



Blockchain for the Next Generation Internet



D3.4 FRAMEWORK SPECIFICATION

05/11/2021



Grant Agreement No.: 957338
Call: H2020-ICT-2020-1

Topic: ICT-54-2020
Type of action: RIA

D3.4 FRAMEWORK SPECIFICATION

WORK PACKAGE	WP3
TASK	T3.3
DUE DATE	31/8/2021
SUBMISSION DATE	05/11/2021
DELIVERABLE LEAD	IEXEC
VERSION	1.0
AUTHORS	Anthony Simonet-Boulogne (IEXEC) Souvik Sengupta (IEXEC) Thanasis Papaioannou (AUEB) Alberto Ciaramella (IS) Petar Kochovski (UL)
REVIEWERS	Marco Ciaramella (IS) Klevis Shkempi (UL)
ABSTRACT	This deliverable provides updated specification of the ONTOCHAIN framework and architecture (compared to the initial architecture described in D3.3), of its components (including those developed by third parties during Open Call 1) and the specification of the ONTOCHAIN pilot deployment which will be used to evaluate the project results.
KEYWORDS	Decentralisation, blockchain, trustworthy content, data traceability, trustworthy knowledge exchange, privacy protection, web semantic, service interoperability

Document Revision History

Version	Date	Description of change	List of contributor(s)
0.1	16/6/2021	Table of content & initial draft	Anthony Simonet-Boulogne
0.2	18/6/2021	Executive summary	Anthony Simonet-Boulogne
0.3	24/6/2021	ONTOCHAIN ARCHITECTURE	Anthony Simonet-Boulogne
0.4	28/6/2021	ONTOCHAIN ARCHITECTURE Diagram modification	Petar Kochovski
0.5	4/8/2021	COMPONENTS DESCRIPTION based on D3 deliverables	Anthony Simonet-Boulogne
0.6	11/8/2021	ONTOCHAIN GATEWAY API & ONTOCHAIN PILOT DEPLOYMENT	Anthony Simonet-Boulogne
0.7	11/8/2021	Update of ONTOCHAIN PILOT DEPLOYMENT	Souvik Sengupta
0.8	11/8/2021	DISTRIBUTED LEDGERS section	Souvik Sengupta
0.9	20/10/2021	Update of COMPONENTS DESCRIPTION based on D4 deliverables	Souvik Sengupta
1.0	25/10/2021	ONTOCHAIN Gateway API	Anthony Simonet-Boulogne
1.1	04/11/2021	ONTOCHAIN Update the Gateway API, Architecture, PILOT DEPLOYMENT	Souvik Sengupta
1.2	04/11/2021	ONTOCHAIN Revision of the conclusion	Alberto Ciaramella

DISCLAIMER

The information, documentation and figures available in this deliverable are written by the "Trusted, traceable and transparent ontological knowledge on blockchain ONTOCHAIN " project's consortium under EC grant agreement 957338, and do not necessarily reflect the views of the European Commission. Neither the European Union institutions and bodies nor any person acting on their behalf may be held responsible for the use which may be made of the information contained therein. The information in this document is provided as is and no guarantee or warranty is given that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability. Moreover, it is clearly stated that the ONTOCHAIN Consortium reserves the right to update, amend or modify any part, section or detail of the document at any point in time without prior information.

The ONTOCHAIN project is funded by the European Union's Horizon 2020 Research and Innovation programme under grant agreement no. 957338.

COPYRIGHT NOTICE

© 2020 ONTOCHAIN

This document may contain material that is copyrighted of certain ONTOCHAIN beneficiaries and may not be reused or adapted without permission. All ONTOCHAIN Consortium partners have agreed to the full publication of this document. The commercial use of any information contained in this document may require a license from the proprietor of that information. Reproduction for non-commercial use is authorised provided the source is acknowledged.

The ONTOCHAIN Consortium is the following:

Participant number	Participant organisation name	Short name	Country
1	EUROPEAN DYNAMICS LUXEMBOURG SA	ED	LU
2	UNIVERZA V LJUBLJANI	UL	SI
3	IEXEC BLOCKCHAIN TECH	IEXEC	FR
4	INTELLISEMANTIC SRL	IS	IT
5	ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS – RESEARCH CENTER	AUEB	EL
6	ELLINOGERMANIKO EMPORIKO & VIOMICHANIKO EPIMELITIRIO	GHCCI	EL
7	F6S NETWORK LIMITED	F6S	IE

EXECUTIVE SUMMARY

This document is the deliverable "D3.4 Framework Specification" for of the ONTOCHAIN project funded under the Horizon 2020 Research & Innovation program "ONTOCHAIN–Trusted, traceable and transparent ontological knowledge on blockchain".

The framework specification was produced by Task 3.3, after the execution of ONTOCHAIN Open Call #1 "Research"; it consolidates the results of the seven projects funded and executed between March and September 2021 and integrates them within a single infrastructure design. Parts of this document will be provided to applicants as input to applicants of ONTOCHAIN Open Call #2 ("Protocol Suite & Software Ecosystem Foundations") and #3 ("Applications & Experimentation"). It may be revised periodically to include the progress made by the third parties funded under the Open Calls and should be used as a guide for implementing and deploying the complete ONTOCHAIN software ecosystem.

The framework specification defines the software components that will compose the ONTOCHAIN ecosystem, the application programming interfaces (APIs) that will allow these individual components to communicate, and the user-facing interfaces. that the ecosystem will provide (graphical user interfaces and APIs).

TABLE OF CONTENTS

EXECUTIVE SUMMARY.....	6
TABLE OF CONTENTS.....	7
LIST OF FIGURES.....	8
LIST OF TABLES.....	9
ABBREVIATIONS.....	10
1 INTRODUCTION.....	11
2 ONTOCHAIN ARCHITECTURE.....	12
2.1 ARCHITECTURE DESIGN.....	12
2.2 COMPONENTS DESCRIPTION.....	19
3 ONTOCHAIN GATEWAY API.....	32
3.1 General API description.....	32
3.2 Service discovery.....	32
3.3 Organization and user accounts.....	33
3.4 Storage service results/output.....	35
4 ONTOCHAIN PILOT DEPLOYMENT.....	38
4.1 Infrastructure for Pilot Deployment.....	38
4.2 Distributed Ledgers.....	43
4.3 Sustainability and Growth of the Pilot Deployment.....	45
5 CONCLUSION.....	46
REFERENCES.....	47

LIST OF FIGURES

FIGURE 1: ONTOCHAIN ARCHITECTURE AND COMPONENTS DIAGRAM..... 13

LIST OF TABLES

TABLE 1: HARDWARE REQUIREMENT FOR ONTOCHAIN PILOT DEPLOYMENT..... 42

TABLE 2: INFRASTRUCTURE PROCUREMENT FOR ONTOCHAIN PILOT DEPLOYMENT..... 44

ABBREVIATIONS

APIs	Application Programming Interfaces
DLT	Distributed Ledger Technology
ERC	Ethereum Request for Comments
NGI	Next Generation Internet
OC	Open Call for participation
P2P	Peer-to-Peer
SDK	Software Development Kit
PKI	Public Key Infrastructure

1 INTRODUCTION

This first version of the ONTOCHAIN framework specification is the combined result of the initial architecture designed before Open Call 1 (OC1), and of revisions made during the execution of the third party projects funded under OC1. This framework specification is thus the result of many discussions between the consortium, the funded third parties and the Advisory Board. Its main goal is to provide common guidelines and recommendations to applicants to OC2 regarding important design choices that go beyond their own project.

However, this document is due to evolve even more in the course of the ONTOCHAIN project; version 2 of the Framework Specification will be delivered at M24 in D3.5, including feedback and lessons learned from the execution of OC2.

The rest of this deliverable is organized as follows:

- In chapter 2, we provide the first version of the ONTOCHAIN architecture, as revised after the execution of OC1; the architecture defines the high-level services and components that will implement the ONTOCHAIN vision and describes the services that were implemented by third parties during OC1;
- In chapter 3, we define application programming interfaces for the ONTOCHAIN ecosystem, i.e. the APIs that will be presented to application developers and users of ONTOCHAIN services; these APIs also define primitives useful to developers of ONTOCHAIN (including participants funded through OC2), e.g. for integrating services and for implementing interoperability between services;
- Chapter 4 describes the planned pilot deployment of the ONTOCHAIN ecosystem which will be built on top of the iExec Ethereum sidechain and on top of geo-distributed hardware provided and hosted by the consortium members.
- Chapter 5 provides concluding remarks for this deliverable.

2 ONTOCHAIN ARCHITECTURE

The ONTOCHAIN software ecosystem consists of a novel protocol suite grouped into high-level application protocols, such as data provenance, reputation models, decentralised oracles, market mechanisms, ontology representation and management, privacy aware and secure data exchange, multi-source identity verification, value sharing and incentives and similar, and core protocols that include smart contracts, authorisation, certification, event gateways, identity management and identification, secure and privacy aware decentralised storage, data semantics and semantic linking.

2.1 ARCHITECTURE DESIGN

For enabling scalability, openness and high performance, we employ a modular approach. The original layered approach introduced before OC1 has been revised to take into account the progress made during the execution of first 18 funded projects. The revised architecture is presented in Figure 1; on the left hand side, the three modules (Applications, Ontologies and Distributed Ledgers) build on the functionalities offered by the lower layers. On the right hand side are cross-layer modules, which mainly focus on interfaces and interoperability. The colour scheme shows at which stage of the project each module will be developed, and it is apparent that after OC1, which focused on foundations for the ONTOCHAIN ecosystem, the focus for OC2 is building an infrastructure and integrating the results from OC1.

At the Solution Domain layer lie different next-generation application solutions, such as trustworthy web and social media, trustworthy crowdsensing, trustworthy service orchestration, decentralised online social networks, which tackle today's Internet problems that can be built upon the use cases Trustworthy Information Exchange and Trustworthy and Transactional Content Handling. Each of the use cases is built upon combined functionalities from the Application Protocols layer, such as Data Provenance, Reputation Models, Decentralised Oracles, etc. The modules at the Application Protocols layer themselves are built upon core Blockchain-based services at the Core Protocols layer, such as Smart Contracts, Identity Management, Secure and Privacy-Aware Decentralised Storage, Certification, Authorisation and Data Semantics. The Core Protocols modules employ basic Distributed-ledger functionality, i.e. Blockchain, Digital Currency and Distributed Storage, which lie on combined proprietary, corporate and public resources.

The *Distributed ledger* module will provide a distributed execution environment for the whole ecosystem; the *Ontologies* module will provide novel, trusted services for managing web ontologies in a fully trusted and secure fashion; the *Applications* module will be the focus of OC3, although OC1 already produced valuable pilot use-cases.

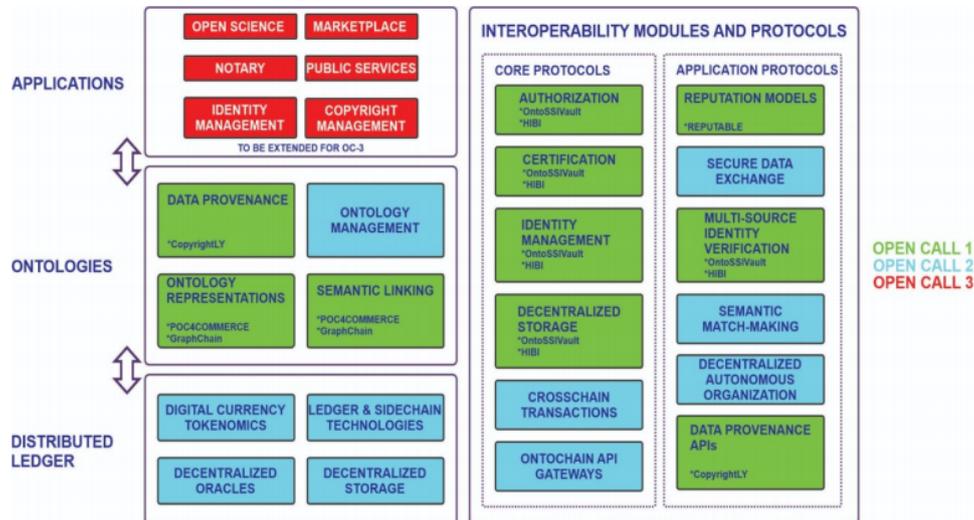


FIGURE 1: ONTOCHAIN ARCHITECTURE AND COMPONENTS DIAGRAM AS REVISED AFTER OPEN CALL 1.

The *Interoperability modules and protocols* has multiple functions; first, it will act as the backbone for interconnecting the other modules; second, it will provide the ONTOCHAIN Gateway API, i.e. the main entry point for application developers; last, it will provide essential services and building blocks to all of the modules and to the applications, e.g. decentralised storage, identity management and data certification.

2.1.1 Distributed Ledgers

The ONTOCHAIN software ecosystem will rely on several interconnected Blockchains in order to ensure performance and scalability metrics that satisfy a business context. The first benefit of this multi-chain environment is to better scale applications by hosting the transactions and data from different business domains in different chains. An additional benefit is to isolate applications that may need it, in order to preserve the confidentiality of transactions when necessary (e.g. in the context of private industrial consortium). ONTOCHAIN will provide native interoperability solutions for exchanging information and assets between the different chain (see 2.1.4).

The main ONTOCHAIN Blockchain will be based on Ethereum¹, which is currently the industry standard for Blockchain-based decentralised applications². Ethereum also

¹<https://ethereum.org/fr/>

²<https://entethalliance.org/>

features several initiatives to support cross chains operations (e.g. Polkadot³, Polygon⁴) that ONTOCHAIN can monitor and eventually integrate for this purpose. The initial pilot deployment will rely on the iExec sidechain, which is based on the POA Network⁵ stack (see Section 4).

In addition to Blockchains based on Ethereum and on the Ethereum Virtual Machine (EVM-based Blockchains), ONTOCHAIN will necessarily features that are not compatible with the EVM; At this time, two services are in this situation, GraphChain which is entirely compatible with Ethereum smart contracts but relies on a modified client (see 2.2.1) and Gimly ID, which is built on top of the Oasis Protocol⁶ (see 2.2.4).

- *Digital currency tokenomics* Beyond the current hype revolving around Bitcoin, Ethereum and over 5,000 altcoins, the potential for social change of what is now being called "Blockchain 2.0" is appearing more and more clearly. For example, cryptocurrencies are praised for allowing cheap and fast money transfer to the 1.7 billion people who are excluded from the banking system around the world, or as a stable alternative to devalued fiat currencies. One very interesting aspect for the Next Generation Internet is the possibility of programming complex self-executing transactions in Smart Contracts. Integrated with ONTOCHAIN's provenance and reputation mechanisms, a crypto token will guarantee a fair compensation to every contributor who participates in the ecosystem.
- *Ledger & sidechain technologies* Blockchain offers a plethora of features, such as traceability, transparency, anonymity, democracy, automation, decentralization and security. Despite these promising features the technical scalability of the network is still a key barrier which can put a strain on the adoption process, especially for real business environments. The complexity and the specialization of all the different real-world ONTOCHAIN applications will lead practitioners to use multiple ledger technologies for implementing different solutions. This will enable higher performance and scalability, while enabling different business logics, access methods and governance models that require specific chains. This component will build the main ONTOCHAIN and associated clients based on Ethereum, which is a stable and well tested system for data transactions and has a cost-effective network for the operation of its applications.
- *Decentralized oracles* By essence, smart contracts must thus run entirely isolated and cannot access data from the external world on their own. Oracles are software components which primary purpose is to collect external data and input it to smart contracts. Because oracles bring arbitrary data to the Blockchain they create a major vulnerability, and the trustworthiness of the imported data is extremely difficult for

³<https://polkadot.network/>

⁴<https://polygon.technology/>

⁵<https://www.poa.network/>

⁶<https://oasisprotocol.org/>

them to assess. This component will provide trustful and trustless oracle prototypes that are capable of interacting with the ONTOCHAIN infrastructure and providing necessary data for the operation of its applications.

- *Decentralized storage* Various decentralised repositories, such as Peer-to-Peer and Content Distribution Networks have existed for decades. Lately, the ENTICE⁷ project developed a series of repositories for the storage of Virtual Machine and container images, including their fragments and it was optimised for delivery upon request. The interfaces used are S3 like⁸. In recent years, with the emergence of Blockchain, we have witnessed a new wave of participatory storage repositories that can help address the security and privacy needs, and may help store practically any kind of data, for example, service such as STORJ⁹ and Filecoin¹⁰. This component will integrate new storage services that will help store private data in encrypted and decentralised ways, manage data replicas for reliability and Quality of Service, while balancing the trade-offs with the storage costs.

2.1.2 Ontologies

Specific domain ontologies have been developed or reused by ONTOCHAIN projects.

Of these, the project PROF4COMMERCE was specifically focused to develop ontologies, in this case for supporting e-commerce applications. More specifically, this project developed 3 domain ontologies, as extensions of well know ontologies, and more specifically: - OC-Found, building on the ontology OASIS, for supporting the semantic description of the stakeholders of the ONTOCHAIN ecosystem, including supply chain of resources, digital identities of agents, quality valuation processes - OC-Commerce, building on the ontology GoodRelations, for supporting the semantic description of offering, auctions and commercial activities. - OC-Ethereum, building on the ontology BLONDIE, for supporting the semantic description of Ethereum Blockchain, smart contracts and tokens.

The project CopyrighTLY reused the Copyright Ontology, which is a background of the group delivering this project. This ontology, available on <https://rhizomik.net/ontologies/copyrightonto>. Models the fundamental building blocks of the copyright domain, including rights, creations, and actions, are also available there.

The project KnowledgeX developed an ontology for matching data science problems with skills of data scientist.

⁷<http://www.entice-project.eu/>

⁸<https://aws.amazon.com/fr/s3/>

⁹<https://www.storj.io/>

¹⁰<https://filecoin.io/>

The project GraphChain developed We have developed the Graph metadata ontology, a lightweight ontology that can be used to represent and interchange provenance information and other metadata generated inside the Graphchain ecosystem. The ontology is mainly intended for describing data in the default graph.

2.1.3 Applications

Applications will be the focus of ONTOCHAIN Open Call 3, which is due to open in the third year of the project. At this time ONTOCHAIN features one complete application, KnowledgeX which proposes a marketplace where individuals can find and hire data scientists and outsource data analytics jobs to a variety of computing infrastructures for satisfying different confidentiality requirements. More information about KnowledgeX can be found in Section 2.2.6.

2.1.4 Interoperability modules and protocols

Core protocols

- *Authorizations* Blockchain has stimulated the idea of self-sovereign digital identity, and few commercial services have already emerged¹¹. Various Role-Based Access Control (RBAC) systems have also existed in the course of the past decades. With ONTOCHAIN one could easily see systems where a patient is self-identified on the Blockchain, while a medical doctor gains access to the medical records based on their role (e.g. surgeon, general practitioner).
- *Certification* Certification refers to the confirmation of certain characteristics of an object, person, or organization. For example, a government may decide to offer certificates to cloud providers that have verified GDPR-compliant handling of private citizens' data. In such cases, certificates can be issued on-chain, and can be used as conditions for performing specific transactions, for example, using AI methods to analyse private data. The specific conditions can be implemented within a Smart Contract to govern the GDPR-handling of private citizens' data only on certified cloud providers.
- *Identity management* Self-sovereign Digital Identity is a specific area of research that is overreaching to be addressed solely by the present project. Nevertheless, ONTOCHAIN technologies and solutions can be used to address parts of the digital identity puzzle. There are two conflicting requirements that drive this development. First is the ability to identify oneself in specific interactions, such as withdrawing money in a bank, and another is to still preserve one's privacy, for example of

¹¹<https://www.ibm.com/blogs/Blockchain/category/trusted-identity/self-sovereign-identity/>

the health data or the web browsing or buyer's habits. This is a feasible endeavour. However, it is necessary to invest more in technologies like ONTOCHAIN to make it happen.

- *Decentralized storage* Various decentralised repositories, such as Peer-to-Peer and Content Distribution Networks have existed for decades. In recent years, with the emergence of Blockchain, we have witnessed a new wave of participatory storage repositories that can help address the security and privacy needs, and may help store practically any kind of data in virtualized block devices. In the near future, one could imagine new storage services that can help store private data in encrypted and decentralised ways, that can help manage data replicas for reliability and Quality of Service, while balancing the trade-offs with the storage costs.
- *Crosschain transactions* A Crosschain transaction module facilitates the interoperability between two relatively independent Blockchains. In other words, this module will enable the functionalities for the Blockchains to speak to one another Blockchain. Crosschain implementation is mainly represented by asset swap and asset transfer, which is both an essential part of the Blockchain world. With the help of this module, the limitations of a single chain can be avoided.
- *ONTOCHAIN API Gateway* This module will support connections between the ONTOCHAIN Blockchain and the outside world, including other Blockchains. Part of its duty will be to help programmers in the upper layers make trade-offs about how much data is stored on-chain, by supporting pointers to offchain decentralized storage, such as IPFS. The module will provide several low-level Application Programming Interfaces (APIs) in the form of Smart Contracts, as well as several higher-level wrappers for at least three programming languages that are commonly used by developers e.g. JavaScript, Java and Python. The interfaces will be generic and extensible in order to allow connections with different ledger technologies in the future, while only external Ethereum-based chains will be supported during the course of the project because of its important community of adopters and developers and because its ecosystem already contains most of the software components that sub-projects will require (e.g. off-chain Computing, Decentralized Oracles). However, sub-grantees will be discouraged from producing designs relying on concepts and optimisations that are specific to any particular Blockchain.

Application protocols

- *Reputation models* This module will provide the functionality of building different decentralized reputation models over the Blockchain infrastructure. The basic building blocks of a reputation system are an approach for casting assessments/votes for a particular subject (person/data/fact), an approach for recording the history of votes per subject and an approach for summarizing votes into a single reputation metric per subject. One important problem with reputation systems is weak identities, referred to as "cheap pseudonyms", through which multiple attacks can be employed,

such as ballot stuffing, bad naming, negative discrimination, sybil attacks, reputation whitewashing, reputation milking and more. Existing solutions include reputation cold start (which introduces a new set of problems) and making pseudonym change more costly. Emerging decentralized reputation mechanisms built upon the Blockchain will enable stronger user identities without sacrificing anonymity. Different reputation models can be defined to assess different aspects, such as data source trustworthiness, data credibility, service trustworthiness, etc. This module is built upon Identity Verification mechanisms.

- *Secure Data Exchange* Secure data exchange comprises the functionality of exchanging data among distributed parties, while verifying the ownership of the data and access rights, authenticity of transacted parties, the integrity of the data exchanged and the confidentiality of the data through Blockchain underlying mechanisms. Most often, off-chain data will be exchanged in data transactions, while on-chain data will store public cryptographic keys and access control lists based on which elevated data access to different portions of data is authorized for specific transacted parties.
- *Multi-source Identity Verification* This module seeks to register and verify individual digital identities of physical objects (in addition, abstract objects, physical and abstract persons) via trusted data from multiple sources. Attributes represent a fundamental part of any ontological concept. In particular, various AI methods can be introduced to operate on sensing data (IoT based, sensors, cameras and similar) so that an assertion can be made whether an individual belongs to a specific ontological concept (e.g. car, chair, container image, the person Elisabeth, and similar).
- *Semantic Match-Making* This module will find correlations between any goods or services that can be described with an ontological representation. Semantically related entities of ontologies can be used for different tasks such as ontology merging, query answering, data translation, etc. Ontology matching is a requirement for finding compatible offer and demand (buy and sell orders) in semantic-based marketplaces. To guarantee the fairness of the transactions in the marketplace, the matching process should be fair to every party, e.g. preventing exclusion, censorship, price manipulation and fraud.
- *Decentralized Autonomous Organization* Based on the encoding rules, the decentralized autonomous organization (DAO) module can run a Blockchain protocol entirely and autonomously with the help of smart contracts. Thus, this module circumvents the need for human intervention or centralized coordination and helps to build a trustless system.
- *Data Provenance APIs* This module will provide application programming interfaces for querying and presenting provenance information from ONTOCHAIN about on-Chain and off-Chain data (pointers to data stored outside of ONTOCHAIN). Provenance information will include the complete trail of transactions that resulted in a

record, including links to the programs that were run (e.g. address of smart contracts, signature of AI models when available), to the input data that was processed and to the contributors who ran the programs or provided original information.

2.2 COMPONENTS DESCRIPTION

The system architecture described in the previous section will be realised by integrating software components developed by successful applicants to the ONTOCHAIN open calls for contribution. The following seven projects have already been selected and developed by third parties under ONTOCHAIN OC1 and will be integrated into the final ONTOCHAIN ecosystem; this section briefly introduces each of these components and describes the interfaces they will provide for integrating with the other ONTOCHAIN components.

In particular, these interfaces should be read as specifications for Topic 3 "Interoperability & API Gateways" of ONTOCHAIN OC2.

2.2.1 GraphChain: a framework for on-chain data management for ONTOCHAIN

Description GraphChain is a framework for on-chain data management for ONTOCHAIN which implements decentralised On-chain graph management technologies, including the ability to perform usual graph operations. GraphChain proposes a radically different approach: instead of encapsulating the semantic data into Blockchain blocks, they propose to design and implement the Blockchain mechanisms on top of semantic data. The GraphChain solution provides different functionalities such as:

- Hashing of subgraphs for the on-chain graph structures;
- Procedural smart contracts with access to the on-chain semantic data;
- Identification, authorisation and data provenance for the on-chain data;
- Sharding mechanisms and strategies.

The whole idea of GraphChain is adding a new level of trust without sacrificing availability, query ability and performance of graph databases so the solution can be integrated in any software ecosystem that uses traditional LPG databases.

Application Programming Interfaces In the GraphChain ecosystem, the core element is an Ontonode, which is a single OntoSidechain node. The general idea of Ontonode

operation is similar to every Blockchain network. The Blockchain nodes process transactions and achieve consensus over data that represent them. Ontonode is based on the combination of Ethereum and RDF-star triplestores. The modification strategy for the Ethereum client node has been thoroughly presented in [1].

Module Ontopod is one of the most important sub-elements of Ontonode. It is an RDF-star compliant graph database that stores all named graphs protected and distributed by Blockchain network. The services offer by the Ontopod are exposed by the REST APIs. Below the detailed descriptions of each REST API calls has been presented:

POST `/ontonode/deploy?contract=type` Deploy the contracts which are essential for proper work of Ontonode. Takes the type of contract as an argument. If the operation is successful then it returns the value "OK" and HTTP 200. Otherwise it will returns "error" and HTTP code 400.

GET `/ontonode/info?size=graph_count` Get metadata about the graphs which are uploaded to Graphchain. This call takes the number of graph as the argument and returns the JSON with graph metadata.

GET `/nodes/get?size=count` Get the information about Ontonodes. Takes the number of desired Ontonodes as the argument and returns HTTP code 200 in case of success. Otherwise, it returns HTTP 400 for erroneous call or HTTP 404 if the information of the Ontonode is not found.

POST `/nodes/adding?nodeId=id` Add a new Ontonode. The function takes the node ID as the argument and returns HTTP code 200 if successful. Otherwise, it will return HTTP code 400 for erroneous call or HTTP 404 if the information of the Ontonode is not found.

POST `/nodes/update?nodeId=id` Modify the Ontonode. Similarly to the previous function, it takes a node ID as the argument and returns HTTP code 200 for success. Otherwise, it returns HTTP 400 for erroneous call or HTTP 404 if the information of the Ontonode is not found.

Module Ontoshell is another important sub-element of Ontonode. It is a set of endpoints and interfaces. It is a crucial component because all Blockchain interactions, which are not internal, work on this layer. The most standard way of interaction with Ontonode is through REST API (Graph update, retrieval, storing and deleting).

PUT `/rdf-graph-store?graph=graph_uri&user=user_addr`

POST `/rdf-graph-store?graph=graph_uri&user=user_addr` Both calls upload RDF graph from the request body to GraphChain. These calls take the URI of a graph and the

Ethereum address of a user as arguments. Successful execution of these calls return "OK" and HTTP code 200. Otherwise, it returns HTTP code 400 for erroneous call or HTTP 415 for wrong serialization.

GET `/rdf-graph-store?graph=graph_uri` Return the latest version of a RDF graph. The function takes the URI of the target graph as the argument. For unsuccessful search operation, the function returns HTTP error code 404.

DELETE `/rdf-graph-store?graph=graph_uri` The delete operation does not remove anything, instead it adds an empty graph to the Triplestore and points to it as the latest version. The argument is the URI of the graph to remove and returns HTTP code 200 for successful call. Otherwise, it returns HTTP code 400 for erroneous call or HTTP 404 if the information of the graph is not found.

HEAD `/rdf-graph-store?graph=graph_uri` Same as GET but returns an empty response body. Similar to ASK SPARQL query. Takes the URI of the retrieved graph as the argument and returns HTTP code 200 for successful call. Otherwise, returns HTTP 400 for erroneous call or HTTP 404 if the information of the graph is not found.

2.2.2 CopyrightLY: Decentralised Copyright Management for Social Media

Description CopyrightLY is an application that helps managing ownership and rights in the ONTOCHAIN ecosystem. From claiming authorship of content or data, to linking these claims to evidence off-chain or associating reuse terms that once agreed set the reuse conditions among the involved parties.

CopyrightLY proposes a system capable of rooting on-chain copyright transactions, especially NFTs, on copyright claims that can be tied to evidence and validated on court. These set of evidence, together with the opportunity to make complaints and use incentives to curate them, makes it possible to build a scalable and community driven content ownership layer.

The central part of the CopyrightLY's architecture is on-chain and based on a set of smart contracts, which take care of registering Manifestations (i.e. authorship claims), Complaints (to denounce authorship claims potentially fraudulent) and Reuses (used to attach reuse terms to a manifestation, including the initial offer, the negotiation steps and the final reuse agreement, if reached).

Application Programming Interfaces To facilitate interoperability and dealing with a complex domain as copyright management, CopyrightLY is based on the use of semantic technologies and the formal conceptualisations provided by the Copyright Ontology.

CopyrightLY's functionality is made available through different mechanisms. If the aim

is to read the state regarding authorship claims (manifestations), evidence accumulated or existing complaints, it is available mainly through a GraphQL API provided using The Graph:

- Manifestation (authorship claim) details;
- Manifestations (authorship claims) list;
- Upload Evidence details;
- Upload Evidence list.

On the other hand, it is also possible to access and modify CopyrightLY's state as stored in the Blockchain, interacting directly with the corresponding smart contracts. Notably, the services will be exposed by the REST API calls. Detailed of those calls are presented below:

POST `/api.studio.thegraph.com/query/1303/copyrightly/0.0.9` Is sending a GraphQL query to the endpoint. Taking GraphQL query as the argument and returning the result in JSON.

More details can be found in [2].

2.2.3 HIBI: Human Identity Blockchain Initiative

Description HIBI encompasses scalable Blockchain, decentralised legal reputation and identity systems and interoperable semantic web technologies. HIBI is provided to developers as a modular SDK for adding specific features to an application. All of the features are based on the eIDAS standard for qualified electronic signatures and require the NFC scans of a legal EU identification document. With HIBI, developers can:

- Support eID-based authentication in their applications;
- Let users sign transactions and documents using their physical eID card;
- Link the legal identity of a user to a Blockchain-based public key;
- Perform key backup and recovery using a legal ID document.

HIBI provides the user with the power and sovereignty of their keys, identifiers, and verifiable credentials. Therefore, it contradicts existing identity management systems that are usually based on centralised data silos managed by identity providers.

Application Programming Interfaces The SDK allows the client application to communicate with the HIBI services over remote API calls. For instance, the APIs let a user request their secrets to be backed up, or the DID on which some credential should get issued. The two SDK modules are *EVERKEY* and *EVERID*; the methods they provide to client applications for consuming HIBI services are listed hereafter.

Module *EVERKEY* Decentralized, Non-custodial Backup and Recovery Mechanism that is using eIDAS-compliant eIDs that can provide a pseudonym.

store(key, userIdentifier, keyIdentifier) Get a secret backed up using the HIBI SDK. There is no need to expose further APIs, as the rest of the system is provided by the eID infrastructure where the HIBI SDK itself receives data autonomously and the decentralized storage solution to post secrets. These endpoints are kept hidden to prevent abuse.

recover(userIdentifier, keyIdentifier) Get a secret that came from the decentralized storage grid and was reassembled within the HIBI SDK.

retrieveIdentifier() Perform the eID Authentication. The user needs to scan their ID with an NFC-enabled device. After entering the correct PIN, the eID is unlocked and can be used for *EVERKEY* and *EVERID*.

Module *EVERID* With this REST API, it is possible to call the EverID service. The service allows for deriving a Verifiable Credential and get it issued, as well as get the Qualified Electronic Signature (QES) from the issuer, and verify the credential.

POST **/everkey.id/v1/getVC** Return a Verifiable Credential that includes the provided identity attributes. The argument is the name of the user and returning verifiable Credential in JSON-LD form.

POST **/everkey.id/v1/verifyVC** Verify the Verifiable Credential. Takes the verifiable credential as argument and returns a boolean.

GET **/everkey.id/v1/cert** Obtain the X.509 Issuer certificate to verify the VC of the user.

GET **/everkey.id/v1/getTcTokenURL** Obtain the endpoint to perform the eID authentication.

More details can be found in [3].

2.2.4 Gimly ID: an SSI application suite

Description Gimly ID is a fully self-sovereign identity solution that brings trust and usability to users without compromising security and privacy of the ecosystem and its members. Gimly ID centers around the mobile application, which offers a passwordless single-sign on experience and selective disclosure of data by leveraging decentralised identifiers (DIDs) and Verifiable Credentials (VCs) and a sovereign data vault. The Gimly ID mobile app can be used as a standalone solution, aiming for full interoperability with other SSI conformant solutions.

Application Programming Interfaces Gimly ID also includes a range of web portals and developer tools. These allow for advanced data management, simplified deployment of the SSO and SSI functionalities in existing systems, the creation and management of DIDs and VCs, and the governance and analytics of deployed ecosystems.

The complete software suite consists of the following React native mobile application and of several web applications:

1. Gimly ID mobile App: authentication and SSI data vault. Used for password-less login, management of credentials and sovereign data, managing data access permissions.
2. SSO login button: used to enable SSO with the Gimly ID app into web applications and web portals.
3. Sovereign MyData web portal: advanced sovereign data management.
4. Developers web portal: allows developers, organisations, companies to implement the password-less SSO flow with the Gimly ID app.
5. Organisations identity admin portal: managing DIDs and VCs, including the creation, issuing and revocation of credentials.
6. Ecosystem governance portal: allows for defining policies and rules for identities, verifications.

More precisely, three new modules have been introduced in addition with the previous modules. In this section we are going to describe the API calls for those three new modules.

Module DID Auth SIOP SIOP v2 is an extension of OpenID Connect to allow End-users to act as OpenID Providers (OPs) themselves. Using Self-Issued OPs, End-users can authenticate themselves and present claims directly to the Relying Parties (RPs), typically a webapp, without relying on a third-party Identity Provider. This

makes the solution fully self sovereign, as it does not rely on any third parties and strictly happens peer 2 peer, but still uses the OpenID Connect protocol.

createURI Create a signed URL encoded URI with a signed SIOP authentication request. Taking `SIOP.AuthenticationRequestOpts` as argument and returning promise `<SIOP.AuthenticationRequestURI>`.

verifyJWT Verify a SIOP Authentication Request JWT. Throws an error if the verification fails. Returns the verified JWT and metadata if the verification succeeds.

createJWTFromRequestJWT Creates an Authentication Response object from the OP side, using the Authentication Request of the RP and its verification as input together with settings from the OP. The Authentication Response contains the ID token as well as optional Verifiable Presentations conforming to the Submission Requirements sent by the RP.

verifyJWT Verifies the OPs Authentication Response JWT on the RP side as received from the OP/client. Throws an error if the token is invalid, otherwise returns the Verified JWT.

resolver.resolve Resolves the DID to a DID document using the DID method provided in `didUrl` and using DIFs `did-resolver` and `Sphereons Universal registrar and resolver client`.

getResolver The `DidResolution` file exposes 2 functions that help with the resolution as well.

resolveDidDocument The `DidResolution` file exposes 2 functions that help with the resolution as well.

createDidJWT Creates a signed JWT given a DID which becomes the issuer, a signer function, and a payload over which the signature is created.

verifyDidJWT Verifies the given JWT. If the JWT is valid, the promise returns an object including the JWT, the payload of the JWT, and the DID Document of the issuer of the JWT, using the resolver mentioned earlier. The checks performed include general JWT decoding, DID resolution and Proof purposes. Proof purposes allows restriction of verification methods to the ones specifically listed, otherwise the 'authentication' verification method of the resolved DID document will be used.

Module Gimly ID Mobile SDK For the most up-to-date information, it is recommended to read the documentation: <https://github.com/Gimly-Blockchain/gimly-id-app-sdk>.

createEncryptedWallet This API creates a wallet. Taking `pin` as the argument and returning a string value.

createDid It creates a DID and taking `identityData` and `Authority` as arguments and

returning IdentityData.

`getUserDid` This call returning the value of IdentityData and resolves a DID.

`updateDID` It updates the DID document by taking arguments as IdentityData, permission info, parental info of the corresponding DID and Authority related info.

`createCredential` Create a credential.

`verifyCredential` Verify the credential.

`createPresentation` Create a presentation.

`signPresentation` Sign a presentation.

`verifyPresentation` Verify a presentation.

`verifyUnsignedPresentation` Verify an unsigned presentation.

Module *Universal DID resolver/registrar client* This is another module related with the SDK. For more details it is recommended to read the online documentation: <https://github.com/Sphereon-Opensource/did-uni-client>.

`Registrar.create` Create a DID and obtain its JSON representation as a string.

`Registrar.update` Update a DID and obtain its JSON representation as a string.

`Registrar.deactivate` Deactivate a DID and obtain its JSON representation as a string.

`Resolver.resolve` Resolve a DID document.

Notably, for this project no services have been exposed through REST API calls as it is aimed at Peer-to-Peer communication. More details are provided in [4].

2.2.5 Reputable: a Provenance-aware Decentralised Reputation System for Blockchain-based Ecosystems

Description Reputable delivers a cross-platform privacy-aware reputation system which leverages Blockchain technology to achieve decentralised, verifiable calculation of reputation scores. Further it enables interaction with end users and systems through a secure, reputation analytics dashboard to facilitate user verification as seamless integration with other systems and services.

Within Reputable, the reputation data consists of two different types. Firstly, it is the individual user feedback i.e. the feedback provided by the users when contacted to share their experiences with a service/seller/marketplace. Secondly, it is the aggregate repu-

tation score which is calculated using the individual user feedback. As these two types of data are linked with each other, the linkage is preserved and utilised it to achieve verifiable reputation scores.

Application Programming Interfaces Reputable provides three different interfaces:

- Query reputation score for a seller
- Query historical reputation score for a seller
- Verify reputation calculation for a user

As the reputation modelling system is expected to interact with external services and users in an off-chain arrangement, Reputable envisions leveraging decentralised oracles to integrate reputation data with the on-chain provenance mechanism. Notably, the developer did not build any APIs for the SDK module. They have exposed their services through REST APIs. An example of these services has presented below. For more details, it is recommended to follow [5].

Module Aggregator This module is responsible to calculate aggregate reputation score using the individual user feedback. The aggregator is a crucial component of the REPUTABLE system.

GET `/localhost:3000/reputation` Aggregate the reputation score for a seller. Takes the identifier of the seller as the argument and returns the seller's reputation score.

POST `/localhost:3000/reputation_score` Post a seller's aggregate score using seller id and user scores. Taking sellerID and individual reputation scores, as arguments.

GET `/localhost:3000/verify_reputation` Verify the reputation of a service/seller. Returns the receipt highlighting on-chain record/hash of reputation feedback.

GET `/localhost:3000/individual_score` Get the individual score of a buyer/consumer.

GET `/localhost:3000/individual_scores` Get the individual scores relating to a buyer's aggregated score and returning the individual scores of buyers who rated the specified seller.

GET `/localhost:3000/token_used` Query the aggregator to find out if a token has previously been used. Takes the token as an argument and returns a Boolean value.

2.2.6 KnowledgeX: Trusted data-driven knowledge extraction

Description KnowledgeX is a trustworthy marketplace for data science. This means data owners can outsource data science tasks to independent contractors without risk-

ing data misuse. Independent data scientists can bid on proposed tasks without getting prior access to confidential data.

Currently data markets are hampered by confidentiality requirements due to competitive (e.g., cost data) or regulatory (e.g., personal data) considerations. Data scientists have to be employed in-house or are contractually restricted by non-disclosure stipulations, which tend to be ambiguous and costly to enforce.

KnowledgeX aims to solve this problem via a process where data privacy, and contract fulfillment are technologically guaranteed, so the need for non-disclosure agreements does not arise. Therefore, KnowledgeX leverages the following technologies:

Emerging technologies:

- Blockchain and Smart Contracts
- Decentralised Cloud Computing
- Trusted Execution Environments

Established technologies:

- Web Applications
- Public Key Infrastructure
- Microservice Backend

Application Programming Interfaces In ONTOCHAIN OC2 and OC3, different higher-level applications will be built. These applications can leverage KnowledgeX' REST API for identity management, microservices and interface to on-chain interactions. The corresponding details of the REST APIs for each individual services are presented in [6].

Module 1 (USER DETAILS SERVICE) Rest API of the user details service is used by frontend user interface to store personal information of the user. The API allows to modify information such as address, billing data or user description. This service stores users IDs shared between the services and allow to detect user Type to render the corresponding views.

Module 2 (EXECUTION SERVICE) Execution service rest API is used by frontend user interface to trigger and manage execution/exploration jobs, datasets, programs as well as blockchain orders or execution results. The Blockchain Middleware use this API to persist the execution data gathered from iExec.

Module 3 (GIG SERVICE) Gig Service API is used by User service to persist new users and by frontend application to manage gigs and offers.

Module 4 (BLOCKCHAIN MIDDLEWARE) For this service, no external integration is allowed. Some examples of API calls for exposing this services are as follows:

GET `/<base_uri>/blockchain-middlewre/iexec/deal/id/results` Taking id of execution job as argument and returning execution results.

POST `/<base_uri>/blockchain-middlewre/iexec/executionJob` Taking the whole JSON containing the task execution details and stores execution job details in iExec and starts execution.

POST `/<base_uri>/blockchain-middlewre/contracts` This REST call stores contract in Blockchain.

2.2.7 POC4COMMERCE: Making ONTOCHAIN practical for eCom-merce

Description POC4COMMERCE innovates the ontological representation of Blockchain-oriented digital commerce by integrating and extending the most representative ontologies for modelling, participants, in particular commercial actors, offers, products, and tokens emitted on the Ethereum Blockchain as digital representation of exchanged assets: providing a semantic descriptions of smart contracts and related transactions, in particular of smart contracts related with tokens trading and associated with commercial means.

POC4COMMERCE reuses and extends the most suitable and relevant ontologies in the domain, namely, OASIS for the representation of commercial participants and smart contracts, GoodRelations for representing commercial offers, and BLONDIE for describing Ethereum essential elements: all these ontologies are conjoined and extended to also cover the gap missing from the literature on the representation of digital tokens, smart contracts, digital identities, valuation mechanisms, and auctions.

Application Programming Interfaces POC4COMMERCE delivers a set of three modular ontologies describing each semantic compartment of eCommerce, from participants, assets, and offerings, to supply chains, smart contracts, and digital tokens. The ontological stack is released together with the SPARQL queries implementing the defined competency questions that enable users and developers to meaningfully probe the knowledge bases they contribute to construct. Also, it delivers an API library consisting of two main modules: *OC-Generator*, in short OCGEN, providing basic APIs for generating agent ontological representations in the mentalistic notion of agent behaviors fashion, adopted by the foundational ontology OC-Found. The second one is

OC-Commerce Search Engine, in short OCCSE, providing APIs that realize the core of a semantic search engine and reasoning system for querying the ONTOCHAIN knowledge bases.

The modules OCGEN and OCCSE have been implemented in Python, version 3.7. Specifically, OCGEN adopts the RDFLib¹² version 5 API library¹³ to generate RDF fragments, whereas OCCSE adopts the API library OWLReady 2¹⁴ version 0.34 to realize the query engine. Whereas RDFLib provides low-level APIs for generating RDF triples in an extremely efficient way, OWLReady 2 provides an higher level interface for integrating OWL reasoners and performing SPARQL queries compliant with the latter version 1.1.

Module OCGEN The OCGEN main object can be instantiated first by creating three RDFLib ontology objects, one for the ontology hosting the agent behaviors, one for the ontology hosting the agent templates, one for the ontology hosting data, then by calling the constructor of the class OCGEN. The module will store RDF graphs in the correct ontology depending on the type of operation required. It is also possible to create or load one single ontology for storing both behaviors, templates, and data. The OCGEN module provides several methods to deal with semantic web agents in the OASIS ontology fashion:

`createAgentTemplate()` is a void method creating an agent template given agent-TemplateName as input parameter, namely a string representing the agent template name.

`createAgentBehaviorTemplate()` is a void method creating a behavior template to be associated with an agent template.

`connectAgentTemplateToBehavior()` is a void method associating a behavior template to an agent template.

`createAgent()` is a void method creating a real agent where MyAgent is a string representing the agent name.

`createAgentBehavior()` is a void method creating a concrete behavior to be associated with a concrete agent.

`connectAgentToBehavior()` is a void method associating a concrete behavior to a concrete agent.

`createAgentAction()` is a void method creating an agent action that is associated with an agent behavior.

¹²<https://pypi.org/project/rdflib/5.0.0/>

¹³RDFLib version 6 has currently a bug preventing the OCGEN prototype from loading remote ontologies.

¹⁴<https://pypi.org/project/Owlready2/>

Module OCCSE Before instantiating the OCCSE, a repository manager and a reasoner interface are required. To create a repository manager it is sufficient to create an object of type RepositoryManager, passing a list of IRIs representing the repositories that will be loaded into the OCCS triple store.

`RepositoryManager()` returns the object RepositoryManager, where [repository1, repository2, ...] is a list of IRIs representing the repository to load.

`ReasonerInterface()` returns the ReasonerInterface, where reasonerName is either pellet or hermit values.

`OCCSE()` instantiates the OCCSE module, where repositoryManager and reasonerInterface are the repository manager and the reasoner interface, respectively, as created before.

The details of other methods are describe in [7]. Notably, for this project no REST services are included in current implementation.

3 ONTOCHAIN GATEWAY API

This section describes all the general APIs for the ONTOCHAIN framework and their functionalities.

3.1 GENERAL API DESCRIPTION

The ONTOCHAIN gateway API is composed of three modules. These three modules are - 1. Service Catalog, 2. Accounts Management, & 3. Data Storage. In below, all the details about the modules are described -

3.2 SERVICE DISCOVERY

In the second year of the project, ONTOCHAIN will already integrate the seven projects that were funded through OC1 (see Section 2.2), and its ecosystem will steadily grow to host between 25 and 30 services by the end of 2023. The exploitation of ONTOCHAIN's results will bring even more services in the future, driving the need for machine-to-machine discovery mechanisms.

In this section we introduce the ONTOCHAIN Services catalog, which will service as a single entry point to accessing ONTOCHAIN services. The catalog will provide search mechanisms and return pointers to the service implementations directly to the clients, in the form of API endpoints when available. The search features will support advanced ontology-based queries so that consumer applications can search for services based on functionalities, implement their own selection algorithm and start consuming the services without requiring specific human interactions.

Module *Services catalog* This module will provide a programmable way of discovering the services provided through the ONTOCHAIN ecosystem. Access to the functions will be authenticated and regular users will only have access to read operations. Write operations will be accessible only to administrators, unless noted in the function's description.

GET `<base_uri>/list/count/page?filter=JSON` Obtain a list of all the available services as a JSON string. Arguments `count` (max. 100) and `page` are used to control the number of results. An optional argument can be provided to filter the types of desired services; the argument is a url-encoded JSON document that can contain every field of the resource description schema. Returns a JSON document with the de-

sired result count (or less) and HTTP code 200 in case of success, HTTP 400 if the provided filter cannot be decoded, and code 401 if the user is not authenticated.

GET `<base_uri>/service/search?query=QUERY` Obtain a list of services which name or description match the provided search terms. The QUERY argument can contain one or several url-encoded terms. Returns a JSON document with matching project descriptions and HTTP code 200 in case of success, HTTP 400 if QUERY is absent, and code 401 if the user is not authenticated.

GET `<base_uri>/service/SERVICE_ID` Obtain the detailed description of a service as a JSON string. The argument is the service identifier that can be obtained with the /list function. Returns a JSON document and HTTP code 200 in case of success, HTTP 404 if the requested service cannot be found, and code 401 if the user is not authenticated.

POST `<base_uri>/service/` Add a new service to the catalog. The request body must contain a JSON document containing the complete service description. Returns HTTP code 200 in case of success, HTTP 400 in case the description has an incorrect format, and HTTP 401 if the user is not authenticated or does not have the required privileges.

PUT `<base_uri>/service/SERVICE_ID` Update the details of a service already stored in the catalog. The function takes the identifier of the service that must be updated as an argument and the request body must contain a JSON document containing the new service description. Returns HTTP code 200 in case of success, 400 in case the description has an incorrect format, 404 if the requested service does not exist, and code 401 if the user is not authenticated or does not have the required privileges.

DELETE `<base_uri>/service/SERVICE_ID` Removes the service pointed by argument SERVICE_ID from the catalog. The service is not actually removed, it will only be marked as deleted and it will no longer be part of the results of the /list and /search functions.

3.3 ORGANIZATION AND USER ACCOUNTS

Users in the ONTOCHAIN ecosystem can be either consumers or providers of services, or both. Users can be attached to an organization or be registered independently. The goal if this membership model is both to drive engagement, by providing ONTOCHAIN with specific features (e.g. a user portal) and to manage accounting, i.e. payment for services and crypto-wallets.

Module Accounts management This module provides interfaces for creating users and organizations, and basic interactions with wallets. All API calls must be authenticated and write operations are accessible only to administrators, unless

specifically noted.

POST `<base_uri>/organization/` Create a new organization with no users. The request body must contain the details of the organization as a JSON document (e.g. its name, short description, contact address). The call returns the identifier of the new organization and HTTP code 200 in case of success. The call can also return HTTP code 400 if the organization description is not a valid JSON document or if required fields are missing, and code 401 if the user is not authenticated.

PUT `<base_uri>/organization/ORGANIZATION_ID` Updates the information related to an existing organization. The identifier of the organization must be provided in the query string, and the request body must contain the new details of the organization as a JSON document. The call returns HTTP code 200 in case of success, code 400 if the organization description is not a valid JSON document or if required fields are missing, 404 if the organization does not exist, and code 401 if the user is not authenticated.

GET `<base_uri>/organization/ORGANIZATION_ID` Obtain the details related to an existing organization. The call takes the identifier of the organization as an argument and returns a JSON document containing the organization's information. The call returns HTTP error code 200 in case of success, 404 if the organization does not exist, and 401 if the user is not authenticated.

GET `<base_uri>/organization/ORGANIZATION_ID/list` Obtain the list of users that are members of an organization. The identifier of the organization is taken as the sole argument to the call. The call returns JSON array of objects describing the users. In case of success, the call return HTTP code 200; the call can also return code 404 if the organization cannot be found, and 401 if the user is not authenticated.

DELETE `<base_uri>/organization/ORGANIZATION_ID` Request the deletion of an organization. The organization will be marked as deleted but no data will actually be removed, in order to preserve the history of the platform. If the organization contains users, the users will not be removed but will appear as independent. The only argument to the call is the identifier of the organization. In case of success, the call returns HTTP code 200. The call can also return code 404 if the organization does not exist and code 401 if the user is not authenticated or does not have the required privileges.

POST `<base_uri>/user/` Create a new user. The request body must contain the details of the user as a JSON document (e.g. its name, organization, email address, phone number). The call returns the identifier of the new user and HTTP code 200 in case of success. The call can also return HTTP code 400 if the user description is not a valid JSON document or if required fields are missing.

PUT `<base_uri>/user/USER_ID` Updates the information related to an existing user. The identifier of the user must be provided in the query string, and the request body must contain the new details of the user as a JSON document. The call returns HTTP

code 200 in case of success, code 400 if the user description is not a valid JSON document or if required fields are missing, 404 if the user does not exist, and code 401 if the requesting user is not authenticated.

GET `<base_uri>/user/USER_ID` Obtain the details related to an existing user. The call takes the identifier of the user as an argument and returns a JSON document containing the user's information. The call returns HTTP error code 200 in case of success, 404 if the user does not exist, and 401 if the requesting user is not authenticated.

DELETE `<base_uri>/user/USER_ID` Request the deletion of a user. The user will be marked as deleted and all of associated information (except its identifier, in order to preserve the history of the platform) will be removed. The only argument to the call is the identifier of the user. In case of success, the call returns HTTP code 200. The call can also return code 404 if the user does not exist and code 401 if the requesting user is not authenticated or does not have the required privileges.

GET `<base_uri>/user/USERNAME/wallet` Obtain the details (chain, address, balance) of the user's wallets in the ONTOCHAIN ecosystem. The call takes the identifier of the user as an argument and returns the wallets information as a JSON document along with HTTP code 200 in case of success. The call can also return code 404 if the user does not exist or has been deleted, and code 401 if the requesting user is not authenticated or does not have the required privileges.

3.4 STORAGE SERVICE RESULTS/OUTPUT

Data storage is a special kind of service in the way they provide a communication medium between other services. In order to facilitate these interactions, the Data storage module provides high-level interfaces for storing and retrieving files and arbitrary blobs of data to and from providers, including external storage providers such as Amazon S3¹⁵, Dropbox¹⁶ and IPFS¹⁷.

Module Data storage The module provides users and applications with high-level interfaces that are common to all of the supported backend storage services, both members of the ONTOCHAIN ecosystem, and external services. Storage providers can be searched in the Service catalog like any other service (see Section 3.2).

POST `<base_uri>/collection/` Create a new collection of objects, similar to a directory in a file system. The call takes a description of the collection (including the

¹⁵<https://aws.amazon.com/s3/>

¹⁶<https://www.dropbox.com/>

¹⁷<https://ipfs.io/>

storage provider) in the JSON format as an argument. The call only creates a record for the collection, it does not store any data object. In case of success, the call returns the identifier of the new collection and HTTP code 200. In case of failure, the call returns code 400 (if the JSON document cannot be interpreted) and 401 if the user is not authenticated or does not have the required privileges.

PUT `<base_uri>/collection/COLLECTION_ID` Updates the information related to an existing collection of objects. The identifier of the collection must be provided in the query string, and the request body must contain the new details of the collection as a JSON document. The call returns HTTP code 200 in case of success, code 400 if the collection description is not a valid JSON document or if required fields are missing, 404 if the collection of object does not exist, and code 401 for unauthorized access.

GET `<base_uri>/collection/COLLECTION_ID` Obtain the details related to an existing collection of objects. The call takes the identifier of the collection of object as an argument and returns a JSON document containing the information of corresponding collection of objects. The call returns HTTP error code 200 in case of success, 404 if the collection of object does not exist, and 401 for the unauthenticated access request.

GET `<base_uri>/collection/list` Obtain the list of collection of objects. The call returns JSON array of objects describing the collection of objects. In case of success, the call return HTTP code 200; the call can also return code 404 if the collection of objects cannot be found, and 401 for unauthenticated access request.

DELETE `<base_uri>/collection/COLLECTION_ID` Request the deletion of a collection of object. The collection of object will be marked as deleted but no data will actually be removed, in order to preserve the history of the platform. If the collection of object contains users, the users' information will not be removed but will appear as independent. The only argument to the call is the identifier of the collection. In case of success, the call returns HTTP code 200. The call can also return code 404 if the collection ID does not exist and code 401 if the requester (user) is not authenticated or does not have the required privileges.

POST `<base_uri>/object/` Create a new object. The call takes a description of the collection (including the storage provider) in the JSON format as an argument. The call only creates a record for the object. In case of success, the call returns the identifier of the new object and HTTP code 200. In case of failure, the call returns code 400 (if the JSON document cannot be interpreted) and 401 if the user is not authenticated or does not have the required privileges.

PUT `<base_uri>/object/OBJECT_ID` Updates the information related to an existing objects. The identifier of the object must be provided in the query string, and the request body must contain the new details of the object as a JSON document. The call returns HTTP code 200 in case of success, code 400 if the object description is not a valid JSON document or if required fields are missing, 404 if the object does not exist, and code 401 for unauthorized access.

GET `<base_uri>/object/OBJECT_ID` Obtain the details related to an existing objects. The call takes the identifier of the object as an argument and returns a JSON document containing the information of corresponding object. The call returns HTTP error code 200 in case of success, 404 if the object does not exist, and 401 for the unauthenticated access request.

DELETE `<base_uri>/object/OBJECT_ID` Request the deletion of a object. The object will be market as deleted but no data will actually be removed, in order to preserve the history of the platform. If the object contains users, the users' information will not be removed but will appear as independent. The only argument to the call is the identifier of the object. In case of success, the call returns HTTP code 200. The call can also return code 404 if the object ID does not exist and code 401 if the requester (user) is not authenticated or does not have the required privileges.

GET `<base_uri>/object/OBJECT_ID/read` Obtain the details related to an existing objects. The call takes the identifier of the object as an argument and returns information of corresponding object. The call returns HTTP error code 200 in case of success, 404 if the object does not exist, and 401 for the unauthenticated access request.

PUT `<base_uri>/object/OBJECT_ID/write` Updates the information related to an existing objects. The identifier of the object must be provided in the query string, and the request body must contain the new details of the object as a document. The call returns HTTP code 200 in case or success, code 400 if the object description is not a valid or if required fields are missing, 404 if the collection of object does not exist, and code 401 for unauthorized access.

POST `base_uri>/object/OBJECT_ID/USER_ID` Checks the access control for a particular user to access a particular object. If the user has the access grant for the accessing the object it returns HTTP code 200. Returns code 400 if the object description is not a valid or if required fields are missing, 404 if the object does not exist, and code 401 for unauthorized access

4 ONTOCHAIN PILOT DEPLOYMENT

Today, blockchain-based systems face many challenges related to scalability, privacy, security, and various other aspects. To solve these issues, many researchers have proposed different solutions. Among all these existing solutions, the sidechain concept [8] solves many extant blockchain issues and opens many new doors for using the blockchain in different application domains. Therefore, considering the advantages of using the sidechain concept in our project, we have adopted this concept for developing the pilot use-cases. Mainly, sidechains are secondary blockchains; they are connected with the other blockchain using the two-way peg methodology. Notably, the two-peg mechanism enables the functionalities for bidirectional transfer of assets between the mainchain and sidechain at a fixed or pre-deterministic exchange rate. Remaining of this section, focusing on the two-peg mechanism, we are going to explain the overall deployment strategy for the pilot use-cases thoroughly.

The choice of relying on a sidechain over layer-2 solutions such as rollups is due to the lack of maturity of layer-2 chains at this time. However, the decision of relying on the EVM for the main ONTOCHAIN chain will ensure that upcoming updates to the Ethereum protocol and implementation (Ethereum 2.0) which are due to be deployed in 2022 and 2023 will benefit past, current and future ONTOCHAIN services and applications. Indeed, since almost all ONTOCHAIN services sit at the application layer, they will remain compatible with Ethereum updates (to the exception of GraphChain which relies on a modified Besu client which may require a small update in order to support Ethereum 2.0).

4.1 INFRASTRUCTURE FOR PILOT DEPLOYMENT

Currently, three different strategies exist for implementing a two-way peg for transferring the assets from mainchain to sidechain and vice-versa. Each of them has several advantages and disadvantages. For example, the Centralised two-way peg strategy is easy to implement over the systems. Also, it ensures the fast transfer of assets between two chains. However, this strategy introduces a degree of political centralisation that one has to trust a single entity to transfer assets between chains. Thus to remove this single remove single-entity trust, the federated or multi-signature two-way peg has been proposed. In that case, instead of one single entity, multiple or a group of entities control the assets transfer process. Though the specialized federation protocol ensures the fast transfer of assets between blockchains, this strategy does not entirely eliminate the political centralisation of authority. Thus, the overall system remains fragile and malicious attacks over the major entities could break the overall system and steal the assets. Therefore, Simplified Payment Verification (SPV)-based two-way peg strategy has emerged to address this issue, which eliminates the necessity of third-party

entities for transferring the assets. However, in this strategy, the user needs to wait for confirmation and reorg; thus, the overall assets transfer process remains slow between the chains.

Following all the pros and cons of each individual strategy, we have adopted the federated two-way peg methodology for developing the sidechain-based deployment in our project. However, in our case, we ensure to lock the assets on one chain and enforce to multiple entities (e.g., bridge agent) for signing to authorize a legal transaction by indicating that the assets remain lock on mainnet to give them on the sidechain and vice-versa. Currently, we are following the centralized management strategy for managing these multiple entities. In the rest of this section, focusing on the various blockchain services and nodes, we will describe the overall deployment strategy for the pilot use-cases.

4.1.1 Nodes and services for pilot deployment

There are several strategies that different projects have followed in order to build the blockchain-based ecosystem. Each project uses different consensus mechanisms. For example, Proof-of-Work (PoW) and Proof-of-Stake (PoS) are the most famous consensus mechanisms, which have been used by Bitcoin and Ethereum, respectively. Besides these two mechanisms, several consensus mechanisms exist; for example, Proof-of-Contribution (PoCo) has been used by the iExec project. However, for most of the projects, the deployment strategies are similar. For each case, there are several computing nodes are involved in developing the blockchain-based ecosystem. Based on the operational functionalities, the definition of nodes varies from application to application. In a Blockchain network, nodes are the electronic devices connected to the network and possessing an IP address. Generally, nodes are the communication endpoints, which means that any user or application that wants to interact with the Blockchain-based system does so through nodes. Therefore, nodes are also a point of communication redistribution. Notably, each node offering different services. Focusing on their functionalities, we will highlight the advantages and disadvantages of hosting them in this section.

Full Node The search results come from the nearest server that maintains all that information in a random search from any search engine. In a similar sense, full nodes maintain all the transaction records in the blockchain and are considered the servers of the blockchain. It may happen in certain situations where many full nodes agree to certain modifications in the blockchain, but they still do not make a majority. In that case, the old blockchain will work as it did, and the new blockchain will work with the proposed changes by the proposing full nodes. Full node, can be further classified into Pruned Full Node and Archive Node.

- Pruned Full Node – This node has a defined memory limit to hold the data. So, in essence, there is no limit to how many blocks can be added to the blockchain, but there is a limit to how a full node can store many blocks. Notably, a pruned full node downloads the blocks to maintain the ledger.
- Archive Node – An archive node is a full node, which is configured in such a way that it can keep every state from the blockchain. They are used to feed extensive data for highly requiring applications like some web interfaces to explore the chain.

Light Node A light node stores and provides just the necessary data to accommodate daily activities or faster transactions. They are not involved in validating the blocks and just stores the Block headers. These are also called Simplified Payment Verification nodes (SPV nodes).

Most importantly, for developing sidechain-based pilot use-cases, we have separated several functionalities of full nodes and light nodes among the various participating nodes. Thus it brings higher stability and reduces the workload for each participating node. In the rest of this section, focusing on the iExec platform, we will describe each node’s functionalities and various services working together to develop the sidechain-based platform.

Validator is a computing node that participates in the blockchain consensus mechanism. They have the critical duty of authoring new blocks on the chain by executing and approving the transactions submitted by users and blockchain clients. In the iExec sidechain, validator nodes are working on Proof-of-Authority (PoA) consensus mechanisms. Validators maintain high transparency by putting their identity at stake. Notably, validators risk their reputations to host a decentralized node. Thus, this amount of transparency creates personal and group accountability as they know their actions and participation are written to the public ledger and available for any blockchain client. Importantly, to achieve higher scalability and protect the main blockchain from extreme circumstances, the sidechain-based deployment has been envisioned within the scope of the Ontochain project. Notably, in the case of sidechain-based deployment scenarios, validators play a crucial role. In the iExec platform, it uses five validators for production.

Authority Round (AuRa) is one of the blockchain consensus algorithms available in OpenEthereum. In this consensus, a list of identities is allowed to validate new blocks. On the other hand, aura provides a slot-based block authoring mechanism. In that case, the list of validators can be specified during the consensus process configuration, and a consensus smart contract can handle it. In our case, all consensus smart contracts used by iExec are forked from the PoA-network project in this repository.

Bootnode is a node that is used by the other nodes to bootstrap their peer discovery. For the iExec platform, four Bootnodes for the production chain.

Access node is a blockchain node that also acts as an entry point for clients to read the chain's state and send transactions for execution. The Access node can be collocated with a Bootnode, it can be a full node or a light node.

Bridge & Bridge agent The bridge is composed of several bridge agents and of smart contracts deployed on the sidechain and on the mainchain. The bridge agents observe the bridge smart contracts on the sidechain and on the main chain to detect when a transfer of assets is being requested. In the case of the two-way peg mechanism, the bridge agents execute the smart contract on the sending chain, which *locks* ERC-20 tokens, and then executes the bridge contract on the destination chain, which *mints* an identical amount of tokens on the receiving end. For the iExec platform, five bridge agents along with one bridge UI agent are hosted in six medium size computing nodes.

Blockscout it is the web interface, which is used for exploring a chain. It provides a comprehensive, easy-to-use interface for users to view, confirm, and inspect transactions on all EVM (Ethereum Virtual Machine) blockchains. This includes the Ethereum main and test networks as well as Ethereum forks and sidechains. In iExec platform, it uses one computing node for the production environment.

Mainnet Node is an independent blockchain node running its own network with its own technology and protocol. In the iExec platform one large computing node is responsible for hosting the Mainnet.

4.1.2 Hardware and Network requirements for pilot deployment

The minimum hardware requirements for each of the blockchain's services are given in Table 1. Following this requirement table one can understand how much hardware resources could required for developing the sidechain-based pilot use-cases. These services can be individually deployed over the instances of the public cloud providers (e.g., Amazon EC2 or Microsoft Azure) or the local resources. Interestingly for developing the pilot UCs, some of the components can be grouped and deploy within a single large cloud instance or even in the small next unit computing resources (e.g., Intel NuC). For example, Bootnode, Validator and Access node can be deployed either over a t3.2xlarge Amazon EC2 instance or even one Intel NUC Kit with 11th Generation Intel Core i9-11900KB Processor (24M Cache, up to 4.90 GHz). Likewise, Bridge agent and Bridge UI services can also be deployed over one t3.xlarge Amazon EC2 instance or even Intel NUC Kit with 11th Generation Intel Core i5-1135G7 Processor (8M Cache, up to 4.20 GHz). As Blockscout enable the web interface facilities for the users to explore and inspect all transactions, thus it is recommended to host this service over a single t3.large EC2 instance or one Intel NUC Kit with 11th Generation Intel Core i3-1115G4 Processor (6M Cache, up to 4.10 GHz). Notably, in iExec, the Mainnet node is hosted over a t3.xlarge EC2 instance, which can also be hosted over an Intel NUC Kit with 11th

Component	No. CPU	RAM (GB)	Disk (GB)	Public IP	Open ports	AWS instance type
Bootnode	2	4	200	Yes	22, 80, 443, 30303	t3.medium
Validator	2	4	100	Yes	22, 30303	t3.medium
Access node	4	8	1000	Yes	22, 80, 443, 30303	t3.xlarge
Bridge agent	2	4	100	No	22	t3.medium
Blockscout	2	8	500	Yes	22, 80, 443	t3.large
Bridge UI	2	4	100	Yes	22, 80, 443	t3.medium
Mainnet node	4	8	600	Yes	22, 80, 443	t3.xlarge

TABLE 1: HARDWARE REQUIREMENT FOR ONTOCHAIN PILOT DEPLOYMENT.

Generation Intel Core i5-1135G7 Processor (8M Cache, up to 4.20 GHz).

Following tight security policies is one of the critical requirements for deploying the sidechain-based pilot use cases. The robust disk encryption techniques and firewall mechanisms are essential to building strong security policies. All the working nodes must have all their disks encrypted. In addition that, all the working nodes should follow a strong network firewall policy. *Block all incoming traffic and allow all outgoing traffic* – this is the default network firewall policy that all the working nodes must maintain. Notably, for each node, some particular ports need to be open to allow inbound and outbound traffic. The detailed information of these ports has been given to the Table 1.

4.1.3 Provided resources for pilot deployment

In this subsection, we will describe the detailed information about the resources provided by the different partners for developing the sidechain-based pilot use cases. Notably, each partner will be in charge of deploying different blockchain nodes. There-

fore, below, we will present thorough details about which partners will deploy which blockchain nodes. Significantly, Table 2 represents the summary of the infrastructure procurement for ONTOCHAIN pilot deployment.

Resources provided by iExec iExec will be in charge of hosting Bootnode, Validator, Access node, Bridge agent, Blockscout, Bridge UI, and Mainnet in the Amazon EC2 public cloud. The VM instances will be provisioned and distributed in different available zones of the Amazon EC2 public cloud. For example, Bootnode will be hosted in the eu-north-1 zone. Whereas the Validator will be hosted in the eu-west-3 zone. Similarly, the Access node will be hosted in the eu-west-2 zone, and the Bridge agent will be hosted in the us-east-1 zone. Meanwhile, the Blockscout, Bridge UI and Mainnet nodes will be hosted in the ca-central-1 and eu-north-1 zones, respectively.

Resources provided by IntelliSemantic Validator, Bridge agent and Access node will be hosted in the public cloud. IntelliSemantic planned to host Validator and Bridge agent in one single instance, whereas the Access node will be hosted in another instance.

Resources provided by Athens University of Economics and Business Athens University of Economics and Business will host Validator and Bridge agent in co-located fashion on one on-premises server, and also they will dedicate another server for hosting one Access node. The details of those physical machines have been presented in table 2.

Resources provided by University of Ljubljana University of Ljubljana will in-charge for hosting three different nodes: Mainnet node, Bridge UI, and Bridge agent. All of these three nodes will be hosted in public cloud. The VM instances will be provisioned and distributed in different available zones of the public cloud.

4.2 DISTRIBUTED LEDGERS

The main chain will be the iExec Bellecour sidechain. To develop that iExec is using the stable version of OpenEthereum¹⁸, which ensures to provide the core infrastructure essential for speedy and reliable services. Importantly, OpenEthereum has some software dependencies. It requires the stable version of Rust¹⁹, Perl²⁰, and Yasm²¹. In the GraphChain and Gimly ID, the developers are using the Ethereum²² to develop their sidechains for providing some specific services.

¹⁸<https://github.com/openethereum/openethereum>

¹⁹<https://www.rust-lang.org/>

²⁰<https://www.perl.org/>

²¹<https://yasm.tortall.net/>

²²<https://ethereum.org/en/>

Partner	Component	No. CPU	RAM (GB)	Disk (GB)	Location
IEXEC	Bootnode	2	4	200	Public cloud
	Validator	2	4	100	Public cloud
	Access node	4	8	1000	Public cloud
	Bridge agent	2	4	100	Public cloud
	Blockscout	2	8	500	Public cloud
	Bridge UI	2	4	100	Public cloud
	Mainnet node	4	8	600	Public cloud
IS	Validator, Bridge agent	4	8	200	Public cloud
	Access node	4	8	1000	Public cloud
AUERB	Validator, Bridge agent	4	16	1,000	On-Premise
	Access node	4	16	1000	On-Premise
UL	Mainnet node	4	8	600	On-Premise
	Bridge UI	2	4	100	Public cloud
	Bridge agent	2	4	100	Public cloud

TABLE 2: INFRASTRUCTURE PROCUREMENT FOR ONTOCHAIN PILOT DEPLOYMENT.

4.3 SUSTAINABILITY AND GROWTH OF THE PILOT DEPLOYMENT

As described above, the initial blockchain network for ONTOCHAIN will be created on a voluntary basis from resources contributed by the core partners of ONTOCHAIN. Apart from the capital expenses associated with the acquisition of hardware resources contributed to the ONTOCHAIN blockchain infrastructure, there are non-negligible operational expenses involved in the decision-making process, for electricity consumption, network access, hosting, support and maintenance, etc. This blockchain network should be sufficient for component experimentation from third parties and for hosting the initial number of dapps for ONTOCHAIN. After all, employing the Proof of Authority consensus model, the size of the network can be expanded up to 10 nodes. For larger networks, another consensus model, e.g., Proof of Stake, should be employed.

We plan to expand our blockchain network based on node contributions by the third-party entities that have been (or will be) selected through ONTOCHAIN open calls. In fact, such an approach is reasonable, because these third-party entities are de-facto part of the ONTOCHAIN ecosystem. Moreover, based on the results of the early economic sustainability analysis of the ONTOCHAIN ecosystem, described in D5.2, being part of the ONTOCHAIN ecosystem is highly beneficial for all involved stakeholders and especially for third-party contributors of ONTOCHAIN functionality through open calls. Therefore, there is some economic incentive associated with contributing to the establishment of the ONTOCHAIN blockchain network. At the same time, we assumed in our early sustainability analysis of D5.2 that a specific portion of the dapp transaction fee is shared (e.g., according to their stakes) among the nodes of the blockchain network to cover their operational expenses and provide them incentives for being part of the blockchain network.

After the blockchain network has been kickstarted, according to the approach described above, then ONTOCHAIN clients could be distributed to the general public to contribute their resources and get compensated based on transaction fees (and according to their stakes), provided that such an expansion serves the decentralisation/performance trade-off of the platform.

Finally, currently, the incentives provided to the nodes of the blockchain network are assumed to be based on FIAT currencies. The employment of utility or other crypto tokens in the ONTOCHAIN ecosystem might provide further incentives for blockchain network expansion. This issue is left for future work within WP5.

5 CONCLUSION

This document consolidates the outcomes of the seven funded projects, which have been developed between March and September 2021. This document will be provided as the guideline material for the applicants of Open Call #2 and Open Call #3.

In this deliverable, we have thoroughly described the revised layered approach of the ONTOCHAIN framework. Mainly the structure of three previously envisioned modules (Applications, Ontologies, and Distributed Ledgers) and their functionalities have been modified and tuned to build within the ONTOCHAIN framework. More precisely, the development of the seven funded Open Call #1 projects added new functionalities of the ONTOCHAIN framework and helped for developing different components within the ONTOCHAIN framework. Moreover, documenting all the related information of the different components and their interfaces in this deliverable will help the Open Call #2 participants further extend those functionalities and add new features to the ONTOCHAIN framework.

The functions described in this deliverable can be the basis of Call 3 developed applications, as e-commerce, copyright management and data marketplaces.

Finally, we have documented the specifications and requirements for an ONTOCHAIN network supported by some ONTOCHAIN participants for deploying the pilot use-cases/demonstrators, which will be used to have a better insight of the results of this project.

REFERENCES

- [1] Dominik Kuziski et al. "D4 Prototype Demonstration- Full Design Specification - GraphChain". In: *Deliverable* (2021).
- [2] Roberto García et al. "D4 Prototype Demonstration- Full Design Specification - CopyrightLY". In: *Deliverable* (2021).
- [3] Rebecca Johnson and Martin Martin Schaffner. "D4 Prototype Demonstration- Full Design Specification HIBI". In: *Deliverable* (2021).
- [4] Caspar Roelofs et al. "D4 Prototype Demonstration- Full Design Specification - OntoSSIVault". In: *Deliverable* (2021).
- [5] Junaid Arshad et al. "D4 Prototype Demonstration- Full Design Specification - Reputable". In: *Deliverable* (2021).
- [6] Marcel Muller et al. "D4 Prototype Demonstration- Full Design Specification - KnowledgeX". In: *Deliverable* (2021).
- [7] Giampaolo Bella et al. "D4 Prototype Demonstration- Full Design Specification POC4COMMERCE". In: *Deliverable* (2021).
- [8] Adam Back et al. "Enabling blockchain innovations with pegged sidechains". In: *URL: <http://www.opensciencereview.com/papers/123/enablingblockchain-innovations-with-pegged-sidechains>* 72 (2014).