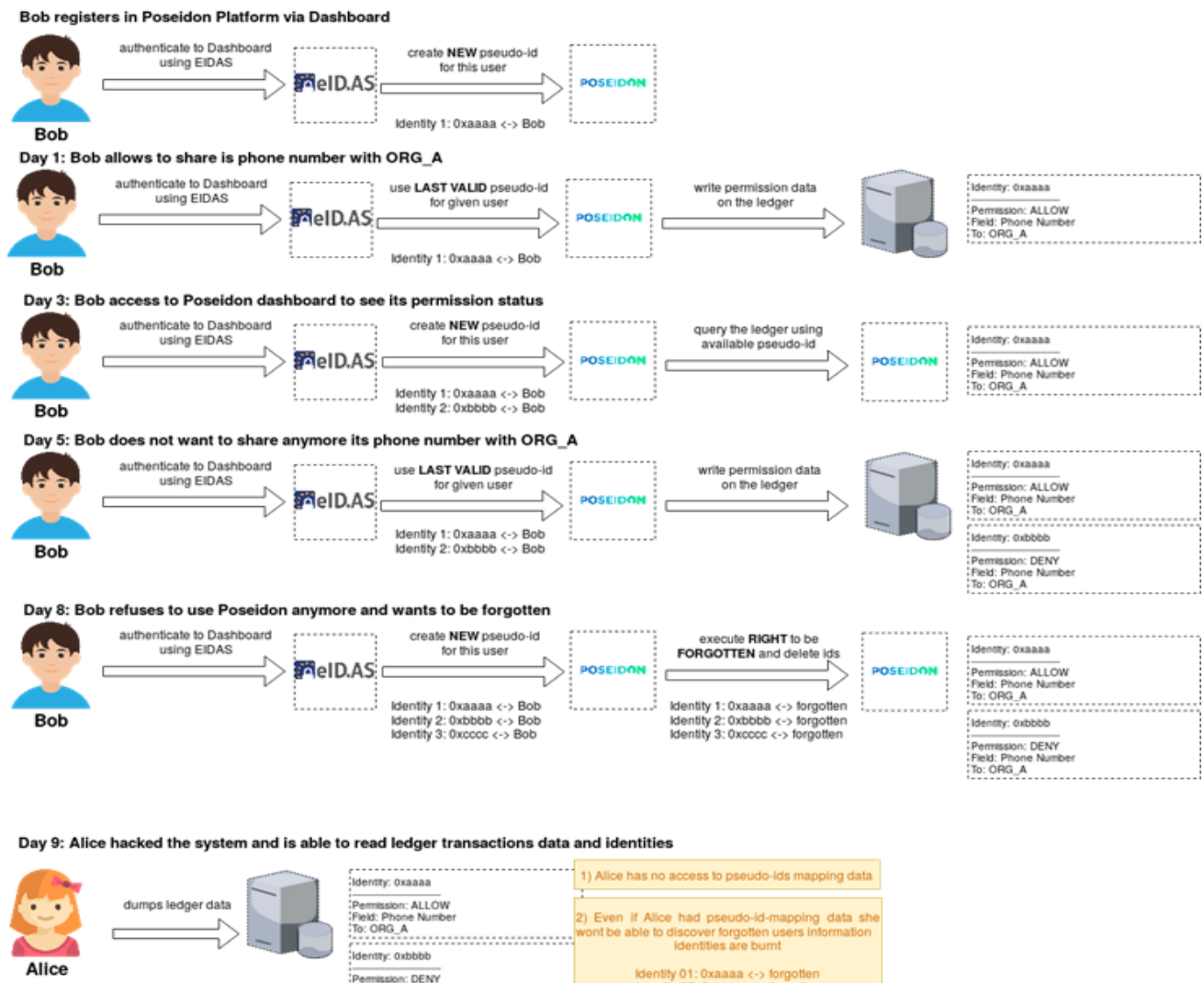


Figure 5 - Flow throw the Burnable Pseudo-Identities Solution Implementation

Given the PoSeID-on requirements, the ledger operational identities must not be persisted as immutable data over the time. Since the current available distributed ledger solutions don't work on this premise, an alternative for this issue needs to be proposed.

The alternative solution given by PoSeID-on project is to generate a set of pseudo identities for each user so that their legit (eIDAS based) identity remains secret for everyone and all ledger operations are made using those burnable pseudo identities.

Since the PoSeID-on platform needs to be GDPR compliant, it needs to operate upon user identity without revealing the user identity. This problem derives from following fact:

"All private and public Blockchain frameworks, store client identity along transaction data"

In the base code implementation selected for the project, Quorum, each data write request that arrives to the ledger is composed by a unique user id and request data as shown in Figure 6 below:



Figure 6 - Request ID & Request Payload

Mnemonic generation

Mnemonic phrases, also known as 'seed words' or 'recovery phrases', are ordered lists of 12 - 24 words which bring you to specific wallet addresses. These are typically used for recovery, and they are not meant to be used as the main method of access for anyone's wallet. These phrases offer direct access to one's wallet, so they should be treated carefully. Mnemonic phrases are just as sensitive as your private key, in terms of privacy. With your phrase, anyone can permanently access your wallet. These phrases cannot be changed, so keeping them safe is crucial.

A mnemonic example would be:

```
beyond enact slight piano fox way sight process side ritual inhale world
fiction mansion pattern
```

Which is automatically mapped to BIP39 seed like:

```
2f5b299827afc15d9129e18def7e3a59104ad86b0b73ef803c2aaa905661f44f4ac65b
e268b3676aed15b25c79dd2d0934216c98f48367fc83484f7347263ed4
```

This is the default mechanism of PoSeID-on that will be used for account recovery.

At implementation stage, account mnemonic generation was done using a secure library called bip39[27] which allow us to provide safe and stable mnemonic generation for all PoSeID-on partners and end users.

The BIP 39 standard

As shown by the official standard description created by Marek Palatinus, Pavol Rusnak, Aaron Voisine and Sean Rowe, BIP39 is the implementation of a mnemonic code or mnemonic sentence -- a group of easy to remember words -- for the generation of deterministic wallets. It consists of two parts: generating the mnemonic and converting it into a binary seed. This seed can be later used to generate deterministic wallets using BIP-0032 or similar methods.

The motivation behind this standard, is the lack of a mechanism to remember a raw binary string or hexadecimal number. In the case of quorum, this raw string represents the seed of a wallet. Thanks to BIP39, this raw string can be converted to a sentence that could be written on paper or spoken over the telephone.

The mnemonic must encode entropy in a multiple of 32 bits. With more entropy security is improved but the sentence length increases. We refer to the initial entropy length as ENT. The allowed size of ENT is 128-256 bits.

From mnemonic to seed

As it is explained in Official Bitcoin Improvement Proposals 39[28]: "A user may decide to protect their mnemonic with a passphrase. If a passphrase is not present, an empty string "" is used instead. To create a binary seed from the mnemonic, we use the PBKDF2 function with a mnemonic sentence (in UTF-8 NFKD) used as the password and the string "mnemonic" + passphrase (again in UTF-8 NFKD) used as the salt. The iteration count is set to 2048 and HMAC-SHA512 is used as the pseudo-random function. The length of the derived key is 512 bits (= 64 bytes).

This seed can be later used to generate deterministic wallets using BIP-0032 or similar methods. The conversion of the mnemonic sentence to a binary seed is completely independent from generating the sentence. This results in rather simple code; there are no constraints on sentence structure and clients are free to implement their own wordlists or even whole sentence generators, allowing for flexibility in wordlists for typo detection or other purposes.

Although using a mnemonic not generated by the algorithm described in "Generating the mnemonic" section is possible, this is not advised, and software must compute a checksum for the mnemonic sentence using a wordlist and issue a warning if it is invalid. The described method also provides plausible deniability, because every passphrase generates a valid seed (and thus a deterministic wallet) but only the correct one will make the desired wallet available."

4.2.2.4. Account generation

Cryptographic information is the most sensitive part of any system in which cryptography is considered one of the fundamental pillars. That is why in Blockchain systems it is necessary to pay special attention to this part, in order to avoid unwanted behaviours and possible security breaches.

For this reason, PoSeID-on has a user account management system in which, managed by the BC API, can provide PoSeID-on users, transparently, secure and reliable access to the Blockchain functionality without disclosing the content of them. Maintaining the security of private keys and their custody, on server side and avoiding communicating them over Internet, PoSeID-on ensures the integrity of the platform, hence, the generation of accounts is delegated to BC API.

The Blockchain API will create new Blockchain accounts upon external request by Data Subjects and Data Processors as it is shown in Figure 7.

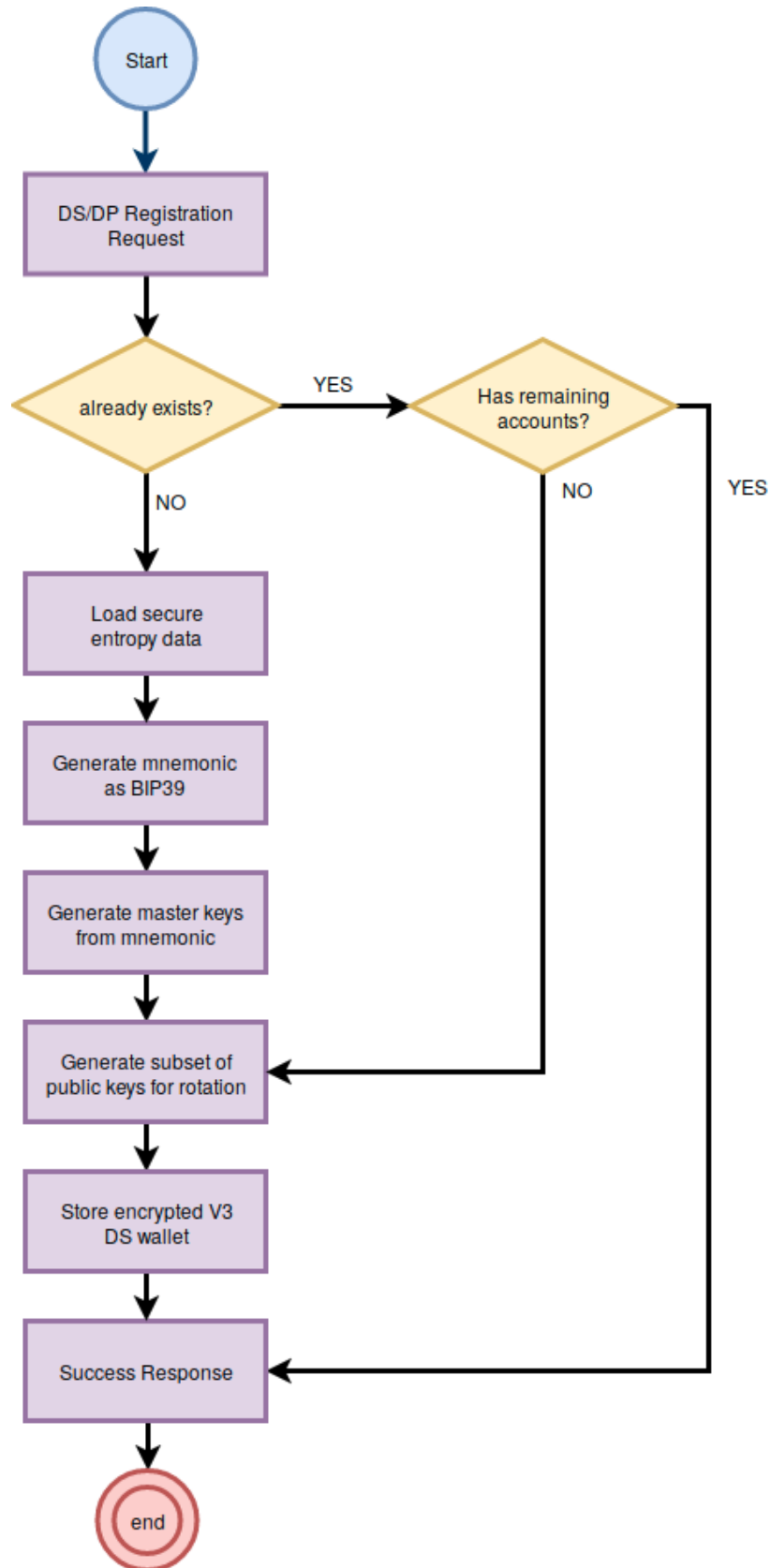


Figure 7 - Account generation simplified flow diagram

4.2.2.5. Account recovery

Since PoSeID-on was designed with a usability criterion in mind, it is required to have some sort of mechanism to allow users to get their accounts back when they forgot the password. However, since PoSeID-on authentication is done against EIDAS, PoSeID-on platform has nothing to do against this service rather than relying on its own EIDAS mechanism. However, from Blockchain point of view, PoSeID-on has included an account recovery mechanism for disaster recovery situation that will help to:

- Recover user account access when blockchain access, or the API is compromised.
- Recover user account access when users lose his/her access keys.

Account recovery flow

Currently, the account recovery assumes that PoSeID-on users can access to the platform using PoSeID-on dashboard (shown in Figure 8). The account recovery mechanism only covers the account managed by the blockchain itself and no other components, therefore there is no a PoSeID-on generic recovery mechanism. This is caused, indeed, by the fact that PoSeID-on Dashboard log-in system is based on eIDAS, and PoSeID-on rely on eIDAS for all account related procedures such as identification, etc.

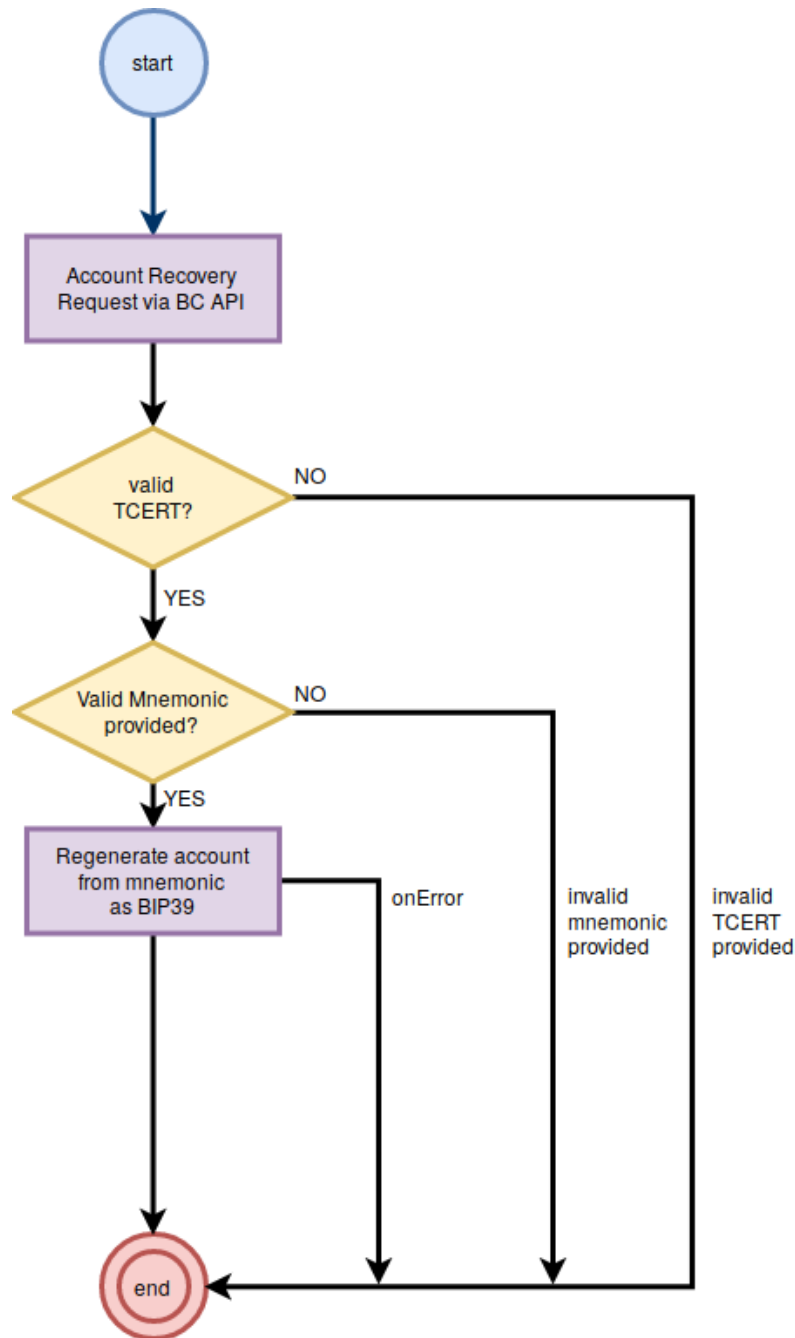


Figure 8 - Account recovery flow diagram.

Given the complexity and need for cryptographic material to execute transactions against the ledger, there are several considerations to make prior any interaction. The ledger transaction flow needs to be followed according to following steps, which is the common way of executing private transaction in Quorum Blockchain.

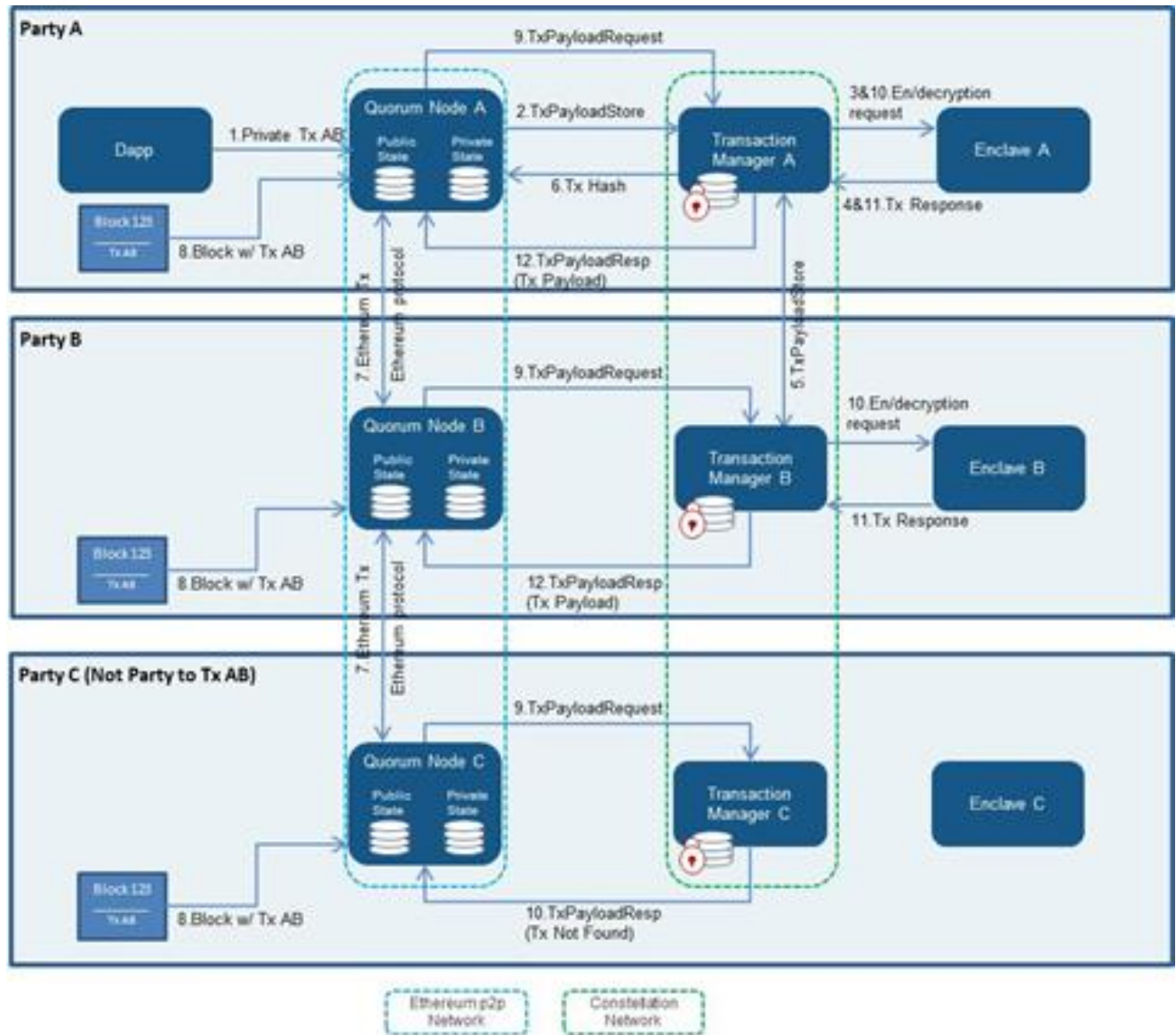


Figure 9 - Steps to be followed from the ledger transaction flow according to official documentation [29] guidelines.

In the previous example (Figure 9), Party A and Party B (considering them different Data Processors) are the ones involved in transacting. Thus, for illustration purposes we will call it Transaction AB, leaving Party C (Data Processor C in the PoSeID-on ecosystem) out of the scope of the transaction even though belong to same Blockchain network.

1. Party A sends a Transaction to their Quorum Node, specifying the Transaction payload and setting privateFor to be the public keys for Parties A and B.
2. Party A's Quorum Node passes the Transaction on to its paired Transaction Manager, requesting for it to store the Transaction payload.
3. Party A's Transaction Manager makes a call to its associated Enclave to validate the sender and encrypt the payload.
4. Party A's Enclave checks the private key for Party A and, once validated, performs the Transaction conversion. This entails:

- a) generating a symmetric key and a random Nonce
 - b) encrypting the Transaction payload and Nonce with the symmetric key from a).
 - c) calculating the SHA3-512 hash of the encrypted payload from b).
 - d) iterating through the list of Transaction recipients, in this case Parties A and B, and encrypting the symmetric key from a). with the recipient's public key (PGP encryption)
 - e) returning the encrypted payload from step b)., the hash from step c). and the encrypted keys (for each recipient) from step d). to the Transaction Manager
5. Party A's Transaction manager then stores the encrypted payload (encrypted with the symmetric key) and encrypted symmetric key using the hash as the index, and then securely transfers (via HTTPS) the hash, encrypted payload, and encrypted symmetric key that has been encrypted with Party B's public key to Party B's Transaction Manager. Party B's Transaction Manager responds with an Ack/Nack response. Note that if Party A does not receive a response/receives a Nack from Party B then the Transaction will not be propagated to the network. It is a prerequisite for the recipients to store the communicated payload.
 6. Once the data transmission to Party B's Transaction Manager has been successful, Party A's Transaction Manager returns the hash to the Quorum Node which then replaces the Transaction's original payload with that hash and changes the transaction's V value to 37 or 38, which will indicate to other nodes that this hash represents a private transaction with an associated encrypted payload as opposed to a public transaction with nonsensical bytecode.
 7. The Transaction is then propagated to the rest of the network using the standard Ethereum P2P Protocol.
 8. A block containing Transaction AB is created and distributed to each Party on the network.
 9. In processing the block, all Parties will attempt to process the Transaction. Each Quorum node will recognise a V value of 37 or 38, identifying the Transaction as one whose payload requires decrypting, and make a call to their local Transaction Manager to determine if they hold the Transaction (using the hash as the index to look up).
 10. Since Party C does not hold the Transaction, it will receive a NotARecipient message and will skip the Transaction - it will not update its Private StateDB. Party A and B will look up the hash in their local Transaction Managers and identify that they do hold the Transaction. Each will then make a call to its Enclave, passing in the Encrypted Payload, Encrypted symmetric key and Signature.
 11. The Enclave validates the signature and then decrypts the symmetric key using the Party's private key that is held in The Enclave, decrypts the Transaction Payload using the now-revealed symmetric key and returns the decrypted payload to the Transaction Manager.
 12. The Transaction Managers for Parties A and B then send the decrypted payload to the EVM for contract code execution. This execution will update the state in the **Quorum Node's Private StateDB only**.

NOTE: once the code has been executed it is discarded so is never available for reading without going through the above process.

4.2.3. Smart Contract Basics

Smart Contracts are independent from the context where they are being executed. Those Smart Contracts can be deployed both for development or pilot stages indistinctly.

A Smart Contract is a computer program that act as agreement, with the main goal to enable two parties to trade and do business with each other over the internet, without the need for a middleman (i.e. a centralized source of trust). Smart Data technical infrastructure can use blockchain technology because the distributed ledger system can be used to confirm whether contractual conditions have been met. In the framework of this project, the Smart Contracts technology will be leveraged to allow data subjects rest well about the way his own PII are managed. Smart Contracts are independent from the context where they are being executed. Those Smart Contracts can be deployed both for development or pilot stages indistinctly. The correct coding of each contract permits it to behave in predefined ways, then it is encrypted and sent out via distributed ledgers to other parties of this network. The major common components of the Smart Contracts are (as the have been described in the proposal of this project): Schema, Logic, Counterparties, External Sources, Ledger, Contract Binding.

The PoSeID-on Platform privacy aware design allow different parties to get the right data at the right time preserving their users' privacy. The goal to achieve this was in fact, because of the internal permission management design which is backed by the blockchain smart contracts. These smart contracts are the key for a proper permission management, check and lookup so every sensitive operation is made within them.

The list of the permission related Smart Contracts supported operations has led the development phase. It is shown below these lines:

Request PII Permission

Request: The content of the request contains the requested data type from an optional pre-defined list. Otherwise, the permission data type will be set as lowercase raw string (i.e. bank account, phone, email, address, blood type, etc.) identifying the type of permission requested. The Permission request will also contain the time when that permission will stop having effect and a field with the description of the finality of the data.

Response: The request returns an OK/NOK for the storage of the request, since the permission will be pending until the Data Subject grants it. At this point, permission will be set to REQUESTED status

Grant PII Permission

Request: The content of the request contains the data type to grant (i.e. phone number), the DP (Data Processor) allowed to request the data and the PII owner of granted data. In those situations where a time interval is required, timing and expiration information will be sent too. If timestamp information is needed for grant request, this information will be sent along other required parameters.

Response: The request returns an OK/NOK status codes as minimal response for those clients that use this call. NOK status will be trigger in situations such as no existent data type grant attempts, no existent DP attempt, etc. OK will be returned when DP grant request is successfully completed allowing DP to use/read requested data type.

Revoke PII Permission

Request: The content of the request contains the data type to revoke (i.e. phone number), the DP to be revoked from allowed DP members, and the PII owner of revocation data.

Response: The request returns an OK/NOK status codes as minimal response for those clients that use this call. NOK status will be trigger in situations such as, already revoked permission

attempts, no existent data type revocation attempts, etc. OK will be returned when DP revocation request is successfully completed.

PII Access Notification

Request: The content of the request contains the data type name (i.e. phone number) accessed by DP, the DP that accessed to that information and timestamp information to record on the ledger.

Response: The request returns an OK/NOK status codes as minimal response for those clients that use this call. NOK status will be trigger when no data type exists for given DP, no existent DP, etc. OK will be returned when DP saves access record successfully.

Check PII Permission

Request: The content of the request contains the data type to check (i.e. phone number) in requested Data Subject.

Response: The request returns an OK/NOK status codes as minimal response for those clients that use this call. NOK status will be trigger when no access to requested data field is given, no existent data check attempts, etc. OK will be returned when DP has permission to use/read requested user PII.

PII Access Notification

Request: The content of the request contains the data type name (i.e. phone number) accessed by DP, the DP that accessed to that information and timestamp information to record on the ledger.

Response: The request returns an OK/NOK status codes as minimal response for those clients that use this call. NOK status will be trigger when no data type exists for given DP, no existent DP, etc. OK will be returned when DP saves access record successfully.

List PII Permission stored

Request: The content of the request contains the Data Subject information data for permission enumeration (GDPR compliance information only)

Response: The request returns Data or a NOK status codes as minimal response for those clients that use this call. NOK will be trigger when no Data Subject information is recorded, etc. DS stored PII permission list will be returned otherwise.

4.2.4. Blockchain Client Flows

The clients (Data Subjects in PoSeID-on) are the ones that can only access to the Blockchain network throw a Quorum Node. These clients are responsible for composing the transactions and signing the messages, providing the needed privacy mechanisms between actors. PoSeID-on considers three different types of actors: Data subjects, Data Processors, Administrators.

In the following diagram, Figure 10, it will be described how those clients are designed and their particularities managing specific PoSeID-on users (for further details in D2.2 deeply explained).

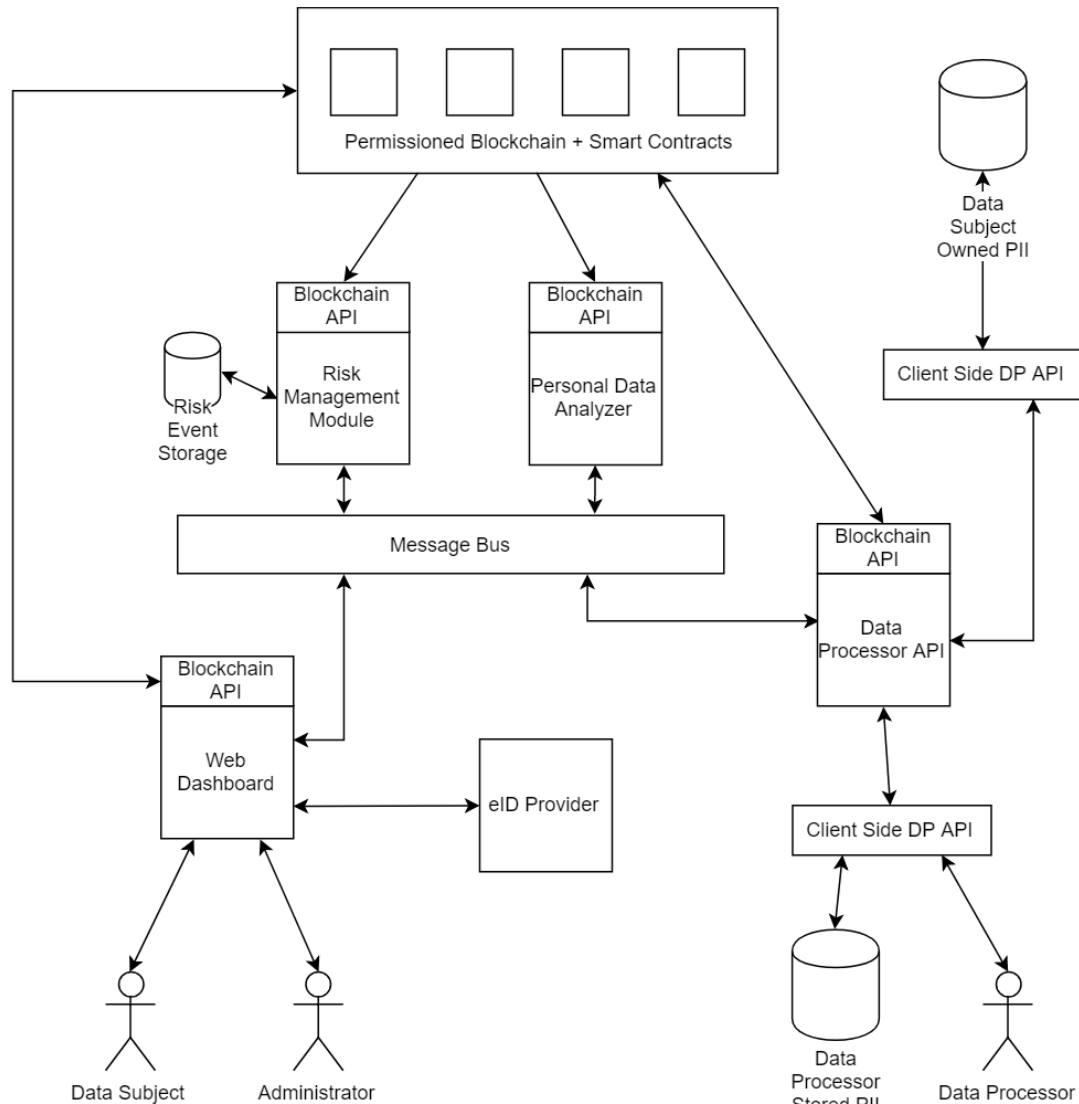


Figure 10 - PoSeID-on General Architecture

The PoSeID-on permissioned blockchain consists of a special-purpose built blockchain implementation that only works within the PoSeID-on system. It will be permissioned, meaning that instead of a proof-of-work or proof-of-stake consensus, a central authority will provide the permission to participate in the PoSeID-on Blockchain network. This will cause faster transaction speeds and will protect the implementation from the risks public blockchains are facing. The functionality of the system will be ridden by Smart Contracts stored within this blockchain, and they will describe the management of the requests and permissions to grant, deny and check PII access. The blockchain nodes are expected to be hosted by the PoSeID-on administration entity and the participating data processors. It is also possible to allow the inclusion of nodes hosted by other entities – which might make sense in some scenarios.

The blockchain API abstracts all Blockchain operations into a high-level API suitable for integration into other applications. Since the use of Blockchain carries some important implications on how

the clients and the servers behave, actions like account management (Data Processor and Data Subject identity on the Blockchain) or system functionality (Smart Contract functions usage) change drastically. For instance, users shall sign locally every call to a Smart Contract, but other components in the overall PoSeID-on architecture, like the Web Dashboard Module will function normally despite the existence of a Blockchain network behind it. Modules writing information to the Blockchain ledger must have a mechanism to access the network and work against it. This nexus will be the Blockchain API, that will also allow the intervening modules to feed the Risk Management and Data Process Analytics Module.

The Blockchain API is not a web-service, but it will be directly used as a wrapper from the client browser, without intermediaries, providing a direct connection to the Blockchain Module. This API will only be accessible once the user has been authenticated by the eID Provider Module (implementing eIDAS).

Since PII permissions and transactions are both a form of PII as well, there is a need of protecting their confidentiality and integrity. Therefore, a Blockchain platform implementation that allows private communications between parties is necessary. To this aim, PoSeID-on solution will rely on a permissioned Blockchain implementation. This means that, in a permissioned way, each deployed peer will belong to a well-known party, and that party can only participate in the consensus of data validation for the operations that are privately shared to it. This will separate ledger (data) access permission not only virtually, but also physically. This mechanism raises the security of private information.

In PoSeID-on, since every Web Dashboard and Data Processor components (and their related API components) belong to a well-known party, it is possible to find a scheme of Blockchain participants. Each of these participants will be aware of a certain set of Data Subject's PII permissions, and then will be able to check their states.

Private contracts and transactions refer to a set of parties. A party can be understood as an organization and its related infrastructure (basically a Quorum node and a Constellation in charge of the process of the private transaction). By restricting the user accounts that can access the party (that is, by defining and guarding an organization), the privacy is transferred to final users.

Organizations participating in the PoSeID-on Blockchain permissioned network are conceptual entities who have permission to maintain pieces of information and interact with other organizations through the distributed ledger.

These organizations are responsible for maintaining their Quorum Party infrastructure which executes the Smart Contracts, maintains the block chain and updates the public and private states.

Quorum manages the privacy of transactions between parties, but the association between parties and user accounts is out of the scope of Quorum itself. Thus, in order to support the concept of organization as referred to a group of authorized participants, a module that associates user accounts to organizations and manages the access of the account to the Quorum party is needed. This module is named Account (Access Control) Manager in the architecture.

In Quorum, permissions can be managed at the individual node level by the "--permissioned" command line flag when starting the node. "If the "--permissioned" flag is set, the node looks for a file named `<data-dir>/permissioned-nodes.json`. This file contains the whitelist of enodes

that this node can connect to and accept connections from. Therefore, with permissioning enabled, only the nodes that are listed in the *permissioned-nodes.json* file become part of the network. If the “—permissioned” flag is specified but no nodes are added to the *permissioned-nodes.json* file then this node can neither connect to any node nor accept any incoming connections.”.[30]

A technical list of interaction flows from and to the BC API is specified by diagrams according to the requirements gathered on the WP2 (summarized in “PoSeID-on_System_requirements_v6.xlsx”)

4.2.4.1. Blockchain API authentication[31]

For authentication, each allowed participant will have each own key-secret data. For authorization however, each call made through Blockchain API will have a token/secret that will grant or access caller to respective endpoints (Figure 11).

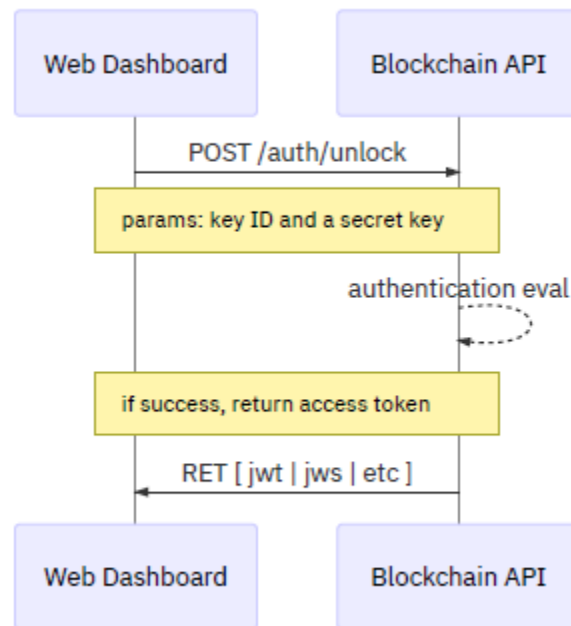


Figure 11 - BC API authentication diagram

Authentication can be made via HTTP headers or query parameters, as shown in following example[32] (added for clarification):

<https://domain.tld/path?id=20a5abac-3519-4b9f-8800-cc8f0808b2b3&secret=aB7nV3nE7uI2nP6w2fC2bH0uX0rB4dH0nL1vU0hY5sV4iA42A>

4.2.4.2. New Data subject registration

This diagram shows how the Web Dashboard or the Data Processor API will register a new 'Blockchain Data Subject' via the BC API (Figure 12). This process involves the creation of a key pair and recovery seed.

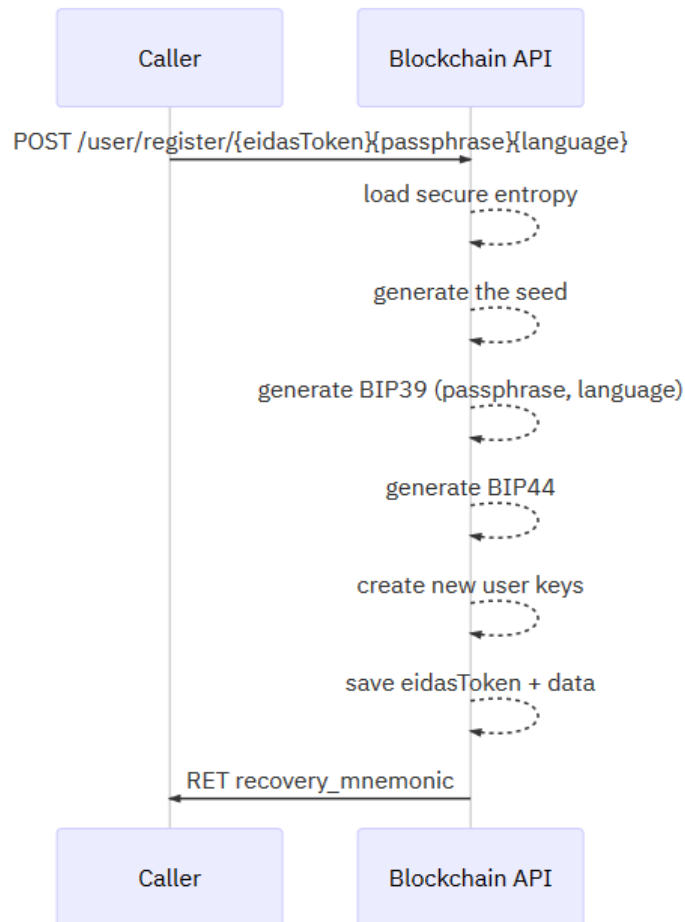


Figure 12 - New Data Subject registration diagram

Note: If a user loses the device, he or she can recover its account providing the content of 'recovery_mnemonic' which is a set of random words like:

void come effort suffer camp survey warrior heavy shoot primary clutch crush
open amazing screen patrol group space point ten exist slush involve unfold

Note about the key creation: Web Browsers does not have a 'cryptographically' secure entropy source, so we cannot leverage on them account generation process, unless, web dashboard implements some 'entropy generation mechanism' such as capturing user mouse movements from a period of time, then converting to a byte array and after that, generating the proper seed and all related cryptographic material.

4.2.4.3. Data subject account recovery/regeneration

This diagram in Figure 13 shows how the Web Dashboard, or the Data Processor API will start the recovery process for previously registered user via Blockchain API.

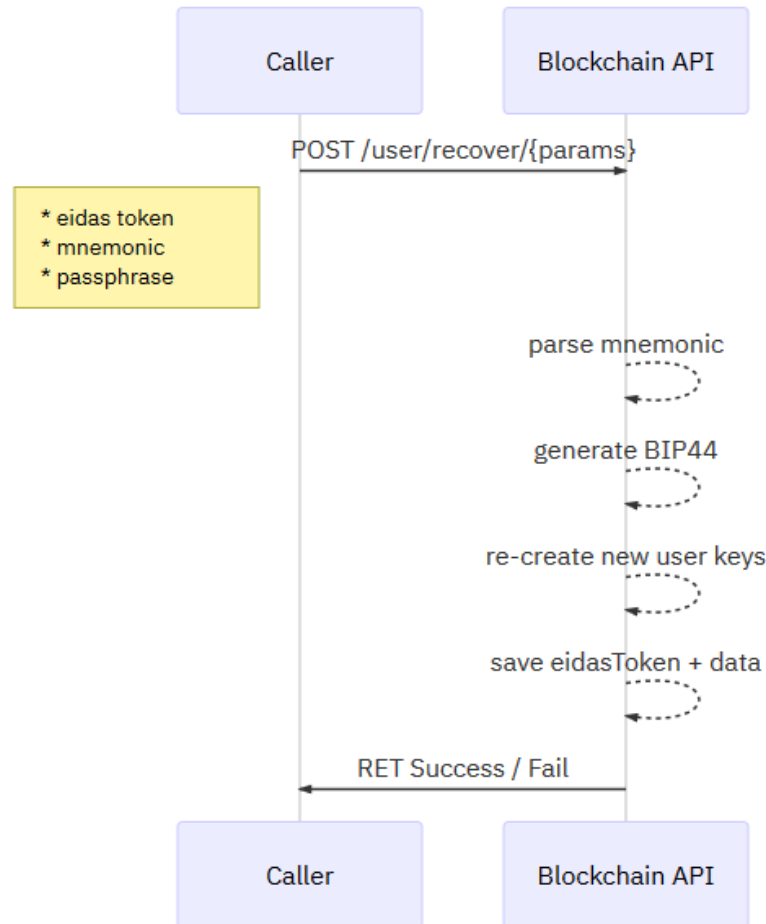


Figure 13 - Data Subject account recovery/regeneration diagram

When having a successful result, the expected situation would be same as registering new user, but instead, no new seeds nor accounts are generated.

4.2.4.4. List access permissions granted by Data Subject

This diagram (Figure 14) shows how the Web Dashboard should interact with Blockchain API.

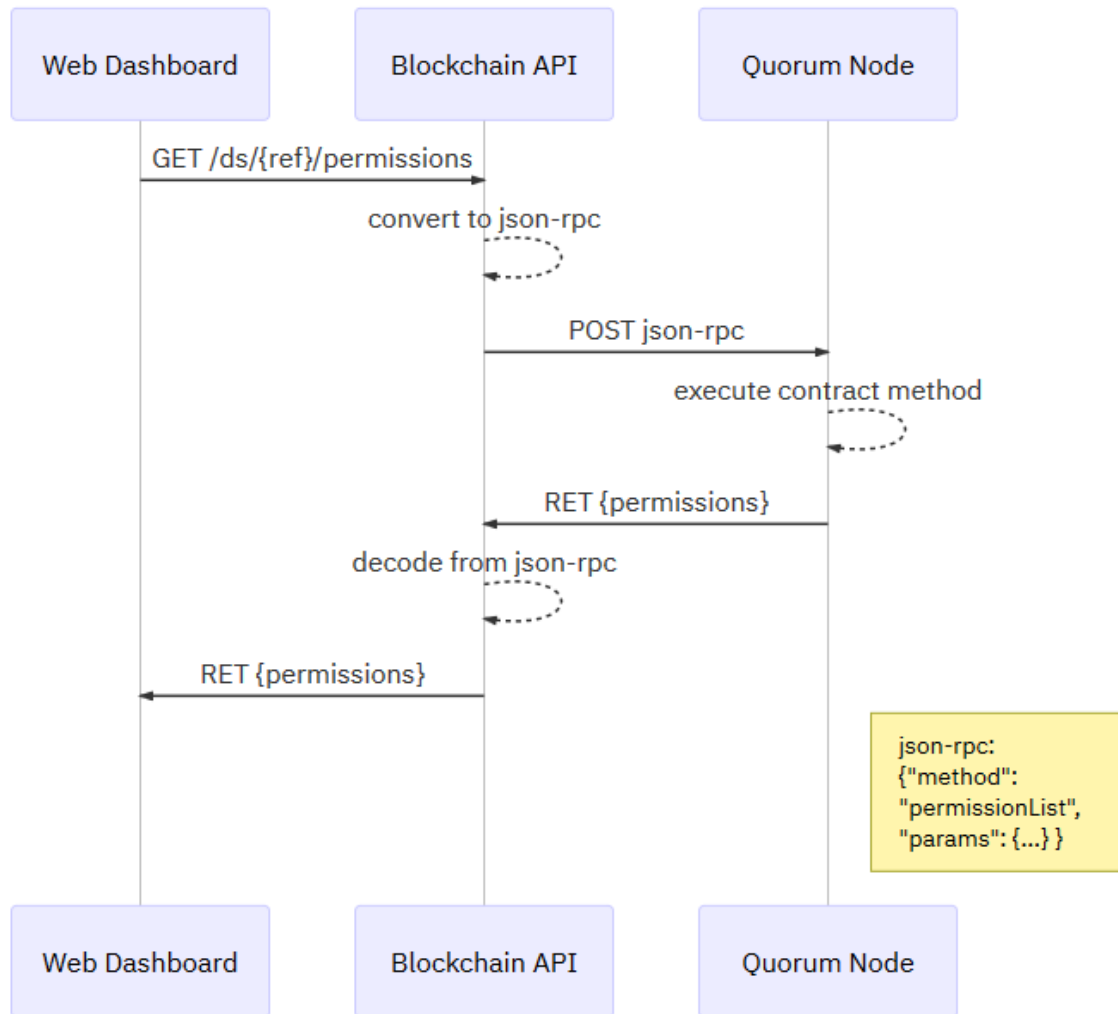


Figure 14 - Access permissions granted by Data Subject diagram

Notice that for a successful information retrieval, the Web Dashboard must provide REFERENCE/TOKEN/ID/UUID/ADDRESS or a similar identification to the Data Subject to be able to find the Data Subject information later. For security reasons complying the GDPR, Data Processor API nor Blockchain API have currently any mechanism to know which Data Subject is being called, unless it is present in the request body itself.

4.2.4.5. Grant access permissions to Data Processor

This diagram (Figure 15) shows how the Data Processor should interact with Blockchain API to grant access permissions.

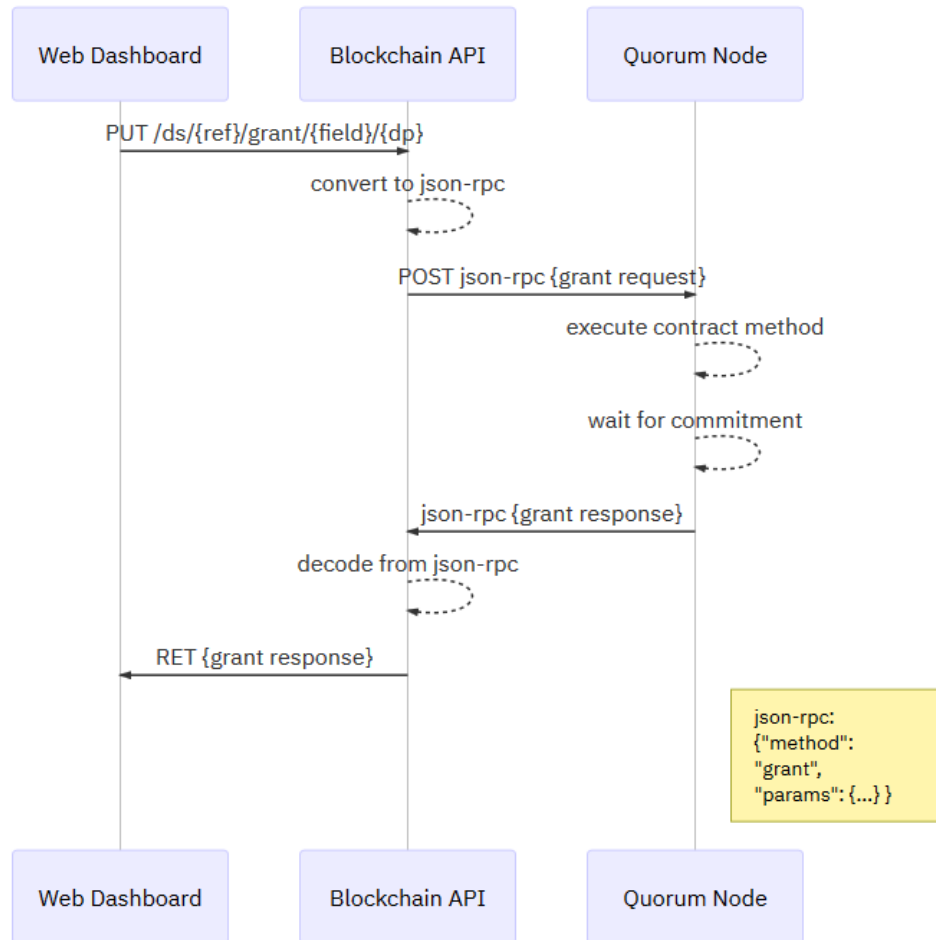


Figure 15 - Grant access permissions to Data Processor diagram

Notice that for a successful information retrieval, the Web Dashboard must provide REFERENCE/TOKEN/ID/UUID/ADDRESS or a similar identification to the Data Subject to be able to find the Data Subject information later. For security reasons complying the GDPR, Data Processor API nor Blockchain API have currently any mechanism to know which Data Subject is being called, unless it is present in the request body itself.

4.2.4.6. Revoke access permissions to Data Processor

This diagram (Figure 16) shows how the Data Processor should interact with Blockchain API to revoke access permissions.

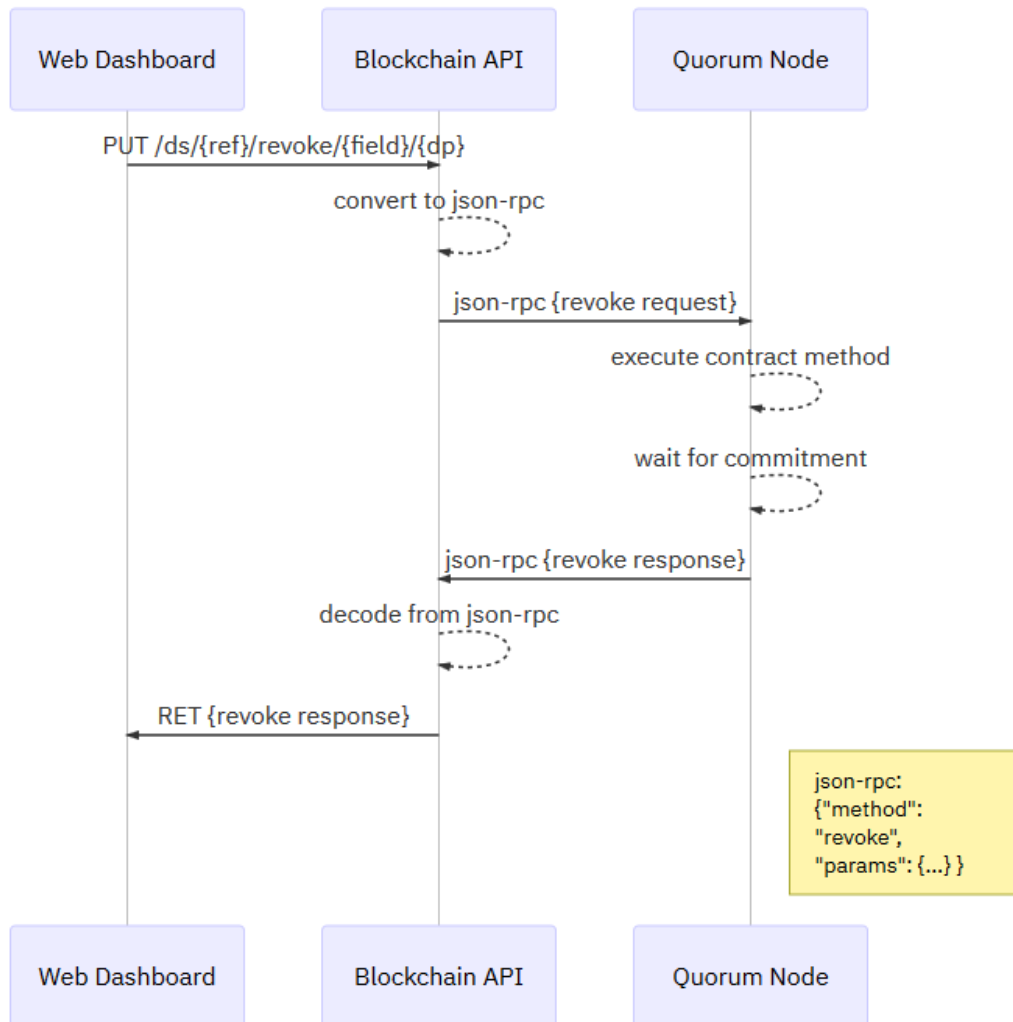


Figure 16 - Revoke access permissions to Data Processor diagram

Notice that for a successful information retrieval, the Web Dashboard must provide REFERENCE/TOKEN/ID/UUID/ADDRESS or a similar identification to the Data Subject to be able to find the Data Subject information later. For security reasons complying the GDPR, Data Processor API nor Blockchain API have currently any mechanism to know which Data Subject is being called, unless it is present in the request body itself.

4.2.4.7. Request access permissions to Data Processor

This diagram (Figure 17) shows how the Data Processor should interact with Blockchain API to request access permissions.

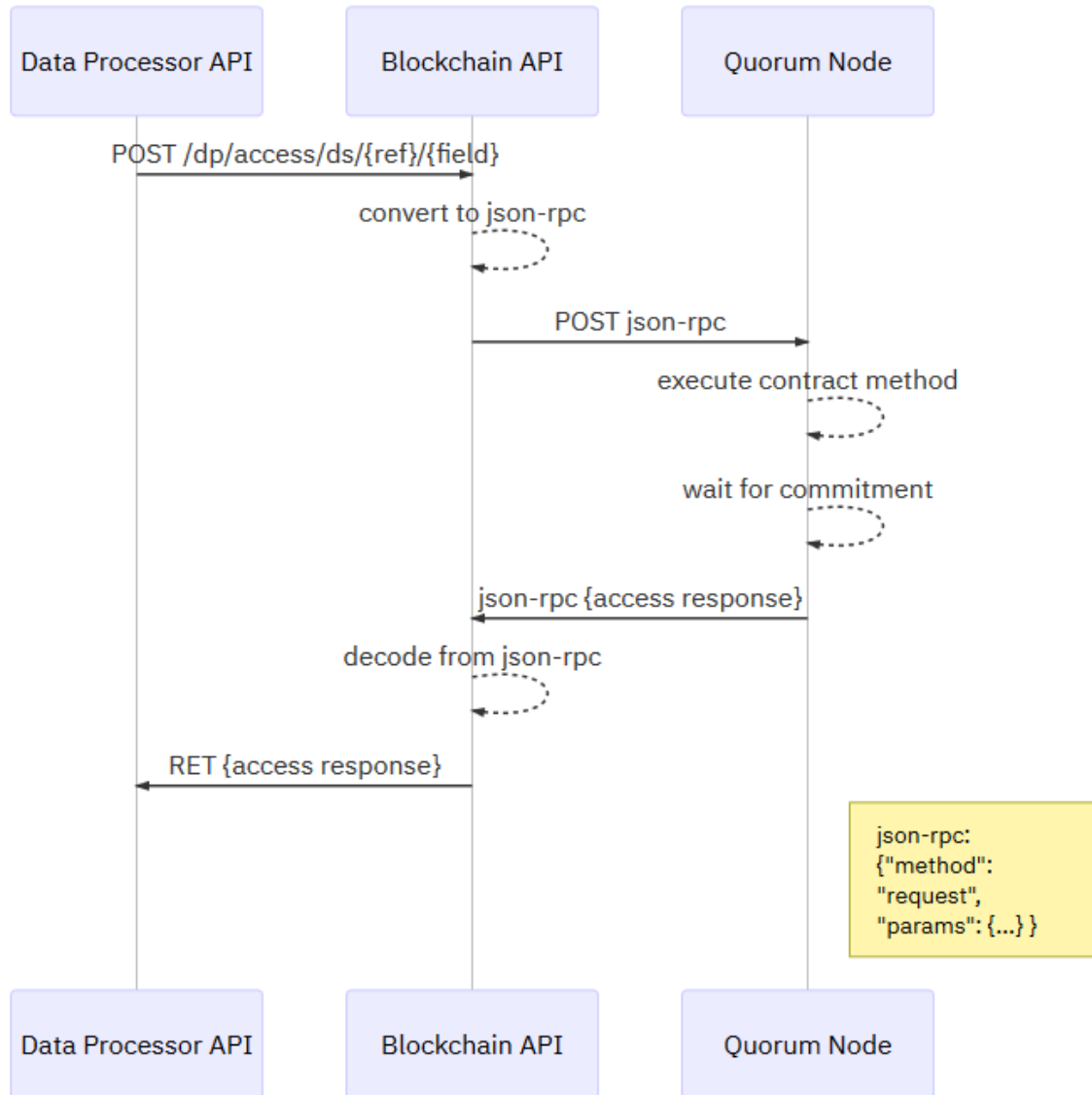


Figure 17 - Request access permissions to Data Processor diagram

Notice that for a successful information retrieval, the Web Dashboard must provide REFERENCE/TOKEN/ID/UUID/ADDRESS or a similar identification to the Data Subject to be able to find the Data Subject information later. For security reasons complying the GDPR, Data Processor API nor Blockchain API have currently any mechanism to know which Data Subject is being called, unless it is present in the request body itself.

4.2.4.8. Batch permissions interaction to Data Processor

This diagram (Figure 18) shows how the Data Processor or Web Dashboard should interact with Blockchain API to execute batch permissions related operations.

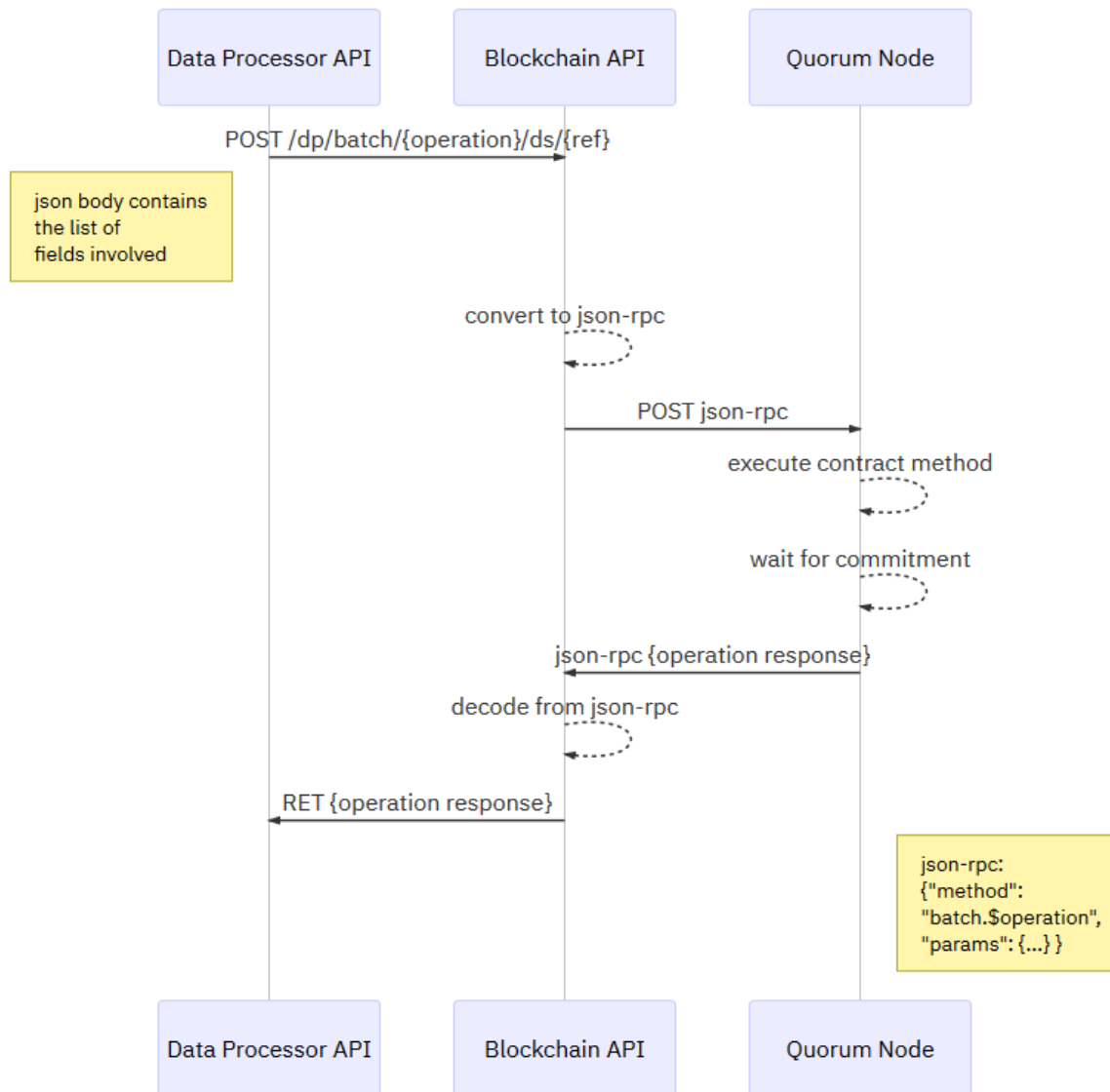


Figure 18 - Batch permissions interaction to Data Processor diagram

Notice that for a successful information retrieval, the Web Dashboard, or the Data Processor API must provide REFERENCE/TOKEN/ID/UUID/ADDRESS or a similar identification to the Data Subject to be able to find the Data Subject information later. For security reasons complying the GDPR, Data Processor API nor Blockchain API have currently any mechanism to know which Data Subject is being called, unless it is present in the request body itself.

4.2.5. Data Processor API

The Data Processor API is the API responsible for communication between the PoSeID-on platform's components and external organisations. In PoSeID-on case, these are always Data Processors; entities enlisted to process, among other things, personally identifiable data.

The data processor API follows a very modular design (see Figure 19). It splits up the API into two components; an API server and an API client. This approach has a number of benefits. Since

the data processor API harbours a fairly large amount of complicated business logic, it hampers the onboarding of new organisations onto the PoSeID-on platform. By supplying both the API server and the API client, this onboarding process is made less obtrusive. Supplying the entire API client also allows the project to push more sensitive data processing into the data processor's security domain. Since all communication is, by design, end-to-end encrypted, the API client is responsible for one end of the encryption. The other end can be any other PoSeID-on subsystem.

To simplify the design of both the API client and the API server, the design of both will follow an event- and message-based approach. This means that any number of both API server and API client instances can run for any associated Data Processor to support the message handling capacity needed.

To even further simplify the design of the data processor API, each associated data processor gets its own endpoint and, by extension, its own (set of) API servers.

Communication between the Data Processor API server and API client must happen over a mutually authenticated, encrypted connection. From a security point of view, based on the security-in-depth philosophy, this is a hard requirement. Even though the messages between components and the data processors are end-to-end encrypted, the transport of those messages needs to be authenticated to prevent any message spoofing attacks. Mutually authenticated TLS (mTLS) is the most obvious, widely adopted, solution for transport security.

On top of the mTLS connection, the messages can be passed bi-directionally. Messages for the message bus are not the only messages that are sent over this connection. The API client also needs access to the functionality exposed by the blockchain API. The blockchain API access is needed to give the data processor API client the means to verify whether access to certain PII should be granted before sending it over to the requesting party through the data processor API server. To encapsulate two very different protocols in one transport, some encapsulation/multiplexing is needed. Since there is a working mTLS connection available, a simple solution using WebSockets is all that is needed.

The data processor API server, with these restrictions, now is a simple service. It listens on the message bus for any incoming messages on its own message queue. This message queue is a data processor-specific message queue. All communication with this particular data processor from other PoSeID-on components, including other data processors, will flow through this queue. Every message is simply forwarded to one of the connected clients. Because operations on an AMQP (Advanced Message Queuing Protocol) queue are, by protocol design, atomic, this creates a simple FIFO load balancing structure on the transmit side of the communication. On the receiving end of the message, the API server simply forwards the message from the API client onto the right recipient queue. Communication with the blockchain API can be done in a similar fashion.

The data processor API client consumes the aforementioned WebSocket protocol and takes care of all the protocol-specific plumbing; message construction, verification, encryption and error handling. For ease of integration, the API client then, on its turn, needs to expose an API that is much more easily integrated into existing systems. The exposed API will need to use a protocol that is both readily available in many programming languages/frameworks, bi-directional and

inherently message-based. The go-to protocol in this scenario is WebSockets; the same protocol family used between the API server and API client. The protocol itself, however, is much simpler.

The way this API can be integrated is by allowing only local connections to this WebSocket API endpoint. Either by means of a firewall or, preferably, using a local UNIX socket with the right access restrictions. Since the number of API client instances does not matter, it allows for a simple one-client-per-machine or even a one-client-per-process deployment scheme on the data processor's systems.

The assumption made in this scenario is that the data processor's systems can handle any synchronisation needed to keep the protocol working. Incoming messages might warrant a response or, conversely, might be responses to messages sent before. The deployment scenario assumes an ACID-compliant RDBMS or something equivalent takes care of the association by keeping state.

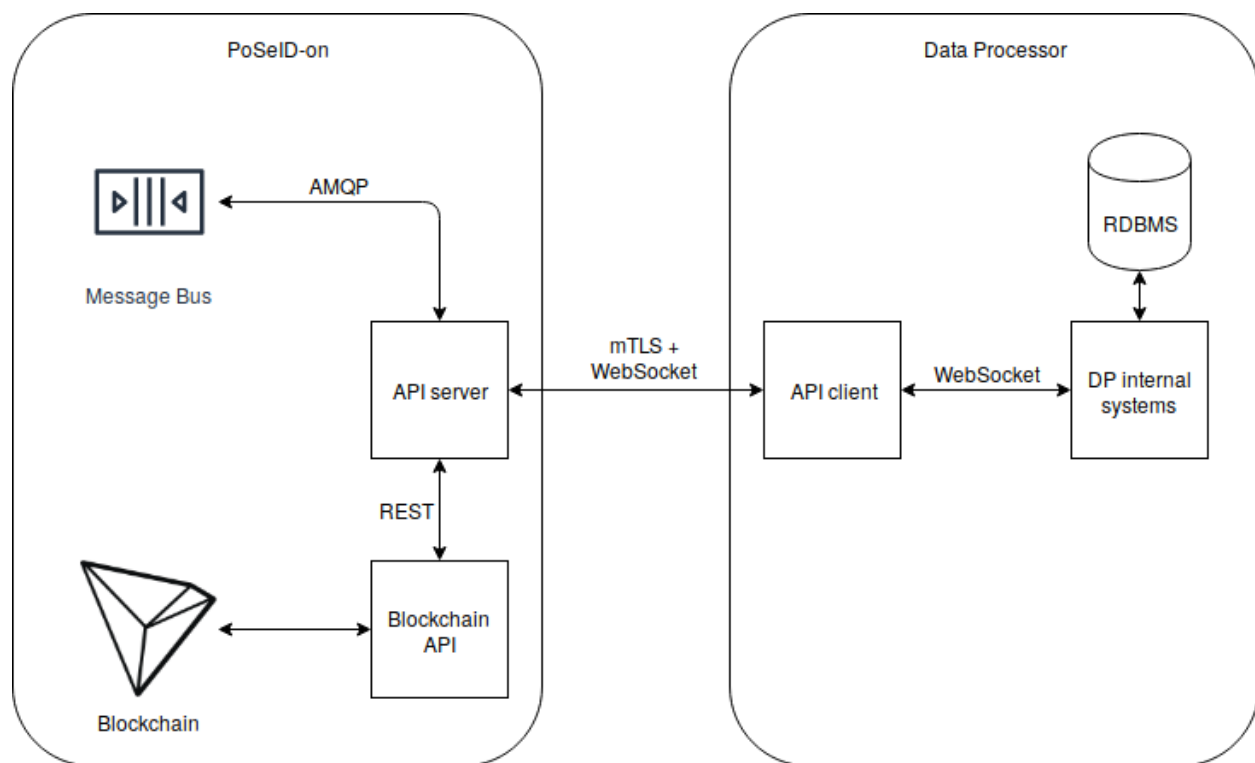


Figure 19 - Communication between PoSeID-on and the Data Processor API

4.3. Development status

This section briefly exposes the important milestones and how they have been organized and achieved. The main issues are denoted by the delay of the ending date compared to the initially intended due date. At this point is important that no major issue has been a problem to the right finalization of other tasks or the deadline for that task itself.

Every WP3 technical task had to be finished for the project General Assembly that took place on 2019, July 9th at Coimbra (Portugal).

Some of the sub-tasks (from T3.1, T3.2 and T3.4) suffered a delay. The initial due date proposed for each task is marked in the Table 3 (its legend in Table 4). The green color is reserved for the week that the task was finally achieved. When the sub-task was finished on time the yellow color doesn't appear.

The impact was minimized because the only need between tasks was that the T3.3 should be completed before the T3.1 works started.

The cryptographic key management system was delayed because it needed decisions that were taken the week of 2nd July. That sub-task is still under revision until the all-technical-components integration is finished, being affected by components and works from WP4 and WP5.

Finally, PoSeID-on Smart Contracts were developed on time for the 4th June, but the testing phase took three more weeks to allow the testing along the Access Manager and the BC Client.

Table 3 - Development status

TASK	21M	28M	4J	11J	18J	25J	2JI	9JI
T3.1: Privacy ensuring Permissioned BC implementation								
Deploy the network nodes and make them available in the cloud								
Deploy the cryptographic key management to enable identification of the parties								
Design an Access Manager to enable the permissioned blockchain								
T3.2: Private data managing smart contracts implementation								
Research Smart Contract generation and testing/verification mechanisms								
T3.3: BC Client implementation								
Design and implement the Blockchain client that will serve as the main point of access to the Blockchain network								
Design and implement an API to allow other PoSeID-on Modules access the Blockchain network								
T3.4: DP API implementation								
Implement the API module that runs on cloud and is able to manage all API requests from all services linked to the PoSeID-on platform								

Table 4 - Legend

	Upcoming week
	Past week
	Final submission week
	Coimbra GA
	Upcoming due date
	Past due date
	Finished task week

4.4. Security remark

Blockchain general philosophy is thought to permit the traceability of the user actions, pseudonymizing their identities and securing their information by cryptographic means. But these mechanisms ensure pseudonymization instead of anonymization for Person Identifiable Information.

The proposed ledger management alternative approach is based on “burnable pseudo-identities” which will ensure that the PII of the Data Subject (the user of services managed by PoSeID-on) is not traceable by anyone. The aim of this proposal is to allow user interactions in a way that can be traceable over the time while the user is using PoSeID-on services and can be compliant with GDPR and “right to be forgotten” when the user requests it.

Moreover, with the application of Blockchain, permissions over Data Subject’s PII and transactions of Data Subject’s PII between Data Processors will be stored in a distributed ledger that allows participant organizations to share Data Subject’s information and yet to share a mechanism to protect that data from being flowed to non-participating organizations. Each of the participant organizations will be a peer participating in the distributed ledger by keeping a private ledger synchronised with the rest. Due to the fact that PII permissions and transactions are both a form of PII as well, there is a need of protecting their confidentiality and integrity.

The PoSeID-on ledger will be based on a permissioned blockchain technology, with a central authority providing the right access to participate as PoSeID-on member. This will cause faster transaction speeds and will protect the implementation from the risks public blockchains are facing.

The functionality of the system will be ridden by Smart Contracts stored within this blockchain, and they will describe the management of the requests and permissions to grant, deny and check PII access.

The blockchain API abstracts all Blockchain operations into a high-level API suitable for integration into other applications. This API will only be accessible through a token/secret that will grant or access caller to respective endpoints.

An important factor responsible for the communication between the components of the PoSeID-on platform and external organizations is the data processor API.

The solution adopted for the data processor API is to split up the API into two components; an API server and an API client. This approach has the following security benefits:

- hampers the onboarding of new organisations onto the PoSeID-on platform. By supplying both the API server and the API client, this onboarding process is made less obtrusive.
- Since all communication is, by design, end-to-end encrypted, the API client is responsible for one end of the encryption. The other end can be any other PoSeID-on subsystem.
- each associated data processor gets its own endpoint and, by extension, its own (set of) API servers.
- Adoption of mutually authenticated TLS (mTLS) for the secure communications between the Data Processor API server and API client
- the data processor’s systems can handle any synchronisation needed to keep the protocol working.

In conclusion, the security of the entire platform is guaranteed, in addition to the intrinsic algorithms of the blockchain, also by conditions that safeguard both the PII and the blockchain itself from external attacks.

5. Initial Prototype of Blockchain Functionality

The heart of the PoSeID-on decentralized platform for user data privacy protection will be designed over a Blockchain network, Smart Contract management driven and accessed by known entities, but opened to every PoSeID-on platform user, so the data will be protected but transparent at the same time.

The development will be led by the design described at this deliverable, according to the proposed architecture and in accordance with the integration with other PoSeID-on modules.

In this section, those principles and developments will be described.

5.1. Smart Contracts and Blockchain functionalities

Along this subsection, Smart Contracts and Blockchain functionalities are going to be explained.

5.1.1. Smart Contracts

Smart Contracts on PoSeID-on have been implemented following the KISS principle. A specific structure has been designed to allow reusing functions for all the use case scenarios. In that line, Solidity good practices have been adopted due to the peculiarities of the EVM. This has shaped the functions and the way they are called/used.

There are several secondary actions, routines and helpers created to design optimized Smart Contracts on PoSeID-on, leveraging the execution of non-trivial operations, such as:

- Multi attribute-based information lookup
- Data concatenation
- Permission state management
- Input data validation

5.1.1.1. Internal Permission Model and Flow

PoSeID-on Permission model has been developed in a way that Data Processors can't become rogue Data Processors nor spammer Data Processors.

According to the internal Smart Contract model design, Data Processors are only allowed to REQUEST a single permission request for only one service and field at a time. This permission can be updated to notify a state change and set the permission to ALLOWED, DENIED or EXPIRED. To better understand the possible state transitions managed from the Blockchain in an auditable (but still private) manner, is shown in the flow diagram from Figure 20.

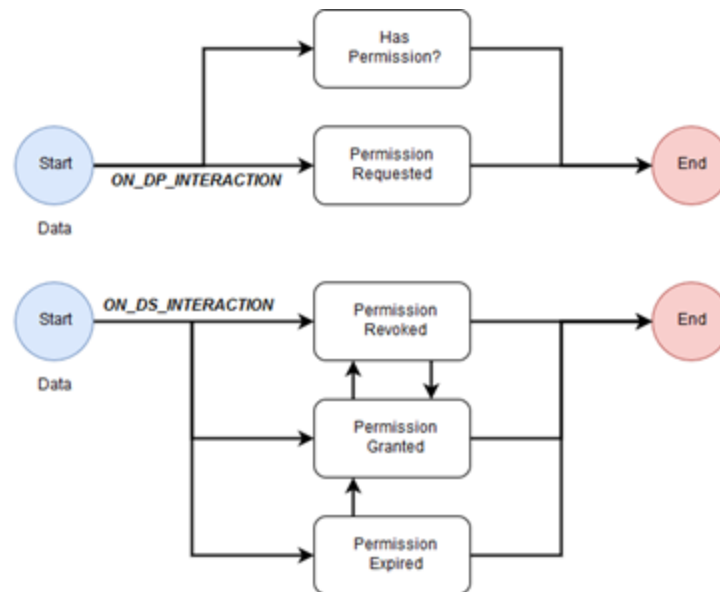


Figure 20 - Internal Permission Model and Flow

Each status has one purpose. Among all, the permission life makes sense (Table 5).

Table 5 - Permission Status / Description

Permission Status	Description
<i>REQUESTED</i>	A DP makes a new permission request for service X and Field Y. This permission is stored in the ledger in order to be resolved by its target DS
<i>ALLOWED</i>	The permission target DS grants the permission and DP will be notified in next attempt that it is allowed to use that DS data.
<i>DENIED</i>	The permission target DS revokes the permission and DP will be notified in next attempt.
<i>EXPIRED</i>	The stored permission data is flagged as expired by initial permission conditions. Therefore, DP is notified about this expiration

Finally, while the Smart Contract code designing phase, two main potential-and-non-trivial scenarios have been considered:

- In the first one, a Data Subject can revoke a specific permission. Later, its action can be rolled back, allowing the Data Processor to access that information again for a specific service without having asked for it.
- In the other one, a Data Subject could grant twice an already expired permission.

5.1.2. Blockchain Functionalities

In order to be able to operate with the Blockchain using Smart Contracts, some permissions have been implemented (following the description in D2.2). There are some strategies which allows to redirect the execution from one entry Smart Contract to different backend Smart Contract, but mainly it will be done by giving governance operations to the entry Smart Contracts. Just specified accounts will be able to implement maintenance actions.

It was stated at the WP2 that "field" is an open field. It is not "single data" nor "series of data". The Data Subject grants or revokes the data requested by the Data Processor, whatever it is.

A set of operations will be permitted to be done in blockchain with a decentralized execution view (Table 6). This set of operations is the functionality face for the blockchain clients and thus the Data Subject and Data Processor operation entry point. These operations will include: Request PII Permission; Grant PII Permission; Revoke PII Permission; Check PII Permission; and PII Access Notification. Next, we discuss each operation.

Table 6 - BC Functionalities

Function name	Description	Status
grant_permission	grant permission for triplet [DP, DS, field_name]	done
revoke_permission	revoke permission for triplet [DP, DS, field_name]	done
request_permission	request permission for triplet [DP, DS, field_name]	done
check_permission	check permission for triplet [DP, DS, field_name]	done
access_notification	write access attempt for permission with [DP, DS, field_name]	done

5.1.2.1. Request PII Permission

According to the GDPR, Data Processors need each Data Subject permission to collect and manage his/her PII. A function will provide the appropriate mechanisms for a Data Processor to request that kind of permission (Table 7).

Table 7 - Request PII Permission

Action defined in D2.2	BC API access (short codes)	Smart contract method	Status
Request PII values from one Data Subject	None (it's done via Data Processor API)	is_granted(dp, ds, field)	done
Request access to PII from one Data Subject	GET /dp/request/:ds/:field	request_permission(dp,ds,field)	done

5.1.2.2. Grant PII Permission

When a specific permission is requested for a specific Data Subject, the Data Processor needs to wait for that permission to be granted by the Data Subject. The operation timestamp will denote the starting time for the permission to take effect, and the permission will automatically stop having effect when the requested end time is reached. Therefore, this operation will be linked to one previous permission request (Table 8).

Table 8 - Grant PII Permission

Action defined in D2.2	BC API access (short codes)	Smart contract method	Status
Grant access to PII to one Data Processor	/ds/:uid/permission/grant/:dp	grant_permission(dp,ds,field)	done

5.1.2.3. Revoke PII Permission

Every permission granted by a Data Subject has a validity period, denoted by the time when the Data Subject grants the permission and the permission end time. But during that period, the Data Subject can decide to revoke intentionally a specific permission for a specific Data Processor. Permission revocation, linked to a permission request (Table 9).

Table 9 - Revoke PII Permission

Action defined in D2.2	BC API access (short codes)	Smart contract method	Status
Revoke access to PII from one Data Processor	/ds/:uid/permission/revoke/:dp	revoke_permission(dp,ds,field)	done

5.1.2.4. Check PII Permission

Each time a Data Processor need to use (e.g. access or management) PII from a Data Subject, the PII must be requested to PoSeID-on and the permission for the specified action will be checked against blockchain. This one will be a “get” operation that returns whether a Data Processor has permission to access specific PII or not. This operation is embedded by the PII Access Notification because every time an access is notified, its corresponding permission will be checked (Table 10).

Table 10 - Check PII Permission

Action defined in D2.2	BC API access (short codes)	Smart contract method	Status
Check PII Permission	/dp/:uid/permission/status	check_permission(dp,ds,field)	done

5.1.2.5. PII Access Notification

Since Data Subject information is not stored by the ledger, the information is not securely managed from the inside of the blockchain system. The information is provided in a way that blockchain can't control its access by the decentralized mechanisms used to hide the information to outsiders. Because of this, PII access by the Data Processors (including both successful data reception and unsuccessful data requests) will have to be notified to blockchain by other PoSeID-on components. This operation will notify the Blockchain Module. When a Data Processor is trying to use PII protected by a specific permission, the access will be granted or denied, and it will be stored in the ledger. This operation will be used to identify anomalous behaviour in the network by Data Processors (Table 11).

Table 11 - PII Access Notification

Action defined in D2.2	BC API access (short codes)	Smart contract method	Status
PII Access Notification	not accessible from REST	notify_access(dp, ds, field)	done

5.1.2.6. Other non-related functionalities

There are some actions that were defined in the deliverable D2.2 that affect the Blockchain platform data and are related to the work done by this module which has been explained along this section. But these functionalities are not directly managed by the Blockchain network. Instead of that, the Data Processor API will be in charge of performing the actions listed in the next table (Table 12).

Table 12 - Other non-related functionalities

Action defined in D2.2	BC API access (short codes)	Smart contract method	Status
Request access to PII from all Data Subjects *	N/A (it's done via Data Processor API)	none	N/A
Send data containing PII to be analysed *	N/A (it's done via Data Processor API)	none	N/A
View PII known to one Data Processor *	N/A (it's done via Data Processor API)	none	N/A
Update PII known to one Data Processor *	N/A (it's done via Data Processor API)	none	N/A

The reason of delegating the work on the Data Processor API module is because this development is affecting Personal Data itself, and not the actions or permissions related to it.

5.1.3. Key Management

This subsection will explain how the key management works in the system and how the Blockchain users (both Data Subjects and Data Processors) will be provided with their own, auto generated, unchangeable, private, recoverable and secure key set. For this purpose, a full offchain-but-distributed managed system has been created.

In order to comply with the highest security and privacy requirements, PoSeID-on members who manage a Blockchain node must individually take care of the proper management of cryptographic content and their private node management keys. This is because there is no central repository where this information is stored. Therefore, the management of the nodes, and the management of the keys, will be carried out by the data processors (DP) themselves. Moreover, due to the Quorum design, the software used for the implementation of the ledger, the management of these keys will have to be done by software.

This implies that:

- The keys cannot be stored in an HSM.
- The keys must be known by at least one physical server or virtual machine.
- The keys that belong to node, mostly those used to setup a Coinbase account, must be stored in the node itself and execute the process of unlocking on demand.

On the other hand, the keys belonging to the users (Data Subjects or DS) will be protected by the BC API component. This custody will also be done by Software and the stored content will be secured through the standard and management model of Ethereum V3 KeyStore.

5.1.3.1. Key Storage

An Ethereum V3 KeyStore also known as V3 wallet, is an encrypted version of a unique Ethereum private key that users will use to sign the transactions. If users lose this file, users lose access to their unique private key which means they lose the ability to sign and execute transactions. If no proper recovery mechanisms are designed, this process is unrecoverable.

The actual keys storage mechanism requires to encode information using at least AES-128-CTR crypto protocol in order to store the data in servers' filesystems. By default, and following conventions, keystores filenames are 128-bit UUID given to the secret key (a privacy-preserving proxy for the secret key's address) so that files will be saved as *uuid.json*. All such files have an associated password. To derive a given *uuid.json* file's secret key, first derive the file's encryption key; this is done through taking the file's password and passing it through a key derivation function as described by the *kdf* key. KDF-dependent static and dynamic parameters to the KDF function are described in *kdfparams* key.

The KDF (key derivation) function is predefined to **pbkdf2**, being PBKDF2 **kdfparams** as follows:

- *prf*: Must be *hmac-sha256*;
- *c*: number of iterations to be made in KDF routine;
- *salt*: salt passed to PBKDF algorithm;
- *dklen*: length for the derived key. Must be bigger than 32.

Once the file's key has been derived, it should be verified through the derivation of the MAC. The MAC should be calculated as the SHA3 (keccak-256) hash of the byte array formed as the concatenations of the second-leftmost 16 bytes of the derived key with the ciphertext key's contents.

An example content of encrypted storage is shown below:

```
{
  "crypto": {
    "cipher": "aes-128-ctr",
    "cipherparams": {
      "iv": "6087dab2f9fdbbfaddc31a909735c1e6"
    },
    "ciphertext": "5318b4d5bcd28de64ee5559e671353e16f075ecae9f99c7a79a38af5f869aa46",
    "kdf": "pbkdf2",
    "kdfparams": {
      "c": 262144,
      "dklen": 32,
      "prf": "hmac-sha256",
      "salt": "ae3cd4e7013836a3df6bd7241b12db061dbe2c6785853cce422d148a624ce0bd"
    },
    "mac": "517ead924a9d0dc3124507e3393d175ce3ff7c1e96529c6c555ce9e51205e9b2"
  },
  "id": "3198bc9c-6672-5ab3-d995-4942343ae5b6",
  "version": 3
}
```

5.1.3.2. Relationship between Transport Key and Blockchain Key

The target of this relationship is to identify the users without letting the Data Processors to access PII that they are not allowed to view. Apart from the Blockchain keys, which are generated and managed by the BC API and kept securely without sharing with the Web Based Dashboard and the Data Processor API, the first step for a successful key management is to be authenticated against the BC API. It will be done using a passphrase and the key (K_i) is decrypted and then used. K_i is used only once, for that moment because a rotation period (R) exists.

The transport key used in the middle, does not rotate as often as Blockchain keys, so the focus can be put on the implementation of Blockchain keys management and rotation.

For Data Processors, this transport key is stored in the Data Processor API client and supplied through a secure channel. This behaviour should be very similar to how TLS certificates and keys are handled in the HTTPS realm/handshake negotiation process.

For Data Subjects, the transport key storage mechanism will be provided by the Web Based Dashboard. As an alternative, the Data Subject keys can also be managed by the BC API, using the same symmetric key used to create the Blockchain key. After authenticating the Data Subject with the correct passphrase, the public/private keypair can be returned to be used by the Data Subject.

5.1.3.3. Key Rotation Period

The strong key rotation mechanism was promoted by the need of adding an extra security layer to power the anonymization in a Blockchain ecosystem.

By design, the majority of the Blockchain platforms are pseudonymous. This means that users don't have real identifications and their real-world identity is still unknown. But most of them allow to link the user pseudonyms to create a trend and trace of their movements in the networks. This means that a third-party viewer inspecting the network can dump and read the ledger, creating and fetching statistical results and further information about the network users.

Among the actions that an external can perform over others information includes, but is not limited to:

- How many transactions are made from one specific account
- How many transactions are made to one specific account
- Create profiles
- Detect activity periods and usage patterns
- Create relationship between Blockchain users

To cover the GDPR needs this is something that should be avoided. To create a platform committed with privacy rules and best practices, a key rotation mechanism has been designed to overcome this functionality.

The key rotation is based on well-known tested working standards in Ethereum, but specifically modelled for PoSeID-on. The target standards are:

- A Heuristic Deterministic Key Generation Algorithm
- A V3 Key Storage

In order to use a *HDKGA* using Ethereum/Quorum compliance algorithm (*secp256k1*), an HDwallet implementation such as *BIP32* needs to be implemented as part of PoSeID-on services. This process is handled by the BC API, becoming transparent to Data Processors and data Subjects.

Internally, Master Keys creation steps involves the generation of a random seed with its proper mnemonic to be used. This process is defined in following figure 21 and figure 22:

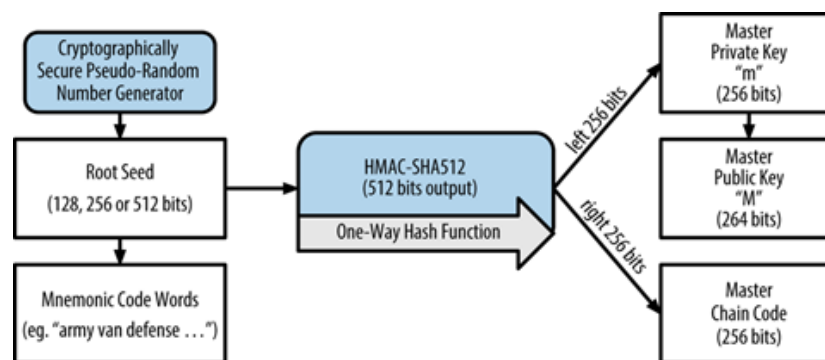


Figure 21 - Multiple account generation algorithm for Burnable Identities and Key rotation support [33]

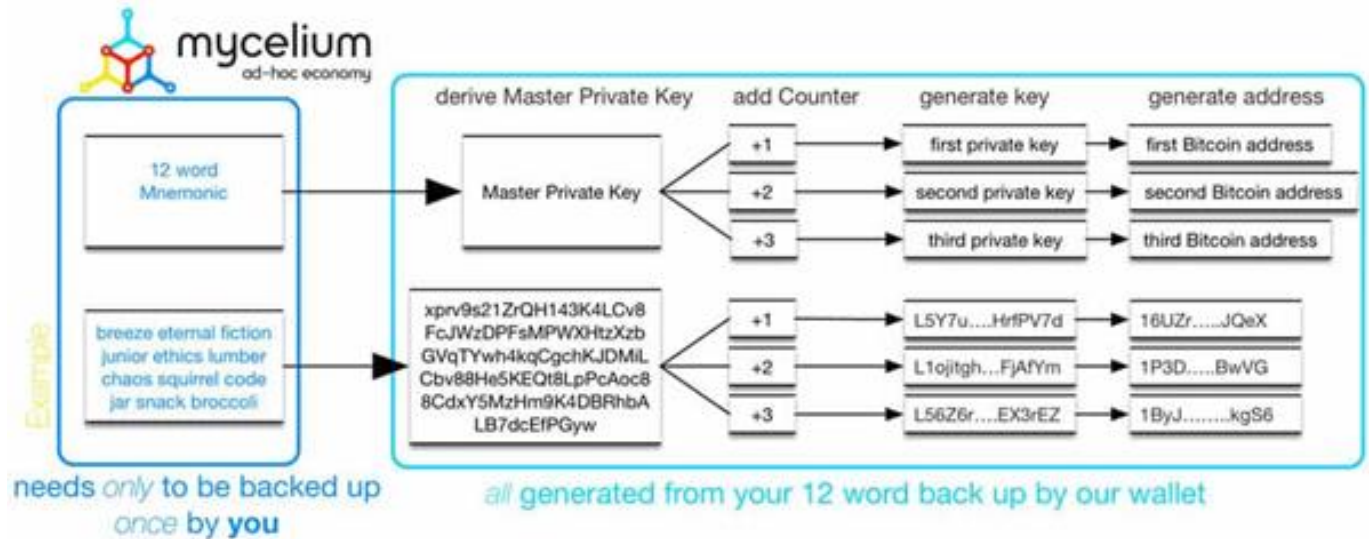


Figure 22 - HDwallet creation example [34]

5.1.4. Blockchain Client

Any external system that needs to be queried in order to get some information requires two things:

- A protocol of communication
- A client to communicate with

In our current scope, PoSeID-on will rely on quorum ledger requirements for both communication protocol and client implementation. This means that communication protocol will be JSON-RPC communication over HTTP connection. Thus, quorum clients will only need to be compliance with HTTP standard defined at Hypertext Transfer Protocol (RFC 2616) and encapsulate their messages as JSON-RPC following the standard (Figure 23).

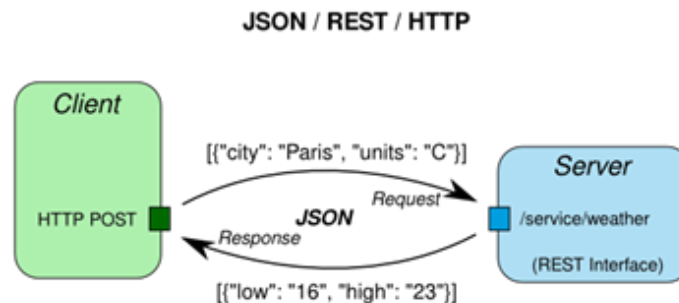


Figure 23 - JSON-RPC message exchange example

This behaviour must remain the same independently of the client which request connection to ledger nodes. It means that all devices, smartphones, desktop applications, etc, will have to

communicate using this standard for those situations in where a direct peer communication is required (Figure 24).

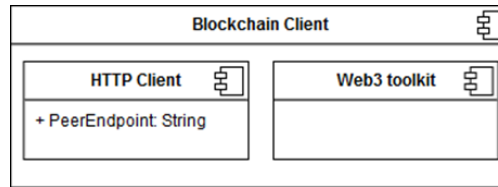


Figure 24 - Default Blockchain Client Implementation for Standard Quorum Implementation

In PoSeID-on, Blockchain complexity is encapsulated over a RESTful API so that, end users and clients will only need to be compatible with REST APIs (Figure 25).

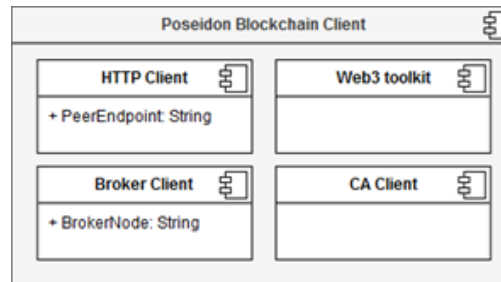


Figure 25 - Blockchain Client Implementation for PoSeID-on platform

The implemented Blockchain API formal description is as documented in the Annex I.

6. Integration in PoSeID-on architecture

The overall integration of the PoSeID-on modules, as it is in a current state, it is described in D4.1 (Section 6), but, in this document, it is described a brief introduction regarding the integration done on several levels.

6.1. Packaging

The web-based dashboard components are built using two language runtimes; Python and Node.js. Both language runtimes get their own OCI base container image, inheriting from the common PoSeID-on base image, in order to centralise all common dependencies between components. This helps with centrally rolling (security) updates across the entire platform. Every component gets its own git repository, containing the source code and accompanying documentation.

Versioning of container images follows the git branching model (not commonly done). The git branch name is used as the tag. End users do not need any configuration changes to update to the latest stable container image; they only need to pull updates and restart their containers.

Container images can be automatically built and published to the image registry out of the source code repositories by simply using the supplied Makefiles (for more information, see Section 6.1 from D4.1).

6.2. Configuration

Following 12-factor standards and the choice for Kubernetes, all configuration of the software making up the components happens through environment variables. From the get-go all variables, including secrets, are supplied through the runtime environment.

This means that no secrets or default configuration can be found in the source code repositories. However, there will be some documentation about what environment variables will be used and what the values should be.

Configuration about how the application should be started is done through the Dockerfile, which is part of the OCI container image.

The final part of the configuration, the deployment settings, are supplied using Kubernetes configuration files in the yaml format. For development purposes, patches using *kustomization* are applied that make the components behave in a more contained way.

6.3. Communication

For communication with the rest of the platform, the web-based dashboard accepts messages for data subjects and it sends messages to other components on behalf of the data subjects. All the communication flows through the message bus (see Figure 26 and Figure 27).

The message bus protocol specifies that the name of the AMQP queue is derived from the recipient's public key. However, having a queue for each data subject would run into scaling issues. Therefore, the dashboard listens on a single queue named dashboard on behalf of all data subjects. As specified before, the queue will be consumed into a Redis cache immediately on message arrival.

The only component that is not accessed by the message bus is the blockchain API. For communication with the API, a separate library still needs to be developed. At the moment of writing this document, the API is not available yet.

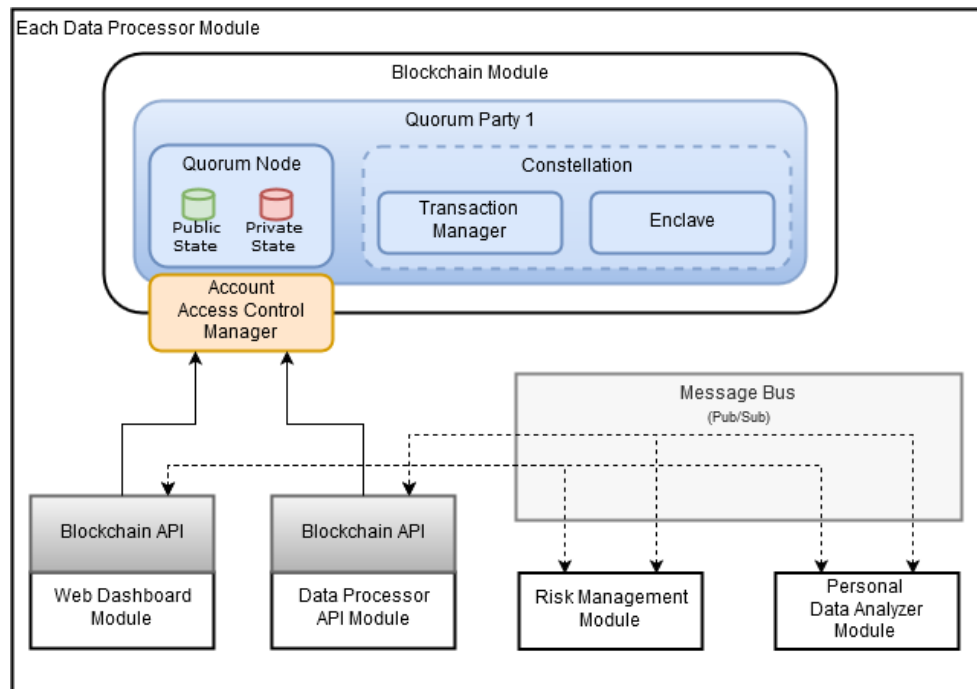


Figure 26 - Data Processor's Blockchain Module

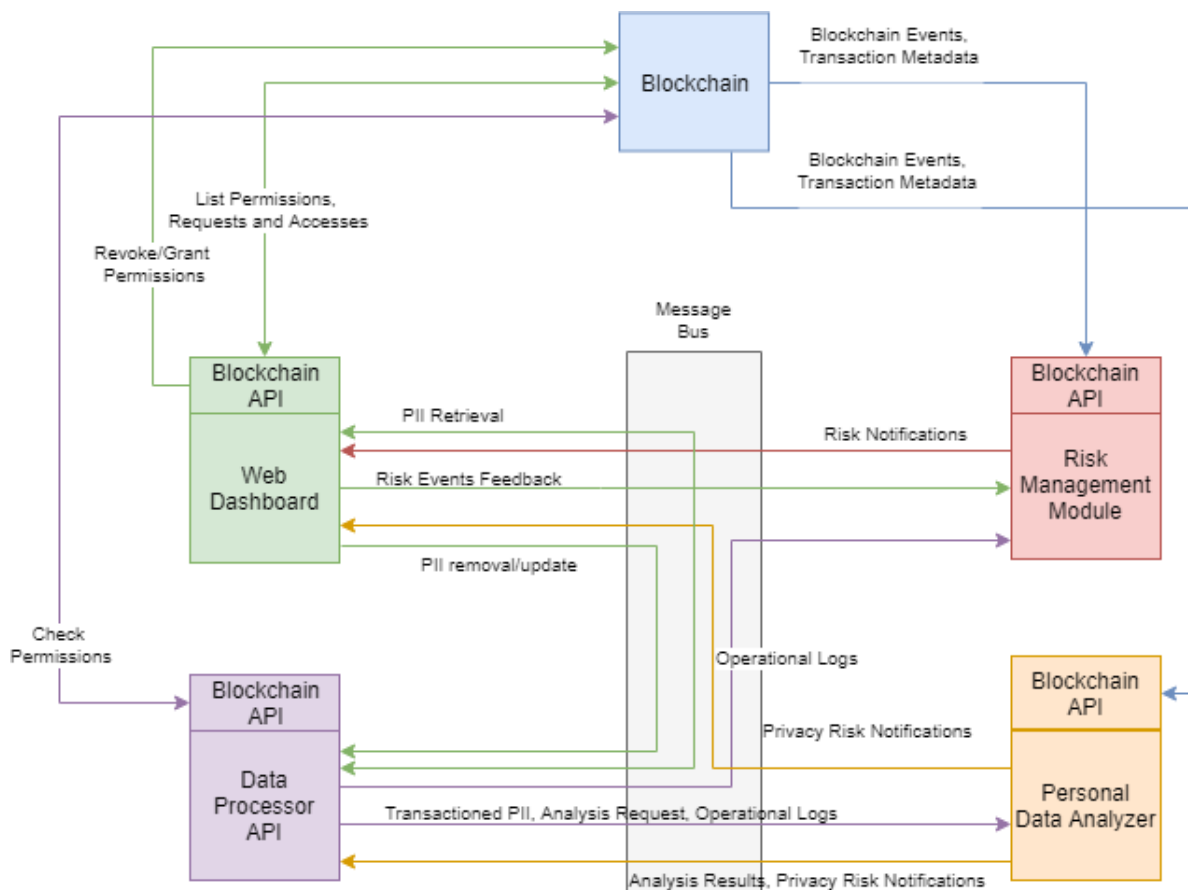


Figure 27 - Integration with Message bus diagram

6.4. Data Storage

The description regarding what data is stored, how it is stores and how this is configures is detailed in the subsection 5.1.3 of this document, nevertheless, as a summary it can be said that: "...PoSeID-on members who manage a Blockchain node must individually take care of the proper management of cryptographic content and their private node management keys. This is because there is no central repository where this information is stored. Therefore, the management of the nodes, and the management of the keys, will be carried out by the data processors (DP) themselves. Moreover, due to the Quorum design, the software used for the implementation of the ledger, the management of these keys will have to be done by software." (see point 5.1.3 for more information).

6.5. Example of the Integrated Use Case between two pilots

In order to provide an integrated platform with all components described, following (Figure 28) is the description of an example of the Integrated Use Case between two pilots:

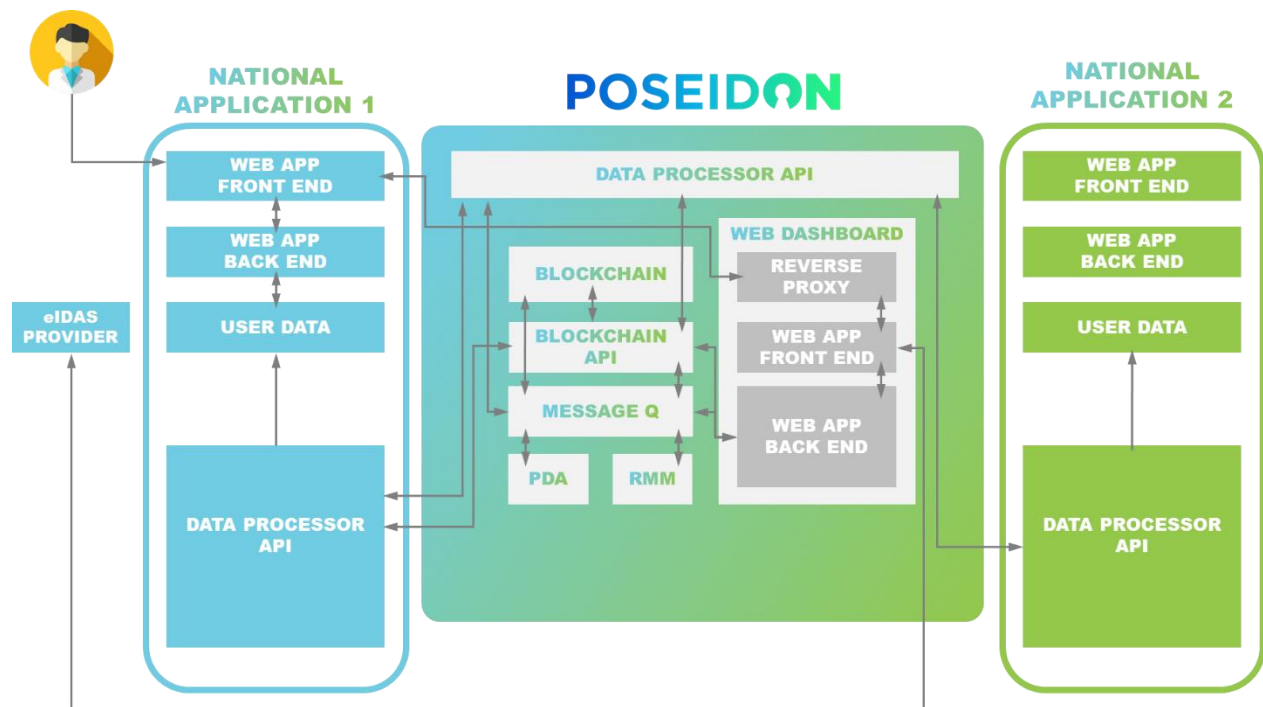


Figure 28 - Integrated Use Case between two pilots

Step 1: The user connects to the web app front end and access to the web service through a dedicated user and password.

Step 2: The user wants to enable a new service that requires the availability of PII not yet managed by the national application

Step 3: The user wants to use "PoSeID-on" to retrieve PII data needed to enable the service. The user clicks on "get my data from PoSeID-on" button.

Step 4: The user XYZ will be redirected to PoSeID-on web dashboard (via reverse proxy) to authenticate himself to the platform (through eIDAS credential). A positive feedback will be sent

to the national application when eIDAS provider confirm that both, user and password for the user XYZ are correct.

Step 5: After authentication process was fully completed, to retrieve PII data needed to enable the service, national application Data Processor API will send a request to the PoSeID-on data processor API module to have this data linked to the user XYZ.

Step 6: The user XYZ (still waiting on PoSeID-on web dashboard) will give his consent to national application to use his data.

Step 7: Web application backend communicates to the blockchain to change data permission and to define a new block and register the transaction on the smart contract.

Step 8: Data processor API ask to blockchain if PoSeID-on manages this data for user XYZ. Blockchain smart contract confirm that the searched data is managed by PoSeID-on and it is archived in the National Application 2 national application 2.

Step 9: PoSeID-on data processor API asks to National Application 2 (data processor API) PII data needed.

Step 10: National Application 2 data processor API read the data required within the local user data repository and returns it to the PoSeID-on data processor API.

Step 11: PoSeID-on data processor API now can send data required to National Application 1 data processor API.

Step 12: Data processor API now can store received data on National Application 1 user data repository.

Step 13: Web app backend can now enable service and use PII data stored on local repository. All the transactions are managed by the message bus and are analysed by PDA and RMM.

7. Innovation summary

The work carried out during the design and implementation of the Blockchain network and the Smart Contracts that coexist inside it serve to guide the functionality of the system and protect the information that is handled by the data subjects to manage the specified permissions.

New advanced identification techniques have been added by PoSeID-on to manage the relationship between transport keys and Blockchain keys. They are generated and managed by the BC API and kept securely without sharing with the Web Based Dashboard and the Data Processor API. The identification of Data Subjects (extensible to any type of user) as a key scientific objective has been achieved without letting the Data Processors to access PII that they are not allowed to view, through the key management against the BC API.

The management of the user Blockchain accounts is another important objective in the innovation achieved by PoSeID-on. This is critical because of the importance the governments are given to the personal data and PII. Following the scientific objective of accomplishing the GDPR challenges with special attention to its Article17, the solution taken is to unlink the data from the system without breaking the block record and without rejecting the Blockchain original philosophy using burnable pseudo-identities. The burnable pseudo-identity is the mechanism to create a pool of pseudo-identities for each Data Subject user that can be erased by request.

8. Conclusion

After the research and the work carried out along WP3 activities, it can be concluded that the objectives (see Section 2) has been achieved successfully, applying the innovation mentioned in Section 7. The following steps are the development of the Blockchain components and the DP API final versions, the integration of the components developed in the WP3 with the other PoSeID-on components and the adjustment to provide the intended Use Case pilots.

9. References

The references mentioned throughout this document are the following:

- [1] [What is a Permissioned Blockchain Network?](#)
- [2] [And then there were 8 - a look at the leading blockchain frameworks](#)
- [3] [Blockchain for government and public services](#), Tom Lyons, Ludovic Courcelas, EuBlockchain Observatory and Forum, 7 December 2018.
- [4] [Zug And uPort See First Citizens' Identity Registered On The Ethereum Blockchain](#), ETHNews, 17 November, 2017.
- [5] [Switzerland's first municipal blockchain vote hailed a success](#), SwissInfo, 2 July 2018.
- [6] [Zug residents can now ride e-bikes using their uPort-powered Zug Digital IDs](#), Alice Nawfal, Medium, 14 November, 2018
- [7] [Bitland's African Blockchain Initiative Putting Land On The Ledger](#), Forbes, 5 April, 2016.
- [8] [Indian State Partners With Blockchain Startup for Land Registry Pilot](#), Coindesk, 10 October, 2017.
- [9] [Sweden's Land Registry Demos Live Transaction on a Blockchain](#), Coindesk, 15 June, 2018.
- [10] [HM Land Registry to explore the benefits of blockchain](#), Gov.uk, 1 October, 2018.
- [11] [Estonian e-health record](#).
- [12] [A Nordic way to blockchain in healthcare](#), HiMiss Europe, 26 February, 2018.
- [13] [Academic Certificates on the Blockchain](#), University of Nicosia Blockchain Initiative.
- [14] [Malta Pilots Blockchain-Based Credentials Program](#), IEEE Spectrum, 5 June, 2018.
- [15] [University consortium set up to authenticate degrees using blockchain technology](#), New Straits Times, 9 November, 2018.
- [16] [bcdiploma.com](#)
- [17] [West Virginia Introduces Blockchain Voting App for Midterm Election](#), Slate, 25 September, 2018.
- [18] [Russia Is Leading the Push for Blockchain Democracy](#), Coindesk, 21 February, 2018.
- [19] E. Yavuz, A. K. Koç, U. C. Çabuk and G. Dalkılıç, "Towards secure e-voting using ethereum blockchain," 2018 6th International Symposium on Digital Forensic and Security (ISDFS), Antalya, pp. 1-7. doi: 10.1109/ISDFS.2018.8355340, 2018.
- [20] [Singapore regulator, OCBC, HSBC, MUFG create 'Know Your Customer' blockchain prototype](#), The Business Times, 3 October, 2017.
- [21] [Smart Dubai: Dubai Blockchain strategy](#).
- [22] [The Application of Blockchain Technology in E-Government in China](#), H. Hou, 2017 26th International Conference on Computer Communication and Networks (ICCCN), Vancouver, BC, 2017, pp. 1-4. doi: 10.1109/ICCCN.2017.8038519
- [23] Blockchain Technology as a Support Infrastructure in e-Government, Ølnes, Svein & Jansen, Arild. (2017). 10.1007/978-3-319-64677-0
- [24] [Art. 17 of GDPR](#)
- [25] [Quorum overview](#).
- [26] [Quorum API details](#).
- [27] <https://www.npmjs.com/package/bip39>
- [28] [BIP39](#)
- [29] [Private transaction execution flow in Quorum](#)
- [30] [Network permissioning](#).
- [31] How to secure REST APIs using the *key id* and the *secret key* can be found at: [here](#) or [here](#).
- [32] [Example taken](#).



[33] [Image.](#)

[34] [Image.](#)

Annex I. Blockchain API formal documentation

URI scheme

BasePath : /api

Schemes : HTTP

I. POST /auth/signin

• Parameters

Type	Name	Schema
Body	SigninVm <i>required</i>	SigninVm

• Responses

HTTP Code	Schema
201	SigninResponseVm
400	ApiException

• Consumes

application/json

• Produces

application/json

• Tags

API#Authentication

II. GET /client/id/{id}

• Parameters

Type	Name	Schema
Path	id <i>required</i>	string

• Responses

HTTP Code	Schema
200	UserVm
400	ApiException

• Consumes

application/json

• Produces

application/json

• Tags

API#Management

III. POST /client/register

- Parameters**

Type	Name	Schema
Body	RegisterVm <i>required</i>	RegisterVm

- Responses**

HTTP Code	Schema
201	UserVm
400	ApiException

- Consumes**

application/json

- Produces**

application/json

- Tags**

API#Management

IV. GET /client/{apikey}

- Parameters**

Type	Name	Schema
Path	apikey <i>required</i>	string

- Responses**

HTTP Code	Schema
200	UserVm
400	ApiException

- Consumes**

application/json

- Produces**

application/json

- Tags**

API#Management

V. POST /dp

Register a new Blockchain Data Processor

- Parameters**

Type	Name	Schema
Body	DPRegisterDto <i>required</i>	DPRegisterDto

- Responses**

HTTP Code	Schema
201	DPVm
400	ApiException

- Consumes**

application/json

- Produces**

application/json

- **Tags**
POSEID-ON#DP

- **Security**

Type	Name
apiKey	bearer

VI. POST /dp/generateTransport/

Generate a new TransportKey

- **Parameters**

Type	Name	Schema
Body	DPCreateTransportKeyDto <i>required</i>	DPCreateTransportKeyDto

- **Responses**

HTTP Code	Schema
201	DPTransportKeyVm
400	ApiException

- **Consumes**
application/json
- **Produces**
application/json
- **Tags**
POSEID-ON#DP

- **Security**

Type	Name
apiKey	bearer

VII. POST /dp/{cert}/permissions/request/{field}/{serviceID}/{certDS}

Request permission from a Data Processor to a Data Subject

- **Parameters**

Type	Name	Description	Schema
Path	cert <i>required</i>	the Transport key	string
Path	certDS <i>required</i>	The Data Processor cert to grant/revoked permission	string
Path	field <i>required</i>	The field permission	string
Path	permissionID <i>required</i>	The permissionID	string
Path	serviceID <i>required</i>	The serviceID	string
Body	DPActionBodyDto <i>required</i>		DPActionBodyDto

- **Responses**

HTTP Code	Schema
201	DPVm
400	ApiException

- **Consumes**

application/json

- **Produces**

application/json

- **Tags**

POSEID-ON#DP

- **Security**

Type	Name
apiKey	bearer

VIII. GET /dp/{cert}/permissions/{permissionID}

DEBUG ONLY: Return permission data permission ID to DS

- **Parameters**

Type	Name	Description	Schema
Path	cert <i>required</i>	the Transport key	string
Path	certDS <i>required</i>	The Data Processor cert to grant/revoke permission	string
Path	field <i>required</i>	The field permission	string
Path	permissionID <i>required</i>	The permissionID	string
Path	serviceID <i>required</i>	The serviceID	string

- **Responses**

HTTP Code	Schema
200	< DPVm > array
400	ApiException

- **Consumes**

application/json

- **Produces**

application/json

- **Tags**

POSEID-ON#DP

- **Security**

Type	Name
apiKey	bearer

IX. GET /dp/{cert}/permissions/{permissionID}/{certDS}

Return true is Data Processor has requested the permission ID to DS

- **Parameters**

Type	Name	Description	Schema
Path	cert <i>required</i>	the Transport key	string
Path	certDS <i>required</i>	The Data Processor cert to grant/revoke permission	string
Path	field <i>required</i>	The field permission	string
Path	permissionID <i>required</i>	The permissionID	string
Path	serviceID <i>required</i>	The serviceID	string

- **Responses**

HTTP Code	Schema
200	< DPVm > array
400	ApiException

- **Consumes**

application/json

- **Produces**

application/json

- **Tags**

POSEID-ON#DP

- **Security**

Type	Name
apiKey	bearer

X. POST /dp/{cert}/recover/

Recover a previously registered user

- **Parameters**

Type	Name	Schema
Path	cert <i>required</i>	string
Body	DPRecoverDto <i>required</i>	DPRecoverDto

- **Responses**

HTTP Code	Schema
201	DPVm
400	ApiException

- **Consumes**

application/json

- **Produces**

application/json

- **Tags**

POSEID-ON#DP

- **Security**

Type	Name
apiKey	bearer

XI. PUT /dp/{cert}/refresh/

Refresh a DP account

- **Parameters**

Type	Name	Schema
Path	cert <i>required</i>	string
Body	DPRecoverDto <i>required</i>	DPRecoverDto

- **Responses**

HTTP Code	Schema
201	DPVm
400	ApiException

- **Consumes**

application/json

- **Produces**

application/json

- **Tags**

POSEID-ON#DP

- **Security**

Type	Name
apiKey	bearer

XII. POST /ds

Register a new Blockchain Data Subject

- **Parameters**

Type	Name	Schema
Body	DSRegisterDto <i>required</i>	DSRegisterDto

- **Responses**

HTTP Code	Schema
201	DSVm
400	ApiException

- **Consumes**

application/json

- **Produces**

application/json

- **Tags**

POSEID-ON#DS

- **Security**

Type	Name
apiKey	bearer

XIII. POST /ds/generateTransport/

Generate a new TransportKey

- **Parameters**

Type	Name	Schema
Body	DSCreateTransportKeyDto <i>required</i>	DSCreateTransportKeyDto

- **Responses**

HTTP Code	Schema
201	DSVm
400	ApiException

- **Consumes**
application/json
- **Produces**
application/json
- **Tags**
POSEID-ON#DS

- **Security**

Type	Name
apiKey	bearer

XIV. POST /ds/{cert}/permissions/{permissionID}/grant/{certDP}

Grant serviceID field permission to a Data Processor

- **Parameters**

Type	Name	Description	Schema
Path	cert <i>required</i>	the Transport key	string
Path	certDP <i>required</i>	The Data Processor to grant/revoke permission	string
Path	field <i>required</i>	The field permission	string
Path	permissionID <i>required</i>	The id permission to grant	string
Path	serviceID <i>required</i>	The serviceID	string
Path	status <i>required</i>	The status of the permissions to get	enum (requested, granted, revoked)
Body	DSActionBodyDto <i>required</i>		DSActionBodyDto

- **Responses**

HTTP Code	Schema
201	DSVm
400	ApiException

- **Consumes**
application/json
- **Produces**
application/json
- **Tags**
POSEID-ON#DS

- **Security**

Type	Name
apiKey	bearer

XV. POST /ds/{cert}/permissions/{permissionID}/revoke/{certDP}

Revoke serviceID field permission to a Data Processor

- Parameters**

Type	Name	Description	Schema
Path	cert <i>required</i>	the Transport key	string
Path	certDP <i>required</i>	The Data Processor to grant/revoke permission	string
Path	field <i>required</i>	The field permission	string
Path	permissionID <i>required</i>	The id permission to grant	string
Path	serviceID <i>required</i>	The serviceID	string
Path	status <i>required</i>	The status of the permissions to get	enum (requested, granted, revoked)
Body	DSActionBodyDto <i>required</i>		DSActionBodyDto

- Responses**

HTTP Code	Schema
201	DSVm
400	ApiException

- Consumes**

application/json

- Produces**

application/json

- Tags**

POSEID-ON#DS

- Security**

Type	Name
apiKey	bearer

XVI. GET /ds/{cert}/permissions/{status}

List permissions filter by status.

- Parameters**

Type	Name	Description	Schema
Path	cert <i>required</i>	the Transport key	string
Path	certDP <i>required</i>	The Data Processor to grant/revoke permission	string
Path	field <i>required</i>	The field permission	string
Path	permissionID <i>required</i>	The id permission to grant	string
Path	serviceID <i>required</i>	The serviceID	string
Path	status <i>required</i>	The status of the permissions to get	enum (requested, granted, revoked)
Body	DSActionBodyDto <i>required</i>		DSActionBodyDto

- Responses**

HTTP Code	Schema
201	DSVm
400	ApiException

- **Consumes**
application/json
- **Produces**
application/json
- **Tags**
POSEID-ON#DS

- **Security**

Type	Name
apiKey	bearer

XVII. POST /ds/{cert}/recover/

Recover a previously registered user

- **Parameters**

Type	Name	Schema
Path	<i>cert required</i>	string
Body	<i>DSRecoverDto required</i>	DSRecoverDto

- **Responses**

HTTP Code	Schema
201	DSVm
400	ApiException

- **Consumes**
application/json
- **Produces**
application/json
- **Tags**
POSEID-ON#DS

- **Security**

Type	Name
apiKey	bearer

XVIII. PUT /ds/{cert}/refresh/

Refresh a DS account

- **Parameters**

Type	Name	Schema
Path	<i>cert required</i>	string
Body	<i>DSRecoverDto required</i>	DSRecoverDto

- **Responses**

HTTP Code	Schema
201	DSVm
400	ApiException

- **Consumes**
application/json
- **Produces**
application/json
- **Tags**
POSEID-ON#DS

- **Security**

Type	Name
apiKey	bearer

XIX. Definitions

a. ApiException

Name	Schema
error <i>optional</i>	string
errors <i>optional</i>	object
message <i>optional</i>	string
path <i>optional</i>	string
status <i>optional</i>	string
statusCode <i>optional</i>	number
timestamp <i>optional</i>	string

b. DPActionBodyDto

Name	Description	Schema
lifetime <i>required</i>	The lifetime of the permission Example : "100"	string
passphrase <i>required</i>	The passphrase needs to unlock dp Example : "tecnalia"	string

c. DPCreateTransportKeyDto

Name	Description	Schema
name <i>optional</i>	Name to use on generation of the transportKey Example : "myName"	string

d. DPRecoverDto

Name	Description	Schema
mnemonic <i>optional</i>	Mnemonic generated to recreate the HDWallet Example : "abba hylu keso"	string
passphrase <i>optional</i>	Passphrase to open the keystore Example : "myS\$cr\$tT@k\$n"	string

e. DPRegisterDto

Name	Description	Schema
cert <i>required</i>	eIDAS (electronic IDentification, Authentication and trust Services) Token Example : "cert1"	string
language <i>optional</i>	Lanaguage to use generatic mnemonic Example : "english"	string
passphrase <i>required</i>	Passphrase to protect the keystore Example : "myS\$cr\$tT@k\$n"	string

f. DPTransportKeyVm

Name	Description	Schema
cert <i>required</i>	Certificate generated Example : "3epo3r2hjoi23ihr32oiuhr23i4r7823r7h823rhiu "	string
createdAt <i>optional</i>		string (date-time)
id <i>optional</i>		string
privateKey <i>optional</i>	PrivateKey generated Example : "83e2iuedwihuedhui32hui32ehiu32i3279yruh4rfhu7y32"	string
updatedAt <i>optional</i>		string (date-time)

g. DPVm

Name	Description	Schema
cert <i>required</i>	eIDAS (electronic IDentification, Authentication and trust Services) Token Example : "cert1"	string
createdAt <i>optional</i>		string (date-time)
id <i>required</i>		string
mnemonic <i>optional</i>	Mnemonic generated to recreate the HDWallet Example : "myS\$cr\$tT@k\$n"	string
passphrase <i>optional</i>	Passphrase to open the keystore Example : "myS\$cr\$tT@k\$n"	string
updatedAt <i>optional</i>		string (date-time)

h. DSActionBodyDto

Name	Description	Schema
passphrase <i>required</i>	The passphrase need to unlock ds Example : "tecnalia"	string

i. DSCreateTransportKeyDto

Name	Description	Schema
name <i>optional</i>	Name to use on generation of the transportKey Example : "myName"	string

j. DSRecoverDto

Name	Description	Schema
mnemonic <i>optional</i>	Mnemonic generated to recreate the HDWallet Example : "abba hylu keso"	string
passphrase <i>optional</i>	Passphrase to open the keystore Example : "myS\$cr\$tT@k\$n"	string

k. DSRegisterDto

Name	Description	Schema
cert <i>required</i>	eIDAS (electronic IDentification, Authentication and trust Services) Token Example : "cert1"	string
language <i>optional</i>	Lanaguage to use generatic mnemonic Example : "english"	string
passphrase <i>required</i>	Passphrase to protect the keystore Example : "myS\$cr\$tT@k\$n"	string

l. DSVm

Name	Description	Schema
cert <i>required</i>	eIDAS (electronic IDentification, Authentication and trust Services) Token Example : "cert1"	string
createdAt <i>optional</i>		string (date-time)
id <i>required</i>		string
mnemonic <i>optional</i>	Mnemonic generated to recreate the HDWallet Example : "myS\$cr\$tT@k\$n"	string
passphrase <i>optional</i>	Passphrase to open the keystore Example : "myS\$cr\$tT@k\$n"	string
updatedAt <i>optional</i>		string (date-time)

m. RegisterVm

Name	Description	Schema
apikey <i>required</i>	Minimum length : 6	string
name <i>optional</i>	Example : "application1"	string
role <i>optional</i>	Default : "User"	string
secret <i>required</i>	Minimum length : 6	string (secret)

n. SigninResponseVm

Name	Schema
token <i>required</i>	string
user <i>required</i>	UserVm

o. SigninVm

Name	Description	Schema
apikey <i>required</i>	Minimum length : 6	string
secret <i>required</i>	Minimum length : 6	string (secret)

p. UserVm

Name	Schema
apikey <i>required</i>	string
createdAt <i>optional</i>	string (date-time)
fullName <i>optional</i>	string
id <i>optional</i>	string
name <i>optional</i>	string
role <i>optional</i>	string
updatedAt <i>optional</i>	string (date-time)

q. Security

bearer *Type* : apiKey

Name : Authorization

In : HEADER