



D4.5 Smart Contracts and DApps implementation tools II

Document Identification			
Status	Final	Due Date	31/08/2022
Version	1.0	Submission Date	12/09/2022

Related WP	WP4	Document Reference	D4.5
Related Deliverable(s)	D2.3 "Pledger Overall Architecture", D3.6 "QoS and SLA assessment and negotiation tools II"	Dissemination Level (*)	PU
Lead Participant	INNOV	Lead Author	Nikos Kapsoulis
Contributors	ICCS/NTUA, i2CAT, HOLO, FILL	Reviewers	Francesco Iadanza (ENG) Alexander Werlberger, Carina Pamminger (HOLO)

Keywords:
Smart Contracts, DApps, Edge, Cloud, Distributed Ledger Technology, Blockchain Network, Service-Level Agreements, Orchestration

This document is issued within the frame and for the purpose of the PLEDGER project. This project has received funding from the European Union's Horizon2020 Framework Programme under Grant Agreement No. 871536. The opinions expressed and arguments employed herein do not necessarily reflect the official views of the European Commission.

The dissemination of this document reflects only the author's view and the European Commission is not responsible for any use that may be made of the information it contains. This deliverable is subject to final acceptance by the European Commission.

This document and its content are the property of the PLEDGER Consortium. The content of all or parts of this document can be used and distributed provided that the PLEDGER project and the document are properly referenced.

Each PLEDGER Partner may use this document in conformity with the PLEDGER Consortium Grant Agreement provisions.

(*) Dissemination level: **PU**: Public, fully open

Document Information

List of Contributors	
Name	Partner
Ariana Polyviou	INNOV
Kassiani Papatiriu	INNOV
Chris Kavvadias	INNOV
George Gerolimos	INNOV
Filia Fillipou	INNOV
Nikos Kapsoulis	INNOV
Estela Carmona Cejudo	i2CAT
August Betzler	i2CAT
Verena Stanzl	FILL
Nour Fendri	HOLO
Alexandros Psychas	ICCS
Antonis Litke	ICCS

Document History			
Version	Date	Change editors	Changes
0.1	29/04/2022	Ariana Polyviou, Kassiani Papatiriu, Chris Kavvadias, George Gerolimos, Filia Fillipou, Nikos Kapsoulis (INNOV)	Extending <i>D4.2 Smart Contracts and DApps implementation tools</i> [1], subsection rearrangement and extraction of new Table of Contents.
0.2	03/06/22	Ariana Polyviou, Kassiani Papatiriu, Chris Kavvadias, George Gerolimos, Filia Fillipou, Nikos Kapsoulis (INNOV)	Components architecture and data models reformulations and updates
0.3	08/07/22	Ariana Polyviou, Kassiani Papatiriu, Chris Kavvadias, George Gerolimos, Filia Fillipou, Nikos Kapsoulis (INNOV), Alexandros Psychas, Antonis Litke (ICCS)	Components architecture elaborations and updates
0.4	26/07/22	Ariana Polyviou, Kassiani Papatiriu, Chris Kavvadias, George Gerolimos, Filia Fillipou, Nikos Kapsoulis (INNOV), Nour Fendri (HOLO)	Updates on technical description and demonstrations
0.5	27/07/22	Ariana Polyviou, Kassiani Papatiriu, Chris Kavvadias, George Gerolimos, Filia Fillipou, Nikos Kapsoulis (INNOV)	Updates on SLA and use cases demonstration section

Document name:	D4.5 Smart Contracts and DApps implementation tools II	Page:	2 of 42
Reference:	D4.5	Dissemination:	PU
	Version:	1.0	Status: Final

Document History			
Version	Date	Change editors	Changes
0.6	27/07/22	Nikos Kapsoulis (INNOV)	Ready for internal review
0.7	24/08/22	Francesco Iadanza (ENG), Alexander Werlberger, Carina Pamminger (HOLO), Nikos Kapsoulis (INNOV)	Internal comments and reviews addressed and merged. Ready for quality assessment.
0.8	26/08/2022	Carmen San Román (ATOS)	Quality Assurance review
0.9	01/09/2022	Nikos Kapsoulis (INNOV)	Project Coordinator comments addressed.
1.0	12/09/2022	Lara López (ATOS)	Final version to be submitted

Quality Control		
Role	Who (Partner short name)	Approval Date
Deliverable leader	Nikos Kapsoulis (INNOV)	24/08/2022
Quality manager	Carmen San Román (ATOS)	26/08/2022
Project Coordinator	Lara López (ATOS)	12/09/2022

Table of Contents

Document Information	2
Table of Contents	4
List of Tables.....	5
List of Figures	6
List of Acronyms.....	7
Executive Summary	8
1 Introduction	9
1.1 Relation to other project work.....	9
1.2 Structure of the document.....	9
2 Functional description	10
3 Technical description.....	13
3.1 Baseline technologies and dependencies	13
3.2 Components Architecture.....	13
3.3 Interfaces provided.....	19
3.4 Data models and functionalities	19
4 Installation and usage guides	25
4.1 Requirements	25
4.2 Installation.....	25
4.3 Usage.....	26
4.4 Licenses.....	26
4.5 Source code repository	26
5 Demonstration	27
5.1 Scenarios Description	27
5.2 Validation and Verification.....	38
5.3 Demonstrator.....	40
6 Conclusions and next steps.....	41
7 References	42

Document name:	D4.5 Smart Contracts and DApps implementation tools II	Page:	4 of 42
Reference:	D4.5	Dissemination:	PU
	Version:	1.0	Status: Final

List of Tables

<i>Table 1: Blockchain subsystem Components [2]</i>	12
<i>Table 2: Baseline technologies and dependencies</i>	13
<i>Table 3: Main security and privacy pillars of Hyperledger</i>	14
<i>Table 4: DLT orchestration main phases</i>	14
<i>Table 5: SCoDeSy stages</i>	15
<i>Table 6: UC1 Handshake entries</i>	20
<i>Table 7: UC2 on-chain essential set of data entries</i>	22
<i>Table 8: UC3 on-chain data entries of one part</i>	23

Document name:	D4.5 Smart Contracts and DApps implementation tools II	Page:	5 of 42
Reference:	D4.5	Dissemination:	PU
		Version:	1.0
		Status:	Final

List of Figures

Figure 1: Pledger Core Subsystem [2]	10
Figure 2: Blockchain subsystem component diagram [2]	11
Figure 3: Blockchain network high-level architecture	13
Figure 4: SLASC Bridge high-level architecture	16
Figure 5: SLASC Bridge Interactions	16
Figure 6: Wallet capabilities and interactions.....	18
Figure 7: Whisper handshake in UC1.....	20
Figure 8: Block diagram of UC2	21
Figure 9: Parts produced data in UC3	23
Figure 10: SLASC Bridge main objectives	27
Figure 11: SLASC Bridge main objectives breakdown.....	28
Figure 12: SLA configuration sent to SLASC Bridge.....	28
Figure 13: SLASC Bridge identifies the user	29
Figure 14: SLASC Bridge connects the required user account on the ledger.....	29
Figure 15: SLASC Bridge joins the contractual terms.....	30
Figure 16: SLASC Bridge associates smart contract structure	30
Figure 17: Contractual terms equivalent is created: SLA to Smart Contract Process finality.....	31
Figure 18: SLA violation sent to SLASC Bridge	31
Figure 19: SLASC Bridge retrieves and triggers related smart contract structure	32
Figure 20: Join violations tracking.....	32
Figure 21: Trigger compensation scheme	33
Figure 22: Compute user instalments	33
Figure 23: SLASC Bridge verifies transactions	34
Figure 24: Violations handled by SLASC Bridge	34
Figure 25: Pledger Wallet landing screen.....	35
Figure 26: Pledger Wallet credentials.....	35
Figure 27: User Wallet balance.....	36
Figure 28: Different results for time range queries on risk level of VRU incidents.....	36
Figure 29: Different results for time range queries on quality degree of manufactured parts	37
Figure 30: Server Node.....	37
Figure 31: Client Node	38
Figure 32: SLASC Bridge Demonstrator blockchain data.....	38
Figure 33: Wallet Demonstrator blockchain data	39

List of Acronyms

Abbreviation / acronym	Description
DApp	Decentralized Application
DLT	Distributed Ledger Technology
Dx.y	Deliverable number y belonging to WP x
OBU	On-Board Unit
RSU	Road Side Unit
SLA	Service-Level Agreement
Tx.y	Task number y belonging to WP x
UC	Use Case
VRU	Vulnerable road users
WP	Work Package

Executive Summary

This document accompanies the delivery of T4.2 "Smart Contracts and DApps on edge and cloud" results during the last phase of the task. This is the last iteration of the deliverable and the prototype DLT (Distributed Ledger Technology) where the trusted nodes and privacy rules that apply under its schema are deployed. The entire prototype is supporting the needs of the edge and cloud environments as requested by the respective Pledger cases.

Task 4.2 focuses on the deployment of the smart contracts and decentralized applications (DApps) in a trusted environment of reliable blockchain nodes. Every Pledger component that demands and realizes blockchain activities is supported by the task features and modules while maintained by the Pledger DLT. Furthermore, the task results are handling the interoperability and scalability framework of the project.

The delivered output on this iteration completes the important foundation of the task. Every module that is built is supported and hosted under the installed prototype of the first iteration. Pledger DLT comprises an orchestrated permissioned blockchain that is dedicated to the project with trusted blockchain nodes where smart contracts and DApps reside. Pledger DLT incorporates private environments that are providing confidential blockchain activity among network users.

Document name:	D4.5 Smart Contracts and DApps implementation tools II	Page:	8 of 42				
Reference:	D4.5	Dissemination:	PU	Version:	1.0	Status:	Final

1 Introduction

Purpose of the document

The purpose of this deliverable is to provide the insights and capabilities of the underlying DLT deployment, the completed network integration along with the applicable functionalities supporting smart contracts and DApps for the latest release.

This deliverable accompanies the prototype during the second iteration of WP4 and provides the necessary installation and usage guidelines of the demonstrator.

1.1 Relation to other project work

The prototype and the deliverable are guided by the work performed in WP2 about the architecture, D2.3 "Pledger Overall Architecture" [2], and WP3 about the Service-Level Agreement (SLA) tools and components, D3.6 "QoS and SLA assessment and negotiation tools II" [9].

The prototype constitutes the integrated foundation that serves the core platform requirements through smart contracts and DApps. The prototype blockchain network provides comprises of trusted nodes and privacy rules that apply under its schema.

1.2 Structure of the document

This document is structured in six major chapters:

Chapter 2 presents the functional overview of the prototype.

Chapter 3 elaborates on the technical description of the prototype, its specific components architecture and data models.

Chapter 4 presents the installation and usage guide of the prototype.

Chapter 5 presents the demonstration of the prototype with the description of scenarios.

Chapter 6 presents the conclusions for the work performed.

Document name:	D4.5 Smart Contracts and DApps implementation tools II	Page:	9 of 42				
Reference:	D4.5	Dissemination:	PU	Version:	1.0	Status:	Final

2 Functional description

The Blockchain subsystem deploys the required blockchain support for the project. Its main purpose is to host and deliver the corresponding functionalities for the SLA subsystem triggers and use cases adaptive demands. In the second iteration, the complete blockchain network capabilities are integrated with the subsystem and the core Pledger platform, while the corresponding rationales are presented throughout the document and most significantly in section 3.2.3.

In Pledger functional architecture, the DLT is graphically placed in the Support layer as shown in Figure 1. The actual blockchain deployment relies on a decentralized architecture that spans across different blockchain networks and locations.

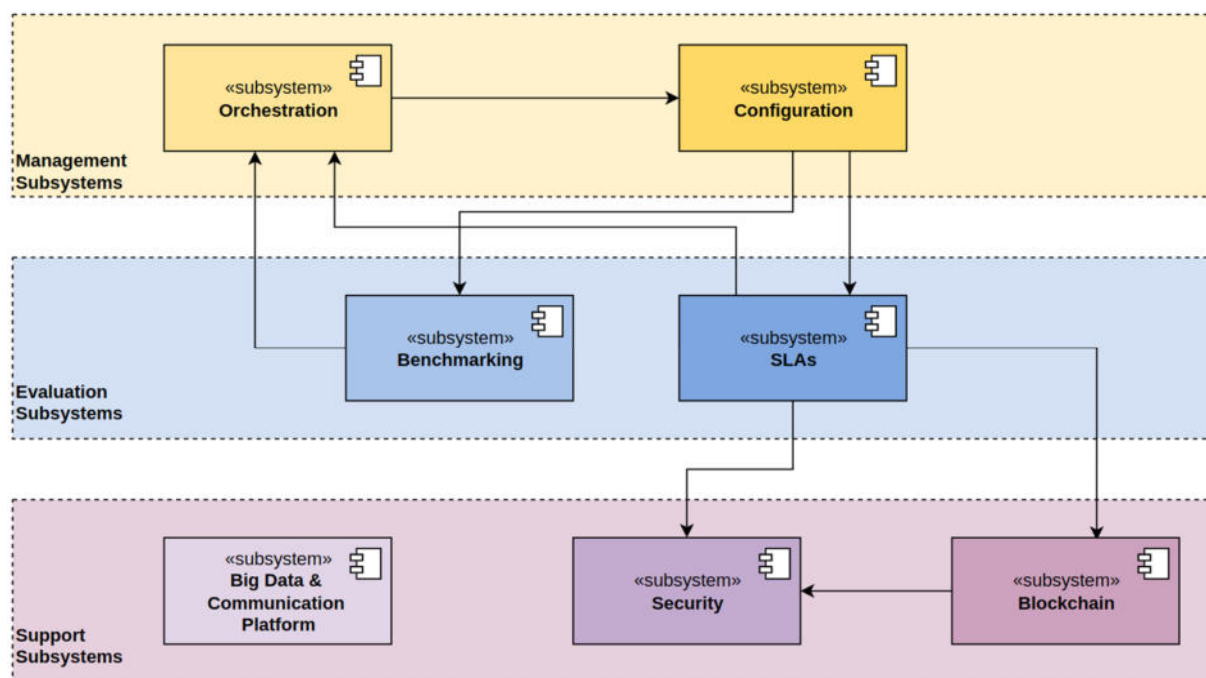


Figure 1: Pledger Core Subsystem [2]

As mentioned, Pledger provides its own blockchain, taking under consideration features such as openness, access, speed, security, consensus mechanisms, etc. As extracted from the architecture deliverable D2.3 Pledger Overall Architecture [2] Figure 2 shows a functional insight of the Blockchain subsystem components and their relations as per the architectural updates of the second iteration.

Document name:	D4.5 Smart Contracts and DApps implementation tools II	Page:	10 of 42
Reference:	D4.5	Dissemination:	PU
	Version:	1.0	Status: Final

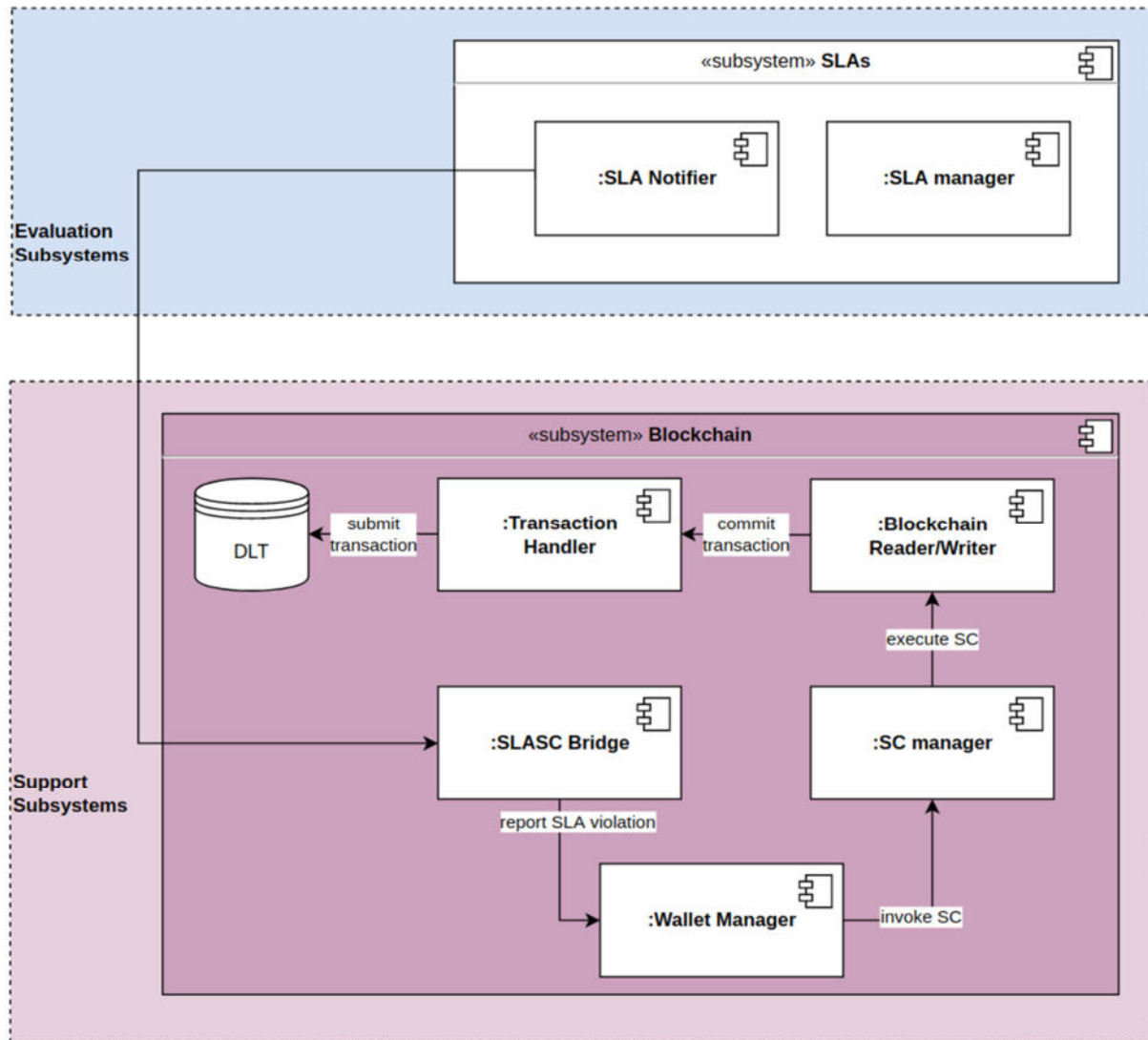


Figure 2: Blockchain subsystem component diagram [2]

The list of the Blockchain subsystem functional components implemented and hosted on the DLT and their roles is reported in Table 1:

Document name:	D4.5 Smart Contracts and DApps implementation tools II	Page:	11 of 42	
Reference:	D4.5	Dissemination:	PU	
	Version:	1.0	Status:	Final

Table 1: Blockchain subsystem Components [2]

ID	Component	Functionality
FC.5.1	Wallet Manager	This component manages the credential information of a blockchain participant entity (either it is a component, administrator, end-user or other). The module is the main method with which an entity is present inside the blockchain network through its transaction activities and smart contract or DApps deployments. In the second period, this component has been extended to support granting access to specific data of the blockchain network, exploiting the corresponding credentials information of the components or users that are requesting the access, i.e. private keys and certificates. The Blockchain Authenticator component of the first iteration is packaged and implemented inside the Wallet Manager.
FC.5.2	Blockchain Reader/ Writer	This component fetches the requested data from the blockchain and submits transactions on it. The module consists of various methods that describe the way and technique or query specification the information is read from the blockchain. Different data need different kinds of reading and fetching, hence the module is constituted from several sub-modules.
FC.5.3	SLASC Bridge	The SLASC Bridge links the SLA contractual terms with the blockchain system. The component describes the methods and functions that define the connection between SLAs and smart contracts in the project while it introduces the one-to-one representation of the SLA terms into smart contract specific contractual terms, being executable code that is triggered by external systems.
FC.5.4	Smart Contract Manager	This component is responsible for all the activities regarding smart contracts, from deployment to maintenance. The component is managing the various deployed smart contracts inside the blockchain network while its main goal is to holistically sustain the Pledger system of smart contracts inside the Pledger blockchain.
FC.5.5	Transaction Handler	This component administers and maintains the transaction flows of the blockchain network. The module subtly checks and confirms the transactions validations and endorsements on the background, while at the same time it offers an apparent role of a gatekeeper where errors or other malfunctions occur.
FC.5.6	DLT	Permissioned blockchain network.

For the latest release, the Blockchain constitutes an integrated solution for Pledger that serves the triggerable smart contracts and deployed DApps. Pledger blockchain provides a permissioned network with secure and trusted nodes which incorporate private environments for confidential transactions. In terms of functional components, the "Blockchain Reader/Writer" along with the "Transaction Handler", the "Smart Contract Manager", the "Wallet Manager" and the "SLASC Bridge" are packaged and deployed as the Blockchain subsystem.

3 Technical description

3.1 Baseline technologies and dependencies

Compared with the first period, there have not been any changes with respect to the baseline technologies and dependencies necessary to support the developed prototype. Such list is reported below for readers' convenience.

The baseline technologies that are used within the prototype are described in Table 2:

Table 2: Baseline technologies and dependencies

Name	Description	Version
Hyperledger Blockchain Framework & Middleware	Blockchain framework provided by Innov-Acts Ltd. that offers platform and tool integration with ease while leveraging from Hyperledger blockchain properties of private and confidential transactions, smart contracts and DApps.	latest
Docker [3]	Container engine for containerized apps running on the edge.	1.18+
Kubernetes [4]	Kubernetes is an open source orchestration software for deploying, scaling, and management of containerized applications. This tool is licensed under Apache License 2.0. Kubernetes is now managed by the Cloud Native Computing Foundation (CNCF).	Vanilla 1.19+
Golang [5]	Go is an open source programming language that makes it easy to build simple, reliable, and efficient software.	1.14+
Node.js [6]	Asynchronous event-driven JavaScript runtime that is designed to build scalable network applications.	12+

3.2 Components Architecture

The implemented functional components of the Blockchain subsystem are packaged and installed in a decentralized way. Every interaction and workflow on the blockchain follow the rules of decentralization across the respective blockchain network. In Figure 3, the updated architecture of the subsystem is depicted approaching its decentralized concept. All Trusted Nodes represent the entire DLT of Pledger. Compared to the first release, the role of Wallet Manager is added to the flow, to support wallet, credentials and key management.

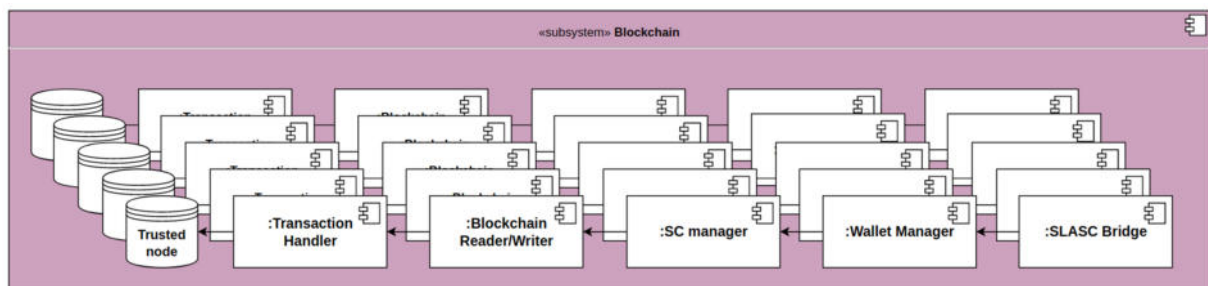


Figure 3: Blockchain network high-level architecture

Document name:	D4.5 Smart Contracts and DApps implementation tools II	Page:	13 of 42
Reference:	D4.5	Dissemination:	PU
	Version:	1.0	Status: Final

The Pledger blockchain network hosts the environment and integration capabilities that are necessary for all the system operations. The dedicated mechanism for enabling smart contracts and DApps for Pledger focuses on the prototype interoperability with the other core components of the project and is mentioned in 3.2.2. Before analysing each of the aforementioned parts of the prototype, the main security and privacy pillars as initially inherited by Hyperledger [7] are described in Table 3 for completeness (no updates from first iteration).

Table 3: Main security and privacy pillars of Hyperledger

Property	Description
Transaction Immutability	Transaction immutability is the functional quality of the blockchain to remain unaltered and unchanged. The blockchain data is kept in sequential blocks forming a chain and each block uses a cryptographic principle or a hash value to keep the data consistent.
Nodes state agreement	In fault-tolerant distributed systems the participants need to agree on the same state. The nodes state agreement is the quality method that applied on all the nodes that participate in the network in order to decide finally.
Permissioned environment	A permissioned blockchain provides a supplementary access control layer that enhances the level of security and improves the role management over public blockchain systems. The permissioned network offers energy efficient operations with low computational needs.

3.2.1 DLT orchestrated

The DLT is an orchestrated blockchain framework. In order to adapt to the project's orchestration and container life-cycle, the prototype of the last iteration is adopting an orchestration nature for the network and each of the hosted components. The deployment of trusted nodes and DApps inside the DLT are respectively modified and re-organized in order to follow the orchestration character of the prototype. Table 4 describes the main phases of the orchestration procedure that Pledger DLT maintains.

Table 4: DLT orchestration main phases

Phase	Description
Migration	The necessary artifacts of the DLT are modified in order to adapt to the YAMLS specifications [8].
Integration	The blockchain framework is pre-configured in order to be deployable on a single cluster.
Deployment	Every DLT component is installed and deployed on Kubernetes.

3.2.1.1 Smart Contract Triggering

In order to intercommunicate securely with the Pledger DLT, a backend mechanism that can reach the inside environment of the blockchain is required, namely Smart Contract Triggering. The latest version consists of the Smart Contract Deployment System (SCoDeSy) described below. In the second period, the SLASC Bridge component has been extended as a separate component and is described in the next subsection (3.2.3).

3.2.1.2 Smart Contract Deployment System

The Smart Contract Deployment System (SCoDeSy) constitutes the main mechanism that deploys the smart contracts on the orchestrated DLT. SCoDeSy is an automated procedure that is delivering smart contracts through a pre-defined life-cycle. Table 5 describes the stages "Package", "Installation", "Approval" and "Submission" that are required to be completed sequentially during the procedure. This component is a necessary backend addition to the blockchain developments and on-chain deployments that eventually eases the smart contract life-cycle procedure on orchestrated environments for the developer.

Table 5: SCoDeSy stages

Stage	Description
1. Package	The smart contract is packaged and delivered to the DLT nodes.
2. Installation	The packaged smart contract is installed on the respective nodes of the DLT.
3. Approval	Every participant of the DLT must approve the smart contract in order to be validated and ready for submission.
4. Submission	The smart contract is submitted on the blockchain and the included business intelligence is enabled on-chain.

SCoDeSy constitutes from the stages described in Table 5. In principle, the development of each stage completes within two major steps of the entire procedure. The first step pertains to the successful creation of the stage's processes, as described in Table 5, and the second step consists of the orchestration of the respective stage, as described in DLT orchestration (Table 4).

3.2.2 SLASC Bridge

The main objective of SLASC Bridge is to link and represent SLA contractual terms on the blockchain deployment. This component describes the methods and functions that define the association between SLAs and smart contracts while introducing the one-to-one representation of the SLA terms into smart contract contractual terms, namely software modules that are packaged and executable at specific point in time. SLASC Bridge incorporates the contractual terms logic from the core platform in order to deliver the blockchain equivalents on the ledger.

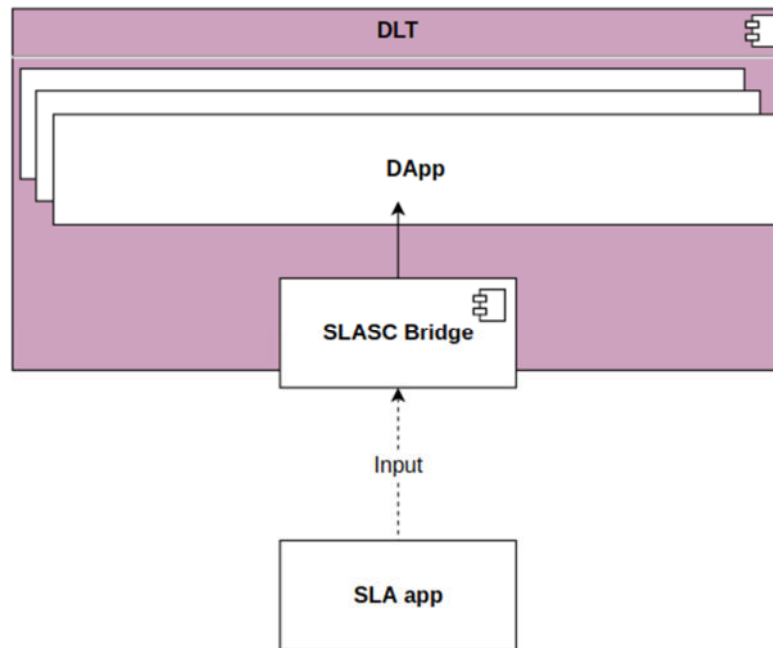


Figure 4: SLASC Bridge high-level architecture

SLASC Bridge intercommunicates with particular DApps that eventually include the business logic. Each event that triggers the specific DApp on the chain executes the respective smart contract workflow followed by the management of the corresponding information on the ledger and by returning the required response when necessary. The high-level architecture of the SLASC Bridge component is depicted in Figure 4.

In order to deliver the business intelligence of the SLA contractual terms into the corresponding blockchain equivalents, the related interoperability between the candidate structural entities is required. Particularly, the formulation of the SLASC Bridge and its capabilities aims at the alignment of its dependent interactions with the components that are established outside of the blockchain. Thus, the deployed interoperability framework is mainly configured on the SLASC Bridge as far as SLA configurations are concerned. The SLASC Bridge is deployed as an architectural layer and serves as the main entry point for the represented SLA terms on the ledger. As illustrated in Figure 4, the corresponding SLA application that is integrated, emits the contractual terms of an SLA configuration and the SLASC Bridge is receiving the respective data.

In the context of Pledger, this component allows for behavioural automation of the corresponding SLAs defined in the SLA Lite which eventually rely on ConfService, ending up with SLA terms bridged to Pledger blockchain network. The elaboration on the SLA workflow that iterates inside Pledger, and among the corresponding integrated components, follows.

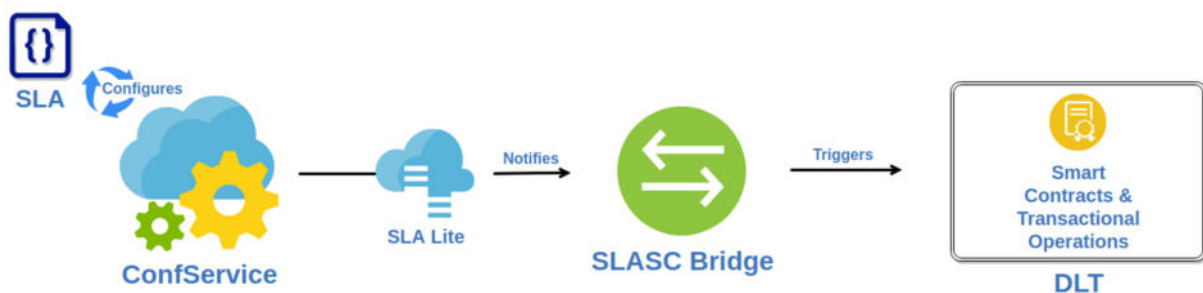


Figure 5: SLASC Bridge Interactions

Document name:	D4.5 Smart Contracts and DApps implementation tools II	Page:	16 of 42
Reference:	D4.5	Dissemination:	PU
		Version:	1.0
		Status:	Final

In Figure 5, the corresponding SLA workflow that unfolds inside Pledger is depicted. The main scenario of the SLASC Bridge workflow and interactions starts from the configuration of an SLA in the ConfService through the SLA Lite. Afterwards, the representation of the SLA contractual terms arises through the SLASC Bridge which resolves the deployment of the equivalent smart contract structures on the DLT network. In each SLA configuration that is completed by the ConfService, a respective notification is sent to the SLASC Bridge. The configured SLA parameters that originate from the ConfService and are included in the notifications sent between the two components, are then consumed by the SLASC Bridge that initiates the process of the smart contract structure creation. The SLASC Bridge defines a new structure that constitutes the smart contract equivalent of the received SLA. The smart contract equivalent is to be submitted on the ledger in the final step. During the SLASC Bridge operation, smart contract equivalents of SLA configurations are formed and ready to be submitted on the ledger as a decentralized representation of the SLA parametrization. Additionally, the received SLAs are automatically included in the refunding mechanism that is deployed on the DLT, and will be described in the next paragraph. In the final step, the equivalent smart contract structures are submitted on the blockchain network. The actuation of the SLA parameters consolidates through the latter's deployment of their representative models on the ledger. Particularly, the entire workflow focuses on the manifestation of an SLA configuration with its smart contract equivalent inside the trusted network while joining with ease the deployed wallet management that occurs on the DLT, as described later in 3.2.4.

In order for the SLASC Bridge to handle on the ledger violations that occur on SLAs, a refund mechanism has been developed and deployed on-chain. The refund mechanism offers on-chain value compensation for the SLA involved parties. Particularly, the refund mechanism operates on the DLT and adopts a punishment and reward scheme in the form of microtransactions for its definition. As in the case of each SLA configuration, the corresponding actors, usually the provider of a service and the consumer of it, participate in the refunding mechanism through their wallets. In particular, on the delivery of a service from the respective provider actor, corresponding SLA violations might occur. The refund mechanism implementation manages the economic balance of the system and offers microtransactions compensation to the consumer of a service. On the occurrence of an SLA violation, a refunding takes place. The provider actor is punished and their wallet balance is decreased, while the respective consumer is rewarded for the service violation and their balance is increased by the same amount (microtransaction adoption). The refund mechanism that is deployed on the DLT, is automatically triggered by the SLASC Bridge when necessary, i.e., a new SLA violation occurs, while it serves the dedicated balance compensations of the related actors.

3.2.3 Wallets

In the second iteration, the wallets functionalities have been added providing the necessary scenarios to the corresponding users that interact with the blockchain ledger. Wallets determine the de facto way for a user to directly interact with their data on the ledger. Wallets support specified data access to trusted users that is defined with regards to the corresponding functionality the latter should be offered. For instance, in an SLA configuration case, the trusted user can access their wallet and determine the amount left on their balance. Another example constitutes the corresponding use cases of the project, where the trusted user is able to perform range queries on the information that is stored on the blockchain and access the response. All the functionality is provided through the user wallet (Figure 6). As far as the high-level architecture of the Blockchain subsystem is concerned, the Wallets are handled by the Wallet Manager component of the subsystem (Figure 2).

Document name:	D4.5 Smart Contracts and DApps implementation tools II	Page:	17 of 42
Reference:	D4.5	Dissemination:	PU
		Version:	1.0
		Status:	Final

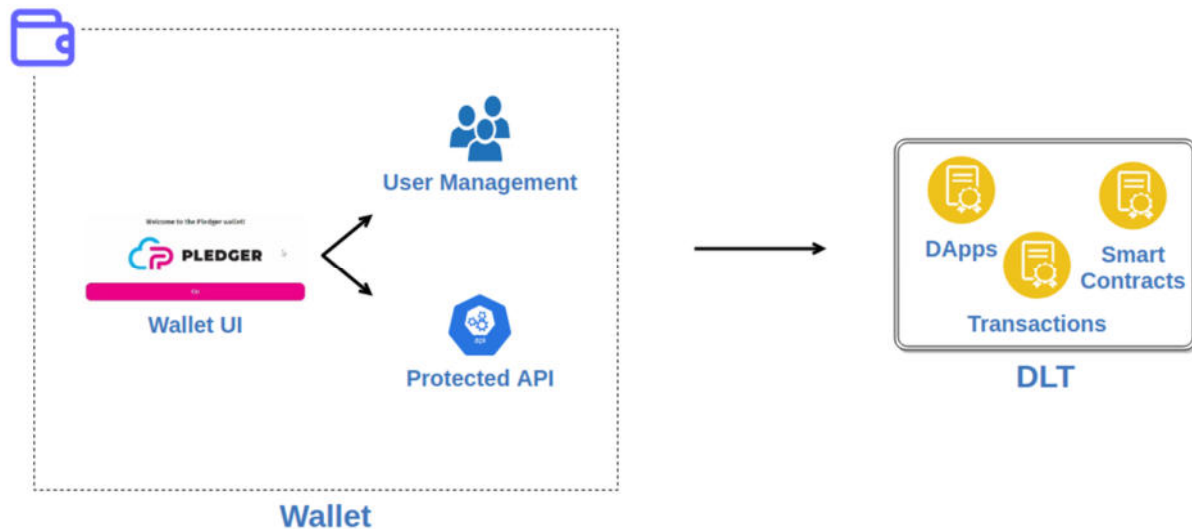


Figure 6: Wallet capabilities and interactions

In Figure 6, Pledger Wallet capabilities and interactions are depicted. As far as its interactions with the corresponding DLT operations are concerned (i.e. transactions submission, DApps, smart contracts), the Wallet backs the related blockchain functionalities of the respective Pledger cases. For instance, a trusted user is able to perform time range queries in the corresponding Use Case (UC) DApps through the Wallet. Finally, the following sub-sections describe the capabilities that are included in Pledger Wallet (Figure 6).

3.2.3.1 Wallet User Interface (Wallet UI)

The Wallet UI is a browser add-on formulated in order to act as the user interface for the protected API. It provides all the aforementioned functionalities, such as entering the keys of a user, viewing the corresponding user balance, and performing range queries on DLT stored data.

3.2.3.2 User management

The user management service is responsible for handling the addition of users on the DLT. Users on the DLT need to be verified so that they can interact with it and use its functionalities. The user management service generates the necessary crypto material for newly created users and provides them back to each user. A user's public key can be verified by the DLT network through signing a transaction or message, while a user's private key is known only to the user. When a user signs a message with their private key, the user management service verifies the validity of the credentials. The user management service is accessible only by authorized and trusted users that participate to the network while the user accounts can be created using the corresponding user credentials.

3.2.3.3 Protected API

A dedicated API is formed in order for trusted users to interact with the wallets. The protected API provides its dedicated endpoints which the users can connect in order to get the information from the available smart contracts and UC DApps. The wallet authorization process occurs through the use of the corresponding public and private keys. The users provide their credentials and after they get verified, they can access the available information and functionalities:

- ▶ Get the balance of a user based on the associated SLA configuration – In the SLA configuration case, the users need to be able to view their balance.
- ▶ Get information about related Vulnerable road user (VRU9 incidents in a specific time range – Users can access the number of VRU incidents that occurred in the time range they select.
- ▶ Get information about parts quality degrees in a specific time range – Users can get the number of parts manufactured quality in the time range they select.

Document name:	D4.5 Smart Contracts and DApps implementation tools II	Page:	18 of 42
Reference:	D4.5	Dissemination:	PU
	Version:	1.0	Status: Final

3.3 Interfaces provided

The latest version of the DLT endpoint capabilities is currently supporting the necessary interoperability mechanisms that ease the integration with the emitted events by the related subsystem functional components. The main endpoints structure to connect with the DLT are available through the SLASC Bridge and Wallet components (3.2). The defined endpoints are set chiefly to leverage the asynchronous communication functionalities with other Pledger required dependencies.

3.4 Data models and functionalities

In this section the goal is to describe the respective data models and functionality interactions between the DLT and its architectural components deployment that is completed through smart contracts and UC DApps. As the underlying blockchain deployments serve the different cases needs inside the project, the following sub-sections clarify how dissimilar data can be handled, transferred, and processed on-chain by the respective smart contracts and UC DApps.

3.4.1 SLAs

The SLA configuration data model and their functional states are described in Section 3.6 of the second iteration deliverable about SLA assessment and negotiation processes, D3.6 "QoS and SLA assessment and negotiation tools II" [9].

Particularly, SLASC Bridge implements the following necessary functionalities and transactional operations when receiving SLA data from the corresponding endpoints of the core platform.

- ▶ *Instantiate a user inside the DLT* – add a new user with a specific balance on the list of users.
- ▶ *Retrieve a user's balance* – get the number of tokens the specific user has.
- ▶ *Mint tokens for a user* – generate tokens for the user.
- ▶ *Transfer tokens from one user to another* – transfer tokens between users; it can be used by both smart contracts and DApps.
- ▶ *Generate a new SLA configuration* – creates a new SLA configuration for the corresponding users, defining the violation cost, the status, and others.
- ▶ *Retrieve an SLA configuration* – returns the details of an SLA configuration, as well as the number of times it has been violated.
- ▶ *Retrieve a user's data* – get the user's information, together with their balance.
- ▶ *Update a user's balance* – add balance to a user through the respective smart contract.
- ▶ *Remove an SLA configuration* – remove an SLA configuration from the DLT.
- ▶ *Transfer tokens on SLA violation* – on an SLA violation, the corresponding amount of tokens are transferred from the one user (provider actor) to the other (client actor).

3.4.2 Secure handshake

Head-Mounted Devices (HMDs) are limited in their processing and computation abilities and thus struggle to provide a smooth experience for the user when the application size (polygon amount) gets too high. To mitigate this, it has to either reduce the quality or reduce the frame rate and both situations cause the user to have a bad experience. The Pledger AR Workspace solves this by running the intensive calculations on a Server (Laptop, etc.) with a powerful GPU and displays the results on the Client (HMDs). For this peer-to-peer connection to be established between the Server and Client, an initial exchange of sensitive data needs to take place. To secure this handshake (signalling), the exchange is secured through the establishment of the Whisper protocol enabled on blockchain nodes.

Document name:	D4.5 Smart Contracts and DApps implementation tools II	Page:	19 of 42
Reference:	D4.5	Dissemination:	PU
	Version:	1.0	Status: Final

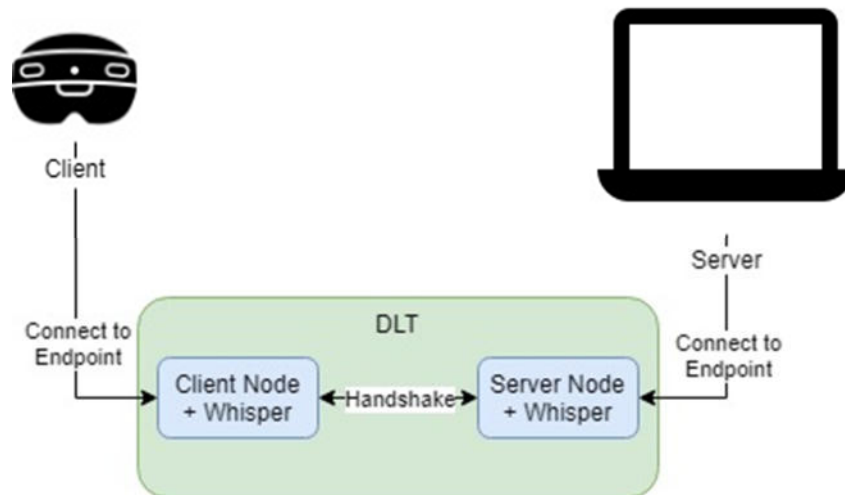


Figure 7: Whisper handshake in UC1

The handshake process is successfully installed in a private blockchain where the Whisper protocol (shh) activated. A blockchain node specified access point (Endpoint) allows to send messages securely between the Server (Laptop) and the Client (Hololens2). To further secure the process, a reverse proxy setup on the corresponding endpoints upgrades the communication protocol from HTTP to HTTPS.

Table 6: UC1 Handshake entries

Key	Value (type)	Description
Version (Server)	String	Signalling Version
SDP Offer (Server)	String	Negotiate the session's parameters (sent to client)
ICE Offers (Server)	String	Standard method of NAT traversal (sent to client)
SDP Offer (Client)	String	Negotiate the session's parameters (sent to server)
ICE Offers (Client)	String	Standard method of NAT traversal (sent to server)

In the second iteration, the established Whisper protocol allows for secure and confidential message exchange between two parties, and is offered as part of the Pledger blockchain. The protocol security functionality is mostly provided for the private communication of the Server and the Client actors in the corresponding use case (UC1). However, the nature of its architecture and the setup that is deployed in the project allows for its adoption from other interested entities. The related deployment subscribes to the following sequence of actions:

- ▶ Select new signalling pair – the two (2) candidate parties are picked in order to exchange messages using the Whisper protocol.
- ▶ Connect candidate parties – both candidate parties receive their digital identities and are ready to exchange messages using the Whisper protocol.
- ▶ Signalling – both parties are exchanging secure messages through the Whisper protocol.
- ▶ Disconnect – both participants leave the protocol.

3.4.3 Risk detection, notification and event log

The most relevant events in UC2 are those related to VRUs approaching a “danger zone” near a tram stop, and the arrival of trams to the same stop. Every time a VRU enters the transmission range of the Road Side Units (RSU), it starts sharing its position with the vehicular software stack to which applications can subscribe. Whenever a tram or bicycles are approaching the stop at the same time, the risk detection app will be triggered, raising an alert message that will be broadcast to all VRUs within the danger zone. The algorithm responsible for this detection of risks allows to define two areas of detection for the bicycles: the distant area (tens of meters before the tram stop) and the close area right next to the stop. By distinguishing these areas where bicycles can be located in, two levels of risk detection are defined: the early detected risk and detection and the imminent risk detection, respectively.

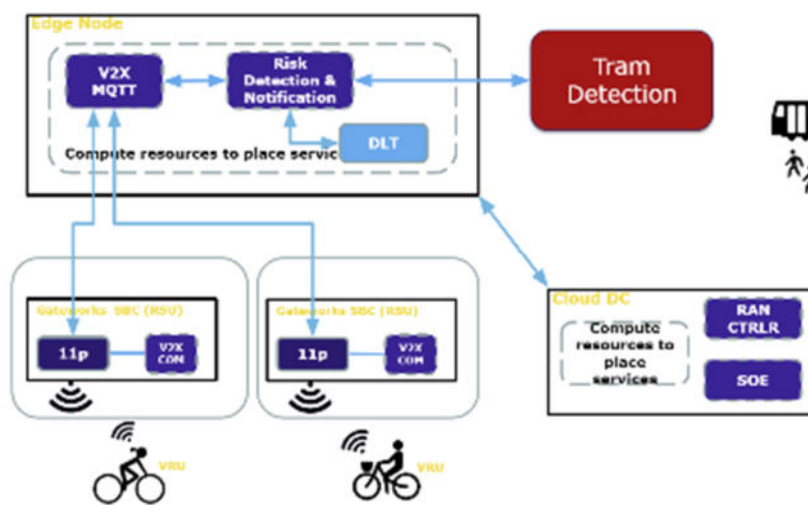


Figure 8: Block diagram of UC2

All these events (bicycle enters or leaves an area, tram arrives at or leaves the station, early or imminent risk detection) generate certain data that must be securely saved in an immutable way. Through the DLT the corresponding management of this kind of data can occur with focus on privacy. In fact, this information can include the actual event, but also metadata, such as the location of bicycles or the number of bicycles in the vicinity when an event happens. Since some of this data is sensitive, and in order to guarantee data privacy, relevant data is written on Pledger’s DLT chain. Figure 8 represents the block diagram of UC2 and the interconnection needed between the DLT and the application-specific modules.

There are two distinct categories of data to be dumped on the DLT:

- ▶ Data about bicycle position. This information can be associated with one of the events mentioned previously and it can be dumped on the chain approximately every three minutes, and have a size of around 1MB.
- ▶ Alerts generated by the risk detection and notification system: these alerts are generated every time a risky situation is detected, and they are stored with additional metadata. These messages are dumped on the chain as required, and they have an approximate size of 100KB.

Table 7 summarises the essential set of data entries that should be written on the DLT chain:

Document name:	D4.5 Smart Contracts and DApps implementation tools II	Page:	21 of 42
Reference:	D4.5	Dissemination:	PU
		Version:	1.0
		Status:	Final

Table 7: UC2 on-chain essential set of data entries

Log entry	Field 1 (type)	Field 2 (type)	Field 3 (type)
Tram enters station	Timestamp (string)	Count number of OBUs (On-Board Units) in the vicinity (integer)	Count number of pedestrians in the vicinity (integer)
Tram leaves station	Timestamp (string)	Count number of OBUs in the vicinity (integer)	Count number of pedestrians in the vicinity (integer)
VRU enters tram stop danger zone	Timestamp (string)	OBU ID (MAC address)	Type of VRU (string)
VRU leave tram stop danger zone	Timestamp (string)	OBU ID (int)	Type of VRU (string)
Risk detected	Timestamp (string)	OBU ID (int)	Type of VRU (string)
Collision imminent	Timestamp (string)	OBU ID (int)	Type of VRU (string)
Tram comes to a halt	Timestamp (string)	Count number of OBUs in the vicinity (integer)	Count number of pedestrians in the vicinity (integer)
External warning sign (VRU approaching tram stop danger zone)	Timestamp (string)	OBU ID (int)	Type of VRU (string)
External warning sign (VRU leaving tram stop danger zone)	Timestamp (string)	OBU ID (int)	Type of VRU (string)

In the second period, with regards to the corresponding smart contract and transactional operations for the current use case, the following actions have been implemented by the related UC DApp:

- ▶ Generate a new VRU entry – add incoming data regarding VRUs to the DLT.
- ▶ Retrieve a VRU entry – return the requested information regarding a specific VRU timestamp.
- ▶ Retrieve time range VRU entries – return the requested information of the VRUs within a specific time period.
- ▶ Retrieve time range VRU entries risk levels – return the requested information of the VRUs within a specific time period as far as risk level is concerned.

3.4.4 Logging information about produced parts

In UC3, information to determine the process stability is gathered from the machine and analysed by analytical services, including average air consumption, average electrical consumption and parts produced. This information gathered about parts produced by the machine including the start and end time as well as information about sub-processes and quality is classified as sensitive. Especially the number of parts produced as well as the information about the quality and resulting number of rejects allows a conclusion to be drawn about the ongoing process as well as how the business is doing. As this kind of data should be confidential among the machine builder and the client, the corresponding secure environment where the exchange occurs privately is provided by the Pledger DLT. This kind of information will be stored on-chain in order to enable access only to authorized parties, e.g. FILL as machine builder and the according customer owning the process.

The analytical service of type Basic Analytics (Parts Produced) gathers relevant data from the message broker, analyses it and pushes the result back to the RabbitMQ, from where it should be stored on the chain. The later connection happens seamlessly through the StreamHandler that populates the dedicated data to the DLT network. The process is illustrated in Figure 9.

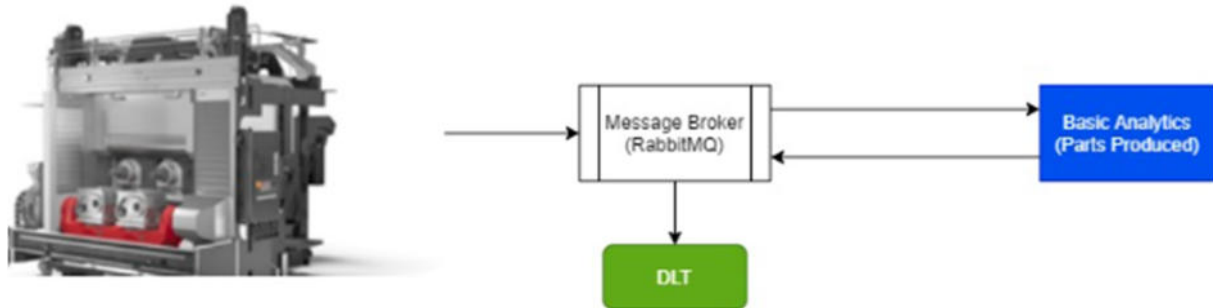


Figure 9: Parts produced data in UC3

The analytical services' result is one document object containing all relevant information about the produced part. This document constitutes one log entry on the blockchain with all information to uniquely identify a part. In Table 8, a summary of this information is given. The amount of subprocesses depends on the type of machine, for Pledger the machine has four subprocesses, where examples of two are given. These values are arrays to account for repeated subprocesses.

Table 8: UC3 on-chain data entries of one part

Key	Value (type)	Description
Start	DateTime	Start of part (Loading)
Stop	DateTime	Stop of part (Unloading)
CycleTime	Double	Cycletime
TargetCycleTime	Double	Targetcycletime
Quality	Integer	Info whether quality of part is OK (1) or not OK (0)
ElectricityConsumption	Array of Doubles	Electricity consumed by the machine
Subprocess1Starts	Array of DateTimes	Starting dates for Subprocess 1
Subprocess1Stops	Array of DateTimes	Stopping dates for Subprocess1
Subprocess1Times	Array of Doubles	Durations of Subprocess 1
Subprocess2Starts	Array of DateTimes	Starting dates for Subprocess2
Subprocess2Stops	Array of DateTimes	Stopping dates for Subprocess2
Subprocess2Times	Array of Doubles	Durations of Subprocess2

Key	Value (type)	Description
SpindleNumber	Integer	Number of the spindle which processed the part
NOK reason 1	Boolean	Reason 1 for part being not OK
NOK reason 2	Boolean	Reason 2 for part being not OK
Code	String	Unique identifier code of the part
Name	String	Type of the part (e.g. cylinder head)

For the second period, the associated UC DApp carries out the following actions in relation to the relevant smart contract and transactional operations for the current use case:

- ▶ Generate a new parts entry – add incoming data regarding parts to the DLT.
- ▶ Verify a parts entry exists – return the requested information regarding a specific part manufactured at the specified timestamp.
- ▶ Retrieve time range parts entries – return the requested information of the parts manufactured within a specific time period.
- ▶ Retrieve time range parts entries quality degrees – return the requested information of the parts manufactured within a specific time period as far as quality degree is concerned.

4 Installation and usage guides

4.1 Requirements

In order to create, orchestrate and deploy the prototype, the following technologies are required:

- ▶ Docker 1.18+
- ▶ Bash 4+
- ▶ Kubernetes 1.19+

Additionally, tools like **jq** [10] are needed along with sophisticated configuration of the necessary manifests that are completing the migration to Kubernetes. In the latest iteration, the aforementioned configuration of the manifests has been completely developed, and is provided seamlessly to the users, who can install and use the entire prototype while such configurations are deployed automatically on the background.

4.2 Installation

In this section, the installation of the subsystem adheres to the following steps: the installation first describes and guides through the necessary references for the 4.1 requirements, and then, it follows with the installation of the subsystem. The prototype is deployed using the Pledger CI/CD platform by accessing the respective images from the dedicated private [Docker registry](#).

Requirements installation.

- ▶ Docker 1.18+: Install Docker from: <https://docs.docker.com/engine/install/ubuntu/>
- ▶ Bash 4+: Linux should be the de facto operating system when trying to run this Pledger prototype. Linux users should have a suitable version already installed on their machine, or install Bash from: <https://launchpad.net/ubuntu/+source/bash>
- ▶ Kubernetes 1.19+: Install Kubernetes from: <https://kubernetes.io/docs/tasks/tools/install-kubectl-linux/>
- ▶ jq: Install the tool from here: <https://stedolan.github.io/jq/download/>

Subsystem installation.

- ▶ Clearing the environments in order to setup up a clean deployment.

```
### Set network down:
./network-deploy.sh down
./scripts/build-images.sh # if images are not ready
```

- ▶ Starting the blockchain network.

```
### Set network up:
./network-deploy.sh up -ca
```

- ▶ Check that the network is up and running.

Ensure on the health of the containers. In the Kubernetes deployment the following command can be applied:

```
kubectl -n core get deploy
```

Document name:	D4.5 Smart Contracts and DApps implementation tools II	Page:	25 of 42
Reference:	D4.5	Dissemination:	PU
	Version:	1.0	Status:
			Final

4.3 Usage

Since the first iteration, the network and the respective components are installed and available on the ENG Kubernetes cluster. In this iteration, the usage of the defined DLT, the SLA supporting smart contracts and use case DApps are holistically deployed on the network in order to serve the dedicated requirements. Particularly, the users and developers are able to utilize the current prototype through the fundamental components of the DLT, the SLASC Bridge, and through Wallets. The rest of the corresponding architectural components, i.e. Wallet Manager, Blockchain Reader/ Writer, Smart Contract Manager, and Transaction Handler, are developed on the backend of the entire blockchain framework while taking significant role on the latter components' functionalities and capabilities.

In this section, users and developers are able to understand the ways they can handle the subsystem and also perform a health check.

- ▶ Create the private environments that host the different supported cases data, as far the SLAs and the data coming through the SLA Lite are concerned.

Up private environments:

```
./network-deploy.sh createChannel -c sla  
./network-deploy.sh createChannel -c vru  
./network-deploy.sh createChannel -c parts
```

- ▶ Deploy the SLASC Bridge component that receives the SLA data through the SLA Lite and performs the dedicate workflow on the chain.

Deploy SLASC Bridge:

```
./chaincode/cc.sh slasc 1 sla
```

- ▶ Ensure the correct deployment of SLASC Bridge.

Last message on terminal should be similar to:

Query chaincode definition successful on peer0.org2 on private environment 'sla'.

4.4 Licenses

Apache License 2.0 [11]

4.5 Source code repository

The source code repository of the DLT is accessible within the public repository of Pledger at <https://gitlab.com/pledger/public/blockchain>.

Document name:	D4.5 Smart Contracts and DApps implementation tools II	Page:	26 of 42				
Reference:	D4.5	Dissemination:	PU	Version:	1.0	Status:	Final

5 Demonstration

This section describes the updated scenarios developed during the second period.

5.1 Scenarios Description

5.1.1 SLASC Bridge

The SLASC Bridge component addresses two (2) main objectives (Figure 10) of the project with regards to SLA configuration on the ledger and the way that the SLA parameters and violations are represented and handled on the blockchain. First of all, SLASC Bridge materializes the SLA contractual terms on the blockchain, e.g., the "SLA to Smart Contract Process" leads to the creation of the Contractual Terms Equivalents (Figure 10 and Figure 11). Particularly, an SLA configuration defined in the ConfService (and passed through the SLA Lite) is transformed into its smart contract equivalent structure that includes and describes the business intelligence (contractual terms and behaviour) of the SLA on the blockchain. Secondly, SLASC Bridge handles the corresponding violations of an SLA on the blockchain, i.e. the "SLA Violation Handling" leads to Violations Handled (Figure 10 and Figure 11). Specifically, the violated contractual terms of an SLA are tracked on the blockchain as per the respective smart contract equivalent, and any violation of the promised services triggers the dedicated compensation scheme on the ledger.

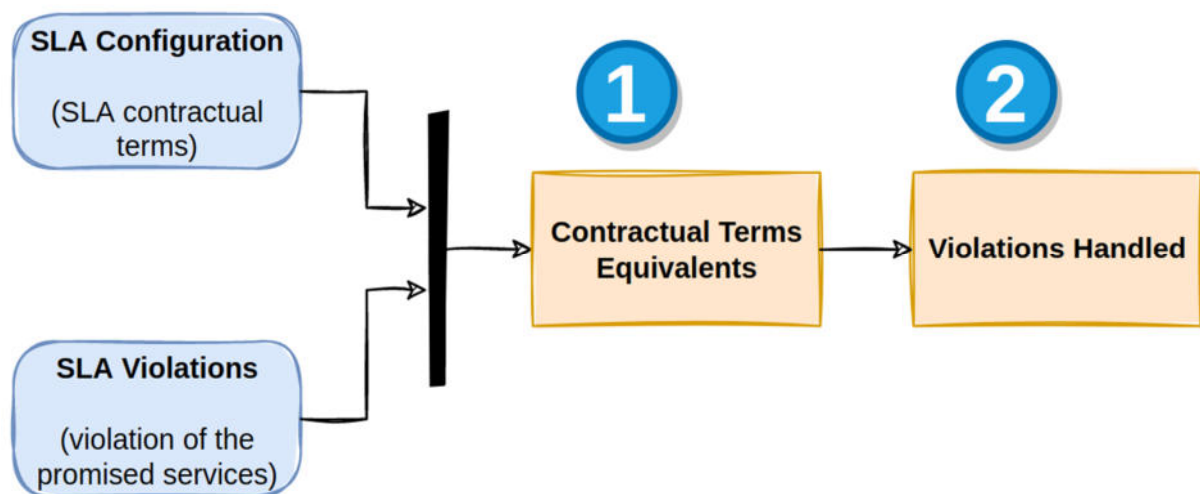


Figure 10: SLASC Bridge main objectives

Each of the objectives of SLASC Bridge are covered by dedicated steps that describe the lower-level actions performed in each case. Figure 11 depicts a general breakdown of the respective actions that occur on the backend.

Document name:	D4.5 Smart Contracts and DApps implementation tools II	Page:	27 of 42	
Reference:	D4.5	Dissemination:	PU	
	Version:	1.0	Status:	Final

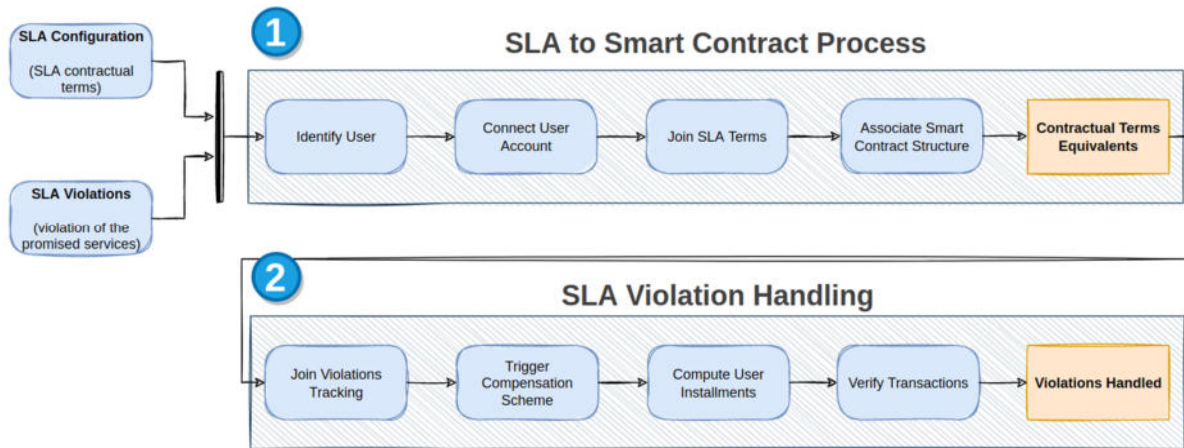


Figure 11: SLASC Bridge main objectives breakdown

5.1.1.1 Scenario 1: SLA to Smart Contract Process

In order to create the smart contract equivalent of an SLA configuration, the following steps occur during the "SLA to Smart Contract Process":

- **Step 1:** The SLA contractual terms (SLA configuration) are sent to the blockchain network from the ConfService through the SLA Lite.

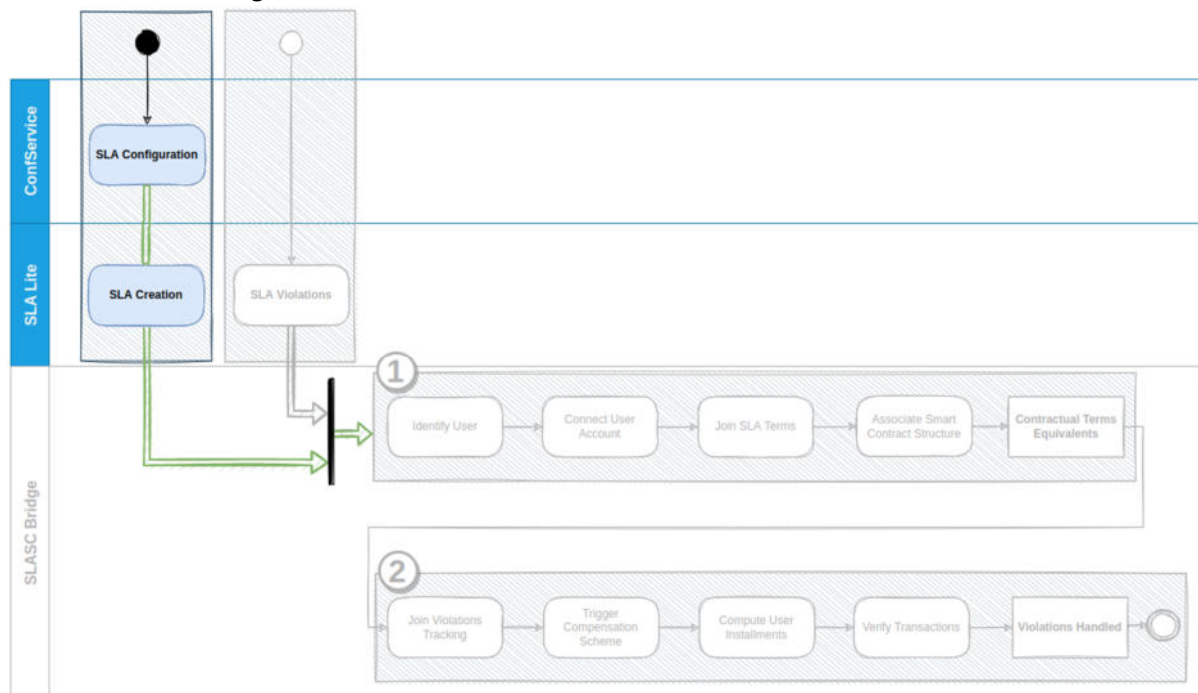


Figure 12: SLA configuration sent to SLASC Bridge

► **Step 2:** SLASC Bridge identifies the respective information for the participating users..

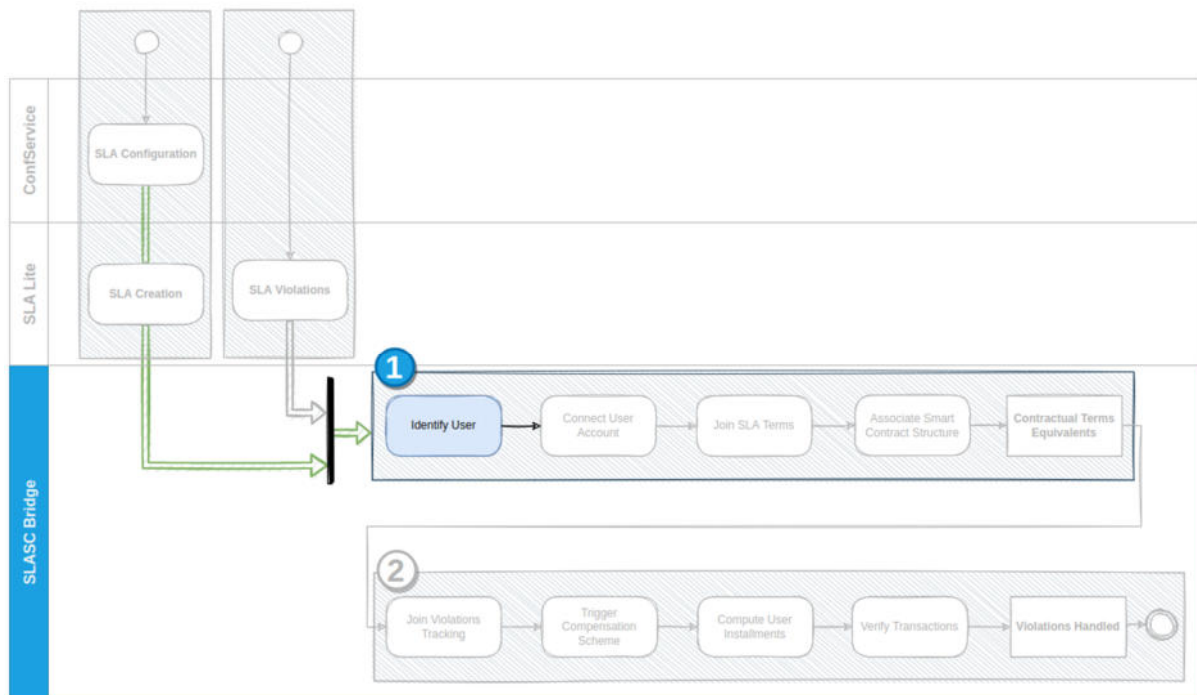


Figure 13: SLASC Bridge identifies the user

► **Step 3:** SLASC Bridge connects the user accounts on the blockchain. When an account does not exist yet, this step is responsible for creating a new account for the new user.

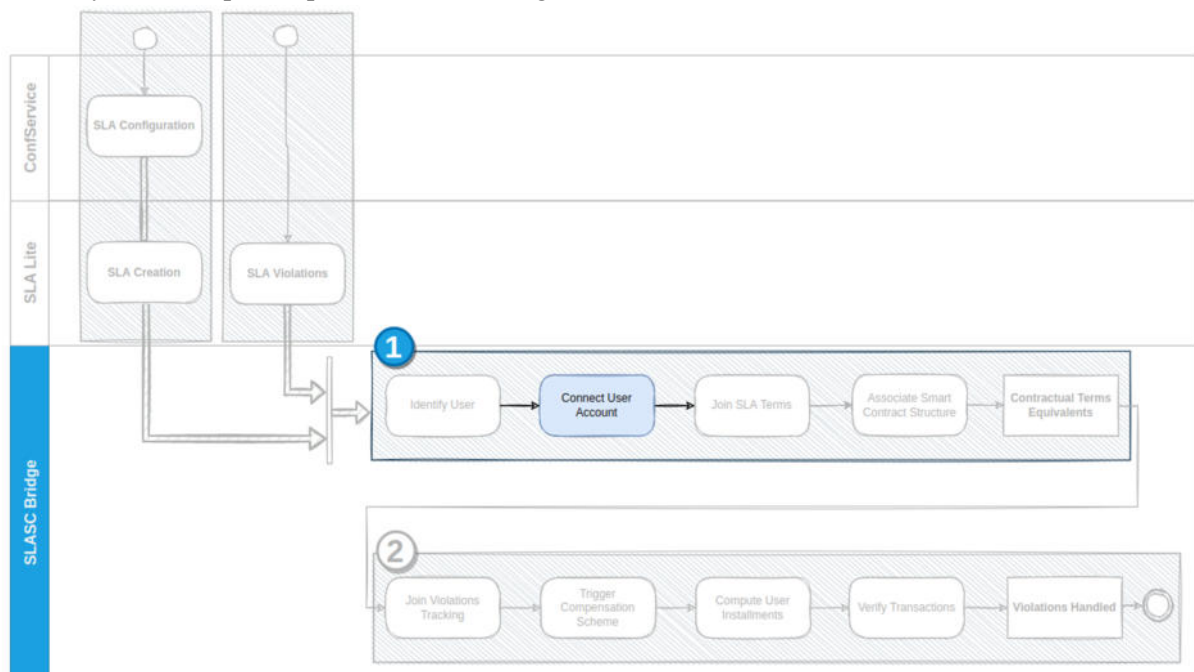


Figure 14: SLASC Bridge connects the required user account on the ledger.

- **Step 4:** SLASC Bridge encapsulates the SLA contractual terms of the SLA configuration with the involved users, and then manages the respective user accounts on the blockchain, in order to prepare for the smart contract structure creation.

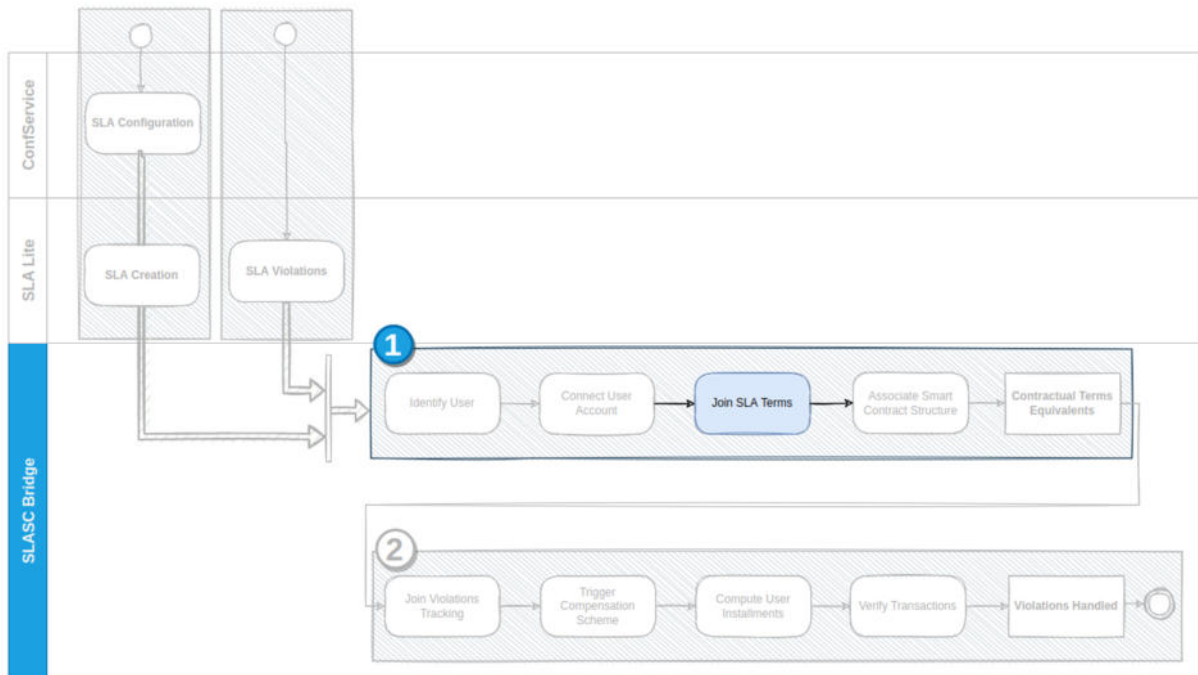


Figure 15: SLASC Bridge joins the contractual terms

- **Step 5:** SLASC Bridge associates the related SLA configuration on a new smart contract structure that defines the dedicated contractual terms on the ledger.

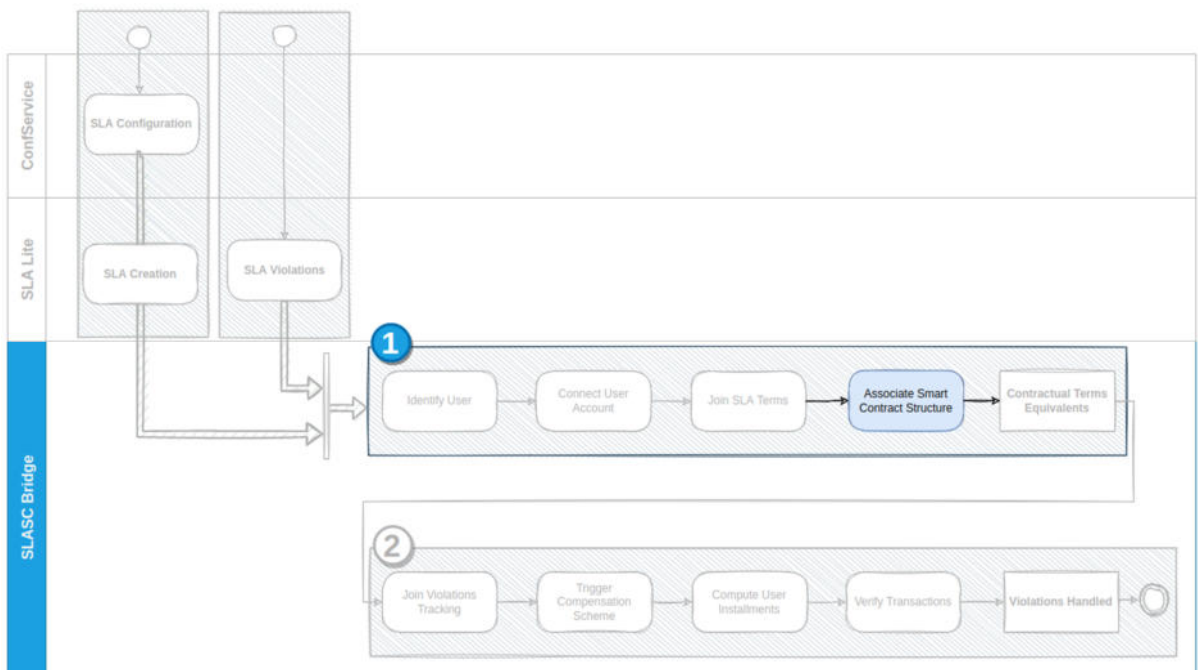


Figure 16: SLASC Bridge associates smart contract structure

► **Step 6:** In the final step of the "SLA to Smart Contract Process", the contractual terms equivalent for the SLA configuration is created (smart contract equivalent structure).

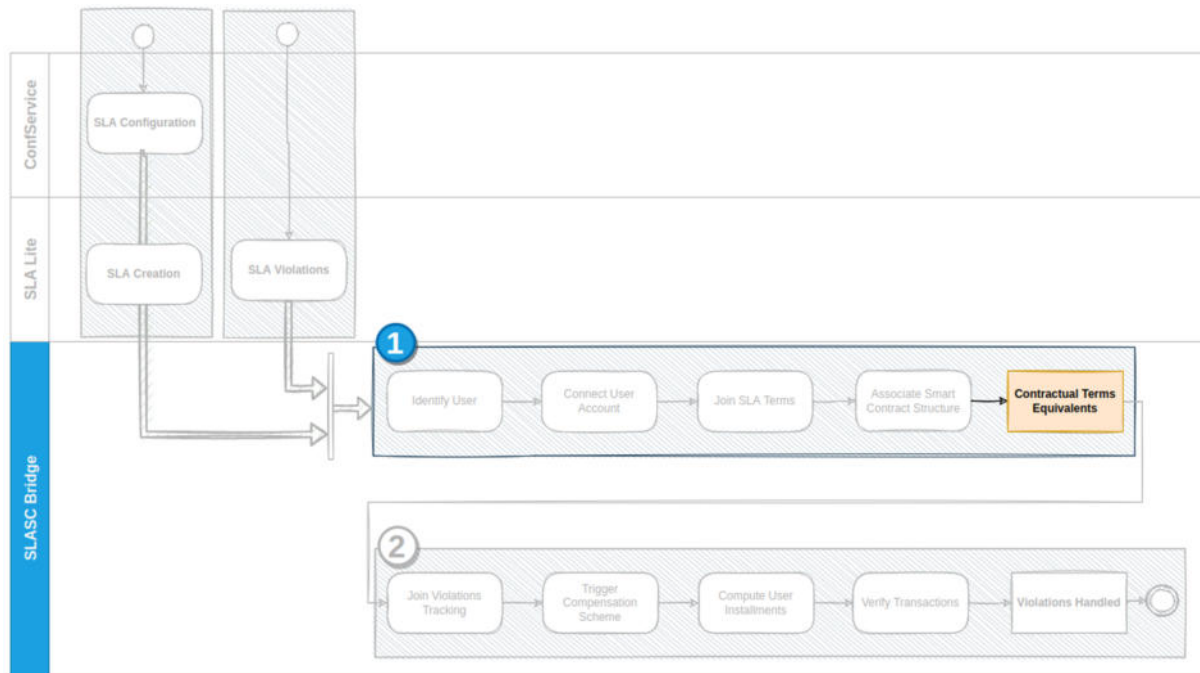


Figure 17: Contractual terms equivalent is created: SLA to Smart Contract Process finality.

5.1.1.2 Scenario 2: SLA Violations Handling

In order to handle the violations of an SLA configuration on the blockchain, the following steps occur during the "SLA Violations Handling":

► **Step 7:** SLA violations are announced to the blockchain from the SLA Lite application.

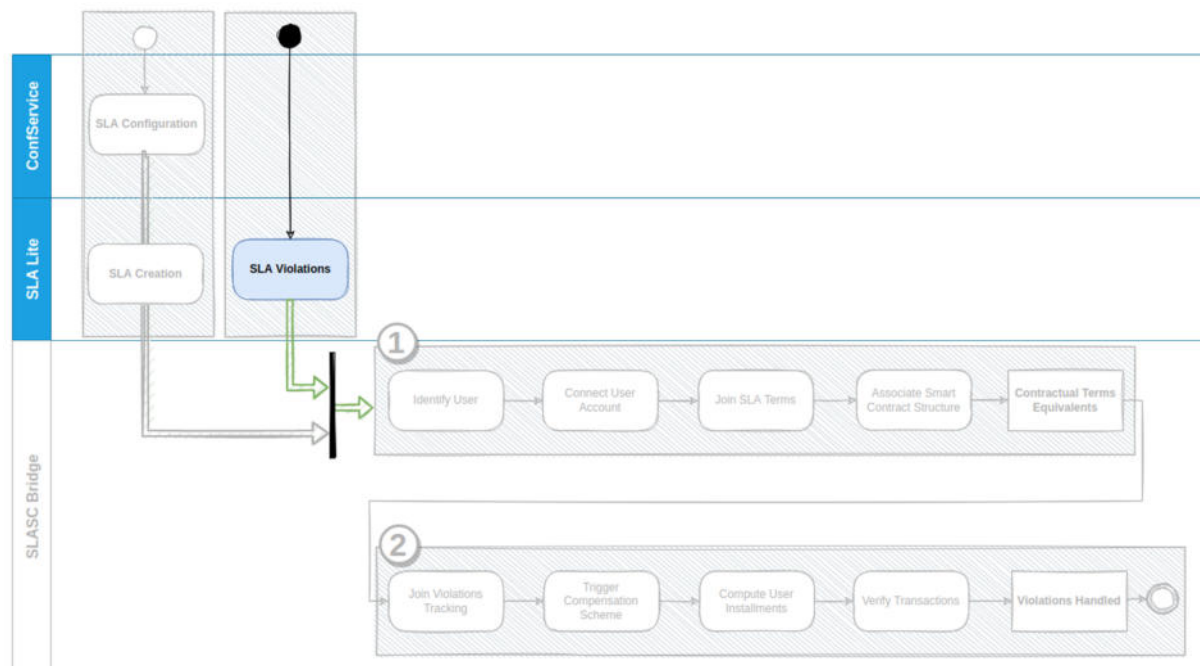


Figure 18: SLA violation sent to SLASC Bridge

- **Step 8:** SLASC Bridge retrieves and triggers the equivalent on chain smart contract structure for the specific SLA configuration.

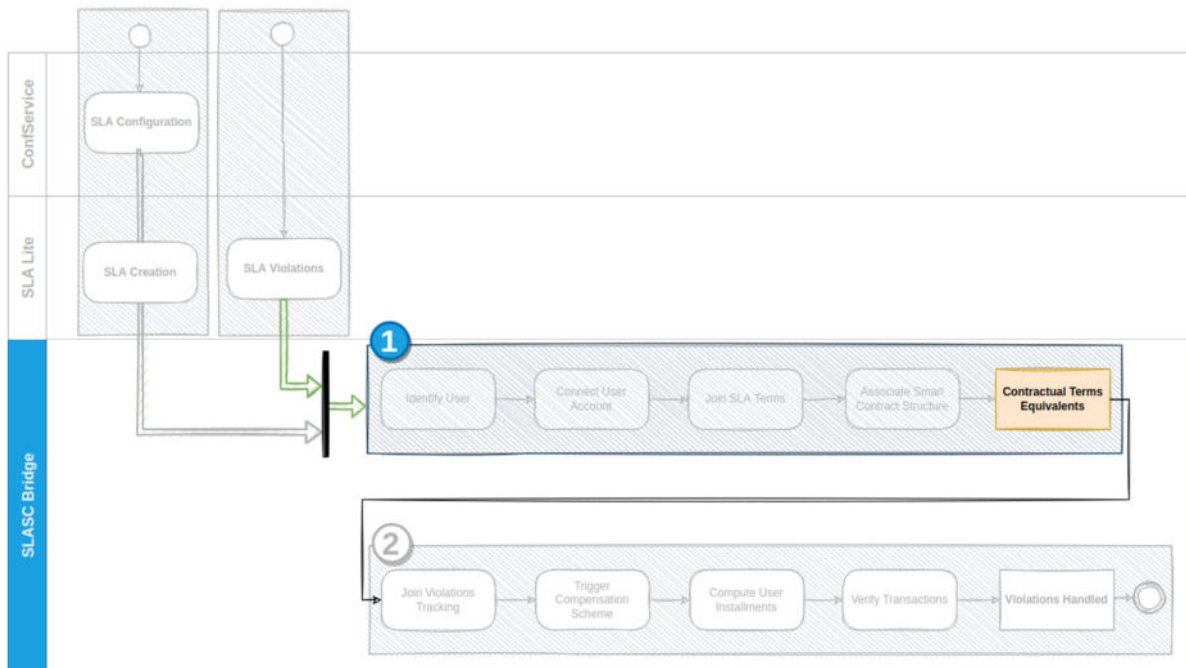


Figure 19: SLASC Bridge retrieves and triggers related smart contract structure

- **Step 9:** SLASC Bridge jointly triggers the violations tracking on the contractual terms on the ledger. In this step, SLASC Bridge is enabling the capability of detecting SLA violations for the corresponding smart contract structure of the initial SLA configuration.

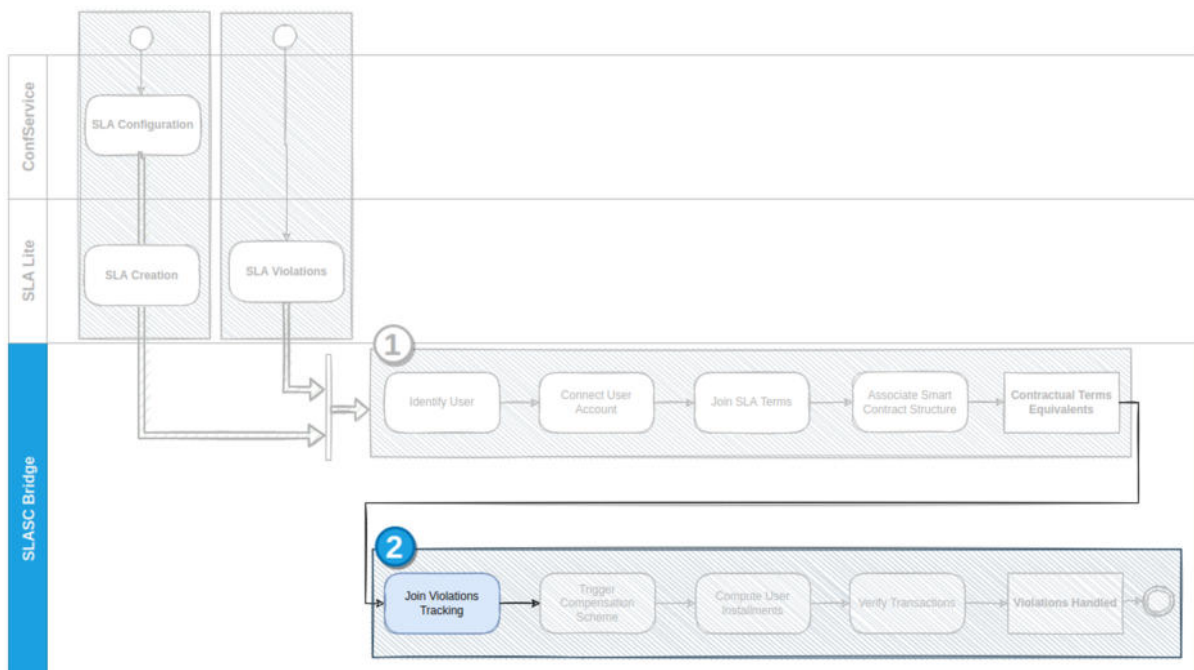


Figure 20: Join violations tracking

- **Step 10:** In this step, the refunding scheme that is deployed on the blockchain network is triggered in order to be able to provide the corresponding compensations for the SLA violation that occurred.

Document name:	D4.5 Smart Contracts and DApps implementation tools II	Page:	32 of 42
Reference:	D4.5	Dissemination:	PU
		Version:	1.0
		Status:	Final

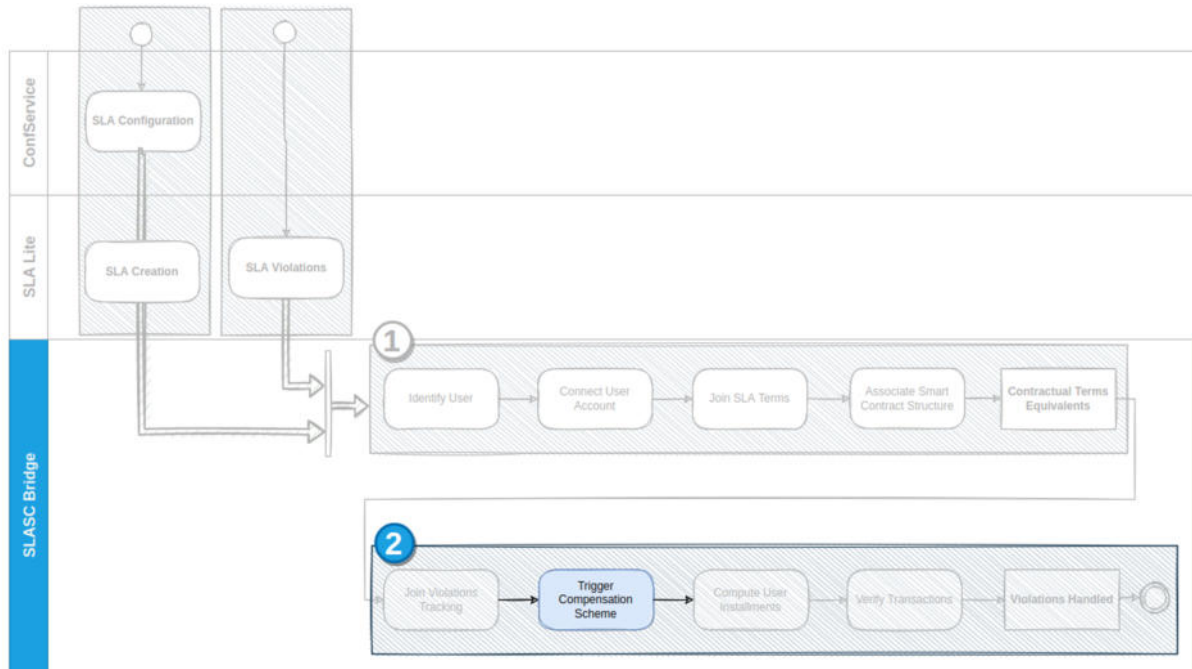


Figure 21: Trigger compensation scheme

- **Step 11:** SLASC Bridge is utilizing the refunding scheme that is deployed on the ledger in order to compute the user instalments to their accounts according to the occurred SLA violation.

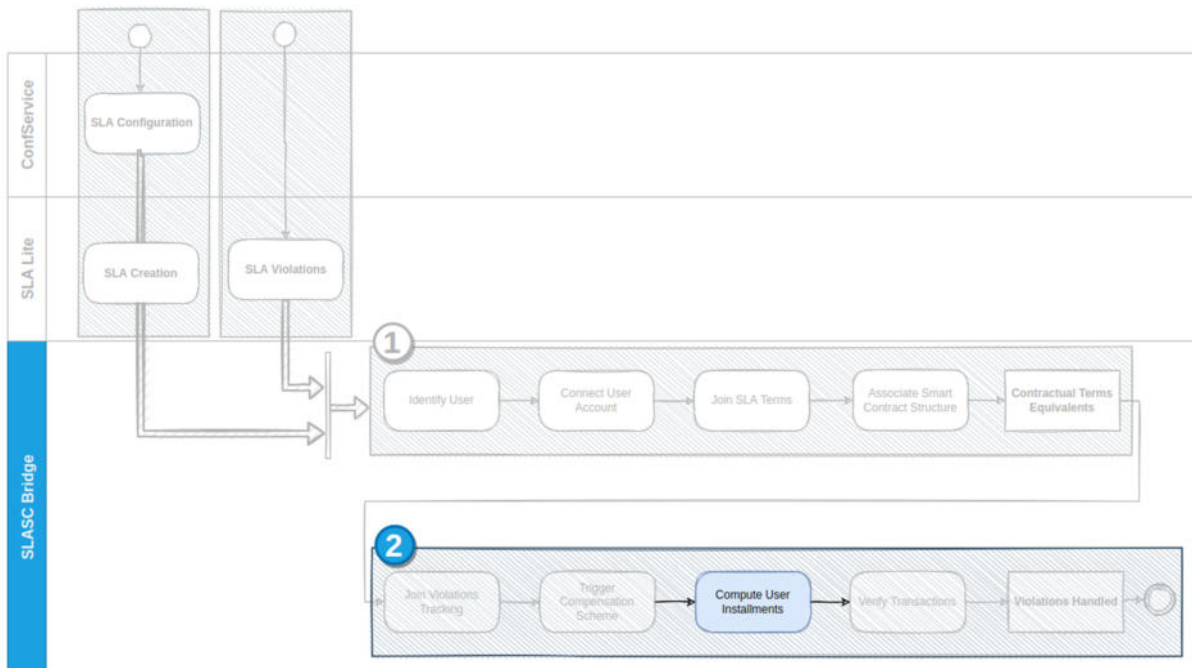


Figure 22: Compute user instalments

► **Step 12:** SLASC Bridge performs verification of the corresponding transactions on the blockchain in order to finalize the violations handling.

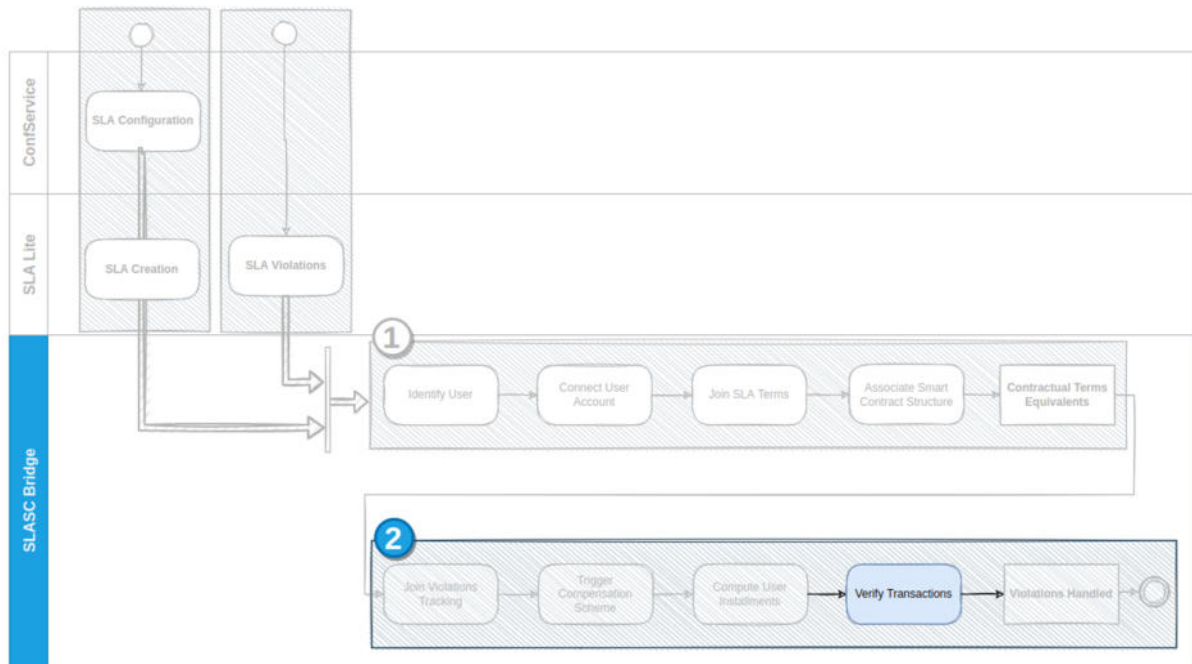


Figure 23: SLASC Bridge verifies transactions

► **Step 13:** On transaction violations finality, the scenario of "SLA Violations Handling" completes its life-cycle.

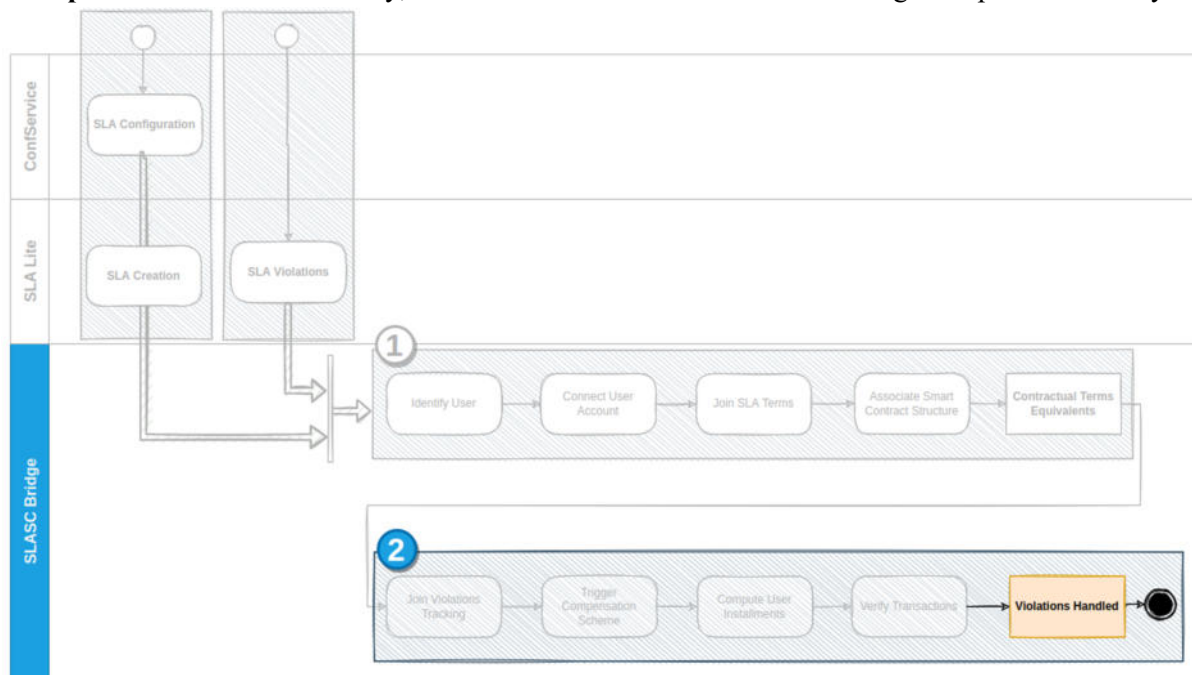


Figure 24: Violations handled by SLASC Bridge

5.1.2 Wallet

Wallet is used by trusted users in order to access authorized information on the ledger through a user-friendly interface. The wallet is applicable for the SLA configuration case, and the use cases 2 and 3. The steps described below comprise the scenario that is generally followed in the aforementioned cases.

- ▶ **Step 1:** Trusted users connect to the Wallet using their public and private keys.
- ▶ **Step 2:** Trusted users access the related information available for the respective and dedicated permitted access.
- ▶ **Step 3:** (Optional) Trusted users perform specific time range queries over sensitive data securely stored on the DLT.

Figure 33 depicts the landing screen of Pledger Wallet where trusted users are able to login and access the dedicated and individually permitted information.

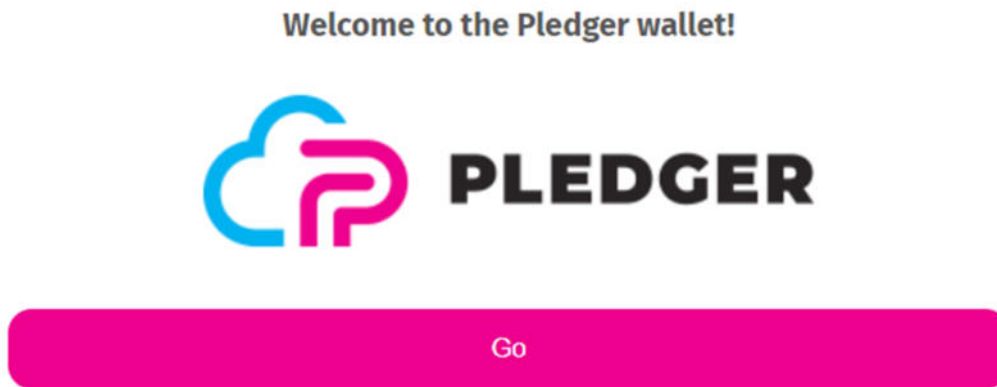


Figure 25: Pledger Wallet landing screen

In order to login to the Wallet, the user enters the trusted credentials provided, as depicted in Figure 26

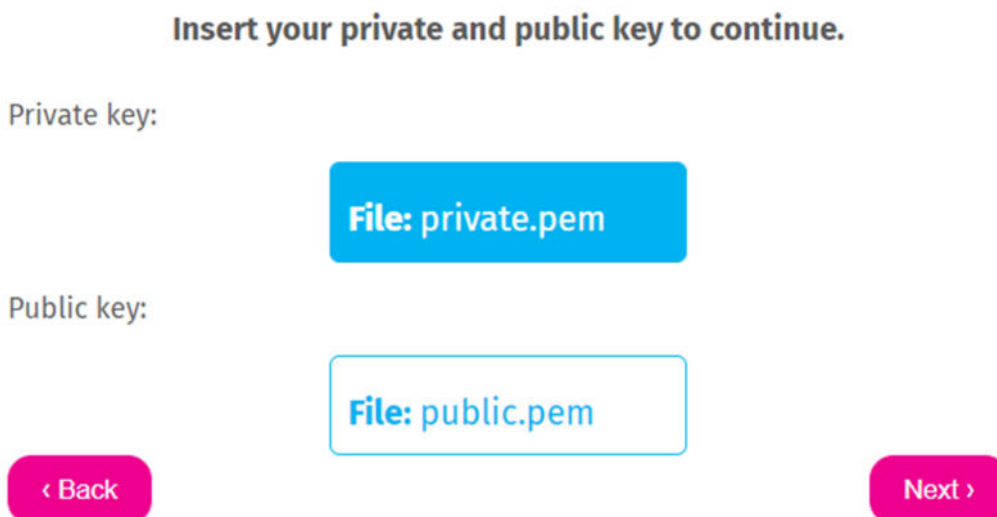


Figure 26: Pledger Wallet credentials

Document name:	D4.5 Smart Contracts and DApps implementation tools II	Page:	35 of 42
Reference:	D4.5	Dissemination:	PU
	Version:	1.0	Status:
			Final

5.1.2.1 SLA Configuration

On an SLA configuration, the respective user is able to view their balance on the network, as depicted in Figure 27.

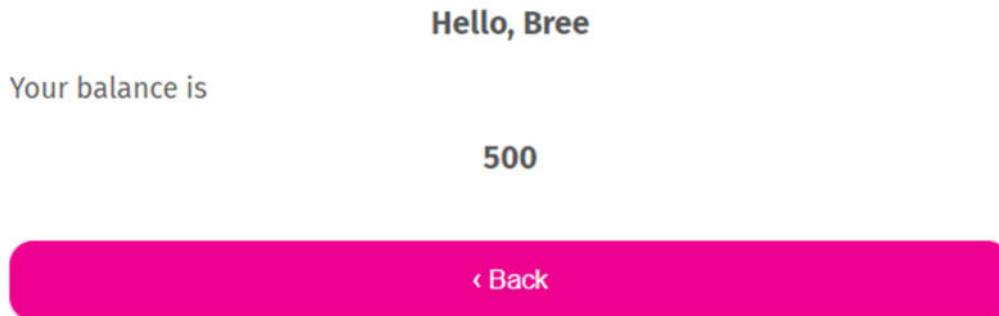


Figure 27: User Wallet balance

5.1.2.2 Risk detection, notification and event log

During the event logging on the DLT, trusted users are able to perform time range queries for the risk level that incidents had during the selected time period. The process is depicted in Figure 28.



Figure 28: Different results for time range queries on risk level of VRU incidents

5.1.2.3 Logging information about produced parts

In the case of manufactured parts information logging, trusted users are able to perform time range queries on the quality degree of the machine parts that were manufactured during the selected time period. The process is depicted in Figure 29.

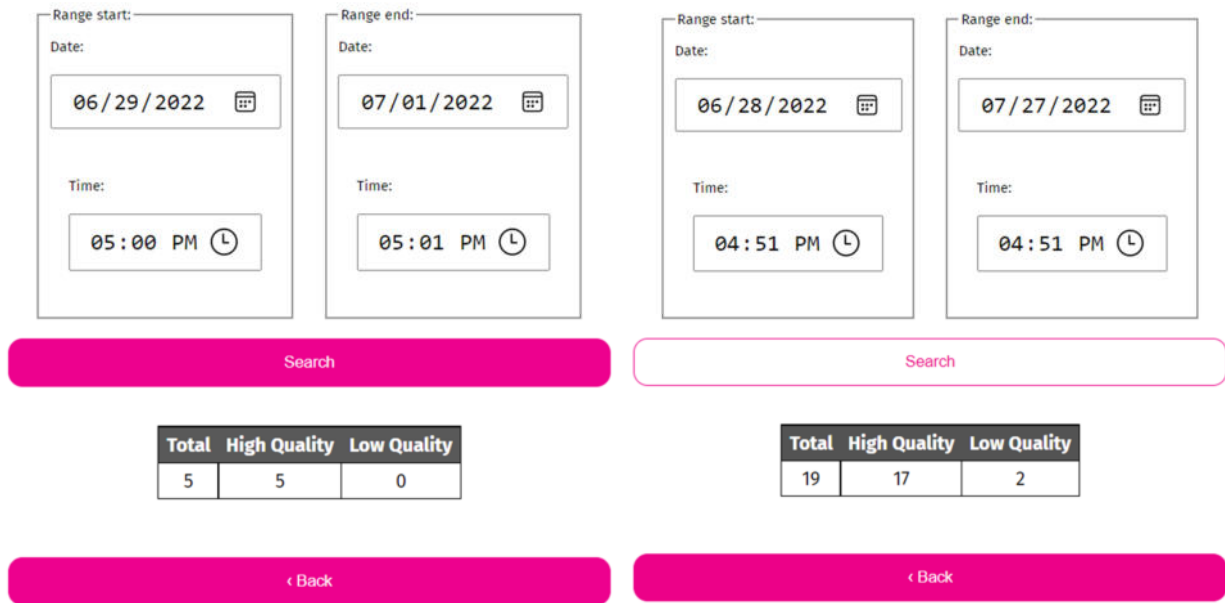


Figure 29: Different results for time range queries on quality degree of manufactured parts

5.1.3 Whisper protocol

The Whisper protocol allows users to encrypt and exchange messages in complete anonymity (i.e., sender and receiver of the message remain unknown to the rest of the network). This protocol is developed by Ethereum and resides in a private Ethereum blockchain created using Go Ethereum (Geth).

This protocol is used by Use Case 1 (UC1) to improve the security of WebRTC signalling in the remote rendering solution, ISAR. The private blockchain hosting the protocol can also be accessed and utilised by all users of the Pledger platform.

Figure 30 and Figure 31 represent two nodes from the DLT hosting nodes running the Whisper protocol. As shown, each node represents a unique HTTP endpoint that users can connect to and utilise by sending JSON-RPC requests. The peer count in the logs is the number of neighbours that each node has a direct connection to.

```

A geth --datadir=server --http --http.addr 192.168.100.00 --http.port 8225 --http.api personal,eth,admin,net,db,web3 --port 30304 --ipcdisable --bootnode= enode://bf42d120f7b2000000001f4376000000/bc8241f7287530f409200000
c044c8cb310f8f1a26402090e19c235af670b0091c656d2074c3a66e02127.0.0.1@8225port=30304"
INFO [07-11-11:36:59.593] Starting Geth on Ethereum mainnet...
INFO [07-11-11:36:59.598] Dumping default: cache on mainnet
INFO [07-11-11:36:59.605] Maximum peer count          proto=1024 update=4095
INFO [07-11-11:36:59.605] Maximum peer count          eth=50 list=total=50
INFO [07-11-11:36:59.605] Set global gas cap          cap=25000000
INFO [07-11-11:36:59.605] Allocated 1024 memory caches size=1024.00MiB
INFO [07-11-11:36:59.675] Allocated cache and file handles database=C:\Users\WouFendri\Desktop\PLEDGER\Pledger-Whisper-PoC\Nodes\server\geth\chaindata cache=2.00GiB handles=8192
INFO [07-11-11:36:59.711] Opened ancient database      config=C:\Users\WouFendri\Desktop\PLEDGER\Pledger-Whisper-PoC\Nodes\server\geth\chaindata\ancient
INFO [07-11-11:36:59.721] Initialised chain configuration
INFO [07-11-11:36:59.721] Initialised chain configuration
INFO [07-11-11:36:59.721] Engine: whkman"
INFO [07-11-11:36:59.722] Disk storage enabled for ethash caches dir=C:\Users\WouFendri\Desktop\PLEDGER\Pledger-Whisper-PoC\Nodes\server\geth\ethash count=3
INFO [07-11-11:36:59.746] Disk storage enabled for ethash DAGs dir=C:\Users\WouFendri\AppData\Local\ethash count=2
INFO [07-11-11:36:59.751] Initialising ethash protocol  version="[65 64 63]"
INFO [07-11-11:36:59.761] Loaded most recent local header number=0 hash="5e1fc7.4790eb" ts=131872 age=53y3mo2w
INFO [07-11-11:36:59.768] Loaded most recent local full block number=0 hash="5e1fc7.4790eb" ts=131872 age=53y3mo2w
INFO [07-11-11:36:59.778] Loaded most recent local fast block number=0 hash="5e1fc7.4790eb" ts=131872 age=53y3mo2w
INFO [07-11-11:36:59.782] Loaded local transaction journal transactions=0
INFO [07-11-11:36:59.789] Regenerated local transaction journal transactions=0 accounts=0
INFO [07-11-11:36:59.823] Allocated fast sync bloom  bloom=2.00GiB
INFO [07-11-11:36:59.825] Initialised fast sync bloom  bloom=2.00GiB
INFO [07-11-11:36:59.825] Starting peer-to-peer node  instance=Geth/v1.9.20-stable-979f968/windows-amd64/go1.15
INFO [07-11-11:36:59.869] New local node record  node=5e1fba035e0e20e2c2c0a12f0.0.0.1:30304 c=30304
INFO [07-11-11:36:59.878] Started P2P networking  url=enode://b4a1bc577b5c0f70b742ad4f80aaef6a0913c6d3379cfe50ff7367a40000f9759578a7da0e2527990b0ff470b0347a5f6bce6520fdba1848805a0127.0.0.1:30304
INFO [07-11-11:36:59.884] HTTP server started  endpoint=192.168.100.00:8225 cors= vhosts=localhost
INFO [07-11-11:36:59.884] started whisper v1.6.0  seq=21 id=fba035e0e20e2c2 ip=212.204.121.236 udp=30304 t=30304
INFO [07-11-11:37:02.583] Mapped network port  proto=tcp endpoint=30304 listen=30304 interface="\\.\NPF{...}"
INFO [07-11-11:37:18.282] Looking for peers  peerCount=0 tr=10-15 static=0
INFO [07-11-11:37:38.268] Looking for peers  peerCount=0 tr=10-15 static=0
INFO [07-11-11:37:38.378] Looking for peers  peerCount=0 tr=10-23 static=0
INFO [07-11-11:37:48.561] Looking for peers  peerCount=0 tr=10-23 static=0
INFO [07-11-11:37:51.108] Looking for peers  peerCount=0 tr=10-23 static=0

```

Figure 30: Server Node

Document name:	D4.5 Smart Contracts and DApps implementation tools II	Page:	37 of 42
Reference:	D4.5	Dissemination:	PU
		Version:	1.0
		Status:	Final

In the latest iteration of the Blockchain subsystem, the dedicated smart contracts and the hosted UC DApps are served by the deployed blockchain network. The listed SUCs describing the blockchain involvement in the core platform are able to be validated for the defined scenarios. In Figure 32 and Figure 33, the respective activities of SLASC Bridge and Wallet are depicted for reference purposes.

In Figure 32, the SLASC Bridge interaction with the StreamHandler is depicted. The respective SLA violations are received from SLASC Bridge and populated on the DLT. A complete functional overview of the component can be found on the corresponding demonstrator video in Pledger YouTube Channel (linked in section 5.3).

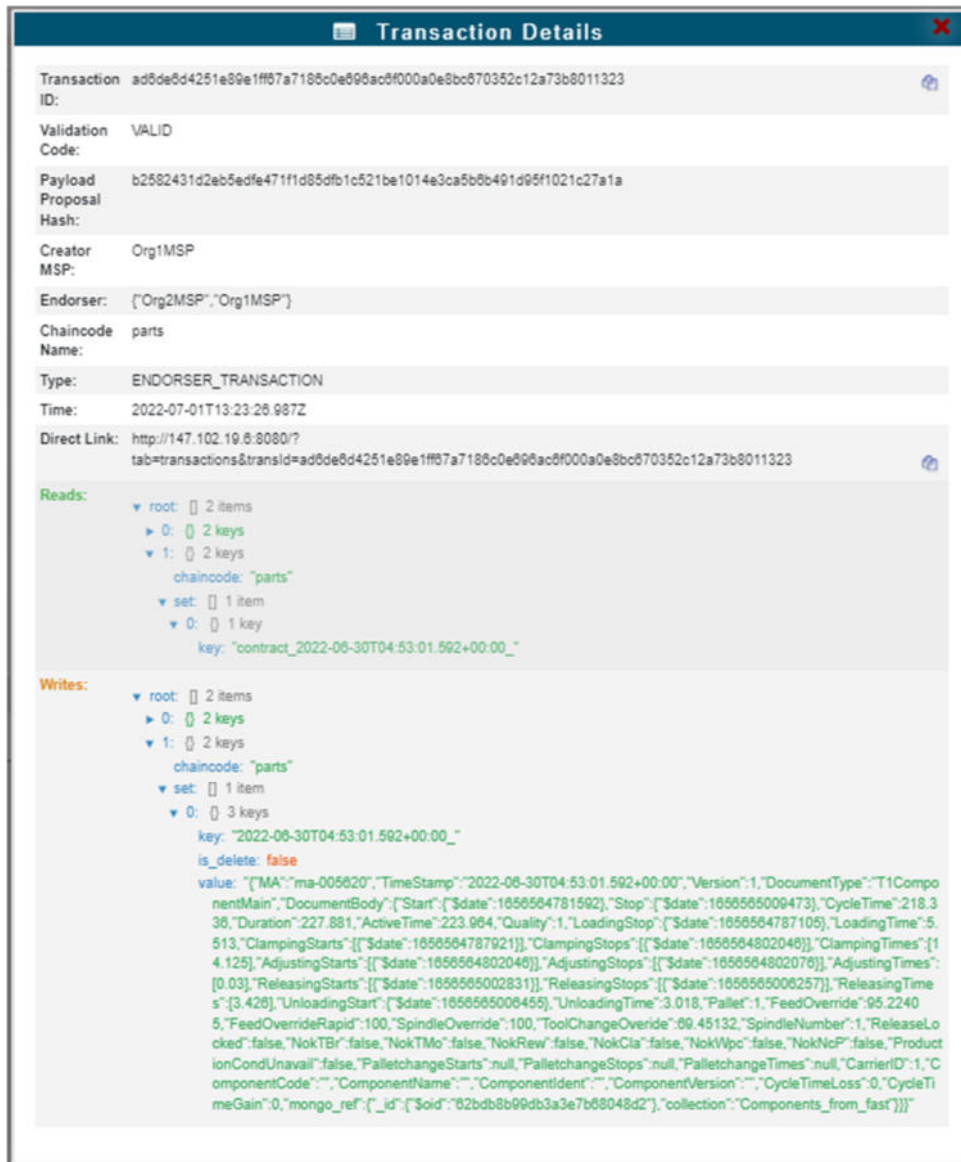


Figure 33: Wallet Demonstrator blockchain data

Similarly, in Figure 33, a data sample that is stored on the DLT and retrieved by the corresponding trusted user is depicted, while the respective video on Pledger YouTube Channel demonstrates the full functionality of the component. In the context of the components evaluation, further assessment parameters and metrics can be found under the work performed in task T5.4 deliverable D5.8.

Document name:	D4.5 Smart Contracts and DApps implementation tools II	Page:	39 of 42
Reference:	D4.5	Dissemination:	PU
	Version:	1.0	Status: Final

5.3 Demonstrator

The demonstratos for SLASC Bridge, Wallet and Whisper protocol can be found in the official YouTube Channel of Pledger:
<https://www.youtube.com/watch?v=v5eRSUmCNfk><https://www.youtube.com/channel/UCXV6V9rJ0ZvWhXeoWvDsArQ>.

Document name:	D4.5 Smart Contracts and DApps implementation tools II	Page:	40 of 42				
Reference:	D4.5	Dissemination:	PU	Version:	1.0	Status:	Final

6 Conclusions and next steps

This document accompanies the delivery of the last iteration of T4.2 "Smart Contracts and DApps on edge and cloud" which subscribes to the work performed during the period M19-M33. The relevant outputs offer secure and permissioned blockchain networks and services including their underlying functionalities and components. The trusted nodes of the permissioned network support all the necessary capabilities for transactions, smart contracts and DApps enabling dedicated privacy rules. The work delivered in this document includes an orchestrated blockchain platform specific to Pledger that serves the middleware architecture methods and the corresponding use cases. The Pledger DLT enhances the SLA on-chain applicability, and it constitutes an integrated environment that finalizes the SLA configurations and the respective use cases workflow. The entire delivered prototype addresses the requests and demands of Pledger core components and subsystems, that are involved in blockchain activities, and simultaneously is available to support similar cases that apply to the Pledger offering.

Document name:	D4.5 Smart Contracts and DApps implementation tools II	Page:	41 of 42				
Reference:	D4.5	Dissemination:	PU	Version:	1.0	Status:	Final

7 References

- [1] PLEDGER D4.2 Smart Contracts and DApps implementation tools I. Kapsoulis, Nikos. 2021. <http://www.Pledger-project.eu/content/deliverables>, retrieved on 26 August 2022.
- [2] PLEDGER D2.3 - Pledger Overall Architecture. Voutyras Orfefs. 2022. <http://www.Pledger-project.eu/content/deliverables>, retrieved on 26 August 2022.
- [3] DOCKER, Docker home. <https://www.docker.com>, retrieved 14/07/2021.
- [4] KUBERNETES, Kubernetes home. <https://kubernetes.io/>, retrieved 14/07/2021.
- [5] GOLANG, Golang home. <https://golang.org/>, retrieved 14/07/2021.
- [6] NODE.JS, Node.js home. <https://nodejs.org>, retrieved 14/07/2021.
- [7] HYPERLEDGER, Hyperledger home. <https://www.hyperledger.org/>, retrieved 14/07/2021.
- [8] YAML, YAML™ Specification Index. <https://yaml.org/spec/>, retrieved 14/07/2021.
- [9] PLEDGER D3.6 QoS and SLA assessment and negotiation tools II. Sucasas, Roi. 2022. <http://www.Pledger-project.eu/content/deliverables>.
- [10] JQ home, <https://stedolan.github.io/jq/>, retrieved 14/07/2021.
- [11] APACHE LICENSE, <https://www.apache.org/licenses/LICENSE-2.0>, retrieved 14/07/2021.

Document name:	D4.5 Smart Contracts and DApps implementation tools II	Page:	42 of 42				
Reference:	D4.5	Dissemination:	PU	Version:	1.0	Status:	Final