SOFIE - Secure Open Federation for Internet Everywhere

779984

DELIVERABLE D4.4

Second Architecture and System Evaluation Report

Project title:	SOFIE – Secure Open Federation for Internet Everywhere
Contract Number:	H2020-IOT-2017-3-779984
Duration	1.1.2018 – 31.12.2020
Date of preparation:	28.04.2020
Authors:	Vasilios A. Siris, Spyros Voulgaris, Nikos Fotiou, Dimitrios Dimopoulos, Iakovos Pittaras, Yiannis Thomas, Michalis Tsenos, George C. Polyzos (AUEB-RC), Tommi Elo, Ektor Arzoglou, Veria Hoseini, Dmitrij Lagutin (AALTO), Filippo Vimini (LMF)
Responsible person:	Vasilios A. Siris (AUEB-RC, vsiris@aueb.gr)
Target Dissemination Level:	Public
Status of the Document:	Completed
Version	1.00
Project web-site:	https://www.sofie-iot.eu/

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 779984.

Table of Contents

List of Figures

List of Tables

List of Acronyms

API	Application Program Interface
AS	Authorization Server
BLE	Bluetooth Low Energy
BP	Business Platform
CLD	Causal Loop Diagram
CS	Charging Station
DEDE	Decentralized Energy Data Exchange
DEFM	Decentralized Energy Flexibility Marketplace
DER	Distributed Energy Resource
DID	Decentralized Identifier
DLT	Distributed Ledger Technology
DNS	Domain Name System
DR	Demand Response
DSO	Distribution System Operator
eIDAS	Electronic Identification, Authentication and Trust Services
ER	Electricity Retailer
ERC	Ethereum Request for Comment
ETH	Ether – Ethereum coin
EV	Electric Vehicle
EVSE	Electric Vehicle Supply Equipment
EVM	Ethereum Virtual Machine
FM	Fleet Manager
FSC	Food Supply Chain
GDPR	General Data Protection Regulation
GW	Gateway
Gwei	Gigawei – equivalent to one nanoether or 10^{-9} ETH
IAA	Identify, Authentication, and Authorization
ICT	Information and Communication Technology
ID	Identifier
ILG	Interledger Gateway
IoT	Internet of Things
JSON	JavaScript Object Notation
JWT	JSON Web Token
KPI	Key Performance Indicator
MAC	Message Authentication Code
MP	MarketPlace
MRMG	Mixed Reality Mobile Gaming
MS	MileStone
OAuth	Open Authorization
P2P	Peer-to-Peer
PaD	Provisioning and Discovery
PDS	Privacy and Data Sovereignty

PoI	Point of Interest
PoP	Proof of Possession
PoW	Proof of Work
PV	PhotoVoltaic
RAXX	Requirement number XX of SOFIE Architecture
RFXX	Requirement number XX of SOFIE Framework Component
RBAC	Role Based Access Control
RES	Renewable Energy Sources
RP	Reference Platform
SD	System Dynamics
SoC	State-of-Charge
SM	Supermarket
SR	Semantic Representation
TC	Test Case
TSO	Transmission System Operator
TR	Transportation
UC	Use Case
UML	Unified Modeling Language
URI	Uniform Resource Identifier
V2G	Vehicular to Grid
VC	Verifiable Credential
VP	Verifiable Presentation
WH	Warehouse
WoT	Web of Things
WP	Work Package
ZKP	Zero Knowledge Proof

1 Introduction

This deliverable contains the evaluation results from WP4's second evaluation cycle and its submission (April 2020/M28) coincides with the completion milestone of the second evaluation cycle (MS14). Since the first evaluation deliverable D4.3 (initial submission June 2019, revised in December 2019), the deliverables containing the second version of the SOFIE federation architecture (D2.4, initially submitted July 2019 and revised December 2019), the second version of the federation framework (D2.5 – August 2019), as well as the initial platform validation (D5.2 – June 2019) and business platforms – pilot release (D3.3 – September 2019) have been submitted. These deliverables contain a more stable description of the SOFIE architecture and federation framework components and of the pilot systems, including the pilot scenarios and their objectives. This has allowed the evaluation and validation results contained in the current deliverable to be more concrete and focused.

Additionally, two previous WP4 deliverables, namely D4.1 (Validation and Evaluation Plan – initial submission September 2018) and D4.3 (First Architecture and System Evaluation Report - initial submission June 2019) were revised in December 2019. The revised D4.1 contains a detailed validation and evaluation planning that includes responsibilities, tools, and methods. This planning has been followed while conducting the evaluation and validation work reported in the current deliverable. Specifically, the validation work and results reported in Section 3 of this deliverable are tracked through three validation matrices, each matrix dedicated to the architecture, component, and pilot validation. The matrices are based on the requirements for the SOFIE architecture and components, defined in deliverable D2.4, and for the pilots, defined in deliverable D5.2.

The revised D4.3 (First Architecture and System Evaluation Report) contains architecture KPIs and pilot-specific system performance KPIs, along with their target values. Section 2.2 of the current deliverable presents an update on the numbers achieved for the architecture KPIs (Table 8). The evaluation results for the framework components (Section 4) and the pilot emulation scenarios (Section 6) are related to their corresponding requirements (identified in deliverables D2.4 and D5.2 for the components and pilots, respectively), which are also considered in the validation matrices (Section 3). Furthermore, each pilot emulation scenario (Section 6) contains a subsection with a table that summarizes the performance results in relation to the corresponding system performance KPI targets.

Finally, the evaluation results reported in Sections 4, 5, and 6 of this deliverable use the tools identified in D4.1 (Validation and Evaluation Plan) and were performed in the testbed environments described in D4.2 (Testbed and Emulation Environment Design and Setup).

1.1 Goals of this deliverable

The goals for this second architecture and system evaluation are the following:

- to provide a second evaluation of the SOFIE approach, architecture, systems, and components, extending the results of the first evaluation, in order to promote the SOFIE approach and establish foundations for its impact on technology and business,
- to provide new results for component evaluation and validation and for the evaluation of pilot emulation scenarios, offering experience with the approach and techniques that can provide guidelines to the pilot validation and evaluation activities,
- to present results and tradeoffs that will help in the selection of the most appropriate ledger and interledger technologies for the SOFIE use cases and pilots, and in the selection of the specific use cases that are most appropriate for promoting the SOFIE approach,
- to present the evaluation approach and techniques internally to the project so as to have them scrutinized, in order to determine if they will be adequate, or if they need to be modified or extended, for a convincing evaluation, and

- to provide feedback that can assist in the further conceptual and technological design, development and evaluation (both regarding methodologies and their application).

1.2 Methodologies and approach

The methodologies employed for evaluation are many and diverse, from simple presentation of arguments and qualitative evaluation, through modelling, analytical evaluation and simulation, to implementation and measurements in real components and systems. The current deliverable followed the detailed validation and evaluation planning that includes responsibilities, tools, and methods contained in the revised D4.1 (Validation and Evaluation Plan – initial submission September 2018, revised submission December 2019). Additionally, the validation and evaluation results reported in this deliverable use the tools identified in D4.1 (Validation and Evaluation Plan) and were performed in the testbed environments described in deliverable D4.2 (Testbed and Emulation Environment Design and Setup).

Since the pilots have a central position in the SOFIE project, an important evaluation direction will be undertaken using each pilot, considering the actual system and evaluating it in a specific application context. These evaluations will be performed towards the end of the pilots' lifetime and they will be integrated with other WP4 evaluation results in the final WP4 evaluation deliverable (D4.5). However, they also provide more concrete systems and applications in which we consider the SOFIE approach and evaluate it. We also generalize from the specific choices made in the pilots to the use cases from which they have been inspired, evaluating many potential alternatives around them. This was started with the first evaluation deliverable D4.3 and continues with the results contained in the current deliverable.

Thus, we are inspired and guided by the pilots and their use cases, as well as the software and solutions developed for them; however, WP4 aims to have a wider scope. It has chosen as one key approach for evaluation the emulation and/or simulation of the use cases considered in the pilots, but in a more general context, considering and evaluating various possible solutions and their parameters, going beyond what is possible within the actual pilots. On the other hand, in order to achieve this breadth, it needs to model and abstract out various aspects of the pilots, as will be explicitly described below.

Since the first evaluation deliverable D4.3, the evaluation and validation approach has been adjusted. Specifically, the detailed description of the services and interfaces, along with the internal operation, of the framework components in D2.5 (Federation Framework, 2nd version – August 2019) has been taken into account in the validation and evaluation approach used to obtain the results in this deliverable. The mapping of pilot use cases to their corresponding application domains and the context of each pilot, as well as the initial platform validation results reported in D5.2 (Initial Platform Validation – June 2019) have been considered in adapting and extending the emulation scenarios considered in this deliverable.

In addition to the use cases, the questions to be answered (i.e., the targets of the evaluation) are diverse. They range from traditional performance metrics, which typically have limited generality, as they must refer to fully specified systems, to more general questions such as security analysis, robustness, usability and even business analysis. It is therefore even more obvious that the tools to be used for evaluation must be diverse and applied at very different abstraction levels and under different assumptions. Our starting point for the evaluation are the KPIs, which initially appeared in deliverable D2.2 (Federation Architecture, 1st version – August 2018) and were further refined and extended in the revised deliverable D4.3 (First Architecture and System Evaluation Report - initial submission June 2019, revised submission December 2019). The refinement and extension included the addition of target and pilot-specific system performance KPIs.

1.3 Structure of this deliverable

Below we present the structure of this deliverable, highlighting the new results compared to the previous (first) evaluation deliverable D4.3.

Section 2 is a high-level architecture evaluation, focusing more on the style and desired properties of the architecture, rather than very specific architectural components and structure. The text is based on the previous deliverable D4.3, verifying that the features identified in the previous deliverable are still relevant. Its content has been updated to take into account the validation and evaluation work and results that have been obtained up to this point.

Section 3 focuses on the validation of the SOFIE approach, including the architecture, component, and pilot validation. The results and current validation status are presented in three validation matrices, corresponding to the each of the three validation directions.

Section 4 focuses on component evaluation. Compared to the first evaluation deliverable D4.3, further evaluation results are presented for the Interledger, Privacy and Data Sovereignty (PDS), and Identification, Authentication, and Authorization (IAA) components. A discussion and new results are also presented for the three other components: Semantic Representation (SR), Marketplace (MP), and Provisioning and Discovery (PaD).

Section 5 contains further results on IoT resource access evaluation. The new results include decentralized authorization for constrained environments, where Ethereum is interconnected with Hyperledger Fabric, access control for multi-tenant IoT systems, and decentralized interledger gateway architectures for enhancing the reliability of the interledger functionality.

Section 6 on evaluation scenarios follows the SOFIE pilots, generalizes them into pilot inspired use cases by including alternative options that are not necessarily considered in the SOFIE pilots, and uses emulation and simulation to consider the various tradeoffs of many potential alternative design decisions. The investigations include alternatives for the hierarchy and interconnection of different types of blockchains (public/private, or permissionless/permissioned, etc.) and the impact of these choices. The new results included in this deliverable compared to the previous evaluation deliverable D4.3 include the following: For the Food Supply Chain (FSC) scenario, we introduce the use of multiple ledgers and interledger technology, as well as the alternative of using arbitrary private storages and storing hashes in a public ledger, comparing the end-to-end performance and cost of all considered architectures. All the results in this deliverable for the Decentralized Energy Flexibility Marketplace (DEFM) are new. For the Decentralized Energy Data Exchange (DEDE) scenario the new results concern new evaluation scenarios for the PDS and IAA components, which implement the authentication and authorization functionality required by the pilot scenario, including the usage of verifiable credentials to support privacy and OAuth 2.0 access tokens based on Ethereum ERC-721 tokens. We also extended the evaluation results of the previous deliverable that illustrate the tradeoffs involving the hash recording frequency and how they depend on various system parameters (transaction and verification costs, rate at which data are produced, and rate of verification requests). For the Mixed Reality Mobile Gaming (MRMG) scenario, we introduce two new emulation settings, the one utilizing Hyperledger Fabric, and the other combining the use of Hyperledger Fabric and public Ethereum, along with evaluation results considering execution cost, response time, throughput, and overall system scalability.

Section 7 addresses business platform evaluation (decentralized, built on SOFIE principles and, eventually, with SOFIE components). A System Dynamics approach is employed, focusing on the pilot-based use cases and considering the interactions among elements and forces. The Food Supply Chain platform is developed to the point of detailed model and system simulation, where we have investigated the impact of one key parameter, the status-quo resistance power factor, which denotes the ratio of the negative publicity of the platform to all publicity of the platform that controls the reluctance to adoption of the platform, a force countering the forces in favour of the adoption of the platform. The other three pilot inspired business platform System Dynamics models are at different stages of development, with the context aware mobile gaming platform model being the next most developed one.

Finally, the document concludes in Section 8 with a summary and outlook towards future WP4 work, which will focus on jointly analysing emulation/simulation testbed performance results with pilot evaluation and validation results.

2 Architecture evaluation and KPIs

In this section we discuss and bring out key aspects of the SOFIE architecture that are critical for the SOFIE approach to IoT system federation and open business platform success. In Section 2.1 we lay out the features of the SOFIE architecture, while in Section 2.2 we present the architecture KPIs and the system performance KPIs grouped by pilot. Table 8 shows the current status of the architecture KPIs. The status of the system performance KPIs are shown later, in the sections for each emulated pilot scenario, namely Sections 6.1.6 (Table 31 for FSC), 6.2.6 (Table 33 for DEFM), 6.3.5 (Table 35 for DEDE), and 6.4.6 (Table 40 for MRMG).

2.1 Architecture features

In this section we discuss the key architecture features of the SOFIE architecture. The text is based on the previous deliverable D4.3, verifying that the features identified in the previous deliverable are still relevant. We have added references to the new results contained in the current second evaluation deliverable. One feature added in the current deliverable is federation. Federation is an inherent notion in SOFIE, as the building of trust, security, and collaboration across many different business entities in private silos (platforms) is central to the design of the project. For this reason, we discuss it in some detail. Finally, the architecture features discussed in this section are loosely related to the architecture KPIs defined in Table 2 and discussed later. For example, decentralization and multiple ledger and interledger technology is related to the ledger and interledger use KPIs. Open business platforms and federation is related to the IoT operability and interoperability KPIs, as well as the ledger and interledger use KPIs. Finally, trust, security, transparency, availability, and accountability are related to basic properties of DLTs and the privacy and data sovereignty KPIs.

2.1.1 Decentralization

Decentralization constitutes an intrinsic characteristic of most large-scale systems in our world, spanning from social structures and financial systems to telecommunication infrastructure and transportation networks. In computer science, decentralization is the core principle around which many applications are designed, including peer-to-peer (P2P) applications, large-scale distributed systems, and network-centric frameworks. Often decentralization brings in various desirable properties, most notably scalability, fault tolerance and resilience to failures, and elasticity with respect to the amount of resources needed and provided.

In the case of blockchains, specifically, decentralization plays a key role in their cornerstone properties, namely *trust* and *immutability*. Cryptographic algorithms guarantee the dependable validation of transactions and trusted consensus protocols, but it is massive decentralization that translates trusted consensus into correct decisions, as the majority of the nodes are with very high probability unlikely to collude into cheating the system.

In SOFIE, we rely on this form of decentralization for guaranteeing the trust and immutability of recorded transactions. Along these lines, SOFIE-enabled solutions explicitly make use of public ledgers, such as Ethereum, to inherit their immutability guarantees.

There is a second flavour of decentralization pertaining to SOFIE-enabled solutions, namely the use of multiple distinct ledgers as opposed to a single one. This allows us to have more control over the required scalability, throughput, and availability of the system, as explained further in Sections 2.1.4 and 2.1.5.

The results in this deliverable assess decentralization via the use of multiple ledgers within the context of the pilot evaluation scenarios considered. Specifically, the FSC scenario introduces the use of multiple ledgers and interledger technology, as well as the alternative of using arbitrary private storages and storing hashes in a public ledger, and compares the end-to-end performance and cost of all considered architectures. The DEFM scenario includes a completely new evaluation. The DEDE scenario adds new evaluation cases covering the PDS and IAA components, considering the use of Verifiable Credentials to support privacy and OAuth 2.0 access tokens supported by Ethereum ERC-721 tokens, and contains new results on the tradeoffs involving the hash recording frequency and various system parameters (transaction and verification costs, rate at which data is produced, and rate of verification requests). Finally, the MRMG scenario introduces two new emulation settings, one utilizing Hyperledger Fabric and the other combining the use of Hyperledger Fabric and public Ethereum, and evaluates them in terms of the execution cost, response time, throughput, and overall system scalability.

2.1.2 Open business platforms

Further to decentralization, a key characteristic expected of future IoT and, more generally, business platforms, is openness. A business platform is a (software mainly) system where business transactions are undertaken with a high degree of automation. Maybe the best-known examples are the Apple App store and Google Play. In both these case (and almost all currently existing business platforms), Apple and Google, respectively, have a defining, central, all powerful, and rulemaking position, deciding on who can “play” and also extracting a hefty fee out of the platform. Not only is the platform not decentralized, but it is also not open, i.e., not open to other players without the explicit and typically not automatic agreement of the defining player and, in particular, not open to competitors or game changers.

The SOFIE philosophy, and claim to be proven, is that open platforms are the future. They can support evolution and fast transformation and provide the correct incentives for players to participate and innovate and for society to benefit more and to better control the process through general rules applied equally to all. An investigation of decentralized open platforms is described in Section 7 of this document using the System Dynamics methodology and with the ultimate goal of determining the conditions under which such platforms can emerge, grow, and prosper. Compared to the previous deliverable D4.3, the results in Section 7 have been extended with more concrete Causal Loop Diagrams (CLDs) and simulation results for the FSC model.

2.1.3 Federation

Federation is primarily a political term. It is used to refer to a political entity characterized by a union of (other) partially self-governing entities (e.g., state governments) under typically a central (federal) government [Wik20a]. Also, typically the powers of the state and federal governments are explicitly (agreed to and) stated in an agreement, e.g., the constitution (of the federation). A related term is Confederation. A Confederation can be considered a looser federation, where the power of the federal government is much weaker than in the case of a Federation. The opposite of a Federation (or Confederation) is a unitary state.

Our use of the term federation has a more technical flavour and applies to technical and business aspects. One can consider a group of component systems that are governed by either a single central authority or by distributed governance, in which each component system serves its own interests (but they can also demonstrate altruism, or desire for cooperation, sometimes against their narrowly defined self-interest). The analogy in real life is with a group of states/countries, e.g., (more like) the United States of America, or (in some respects also) the European Union. An example of a federated system in Information and Communication Technology (ICT) is the interconnection of mobile phone networks allowing roaming: each network operator runs their own network, but communications can happen also between them and various decisions are often made at the “federal” level (e.g., for rules on clearing procedures for roaming charges).

We will refer to a group of interacting ICT systems or platforms (see above and Section 7.1 for a definition and discussion of digital and business platforms) as a *federation*, if the composing systems have governing structures and rules each, but they also participate in a large common platform, which governs (e.g., through pre-set rules or distributed governance) some aspects of the group, in particular the inter-platform exchanges and some common aspects of all composing systems/platforms that define the federation, i.e., the common, group platform.

The *IoT systems* considered for this discussion are assumed to be business platforms relying on Distributed Ledger Technologies (DLTs) for trust and transactions immutability. In a *federation of IoT systems*, the platforms are connected via combining the distributed ledgers so that agreed upon transactions are valid and trusted, also from the point of view of others in the whole federation. SOFIE, in particular, employs DLTs to enable both the individual platforms and facilitate their federation. There is a straightforward economic logic of how a federation can scale up the ecosystem and, potentially, also increase the value generation, by what seems like a very straightforward governance action of deciding to share benefits and, thus, increase benefits for all participants. This relies on network effects, i.e., the value of the federation increasing super-linearly with the number of attached devices and users, therefore the value of the federation being higher than the sum of the values of its constituent platforms; Metcalfe's law [Wik20b] claims that the value of the network increases quadratically with the number of endpoints.

For a federation to be realised, e.g., among two or more commercial business platforms, in the past human social interaction and decision making were required, because federation is largely in the realm of governance. Until recently, technical methods were insufficient to bring about a successful federation on their own, since a federation relies on social and legal contracts and value sharing models outside the scope of the technical business platform enabling the federation.

Interledger technologies can be technical enablers for federation and a technical solution in case DLTs are part of the constituent platforms. In particular, if these DLTs support smart contracts, most of the governance decisions of the federation can be set by some original parties and expressed as smart contracts, so that the decision for others to join or not might mostly require business analysis, with fewer legal negotiations. In some cases, in the future, one could imagine that the governance rules could be gradually modified by the parties through off-line discussions, but voted on and implemented through the consensus mechanisms of the federation DLT, or even imagine splits of the federation as has been observed in Bitcoin forks.

2.1.4 Multiple ledgers and interledger technology

The single-ledger model may be sufficient for specific, well-defined application types, such as exchanging money through Bitcoin or registering DNS name entries through NameCoin, but may prove to be a bottleneck for complex applications involving a number of distinct roles, diverse types of interaction, and a large volume of data being processed at a high rate. In such cases of complex applications, the key to adopting DLTs is *flexibility.*

Public ledgers typically incur relatively high fees and exhibit longer latency to register a transaction, making them unsuitable for storing large volumes of data and at a high rate. Private ledgers, on the other hand, can register transactions at a negligible cost and are relatively faster than public ones, but their immutability guarantees are very low compared to those of public ledgers. A well-designed interaction between ledgers of both types can bring significant benefits to a deployed framework, combining low-cost and fast data storage with high immutability guarantees. This is why a key component of the SOFIE approach is the use of interledger technologies. We provide an up-to-date and comprehensive survey, review, characterization, and evaluation of interledger approaches in [Sir+19d].

In this deliverable's evaluation scenarios (Section 6), we explore and evaluate the use of multiple ledgers and we compare it against the use of a single centralized ledger, be it public or private. Our evaluation considers various alternative interledger architectures, including combinations of public and private, as well as permissioned and permissionless ledgers, quantifying their pros and cons and exploring the tradeoffs stemming from them.

2.1.5 Trust, security, transparency, availability, and accountability

These are key properties of Distributed Ledgers, allowing transactions stored in blockchains to be considered trusted, final, and secure. However, some of them may be complementary. For example, a blockchain's transparency property, that is, the fact that all data recorded in a blockchain is readily available to all nodes, may hurt security, for instance, when sensitive personal or business information needs to be stored in a ledger.

Regarding availability, the fact that any participating node stores the entire history and state of a blockchain means that any interested entity can run one or more nodes in that blockchain, increasing availability. Finally, by using a blockchain for authorization and for acquiring the keys to access a service or a device, all accesses are recorded, providing accountability. These properties are particularly significant given the envisioned interactions among entities and businesses with diverse, and often conflicting interests.

In this deliverable we explore and compare a number of different ledger organizations, involving a single public ledger, a public and a private ledger, and multiple private ledgers, shedding light on how the aforementioned properties are affected, and examining alternatives depending on an application's specific requirements.

Another relevant measure is usability, a qualitative measure that shows how easy it is for a system to be used, to be connected, etc. In this context it is important to also consider usability because DLTs might introduce significant complications in systems relying on them.[1]

2.1.6 How features are fulfilled

The following table presents how each of the aforementioned features are addressed in the SOFIE architecture.

Table 1: Architecture features

Feature	How feature is addressed
Decentralization	SOFIE is decentralized by design, pertaining to the collaboration of distinct business entities with private data silos. SOFIE's main target is to enable interaction in such a decentralized set of private silos. At an architectural level, decentralization refers to the segregation of SOFIE's architecture into multiple self-contained components, which are subsequently combined to serve SOFIE applications. It also refers to the use of multiple distinct ledgers to support SOFIE applications, distributing the load of ledger operations to multiple entities to increase scalability and throughput. Finally, at a low, implementation level, decentralization refers to the main technology behind SOFIE, blockchains, which are inherently decentralized; this is even more so when considering the interoperation of multiple diverse blockchains through the use of interledger technology.
Open business platforms	Openness is an intrinsic feature of the SOFIE project. The architecture, framework, and components proposed in SOFIE are open, with clearly defined operations and interfaces. Notably, SOFIE APIs are not hardcoded, but can be customized via SOFIE adapters. Moreover, the SOFIE architecture, framework, and components, enable open business platforms,

[1] ISO 9241-11:2018 provides usability definitions and guidelines.

	in the sense that these platforms can be open for all to join by simply conforming to the SOFIE architecture and using the SOFIE framework.
Federation	Federation is an inherent notion in SOFIE, as the building of trust, security, and collaboration across many different business entities in private silos (platforms) is central to the design of the project. In the case of SOFIE, more specifically, federation refers to the seamless collaboration across many entities with distinct administrations in such a way that the final result appears as a single, well-integrated platform.
Multiple ledgers and interledger technology	The use of multiple ledgers in SOFIE is dictated by two goals. First, in a multi-party business scenario, the use of multiple ledgers may reflect more accurately the interaction between different parties. For example, one blockchain could serve the interaction between parties A, B, and C, while a separate blockchain could serve parties X, Y, and Z. Second, different blockchains have different technical properties, such as transaction cost, block generation speed, smart contract capabilities, etc. Combining different blockchains offers SOFIE applications more flexibility in fulfilling specific requirements using best-of-both-worlds features. In our tests we conclude that there is no single universally best option, therefore we compare a wide range of architectures involving multiple ledgers to be able to assess the best option for each case.
Trust and Accountability	Trust and accountability are enforced in SOFIE via blockchains. For example, in the FSC pilot, recording box handovers in blockchains guarantees trust between trading parties, while parties responsible for inappropriate handling of produce can be held accountable based on ground-truth records.
Security	Security in SOFIE is managed by the IAA, and PDS components, which encompasses Hyperledger Indy, a popular Decentralized Identifiers (DIDs) and Verifiable Credentials (VCs) implementation, JSON web tokens, and OAuth2.0. This deliverable expands on the evaluation of these components.
Transparency	Transparency is enforced via the use of blockchains. For example, in the DEFM pilot, the Marketplace component is responsible to guarantee transparency regarding current energy prices among electric vehicles (EVs), charging stations (CSs) and distribution system operators (DSOs). This is demanded by pilot requirements and is tested in our corresponding evaluation scenarios.
Availability	Availability is an inherent feature of blockchains. As a blockchain is maintained by many nodes distributed across diverse geographic locations and administrative jurisdictions, the probability of all nodes crashing or becoming unresponsive simultaneously becomes extremely small. This is further strengthened when multiple distinct ledgers are used in parallel.

2.2 Architecture and System Performance KPIs

Deliverable D2.2, Annex 1, defined the KPIs to be used for the evaluation of the SOFIE architecture. In this section we further refine and extend these KPIs, and report our progress on the architecture evaluation, based on the emulated scenarios considered in previous deliverable D4.3 and in the current deliverable, with respect to the defined architecture KPIs. The system performance KPIs for the pilot scenarios will be investigated in Section 6.

The KPIs are shown collectively in the table below. For each KPI, the table indicates the metric for measuring the KPI, the method of verification or measurement and the target value. The KPIs related to system performance are shown later, in individual tables for each pilot.

Table 2: KPIs with targets

KPI	Goal	Description	Metric	Method of verification	Target
1	IoT operability	Prove operability of the implementation with IoT silos	Number of IoT silos	Detection of data flow in silos during implementation use case	5
2	IoT inter-operability	Prove interoperability across multiple IoT silos of the reference architecture	Number of IoT silo pairs	Implementation use case accesses data or actuates operations in different IoT silos	3
3	Ledger use	Validate SOFIE implementation capability with multiple ledgers	Number of distributed ledgers	Ledgers have detectable data passing through SOFIE implementations	5
4	Interledger use	Validate SOFIE implementation operating across multiple ledgers	Number of distributed ledger pairs	Implementation use case shown to result in operations across multiple ledgers	3
5	Ledger independen ce	Demonstrate capability of developing applications using ledgers, where a sufficient abstraction can be provided to applications to allow them to be targeted simultaneously to multiple ledger technologies	Number of Business Platforms (BP) samples classified into success or partial success	Demonstrate that a BP sample can be deployed on two ledgers with only configuration changes, and the BP sample users are able to use either one with only configuration item changes	3
6	Privacy designed in as a fundamental requirement	Demonstrate GDPR compliance where relevant	Number of operational GDPR features referenced and supported.	Final specifications have clear references to features implementing named GDPR requirements. Relevant pilot specifications also refer to the needed features	4
7	Device owner payments across ledgers	Ability of silo owners to send and receive payments or other value transfers	Number of ledger pairs supporting value transfer	Observation of value transfer as part of a use case in an implementation	2
8	Data sovereignty	Ability of data owners to reject or allow access, possibly for a specific time interval, to their data Each datum has an accompanying	Number of pilot use cases utilizing data owner data sovereignty features where data owner is from a different silo than the storage silo	Count the number of use cases	3

		authorization list, which the data owner can modify			
9	User responsiveness	Apparent responsiveness of system for end users	Number of seconds within which user gets response for an action initiated by the user	Measuring from the onset of user action until the user gets a response by the system (to the user interface he or she is using)	See system performance KPI table.
10	System performance	Overall system performance reflecting the diverse needs and requirements of different use cases	Acceptable system performance for users and pilots	Qualitative evaluation of system metrics.	See system performance KPI table.

The general system performance KPIs together with their method of measurement and target are shown in the table below. Later tables will adapt this table to present pilot scenario specific KPIs.

Table 3: System performance KPIs

KPI	**Name**	**Description**	**Metric**	**Method of measurement**	**Target**
10.1	Ledger execution cost	Cost for executing operations on a ledger	Ledger execution cost units (e.g., gas in Ethereum)	Measure the total execution cost for all operations that a transaction involves	As low as possible
10.2	Configuration time	Time for configuration to complete	Time units (e.g., seconds)	Measure time between start of configuration until completion of configuration	<15 s
10.3	Response time or latency (or transaction delay)	Time for the system to respond to a request or to execute a transaction	Time units (e.g., seconds)	Measure time between instant system receives a request or transaction until the instant that the system responds	<5 s (if human involved) <1 s (if no human involved)
10.4	Throughput	Maximum number of transactions per time unit that the system can support	Number of transactions per time unit	Measure number of transactions per time unit that can be supported while the QoS (e.g., in terms of maximum response time) is satisfied	Domain specific
10.5	Scalability – cost	Increase of cost as load (e.g., number of transactions per time unit, number of nodes) increases	Ratio of delta cost over delta of load (number of transactions/nodes)	Measure cost for different loads	Linear or sublinear
10.6	Scalability – time	Increase of response time as load (e.g., number of transactions per time unit, number of nodes) increases	Ratio of delta time over delta of load (number of transactions/nodes)	Measure response time for different loads	Linear or sublinear

A number of performance KPIs have been laid out for the evaluation of the FSC scenarios. These KPIs are presented in Table 4, below.

Smart contract execution on public ledgers, such as Ethereum, incurs a cost. Thus, our first KPI refers to this cost, demanding that it is kept as low as possible. Apart from some monetary cost, each interaction with a ledger also incurs a time cost. Given that Ethereum blocks are being generated every 15 seconds *on average* (thus, they could occasionally take much longer), and that a transaction submitted to public Ethereum is not guaranteed to be included in the exact next block, we believe that a target of 1 minute for handover transactions and 0.5 minutes for internal state transitions, constitutes a reasonable target for the anticipated system. Besides cost and timing limits for individual transactions, the FSC scenario is mainly concerned with *throughput*, that is, the number of products that can be processed through the chain per time unit. Having received input from a large association of producers in Greece, a number of 6000 boxes are produced per day during peak harvesting season. Thus, we set this number as a target for box processing throughput. Then follows the issue of scalability, which comes in two flavours, namely time and cost scalability. Scalability refers to the effect of the volume of box processing on the actual time delay and cost individual boxes incur. Time and cost per individual box should not increase by the volume of box transfers, which would render the system non-scalable. This is expressed by the two scalability requirements demanding that time and cost for transactions involving a number of boxes grow at most linearly with the number of boxes. Finally, a KPI is defined for the time it takes to retrieve all data needed for auditing, that is, to resolve a potential dispute. Given that multiple blockchains might have to be accessed to retrieve all relevant data, this value should not be higher than 1 minute.

Table 4: System performance KPIs for FSC scenarios

KPI	Name	Description	Metric	Method of measurement	Target
KPI_FSC_1	Ledger execution cost in public ledger	Cost for executing operations on a ledger	Ledger execution cost units (e.g., gas in Ethereum)	Measure the total execution cost per box	As low as possible
KPI_FSC_2	Handover time	Time to register data to blockchain during a handover between two stages	Time unit (e.g., seconds)	Measure the total time required for blockchain-related operations during a handover of a box between two stages	<1 min
KPI_FSC_3	Internal state transition time	Time to register data to blockchain during a box's state transition occurring internally within a single stage	Time units (e.g., seconds)	Measure the total time required for blockchain-related operations during a state transition of a box within a single stage	<30 s
KPI_FSC_4	Throughput	Number of boxes that can be processed per time unit in any possible handover or internal state transition	Number of boxes per time unit	Measure the handover and state transition delays	> 6000 boxes per day
KPI_FSC_5	Scalability - time	Blockchain registration time for a	Derivative of the blockchain	Measure handover and state transition blockchain	Linear or sublinear

		handover or internal state transition, as a function of the number of boxes involved	registration time with respect to the number of boxes involved	registration time as a function of the number of boxes involved	
KPI_FSC_6	Scalability - cost	Public blockchain costs associated with box handovers or internal state transitions, as a function of the number of boxes involved	Derivative of public ledger cost with respect to the number of boxes involved	Measure public blockchain cost for handovers and state transitions as a function of the number of boxes involved	Linear or sublinear
KPI_FSC_7	Response time for audit requests	The time it takes to respond to an audit request, by pulling out all data related to the box in question	Time units (e.g., seconds)	Measure the time it takes to pull out all records related to a given box, and to cross check them to identify potential issues	<1 min

The system performance KPIs for the DEFM scenarios are shown in the table below.

As in the previous scenario, Smart contract execution on public ledgers, such as Ethereum, incurs a cost. Thus, our first KPI refers to this cost, demanding that it is kept as low as possible. The next three KPIs concern response time, which previous studies have identified as an important factor for energy trading, e.g. see [Haa+18]. Moreover, previous studies suggest that the maximum value for the latency is of the order of minutes [Smart16]. The throughput and scalability of an energy marketplace system is a significant metric that characterizes the capability of the system to handle energy transactions. Similar to the other pilot scenarios, the scalability of the system should be linear or sublinear.

Table 5: System performance KPIs for DEFM scenarios

KPI	Name	Description	Metric	Method of measurement	Target
KPI_DEFM_1	Ledger execution cost	Cost for executing operations on a ledger	Ledger execution cost units (e.g., gas in Ethereum)	Measure the total execution cost for all operations involved	As low as possible
KPI_DEFM_2	Response time for requests, offers, and charging event notifications	Latency of placing flexibility requests and offers on the marketplace	Minutes	Measure the time between the issuance of transaction by respective party until the transaction is recorded on the marketplace	<5 min
KPI_DEFM_3	Response time for determining the winner of the auction	Latency of determining and notifying the winner of the marketplace auction	Minutes	Measure the time between the deadline of bids and offers until the winner of the auction has been determined and notified	<5 min

KPI_DEFM_4	Response time for verifying the winning bid and compensating (or finding) the winner	Latency of verifying the winning bid and compensating (or fining) the winner	Minutes	Measure the time between sufficient charging events have been recorded on the marketplace, until the events have been verified and the winner has been properly compensated. If recorded charging events did not satisfy the requirement of the bid during its timeframe, measure time between the end of the bid's deadline until the verification of the failure of the bid and finding the winner of the bid.	<5 min
KPI_DEFM_5	Throughput	Number of transactions (bids, offers, selections of winning bid, charging event notifications, bid verifications, etc.)	Number of transactions per time unit (hour)	Measure number of transactions per time unit (hour) that can be supported while the QoS (e.g. in terms of maximum response time) is satisfied	>100 per hour
KPI_DEFM_6	Scalability – time	Increase of response time as load (e.g., number of transactions per time unit, number of nodes) increases	Ratio of delta time over delta of load (number of transactions/nodes)	Measure response time for different loads	Linear or sublinear

The system performance KPIs for the DEDE scenarios are shown in the table below.

As in the previous scenarios, Smart contract execution on public ledgers, such as Ethereum, incurs a cost. Thus, our first KPI refers to this cost, demanding that it is kept as low as possible. The response time metrics reflect the importance of latency in energy related marketplaces and data exchange platforms, e.g., see [SysFI19]. The scalability of data exchange systems is a significant metric that characterizes the capability of the system to handle data exchange transactions. Similar to the other pilot scenarios, the scalability of the system should be linear or sublinear.

Table 6: System performance KPIs for DEDE scenarios

KPI	Name	Description	Metric	Method of measurement	Target
KPI_DEDE_1	Cost for computing discounts	Cost for executing discount operations on a ledger	Ledger execution cost units (e.g., gas in Ethereum)	Measure the total execution cost for all operations involved	As low as possible
KPI_DEDE_2	Cost for recording hashes	Cost for recording hashes on a ledger	Ledger execution cost units (e.g., gas in Ethereum)	Measure the total execution cost for recording hashes	As low as possible
KPI_DEDE_3	Response time for access requests	Time for the system to respond to metering data access requests	Time units (e.g., seconds)	Measure time between instant system receives a request until the instant that the system responds	<5 s

KPI_DEDE_4	Response time for DID operations	Time for performing read/write operations on the identity ledger (Hyperledger Indy)	Time units (e.g., seconds)	Measure time between instant system receives a request until the instant that the system responds	<5 s
KPI_DEDE_5	Response time for KSI Blockchain signatures	Time for retrieving KSI Blockchain signature	Time units (e.g., seconds)	Measure time between instant system receives a request until the instant that the system responds	<2 s
KPI_DEDE_6	Processing time of requests in adapter	Time for the processing incoming requests in adapter - includes audit log entry, verifying credentials, setting up secure channel	Time units (e.g. seconds)	Measure time between instant system receives a request until the instant that the system responds	<5 s
KPI_DEDE_7	Response time for audit logs	Time for the system to respond to audit log requests	Time units (e.g., seconds)	Measure time between instant system receives a request until the instant that the system responds	<15 s
KPI_DEDE_8	Scalability – cost	Increase of cost as load (number of discount computations or hash recordings per time unit) increases	Ratio of delta cost over delta of load (number of discount computations or hash recordings per time unit)	Measure cost for different loads	Linear or sublinear
KPI_DEDE_9	Scalability – time	Increase of response time as load (e.g. number of transactions per time unit, number of nodes) increases	Ratio of delta time over delta of load (number of transactions/nodes)	Measure response time for different loads	Linear or sublinear

For a quantitative performance evaluation of the emulated MRMG pilot, various measurable metrics are required. First, transactions on a public blockchain incur a transaction cost, which in Ethereum is expressed as the cost of gas for executing transactions on the Ethereum Virtual Machine (EVM). The desirable target for this metric is to have as low a cost as possible. Furthermore, the most common performance metric of any system is the response time required by the system to execute read and write requests. In our case, where the gaming system utilizes blockchains, the response time metric corresponds to the time that the system performs read and write transactions. Kalra et al. [KSD18], in the evaluation of their presented system, present the latency of various multiplayer FPS games. The average latency for these games is 250 milliseconds. Moreover, Cai et al. [Cai+18] state that the desirable latency of any blockchain-based system, even for games that utilize blockchains, is 2 to 3 seconds. Thus, for a blockchain-based mobile game, the latency should be 3 seconds for write requests and 1 second for read requests, respectively.

Other performance metrics that are important in mobile gaming include the time that an IoT device needs to detect the player arriving at a particular location. The average time for an Android smartphone to detect a beacon is 5 seconds. So, we believe that a reasonable target for this metric is 4 seconds. Finally, the throughput and scalability of the mobile gaming system is a significant metric, since it characterizes the capability of the system to handle many users and games. We believe that the scalability of the system should be linear or sublinear in order for the system to support many users and transactions.

The aforementioned metrics along with their targets constitute the KPIs for the MRMG scenarios and are shown in the following table.

Table 7: System performance KPIs for the MRMG scenarios

KPI	Name	Description	Metric	Method of measurement	Target
KPI_MRMG_1	Public ledger execution cost	Cost for executing operation on a public ledger	Ledger execution cost units (e.g., gas in Ethereum)	Measure the total execution cost for all operations that a transaction involves	As low as possible
KPI_MRMG_2	Response time for write requests	Time for the system to respond to game state altering transactions, such as challenge creation & completion, skipping tasks and buying in-game items	Time units (e.g., seconds)	Measure time between instant system receives a request or transaction until the instant that the system responds	< 3 s
KPI_MRMG_3	Response time for read requests	Time for the system to respond to non-altering requests such as getting player's currencies and items	Time units (e.g., seconds)	Measure time between instant system receives a request or transaction until the instant that the system responds	< 1 s
KPI_MRMG_4	BLE beacon detection time	The time player has to wait between walking into the correct location and receiving the context-dependent task	Time units (e.g., seconds)	Measure average time between the instant player walks into the correct location and the client detects the beacon	< 4 s
KPI_MRMG_5	Throughpu t	Maximum number of transactions per time unit that the system can support	Number of transactions per time unit	Measure transactions per time unit	> 222 read and > 133 write transactions per second
KPI_MRMG_6	Scalability – cost	Increase of cost as number of challenges or active users increases	Ratio of delta cost over delta of challenges or active users	Measure cost for different numbers of challenges or active users	Linear or sublinear
KPI_MRMG_7	Scalability – time	Increase of response time as number of challenges or active users or increases	Ratio of delta time over delta of challenges or active users	Measure response time for different numbers of challenges or active users	Linear or sublinear

Next, we discuss each system architecture KPI in Table 2, identifying their current values; they are summarized in Table 8. The user response and overall system performance KPIs will be presented in Sections 4, 5, and 6.

KPI	Goal	Description	Metric	Method of verification
1	IoT operability	Prove operability of the implementation with IoT silos	Number of IoT silos	Detection of data flow in silos during implementation use case

The objective of this KPI is to prove the applicability of the SOFIE federation architecture and its components to existing IoT silos. The corresponding metric is the number of IoT silos where the architecture has been applied. The current version of the architecture and a subset of its components have been applied and evaluated to the following five scenarios/silos:

- IoT resource access
- FSC
- DEFM
- DEDE
- MRMG

Each scenario utilizes different features of the architecture and its components, such as authentication and authorization, recording of data or hashes and execution of smart contracts in private/permissioned and public DLTs.

In the current deliverable we present new evaluation results for the five aforementioned scenarios: for IoT resource access in Section 5 and for the four pilot scenarios in Section 6. Hence, the achieved number of this KPI is 5.

KPI	Goal	Description	Metric	Method of verification
2	IoT inter-operability	Prove interoperability across multiple IoT silos of the reference architecture	Number of IoT silo pairs	Implementation use case accesses data or actuates operations in different IoT silos

This KPI focuses on the application of the architecture and its components to allow communication between different silos. For example, these silos can involve the IoT platforms of different entities, such as the smart farming platform, the transportation platform, and the logistics platform in the FSC scenario. The corresponding metric that represents this KPI is the number of IoT silo (platforms) pairs that exchange data through the SOFIE architecture. The interoperability of the IoT platforms will necessarily consider the SR component. The evaluation results for the pilot scenarios that are reported in Section 6 emulate the various entities (IoT platforms) and focus on the cross-ledger interactions. They do not consider the interoperability of the platforms at the semantic level. Based on this, the evaluation scenarios do not contribute to this KPI. Rather, the IoT platform interoperability KPI is addressed by the SOFIE pilot evaluation work. Specifically, as reported in deliverable D2.5 (Federation Framework, 2nd version) and D5.2 (Initial Platform Validation), the FSC pilot will demonstrate the interoperability of three IoT platforms, namely the smart farming IoT platform, the transportation IoT platform, and the logistics IoT platform, hence two IoT platform pairs. The DEFM pilot will demonstrate the interoperability of the electric vehicle and supply equipment platforms. The DEDE pilot will demonstrate the interoperability of a national data hub platform with smart meter platforms. The MRMG pilot will demonstrate the interoperability of the gaming, IoT beacon services, and asset platforms, hence two IoT platform pairs. The above give a total of six IoT platform pairs.

Finally, it is worth mentioning that the SOFIE project within WP5 will investigate cross-pilot use cases that involve the transfer of data and/or value between pilots.

KPI	Goal	Description	Metric	Method of verification
3	Ledger use	Validate SOFIE implementation capability with multiple ledgers	Number of distributed ledgers[2]	Ledgers have detectable data passing through SOFIE implementations

A key goal of the SOFIE platform is to utilize different DLTs with different performance tradeoffs (execution cost and transaction time) and features, such as privacy, transparency, and trust. The DLTs that have been used in the evaluation experiments reported in deliverable D4.3 and in the current deliverable, along with the pilots defined in WP5, include the following ledgers:

- Public Ethereum, including the Rinkeby and Ropsten public Ethereum test networks
- Private Ethereum network
- Hyperledger Fabric
- Hyperledger Indy
- Sovrin
- KSI blockchain

This deliverable contains evaluation results for the first four DLTs. The PDS and IAA code related to DIDs and VCs has been tested and works with Sovrin. The interaction with Sovrin is orthogonal (since Sovrin is a registry and all calculations take place locally, in the entities considered in our evaluation scenarios) and does not affect the performance results reported in the current deliverable.

KSI is utilized in the decentralized data exchange pilot. Although not considered directly in the current deliverable, KSI can be the public ledger considered in the hierarchical ledger architectures for the food supply chain investigated in Section 6.1 and the public ledger in the high level model capturing the tradeoffs for the hash recording frequency that are presented in Section 6.3.4.

Based on the above, the current number achieved for the ledger use KPI is 5.

KPI	Goal	Description	Metric	Method of verification
4	Interledger use	Validate SOFIE implementation operating across multiple ledgers	Number of distributed ledger pairs	Implementation use case shown to result in operations across multiple ledgers

A primary goal of SOFIE is to enable the interoperation and exchange of information across different DLTs with different performance tradeoffs and features. The evaluation experiments reported in the previous deliverable D4.3 focused on the interoperation of the Rinkeby and Ropsten public Ethereum testnets with a private Ethereum network. This deliverable contains new results investigating the cross-ledger interaction of public Ethereum testnets and a private Ethereum network for the food supply chain (Section 6.1) and mobile gaming (Section 6.4).

The current deliverable also reports evaluation experiments considering the interoperation of the public Ethereum testnets with a Hyperledger Fabric permissioned ledger. Specifically, this cross-ledger interaction is investigated for the interledger component in Section 4.1, the decentralized authorization for constrained IoT resource in Section 5.1, the interledger gateway architectures investigated in Section 5.3, and the MRMG evaluation results in Section 6.4.

[2] Across significantly different ledger technologies; e.g., Ethereum and Ethereum Classic are not considered different ledgers, as their differences are small enough to allow applications developed on Ethereum to be deployed on Ethereum Classic with only minor changes.

The PDS component, as described in Section 4.2., can generate an access token based on the verification of a client's credentials, using Hyperledger Indy. Then this access token can be recorded in an Ethereum smart contract and used by the IAA components, as discussed in Section 4.3.

Based on the above, our results have demonstrated the interaction between two ledger pairs: 1) public Ethereum and private Ethereum and 2) Ethereum (both public and private) and Hyperledger Fabric. Hence, the current number achieved for the interledger KPI is 2.

Ongoing evaluation work is investigating the interaction between the following pairs of DLTs:

- Private Ethereum - KSI blockchain
- Hyperledger Fabric - KSI blockchain

KPI	Goal	Description	Metric	Method of verification
5	Ledger independence	Demonstrate capability of developing applications using ledgers, where a sufficient abstraction can be provided to applications to allow them to be targeted simultaneously to multiple ledger technologies	Number of Business Platforms (BP) samples classified into *success* or *partial success*	Demonstrate that a BP sample can be deployed on two ledgers with only configuration changes, and the BP sample users are able to use either one with only configuration item changes

The goal of this KPI is to demonstrate the capability of developing applications using sufficient abstractions that allow the applications to run over multiple ledger technologies.

The experiments in the previous deliverable D4.3 and the current deliverable (Sections 4.1 and 5.1) demonstrate the implementation of functions and services related to IoT resource access both in a private Ethereum network and a Hyperledger Fabric permissioned ledger. The results in Section 4.2 utilize VCs as an authorization grant compatible with the OAuth 2.0 authorization framework. The proposed approach should be compatible with any ledger technology that follows W3C specifications; we have verified our constructions in Hyperledger Indy and in Sovrin's testing network. Section 4.3 investigates the use of authorization tokens backed by Ethereum ERC-721 tokens. On the other hand, the results in the previous deliverable D4.3 considered off-chain authorization tokens. Based on the above, our experimental results have demonstrated that subcomponents of the IoT resource access application, and both the PDS and the IAA can be deployed on different ledgers (Ethereum, Hyperledger Fabric, and Hyperledger Indy)

The usage of different DLTs for implementing various services is demonstrated by the results for MRMG in Section 6.4, where various functions that include main gaming functions, advertisement functions, and token and reward functions can be implemented on both an Ethereum network (private and public) and Hyperledger Fabric.

Based on the above, the current number achieved for the ledger independence KPI is 2 (IoT resource access and mobile gaming scenarios).

KPI	Goal	Description	Metric	Method of verification
6	Privacy designed in as a fundamental requirement	Demonstrate GDPR compliance where relevant	Number of operational GDPR features	Final specifications have clear references to features implementing named GDPR

			referenced and supported.[3]	requirements. Relevant pilot specifications also refer to the needed features

This KPI concerns the compliance of the SOFIE architecture with the GDPR and its metric is the number of operational GDPR features referenced and supported. There are various features of the SOFIE architecture that are related to privacy and GDPR. The relevant GDPR articles based on the GPDR checklist[4] are identified in parentheses. Firstly, as discussed and investigated in Section 4, the SOFIE architecture does not record personal data to immutable ledgers. This is necessary to support the "right to be forgotten" (GDPR Article 17 – Right to erasure ('right to be forgotten')). Instead, the immutability of data recorded in local databases is ensured by recording hashes of the data in public ledgers. Secondly, in the various scenarios only the minimum set of data is stored in a public ledger, in order to ensure the correct operation and functionality that pertains to the specific scenario. Also, based on DIDs the SOFIE architecture can support pseudonymisation (GDPR Article 25 – Data protection by design and by default, GDPR Article 32 – Security of processing).

SOFIE's applications do not process data without having permissions granted by users (consent), e.g., in the MRMG scenario when the user installs the app, a pop-up screen is displayed asking for Storage, Access location, etc., permission. Furthermore, in the IoT resource access use case consent is provided through access tokens, which can be revoked or valid for a specific time duration, and whenever VCs are used as authorization grants, a user can select which claims of a VC can be revealed (GDPR Article 7.3 – Conditions for consent). Authorizations are recorded in an immutable manner on a DLT, which allows verification in a non-repudiated way that the user (owner of data) provided consent (GDPR Article 7.1 – Conditions for consent, Article 6 – Lawfulness of processing). Moreover, through the PDS and IAA components, IoT resource owners can provide access to clients (GDPR Article 20 – Right to data portability).

Based on the above discussion, the following GDPR features are referenced and supported by the SOFIE architecture: 1) 'right to be forgotten', 2) pseudonymisation, 3) user selects which claims can be revealed, 4) authorizations immutably recorded on a DLT serve as proof of user consent, and 5) users can provide access to their data. This gives a total of 5 operational GDPR features referenced and supported by the SOFIE architecture.

KPI	Goal	Description	Metric	Method of verification
7	Device owner payments across ledgers	Ability of silo owners to send and receive payments or other value transfers	Number of ledger pairs supporting value transfer	Observation of value transfer as part of a use case in an implementation

Whereas KPI 4 on "Interledger use" focuses on the interoperability, in general, between different ledgers, this KPI concerns the transfer or, more accurately, the exchange of value, between different ledgers. An example of such an exchange of value is discussed in detail in the previous deliverable D4.3, which involved the exchange of a payment token, stored in a public Ethereum blockchain, with an access token stored in a private Ethereum blockchain. This exchange is performed using functionality of the interledger component. The exchange of value between

[3] The number of GDPR articles which lead to operational goals is generally thought to be about 10. See e.g., https://iapp.org/resources/article/top-10-operational-impacts-of-the-gdpr/

[4] GDPR checklist for data controllers: https://gdpr.eu/checklist/

Ethereum and Hyperledger Fabric utilizing the interledger component is illustrated in the IoT resource access scenarios reported in Sections 4.1 and 5.1).

Based on the above, our results have demonstrated value transfer between two ledger pairs: 1) public Ethereum and private Ethereum and 2) Ethereum (both public and private) and Hyperledger Fabric.

KPI	Goal	Description	Metric	Method of verification
8	Data sovereignty	Ability of data owners to reject or allow access, possibly for a specific time interval, to their data Each datum has an accompanying authorization list, which the data owner can modify	Number of pilot use cases utilizing data owner data sovereignty features and data owner is from a different silo than the storage silo	Count the number of use cases

This KPI is related to the ability of data owners to reject or allow access, possibly for a specific time interval, to their data. This KPI can be verified with the number of pilot use cases utilizing data owner data sovereignty features, where the owner can be in a different silo than the storage silo.

All scenarios presented in the previous deliverable and Section 6 can leverage the PDS component of the SOFIE architecture to achieve data sovereignty. Solutions related to IoT resource access with specific functionality and features are investigated in detail in the previous deliverable D4.3 and in Section 5 of the current deliverable. The number reported for this KPI is 4 (same as in previous deliverable), based on the emulated scenarios considered.

The table below summarizes the project's current achieved numbers for the architecture KPIs. The specific achieved numbers for each architecture KPI are discussed above. The values in the table include the emulated scenarios (i.e., scenarios that include emulated entities such as IoT devices/platforms and users) reported in this report.

Table 8: Current status of the SOFIE architecture KPIs

KPI	Goal	Metric	Target	Number in D4.3	Current achieved number
1	IoT operability	Number of IoT silos	5	5	5
2	IoT interoperability	Number of IoT silo pairs	3	-	-[5]
3	Ledger use	Number of distributed ledgers	5	4	5
4	Interledger use	Number of distributed ledger pairs	3	1	2
5	Ledger independence	Number of BP samples classified into success or partial success	3	-	2
6	Privacy designed in as a fundamental requirement	Number of operational GDPR features referenced and supported	5	3	5
7	Device owner payments across ledgers	Number of ledger pairs supporting value transfer	2	1	2
8	Data sovereignty	Number of pilot use cases utilizing data owner data sovereignty features and data owner is from a different silo than the storage silo	3	4	4[6]
9	User responsiveness	Number of seconds user gets response for an action initiated by the user			Evaluation measurements are reported in Sections 4 and 6
10	System performance	Acceptable system performance for users and pilots			Evaluation measurements are reported in Sections 4 and 6

[5] This KPI will be addressed by the WP5 pilot validation work. See deliverable 2.5 (Federation Framework, 2nd version) and 5.2 (Initial Platform Validation) for more details. The evaluation results in the current deliverable consider emulation/simulation scenarios, hence do not consider this metric.
[6] This number refers to emulated pilot scenarios presented in the previous deliverable D4.3 and the current deliverable.

3 Validation

The validation work focuses on checking whether the SOFIE platform meets the stakeholders' requirements. The validation process includes quantitative and qualitative tests of all components, pilots and KPIs. The specific tests considered are identified in the corresponding validation matrices that are presented in this section.

3.1 Validation strategy

The validation process is divided into internal validation and external validation. Internal validation consists of testing the SOFIE architecture, components and pilots against the requirements, while external validation uses the pilot to demonstrate SOFIE capabilities to the end-users and other relevant stakeholders, as defined by the corresponding pilot requirements. The requirements to perform internal validation are used as a starting point to create tests which are then run in the validation environment (more information about validation tools are found in D4.1 and information about the validation environment are found in WP3 documentation). The external validation is performed for the pilots and KPIs.

The validation work is an ongoing process. In this deliverable we report the initial outcomes of the validation work. The final outcomes of the validation can be found in D5.3, for SOFIE components and D5.4, for the pilots and KPIs.

3.1.1 The Validation Matrix

The validation work is tracked using the validation matrix. The validation matrix is a table which reports all relevant information of the validation process. The purpose of the matrix is to facilitate the validation process among SOFIE partners and, later, auditing from an external reader. The validation process is defined in four main steps and for each step the matrix collects the outcome and important information:

1. *What is validated*; the focus of the validation is SOFIE requirements, the matrix defines the requirement ID and requirement description. These fields are coherent with the corresponding fields found in D2.4 for the SOFIE architecture and components, and in D5.2 for the pilots.
2. *How the requirements are met*; the matrix defines the test approach used to assure the requirements are met. The test approaches can be software testing (functional test, unit-test, integration test), documentation or field test. In addition to the test approach, the table includes the test description and a rationale which describes why the test approach is compelling for validating a requirement.
3. *How to verify our claim*; for every test the test location is defined which, as the name suggests, points to the location of the test/s. The software tests for component and pilots are in the SOFIE GitHub repositories; for these tests, the table defines the name of the repository and the directory path of the repository where a specific test if located.
4. *Validation results*; the matrix define the outcome produced by the validation process.

In the following chapters of this deliverable we present the initial version of the validation matrix for components and pilots.

3.1.2 Work package responsibilities

The validation is a combined work between WP2, WP3, WP4 and WP5. Each work package is responsible for handling different aspects of the validation process. For instance:

- WP2 is responsible for creating the internal validation tests for the SOFIE components
- WP3 is in charge of the development, the set up and maintenance of the validation environment

- WP4 acts as the initial validator and validation process orchestrator
- WP5 is responsible for handling the pilots and KPI external validation process

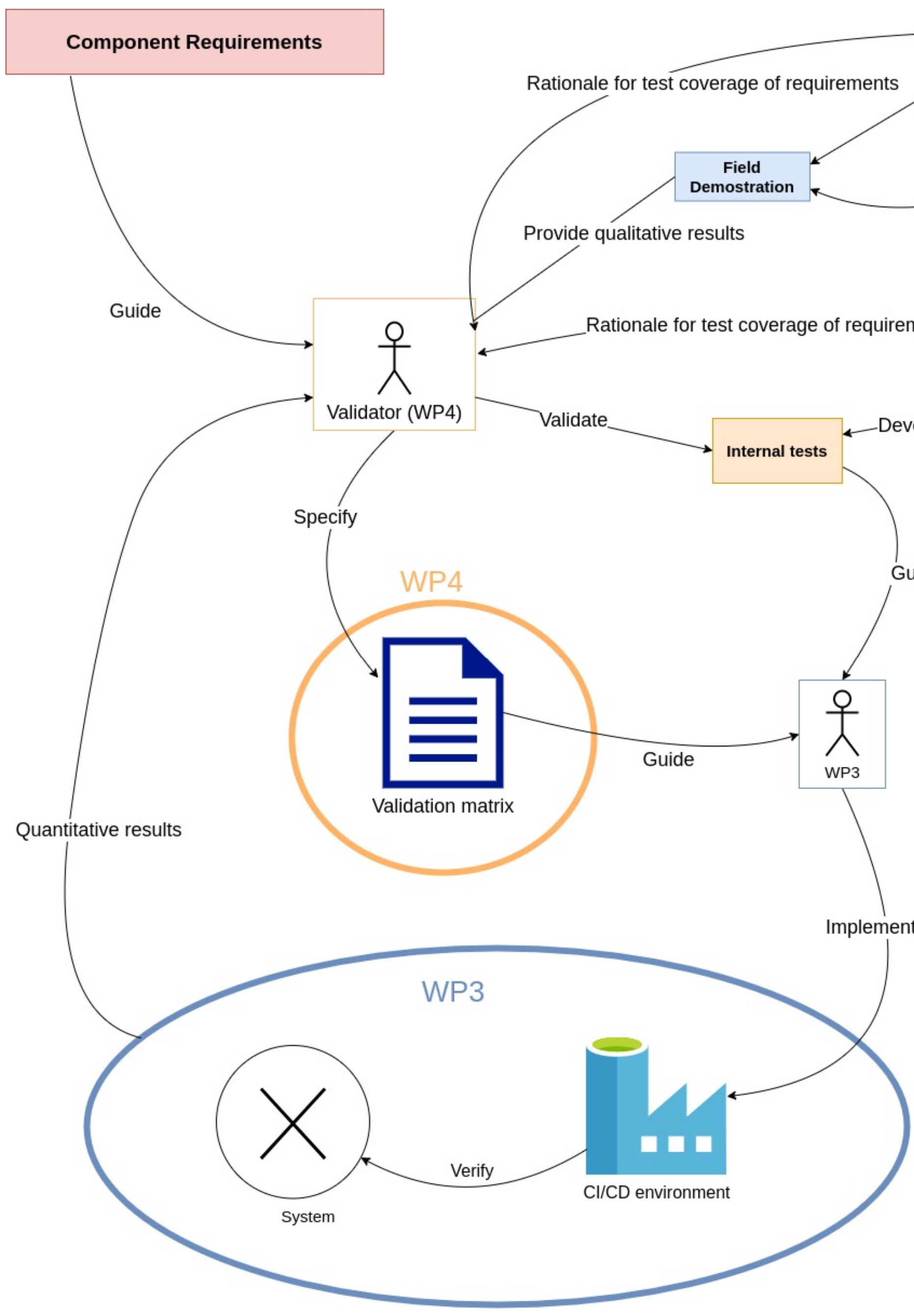


Figure 1: SOFIE validation process

Figure 1 provides an overview of the validation process. WP2 and WP3 define the tests used to validate, respectively, components and pilots and a rationale describing why these tests are feasible for the validation process. These tests are then handled by WP4 which analyses their acceptability and returns feedback to the test creators. When the tests are acceptable, WP4 creates the validation matrix and WP3, together with WP2 implements the tests in the validation environment. The validation environment returns quantitative results, which will be used by relevant work packages to understand the validation status of the requirements.

3.2 Architecture Validation

This section defines the validation matrix for the SOFIE architecture. This matrix is a table composed by the ID of the architecture requirements found in D2.4. For each requirement ID we define the approach used to test a requirement, which may be unit test, functional test, integration test, documentation or any other valid approach to validate a requirement. The table is then completed by a short description of the approach used in each requirement.

Table 9: SOFIE architecture validation matrix

ID	***Validation Process***		***Result***
RA01	*Requirement Description*	SOFIE architecture must define a clear separation between data management, control, and representation processes.	OK
	Test approach	Documentation	
	Test Description	Architecture is divided into multiple components that carry out the different processes independently from each other.	
	Test location	D2.4, Section 2.1, page 7	
RA02	*Requirement Description*	SOFIE architecture must be modular to enable different use cases and reuse of components.	OK
	Test approach	Documentation	
	Test Description	Architecture is divided into extendable components. SOFIE pilots are examples of how the components can be used for different use cases.	
	Test location	D2.4, Section 2.1, page 7 & Section 3, page 18-	
RA03	*Requirement Description*	The interfaces of the SOFIE components must be well-defined and fully documented.	OK
	Test approach	Documentation	
	Test Description	Component interfaces are described in the Framework documentation.	
	Test location	D2.5, Sections 3-8, page 9-	
RA04	*Requirement Description*	Transactions must be immutable and verifiable. Parties must not be able to modify existing transactions without other parties noticing it. Every party should be able to independently verify the validity of transactions.	OK
	Test approach	Functional test	
	Test Description	Event on one ledger automatically triggers the transfer of data/asset to another ledger. All information (initiation of transfer, acceptance of transfer, confirmation of acceptance) is stored on the ledgers, which makes the information immutable and verifiable.	
	Test location	Interledger: tests/system/test_interledger_ethereum.py	
RA05	*Requirement Description*	The system must provide auditability.	TBD
	Test approach	Functional test	
	Test Description	TBD	
	Test location	TBD	

RA06	*Requirement Description*	Support for transactions, where only authorized entities can participate. Minimal amount of information should be disclosed during authentication.	OK
	Test approach	Documentation	
	Test Description	Architecture (through the IAA component) can be configured to support any type of authorization server including servers supporting minimal disclosure of information.	
	Test location	D2.4, Section 4.2, page 31	
RA07	*Requirement Description*	All external and internal interfaces and communication links of the system must conform to the principle of least privilege.	OK
	Test approach	Documentation	
	Test Description	Architecture has been designed with the principle of least privilege.	
	Test location	D2.4, Section 4.3, page 31	
RA08	*Requirement Description*	The SOFIE architecture should be flexible and support different means of user authentication, including password-based, certification-based, and token-based.	OK
	Test approach	Documentation	
	Test Description	IAA component can be configured to support any type of authorization server including servers supporting different means of authentication.	
	Test location	D2.4, Section 4.2, page 31	

3.3 Component validation

Similarly to the architecture validation matrix, the component validation matrix is a table indexed by the ID of the architecture requirements found in D2.4. This table is identical to the architecture matrix in its form and purpose.

Table 10: SOFIE components validation matrix

ID	***Validation Process***		***Result***
Interledger			
RF01	*Requirement Description*	User interaction is not required for interledger operations.	OK
	Test approach	Functional test	
	Test Description	Event on one ledger automatically triggers the transfer of data/asset to another ledger	
	Test location	Interledger: tests/system/test_interledger_ethereum.py	
RF02	*Requirement Description*	There should be support for atomic interledger operations.	OK
	Test approach	Functional test	
	Test Description	Status of asset transfers is atomic, so that the asset can be accessible only in one ledger	
	Test location	Interledger: tests/system/test_interledger_ethereum, solidity/test/tokenTest (testing contract for gameToken)	

Identification, Authentication, and Authorization (IAA)			
RF03	*Requirement Description*	Resource owners must be able to delegate the authentication and authorization tasks for their resources.	OK
	Test approach	Documentation	
	Test Description	The IAA is configured with the public key of the preferred authorization server	
	Test location	IAA's repository documentation, "Configuration" chapter	
RF04	*Requirement Description*	The IAA component must provide users the capability to revoke authorizations.	TBD
	Test approach	Functional test	
	Test Description	A token is created, and it is logged in an ERC-721 smart contract. Then it is marked as revoked in the smart contract. IAA rejects the token.	
	Test location	TBD	
RF05	*Requirement Description*	The IAA component must allow individuals to control their personal information and digital identities (e.g., support self-sovereign identity technology).	OK
	Test approach	Functional test	
	Test Description	The test is configured with a valid DID and a valid VC. It interacts with indy_agent.py which generates a challenge. The test sends a report to the challenge.	
	Test location	IAA tests/test_indy_agent.py	
RF06	*Requirement Description*	The IAA component must support secure, tamper-proof, and verifiable logging of transactions and events.	OK
	Test approach	Functional test	
	Test Description	The test is configured with a valid token. It interacts with iaa_logger.py which records the token in a configured Ethereum smart contract. The test verifies the record.	
	Test location	IAA test/test_logging.py	
RF07	*Requirement Description*	The IAA component must support Role Based Access Control (RBAC).	OK
	Test approach	Documentation	
	Test Description	RBAC is implemented with the use of VCs. IAA can be used to verify a VC.	
	Test location	IAA's repository documentation, "Examples" chapter	
RF08	*Requirement Description*	Cryptographic algorithms used by SOFIE should be open-source, transparent, and as independent as possible of any particular architecture.	OK
	Test approach	Documentation	
	Test Description	IAA supports standardized cryptographic algorithms.	
	Test location	IAA's repository documentation, "Key technologies" chapter	
RF09	*Requirement Description*	SOFIE should support the execution of authorization and authentication functionality on devices with constrained processing, storage, battery, and network connectivity.	TBD
	Test approach	Functional test	
	Test Description	The test and IAA are pre-configured with a shared key. The test executes the token verification function of iaa.py.	
	Test location	TBD	

Privacy & Data Sovereignty (PDS)			
RF10	*Requirement Description*	SOFIE must follow the data minimization principle for personal data and only request or process what is necessary for the situation and purpose.	OK
	Test approach	Documentation	
	Test Description	PDS can be configured with a specific proof request	
	Test location	PDS's repository documentation, "Configuration" chapter	
RF11	*Requirement Description*	Processing of an individual's personal data is justified by a valid legal basis, e.g., a valid consent from the individual.	OK
	Test approach	Functional test	
	Test Description	The test is configured with a valid VC. The test invokes the VC verification, which generates a proof request. The test generates the proof and outputs the verification result.	
	Test location	PDS tests/test_indy_agent.py	
RF12	*Requirement Description*	Consent to process personal data must be revocable at any time.	OK
	Test approach	Documentation	
	Test Description	The documentation described how to set an expiration time on a VC	
	Test location	PDS's repository documentation, "Examples" chapter	
RF13	*Requirement Description*	SOFIE must allow organisations and actors to manage (create, update, delete) their own data privacy policies.	OK
	Test approach	Documentation	
	Test Description	PDS can be configured with arbitrary VC schemas.	
	Test location	PDS's repository documentation, "Configuration" chapter	
RF14	*Requirement Description*	SOFIE should support user privacy even when aggregate statistics are made public (e.g., using differential privacy mechanisms).	TBD
	Test approach	TBD	
	Test Description	TBD	
	Test location	TBD	
Semantic Representation (SR)			
RF15	*Requirement Description*	SOFIE must define an IoT things description model based on well-known standards (e.g. W3C standards).	OK
	Test approach	Functional test	
	Test Description	The test shows that only objects conforming to the component schema (W3C standards) are validated.	
	Test location	SR: tests/test_validation.py	
RF16	*Requirement Description*	SOFIE must implement standardised metadata and data representation formats and support various data modalities.	OK
	Test approach	Documentation	
	Test Description	The component uses JSON objects.	
	Test location	SR's repository documentation, "Main decision" chapter	
RF17	*Requirement Description*	The SR model of the system must be open and extensible by third parties (e.g., support the extension of the existing knowledge base and associations by extracting supplementary triples from RDF documents).	TBD
	Test approach	TBD	

	Test Description	TBD	
	Test location	TBD	
RF18	*Requirement Description*	SOFIE must provide service discovery and resources selection processes based on multiple criteria over the features, associations, and interaction patterns of integrated resources.	TBD
	Test approach	TBD	
	Test Description	TBD	
	Test location	TBD	
RF19	*Requirement Description*	SOFIE should support the semantic update and enhancement of resources' descriptions and associations in a dynamic way.	TBD
	Test approach	Functional tests	
	Test Description	Same SR tests will pass using different schema implementation	
	Test location	TBD	
Marketplace (MP)			
RF20	*Requirement Description*	The marketplace must log the configuration of all trading actions (including offers, bids, parameters of resources, transactions etc.).	OK
	Test approach	Functional test	
	Test Description	The test sets up an auction, accepts bids, and decides which offer wins - and verifies all the related information is stored on the ledger.	
	Test location	Marketplace: solidity/test/flowermarketplace	
RF21	*Requirement Description*	The marketplace must provide actors the capability to post/claim offers and sell/negotiate/exchange/buy resources and digital objects.	OK
	Test approach	Unit tests	
	Test Description	The test sets up an auction, accepts bids, and decides which offer wins - and verifies all the related information is stored on the ledger.	
	Test location	Marketplace: solidity/test/flowermarketplace	
RF22	*Requirement Description*	The marketplace must support transparent trading of resources, i.e. the bids/offers matching process and the payments must be transparent.	OK
	Test approach	Functional test	
	Test Description	The test sets up an auction, accepts bids, and decides which offer wins - and verifies all the related information is stored on the ledger.	
	Test location	Marketplace: solidity/test/flowermarketplace	
RF23	*Requirement Description*	The marketplace must provide evidence once trades have been completed and resources have been properly delivered to the buyers.	TBD
	Test approach	Functional test	
	Test Description	The transaction determining the winning bid is logged on the distributed ledger. Evidence of the delivery of resources must also be logged on the distributed ledger by the winner and seller, after which the evidence can be verified.	
	Test location	TBD	
RF24	*Requirement Description*	The marketplace should allow integration of payment technologies.	TBD
	Test approach	Documentation	
	Test Description	The marketplace component provides interfaces for integrating payment solutions; an example will be provided.	
	Test location	TBD	

3.4 Pilot validation

The pilot validation matrix is a table indexed by the ID of the architecture requirements found in D5.2. Its structure and purpose are similar to the architecture and component validation matrices. Pilot validation will be reported in deliverables D5.3 and D5.4.

3.4.1 FSC pilot

Table 11: FSC validation matrix

ID	Validation Process		Result
REQ_FSC0.1	*Requirement Description*	The services must be provided (to the actors) through the same web application.	TBD
	Test approach	Field test	
	Test Description	Each registered actor of any type (e.g., producer, transporter, warehouse, supermarket employee) can access and perform all the services provided by the FSC web application based on its role.	
	Test location	This requirement is tested as part of the FSC_TC02, as defined in D5.1.	
REQ_FSC0.2	*Requirement Description*	The services must be accessible (by the actors) under a Role-based Access Control (RBAC) policy.	TBD
	Test approach	Field test	
	Test Description	Each registered actor of any type (e.g. producer, transporter, warehouse, supermarket employee) can access and perform all the services provided by the FSC web application based on its role. The actors have already registered on the pilot platform. Roles for the actors are granted by the Keycloak server (which is a component of the platform) during their registration.	
	Test location	This requirement is tested as part of the FSC_TC02, as defined in D5.1.	
REQ_FSC0.3	*Requirement Description*	Each actor must be identified in a unique way	TBD
	Test approach	Unit test	
	Test Description	Authorization server is configured so each registered actor is bound to a unique ID. One of the attributes in an actor's profile (e.g., name) is used as a key to avoid duplicate IDs for the same actor.	
	Test location	TBD	
REQ_FSC0.4	*Requirement Description*	Each federated IoT environment must have a unique identifier in the system architecture.	TBD
	Test approach	Unit test	
	Test Description	Each federated IoT platform is bound to a unique ID. Federated platforms register themselves by using an Ethereum client such as Geth to create accounts by using built-in encryption policies. The smart contract executed in the private ledger (consortium ledger) implements multiple checks to verify the identities of the IoT platforms requesting transactions. The test will compare datasets which are sent by each federated platform for a specific period to the records in the consortium ledger to decide whether each IoT platform is bound to a unique ID or not.	
	Test location	TBD	
REQ_FSC0.5	*Requirement Description*	Authentication and access control logic must be applied to common storage resources.	

	Test approach	Documentation	TBD
	Test Description	An authorization and access management server had been integrated to enable the actors' registration in the supervisor data management layer and establish role-based accessibility to the provided services (D5.2, Chapter 3.3.2, page 38). The test will verify that each registered actor can make transactions to the private ledger based on its role (and the defined use cases).	
	Test location	This requirement is tested as part of the FSC_TC06, as defined in D5.1.	
REQ_FSC1.1	*Requirement Description*	Registration of a crop must be timestamped.	TBD
	Test approach	Integration test	
	Test Description	Data and metadata provided by the actors through the FSC web application are recorded in DLTs. The payload of any transaction is verified.	
	Test location	This requirement is tested as part of the FSC_TC06, as defined in D5.1.	
REQ_FSC2.1	*Requirement Description*	The QR code that summarizes product history must include farm location, harvesting date, used fertilizers (dates), and the type of the product (from the perspective of the farming system)	TBD
	Test approach	Functional test	
	Test Description	Readability of all included information in QR codes is confirmed.	
	Test location	This requirement is tested as part of the FSC_TC08, as defined in D5.1.	
REQ_FSC3.1	*Requirement Description*	Handovers must be recorded in an immutable way where all federated IoT environments must have access.	TBD
	Test approach	Integration test	
	Test Description	Data and metadata provided by the actors through the FSC web application are recorded in DLTs. The payload of any transaction is verified.	
	Test location	This requirement is tested as part of the FSC_TC06, as defined in D5.1.	
REQ_FSC3.2	*Requirement Description*	The boxes could be sealed upon the delivery to the transportation company (from the producers).	TBD
	Test approach	TBD	
	Test Description	TBD	
	Test location	TBD	
REQ_FSC4.1	*Requirement Description*	Upon delivery to the WH employee, boxes could be unsealed by the TR employee.	TBD
	Test approach	TBD	
	Test Description	TBD	
	Test location	TBD	

REQ_FSC5.1	*Requirement Description*	Each box must have a unique RFID tag identifier.	TBD
	Test approach	Functional test	
	Test Description	Test that box reuse is possible (after its release) and that registration of a box with an ID that is already used by another box is impossible (box unique identifier).	
	Test location	This requirement is tested as part of the FSC_TC05, as defined in D5.1.	
REQ_FSC5.2	*Requirement Description*	Boxes must be considered as things of the transportation IoT platform.	TBD
	Test approach	Functional test	
	Test Description	An RFID tag is attached to each box. The test will verify that the RFID reader detects all tags which are placed within its range at any moment.	
	Test location	This requirement is tested as part of the FSC_TC06, as defined in D5.1.	
REQ_FSC5.3	*Requirement Description*	Box registration in the supply chain must define also the producer from whom it will be used.	TBD
	Test approach	Integration test	
	Test Description	The payload of the transaction which corresponds to the specific use case (FSC_UC5 “register session”) is verified to include also the ID of the farmer who will use the boxes.	
	Test location	This requirement is tested as part of the FSC_TC06, as defined in D5.1.	
REQ_FSC5.4	*Requirement Description*	Registration of a box must be timestamped.	TBD
	Test approach	Integration test	
	Test Description	The payload of the transaction which corresponds to the specific use case (FSC_UC5 “register session”) is verified to also include a timestamp.	
	Test location	This requirement is tested as part of the FSC_TC06, as defined in D5.1.	
REQ_FSC6.1	*Requirement Description*	Transportation trucks must have internet connection to communicate and exchange data with the transportation IoT platform.	TBD
	Test approach	Field test	
	Test Description	The SOFIE platform receives data from the transportation GW deployed in the truck i) as the vehicle moves, and ii) as the vehicle engine is turned off.	
	Test location	This requirement is tested as part of the FSC_TC05, as defined in D5.1.	
REQ_FSC6.2	*Requirement Description*	A TR employee (driver) must be able to use different transportation trucks on different occasions.	TBD
	Test approach	Functional test	
	Test Description	By using the FSC web application, the TR employees can select any of the available trucks to transport boxes between two sites.	
	Test location	This requirement is tested as part of the FSC_TC05, as defined in D5.1.	
REQ_FSC7.1	*Requirement Description*	Measurements from IoT devices are stored locally in the corresponding IoT platform.	TBD
	Test approach	Integration test	
	Test Description	Measurements from each deployed sensing device are collected by the corresponding IoT platform and they are properly stored in its database system.	
	Test location	This requirement is tested as part of the FSC_TC01, as defined in D5.1.	

REQ_FSC8.1	*Requirement Description*	Upon delivery to the SM employee, boxes could be unsealed by the TR employee.	TBD
	Test approach	TBD	
	Test Description	TBD	
	Test location	TBD	
REQ_FSC9.1	*Requirement Description*	The temperature within each storage room of the WH must be continually monitored.	TBD
	Test approach	Integration test	
	Test Description	The test will verify that temperature measurements from each deployed sensing device are collected by the corresponding IoT platform	
	Test location	This requirement is tested as part of the FSC_TC01, as defined in D5.1.	
REQ_FSC9.2	*Requirement Description*	In the WH, a notification appears in the monitoring service of the Aberon IoT platform each time a predefined temperature range is violated.	TBD
	Test approach	Integration test	
	Test Description	The test will create datasets that include some values out of the predefined temperature domain and verify that corresponding notifications are created.	
	Test location	This requirement is tested as part of the FSC_TC01, as defined in D5.1.	
REQ_FSC10.1	*Requirement Description*	The (unreleased) boxes in the WH must contain either raw or packetized products.	TBD
	Test approach	Documentation	
	Test Description	This requirement has been merged into the workflow (action in the physical space) that accompanies the use of services. Not a technical requirement, no test is applied	
	Test location	N/A	
REQ_FSC11.1	*Requirement Description*	QR codes must include data collected from the federated IoT environments, as well as provided by the actors through the FSC web application	TBD
	Test approach	Functional test	
	Test Description	Creation of QR codes by using the FSC web application.	
	Test location	This requirement is tested as part of the FSC_TC08, as defined in D5.1.	
REQ_FSC11.2	*Requirement Description*	The same QR label must be attached to every packet containing grapes which were transferred into the same box.	TBD
	Test approach	Documentation	
	Test Description	This requirement has been merged into the workflow (action in the physical space) that accompanies the use of services. Not a technical requirement, no test is applied	
	Test location	N/A	

REQ_FSC11.3	*Requirement Description*	Labelling of products must be based on a common vocabulary for the food supply domain that maximizes reuse of data and acceptance by the customers.	TBD
	Test approach	Documentation	
	Test Description	TBD	
	Test location	TBD	
REQ_FSC11.4	*Requirement Description*	The QR codes must be self-contained, so internet connection is not needed to read their content.	TBD
	Test approach	Functional test	
	Test Description	Multiple QR codes are scanned and it is verified that they include product information from all segments of the supply chain	
	Test location	This requirement is tested as part of the FSC_TC08, as defined in D5.1.	
REQ_FSC11.5	*Requirement Description*	The QR codes must contain product information relate to all the segments of the chain.	TBD
	Test approach	Functional test	
	Test Description	Multiple QR codes are scanned and it is verified that they include product information from all segments of the supply chain	
	Test location	This requirement is tested as part of the FSC_TC08, as defined in D5.1.	
REQ_FSC12.1	*Requirement Description*	Boxes must be able to be re-used in the future (to carry other products) after they have been released of the current transfer.	TBD
	Test approach	Functional test	
	Test Description	Test that box reuse is possible (after its release) and that registration of a box with an ID that is already used by another box is impossible (box unique identifier).	
	Test location	This requirement is tested as part of the FSC_TC03, as defined in D5.1.	
REQ_FSC13.1	*Requirement Description*	QR labels must be accessible by everyone by using a smartphone device.	TBD
	Test approach	Functional test	
	Test Description	A QR code which is created by the supermarket employee using the FSC web application can be read offline by using different smartphones devices. Readability of all included information is confirmed.	
	Test location	This requirement is tested as part of the FSC_TC08, as defined in D5.1.	
REQ_FSC14.1	*Requirement Description*	In case of an audit, requested organizations must be able to provide proof of their claims about the historic data of assets which are stored locally.	TBD
	Test approach	Functional test	
	Test Description	The test verifies that all measurements of interest can be retrieved by the API of the corresponding IoT platform	
	Test location	This requirement is tested as part of the FSC_TC09, as defined in D5.1.	
REQ_FSC14.2	*Requirement Description*	Transfer of responsibility over boxes (assets) must be timestamped.	TBD
	Test approach	Functional test	
	Test Description	The payload of transactions is verified to include correct timestamps.	
	Test location	This requirement is tested as part of the FSC_TC06, as defined in D5.1.	

REQ_FSC14.3	*Requirement Description*	A transaction must be confirmed by both transacting parties.	TBD
	Test approach	Functional test	
	Test Description	TBD	
	Test location	TBD	
REQ_FSC14.4	*Requirement Description*	Both parties of a transaction must be able to access the details of the transaction at any time.	TBD
	Test approach	Functional test	
	Test Description	Metadata related to an actor's activity (in the FSC app.) is accessible by that actor at any time and is invisible to any other actor.	
	Test location	This requirement is tested as part of the FSC_TC07, as defined in D5.1.	

3.4.2 DEDE pilot

Table 12: DEDE validation matrix

ID	Validation Process		Result
REQ_DEDE 1.1	*Requirement Description*	Data owner can access info about his data, full visibility of data use.	TBD
	Test approach	Functional test	
	Test Description	Data owner will have an overview about the usage of the metering data.	
	Test location	TBD	
REQ_DEDE 1.2	*Requirement Description*	Each actor must be identified.	TBD
	Test approach	Functional test	
	Test Description	All actors in the pilot system will have unique identifiers (DIDs).	
	Test location	TBD	
REQ_DEDE 2.1	*Requirement Description*	Owner must be able to decide who gets access to his/her data.	TBD
	Test approach	Functional test	
	Test Description	TBD	
	Test location	TBD	
REQ_DEDE 2.2	*Requirement Description*	All user info handling must be GDPR compliant.	TBD
	Test approach	Documentation	
	Test Description	Data handling in the system will be GDPR compliant	
	Test location	D5.2, Chapter 4.2.2.1, page 55	
REQ_DEDE 2.3	*Requirement Description*	Data handover must be registered and proved at every transaction.	TBD
	Test approach	Functional test	

	Test Description	Architecture supports transactions involving ledgers to achieve immutability and transparency.	
	Test location	TBD	
REQ_ DEDE 2.4	*Requirement Description*	Service provider must be able to define the energy consumption data parameters.	TBD
	Test approach	Integration test	
	Test Description	Service provider has full control over the offered services.	
	Test location	TBD	
REQ_ DEDE 2.5	*Requirement Description*	Service provider must be able to download the energy consumption data.	TBD
	Test approach	Integration test	
	Test Description	When secure connection is established and credentials exchanged, consumption data can be fetched and validated.	
	Test location	TBD	
REQ_ DEDE 2.6	*Requirement Description*	Authentication toolkit for all actors (eIDAS compliant).	TBD
	Test approach	Documentation	
	Test Description	Authentication with existing approaches (e.g., eIDAS) will be supported	
	Test location	D5.2, Chapter 4.2.2.1, page 55	
REQ_ DEDE 2.7	*Requirement Description*	Processes monitoring the system must be logged, stored (in local environment)	TBD
	Test approach	Functional test	
	Test Description	Adapter audit log	
	Test location	TBD	
REQ_ DEDE 5.1	*Requirement Description*	Service provider must be able to get proof of receiving the energy consumption data	TBD
	Test approach	Functional test	
	Test Description	Proofs can be downloaded and verified	
	Test location	TBD	
REQ_ DEDE 5.2	*Requirement Description*	System logs integrity must be 3rd party verifiable (auditor)	TBD
	Test approach	Functional test	
	Test Description	3rd parties can verify the interactions	
	Test location	TBD	

3.4.3 DEFM pilot

Table 13: DEFM validation matrix

ID	Validation Process		Result
	Requirement Description	DSO shall be able to forecast of electricity production/consumption	TBD

REQ_ DEF M1.1	*Test approach*	Integration test	
	Test Description	Described in D5.1 v2.0 as DEFM_TC01 and DEFM_TC03	
	Test location	TBD	
REQ_ DEF M1.2	*Requirement Description*	DSO shall be able to check the load and production forecasting of the whole distribution grid	TBD
	Test approach	Integration test	
	Test Description	Described in D5.1 v2.0 as DEFM_TC01 and DEFM_TC03	
	Test location	TBD	
REQ_ DEF M1.3	*Requirement Description*	DSO shall be able to forecast of electricity production / consumption at the grid level	TBD
	Test approach	Integration test	
	Test Description	Described in D5.1 v2.0 as DEFM_TC01 and DEFM_TC03	
	Test location	TBD	
REQ_ DEF M1.4	*Requirement Description*	DSO shall be able to shave peaks of energy produced locally the day after so that instability of the system, overvoltage on the feeder, protection discoordination, increased fault currents, and incorrect operation of equipment could be avoided	TBD
	Test approach	Integration test	
	Test Description	Described in D5.1 v2.0 as DEFM_TC01 and DEFM_TC03	
	Test location	TBD	
REQ_ DEF M1.5	*Requirement Description*	DSO shall be able to estimate the energy flexibility availability; assess flexibility availability by using available historical data.	TBD
	Test approach	Integration test	
	Test Description	Described in D5.1 v2.0 as DEFM_TC01 and DEFM_TC03	
	Test location	TBD	
REQ_ DEF M1.6	*Requirement Description*	DSO shall be able to forecast that system indicates a potential reverse power flow to be mitigated and DSO system is connected to the flexibility marketplace. The DSO system is connected to the flexibility marketplace.	TBD
	Test approach	Integration test	
	Test Description	Described in D5.1 v2.0 as DEFM_TC01 and DEFM_TC03	
	Test location	TBD	
REQ_ DEF M2.1	*Requirement Description*	When the Fleet Manager obtains the responsibility to provide the flexibility required by the DSO, a micro contract between the Fleet Manager and the DSO is executed.	TBD
	Test approach	Integration test	
	Test Description	Described in D5.1 v2.0 as DEFM_TC02	
	Test location	TBD	
REQ_ DEF M2.2	*Requirement Description*	When the Fleet Manager obtains the responsibility to provide the flexibility required by the DSO and EV users not belonging to the fleet manager EV fleet are involved in the DR campaign, a micro contract between the Fleet Manager and the EV user is executed.	TBD
	Test approach	Integration test	

	Test Description	Described in D5.1 v2.0 as DEFM_TC01	
	Test location	TBD	
REQ_ DEF M4.1	*Requirement Description*	With the objective of performing Demand Response (DR) campaigns, it is necessary that the management systems of electric vehicles and charging stations communicate with each other, so that it is possible to verify in real time the interaction between the two systems.	TBD
	Test approach	TBD	
	Test Description	TBD	
	Test location	TBD	
REQ_ DEF M4.2	*Requirement Description*	To provide DSO flexibility in an efficient way, the data of electric vehicles and charging stations must be collected in real time (or very close to real time). Data coming from EVSEs and the EVs should be consistent, reliable, transparent and accessible to the partners. Furthermore, to perform optimized DR campaigns it is necessary to constantly calculate EV load forecasting to estimate the amount of energy that electric vehicles could consume to meet the DSO's flexibility demand.	TBD
	Test approach	Functional test	
	Test Description	When secure connection is established, data from electric vehicles and charging stations are collected	
	Test location	TBD	
REQ_ DEF M4.3	*Requirement Description*	It is necessary that the data of electric vehicles and charging stations are stored so that they can then be reprocessed, producing charts that show the effectiveness for the purposes of the DSO of DR campaigns performed during the trial.	TBD
	Test approach	Integration test	
	Test Description	Data collected from electric vehicles and charging stations are stored in the fleet manager server	
	Test location	TBD	
REQ_ DEF M4.4	*Requirement Description*	As there will be more than one charging station on the pilot site, each individual charging station must have its own unique identifier.	TBD
	Test approach	Unit test	
	Test Description	All charging stations in the pilot system will have unique identifiers (IDs)	
	Test location	TBD	
REQ_ DEF M4.5	*Requirement Description*	As there will be more than one electric vehicle on the pilot site, each individual electric vehicle must have its own unique identifier.	TBD
	Test approach	Unit test	
	Test Description	All electric vehicles in the pilot system will have unique identifiers (IDs)	
	Test location	TBD	
REQ_ DEF M4.6	*Requirement Description*	To allow the EV user to realize the available charging stations and the fees associated with them, a web platform is required.	TBD
	Test approach	Integration test	
	Test Description	Fleet manager and EV users can authenticate on the web platform and check the electric vehicles and charging stations real time status and historical data	
	Test location	TBD	
REQ_ DEF M4.7	*Requirement Description*	Both charging stations and electric vehicles must be connected to the internet in order to send data.	TBD
	Test approach	Integration test	

	Test Description	Charging stations and electric vehicles must be connected to internet to communicate with the fleet manager server	
	Test location	TBD	
REQ_ DEF M5.1	*Requirement Description*	The charging station must be remotely controlled to start/stop charging sessions and to modulate the power output.	TBD
	Test approach	Unit test	
	Test Description	TBD	
	Test location	TBD	
REQ_ DEF M7.1	*Requirement Description*	DSO shall be able to constantly calculate building consumption forecasting, PV production forecasting and manage batteries to estimate the amount of energy demand at ASM substation. Forecasting will be calculated periodically (every day). Need to reduce undesired reverse power flows	TBD
	Test approach	Integration test	
	Test Description	From the DSO local network, load forecast for the two network zones can be fetched	
	Test location	TBD	
REQ_ DEF M8.1	*Requirement Description*	When the Fleet Manager obtains the responsibility to provide the flexibility required by the DSO, a micro contract between the Fleet Manager and the Retailer is executed for the energy supply to charge electric vehicles	TBD
	Test approach	Integration test	
	Test Description	The marketplace backend exposes the APIs needed by the actors for interacting with the system, participating with requests and offers.	
	Test location	TBD	

3.4.4 MRMG pilot

Table 14: MRMG validation matrix

ID	***Validation Process***		**Result**
REQ _ MRM G0.1	*Requirement Description*	Each person interacting with the game should have a unique identifier.	TBD
	Test approach	Unit test	
	Test Description	The test passes if all player IDs are different.	
	Test location	TBD	
REQ _ MRM G1.1	*Requirement Description*	Game challenges are accessible using the Android application	TBD
	Test approach	Field test	
	Test Description	In the test, the user opens the Scavenger Hunt game application and enters the Nearby Challenges tab. The user should see a list of (uncompleted) challenges that start in GPS coordinates that are within a set radius from the user. The requirement is met if the nearby challenges that exist on the backend are indeed visible in the Nearby Challenges tab.	
	Test location	TBD	
REQ _ MRM G1.2	*Requirement Description*	Players can join any nearby challenge from the game app.	TBD
	Test approach	Field test	
	Test Description	The requirement is met if a challenge is added to the list of the player's current challenges, after the player presses the Start button in the client.	
	Test location	TBD	

REQ _ MRM G1.3	*Requirement Description*	Each challenge should have a unique identifier	TBD
	Test approach	Unit test	
	Test Description	The test passes if the IDs of all Scavenger Hunt challenges are different.	
	Test location	TBD	
REQ _ MRM G1.4	*Requirement Description*	Time should be recorded for each player, starting after joining the challenge till the player completes it.	TBD
	Test approach	Field test	
	Test Description	The requirement is met if, after a user plays the challenge, the completed challenge's start and end time fields are populated.	
	Test location	TBD	
REQ _ MRM G1.5	*Requirement Description*	Players should receive unique tasks when near the IoT beacons based on their challenge.	TBD
	Test approach	Field test	
	Test Description	The requirement is met if a user standing next to a BLE beacon receives a task in the mobile application.	
	Test location	TBD	
REQ _ MRM G1.6	*Requirement Description*	Players should be able to skip any task and receive location of next IoT beacon using the In-App tokens.	TBD
	Test approach	Field test	
	Test Description	If a player has Star items in-game, they can use one start to skip a task. The requirement is met if, when presented with a task and using a star, the current task auto-completes and the user receives the clue to the next beacon.	
	Test location	TBD	
REQ _ MRM G1.7	*Requirement Description*	Players can buy In-App tokens using in-game currency	TBD
	Test approach	Field test	
	Test Description	The requirement is met if the player can spend in-game coins in the application to buy Gem and Star tokens - increasing Gem and Star amounts in possession and decreasing Coins in possession.	
	Test location	TBD	
REQ _ MRM G1.8	*Requirement Description*	System should automatically calculate rewards after player has completed a challenge	TBD
	Test approach	Field test	
	Test Description	After a player completes a challenge, the requirement is met if the player sees rewards in the client application.	
	Test location	TBD	
REQ _ MRM G2.2	*Requirement Description*	System should automatically add the rewards to the player's account after the challenge ends.	TBD
	Test approach	Functional test	
	Test Description	The test passes if, after the reward transaction, the amount of coins in the escrow has decreased and the amount of coins in the player's account has increased by the reward amount.	
	Test location	TBD	
REQ _ MRM G3.1	*Requirement Description*	Player should be given the option to view advertisements while playing a challenge.	N/A
	Test approach	N/A	
	Test Description	This requirement has been replaced with REQ_MRMG 9	

	Test location	N/A	
REQ _ MRM G3.2	*Requirement Description*	Player should receive tokens for viewing the advertisement.	N/A
	Test approach	N/A	
	Test Description	This requirement has been replaced with REQ_MRMG 9	
	Test location	N/A	
REQ _ MRM G3.3	*Requirement Description*	Every ad viewability data should be recorded as a transaction on the blockchain.	TBD
	Test approach	Functional test	
	Test Description	This requirement has been replaced with REQ_MRMG 9	
	Test location	TBD	
REQ _ MRM G4.1	*Requirement Description*	Players can buy and sell Blockmoji assets on the blockchain	TBD
	Test approach	Field tests	
	Test Description	The requirement is met if players can buy and sell Blockmoji items on the blockchain.	
	Test location	TBD	
REQ _ MRM G4.2	*Requirement Description*	Every asset traded on the platform should be recorded as a transaction on the blockchain.	TBD
	Test approach	Functional test	
	Test Description	The test passes if, after a Blockmoji trading transaction has occurred, that transaction can be read from the blockchain.	
	Test location	TBD	
REQ _ MRM G5.1	*Requirement Description*	Web application for designing new challenges and uploading advertisements.	N/A
	Test approach	N/A	
	Test Description	This requirement has been replaced with REQ_MRMG 9	
	Test location	N/A	
REQ _ MRM G5.2	*Requirement Description*	Access control to the web services based on the role of the user.	N/A
	Test approach	N/A	
	Test Description	This requirement has been replaced with REQ_MRMG 9	
	Test location	N/A	
REQ _ MRM G7.1	*Requirement Description*	Blockmoji item rewards be can offered to players through challenges	TBD
	Test approach	Field test	
	Test Description	If a challenge offers a Blockmoji item reward, the player should see it in their mobile application reward screen after completing the challenge.	
	Test location	TBD	
REQ _ MRM G7.2	*Requirement Description*	Blockmoji rewards should be added and recorded on the blockchain.	TBD
	Test approach	Functional test	
	Test Description	The requirement is met if, after a player completes a challenge that awards a Blockmoji item, the receiving of the item can be read as a transaction on the blockchain.	
	Test location	TBD	

REQ_MRMG8.1	*Requirement Description*	Ads manager should publish any ad video using the web application	N/A
	Test approach	N/A	
	Test Description	This requirement has been replaced with REQ_MRMG 9	
	Test location	N/A	
REQ_New REQ_MRMG9.1	*Requirement Description*	Every user that signs in with Decent ID should have a unique decentralized ID with the connection.	TBD
	Test approach	Unit test	
	Test Description	The test passes if all connection DIDs of a user are different.	
	Test location	TBD	
New REQ_MRMG9.2	*Requirement Description*	Companies send pieces of the user's ad profile data to the user as credentials	TBD
	Test approach	Functional test	
	Test Description	The test validates that ad profile data is received by the connection if the user has allowed the request.	
	Test location	TBD	
New REQ_MRMG9.3	*Requirement Description*	Companies request access to the user's ad profile credentials, and user can accept them	TBD
	Test approach	Field test	
	Test Description	The requirement is met if, after a service requests for credentials, the user sees a prompt in the mobile application to accept the request.	
	Test location	TBD	
New REQ_MRMG9.4	*Requirement Description*	User can revoke connections' access to credentials by resetting the decentralized ID for the connection	TBD
	Test approach	Field test	
	Test Description	The requirement is met if the player can unlink connections from the mobile application.	
	Test location	TBD	

4 Component evaluation

This section presents evaluation results for SOFIE's framework components. The results presented in this section focus on the internal (basic) functionality of the components. Evaluation results that consider the functionality of the components within the pilots are presented in Section 6. Furthermore, in this section we also present the requirements of each component, defined in deliverable D2.4, and the related evaluation/emulation scenarios and/or how these requirements are met. The Provisioning and Discovery (PaD) component will be used and demonstrated in the MRMG pilot, where it is used to discover and provision new IoT devices, such as BLE beacons. This component is not evaluated in the emulation scenarios of the current deliverable, but will be considered in the corresponding MRMG pilot validation work. Finally, we note that the evaluation results presented in this and in the next two sections consider simulation and emulation scenarios that have been implemented in corresponding testbeds. These scenarios are distinct from the functional and documentation tests and the pilot integration and field tests identified in Section 3, since the evaluation work in WP4 has a wider scope than the pilots, seeking to evaluate many potential alternatives going beyond what is possible within the pilots.

4.1 Interledger

The main purpose of the SOFIE interledger component is to enable transactions between actors and devices belonging to different (isolated) IoT platforms or silos. Each IoT silo either utilizes or is connected to one or more DLTs. The interledger component then enables interaction between these DLTs. The first system evaluation in deliverable D4.3 focused on evaluating the transaction cost (gas) and delay in the case where a single (public) blockchain is used and in the case where two blockchains, a public and a private one, are used. The second system evaluation contained in the current deliverable focuses on evaluating the interledger functionality, which involves transferring information from one ledger to another, and the end-to-end delay when a public ledger (Ethereum) is interconnected with Hyperledger Fabric, which is a permissioned ledger. Note that the transaction cost for the public Ethereum ledger in this scenario is the same as in the scenario where the public Ethereum ledger is interconnected with some other private or permissioned DLT. Hence, the results for the transaction cost presented in the previous deliverable D4.3 continue to apply when public Ethereum is interconnected with Hyperledger Fabric.

The interledger component can utilize different mechanisms depending on the specific scenario and its requirements. For example, interactions between a public and a permissioned ledger can use hashed time-lock contracts to cryptographically link transactions and events on the two ledgers. In such a scenario, the public ledger can record payments while the permissioned ledger can record authorization transactions and events. Alternatively, hashes of records stored on the permissioned ledger can be periodically recorded on the public ledger in order to provide a timestamped anchoring point, exploiting the wide-scale decentralized trust provided by the public ledger. This functionality is considered in the FSC and DEDE scenarios in Section 6. Finally, interactions between a public or permissioned ledger and a ledger storing DID documents can focus on the resolution of DIDs to DID documents. The interledger functionality can be implemented in different entities, which include the entities that are interacting, a third party, or multiple third parties. In the latter case, some coordination between the entities may be necessary. A detailed survey of interledger approaches is contained in [Sir+19d]. Below we provide more details on the hash-lock and time-lock mechanisms, which are utilized by the interledger component evaluated in this section.

A **hash-lock** is a cryptographic lock that can be unlocked by revealing a secret whose hash is equal to the lock's value, *h*. Unlocking a hash-lock can be one of the conditions for performing a transaction or for executing a smart contract function. On a single blockchain, a hash-lock can be linked to an off-chain capability, e.g., message decryption, if the hash-lock secret is the secret key that can decrypt the message. Hash-locks can be used on two or more blockchains, which support the same hash function, to link a transaction on one chain to a transaction on the other chain: if the two transactions have hash-locks with the same value, then unlocking one hash-lock would reveal the secret that unlocks the other; hence, the two transactions are cryptographically linked through a dependence relation. More generally, hash-locks combined with AND/OR logic operators can implement elaborate dependencies involving transactions on multiple chains.

Time-locks are blockchain locks that can be unlocked only after an interval has elapsed. This interval can be measured in absolute time or in the number of blocks mined after a specific block. One usage of time-locks are refunds: a user (payer) can make a deposit to a smart contract address. The smart contract can have a function, which typically also includes a hash-lock, for a second user to transfer the deposit to another account (the payee's account). However, if the second user never calls this function, then the first user's deposit could be locked indefinitely in the smart contract's account. To avoid this, the smart contract can also include a refund function that allows the first user to transfer the amount he/she deposited back to his/her account; however, this function can be called only after some time interval, which is the interval in which the second user must transfer the deposit from the smart contract account to the payee's account.

4.1.1 Experiment setup

For the evaluation of we deployed an instance of the component that implements the interledger functionality and is connected to a Hyperledger Fabric permissioned blockchain and to the Rinkeby public Ethereum testnet. The version of Hyperledger Fabric used in the experiments was 1.4. A key feature of Hyperledger Fabric permissioned ledger is that smart contracts, called chaincode, are executed in a distributed manner following an execute-order-commit flow pattern instead of the common order-execute-commit pattern. The chain-code is submitted by clients to special nodes, called endorsing nodes, that execute (or simulate) the chaincode. This execution identifies the read-write dependencies and can involve checking that a transaction conforms to business rules identified in the chaincode. After a transaction has received a sufficient number of endorsements, it is sent to ordering nodes that order and group the transactions in blocks. Finally, the transactions are validated to ensure the consistency of the read-write dependencies. After the transactions are validated, they are committed, i.e., the ledger is updated.

We consider the authorization use case for constrained IoT resources presented in Section 5.1, where users deposit some funds in order to receive a decryption key for accessing an IoT resource. The payment transaction is performed on the Ethereum blockchain (Rinkeby Ethereum public testnet) while the authorization transaction is performed on the Hyperledger Fabric permissioned blockchain. The transactions on the two ledgers are cryptographically linked using the hash-lock and time-lock mechanisms described above. The focus of this section is on evaluating the end-to-end delay for the transactions on the public ledger (Ethereum) and Hyperledger Fabric, which are interconnected with the interledger component. On the other hand, the results in Section 5.1 consider the interaction with other components and functionalities related to decentralized authorization in constrained IoT environments.

4.1.2 Results

The previous deliverable D4.3 contained results showing the gains in terms of smaller transaction costs when an Ethereum network (public Ethereum testnet) is interconnected with a permissioned ledger, thus reducing the transactions that need to be conducted on the public ledger. The new results presented below show the gain in terms of reduced end-to-end delay, in the case of an authorization scenario that involves the interconnection of two ledgers, the public Ethereum testnet and the Hyperledger Fabric permissioned ledger [Sir+20]. Because the processing delay at the interledger component is minimal, the end-to-end delay is determined by the transaction delay on the Ethereum and Fabric ledgers.

The end-to-end delay is shown in Figure 2. The results show that the delay due to Fabric is approximately 13% of the total delay. Moreover, our results show that each transaction on the Hyperledger Fabric network has a delay of approximately 2 seconds. In Section 5.1.1 we present evaluation results for the delay and the Ethereum transaction in more complex IoT authorization scenarios involving multiple authorization servers.

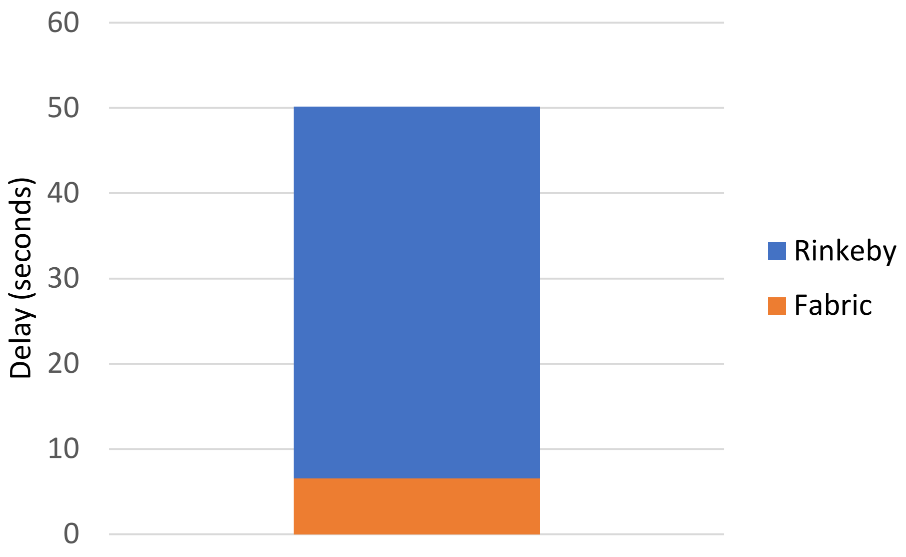


Figure 2: End-to-end delay when Hyperledger Fabric (authorization blockchain) is interconnected with the public Ethereum testnet Rinkeby (payment blockchain).

The table below relates the evaluation scenarios to the corresponding interledger component requirements identified in deliverable D2.4 and describes how each requirement is achieved.

Table 15: Requirements for the SOFIE Interledger component

Req. ID	Requirement Description	Priority	Evaluation/Emulation Scenarios and how the requirement is achieved
	Interledger		
RF01	User interaction is not required for interledger operations.	MUST	Sections 5.1 and 5.3. The procedures of this component are event-triggered and the user interaction is not required.
RF02	There should be support for atomic interledger operations.	SHOULD	Sections 5.1 and 5.3. The scenarios investigated involve hashed time-lock contracts, which support atomic cross-chain operations.

4.2 Privacy and Data Sovereignty

The Privacy and Data Sovereignty (PDS) component achieves data sovereignty using the OAuth 2.0 authorization framework and privacy by using verifiable technologies. In this section, we discuss and evaluate how these two concepts can be combined. The evaluation considers the time required for implementing VC (Verifiable Credential) related operations.

VCs in PDS can play the role of the "authorization grant." This design choice, illustrated in the following figure, is compatible with the OAuth 2.0 RFC, which specifies that "client credentials (or other forms of client authentication) can be used as an authorization grant". Using PDS, a resource owner generates VCs for the clients authorized to access a resource (step 1 in Figure 3). VC generation is performed independently of the authorization server (AS). Then, a client can request an access token from an authorization server by presenting a VC (steps 2 in Figure 3): the authorization server issues a VC proof request and the client generates the appropriate proof. If the latter proof is valid, the AS proceeds with the token generation process (step 3). The whole process does not require interaction with the resource owner. Furthermore, the authorization server learns no information about the client, apart from the fact that it is authorized to access a resource: even if the same client tries to generate a new token, for the same resource, the AS will not be able to tell if this is the same client or not. In this way, the data minimization requirement for this component is achieved. Finally, authorization servers can be preconfigured with the appropriate proof request parameters: in this case, the only operation an authorization server has to perform is the verification of the correctness of a proof.

The generated token can be used for accessing a protected resource: the validity of the presented token is verified by the IAA component (see next section).

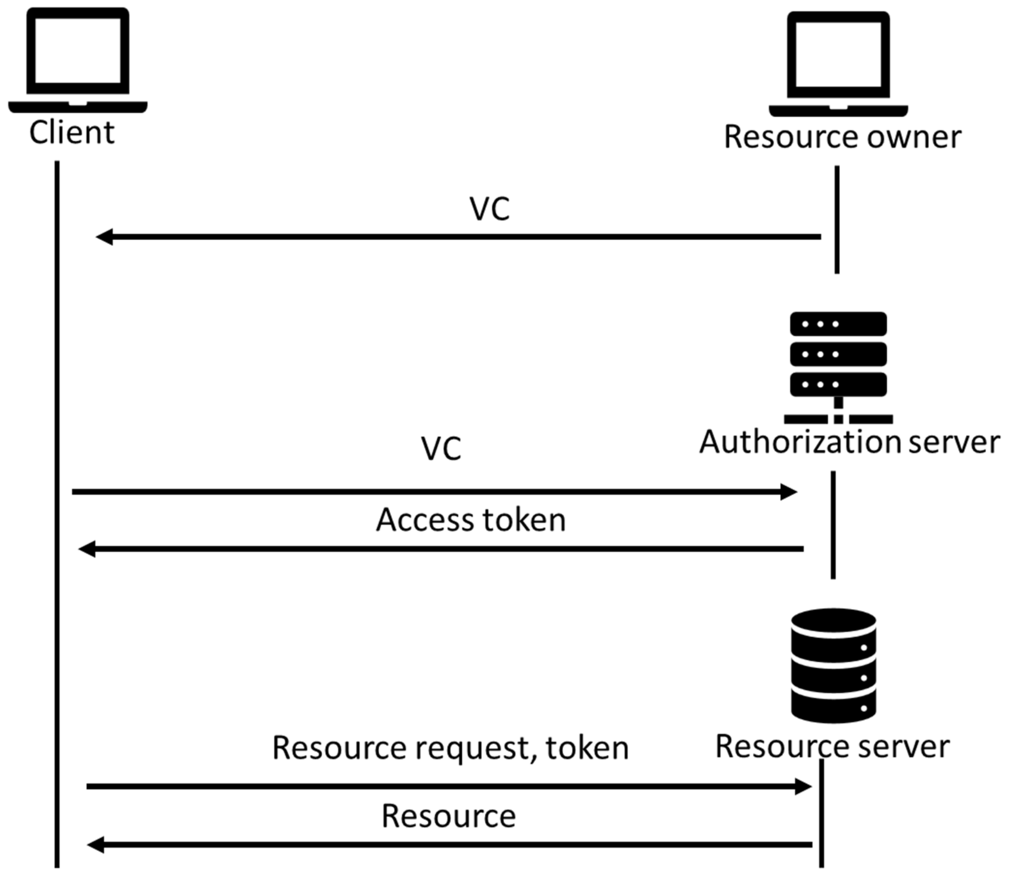


Figure 3: OAuth2.0 using VCs as an authorization grant.

4.2.1 Experiment setup

The main computation intensive operations of our system are the following:

- key-pair generation: each client must generate a public/private key pair before requesting a VC. Similarly, each owner must generate such a key-pair once. The keys used in our implementation are based on Curve25519. This key is associated with a Decentralized Identifier (DID): the DID of an owner is included in the generated VCs, so that the provenance of a VC can be verified.
- VC generation: a resource owner should generate a VC for each authorized client.
- verifiable presentation generation: a client can prove the possession of one or more VCs to an AS by generating a "verifiable presentation" (VP). A VP includes data from one or more VCs and is packaged in such a way that the authorship of the data is verifiable.
- verifiable presentation validation: each AS must verify the validity of a VP using interactive Zero Knowledge Proofs.

Each VC is described by a "schema" that includes the number and the type of claims included in a VC. Our VC schema includes a single attribute used for storing the URIs of the resources a client can access. We have implemented our solution using Hyperledger Indy python3 SDK. Hyperledger Indy (henceforth, simply Indy), which is hosted by the Linux foundation, aims at providing a blockchain-based system for decentralized identification. Indy offers tools, libraries, and reusable components for implementing W3C compatible DIDs and VCs. Indy's blockchain, which holds the role of the registry, is public, in the sense that anybody can read from it, and permissioned, in the sense that only entities that hold a "special" role have write access. We have measured the performance of our implementation in a Xubuntu 18.04 virtual machine that uses two cores with an Intel-i7 7700 CPU and 4GB RAM.

4.2.2 Results

Table 16 shows the time (in milliseconds) required to perform each operation identified in the previous section.

Table 16: Time required to perform a VC-related operation

Operation	Time to complete
Key generation	191.26
VC generation	82.37
VP generation	101.46
VP validation	58.32

Furthermore, the use of VCs as an authorization grant has the following security related properties:

Users' secret information protection. User entities (i.e., owners and clients) generate and store the required secret information by themselves, i.e., at no point the involvement of a 3rd party is required. This property contributes to the self-sovereignty of users' personal information and decreases the probability of a data breach. Furthermore, by including the DID of an owner in VCs, instead of their public key, the owner's key can be refreshed: since the DID of an owner remains always the same, an owner can update his key without needing to update all issued credentials (recall that the owner's DID is used for looking up his public key in the registry).

Client privacy preservation. Every time a client requests a VC, he generates a new key-pair. Therefore, it is not possible, even for the same authorization server, to tell if two VCs belong to the same client. Hence, clients are protected against tracking. Moreover, by using zero-knowledge proofs, clients can control the amount of information they reveal when constructing a VP. For example, a client does not have to include in a VP all the resources he is authorized to access.

Attack surface reduction. The amount of verifications an authorization server needs to perform is less compared to a traditional access control system. Indeed, in our system, an authorization server has only to verify the validity of a VC, whereas in most legacy access control systems an authorization server would have to verify an access control policy as well. Furthermore, authorization servers are not required to store any additional secret information in order to implement our protocols, neither do they have to maintain user accounts. Similarly, IoT resources do not have to be modified. Moreover, an owner does not have to participate in the access control process; indeed, after VC generation an owner can go offline.

Fast revocation. Every time a client requests access, he must provide a proof that his VC has not been revoked. In our implementation, the verification of this proof requires a lookup in a data structure stored in the registry. Whenever a VC is revoked, an owner updates this data structure: after the update, the VC cannot be used. Therefore, as long as an owner keeps the information stored in the registry up-to-date, VC revocation is instantaneous.

Further evaluation scenarios concerning the PDS component functionality are presented in subsequent subsections. The table below relates the evaluation scenarios to the corresponding PDS component requirements identified in deliverable D2.4 and describes how each requirement is achieved.

Table 17: Requirements for the SOFIE PDS component

Req. ID	Requirement Description	Priority	Evaluation/Emulation Scenarios and how the requirement is achieved
PDS			
RF10	SOFIE must follow the data minimisation principle for personal data and only request or process what is necessary for the situation and purpose.	MUST	Using VCs as an authorization grant (evaluated in this section) allows a user to disclose only the required information.
RF11	Processing of individual's personal data is justified by a valid legal basis, e.g., a valid consent from the individual.	MUST	Using VCs as an authorization grant (evaluated in this section) requires the user's consent.
RF12	Consent to process personal data must be revocable at any time.	MUST	VCs can contain an expiration time after which they are not valid.
RF13	SOFIE must allow organisations and actors to manage (create, update, delete) their own data privacy policies.	MUST	Privacy policies are expressed using VC schemas (evaluated in this section). There is no restriction on the format of the schema.
RF14	SOFIE should support user privacy even when aggregate statistics are made public (e.g., using differential privacy mechanisms).	SHOULD	Not evaluated in the current deliverable.

4.3 Identification, Authentication, and Authorization

In D4.3 we presented and evaluated how the Identification, Authentication and Authorization (IAA) component can be used to authenticate users using Decentralized Identifiers (DIDs). In this section we evaluate the ability of the IAA component to authorize users based on JSON Web Tokens (JWTs), which are generated by the PDS component (presented in the previous section). In particular, we consider JWT access tokens supported by ERC-721 Ethereum tokens as presented in [Fot+20].

ERC-721 is an open standard that describes how to build "non-fungible or unique tokens on the Ethereum blockchain." This standard is in many ways similar to ERC-20, which is probably the most popular Ethereum standard and is used for creating custom Ethereum tokens. However, in contrast to ERC-20 tokens, ERC-721 tokens are "unique" and non-interchangeable with other tokens (non-fungibility). Many Ethereum wallets, such as Metamask, can handle these tokens. All ERC-721-based tokens are identified by a unique identifier and can be owned by one user only. This standard, like every other token standard in Ethereum, defines some functions that a smart contract should implement in order to be able to create and handle ERC-721 tokens. Furthermore, the ERC-721 metadata extension defines some additional functions that can be used for associating an ERC-721 token with metadata. The following table describes the functions used by the PDS component and defined in the ERC-721 standard and in the ERC-721 metadata extension. Additionally, functions *transferFrom* and *approve*, when invoked generate an event, named *Transfer* and *Approval* respectively. Both these events have three attributes; the attributes of the *Transfer* event are the *from* address, the *to* address, and the *tokenid*, while the attributes of the *Approval* event are the *owner* address, the *approved* address, and the *tokenid*.

Table 18: ERC-721 and ERC-721 metadata extension functions used by the PDS component

Function	Purpose
ownerOf(tokenid)	Accepts as input a tokenid and returns the address of the token owner
transferFrom(from, to, tokenid)	Transfers a tokenid from one Ethereum address to another
approve(address, tokenid)	Approves an Ethereum address to manage a tokenid on the owner's behalf
getApproved(tokenid)	Retrieves the Ethereum address allowed to manage tokened
tokenURI(tokenid)	Accepts as input a tokenid and returns a URI that points to the token's metadata

From a high-level perspective, a system with the PDS component operates as follows. Using the PDS component, the client requests an access token from the authorization server, which first generates a JWT and an ERC-721 token, then transfers the ERC-721 token to the Ethereum address of the client, and finally sends the JWT to the client. The client requests access from the resource server, providing the JWT. The resource server uses the IAA component and retrieves the corresponding ERC-721 token which is used for verifying the validity and ownership of the JWT: if all verifications are successful the resource server allows the client request. This process is illustrated in the following figure.

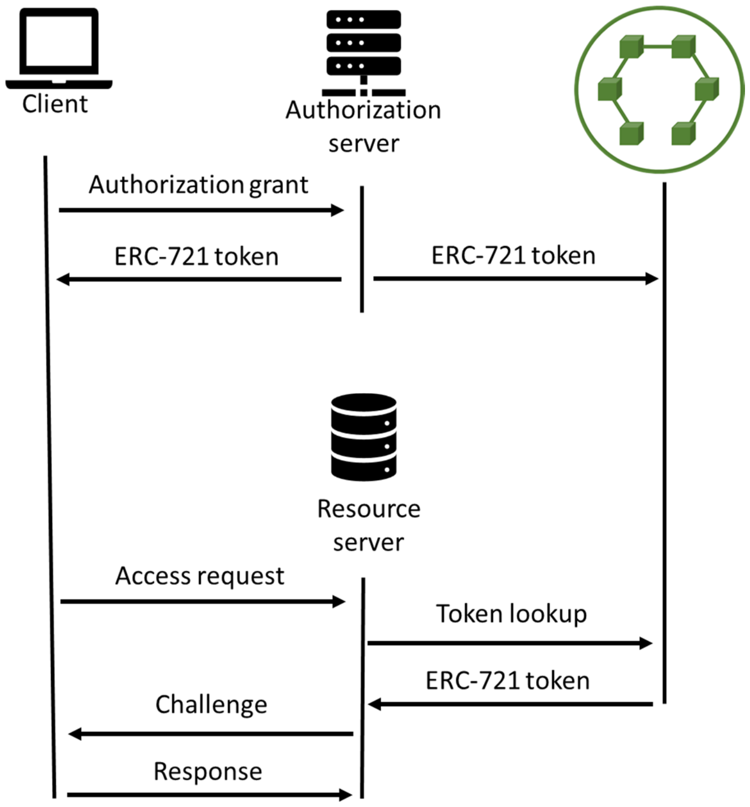


Figure 4: User authorization using JWT access token supported by ERC-721 tokens.

4.3.1 Experiment setup

The experiment scenario considers the case of an IoT gateway access, whose owner wishes to grant access to client users. As an IoT gateway, we used Mozilla's WebThings Gateway that implements the Web of Things (WoT) standard. For our proof of concept, we chose not to modify the gateway itself; we instead developed an application that acts as a proxy, between the client, the blockchain, and the gateway (which holds the role of the resource server). For an Ethereum wallet, we used the Metamask Firefox extension, which can handle ERC-721-based tokens. We implemented clients as JavaScript web applications using the web3.js Ethereum JavaScript API.

The main component of our system is the smart contract that implements the functions of the ERC-721 interface. In addition to the functions specified in the previous section, we implemented two more functions. The first one, named *mint*, is for creating new tokens, and the second, named *burn*, is for "burning" tokens, i.e., destroying them.

4.3.2 Results

We tested our proposed system in the Rinkeby Ethereum test network. The smart contract transaction cost is measured in gas, which is Ethereum's unit for measuring the computational and the storage resources consumed by a smart contract. Each operation of a smart contract costs a fix amount of gas. Gas cost is the number of units of gas required to perform an action, while gas price is the amount of "ether" (i.e., Ethereum's specific coin) a client is willing to pay per unit of gas. The average price of a unit of gas when the experiment was executed was $0.011 × 10^{-4}. The following table shows the cost of deploying the smart contract in the blockchain network, as well as the cost of the operations performed by our system in terms of gas units.

Table 19: Cost of ERC-721 token management functions

Operation	Cost measured in gas
Contract Deployment	1 585 444
Create a token	254 141
Burn a token	85 791
Transfer a token	63 858
Approve	45 735

When a Client performs an access request, the Resource server has to lookup a token in the blockchain and calculate a random number (the challenge); both operations are instantaneous and they are bound to the communication delay between the resource server and the Ethereum RPC point. Then, the client has to compute a digital signature over the challenge and transmit the response. Finally, the Resource server has to verify the signature. All cryptographic operations require a few ms.

With the above implementation of the IAA component, the authorization server does not communicate directly with the resource server, not even in the case of a token revocation. The authorization server learns only the URI of the resource that a client wants to access, which does not have to be the publicly known URI of the resource: any pseudonym that the resource server can understand can be used instead. On the other hand, the metadata of an ERC-721 token are immutable and visible to anybody, constituting a privacy threat. In order to address this shortcoming and enhance the clients' privacy, metadata stored on the public blockchain can be encrypted using a key known only to the resource owner, the client, and the resource server.

ERC-721 specifications define a method *approve* that allows the token owner to specify a different Ethereum address that will manage its token. However, in our system a delegee never interacts with the blockchain, therefore the input to the *approve* method does not have to be an Ethereum address. Other types of delegee identifiers can be considered, such as legacy public keys, or even contemporary forms of authentication such as VCs. This increases the flexibility of the current implementation of the IAA component.

Further evaluation scenarios concerning the IAA component functionality are presented in subsequent subsections. The table below relates the evaluation scenarios to the corresponding IAA component requirements identified in deliverable D2.4 and describes how each requirement is achieved.

Table 20: Requirements for the SOFIE IAA component

Req. ID	Requirement Description	Priority	Evaluation/Emulation Scenarios and how the requirement is achieved
IAA			
RF03	Resource owners must be able to delegate the authentication and authorization tasks for their resources.	MUST	The way access token validation is implemented in the IAA component allows configuring it with the smart contract of any authorization server.
RF04	The IAA component must provide users the capability to revoke authorizations.	MUST	This can be achieved by transferring the corresponding ERC-721 token (evaluated in the current section)
RF05	The IAA component must allow individuals to control their personal information and digital identities (e.g., support self-sovereign identity technology).	MUST	The IAA component also supports user authentication using VCs (evaluated in Section 5.2)
RF06	The IAA component must support secure, tamper-proof, and verifiable logging of transactions and events.	MUST	All operation on ERC-721 tokens are recorded in the blockchain (evaluated in the current section)
RF07	The IAA component must support Role Based Access Control (RBAC).	MUST	RBAC can be implemented using VCs (evaluated in Section 5.2)

4.4 Semantic Representation

The main purpose of the SOFIE Semantic Representation (SR) component is to define a common representation model for IoT devices (Things), their services, and their data. The goal of this component is to enable interoperability and automation among different IoT environments.

This component will not be a separate software module, it is more of a logical component that will be implemented as part of other components or pilots. For this reason, we do not include an experimental setup in this section. All the pilots use the W3C Web of Things (WoT) standard [Kov+19]. The following table lists the requirements for the SR component and describes how each requirement is achieved.

Table 21: Requirements for the SOFIE SR component

Req. ID	Requirement Description	Priority	Evaluation/Emulation Scenarios and how the requirement is achieved
Semantic Representation			
RF15	SOFIE must define an IoT things description model based on well-known standards (e.g., W3C standards)	MUST	The component uses the W3C Web of Things standard, and, in particular, the WoT's Thing Description (TD), which defines an information model based on a semantic vocabulary and representation using JSON.

RF16	SOFIE must implement standardized metadata and data representation formats and support various data modalities.	MUST	The WoT TD provides metadata for Things in both a human-readable as well as in machine readable format (evaluated in Section 5.2)
RF17	The semantic representation model of the system must be open and extensible by third parties (e.g., support the extension of the existing knowledge base and associations by extracting supplementary triples from RDF documents).	MUST	The W3C WoT TD is open, and is extensible by definition.
RF18	SOFIE must provide service discovery and resources selection processes based on multiple criteria over the features, associations, and interaction patterns of integrated resources.	MUST	The W3C WoT enables clients to learn a Thing's attributes, functionalities and its access points, prior to accessing the Thing itself. It allows any client to search for Things by their attributes and functions or to perform a semantic search based on a unified vocabulary (evaluated in Section 5.2).
RF19	SOFIE should support the semantic update and enhancement of resources' descriptions and associations in a dynamic way.	SHOULD	As mentioned above, the semantic representation of a Thing, using the W3C WoT standard, can be extended and updated any time.

As we can observe in the above table, all the requirements for the SR component can be met by the W3C WoT standard. The use of the WoT standard is evaluated for access control in multitenant IoT systems Section 5.2.

4.5 Marketplace

The goal of the SOFIE Marketplace (MP) component is to enable the trade of different types of assets in an automated, decentralized, and flexible way. The actors (buyers and sellers) can carry out trades by placing bids and offers using the MP component, which utilizes Ethereum smart contracts.

The implementation of the MP component exposes an API that offers two actions: a request and an offer. Through the request action an actor declares that an asset is available for sale, thus creating an auction, while through the offer action an actor declares his interest in buying an asset, thus placing a bid. These two actions are highly parametrizable, thus supporting the needs of diverse services. The performance results presented in this section consider two metrics: the response time, which is the time until a transaction is mined, and the execution cost.

4.5.1 Experiment setup

The prototype implementation of the MP component introduces a series of Solidity-based smart contracts that are deployed in an Ethereum node. These smart contracts define the basic functionality of the marketplace, forming the foundation upon which sophisticated services can be build. For instance, the prototype implementation of the smart contract, which is used in the decentralized energy flexibility marketplace pilot, extends the prototype implementation of the marketplace component by inheriting and implementing the component's contracts and interfaces (or abstract contracts), respectively. Thereupon, the component implementation includes basically the source code for submitting a request and an offer, while providing the template for developing other necessary functions that can be specific to a service. For example, the function that decides an offer is left abstract, allowing the specific service to define it according to its individual requirements.

We have tested the prototype implementation of the MP component in a local Ethereum node that is built with the Ganache blockchain. With respect to the parameters of Ganache, we fixed the average block mining time to 15 s and set the block gas limit to 10,000,000 gas units. Both these values reflect the corresponding values in the public Ethereum main net. We deployed the smart contract and invoked the available functions through a JavaScript script that utilizes the web3.js Ethereum API.

4.5.2 Results

We assess the performance of the MP component in two directions: the response time, which is the time until a transaction is mined, and the execution cost, since transactions in Ethereum incur a cost. Regarding the response time, all functions that result in blockchain transactions, are mined within one mining period, which is 15 seconds, hence their expected response time is 7.5 seconds.

The actions of our system that involve the invocation of the smart contract functions incur some computational overhead. The following table shows the cost of deploying the smart contract in the Ethereum network, as well as the cost of operations performed by our system measured in gas units.

Table 22: Cost of MP component's smart contract operations

Operation	Cost measured in gas
Contract Deployment	2 117 971
Submit Request	137 875
Submit Offer	142 634
Close Request	88 954
Delete Request	75 476

The above results can be considered as the minimum cost for operating a marketplace-based service. Depending on the requirements of the service, e.g., the amount of data that an offer includes, the cost is expected to be higher. The execution costs of a realistic service is presented in Section 6.2Decentralised Energy Flexibility where we assess the execution cost of the emulated decentralized energy flexibility marketplace pilot.

The table below relates the evaluation scenarios to the corresponding MP component requirements identified in deliverable D2.4 and describes how each requirement is achieved.

Table 23: Requirements for the SOFIE MP component

Req. ID	Requirement Description	Priority	Evaluation/Emulation Scenarios and how the requirement is achieved
Marketplace			
RF20	The marketplace must log the configuration of all trading actions (including offers, bids, parameters of resources, transactions etc.).	MUST	The pilot emulation scenario is investigated Section 6.2. The marketplace component is implemented in Ethereum smart contracts. Hence, trading actions are recorded in the Ethereum blockchain, making them traceable. This is also discussed in deliverable D5.2.
RF21	The marketplace must provide actors the capability to post/claim offers and sell/negotiate/exchange/buy resources and digital objects.	MUST	The marketplace component offers two interfaces: Request Maker for sellers to create, manage and conclude auctions, and Offer Maker for buyers to participate and bid in auctions. These functions are illustrated in the emulation scenario of Section 6.2.
RF22	The marketplace must support transparent trading of resources, i.e. the bids/offers matching process and the payments must be transparent.	MUST	The marketplace component utilizes Ethereum smart contracts to record bid/offers and payments, thus supporting transparency. This functionality is illustrated in Section 6.2.
RF23	The marketplace must provide evidence once trades have been completed and resources have been properly delivered to the buyers.	MUST	Ethereum transaction receipts, that describe the state of the blockchain after a transaction took place, can be used as evidence of a trade. This functionality is illustrated in Section 6.2.
RF24	The marketplace should allow integration of payment technologies.	SHOULD	Currently, the emulation of the marketplace component, as described in Sect.6.2, supports transferring ETH coins and ERC20 tokens. The prototype implementation of the component does not disallow the integration of other payment technologies.

As we can observe from the above table, the requirements for the MP component can be met, since it is based on Ethereum, which, as any public blockchain technology, natively offers the majority of the required features.

4.6 Provisioning and Discovery

The goal of the Provisioning and Discovery (PaD) component is to enable the discovery of IoT resources and their metadata. This component provides service discovery using Bluetooth and

DNS, and device and license provisioning. The following table presents the requirement, defined in Deliverable 2.5, for this component.

Table 24: Requirement for the SOFIE PaD component

Req. ID	Requirement Description	Priority	Scenarios and how the requirement is achieved
Provisioning and Discovery			
RF18	SOFIE must provide service discovery and resources selection processes based on multiple criteria over the features, associations, and interaction patterns of integrated resources.	MUST	The component will use W3C's WoT standard. It will be used and demonstrated in the MRMG pilot (WP5)

The PaD component uses Bluetooth and DNS service discovery protocols to search for IoT devices. When the component finds a new IoT device it checks the device's semantic description, which is defined with the W3C WoT standard. If the description matches the search requirements, the device can be provisioned. Based on the above functionality, the component can meet its requirement. The PaD component will be used and demonstrated in the MRMG pilot and is not evaluated in the emulation scenarios of the current deliverable.

5 Further IoT resource access evaluation

In this section we present further evaluation results for IoT resource access scenarios and solutions. Similar to the corresponding section in the previous evaluation deliverable D4.3, this section goes to much more depth and considers more alternatives and their tradeoffs than what we have seen in Section 4, which focuses on SOFIE's framework components. Other sections of this deliverable do not intimately depend on the results of this section. But, this section shows that more detailed design and evaluation is possible and can uncover many useful alternatives in the IoT world, which includes a multitude of use cases with different restrictions and requirements, which, in turn, need solutions with different tradeoffs in terms of complexity, performance (cost and delay), reliability, privacy, and transparency.

The first two solutions utilize the OAuth 2.0 authorization framework combined with blockchain functionality. Specifically, in Section 5.1 we present results for decentralized authorization in constrained IoT environments that extend the results presented in the previous evaluation deliverable D4.3. The new results consider scenarios where the interledger component interconnects a public Ethereum testnet with a Hyperledger Fabric network. Moreover, the new results investigate two policies for selecting the subset of the authorization servers that are required for authorization. In Section 5.2 we present an access control solution for Web of Things (WoT) IoT "hubs" based on Verifiable Credentials (VCs). This solution is planned to be included in the IAA component and is suitable for scenarios where a resource owner owns a few (usually a single) resource, powerful enough to implement public key cryptographic operations.

These solutions consider OAuth 2.0, which is a framework for delegating authorization to access a protected resource [Har+12]. It enables a third-party application (client) to obtain access with specific permissions to a resource, with the consent of the resource owner. Access to the resource is achieved through access tokens, created by an authorization server. The specific format of the access tokens is opaque to the clients and to OAuth 2.0. The authorization consent by the resource owner is provided after the owner is authenticated; however, the authentication procedure is not part of OAuth 2.0. Authorization is provided for different levels of access, such as read and write/modify, which are termed scopes, and for a specific time interval.

The advantages of combining authorization based on frameworks such as OAuth 2.0 with blockchains and smart contracts are the following:

- Blockchains can immutably record hashes of the information exchanged during authorization and cryptographically link authorization grants to payments and other IoT events recorded on the blockchain. These records serve as indisputable receipts in case of disagreement.
- Smart contracts can encode authorization policies in an immutable and transparent manner. Policies can depend on payments as well as on other IoT events recorded on the same or on different blockchains.
- Smart contracts run on all nodes of a blockchain. Hence, sending resource access requests to smart contracts can protect against DoS attacks that involve a very high resource request rate, since requests are not handled by one node, which would be a single point of failure.

Further advantages are identified in the subsections below. Moreover, the solutions can leverage the hash-lock and time-lock mechanisms presented in Section 4.1.

Finally, in Section 5.3 we present decentralized interledger gateway architectures for IoT authorization scenarios involving the interconnection of two ledgers: an authorization ledger and a payment ledger, which is similar to the scenario considered in Section 5.1 for constrained IoT environments. The proposed architectures differ in their complexity, transaction cost, and ability to handle transactions involving multiple ledgers.

5.1 Decentralized authorization for constrained IoT devices

In Deliverable D4.3 (First Architecture and System Evaluation Report), we presented four models that we have developed [Sir+19a] for decentralized authorization in environments with constrained IoT devices that provide different tradeoffs in terms of cost, delay, complexity, and privacy:

1. Linking authorization grants to blockchain payments
2. Smart contract handling of authorization requests
3. Smart contract and two blockchains for authorization and payment with interledger mechanisms
4. Decentralized authorization with multiple Authorization Servers

The first two models are our baseline scenarios: in the first, only hashes of authorization information are immutably recorded on the blockchain and smart contracts are not used, whereas the second model utilizes a smart contract, but on a single (public) blockchain. The third model exploits two blockchains whose transactions are securely linked using interledger mechanisms provided by the interledger component and quantifies the significant cost reduction that can be achieved by moving smart contract authorization functionality to a permissioned or private blockchain. The fourth model focuses on decentralized authorization for constrained IoT devices utilizing two blockchains with interledger mechanisms.

Below we present new results for the fourth model, which considers decentralized authorization for constrained IoT environments involving multiple Authorization Servers (ASes) [Sir+20]. The new results consider scenarios where the interledger component interconnects a public Ethereum testnet with a Hyperledger Fabric network. Moreover, the new results investigate two policies for selecting the subset of ASes, i.e., the m-out-of-n ASes, that are required for authorization.

We first recall the model that involves multiple ASes in the authorization procedure. The client sends a resource access request to the address of the smart contract responsible for handling access to the IoT device. The smart contract address can be obtained by the client sending a query to the IoT device or, e.g., reading a QR code on it. However, this approach cannot ensure that the legitimate URL or smart contract address is provided by the IoT device. This can be ensured if the client uses a registry service that resides on the blockchain and contains a binding between the IoT device's URI and the URL of the AS or the smart contract address handling authorization, or by including this information in Decentralized Identifier (DID) documents [Ree19]. Furthermore, we assume that the client, the resource owner, and the ASes have an account (public/private key pair) on both the authorization blockchain, which is implemented on Hyperledger Fabric, and the payment blockchain, which is implemented on the public Ethereum testnet.

Let *n* be the number of ASes that are collectively responsible for providing authorization. Each AS *i* shares a different secret key, $KThing_i$, with the protected resource (Thing). Authorized access to the Thing requires tokens from *m* out of *n* servers. The policy specifying the required number of ASes is defined in the smart contract and is also known to the Thing. Fault tolerance is provided by having *n* ASes which can respond to requests, but requiring only $m < n$ for authorization to proceed. Compared to having a single AS, the proposed scheme provides higher security since *m* ASes need to agree for the client to access the protected resource. We investigate two policies for how the *m* servers are selected. With the first, the smart contract selects the specific *m* servers, based on their previous behaviour that can involve response time or transaction cost. The second policy selects the first *m* servers that respond to an authorization request. We describe the two policies in more detail below.

In response to the client's authorization request, each AS sends a different PoP key PoPi, encrypted with the Thing's secret key and the client's public key, and an access token with a

MAC tag to ensure its integrity (Figure 5). The client thus obtains *m* different PoP keys, which it XORs to obtain the secret PoP key that will be used to establish a secure communication link with the Thing. These *m* PoP keys, encrypted with the Thing's key *KThing$_i$* that it shares with each of the *m* ASes, are also sent to the Thing. Hence, if the Thing performs the same XOR function on the *m* PoP keys, it will obtain the same PoP key as the client.

In order to reduce the amount of data transmitted to constrained devices we propose two schemes for reducing the authorization information the client sends to the Thing: (a) aggregate MAC tags and (b) transmission of common token fields once. With aggregate MAC tags [KL08], the client does not send to the Thing the token payloads received from the *m* ASes, but only one aggregate MAC tag that is computed by taking the XOR of the *m* MAC tags the client receives from the *m* ASes. With the second optimization, the client sends the token fields that are common to all ASes only once (these correspond to token$_1$, . . . token$_m$ in Figure 5). The common token fields include the subject (Thing) the token refers to, the scope of access, the token creation time, the token validity time, and the token type. The fields which are different include the AS and token ID fields.

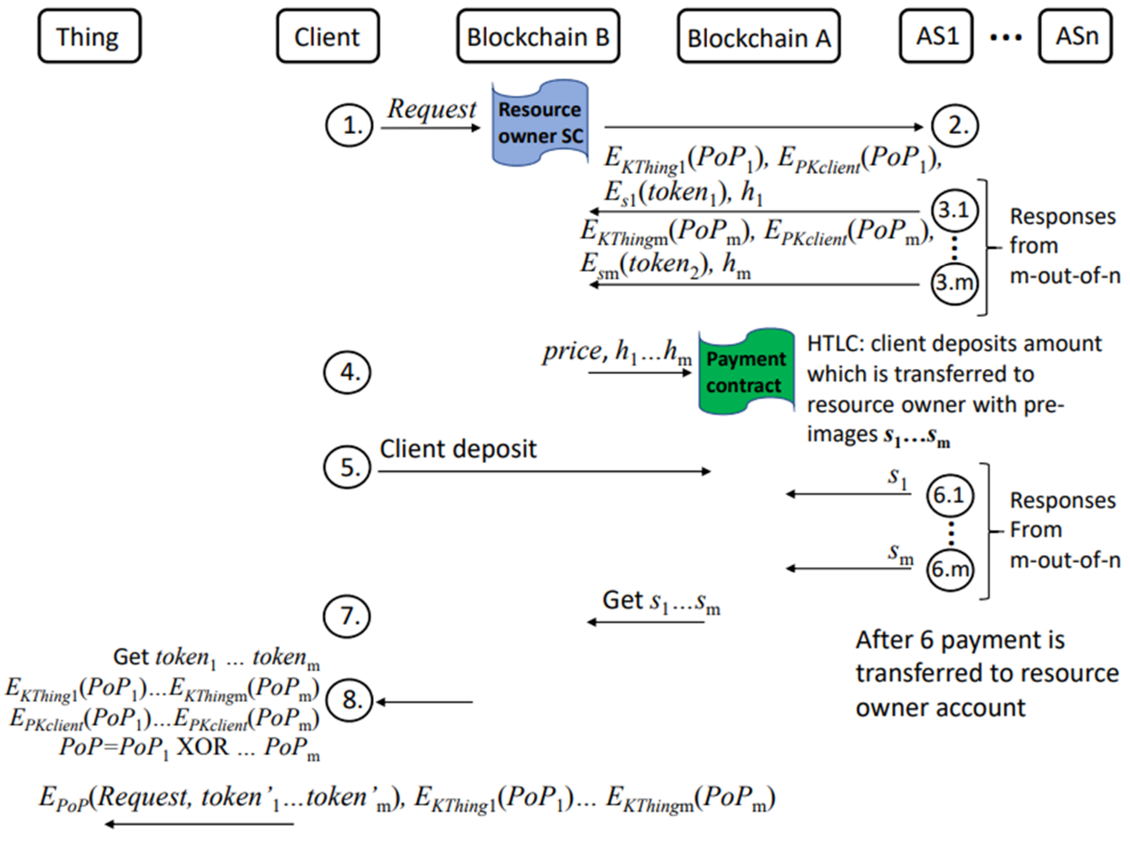

Thing
Client
Blockchain B
Blockchain A
AS1
ASn
1.
Request
Resource owner SC
2.
$E_{KThing1}(PoP_1)$, $E_{PKclient}(PoP_1)$, $E_{s1}(token_1)$, h_1
3.1
$E_{KThingm}(PoP_m)$, $E_{PKclient}(PoP_m)$, $E_{sm}(token_2)$, h_m
3.m
Responses from m-out-of-n
4.
price, $h_1 \ldots h_m$
Payment contract
HTLC: client deposits amount which is transferred to resource owner with pre-images $s_1 \ldots s_m$
5.
Client deposit
s_1
6.1
s_m
6.m
Responses From m-out-of-n
7.
Get $s_1 \ldots s_m$
After 6 payment is transferred to resource owner account
Get $token_1$ ... $token_m$
$E_{KThing1}(PoP_1) \ldots E_{KThingm}(PoP_m)$
$E_{PKclient}(PoP_1) \ldots E_{PKclient}(PoP_m)$
$PoP = PoP_1$ XOR ... PoP_m
8.
$E_{PoP}(Request, token'_1 \ldots token'_m)$, $E_{KThing1}(PoP_1) \ldots E_{KThingm}(PoP_m)$

Figure 5: Decentralized authorization; each authorization grant requires m out of n AS responses

Policy 1: Selection of the *m* ASes based on previous performance

This policy requires that the smart contract handling the requests maintains the list of ASes along with their performance for previous authorization requests. The performance can be in terms of the time they took to respond to requests or their transaction cost. Such information allows the smart contract to prioritize ASes in order to select those that respond quickly, hence avoiding ASes that have a high delay, are faulty, or have a high transaction cost. This intelligent selection of ASes requires maintaining historic information, which would require higher memory and processing by the smart contract at the authorization ledger.

This policy can be implemented by including in the notification event in step 2 of Figure 5 the information for the *m* ASes that were selected by the smart contract. Moreover, in step 3 the hashes corresponding to the secrets of the selected m ASes are included in the payment contract. Hence, in step 6 only the selected *m* ASes can respond. Note that in case of an error or fault of one of the selected *m* ASes, the client can use a timeout to restart the authorization procedure from the beginning. Moreover, in the case of an error or fault the smart contract can keep information for the faulty AS, in order to avoid it in subsequent authorization requests.

Policy 2: Selection of the *m* ASes that respond first

With the second policy the smart contract allows all ASes to respond to the authorization request in step 3 of Figure 5, and selects the first *m* ASes that respond. With this approach the smart contract does not need to maintain the list of ASes, nor information of their performance for previous authorization requests. However, there is a possibility that the smart contract receives more than *m* responses. This depends on the duration for mining a block on the blockchain (in the case of public blockchains with Proof-of-Work consensus) or for obtaining consensus to add it to the blockchain (in the case of permissioned blockchains). In public blockchains, these responses can incur a gas cost independent of whether the ASes that gave the response were among the *m* ASes to provide decentralized authorization. Once the first *m* AS responses are received, then in step 4 of Figure 5 the hashes corresponding to the secrets of the first *m* ASes that responded are included in the payment contract. Hence, in step 6 only the first *m* ASes that responded in step 3 can submit the secret corresponding to their hash. As in the previous policy, to handle the case where an AS exhibits an error or is faulty in step 6, the client has a timeout after which it restarts the authorization procedure from the beginning.

By exploiting the simplicity of the procedure for selecting the subset of ASes, a variant of the second policy is to avoid having the client send the initial request to the smart contract on the authorization chain in step 1. Instead, the client can send the request directly to all ASes; this requires that the client knows the ASes. The advantage of this variant of the second policy, as we will see in the evaluation results below, is the lower total delay, since the number of transactions on the authorization chain is reduced by one.

5.1.1 Evaluation

The evaluation results considered the interledger component that interconnects a Hyperledger Fabric permissioned blockchain and the Rinkeby public Ethereum testnet. The version of Hyperledger Fabric used in the experiments was 1.4. The AS functionality was based on a PHP implementation of the OAuth 2.0 framework, extended to support CWT's CBOR encoding. The client used Web3.js to interact with the blockchain.

Comparison of the two policies for selecting *m* ASes

We compare the two policies discussed above for selecting *m* ASes, in terms of the execution cost. The first policy requires that the past performance of all ASes is maintained and the smart contract selects the subset of ASes based on this performance. Unlike the first policy, the second policy allows all ASes to respond to authorization requests and selects the subset of ASes that responded first. The two policies are implemented in the smart contract that handles authorization requests in step 1 of Figure 5. The results we present below consider the case where the authorization contract and the payment contract run both in the Ethereum public testnet and the case where the authorization contract runs on a private blockchain and only the payment contract runs on the Ethereum public testnet.

Figure 6 shows that in the case of one blockchain, i.e., both the authorization smart contract and the payment contract run on the public Ethereum testnet, the first policy has a higher execution cost, measured in terms of the execution cost of the smart contract running on the Ethereum network, compared to the second policy. This is due to the past history that this policy maintains in order to select the ASes for authorization; maintaining the past history incurs a high execution and storage cost when it is maintained on a public blockchain. On the other hand, the second policy does not maintain historic information but rather selects the first *m* ASes that respond. As expected, when more ASes need to send authorizations the execution cost is higher. Moreover, the incremental cost is the same for both policies. The higher execution cost is exchanged with the higher level of security and trust when more ASes are required to send authorizations.

The figure below also shows that both policies have the same execution cost in the case of two blockchains, a public and a private/permissioned blockchain, since in this case only the payment transactions on the Rinkeby testnet, which are the same for both policies, incur an execution cost. The results for two blockchains illustrate the gains from utilizing a private/permissioned and a public ledger: The private/permissioned ledger can implement more elaborate authorization policies and store information that would incur a high cost on a public ledger. Of course, in a private/permissioned ledger, trust is maintained among the participating nodes rather than on a wide-scale, as in the case of public ledgers. Depending on the application requirements, trust among a limited set of nodes may be sufficient, making the combination of private and public ledgers an attractive and practical approach.

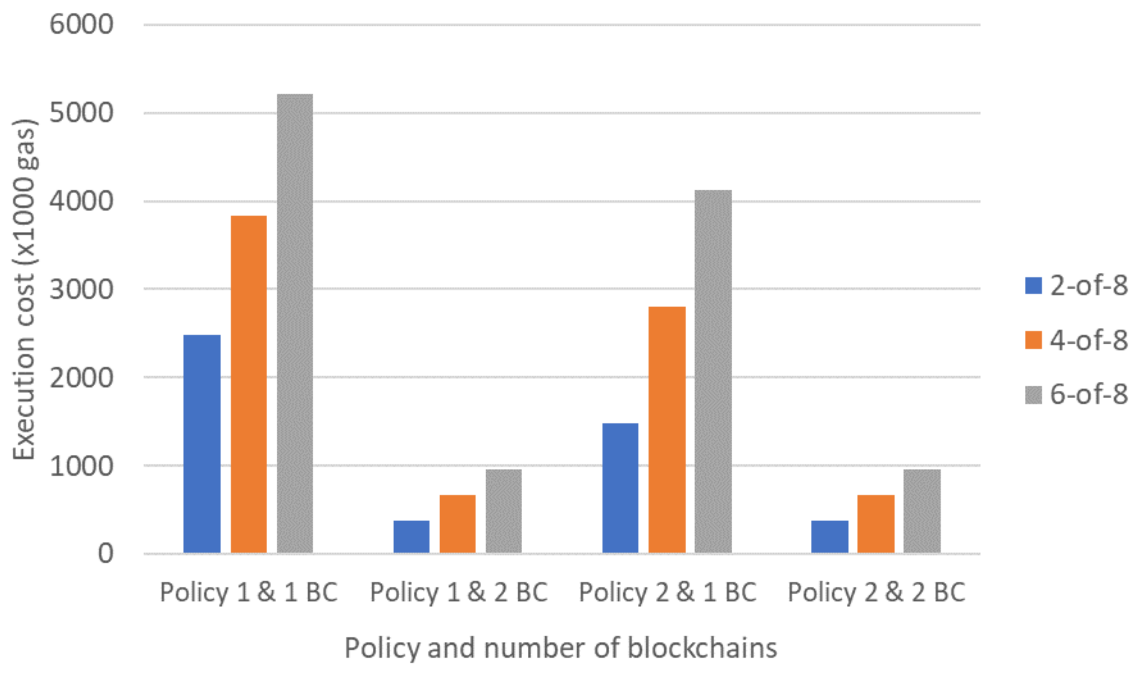


Figure 6: Execution cost (gas) for the two policies, for the case of one and two blockchains. For two blockchains, the cost involves only the cost for transactions on the payment blockchain.

End-to-end delay when Hyperledger Fabric is interconnected with the Rinkeby public Ethereum testnet

Next, we investigate the delay induced when the authorization contract is implemented on the Hyperledger Fabric permissioned ledger, identifying how different implementation choices influence this delay. Fabric's execute-order-commit flow pattern allows faster execution of parallel transactions. However, when concurrent transactions access the same state variable then a commit error may occur. Specifically, if prior to committing a transaction that modifies a variable, a second transaction that modifies the same variable is executed and endorsed, then the second transaction will fail validation and a commit error will occur. Concurrent transactions occur in our scenario, since multiple ASes can respond to an authorization request. One approach to address the issue of concurrent transactions is to add a FIFO (First-In-First-Out) module that serializes the transactions from ASes and sends them to the underlying Fabric system with a small delay between consecutive transactions. A disadvantage with this approach is that the serialization operation adds delay, which increases as the number of ASes that need to respond for authorization increases.

An alternative approach, which is the approach we consider in the evaluation results shown in Figure 7, is to maintain separate state, in the form of a table, where the information submitted by each AS is stored. This table also maintains a boolean variable indicating whether the specific AS has responded to the authorization request. Then, a module external to Hyperledger Fabric periodically reads the aforementioned boolean variables to determine whether the necessary number of ASes has responded; this corresponds to steps 3.1 to 3.m in Figure 5. Once the required number of ASes has responded, the procedure can continue with step 4.

Finally, a third approach is to perform the above functionality in the interledger gateway. The interledger gateway is responsible for reading the price and hash *h* from the authorization chain and submitting it to the payment chain to initiate a payment. After the secret is submitted by the ASes on the payment blockchain, the interledger gateway reads the *m* secrets *s1* to *sm* from the payment chain and submits them to the authorization chain. Moreover, instead of periodically reading the number of ASes that responded as done by the external module in the second approach described above, the interledger gateway can be made aware of the number of ASes *m* that need to respond, hence can wait for *m* events that correspond to the responses 3.1 to 3.m of the *m* ASes in Figure 5. After the interledger gateway receives the m events, the procedure can continue with step 4.

Note that all three of the approaches presented above for handling concurrent transactions from multiple ASes can be used to implement the two policies presented above. Indeed, the performance in terms of delay is the same for both policies.

Figure 7 shows the end-to-end delay when Hyperledger Fabric (authorization blockchain) is interconnected with Rinkeby (payment blockchain), when the second approach described above is used for handling concurrent transactions on Hyperledger Fabric. The 95% confidence interval for these results are less than 2% of the average value shown. Moreover, the figure shows that approximately 16-18% of the total delay is due to Hyperledger Fabric. With the third approach presented above, which requires that the interledger gateway is aware of the number of ASes that need to respond and receives individual event notifications from the ASes that respond to authorization requests, the delay on the Hyperledger Fabric network is further reduced by approximately 28% (Figure 8) and becomes 12.5-13.5% of the total delay.

The above delay results apply to both policies for selecting the subset of ASes that participate in the authorization. However, the simplicity of the second policy allows an alternative implementation that avoids having the client send the request to the smart contract on the authorization chain in step 1 of Figure 5. Instead, the client can send the request directly to all ASes. This variant, which is only possible with the second policy, can further reduce the delay on the Hyperledger Fabric network by approximately 29% (Figure 8).

A conclusion from the above evaluation results is that combining public and private/permissioned ledgers can reduce the overall delay. Moreover, the results also show that the delay on the private/permissioned ledger can be influenced by various implementation choices.

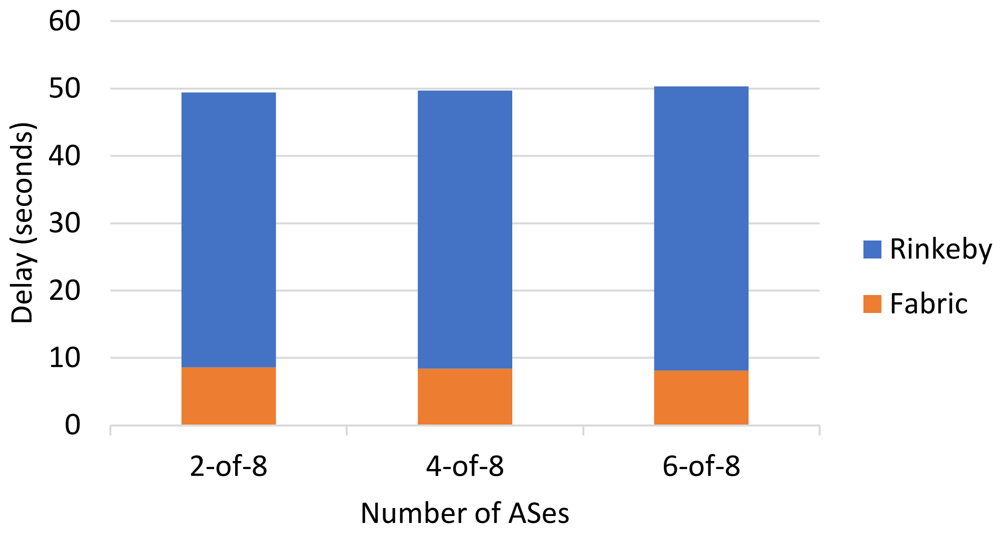


Figure 7: End-to-end delay when Hyperledger Fabric (authorization blockchain) is interconnected with the public Ethereum Rinkeby testnet (payment blockchain).

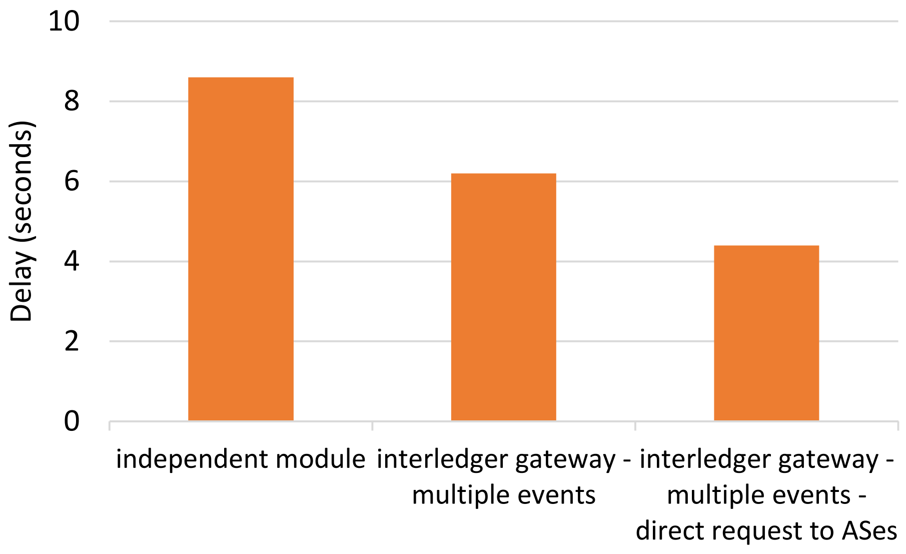


Figure 8: Transaction delay on Hyperledger Fabric (authorization blockchain) for different implementations: i) independent module, ii) interledger gateway receiving individual event notifications when ASes respond, iii) interledger gateway and client sending initial request to the ASes rather than to the smart contract.

5.2 Access control for multi-tenant IoT systems

In this section we present an access control solution for WoT IoT “hubs”, based on VCs. This solution is planned to be included in the IAA component and it is suitable for scenarios where a resource owner owns a few (usually a single) resource, powerful enough to implement public key cryptographic operations.

Our solution considers an IoT system composed of the following entities: an IoT hub connected to multiple IoT resources, a hub *owner* who administrates the hub, and multiple *client* users wishing to gain access to the hub in order to interact with an IoT resource. Moreover, we consider a verifiable data registry built using a permissioned distributed ledger. Owners are the only entities that must be able to perform write operations in the registry, whereas hubs and clients need only read access. For this reason, owners own a wallet used for performing secure transactions with the ledger. Each resource and the hub have their own authentication and authorization systems; hence the owner has an “account” in all these systems. Furthermore, the owner has “linked” the hub account with all other accounts using OAuth 2.0 (or any other similar protocol). In other words, the owner has authorized the hub to communicate with the resources on his behalf. Our goal is to apply the same principle for clients who, however, do not have, neither plan to have, an account with the hub or the resources.

Owners are assumed to maintain a DID in the registry. The key that corresponds to this DID is used for signing the generated VCs.

The WoT paradigm is used for interacting with hubs, hence, each resource is identified by a URI-encoded identifier. Additionally, the properties, actions, and events of a resource are accessed using a REST API. IoT resources use their own (proprietary or open) protocol, therefore, an authorized client sends an appropriate HTTP request to the hub, and the hub translates it to the resource specific message. However, the communication between the hub and the resources is out of the scope of the discussion in this section.

From a high-level perspective, the entities interact with each other as follows (see also Figure 9). Clients ask an owner to access a particular resource. The owner creates and signs a VC that “claims” that a client is authorized to access the requested resource identifier; if required, the VC can define for which specific properties, actions, and events of a resource the client is authorized. The VC is bound to a public key owned by the client, which is used for generating the verifiable presentation (VP) of the VC. Then, the owner sends the VC to the client and stores all necessary auxiliary information in the registry. As a next step, the client performs an HTTP request to the hub, and the hub responds with a VP request. The client creates the VP and sends it to the hub. Finally, the hub verifies the validity of the VP (including that the VC has not been revoked), and if the client is authorized the hub proceeds with the appropriate operations.

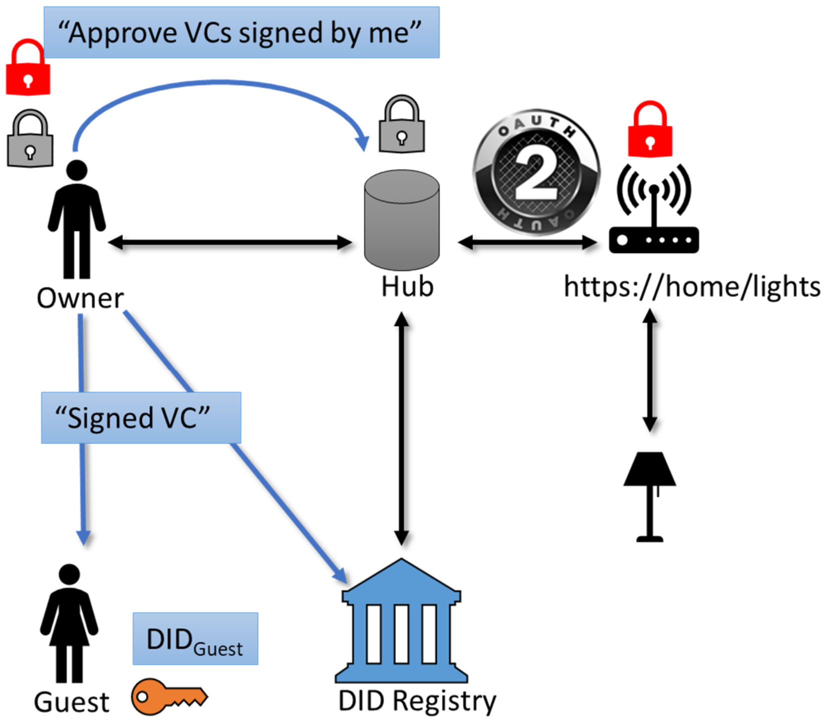


Figure 9: IoT hub access control using VCs.

5.2.1 Evaluation

We have implemented the proposed solution using Hyperledger Indy, and Mozilla's WebThings gateway. In our implementation only owners have "write" access to Indy's blockchain (i.e., the registry). VCs in Indy are based on the privacy preserving Idemix credential system developed by IBM. This credential system enhances the degree of privacy provided to credential holders by enabling pseudonimity: a credential holder is never known by two different parties with the same identity.

For our implementation we have selected to not modify the WebThings gateway, instead we have used a proxy-based approach. In particular, we have implemented a proxy, using Node.js, that intercepts the communication between a guest and the WebThings gateway (i.e., the hub), and implements the requires protocols. Eventually, the proxy forwards all approved requests to the hub. The proxy is configured using a Thing description file that includes elements of the WoT Security Ontology, i.e., for each resource (and optionally for each property, attribute, and event) a security scheme is defined. For example, the following listing contains a snippet of a description file, where access to the toggle action requires a valid VC. In more detail, at lines 4-8 a security method is defined, whereas at line 16 it is declared that this action is protected by the defined security mechanism.

```
{
“context”: “https://www.w3.org/.../v1”,
“id”: “hub.home/switch1”,
“securityDefinitions”: {
    “vcauth”: {
    “scheme”: “bearer”,
    “format”: “cred:VerifiablePresentation”,
    “alg”: “sec:RsaSignature2018”
  }
}
“properties”: {...},
“actions”: {
  “toggle”: {
    “forms”: [
      {
      “security”: [“vcauth”]
      “op”: “invokeaction”,
      “href”: “https://hub.home/switch1/toggle”,
      “htv:methodName”: “POST”
      }
    ]
  }
},
“events”: {...}
}
```

Figure 10: A WoT-based device description that specifies VCs as the access control mechanism.

So far, we have applied the same implementation used for the PDS component AS. Therefore, the performance and security evaluation of Section 4.2.2 also applies here.

5.3 Decentralized interledger gateway architectures

In this section we present decentralized architectures for an interledger gateway (ILG) system that ensure that the interledger operations are performed in a reliable and trusted manner. The functionality of a single ILG can be implemented by the interledger component. The goal of this section is to consider architectures that go beyond that of a single ILG, hence go beyond the design and functionality of the interledger component considered in Sections 3 and 4.

Although the decentralized ILG architectures are presented within the context of IoT authorization scenarios that include the interconnection of two ledgers (an authorization ledger and a payment ledger), they are applicable in a more general context where transactions on multiple blockchains are cryptographically linked with hash-lock and time-lock mechanisms. In the case where transactions on different chains are not linked through such mechanisms, additional mechanisms for verifying that transactions have been performed on different chains would be necessary.

In the case of a single ILG shown in Figure 11, the ILG is responsible for i) listening to events generated on ledger A and ii) submitting transactions to ledger B. An event on ledger A can be generated when the secret to a hash-lock is submitted on this ledger. When the event on ledger A is generated, the gateway can obtain the secret and submit it to ledger B. Submission of the secret to ledger B is performed as a transaction using the account that the ILG has on ledger B. The verification that the gateway receives from ledger B can be a confirmation that the transaction was added to the transaction pool or a confirmation that the transaction was included in a mined block. In the latter case, the verification can be received by having the ILG listen to an event on ledger B that is generated when the block containing the submitted transaction is mined.

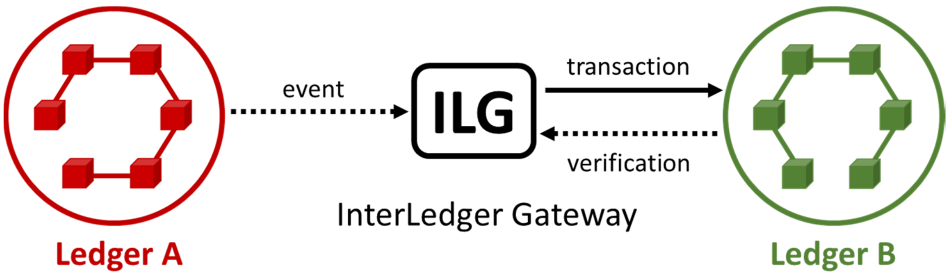


Figure 11: In a single ILG architecture an event notification from ledger A triggers the ILG to submit a transaction to ledger B.

A key problem with a single ILG system is that the ILG is a single point of failure: If the ILG fails, then the interaction between the two ledgers is not possible. Below we present three decentralized ILG architectures that address the aforementioned problem and which differ in their complexity, transaction cost, and ability to handle transactions involving multiple ledgers. The first involves multiple ILGs that are operated by the same organization and the ILGs use the same account for submitting transactions to a ledger. The second architecture involves multiple ILGs that are operated by different organizations and the ILGs use different accounts for submitting transactions to a ledger. Finally, the third architecture utilizes a Hyperledger Fabric permissioned ledger for coordinating the interledger functions.

5.3.1 Multiple interledger gateways operated by the same organization

In this architecture multiple ILGs that are operated by the same organization, as shown in Figure 12, perform the interledger functions that include the following:

- Listen to events on ledger A indicating that the secret for a hash-lock is recorded on ledger A. These events will trigger the interledger operations.
- All ILGs submit transactions to ledger B using the same account.
- All ILGs listen to an event on ledger B to confirm that the transaction has been successfully performed.

Every transaction in Ethereum has a nonce, which corresponds to the number of transactions that are sent from a given account. Each time a transaction is submitted, the nonce value increases by one. Furthermore, transactions from the same account must be ordered according to their nonce values and nonce values cannot be skipped. Because the same account is used by all ILGs for submitting transactions to ledger B, if ledger B is an Ethereum blockchain, synchronization of the nonce values is necessary to satisfy the above rules. Specifically, the ILGs need to use the same nonce in transactions that correspond to the same event (e.g., recording of the same secret) on ledger A. Synchronization of the nonce values can be achieved in a centralized manner by the organization managing the ILGs. Because the ILGs use the same account, even though N transactions are submitted to ledger B, only one will be included in the mined block; hence, the transaction fee will be incurred once. Based on the above, this solution provides decentralization of the interledger operations, but does not provide decentralization at the organization level since the ILGs are managed by a single organization. The architecture presented in the next section provides decentralization of both the interledger functions and the management of the ILGs.

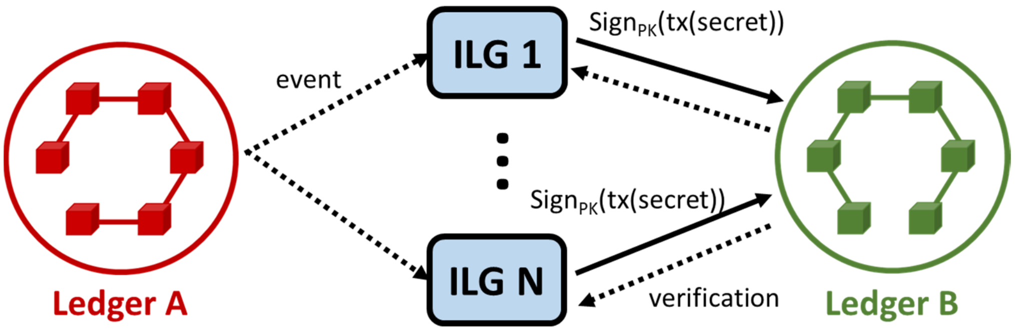

Ledger A
event
ILG 1
ILG N
Sign_{PK}(tx(secret))
Sign_{PK}(tx(secret))
verification
Ledger B

Figure 12: Decentralized interledger gateway architecture when the ILGs are operated by the same organization. When they receive an event notification from ledger A, all ILGs submit a transaction to ledger B using the same account (which corresponds to a public key PK).

5.3.1.1 Management of ILGs

The management of ILGs is performed by a single organization and includes managing the nonce value that the ILGs include in the transactions they submit to ledger B using the same account. The organization managing the ILGs must be trusted in order to achieve reliable interledger gateway operation. Note, however, that in the authorization scenario discussed in this section the interledger operation involves copying a secret from ledger A to ledger B. Copying the correct secret is ensured since ledger B will check that the submitted secret's hash is equal to the hash of the hash-lock in the smart contract on ledger B.

Instead of using the same account on ledger B, each ILG can use a different account. With this approach, synchronization of the nonce values if ledger B is an Ethereum network is not necessary. Moreover, only one (or a few) ILGs can be selected in a centralized manner to submit transactions to ledger B, making this scheme more efficient from this perspective.

5.3.1.2 Reliability

Reliability is achieved by having all ILGs submit a transaction to ledger B, once they receive an event notification from ledger A. Moreover, after they submit a transaction to ledger B, all ILGs listen to an event on ledger B that confirms that the transaction with the hash-lock's secret is included in a mined block of ledger B. In this way, ILGs can verify that the transaction is successfully submitted to ledger B. If the ILGs do not receive an event verifying the submission of the transaction until some timeout, they resubmit the transaction to ledger B.

5.3.2 Multiple interledger gateways operated by different organizations

The decentralized interledger gateway architecture when the ILGs are operated by different organizations is shown in Figure 13. In this architecture, the interledger functions that need to be supported include the following:

- Listen to events on ledger A indicating that the secret for a hash-lock is recorded on ledger A. These events will trigger the interledger operations.
- Selection of one (or more) ILG(s) that will submit transaction(s) to ledger B. These transactions contain the secret copied from ledger A.
- The selected ILG (or ILGs) submits the transaction to ledger B.
- The ILGs verify that the transaction was successfully submitted to ledger B. If the submission is not verified after some timeout, a new ILG (or ILGs) is selected to submit the transaction.

In this model, N ILGs listen for an event on ledger A indicating that the hash-lock's secret has been submitted. The ILGs are operated by different organizations and use different accounts to submit transactions to ledger B. Because the ILGs use different accounts, synchronization of the nonce values if ledger B is an Ethereum network is not necessary.

Once the event notification is received by the ILGs, one of them (ILG k in the figure) is selected to submit a transaction to ledger B containing the secret recorded on ledger A. The selection can be made by taking the modulo N of the last or the next block on ledger B. Specifically, each of the N ILG is assigned a unique identifier from 0 to N-1 and the result of the ledger B's block hash modulo N operation determines the ILG that is responsible for submitting the transaction to ledger B. The hash of ledger B's block must be used because ledger B must also verify that the correct ILG submitted the transaction. Below we discuss alternatives for managing the bindings of ILGs to unique identifiers.

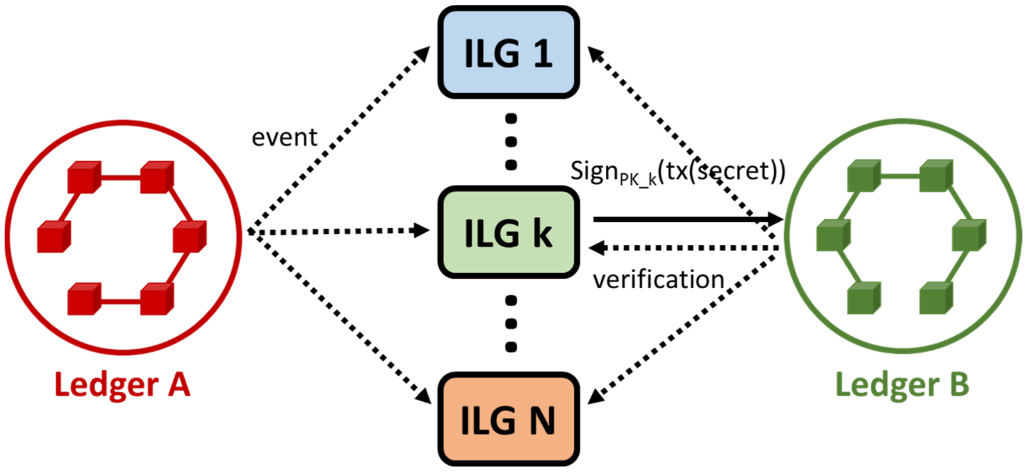


Figure 13: Decentralized interledger gateway architecture where the ILGs are operated by different organizations. One of the N ILGs (ILG k in the figure) is selected to submit the transaction to ledger B. The selection is based on the hash of the last or the next block mined on ledger B. ILGs use different accounts to submit transactions to ledger B.

Next, we discuss the implications from using the hash of the last or the next mined block of ledger B for determining the ILG that is responsible for submitting the transaction to ledger B. If the hash of the last block is used, then anyone can know which of the ILGs is responsible for submitting the transaction to ledger B that corresponds to the next event on ledger A. This allows the possibility of a DoS attack to the ILG responsible for sending the transaction *before* the event on ledger A occurs. On the other hand, if the ILG selection uses the hash of the next block that is mined, then the ILG that is responsible for submitting the transaction on ledger B is known *after* the next block on ledger B is mined. Assuming that the selected ILG submits a transaction *immediately after the next block is mined*, the window for conducting a DoS attack on this ILG can be very small. Finally, we note that the generation of new blocks is determined by miners based on the PoW in blockchains such as Ethereum and Bitcoin. As long as the fees for interledger services are much lower than the block mining fees, the approach described above is not susceptible to attacks by miners withholding new blocks. The requirement that the economic gains from withholding blocks is lower than the mining fees does not hold for high-stake lottery applications, hence mined block hash approaches for generating random numbers are not appropriate in such applications.

Because the N ILGs are operated by different organizations, their trustworthy operation must be ensured, i.e., the system must ensure that an ILG must not submit a transaction to ledger B if it is not entitled to do so. Two observations can be made regarding this issue: First, even if more than one ILGs submit transactions with the hash-lock secret to ledger B, the interledger operation is still executed correctly. Moreover, if ledger B is a public blockchain such as

Ethereum, multiple transactions would incur an execution cost for the ILGs that submit the transactions. Second, checking that the transaction is submitted by the ILG that is entitled to do so is necessary if the ILGs receive some compensation for their services. To achieve this, the smart contract on ledger B must verify that the ILG submitting the transaction is the one that was entitled to do so; the smart contract can perform this verification based on the hash of the last block mined on ledger B and the ILG's ID. Moreover, once the smart contract verifies that the transaction was submitted by the correct ILG, it can proceed to automatically reimburse that ILG for its interledger services.

5.3.2.1 Management of ILGs

Next, we discuss how the ILGs that perform the interledger transactions can be managed. In the scenario described in the introduction that includes linking payments to authorizations for accessing IoT resources, the two parties involved are the client (buyer) and the IoT resource owner (seller). The scenario requires that smart contracts containing hash-locks and time-locks are created on the two ledgers: the payment ledger and the authorization ledger. The interledger services for obtaining authorization to an IoT resource can be managed by the IoT resource owner, which is the interledger services client. This IoT resource owner can determine the ILGs that can provide interledger services. One approach to achieve this is to submit to the smart contract on ledger B the list of ILGs that can submit transactions for interledger services related to the owner's IoT resource. Alternatively, the ILGs can obtain a signed credential from the owner that allows them to submit transactions to the smart contract on ledger B. Such credentials can be based on the Verifiable Credentials Model developed by W3C's Credentials Community Group [Spo++19]. The credential can be submitted inside each transaction and the smart contract on ledger B can verify the IoT resource owner's signature to ensure that the ILG submitting the transaction has authorization from the owner. The two options for determining the set of ILGs that can submit interledger transactions differ in terms of privacy and the operations and cost for performing revocation. Namely, recording the set of legitimate ILGs in the smart contract allows all entities that have read access on the ledger, which in the case of public ledgers is anyone, to know the ILGs that provide interledger services for a particular IoT resource. Revocation, i.e., removing an ILG from the list of ILGs, would incur the cost of a transaction to update the list in the smart contract. On the other hand, using credentials has higher privacy, since in this case the smart contract does not contain the list of ILGs. However, the credential-based approach requires different actions to perform revocation. One approach is to have credentials with fixed time validity, which implies that they must be periodically renewed for an ILG to continue to provide interledger services.

An additional management task is the assignment of unique numbers 0 to N-1, which is necessary for the ILG selection discussed above. The assignment needs be known, in a reliable manner, by the smart contract running on ledger B. This can be achieved by including the number assigned to each ILG in the list maintained by the smart contract or include the number in the credential that the interledger services client provides to each ILG.

5.3.2.2 Reliability

After an event is generated on ledger A, all ILGs, including the ILG that was selected to submit the transaction to ledger B, listen to an event on ledger B that verifies that the transaction with the hash-lock's secret is included in a mined block of ledger B. In this way, ILGs can verify that the transaction is successfully submitted to ledger B. If the ILGs do not receive an event verifying the submission of the transaction until some timeout, then a new ILG can be selected, using the same procedure as the one described above, which will (re)submit the transaction to ledger B.

Above, we assumed that one ILG is selected for submitting the transaction to ledger B. Alternatively, more than one ILGs can be selected and submit transactions from their accounts. Such an approach can yield a smaller delay for completing the interledger operations, if the probability of faulty or misbehaving ILGs is high. The tradeoff is that if more than one ILGs are selected then, in the case of the Ethereum blockchain, the total transaction cost would increase with the number of submitted transactions. One way to reduce this cost in Ethereum is for the smart contract to use Solidity's Revert call, with which duplicate transactions incur only the cost (gas) of the transaction invocation.

5.3.3 Use of Hyperledger Fabric in the interledger gateway system

In this section we discuss the use of Hyperledger Fabric, which is a permissioned distributed ledger that is part of the Linux Foundation's open-source Hyperledger project, for coordinating the interledger operations. The advantages of using a permissioned ledger for interledger operations are the following:

- A permissioned ledger can implement elaborate consensus logic and rules that can jointly consider events from different ledgers. This is necessary if the interledger operations involve more than copying a hash-lock secret between two ledgers.
- A permissioned ledger can record, in a reliable and immutable manner, transactions across different ledgers.
- Even though some of the logic can be implemented in ledger B (provided it supports smart contracts), the execution cost when the logic is implemented in a permissioned ledger will be significantly lower compared to the cost if ledger B is a public ledger.

These advantages allow a permissioned ledger such as Hyperledger Fabric to serve as a hub for interledger services among multiple ledgers. The above advantages are achieved with a higher complexity compared to the two previous architectures, since Fabric is used for coordinating the functionality of the interledger mechanisms.

Figure 14 shows the architecture for an interledger system based on Hyperledger Fabric. Observe that the functionality of the interledger gateways of the first two architectures has been separated. The architecture shown in this figure has two types of ILGs: level 1 ILGs listen to events on Ledger A and level 2 ILGs submit transactions to ledger B. Hyperledger Fabric can implement elaborate consensus rules that can include selecting one or more level 2 ILGs that are responsible for submitting transactions to ledger B. From an implementation perspective, level 2 ILGs can run on the same nodes where Fabric peers reside. Similarly, level 1 ILGs can also reside on Fabric peer nodes.

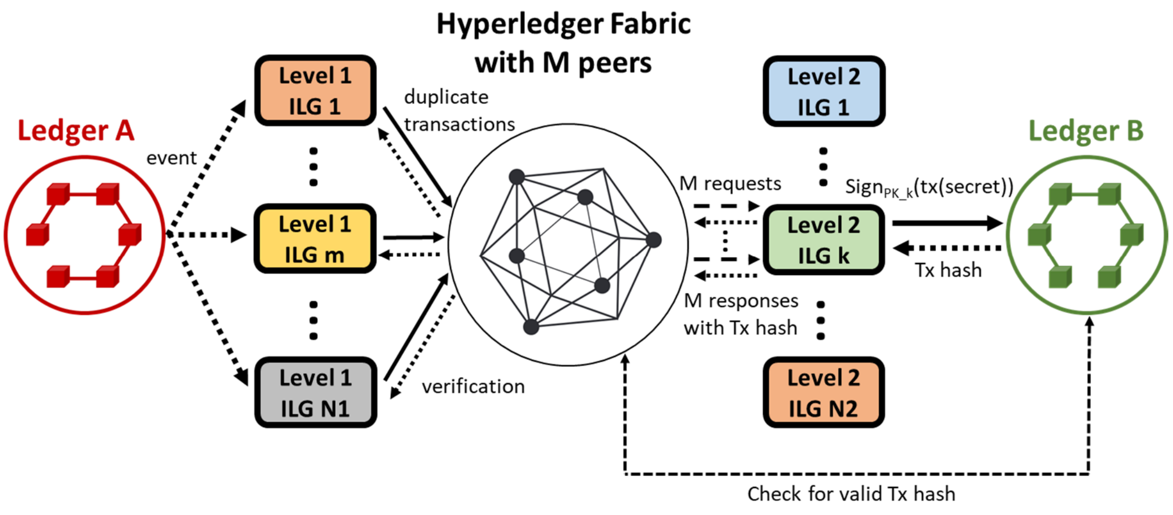


Figure 14: Use of Hyperledger Fabric in the ILG system. Compared to the previous two architectures, the interledger functionality is now split: Level 1 ILGs listen to events on ledger A and level 2 ILGs submit transactions to ledger B.

5.3.3.1 Management of ILGs

In this architecture the management of ILGs can be performed as in the previous architecture. A difference is that the Fabric peers now select the level 2 ILG that will be responsible for submitting the transaction to ledger B. In this step we utilize the capability of Fabric peers that all execute the same smart contract (called chaincode in Fabric) to call an external API. Hence, in the above figure the selected level 2 ILG receives a call from all M Fabric peer nodes.

5.3.3.2 Reliability

The reliability of the functionality of level 1 ILGs, namely of sending a transaction to the Fabric network when an event occurs on ledger A, is ensured by having all level 1 ILGs send a transaction to Fabric (denoted duplicate transactions in the figure). Fabric can handle multiple concurrent transactions but cannot handle concurrent transactions that affect the same key in the ledger. Hence, only one transaction from Level 1 ILGs will eventually be valid and the others will cause Multi Version Concurrency Control (MVCC) failures. Nevertheless, only one valid transaction is necessary for the interledger functionality to operate correctly, hence the above behaviour is sufficient. Note that, since this section focuses on interledger functions that involve copying hash-lock secrets from one ledger to another, it is not necessary for the Fabric network to verify the transaction on ledger A that reveals the hash-lock secret; if the secret was not revealed on ledger A, then the corresponding transaction on ledger B would fail.

The reliability of submitting transactions to ledger B is achieved differently than in the approach discussed for the previous architecture, since in this model the Fabric peers select the ILG that will submit the transaction to ledger B. Specifically, recall that all Fabric peers select the same level 2 ILG to submit the transaction to ledger B, hence the selected level 2 ILG receives a call from all M Fabric peer nodes. Once this level 2 ILG submits the transaction to ledger B, it returns the transaction hash (Tx hash in the figure) to the Fabric peer nodes. The Fabric peer nodes check ledger B to verify that a valid transaction with the transaction hash (Tx hash) has indeed appeared in a mined block. If the transaction is not verified, then the Fabric peer nodes select a new level 2 ILG to submit the transaction. The Fabric peer nodes also select a new level 2 ILG to submit the transaction to ledger B if the original level 2 ILG does not respond after some timeout.

Finally, as in the previous architecture, more than one level 2 ILGs can be selected for transmitted transactions to ledger B. This can increase the reliability and reduce the delay, at the expense of a higher transaction cost on ledger B.

5.3.4 Evaluation

The evaluation results presented in this section consider the architecture shown in Figure 13. The evaluation setup consists of the Rinkeby public Ethereum testnet that was ledger B in the figure. We used the Infura Ethereum node cluster for submitting transactions to the Rinkeby. For ledger A in the figure we used Hyperledger Fabric. We consider the case of 3 ILGs and investigate the reliability of the interledger functionality that the architecture shown in Figure 13 can support in the presence of ILG errors.

Figure 15 shows the interledger delay for different ILG error percentages. The results are the average from 20 executions. Also shown are the 95% confidence intervals. The interledger delay is the time interval from the point an event on ledger A (Hyperledger Fabric) is generated until the time that a transaction is successfully submitted to ledger B (Ethereum). Since ledger B is an Ethereum network, the interledger delay is determined by the Ethereum transaction delay, which depends on the Ethereum block mining time. Hence, for zero errors, the figure shows that the delay is approximately 15 seconds, which is the average time for mining a new Ethereum block. In the presence of errors, Figure 15 shows that the delay increases by approximately 70% and 270% when the percentage of ILG errors is 20% and 50%, respectively. The transaction delay in the presence of ILG errors can be reduced by having more than one ILGs submit interledger transactions to ledger B.

The delay results shown in Figure 15 also pertain to the architecture in Figure 12, since the interledger delay is determined by the transaction delay on ledger B and the differences of the interledger operations in Figure 12 and Figure 13 do not influence this delay.

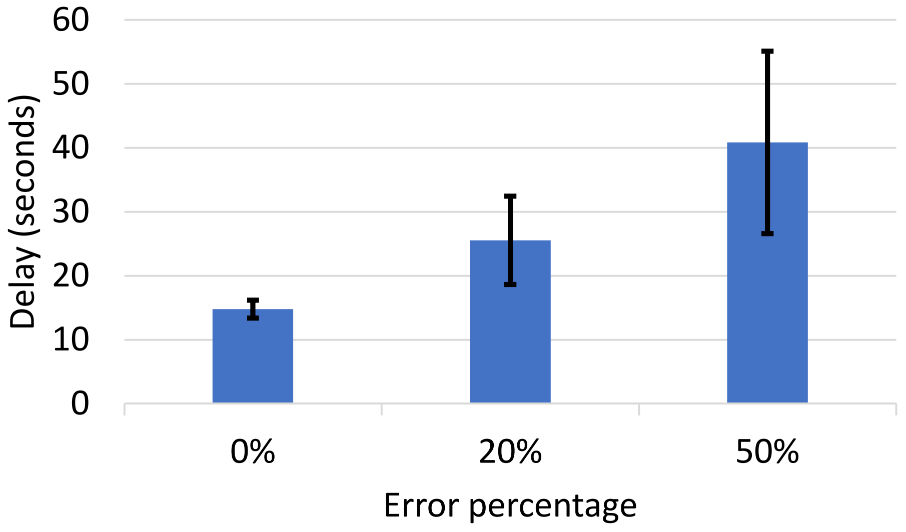


Figure 15: Interledger delay for the architecture in Figure 13 with three ILGs in the presence of ILG errors.

Table 25 shows the gas, which quantifies the amount of EVM (Ethereum Virtual Machine) resources, for the transaction on ledger B (Ethereum testnet) that the ILG submits for three cases: i) the architecture in Figure 12, ii) the architecture in Figure 13, and iii) the architecture in Figure 13 when a revert is called in the transaction. The transactions are submitted with gas price 2.5 Gwei. The results show that the transaction cost for the architecture in Figure 13 is higher than the architecture in Figure 12. This occurs because of the additional checks in the latter architecture that is necessary to ensure that the correct ILG has submitted the transaction. Specifically, we assume that the smart contract has a table containing the address of the ILGs that can submit an interledger transaction and their corresponding identifiers. The specific ILG that should submit the transaction is selected as described in Section 5.3.2, and is based on the modulo 4 (since the number of ILGs is 3) of the hash of the last block on ledger B.

The table also shows that the transaction cost with a revert is less than 40% of the transaction cost without a revert (for the architecture in Figure 13). The revert can be used when more than one ILGs are selected to submit transactions to ledger B, in order to increase the reliability and reduce the delay of the interledger operation.

Table 25: Ledger B (Ethereum) transaction cost (gas)

Transaction	Cost measured in gas
Architecture in Figure 12	64 425
Architecture in Figure 13	66 968
Architecture in Figure 13 with revert	27 132

6 Evaluation scenarios

This section considers the SOFIE pilots and generalizes them into pilot-inspired scenarios by including alternatives not selected in the SOFIE pilots, while also abstracting various aspects to an appropriate degree, so that they can be emulated and/or simulated. Our focus is on identifying and quantifying the various tradeoffs of many potential alternative design decisions and the impact of various system parameters on the resulting performance. We relate the specific use cases considered to pilot requirements identified in deliverable D5.2 (Initial Platform Validation) and our performance results are presented in terms of the pilot-specific system performance KPIs and targets, which were initially presented in the revision of deliverable D4.3 (First Architecture and System Evaluation Report) and included in Section 2.2 of this deliverable.

6.1 Food Supply Chain

The Food Supply Chain (FSC) pilot aims at leveraging distributed ledgers to provide traditional supply chains with reliable and tamper-proof provenance and tracing data. More specifically, it targets an FSC transferring agricultural products from producers to supermarkets and it aims at providing the following features:

- Traceability of agricultural products from the producer to the consumer
- Traceability of transport and storage conditions
- Resolution of disputes in case of customer complaints

The new results contained in this deliverable compared to the first evaluation deliverable D4.3 include two new emulation scenarios. The first one employs four distinct private Ethereum blockchains, each registering the handovers between two consecutive stages, while the second one considers the case where each stage utilizes a custom local storage, and periodically stores hashes of newly added data to a public Ethereum ledger. Notably, our emulation is substantially extended from evaluating individual transactions (like handovers, or sensor logs) in D4.3 to providing an end-to-end evaluation of the entire FSC, and comparing all scenarios with respect to their cost (in gas and euros) and their throughput.

6.1.1 Overview

This pilot assumes smart boxes (or, simply, boxes) as the end-to-end unit of transfer. That is, products are packaged into boxes by the producer and they remain in these boxes throughout the entire transfer until they reach the consumer. Each box is equipped with an RFID tag, which is scanned and registered when the box is handed over by one stage of the supply chain to another.

The chain consists of five stages, shown in the following table.

Table 26: The stages of the FSC pilot use case

Stage no.	Stage name	Abbreviation	Role
1	Farm (Table Grapes Field)	TGF	Grows table grapes and packs them into boxes
2	Transport A	TRA	Transfers boxes from TGF to SDC
3	Depot (Storage & Distribution Centre)	SDC	Collects, stores, and dispatches boxes
4	Transport B	TRB	Transfers boxes from SDC to SM
5	Supermarket	SM	Displays boxes and sells them to consumers

Figure 16 Illustrates the FSC model described above. Each stage of the FSC is equipped with a number of sensors, periodically reporting data concerning the conditions at this stage.

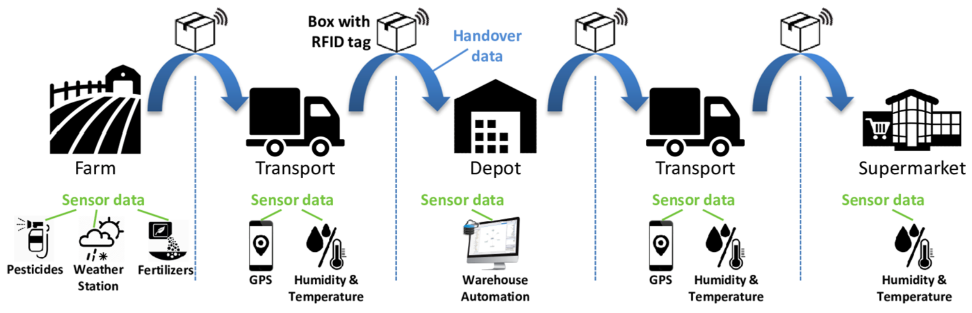


Figure 16: Food Supply Chain with five stages.

Data collected, recorded, and processed in our FSC model can be split into three types:

- Box data constitutes the first type. It concerns all data related to individual boxes in the FSC, such as a box' entry in the first stage to start a new transport session, its consecutive handovers between adjacent stages, and its exit from the FSC when this session is complete. Metadata of all aforementioned actions are also part of this data type, such as which employees were involved in each action, at which time each action happened, and some box state at that point, such as the box' weight. Handover data are recorded on demand, as soon as a handover action takes place.
- Sensor data forms the second type. It refers to all environmental and location data collected by each individual stage concerning the conditions that may affect produce quality. Sensor data are recorded periodically.
- Anchoring metadata constitutes the third type. Anchoring per se does not directly represent any state or action taking place in our model. Instead, it refers to a series of block hashes of any private ledger(s) employed in our model, which is expected to be stored in a public ledger to provide strong immutability guarantees. More on anchoring in the following section.

6.1.2 Architecture space

We consider four different architectures for the proposed FSC framework. Two of them use a single public ledger, while the other two employ hierarchical designs involving a combination of public and private ledgers.

More specifically, we consider the following four architecture scenarios:

Scenario 1 – Public ledger: In this scenario (Figure 17), both handover data and sensor readings, are directly stored in a public ledger. This is a straightforward architecture, inherently guaranteeing immutability and trust among chain members. However, as we will see in our evaluation results, the large volume of data to be stored place an enormous burden in terms of cost and increase delay.

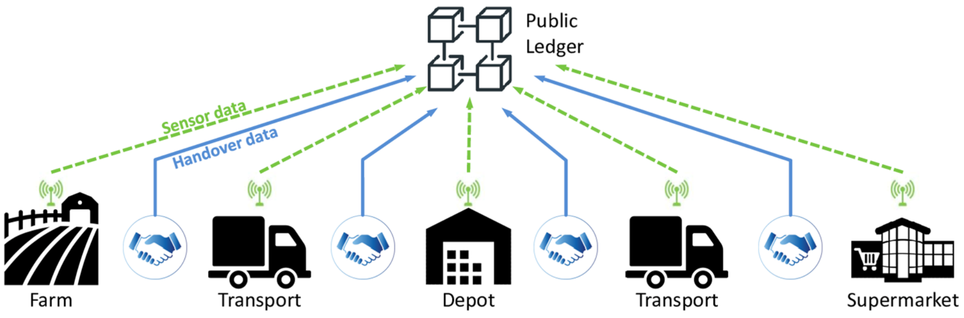


Figure 17: Scenario 1 – Public ledger: All sensor data (dashed green lines) and handover data (solid blue lines) are registered in a public ledger.

Scenario 2 – Single shared ledger: Architecturally this scenario (Figure 18) is identical to the previous one, other than using a shared private ledger (run collaboratively by all chain members) in place of a public ledger, to avoid high costs and delays. A public ledger, however, is still needed to add strong immutability guarantees to the private ledger. More specifically, the private ledger's latest block hash is periodically stored on the public ledger to strengthen the former's immutability, a process referred to as anchoring.

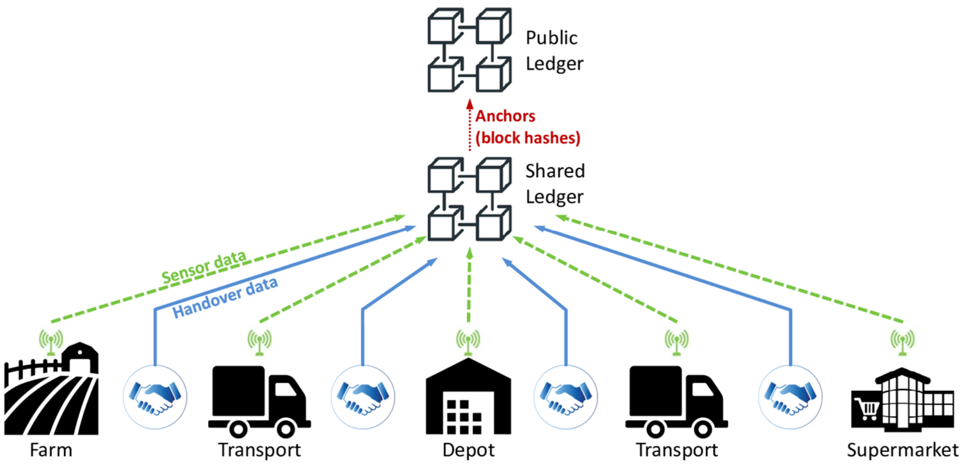


Figure 18: Scenario 2 – Single shared ledger: All sensor and handover data are registered in a shared ledger operated by the entire consortium.

Scenario 3 – One private ledger per pair: This scenario (Figure 19) employs multiple private ledgers, one per pair of adjacent chain stages. It improves on Scenario 2 with respect to data privacy, as well as in overall throughput, as we show in the evaluation results. Anchoring to a public ledger is necessary here too, to guarantee immutability.

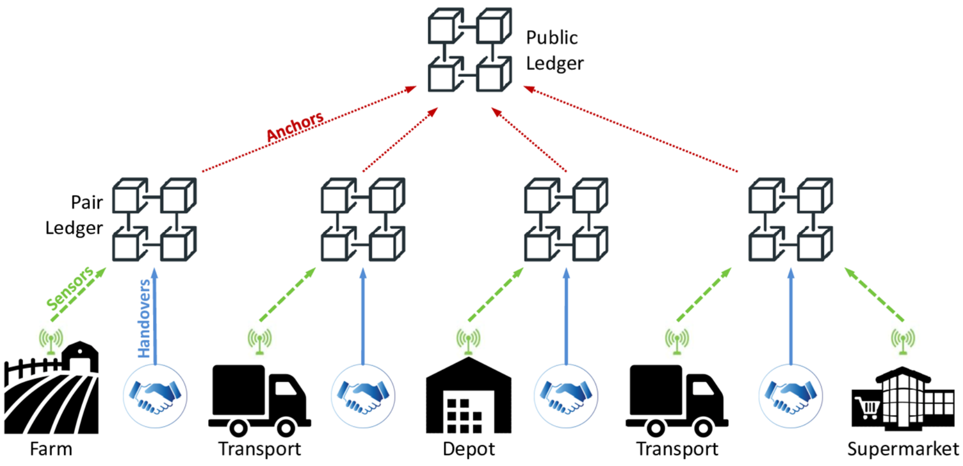


Figure 19: Scenario 3 – One private ledger per pair: Each pair of consecutive stages maintain a separate ledger for recording box handovers between themselves.

Scenario 4 – Private storage: This scenario (Figure 20) maximizes privacy with respect to sensitive data, by each business entity storing all their data in private storages. These storages need not be ledgers (although they could be). They can be local databases, cloud storage, or even local permissioned ledgers. In the absence of a ledger to store mutually approved handover transactions between adjacent stages, handover records should be signed by both stages involved, and stored individually by both. In order to guarantee immutability of private storages, each stage is responsible to implement anchoring for their private storage by periodically storing a hash digest (Merkle tree root or alternative cryptographic tool of their choice) of their contents.

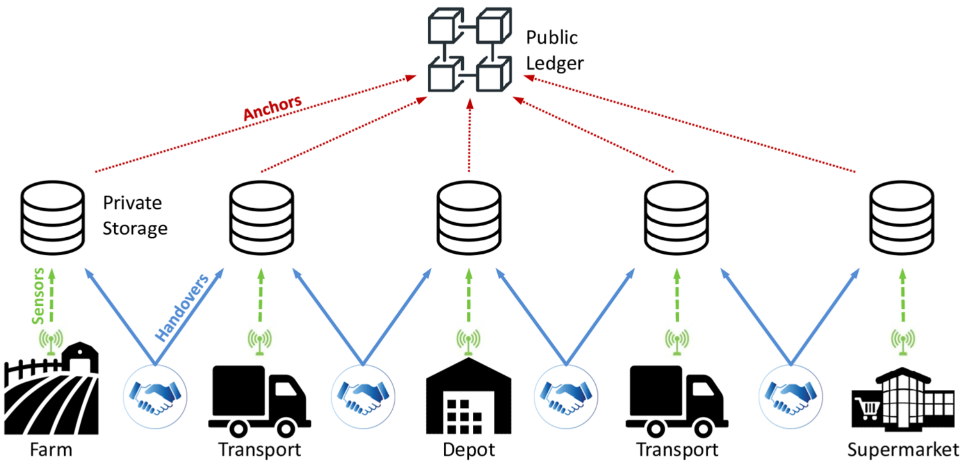


Figure 20: Scenario 4 – Private storage: Each stage maintains their own private storage.

Clearly, an unlimited number of architecture designs can be devised, either as combinations of the above, or by introducing completely new schemes. However, the selected scenarios form a wide range of clearly defined baselines, whose evaluation identifies their strengths and weaknesses, and help us make educated decisions in using them as is or in combinations.

6.1.3 Emulation setup

We evaluated and compared all four scenarios, implementing the required smart contracts in Solidity (the mainstream Ethereum language) and deploying them on Ethereum. More specifically, we spawned our own private Ethereum instances for Scenarios 2 and 3, while we used the Ethereum Ropsten testnet as a public ledger.

We made some configuration decisions to reduce the parameter space and to allow for a fair comparison. With respect to the parameters of our local Ethereum instances, we fixed the average block mining time to 15 seconds, and we set the block gas limit to 10,000,000 gas units. Both values reflect the respective values in the public Ethereum main net.

With respect to our implementation, we fixed the data schema of transactions across all four scenarios, as well as the lengths of several fields. We also fixed both the anchoring period and the sensor logging period to 5 minutes, for all stages in all scenarios.

The following five types of transactions, grouped by the type of data they produce, form the basis for the FSC functionality supported in all considered scenarios.

Table 27: Transaction types.

Type of data	Transaction	Description
Box	Box entry	Marks the start of a new session for a given box and assigns this box to a given farmer.
	Handover	Hands over a box from one stage to the next, recording the two employee IDs involved, the box weight, and a timestamp.
	Box exit	Marks the end of the current session of a given box. Called when the box is emptied and ready for a new session.
Sensor	Logging	Logs a stage's sensor readings to the ledger. Applied periodically, every 5 minutes.
Anchoring	Anchoring	Stores a private Ethereum's latest block hash into the public Ethereum. Called periodically, every 5 minutes.

Finally, Table 28 below, shows, for each of the basic transactions, the fields it stores in the ledger. Note that we assumed a uniform format across all four handovers, and a uniform length across all five stages of the sensor logging transactions.

Table 28: Fields associated with each transaction

Box Entry	**Handover**	**Box Exit**	**Sensor Logging**	**Anchoring**
Session ID (256)	Empl ID (32)	Empl ID (32)	Sensor data (256)	Block hash (256)
Empl ID (32)	Empl ID (32)	Time (32)		
Time (32)	Weight (32)			
	Time (32)			

6.1.4 Emulation results

6.1.4.1 Transaction cost evaluation

We start our evaluation by focusing on the transaction cost associated with each scenario for executing smart contracts.

In Ethereum, each call to a smart contract function incurs a cost to the caller, measured in gas units. Gas has no fixed monetary value. Instead, when calling a smart contract you are expected to specify the rate (in ETH, i.e., the Ethereum coin) you are willing to pay per gas unit to whoever mines a block that includes your transaction. This policy serves as an incentive for miners to include your transaction.

In Figure 21 we present the gas cost of an individual call to each basic operation. Implementations are similar across different scenarios. However, box entry and handovers are slightly more costly when multiple ledgers are used (Scenario 3), as the box ID has to be stored anew with each handover.

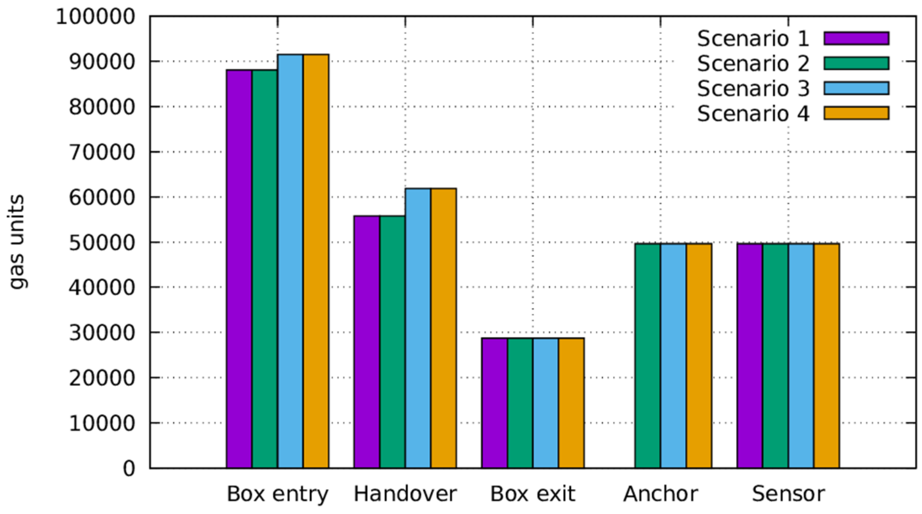


Figure 21: Isolated cost (in gas) for single execution of basic operations.

Figure 22 shows the aggregate gas spent for an entire route of a single box through the FSC. Only the costs directly associated with tracing a box are included (i.e., box entry/exit and the four handovers), leaving out the costs of periodic anchoring and sensor logging. As expected, the first two scenarios exhibit precisely the same costs, in gas. The third scenario incurs a slightly higher overall gas cost, however this cost is distributed among the four pairs of ledgers.

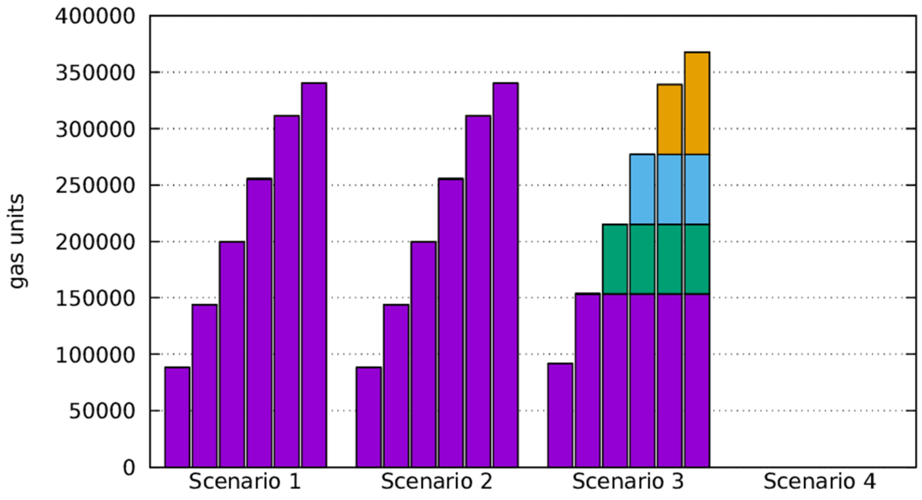


Figure 22: Aggregate cost (in gas) for a full session of a single box through the FSC. Colour coding denotes which ledger each cost fraction refers to. This corresponds to one ledger for Scenarios 1 and 2, and to four ledgers (hence the four different colours) for Scenario 3.

It is important to realize that only the gas spent in Scenario 1 translates into actual monetary value (in ETH, and indirectly in EUR). Gas spent on privately managed Ethereum instances can be paid for with ethers collected through very lightweight mining or an ether faucet. However, as we will see below, the gas required by each operation has an effect on the overall throughput of the system in all scenarios, as each mined block can include a limited amount of gas.

Figure 23 shows the public ledger gas costs associated with each periodic action (anchoring and sensor logging). Note that Scenario 1 incurs periodic costs only for sensor logging, while anchoring is irrelevant to it as data is recorded directly in the public ledger. In contrast, Scenarios 2, 3, and 4 incur only periodic anchoring costs to the public ledger, as all sensor logging takes places in private ledgers or private storage.

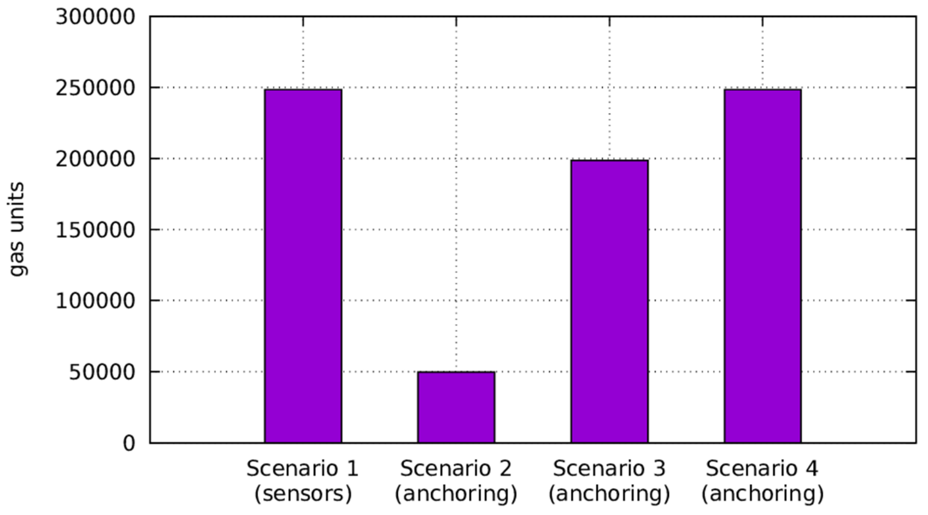


Figure 23: Public ledger gas costs incurred every 5 minutes due to periodic operations. For Scenario 1 this applies to the five sites directly recording sensor readings on the public ledger. For the other scenarios it applies to periodic storing of block hashes on the public ledger (anchoring).

Table 29 summarizes the costs presented above, giving an estimate of the total cost for a full day's (24 hours) operation in each scenario. We assume a typical daily turnover of 6000 boxes. For the conversion of gas to ether, we assume a price of 10^{-8} ETH per gas unit (referred to as 10 nanoether or 10 gwei). Finally, for the conversion to euros, we assume the price of €200 per Ether, an average price for the last few months.

Table 29: Indicative costs for full-day operation

	Cost for 6000 boxes			Full-day periodic costs			Total
	Gas	Ether	EUR	Gas	Ether	EUR	EUR
Scenario 1	2040M	20.4	€4080	71.5M	0.715	€143	€4223
Scenario 2	0	0	€0	14.3M	0.143	€28	€28
Scenario 3	0	0	€0	57.2M	0.572	€114	€114
Scenario 4	0	0	€0	71.5M	0.715	€143	€143

Finally, Figure 24 illustrates these costs as a function of the number of boxes processed through the supply chain.

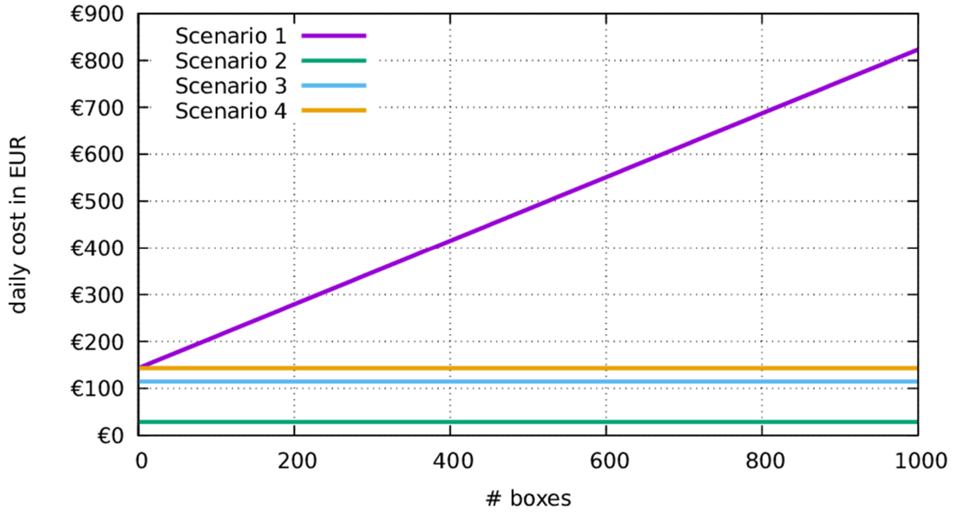


Figure 24: Total cost (in EUR) for a full day's run in each scenario, as a function of the number of boxes transferred. Calculated based on an exchange rate of 200 EUR per 1 ETH, and a gas price of 10^{-8} ETH per gas unit.

6.1.4.2 Throughput evaluation

Cost is not the only concern when employing blockchains in a real-world application. The time to execute a transaction, and, in particular, the volume of transactions the system can process in a given time window (i.e., the system's throughput), may be a deciding factor on designing its architecture.

The limitations on transaction delay and throughput are explicitly imposed by blockchain rules that prevent the creation of arbitrary-size blocks. In the case of Ethereum, this limit is imposed by means of a maximum amount of gas a miner is willing to pack in a single block. Although this is a miner-specific parameter rather than a globally configured one, in main net public Ethereum it has converged to a maximum of 10 million gas units per block. In that regard, optimizing smart contract functions to spend less gas allows us to fit more calls per block, thus increasing the overall throughput.

To assess the throughput of different scenarios, we ran experiments in which we submitted 1000 boxes at once at the beginning of the FSC, and we let the system run as fast as possible to record the rate at which boxes go through the chain. In these experiments, we emulated all harvesting, transportation, storage, and selling times to be zero, in order to assess the net delays imposed by our tracing system.

Figure 25 presents results of these experiments, namely the number of boxes that have traversed the entire FSC as a function of time. The stepwise shape of these plots is due to the grouping of multiple transactions in blocks, which are being generated roughly every 15 seconds. As each box needs six transactions to traverse the entire FSC, no box makes it to the end before the sixth block (at ~90 seconds).

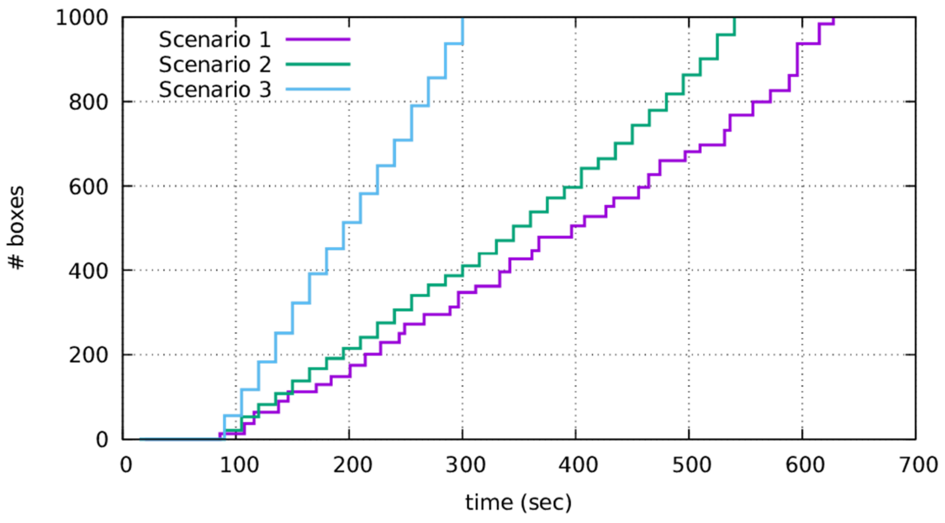


Figure 25: Box handling throughput

Scenario 3 has a clear advantage over the rest, with a throughput of around 285 boxes per minute, while Scenarios 1 and 2 perform at around 111 and 133 boxes per minute, respectively.

Although intuitively one might expect Scenario 3's throughput to be four times as high as that of Scenario 2, given the fact that transactions are distributed across four private ledgers instead of just one, it is in fact just over twice as high. The reason for this is that in queuing systems throughput is governed by the speed of the slowest component in a chain, i.e., its bottleneck. In this case, Scenario 3's bottleneck is the leftmost pair ledger (shared between stages 1 and 2), as it registers both box entry transactions for registering new boxes, as well as handovers from stage 1 to stage 2. As shown in Figure 22, the total gas used on the single shared ledger of Scenario 2 (which is what determines how many transactions fit in one block) is a bit over twice as much as the total gas spent on the leftmost pair ledger of Scenario 3, a ratio reflected on their respective throughputs.

Finally, Scenario 1 seems to perform slightly worse than Scenario 2. This is because it ran on a public Ethereum instance, Ropsten, where our transactions were not guaranteed to be included in the next block as they were competing against other users' transactions.

6.1.5 Pilot requirements in emulation scenarios

The following table, based on Table 5 of D5.2, lists the requirements for the FSC pilot, and which of our emulation scenarios address each of them. Note that many of the requirements refer to specific features the final pilot application should support and are orthogonal to the performance evaluation done through emulation. Such requirements will be investigated and demonstrated in the pilot validation of WP5 and are marked as "Not Applicable".

Table 30: FSC requirements and emulation scenarios

ID	Name	Description	Emulation Scenarios
REQ_FSC0.1	FSC Web application	The services must be provided (to the actors) through the same web application.	Not Applicable
REQ_FSC0.2	RBAC over provided services	The services must be accessible (by the actors) under a Role-based Access Control (RBAC) policy.	Not Applicable
REQ_FSC0.3	Actors unique identifiers	Each actor must be identified in a unique way	All scenarios

REQ_FSC0.4	IoT environments unique identifiers	Each federated IoT environment must have a unique identifier in the system architecture.	All scenarios
REQ_FSC0.5	Authentication management	Authentication and access control logic must be applied to common storage resources.	Not Applicable
REQ_FSC1.1	Timestamped crop registration	Registration of a crop must be timestamped.	All scenarios
REQ_FSC2.1	Farming (meta)data of boxes	The QR code that summarizes product history must include farm location, harvesting date, used fertilizers (dates), and the type of the product (from the perspective of the farming system),	Not Applicable
REQ_FSC3.1	Record of handovers	Handovers must be recorded in an immutable way where all federated IoT environments must have access.	All scenarios
REQ_FSC3.2	Sealing of boxes	The boxes could be sealed upon the delivery to the transportation company (from the producers).	Not Applicable
REQ_FSC4.1	Unsealing Box(es) at the WH	Upon delivery to the WH employee, boxes could be unsealed by the TR employee.	Not Applicable
REQ_FSC5.1	Box unique identifier	Each box must have a unique RFID tag identifier.	All scenarios
REQ_FSC5.2	Boxes as Things	Boxes must be considered as things of the transportation IoT platform.	All scenarios
REQ_FSC5.3	Box registration	Box registration in the supply chain must define also the producer from whom it will be used.	All scenarios
REQ_FSC5.4	Timestamped box registration	Registration of a box must be timestamped.	All scenarios
REQ_FSC6.1	Transportation truck connectivity	Transportation trucks must have internet connection to communicate and exchange data with the transportation IoT platform.	All scenarios
REQ_FSC6.2	Use of transportation truck	A TR employee (driver) must be able to use different transportation trucks on different occasions.	Not Applicable
REQ_FSC7.1	Local storage of IoT data	Measurements from IoT devices are stored locally in the corresponding IoT platform.	Scenario 4
REQ_FSC8.1	Unsealing Box(es) at the SM	Upon delivery to the SM employee, boxes could be unsealed by the TR employee.	Not Applicable
REQ_FSC9.1	Tracking warehouse conditions	The temperature within each storage room of the WH must be continually monitored.	All scenarios
REQ_FSC9.2	Warehouse alarms	In the WH, a notification appears in the monitoring service of the Aberon IoT platform each time a predefined temperature range is violated.	Not Applicable
REQ_FSC10.1	Packetizing products	The (unreleased) boxes in the WH must contain either raw or packetized products.	Not Applicable
REQ_FSC11.1	QR code creation	QR codes must include data which is collected from the federated IoT environments, as well as provided by the actors through the FSC web application	Not Applicable
REQ_FSC11.2	QR labels of packets	The same QR label must be attached to every packet containing grapes which were transferred into the same box.	Not Applicable
REQ_FSC11.3	Vocabulary of QR labels	Labelling of products must be based on a common vocabulary for the food supply domain that	Not Applicable

		maximises reuse of data and acceptance by the customers.	
REQ_FSC11.4	Self-contained QR codes	The QR codes must be self-contained, so Internet connection is not needed to read their content.	Not Applicable
REQ_FSC11.5	Information recorded in QR codes	The QR codes must contain product information related to all the segments of the chain.	Not Applicable
REQ_FSC12.1	Boxes reuse	Boxes must be able to be re-used in the future (to carry other products) after they have been released of the current transfer.	All scenarios
REQ_FSC13.1	QR code reading	QR labels must be accessible by everyone by using a smartphone device.	Not Applicable
REQ_FSC14.1	Traceability of historic data	In the case of an audit, requested organizations must be able to provide proof of their claims about the historic data of assets which are stored locally.	All scenarios
REQ_FSC14.2	Timestamps of handovers	Transfer of responsibility over boxes (assets) must be timestamped.	All scenarios
REQ_FSC14.3	Confirmation of transactions	A transaction must be confirmed by both transacting parties.	All scenarios
REQ_FSC14.4	Retrieve of past transactions	Both parties of a transaction must be able to access the details of the transaction at any time.	All scenarios

6.1.6 Pilot system performance KPIs in emulation scenarios

The following table, based on Table 4 of this deliverable, summarizes our emulation results regarding the respective pilot system performance KPIs.

Table 31: System performance KPIs for the FSC emulation scenarios

KPI	Name	Description	Metric	Method of measurement	Target	Results
KPI_FSC_1	Ledger execution cost in public ledger	Cost for executing operations on a ledger	Ledger execution cost units (e.g., gas in Ethereum)	Measure the total execution cost per box	As low as possible	Managed to reduce indicative daily cost to the order of ~€100 per day, as opposed to the direct use of public Ethereum (Scenario 1) incurring a cost of over €4000 a day. See Table 29, Figure 24.
KPI_FSC_2	Handover time	Time to register data to blockchain during a handover between two stages	Time unit (e.g., seconds)	Measure the total time required for blockchain-related operations during a handover of a box between two stages	<1 min	In Scenarios 2, 3, and 4, handovers are registered in private Ethereum instances, where transactions are included in the following block, generated within

						the following 15 seconds.
KPI_FSC_3	Internal state transition time	Time to register data to blockchain during a box' state transition occurring internally within a single stage	Time units (e.g. seconds)	Measure the total time required for blockchain-related operations during a state transition of a box within a single stage	<30 sec	See previous result (KPI_FSC_2).
KPI_FSC_4	Throughput	Number of boxes that can be processed per time unit in any possible handover or internal state transition	Number of boxes per time unit	Measure the handover and state transition delays	> 6000 boxes per day	In Scenarios 1, 2, and 3, respectively, 6000 boxes can be processed end-to-end within 3720 s (~1h), 3240 s (<1h), and 1800 s (30 min), respectively, as deduced by extrapolation on Figure 25. Scenario 4 has virtually unlimited speed, as transactions are only recorded in local databases, not ledgers.
KPI_FSC_5	Time scalability	Blockchain registration time for a handover or internal state transition, as a function of the number of boxes involved	Derivative of the blockchain registration time with respect to the number of boxes involved	Measure handover and state transition blockchain registration time as a function of the number of boxes involved	Linear or sublinear	Processing time increases linearly for Scenarios 1, 2, and 3 (Figure 25), while it is negligible for Scenario 4, where transactions are stored in local databases instead of ledgers.
KPI_FSC_6	Cost scalability	Public blockchain costs associated with box handovers or internal state transitions, as a function of the number of boxes involved	Derivative of public ledger cost with respect to the number of boxes involved	Measure public blockchain cost for handovers and state transitions as a function of the number of boxes involved	Linear or sublinear	Cost increases linearly as a function of the number of boxes for Scenario 1, while it is constant for Scenarios 2, 3, and 4, where it only depends on the periodic logging of sensors and anchors, irrespectively of the

						number of boxes being processed.
KPI_FSC_7	Response time for audit requests	The time it takes to respond to an audit request, by pulling out all data related to the box in question	Time units (e.g., seconds)	Measure the time it takes to pull out all records related to a given box, and to cross check them to identify potential issues	<1 min	All data from one or more ledgers and/or local databases can be accessed in parallel, with negligible latency, as reading from a ledger is performed instantaneously and does not involve any transaction processing.

6.1.7 Conclusions

It turns out that the use of a public ledger to directly store FSC data (Scenario 1) is exorbitantly expensive, prohibiting this option from further consideration. Even worse, it demonstrates the lowest throughput of all options.

Using a single shared private ledger based on a local Ethereum instance (Scenario 2) to store FSC data and periodically anchoring it to a public ledger is the least expensive solution. However, it demonstrates mediocre throughput, which might prove detrimental if the system scales up with a large number of farms, supermarkets, and transportation companies getting on board.

Splitting a single shared ledger into multiple ones, each serving a subset of the chain (Scenario 3), has a very positive effect on the system's throughput, while the cost remains within very reasonable levels. This approach is expected to demonstrate unlimited scalability with respect to the number of businesses joining the FSC, as each new pair may spawn a new private ledger.

Finally, using local private storages (Scenario 4) for FSC data appears to be the simplest architecture, as entities may manage them independently without having to maintain a joint ledger. Additionally, throughput is not hindered by any ledger operation in this architecture. However, availability is expected to be lower compared to the other scenarios, a topic that is hard to quantify, and which deserves more attention in future work.

6.2 Decentralised Energy Flexibility Marketplace

The Decentralized Energy Flexibility Marketplace (DEFM) pilot aims to balance the load on a real energy network, namely the distribution grid of the city of Terni, located in central Italy, by charging electrical vehicles.

Note that the evaluation results presented in the current deliverable for the decentralized energy flexibility marketplace are new.

6.2.1 Overview

In Terni, a significant amount of energy is produced locally by distributed photovoltaic plants, which on occasion can cause Reverse Power Flow, when imbalances between produced and consumed energy occur. To avoid this abnormal operation, Electric Vehicles (EVs) will be offered incentives to match their charging needs with the distribution network's requirements, through a decentralized marketplace that allows electricity producers and consumers to place offers and bids for selling and buying electricity.

The actors of the pilot use case are the following:

DSO:	Distribution System (Grid) Operator, responsible for grid management
CSO:	Charging Station Operator, operates multiple charging stations
CS:	Charging Station, that can charge electric vehicles
EV:	Electric Vehicle
EVU:	EV User
FM:	Fleet Manager, represents a group of EVs in the energy price negotiations
ER:	Electricity Retailer (that may be included in a later stage of the pilot, but is included in the use case and) that would act between the DSO, CSOs, and electricity users, such as FMs

The flow of the use case is:

1. The DSO puts flexibility requests to the decentralized marketplace (which utilizes a blockchain), asking for a specific amount of energy (kWh) to be drawn at specific time intervals, at a specific location (expressed as GPS coordinates), while providing specific incentives (expressed as tokens) in order to shave peaks of locally produced energy.
2. The FM places offers to the marketplace in order to maximize the incentives. The offers include user type (e.g., electricity load imposed), current location, residual autonomy (i.e., how long the vehicle can continue before it has to be charged), and the EV's current status (e.g., parked).
3. The marketplace identifies potential candidates taking both offers (by DSO and FM) into account and notifies selected EV users that they will receive a token incentive if they fulfil the conditions of the DSO's offer (i.e., charge the vehicle with a specific amount of electricity, using the assigned charging station, or group of charging stations, within a specific time interval).
4. Some EV users accept the offer and the acceptance is recorded in the blockchain used by the marketplace.
5. The EV user (who has accepted the offer) charges the EV to fulfil the conditions of the offer. The charging event will be recorded in the blockchain.
6. The smart contract (running on the Ethereum blockchain) notices that the conditions of the accepted offer have been satisfied and sends the agreed amount of incentive tokens to the EVU on the (Ethereum) blockchain.
7. For accepted bids that failed to fulfil their requirements, the EVU should be “fined” by sending a corresponding transaction on the (Ethereum) blockchain.

6.2.2 Emulation overview and setup

The design of the software that emulates the DEFM pilot is driven by the KPIs and requirements that are defined in D4.3 and D5.2, respectively. The use case scenarios that define the functionality of the pilot are presented in D5.1 and D5.2. In this section, we first introduce the design of the emulated setup and then we discuss the satisfaction of the (qualitative) requirements and the assessment of the (quantitative) KPIs.

The design of the emulated pilot is based on the scenarios that (jointly) cover the specified requirements and KPIs, namely, we focus on the Pull scenario (D5.1, Section 3.3.3.1), the Push scenario (D5.1, Section 3.3.3.2) and the Electricity Supply Offer use case (D5.2, Section 5.2.1.2).

In order to enhance the flexibility of our evaluation, we exploit our own implementation of the marketplace smart contract (*emuEnergyFlexibilityMarketPlace*), which allows us to explore clean-slate solutions for managing the energy requests and offers. We discuss the specifics of the *emuEnergyFlexibilityMarketPlace* smart contract in the following section.

Emulation setup

In order to evaluate and compare the emulated pilot, the *emuEnergyFlexibilityMarketPlace* smart contract was implemented in Solidity (the mainstream Ethereum language) and deployed on Ethereum. In order to experiment with numerous Ethereum accounts, thus being able to emulate multiple FM offers and DSO requests, we exploit a local Ethereum node, that was implemented through the Ganache ledger. This setup allows efficient investigation of the quantitative requirements. To assess the qualitative KPIs, we use the Ethereum Ropsten testnet as a public ledger, which induces realistic latency and throughput.

We made some configuration decisions to reduce the parameter space and to allow for a fair comparison. With respect to parameters of our local Ethereum instances, we fixed the average block mining time to 15 seconds, and we set the block gas limit to 10,000,000 gas units. Both these values reflect the respective values in the public Ethereum main net.

The *emuEnergyFlexibilityMarketPlace* smart contract offers the required functions and events in order to meet and explore the defined requirements and KPIs, respectively. The fact that the methods of smart contracts can be called by asynchronous methods allows the development of scripts that can implement complex sequences of function calls, thus implementing diverse scenarios with low development overhead. We orchestrated the emulation scenario using a single JavaScript script, named EmuApp, that emulates the actions of all involved actors (DSO, FM, EVuser and ER). The main gain of exploiting a single script is that development is relieved from the burden of interprocess communication, which is required for emulating the off-band communication, that is, communication that does not go through the blockchain. The monitoring of the fleet of vehicles and the status of the EVSEs qualifies in this category. Being emulated by the same process, thus sharing the same memory, inconsistencies of memory state among the actors are unlikely to occur. Finally, in the local node setup, we exploit different Ethereum account addresses in order to emulate the different users and, in turn, actors.

The status of the actors is synthetic, being initialized in the beginning of the experiment; we currently introduce some random variables, such as the amount of energy offered flexibility requests, the probability of an EV to accept an offer, the price of the energy supply offer, and some fixed ones, such as the number of users per actor type that participate in an experiment. We can create numerous synthetic traces in order to stress test the pilot, but our design foresees the exploitation of realistic (or even real) traces in order to position our results with precision under real-life conditions.

In the following we describe the emulated scenario and then summarize the fundamental parts of the contract.

6.2.3 Emulation scenario

We assume that the DSO adds a flexibility request in the marketplace used by the FMs for placing offers. The flexibility request carries two deadline timers (among other parameters), one for configuring when the best offer must be selected and a second for determining the timeframe within which energy must be consumed. When the first deadline expires, the marketplace is instructed by the DSO to accept an offer. If the offer includes transferring ETH coins, the marketplace executes the trade. At the same time, two events are emitted, one event notifies the FMs which offer has been accepted and a second event notifies EV users that there are available offers for them (EVoffers). At this point, two parallel procedures can take place. First, the FM can initiate an off-band auction with a set of ERs requesting for energy supply (SupplyOffers). The outcome of the auction is registered in the marketplace by both parties and then the trade of ETH coins takes place. Second, the EV users can notify the marketplace that they accept an offer and then their response is associated with the FM's accepted offer and registered in the Ethereum blockchain. When the second deadline expires, the FM requests from the marketplace to check if the EV users did charge their vehicles and accordingly apply the corresponding meters: transfer the tokens to users or mark that the user did not comply with the terms of the agreement. The integrity and consistency of the offers and requests (including EVoffers and SupplyOffers) is guaranteed by the marketplace, e.g., the positive response of an EV user will be rejected if a number of previous ER supply offers and EV responses did cover the amount of energy specified in the flexibility request.

The fundamental functions and events of the *emuEnergyFlexibilityMarketPlace* smart contract are the following:

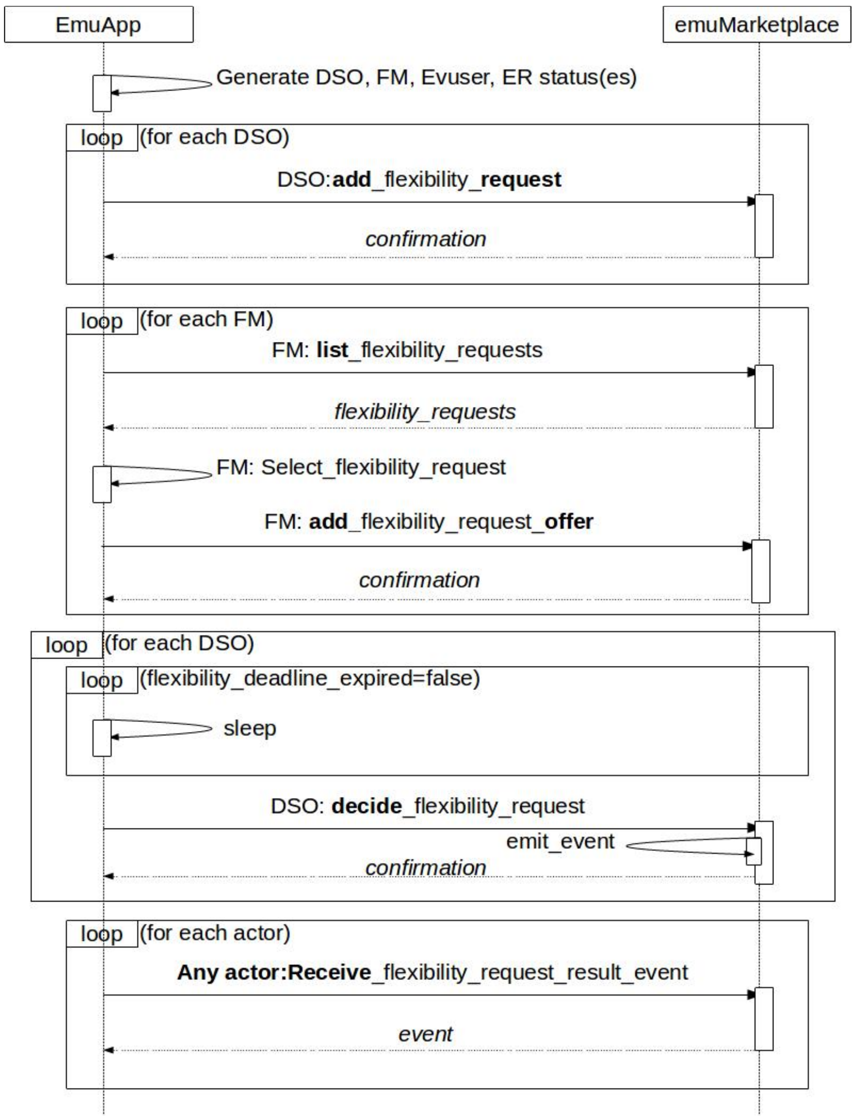
- Getter/setter flexibility request functions: setter is used by the DSO for adding a flexibility request, getter is used by the FM for reading the available flexibility request.
- Getter/setter offer functions: setter is used by the FM for adding an offer to an available flexibility request, getter is used by the marketplace when emitting events in order to notify the EV users.
- Decide function: used be the DSO to request by the marketplace that an offer must be selected (if the corresponding deadline has passed).
- Getter/setter EV user response functions: setter is used by the DSO for adding a user's positive response to an accepted FM offer, getter is used by the marketplace for checking that EV user conformed to the terms of agreement.
- ConfirmCharging function: used by the EVSE for registering that an EV user fulfilled its commitment.
- Transfer token function: used be the marketplace for pushing the agreed number of tokens to the EV user or mark that user did not comply with the agreed terms.

The public “getter” functions return the state of the related trading action (flexibility request, offer etc.) in the blockchain, which also includes the result of the trade, therefore making publicly accessible the outcome of any marketplace trade.

6.2.3.1 Sequence diagrams

A more technical representation of the evaluation scenario is depicted in the following figures that illustrate the sequence diagrams of the procedure. The sequence diagrams define two actors, the EmuApp and the *emuMarketplace*, which is short for the *emuEnergyFlexibilityMarketplace*. In the label of the function calls, the name of the emulated actor is also indicated in order to make the procedure easier to follow. Without loss of correctness, the figures suggest that the functions of marketplace contract return data to the EmuApp, which is not always possible in Ethereum. In these cases, the EmuApp reads the result of the function via a function-specific solidity event or a solidity “view” function.

In Figure 26, we present the sequence diagram of the emulated scenario until an FM's offer is decided. The figure reveals that our design supports multiple DSOs and FMs per deployment; multiple flexibility requests and offers per DSO and FM, respectively, are supported, albeit not being presented in this diagram. Currently, one deadline for all flexibility offers is supported. Finally, for reasons of convenience, the reading of the event is presented as a blocking function, albeit being implemented as a non-blocking call with a registered call-back function.



EmuApp
emuMarketplace
Generate DSO, FM, Evuser, ER status(es)
loop (for each DSO)
DSO:add_flexibility_request
confirmation
loop (for each FM)
FM: list_flexibility_requests
flexibility_requests
FM: Select_flexibility_request
FM: add_flexibility_request_offer
confirmation
loop (for each DSO)
loop (flexibility_deadline_expired=false)
sleep
DSO: decide_flexibility_request
emit_event
confirmation
loop (for each actor)
Any actor:Receive_flexibility_request_result_event
event

Figure 26: Sequence diagram of the emulated Pull Flexibility Request Scenario.

Figure 27 presents the sequence diagram for the DR campaign with the EV users. The diagram shows that EV users can accept an offer given that they are compatible with the offer, based on their location and autonomy, and interested in the offer, based on a random variable. Then, they can fulfil the agreement, again based on a random variable, and, finally, receive the agreed number of tokens. The parameters of the EVoffer, such as the number of tokens, are determined by the marketplace. If the positive responses by the EV users surpass the number of tokens or energy of the flexibility request, only a subset of responses will be accepted following the “First Come First Serve” policy.

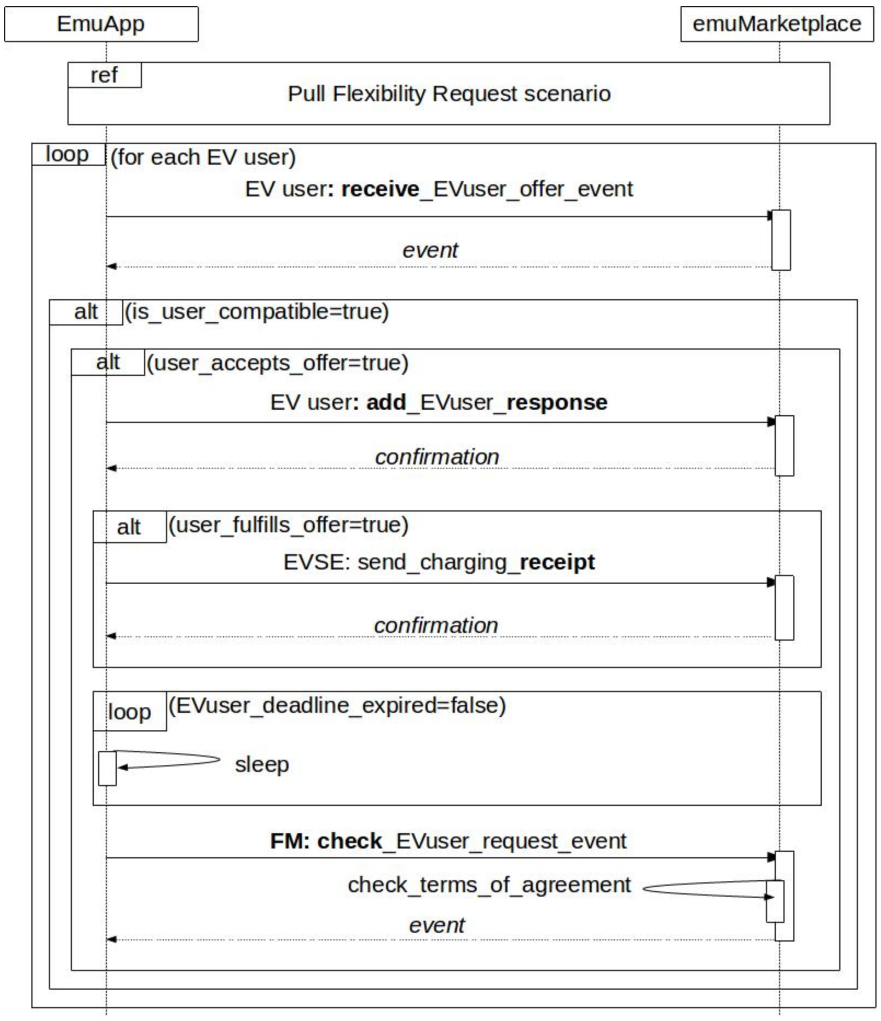

EmuApp
emuMarketplace
ref
Pull Flexibility Request scenario
loop
(for each EV user)
EV user: receive_EVuser_offer_event
event
alt
(is_user_compatible=true)
alt
(user_accepts_offer=true)
EV user: add_EVuser_response
confirmation
alt
(user_fulfills_offer=true)
EVSE: send_charging_receipt
confirmation
loop
(EVuser_deadline_expired=false)
sleep
FM: check_EVuser_request_event
check_terms_of_agreement
event

Figure 27: Sequence diagram of the emulated Push Flexibility Request Scenario with EV users.

Finally, Figure 28 illustrates the sequence diagram for the DR campaign with the ERs. This procedure emulates the auction with the ER bids, which constitutes an off-band communication, hence each bid is determined by a random variable. Thereon, the winner of the auction is determined and the two parties, the FM and the winner ER, establish an agreement that is registered in the marketplace and, in turn, in the blockchain.

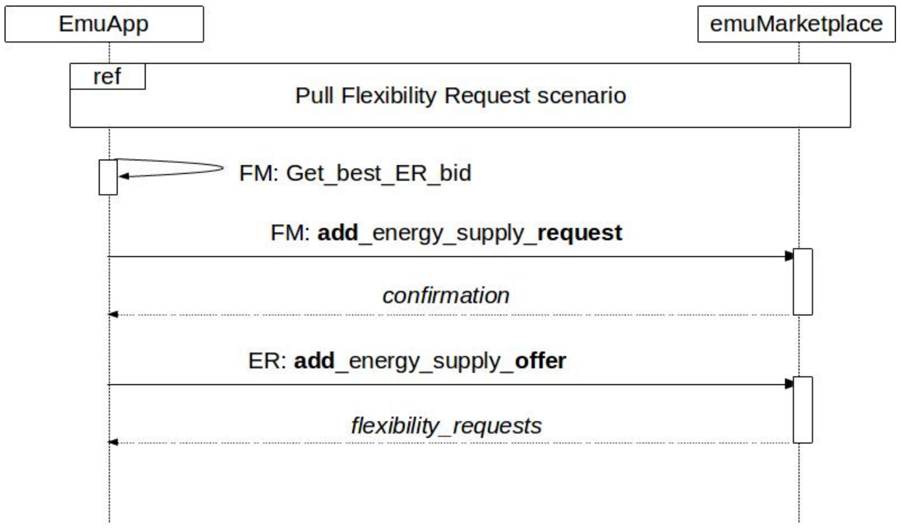

EmuApp
emuMarketplace
ref
Pull Flexibility Request scenario
FM: Get_best_ER_bid
FM: add_energy_supply_request
confirmation
ER: add_energy_supply_offer
flexibility_requests

Figure 28: Sequence diagram of the emulated Push Flexibility Request Scenario with ERs.

6.2.4 Emulation results

We preliminarily explore the performance of the emulated pilot based on the indicated pilot-specific KPIs. We first discuss the performance assessment in detail and then summarize the results in Table 33.

In order to assess the KPIs, we deployed the smart contracts that implement the emulated marketplace in a local Ethereum node, which is set up through the Ganache software tool (Linux 64-bit, 4 cores @ 3.40 Ghz, 4GB RAM), and in the Ropsten public testnet. The performance was found to be similar in both setups, therefore we discuss only the results that were measured in the local node.

We measure “response time” by subtracting the time that the smart contract function was called from the time the response of the function is received. We measure “execution cost” through the Ganache logging system, that offers the complete description of transactions. We measure “throughput” through the Ganache logging system, that documents all the transactions of the smart contract.

6.2.4.1 Response Time

In all the runs that we performed, the response time of any **individual** execution of a smart contract function was defined by the mining period of the Ethereum node, therefore it was less than 15 seconds. The result differs in case of **multiple** executions of a smart contract function. Specificcally, as discussed in the “Throughput” section, we observe that up to 40 transactions (caused by the most expensive function, which is to add a new flexibility request) can be mined in every mining round, hence in case of 41 cumulative executions, the 41th will be mined in the following mining round. In this case, the response time in seconds, *R*, is a function of the number of (roughly) simultaneous transactions, *t*, that are sent to the marketplace, increasing linearly by 15 seconds for every 40 transactions:

$$R = 15 \lfloor t : 6 \rfloor$$

6.2.4.2 Execution Cost

First, we measure the execution cost of the smart contract functions that write to the blockchain. The functions are invoked through a local script that exploits the Ethereum JavaScript API web3.js. In the list below, we present the results per function. The careful reader may notice that we also present the response latency of each function (in the parenthesis); we configured Ganache to auto-mine transactions (only for this experiment) in order to tentatively examine the processing overhead per request.

- smart contract deployment: 1800k gas
- add_flexibility_request: 220k gas (200ms)
- add_flexibility_offer: 210k (200ms)
- add_EVuser_response: 210k (200ms)
- decide_flexibility_request: (1/ 50/ 100 offers): 60/ 150/ 250k (140/ 700/ 1200ms)

Notice that the costs are bound to change if requirements are modified, e.g., we assume that offer and request identifiers are unique (through an out-of-band mechanism), hence we do not check for duplicate entries when adding flexibility requests and offers, thus reducing the cost.

Also, we expect a reasonable increase of the execution cost of stateful functions that read the blockchain, such as a flexibility request which examines the “active” offers. The “active” offers are a (redundant) struct that is used similarly to a cache: it keeps the identifiers of the requests and offers that have not yet expired in order to conduct stateful operations with reduced cost. As presented in the results, the execution cost (and processing latency) of deciding a request increases linearly as the number of offers increases, requiring roughly 90k more gas for every 50 active offers.

6.2.4.3 Throughput

The logging system of Ganache yields that up to 40 marketplace transactions are written in the blockchain every mining round, which is 15 seconds. The estimation is based on adding new flexibility requests and offers, which cost roughly 220k gas, for two reasons. First, they are expected to be the most frequently executed contract functions, thus constituting the primary factors regarding service scalability. Second, the other actions, such as deciding a request, presuppose specific Blockchain state that is hard to orchestrate in large numbers. Our results yield that the throughput of the marketplace for adding new flexibility requests and offers can be roughly 9600 transactions per hour.

6.2.5 Pilot requirements in emulation scenarios

In the following table, we summarize how the requirements for the emulated pilot are addressed by the introduced design. The requirements that are not addressed by the emulated pilot (being described as “Not applicable”) are specific to the infrastructure of the energy grid; they are not related to the software implementation of the marketplace component, which assesses the performance of the system regarding the exploitation of the Ethereum blockchain.

Table 32: DEFM requirements and emulation scenarios

ID	Name	Description	Emulation scenarios
REQ_ DEFM1.1	DR strategies assessment	DSO shall be able to forecast of electricity production/consumption	Not applicable
REQ_ DEFM1.2	Checking load and production forecast	DSO shall be able to check the load and production forecasting of the whole distribution grid	Not applicable

REQ_ DEFM1.3	Grid System flexibility DR services	DSO shall be able to forecast of electricity production / consumption at the grid level	Not applicable
REQ_ DEFM1.4	DSO foresees and provides flexibility	DSO shall be able to shave picks of energy produced locally the day after so that instability of the system, overvoltage on the feeder, protection discoordination, increased fault currents, and incorrect operation of equipment could be avoided	Not applicable
REQ_ DEFM1.5	Flexibility estimation	DSO shall be able to estimate the energy flexibility availability; Assess flexibility availability by using available historical data.	Not applicable
REQ_ DEFM1.6	Flexibility Request	DSO shall be able to forecast system indicates a potential reverse power flow to be mitigated and DSO system is connected with the flexibility marketplace. The DSO system is connected with the flexibility marketplace	Yes. Using synthetic statuses, the DOS estimates flexibility request and sends it to the marketplace.
REQ_ DEFM2.1	DSO/Fleet Manager Micro-Contract	When the Fleet Manager obtains the responsibility to provide the flexibility required by the DSO, a micro contract between the Fleet Manager and the DSO is executed	Yes. The acceptance of the FM's offer is registered in the blockchain.
REQ_ DEFM2.2	Fleet Manager/ EV User Micro-Contract	When the Fleet Manager obtains the responsibility to provide the flexibility required by the DSO and EV users not belonging to the fleet manager EV fleet are involved in the DR campaign, a micro contract between the Fleet Manager and the EV user is executed	Yes. When the user accepts an offer, the acceptance is associated with the concerning FM and is registered in the blockchain.
REQ_ DEFM4.1	EV/EVSE Systems Interoperability	With the objective of performing Demand Response (DR) campaigns, it is necessary that the management systems of electric vehicles and charging stations communicate with each other, so that it is possible to verify in real time the interaction between the two systems.	Yes. All the actor types are emulated by a single deployment script, thus sharing the same memory.
REQ_ DEFM4.2	EV/EVSE Data Collection	To provide DSO flexibility in an efficient way, the data of electric vehicles and charging stations must be collected in real time (or very close to real time). Data coming from EVSEs and the EVs should be consistent, reliable, transparent and accessible to the partners. Furthermore, to perform optimized DR campaign it is necessary to constantly calculate EV load forecasting to estimate the amount of energy that electric	Not applicable

		vehicles could consume to meet the DSO's flexibility demand.	
REQ_ DEFM4.3	EV/EVSE Data Storage	It is necessary that the data of electric vehicles and charging stations are stored so that they can then be reprocessed, giving fruit to charts that show the effectiveness for the purposes of the DSO of DR campaigns performed during the trial.	Not applicable
REQ_ DEFM4.4	EVSE Unique Identifier	As there will be more than one charging station on the pilot site, each individual charging station must have its own unique identifier.	Not applicable
REQ_ DEFM4.5	EV Unique Identifier	As there will be more than one electric vehicle on the pilot site, each individual electric vehicle must have its own unique identifier.	Not applicable
REQ_ DEFM4.6	EV/EVSE Web Platform	To allow the EV user to realize the available charging stations and the fees associated with them, a web platform is required.	Not applicable
REQ_ DEFM4.7	EV/EVSE Connectivity	Both charging stations and electric vehicles must be connected to the internet in order to send data.	Not applicable
REQ_ DEFM5.1	EVSE Remote Control	The charging station must be remotely controlled to start/stop charging sessions and to modulate the power output.	Not applicable
REQ_ DEFM7.1	District forecasting	DSO shall be able to have to constantly calculate building consumption forecasting, PV production forecasting and manage batteries to estimate the amount of energy demand at ASM substation. Forecasting will be calculated periodically (every day). Need to reduce undesired reverse power flows	Not applicable
REQ_ DEFM8.1	Fleet Manager/ Retailer Micro-Contract	When the Fleet Manager obtains the responsibility to provide the flexibility required by the DSO, a micro contract between the Fleet Manager and the Retailer is executed for the energy supply to charge electric vehicles	Yes. When the winning ER is found, the energy supply is associated with the concerning FM and is registered in the blockchain.

6.2.6 Pilot system performance KPIs in emulation scenarios scenarios

The experimental evaluation reveals the preliminarily operational cost and gains of the DEFM service when it exploits a public or private Ethereum blockchain node. The performance of the emulated pilot regarding the related KPIs is summarized on the following table.

Table 33: System performance KPIs for the DEFM emulation scenarios

ID	**Name**		**Target**	**Result**
KPI_DEFM_1	Ledger execution cost		As low as possible	Roughly 250k gas for *write* functions

KPI_DEFM_2	Response time for requests and offers		< 5 min	7.5 s
KPI_DEFM_3	Response time for determining the winner of the auction		< 5 min	7.5 s
KPI_DEFM_4	Response time for verifying the winning bid and compensating (or fining) the winner		< 5 min	7.5 s
KPI_DEFM_5	Throughput		> 100 per hour	40 transactions per 15 s (which is equivalent to 9600 transactions per hour)
KPI_DEFM_6	Scalability - time		Linear of sublinear	Linear

6.2.7 Conclusions

The emulation results for the DEFM scenario showed that the response time of any individual execution of a smart contract function was determined by the mining period of the Ethereum node, therefore being less than 15 seconds. The result differs in case of multiple executions of a smart contract function. Specifically, as discussed in the “Throughput” section, we observe that up to 6 transactions can be mined every mining round, hence in case of 7 cumulative executions, the 7th will be mined in the following mining round.

Regarding execution cost, we presented the cost for each function of a smart contract. These costs depend on the specific implementation and will increase if additional functionality is added to the smart contract. Nevertheless, our results show that the execution cost (and processing latency) of deciding a request increases linearly as the number of offers increases, requiring roughly 90k more gas for every 50 active offers.

Finally, regarding the system throughput, our results show that the logging system supports up to 6 marketplace transactions are written in the blockchain every mining round (15 seconds), which is an estimation based on adding new flexibility requests and offers, which are expected to be the most frequently executed contract functions, thus constituting the primal factors regarding service scalability. Our results show that, for the system parameters considered in the emulation, the throughput of the marketplace for adding new flexibility requests and offers can be up to 1440 transactions per hour.

6.3 Decentralized Energy Data Exchange

The aim of the Decentralized Energy Data Exchange (DEDE) pilot is to enable trust between parties who exchange energy meter readings. The work related to this pilot included in this section consists of two directions. The first direction involves mapping the pilot scenarios to PDS and IAA scenarios presented in different sections of this deliverable. This mapping is discussed in Section 6.3.2. The second direction focuses on the verification of smart meter data, which is a function highlighted in one of the pilot scenarios. The latter work involves developing and evaluating a model that captures the cost tradeoffs related to the frequency with which hashes of the smart meter measurements are recorded on a public blockchain and is presented in Section 6.3.4.

For the DEDE scenario, the new results concern new evaluation scenarios for the PDS and IAA components, which implement the authentication and authorization functionality required by the pilot scenario. The new scenarios consider the usage of VCs to support privacy (for the PDS component) and OAuth 2.0 access tokens supported by Ethereum ERC-721 tokens (for the IAA component). We also extended the evaluation results that illustrate the tradeoffs involving the hash recording frequency and how they depend on various system parameters (transaction and verification costs, rate at which data is produced, and rate of verification requests.

6.3.1 Overview

The four actors of the pilot are the following:

- Smart meter system operator: this is the entity responsible for some part of the energy grid. This entity is also the smart meter data collector (granted by data owner) and provides access to the smart meter data to third parties (Data consumers - energy service providers), after the request of the data owner.
- Smart meter data owner: this is the entity who is legally bound to the smart metering point and has the right to allow access to its smart meter data to third parties (Data consumers - energy service providers).
- Data consumer - energy service provider: this entity is responsible for providing the energy service to the end-user (data owner) and is the main user of the smart meter data to which the smart meter data operator, after the request of the smart meter data owner, provides access to.
- Auditor: this entity has an auditing role in energy grid operations and handles disputes between parties.

The aim of the pilot can be expressed in the following two high level scenarios identified in deliverable D5.2 (Initial Platform Validation):

- Data exchange scenario, which covers the sequence from identification, authorization to granting, requesting access and exchanging the smart meter data
- Data exchange verification scenario, which includes audit logging, maintaining tamper-proof evidence in case of disputes, and verification of the integrity of smart meter data.

6.3.2 Mapping of pilot actors to resource access entities

The pilot actors identified above have the following relation to the entities in the resource access scenarios of PDS and IAA components, shown in the message exchange diagrams of Figure 3 and Figure 4:

- Smart meter system operator: This entity can implement the authorization server (AS) and the resource server. Indeed, the OAuth 2.0 framework indicates that the authorization server may be the same as the resource server [Har+12].
- Smart meter data owner: This entity corresponds to the resource owner.
- Data consumer - energy service provider: This entity corresponds to the client, who requests access to smart meter data (one type of resource). This entity would participate in the data exchange scenario mentioned in the previous subsection (and defined in D5.2)
- Auditor: This entity can correspond to another client, who requests access to audit logs (another type of resource). This entity would participate in the data exchange verification scenario mentioned in the previous subsection (and identified in D5.2).

Hence, from the above it is clear that both the data exchange and data exchange verification scenarios correspond to PDS and IAA authentication and authorization scenarios. In the next subsection we discuss how the pilot requirements are met by the PDS and IAA component features and the corresponding evaluation scenarios.

6.3.3 Pilot and PDS/IAA requirements in emulation scenarios

The corresponding DEDE requirements from deliverable D5.2 are shown below, together with a description of how each requirement is considered and the corresponding evaluation scenarios. Note that the access control functionality is provided through the Privacy and Data Sovereignty (PDS, Section 3.2) and Identification, Authentication, and Authorization (IAA, Section 3.3) components, hence the corresponding requirements refer to those components.

Table 34: DEDE requirements and emulation scenarios

ID	**Name**	**Description**	**IAA & PDS requirements and Emulation Scenarios**
REQ_DEDE1.1	Data access	Data owner can access info about his data, full visibility of data use	Achieved by RF06 in Table 20 with IAA component requirements. IAA evaluation in Section 4.3.
REQ_DEDE1.2	Unique identifiers for actors	Each actor must be identified	Achieved by RF05 in Table 20 with IAA component requirements. IAA evaluation in Section 4.3 and Section 5.2.
REQ_DEDE2.1	Data access	Owner must be able to decide who gets access to his/her data	Achieved by RF13 in Table 17 with PDS component requirements. PDS evaluation in Section 4.2.
REQ_DEDE2.2	Auditability / Security	All user info must be GDPR compliant	Achieved by RF10-14 in Table 17 with PDS component requirements. PDS evaluation in Section 4.2.
REQ_DEDE2.3	Auditability / Security	Data handover must be registered and proved at every transaction	Achieved by RF06 in Table 20 with IAA component requirements. IAA evaluation in Section 4.3.
REQ_DEDE2.4	Data access	Service provider must be able to define the energy consumption data parameters	Achieved by RF06-07 in Table 20 with IAA component requirements. IAA evaluation in Section 4.3 and Section 5.2.
REQ_DEDE2.5	Data transfer	Service provider must be able to download the energy consumption data	Achieved by RF06-07 in Table 20 with IAA component requirements. IAA evaluation in Section 4.3 and Section 5.2.
REQ_DEDE2.6	Authentication	Authentication toolkit for all actors (eIDAS compliant)	Achieved by RF07-08 in Table 20 with IAA component requirements. IAA evaluation in Section 4.3 and Section 5.2.
REQ_DEDE2.7	Auditability / Security	Processes monitoring the system must be logged, stored (in local environment)	Achieved by RF06 in Table 20 with IAA component requirements. IAA evaluation in Section 4.3.

REQ_DEDE5.1	Auditability / Security	Service provider must be able to get proof of receiving the energy consumption data	Achieved by RF06 in Table 20 with IAA component requirements. IAA evaluation in Section 4.3.
REQ_DEDE5.2	Auditability / Security	System logs integrity must be 3rd party verifiable (auditor)	Achieved by RF06 in Table 20 with IAA component requirements. IAA evaluation in Section 4.3.

6.3.4 Hash recording frequency and opportunity cost

In this subsection we present a simple model that captures the cost tradeoffs and the impact of the frequency with which hashes of the smart meter measurements are recorded on a public blockchain. Compared to the previous deliverable D4.3, the current report contains new numerical results that illustrate the tradeoffs involving the hash recording frequency and how they depend on various system parameters such as the public ledger transaction costs, the verification costs, the rate at which data is produced, and the rate of verification requests.

The proposed model captures the following costs:

- Cost (monetary) for recording data on a public blockchain. Alternatively, this can refer to the cost for using a timestamping service, which records data together with a timestamp on some immutable ledger.
- Cost for verifying that the data (e.g., smart meter measurements stored on the platform) is consistent with the hashes recorded on the public blockchain. This cost corresponds to the processing cost or consumed power for performing the verification computations.
- Cost that quantifies the opportunity to modify or the impact from actually modifying the data from the time the last hash was recorded on the public chain until the time the next hash will be recorded.

The cost per unit of time for recording hashes on the public chain (or for using a timestamping service) can be expressed as a function P(f), which we assume is a linear function of the hash recording rate f. Alternatively, the function P(f) can be a concave function, if the incremental cost for recording hashes decreases as the hash recording rate f increases.

The verification cost can be expressed as $r_{ver}V(D)$ where r_{ver} is the rate of verification requests and V(D) is a function of the amount of data D required to perform verification. The verification cost is expressed as a cost per unit of time, similar to the hash recording cost. The shape of the function V() depends on how the hashes are computed. If the hash that is recorded on the public chain is computed by applying a hash function on all the data that has been produced since the last hash was recorded, then the verification cost V(D) is a linear function of the amount of data D produced between two consecutive hash recordings. D is equal to r_{data}/f, where r_{data} is the rate at which data is produced and f is the hash recording rate. On the other hand, if a Merkle tree is used to compute the hashes that are recorded on the public chain, then the verification cost V(D) is a logarithmic function of $D=r_{data}/f$.

The cost that quantifies the opportunity to modify or the impact from modifying the data from the time the last hash was recorded on the public chain until the time the next hash will be recorded can be expressed as a function Q(D) of the amount of data D produced between consecutive hash recordings. The actual shape of Q() is application and scenario specific. Possible shapes are the following:

- Linear: Such a shape corresponds to the case where the impact of modifying additional data is independent of the total amount of data modified.

- Concave: Such a shape corresponds to the case where modifying data initially has a large impact, which becomes smaller as more data is modified.
- Convex: This shape corresponds to the case where modifying data has a small impact up to some amount of modified data, after which the impact increases when more data is modified.
- Sigmoid or stepwise function: In this case, initially modifying data has a small impact (or zero impact in the case of a stepwise function), which increases sharply at some point. Then the impact from modifying additional data is smaller (or zero in the case of a stepwise function).

6.3.4.1 Evaluation results

In this subsection we present numerical results for the model presented above. Our goal is to illustrate how the overall cost and tradeoffs involving the hash recording frequency and how they depend on various system parameters such as the public ledger transaction costs, the verification costs, the rate at which data is produced, and the rate of verification requests.

Based on the model presented in the previous subsection, if c_{pub} is the cost for recording a hash on the public ledger, then the cost per unit of time for recording hashes on the public chain is $P(f)=f\ c_{pub}$. The verification cost is $r_{ver}\ V(r_{data}/f)\ c_{ver}$, where r_{ver} is the rate of verification requests, r_{data} is the rate at which data is produced, and c_{ver} is the cost for each verification. The cost c_{ver} represents the cost for a unit of computation that is necessary for performing verification, while the function V() depicts the amount of computation necessary for performing the verification. In the numerical results presented below, we assume that $c_{pub}/c_{ver} = 2$.

The function V() depends on how the hash recorded on the public ledger is recorded. If the hash recorded on the public blockchain is computed by applying a hash function on all the data that has been produced since the last hash was recorded, then $V(r_{data}/f)$ is a linear function. On the other hand, if a Merkle tree is used, then $V(r_{data}/f)$ is a logarithm of r_{data}/f.

The verification cost $r_{ver}V(r_{data}/f)$ and the total cost $P(f) + r_{ver}V(r_{data}/f)$ as a function of the hash recording rate f in the case of a linear function V() are depicted by the lines containing the label “linear” in the figure below. We have assumed that $r_{data}/r_{ver}=1000$. Observe that the optimal hash recording frequency, which is the frequency with the lowest total cost, is approximately 16. Note that the time units for the hash recording frequency are the same time units that the rate r_{data} data is produced and the verification request rate r_{ver}. The verification cost and total cost for the case of a logarithmic function V() correspond to the lines with the label that includes “log”. Observe that in this case, the optimal frequency is smaller than one, i.e., it is much smaller than the optimal frequency in the case of a linear function V(). The reason for the smaller optimal frequency in the case of a logarithmic function V() is because of the concave dependence of the corresponding verification cost on the hash recording frequency.

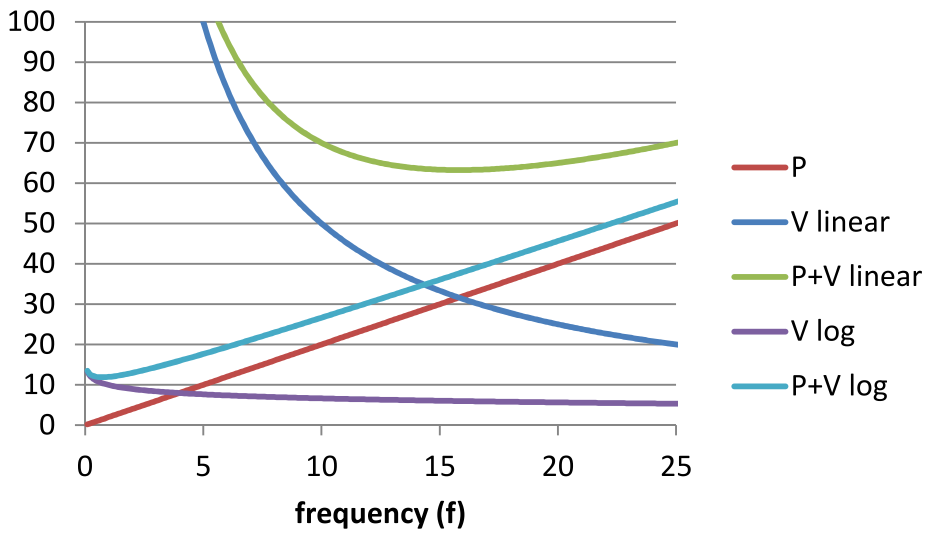


Figure 29: Cost as a function of frequency for a linear and logarithmic verification function V(f)

The optimal frequency as a function of the ratio c_{pub}/c_{ver} when the verification cost is a linear and logarithmic function is shown in the figure below. Note that the optimal frequency depends on the values of c_{pub}, c_{ver} only through their ratio and not separately on the absolute values of these parameters. The figure shows that for both types of verification cost functions, the optimal frequency decreases. However, Figure 31 shows that the optimal frequency decreases slower in the case of a linear verification function, compared to a logarithmic verification function.

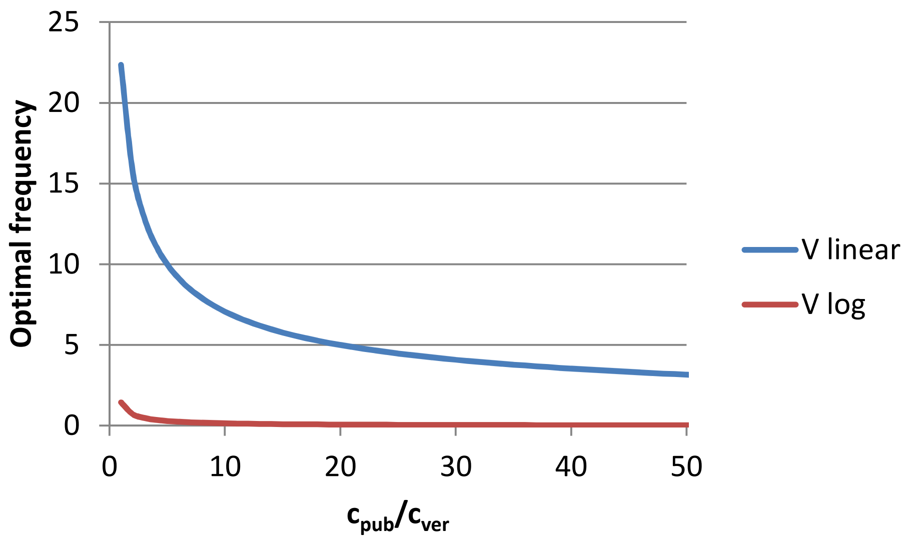


Figure 30: Optimal frequency as a function of the ratio c_{pub}/c_{ver}

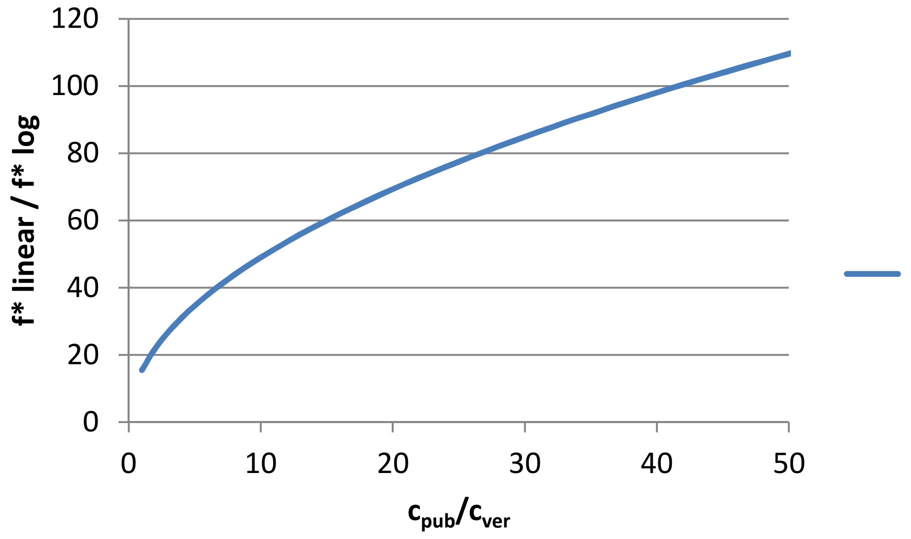


Figure 31: Ratio of optimal frequency for a linear and logarithmic verification cost as a function of the ratio c_{pub}/c_{ver}

The optimal frequency as a function of r_{data}, when r_{ver} is constant and equal to one, for a linear and logarithmic verification function is shown in the figure below. Observe that for a linear verification cost, the optimal frequency increases with r_{data}. On the other hand, in the case of a logarithmic verification cost the optimal frequency is independent of r_{data}. This is because the optimal cost depends on the logarithm of r_{data}.

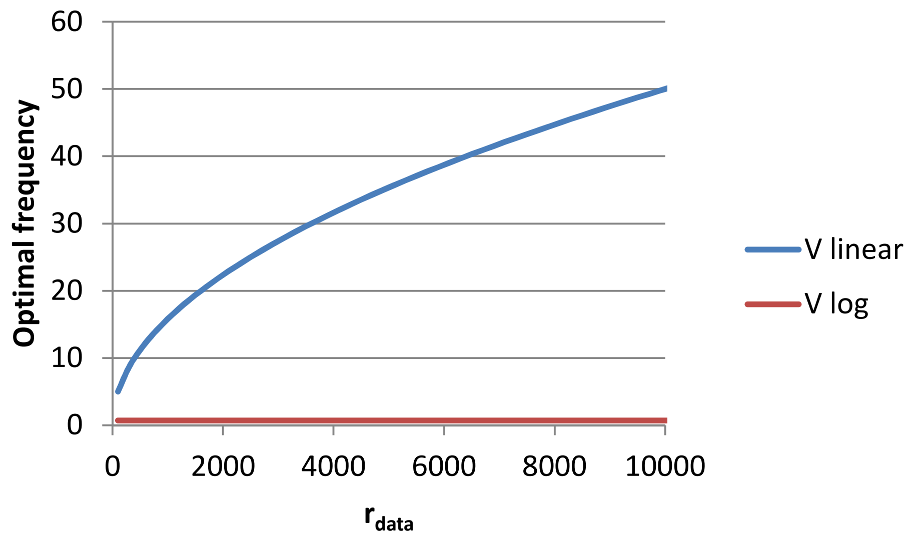


Figure 32: Optimal frequency as a function of r_{data}. The verification rate r_{ver} is constant and equal to one.

The EVM cost for recording a 32-bit hash on the Rinkeby public is approximately 22,600. Hence, if the optimal hash recording frequency is used, the EVM cost for recording hashes as a function of r_{data} is shown in the figure below, whose shape follows that of Figure 32.

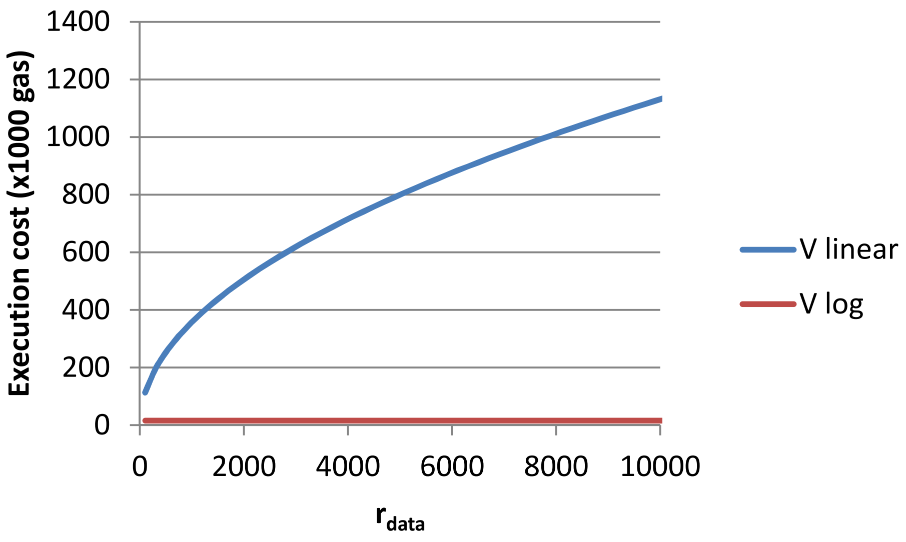


Figure 33: EVM gas cost as a function of r_{data} when the optimal hash recording frequency is used. The verification rate r_{ver} is constant and equal to one.

Nex,t we estimate the cost in euros that corresponds to the above model for selecting the optimal hash frequency. We take the time unit of r_{data} and r_{ver} to be one day, hence $r_{ver}=1$ corresponds to one verification request per day and $r_{data}=720$ (=12*60) is the number of smart meter measurements in one day, if a smart meter measurement is taken once every minute. For the conversion of gas to ether, we assume a price of 10^{-8} ETH per gas unit (referred to as 10 nanoether or 10 gwei) and the price of €200 per ETH. The resulting cost as a function of r_{data} is shown in the figure below. As the above figure, the shape of the figure below follows that of Figure 32.

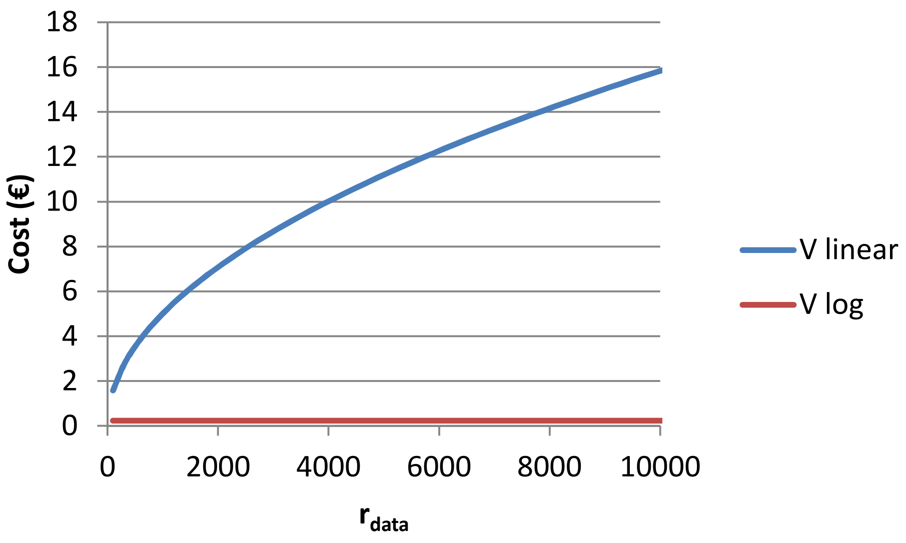


Figure 34: Cost (€) as a function of r_{data} when the optimal hash recording frequency is used. The verification rate r_{ver} is equal to one verification per day.

Our next investigation considers the cost of the opportunity for modifying data before a hash is recorded on the public ledger. We consider two cases for the function Q(D), where D is the amount of data produced between consecutive hash recordings, that models the opportunity cost: 0.1*D and 1*D. The second function reflects the case where the opportunity cost is ten times higher than the first function. The amount of data D between two consecutive hash recordings is D=r_{data}/f.

Figure 35 shows that for opportunity cost Q(D)=0.1*D and a linear verification cost V(), the optimal hash recording frequency is approximately 35, which is higher than the value 16 when the opportunity cost is not considered. For a logarithmic verification cost the optimal recording frequency is approximately 15, compared to less than one in the case where the verification cost is not considered. Hence, we observe that the opportunity cost has a comparatively higher impact in the case where the verification cost is a logarithmic function.

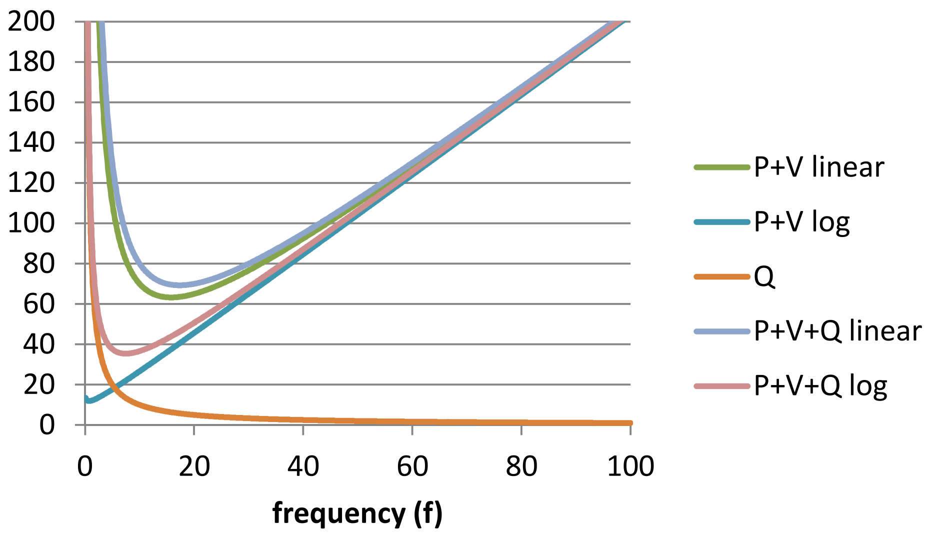


*Figure 35: Cost as a function of frequency for a linear and logarithmic verification function V(f) and Q(D)=0.1*D*

Figure 36 shows that results when the opportunity cost Q(D)=1*D is considered. Compared to the previous figure, we see that, as expected, the impact of the opportunity cost is higher. Moreover, the figure illustrates that as the opportunity cost becomes higher, the difference of the optimal hash recording frequency for the linear and logarithmic verification cost functions becomes smaller.

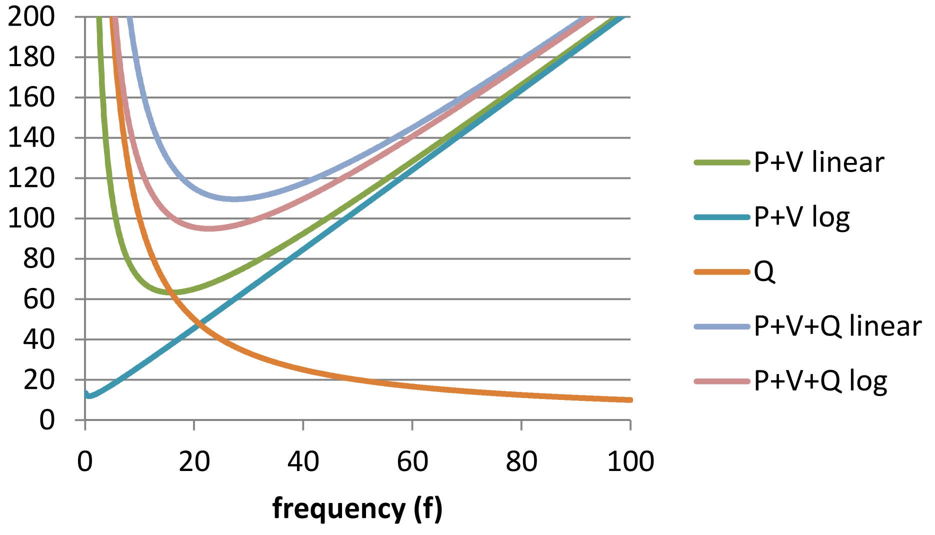


*Figure 36: Cost as a function of frequency for a linear and logarithmic verification function V(f) and Q(D)=1*D*

Finally, we discuss the maximum throughput for recording hashes. For a private Ethereum network, the throughput is determined by the maximum number of transactions that can be recorded in one block and the block mining time. The number of transactions that can be recorded in one block depends on the block gas limit and the gas required for the transaction that records hashes. If we assume that the block gas limit is 10,000,000 gas (which is the default value for the public Ethereum main net) and the hash recording transaction requires 22,600 gas units, then approximately 440 hash transactions can be recorded in one block. If we assume that a new block is mined once every 15 seconds (default average value for the public Ethereum main net), then we have a throughput of 1,760 hash recording transactions per second. The above throughput estimate refers to a private Ethereum network which is used solely for hash recording transactions. A public Ethereum network is shared by many users submitting transactions, hence the hash recording throughput would depend on the rate that these users submit transaction.

6.3.5 Pilot system performance KPIs in emulation scenarios

The evaluation results for scenarios related to this pilot and the corresponding system performance KPIs are presented in the following table.

Table 35: System performance KPIs for the DEDE emulation scenarios

KPI	Name	Target	Results
KPI_DEDE_1	Cost for computing discounts	As low as possible	Results presented in D4.3 (Section 5.3.1). When only hashes are recorded on the public blockchain cost can be more than 80% lower compared to having smart contracts handle discounts. Moreover, the cost for recording hashes is proportional to the hash recording frequency (see also Section 6.3.4).
KPI_DEDE_2	Cost for recording hashes	As low as possible	Cost for recording hashes on public DLT is proportional to the hash recording frequency. The tradeoff with the verification cost is investigated in Section 6.3.4.
KPI_DEDE_3	Response time for	<5 s	Responding to a resource access request for the first time requires 2 roundtrips, one lookup in the blockchain, one

	access requests		signature generation, and one signature verification (see Section 4.3.2). All operations can be done in less than 1 s.
KPI_DEDE_4	Response time for DID operations	<5 s	DID operations are evaluated in D4.3, Section 3.3.2 and they required less than 200 ms, even in constrained devices.
KPI_DEDE_5	Response time for KSI Blockchain signatures	<2 s	Will be investigated in pilot (WP5 and D5.3).
KPI_DEDE_6	Processing time of requests in adapter	<5 s	Will be investigated in pilot (WP5 and D5.3).
KPI_DEDE_7	Response time for audit logs	<15 s	Will be investigated in pilot (WP5 and D5.3).
KPI_DEDE_8	Scalability – cost	linear or sublinear	Results presented in D4.3 (Section 5.3.1) showed linear dependence of transaction cost on the frequency of discounts computations and hash recording frequency. The tradeoff with the verification cost is investigated in Section 6.3.4. If the optimal hash frequency is considered, the public ledger transaction cost as a function of the data rate is sublinear for a linear verification cost and constant for a logarithmic verification cost (Figure 33 and Figure 34).
KPI_DEDE_9	Scalability – time	Linear or sublinear	The computational cost and delay of IAA and PDS is linear to the number of clients. Nevertheless, with respect to a single client, the computational cost and delay are sublinear to the number of requests, since a VC (in the case of PDS) and an access token (in the case of IAA) can be re-used.

6.3.6 Conclusions

For the DEDE scenario the new results include new evaluation scenarios for the PDS and IAA components, which implement the authentication and authorization functionality required by the pilot scenario. The new scenarios consider the usage of VCs to support privacy (for the PDS component) and OAuth 2.0 access tokens based on Ethereum ERC-721 tokens (for the IAA component).

We also extended the evaluation results that illustrate the tradeoffs involving the hash recording frequency and how they depend on various system parameters such as the public ledger transaction costs, the verification costs, the rate at which data is produced, and the rate of verification requests. The results show that the tradeoffs and optimal system performance depends on how the hash recorded on the public blockchain is recorded. Namely, if the hash is computed by applying a hash function on all the data that has been produced since the last hash was recorded or if a Merkle tree is used to compute the hashes that are recorded on the public chain. The optimal hash recording frequency is significantly smaller in the latter case (Merkle tree) compared to the former case (linear computation of hashes).

Furthermore, new results include the influence of the opportunity cost (or gain) for modifying data before a hash is recorded on the public ledger. These results show that the opportunity cost has a comparatively higher impact when the hashes are computed using a Merkle tree.

Moreover, when the opportunity cost increases, the difference of the optimal hash recording frequency for the linear computation of hashes and computation of hashes using a Merkel tree becomes smaller.

6.4 Mixed Reality Mobile Gaming

The Mixed Reality Mobile Gaming (MRMG) pilot will prototype a scavenger hunt location-based game. The goal of this game is to explore various use cases of blockchains within gaming, like payments based on cryptocurrency or cryptocash (regulated real-world money) and advertisements. The game is based on IoT devices (e.g., IoT beacons) and on an open ecosystem built on top of DLTs.

The new results contained in this deliverable compared to the first evaluation deliverable D4.3 include two new emulation scenarios. The first of these scenarios utilizes the Hyperledger Fabric blockchain to support the functionality of the mobile gaming pilot, while the second newly added scenario leverages both public Ethereum and Hyperledger Fabric. Furthermore, we extended the evaluation results of the scenarios defined in previous deliverable D4.3. In particular, the evaluation results, in this deliverable, contain execution cost, response time, throughput and scalability of the system.

6.4.1 Overview

The goal of the scavenger hunt location-based game, described in Deliverable 5.1 is for the players to collect as many points as they can, in order to get rewards. In order to collect points, a player must physically visit an area where an IoT device is deployed. When he visits the location, then a riddle alongside with some clues, provided by the gaming company or other players, are shown on his mobile phone. Solving the riddle will reveal the location of the next Point of Interest (PoI), in order for the player to go there, collect his points and download the next riddle. This procedure continues until the last IoT device is reached.

In addition to the main functionality of the game described above, the mobile game provides some additional features. A player will be able to skip any challenge he wants by viewing an advertisement, by paying in in-app tokens or in real-world money. Furthermore, if a player watches an advertisement, irrespective of skipping the challenge, he will get a reward given by the ads company. Moreover, the player is able to get in-app tokens by paying or by viewing an advertisement. Finally, the player can, at any point of the game, redeem his points in order to get rewards given by the game company. The rewards can be assets (e.g., a sword or a shield) that can be used in the game or in any other games that use the same blockchain platform, thanks to the blockchain's properties (immutability, transparency, etc.).

For the IoT part of the game, IoT beacons will be used to provide the proximity location of a player when he visits the appropriate location. The beacons will communicate with the smartphone of the player using Bluetooth Low Energy (BLE).

6.4.2 Emulation overview

To evaluate the performance of the aforementioned MRMG use case, we used an emulation environment. The emulation will help us obtain a better understanding of the application of blockchain to mobile gaming and compare different DLT setups that utilize public and private/permissioned blockchains. Specifically, our emulation environment supports public (Rinkeby public Ethereum testnet) and private (private Ethereum or Hyperledger Fabric) ledgers, allowing us to compare different configurations with the two types of blockchain that have different properties and trade-offs, in terms of transaction cost, latency, transparency and privacy. Our evaluation focuses on the system performance metrics, defined on the KPIs (Chapter 2.2) for this specific pilot. On the other hand, with the emulation environment we cannot investigate the beacon detection time or other metrics such as the deployment of IoT beacons or business-oriented metrics such as player satisfaction.

To start with, the mobile gaming client was emulated as a Web application (implemented in React, which is a JavaScript library for building user interfaces) in some scenarios and as a JavaScript application in others. The client application has a simple UI that both the players and the game administrators use. For emulation purposes, we assume that the players already have created an account in the gaming system and a blockchain wallet too. The latter corresponds to a public/private key pair for performing transactions on the Ethereum or Fabric blockchain network. Each challenge is identified by a unique identifier. Players can select the challenges they wish to play by choosing the corresponding challenge ID. The client application has a complete button, which emulates the solution of the challenge's riddle. To emulate the in-app tokens, we developed a smart contract that implements the required functionality. In Ethereum, this smart contract implements the ERC20 token standard. Finally, the action of viewing advertisements is emulated as a function of a smart contract.

The following figure shows the use case diagram of the emulation environment. The figure shows the actors of the mobile gaming ecosystem and the functions of the emulated mobile gaming system they can perform or interact with. As mentioned above, the IoT related functions (such as beacon discovery) are not included in the gaming emulation and for this reason the use case diagram does not include such functions nor the corresponding entities (e.g., POI employee).

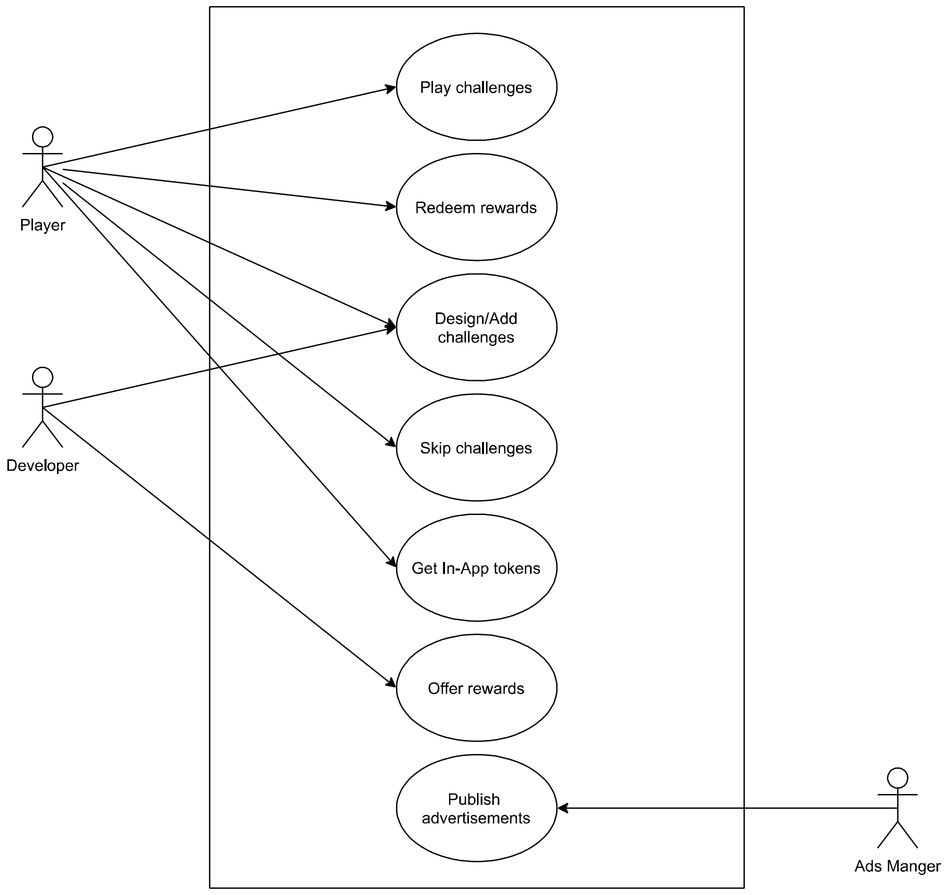


Figure 37: Actors' interaction with the mobile gaming system

The core functionality of the scavenger hunt location-based mobile game, described above, was emulated by three smart contracts. The first smart contract, named game smart contract, communicates directly with the client application. It implements the functionality related to the challenges and rewards. Specifically, it records the challenges on the blockchain and the mapping of players and challenges and whether a particular player has completed a challenge or not. Furthermore, the smart contract automatically calculates the points each player obtains from completing a challenge and implements the function for skipping a challenge. Finally, it has a function for redeeming the rewards. The second smart contract is the ads smart contract, which checks whether the user “watched” an advertisement or not. The last smart contract, called token smart contract creates and manages the in-app tokens.

6.4.3 Emulation scenarios

One of the advantages of using emulation is that we can experiment with various configurations and scenarios involving different DLTs setups. We evaluate the scavenger hunt location-based mobile game in four different scenarios, presented below.

6.4.3.1 Scenario 1

The first scenario considers a single (public) Ethereum blockchain, which implements all the gaming functions of the three smart contracts. The experiments in this scenario took place on the Rinkeby testnet. The components of this scenario are:

- Web application: This component performs the interaction between the actors and the functions of the mobile game.
- Ads smart contract: This component checks whether a user “watched” the advertisement or not.
- Tokens smart contract: This component implements the ERC20 standard. It creates and manages the in-app tokens.
- Game smart contract: This component implements all the main functionality of the game.

In this scenario, all the smart contracts are deployed on the same (public) Ethereum network, thus there is no need for an Interledger Gateway (ILG) for the contracts to communicate with each other. The UML class diagram for this scenario is shown below.

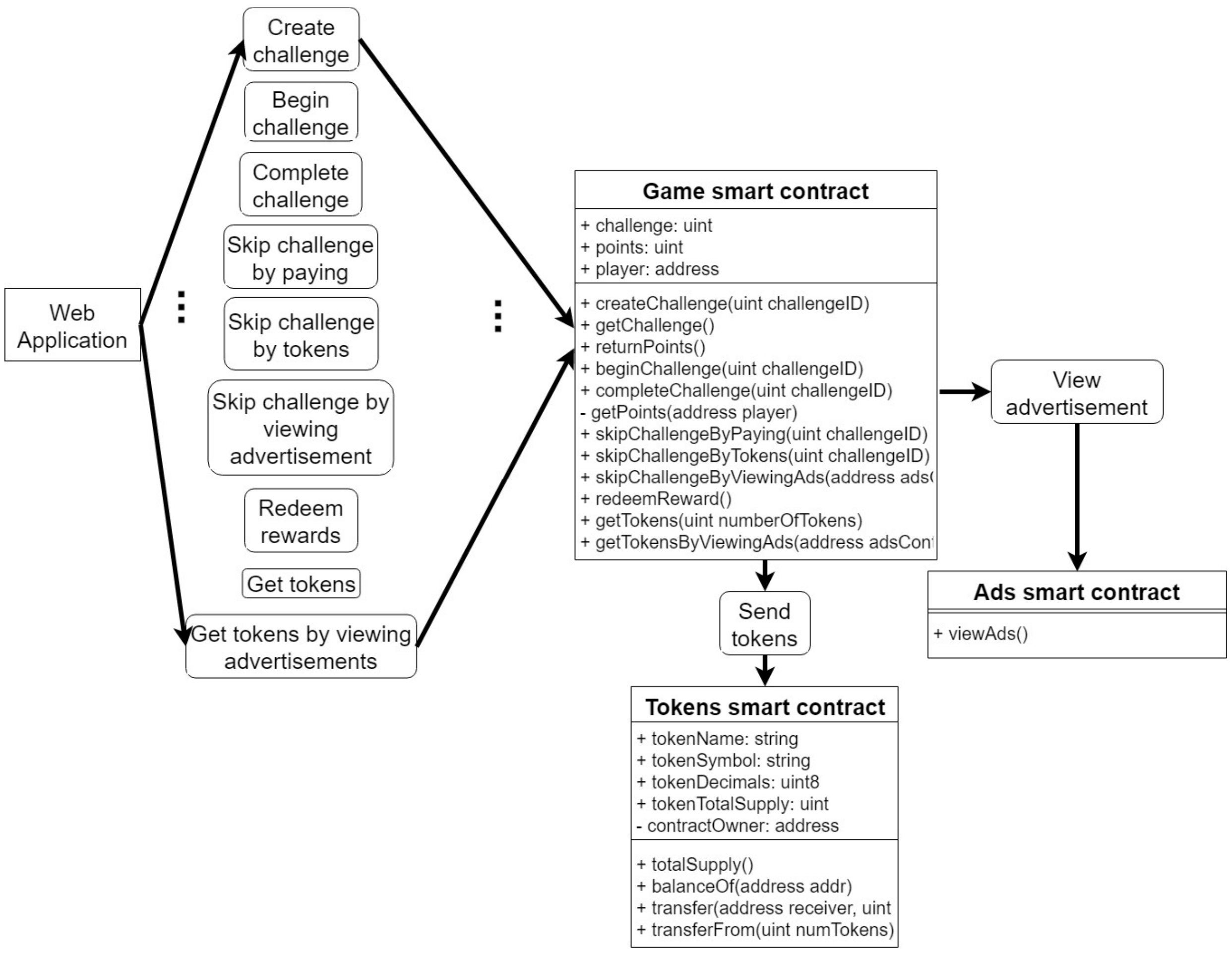
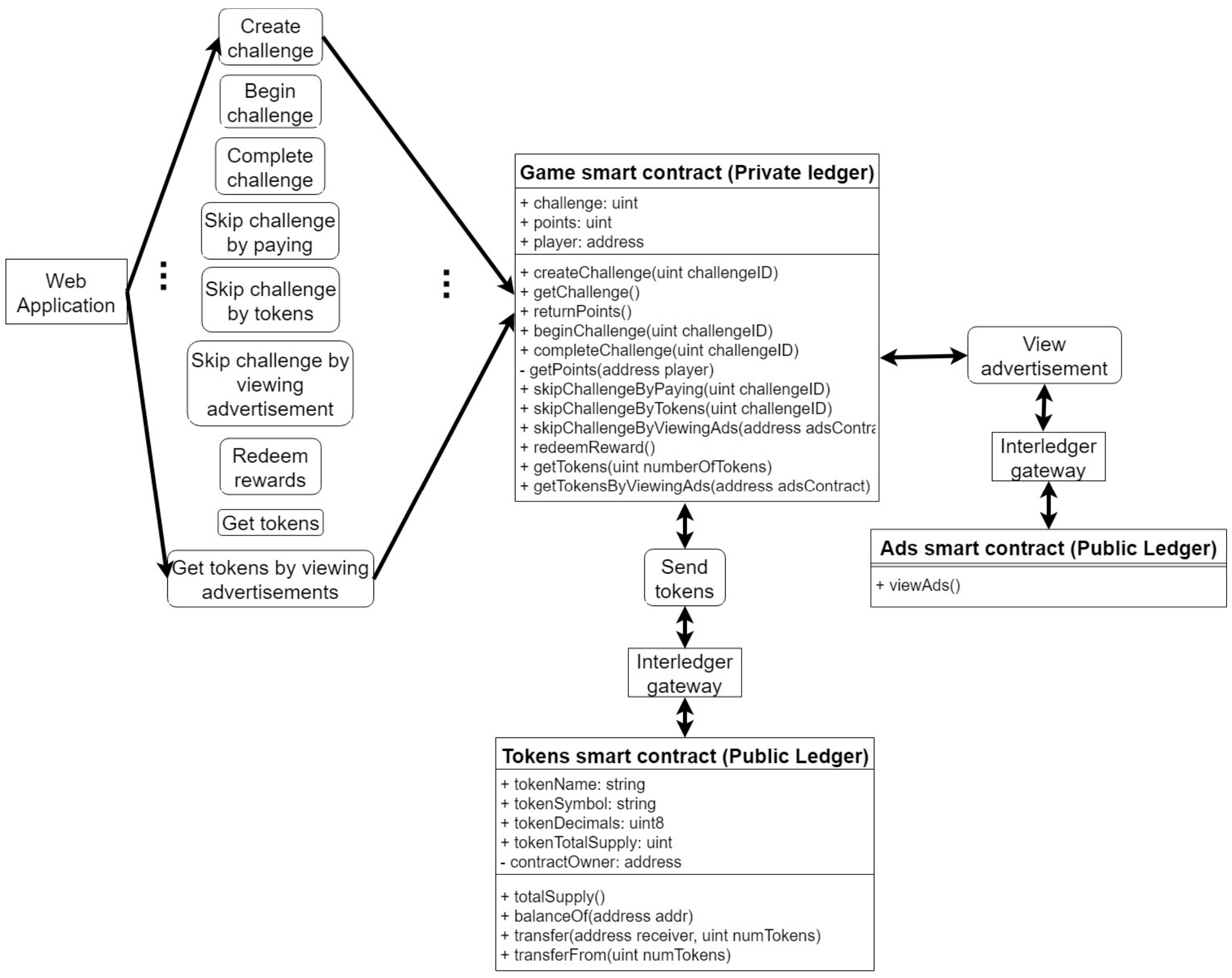

Web Application
Create challenge
Begin challenge
Complete challenge
Skip challenge by paying
Skip challenge by tokens
Skip challenge by viewing advertisement
Redeem rewards
Get tokens
Get tokens by viewing advertisements
Game smart contract
+ challenge: uint
+ points: uint
+ player: address
+ createChallenge(uint challengeID)
+ getChallenge()
+ returnPoints()
+ beginChallenge(uint challengeID)
+ completeChallenge(uint challengeID)
- getPoints(address player)
+ skipChallengeByPaying(uint challengeID)
+ skipChallengeByTokens(uint challengeID)
+ skipChallengeByViewingAds(address ads(
+ redeemReward()
+ getTokens(uint numberOfTokens)
+ getTokensByViewingAds(address adsCon
View advertisement
Ads smart contract
+ viewAds()
Send tokens
Tokens smart contract
+ tokenName: string
+ tokenSymbol: string
+ tokenDecimals: uint8
+ tokenTotalSupply: uint
- contractOwner: address
+ totalSupply()
+ balanceOf(address addr)
+ transfer(address receiver, uint
+ transferFrom(uint numTokens)

Figure 38: UML component diagram for the first MRMG scenario

6.4.3.2 Scenario 2

The second scenario implements the class diagram shown in the following figure and investigates the gains from utilizing two types of blockchain, a public blockchain and a private/permissioned blockchain. In this scenario, the first blockchain is a public Ethereum, while the second blockchain is a private instance of Ethereum. The components of this scenario are the same ones shown above for Scenario 1. However, in this scenario the smart contracts are deployed on different blockchain networks: the game smart contract is deployed on the private Ethereum blockchain, while the other two smart contracts are deployed on the public Ethereum. The interconnection of these two ledgers is performed through an ILG.

- ILG: “listens” for events on both Ethereum blockchains. Such events are generated each time a player invokes a function of the game smart contract that needs to perform an action involving the in-app tokens or advertisements.



Web Application
Create challenge
Begin challenge
Complete challenge
Skip challenge by paying
Skip challenge by tokens
Skip challenge by viewing advertisement
Redeem rewards
Get tokens
Get tokens by viewing advertisements
Game smart contract (Private ledger)
+ challenge: uint
+ points: uint
+ player: address
+ createChallenge(uint challengeID)
+ getChallenge()
+ returnPoints()
+ beginChallenge(uint challengeID)
+ completeChallenge(uint challengeID)
- getPoints(address player)
+ skipChallengeByPaying(uint challengeID)
+ skipChallengeByTokens(uint challengeID)
+ skipChallengeByViewingAds(address adsContra
+ redeemReward()
+ getTokens(uint numberOfTokens)
+ getTokensByViewingAds(address adsContract)
View advertisement
Interledger gateway
Ads smart contract (Public Ledger)
+ viewAds()
Send tokens
Interledger gateway
Tokens smart contract (Public Ledger)
+ tokenName: string
+ tokenSymbol: string
+ tokenDecimals: uint8
+ tokenTotalSupply: uint
- contractOwner: address
+ totalSupply()
+ balanceOf(address addr)
+ transfer(address receiver, uint numTokens)
+ transferFrom(uint numTokens)

Figure 39: UML component diagram for the second MRMG scenario

6.4.3.3 Scenario 3

The third scenario considers a single ledger, but differs from the first scenario in that the blockchain is a permissioned blockchain, namely Hyperledger Fabric, rather than the public Ethereum. The components of this scenario are the same as the first scenario, so there is no need for an ILG. However, there are some differences between these two scenarios, due to the different blockchain technology used. In particular, in this scenario a player cannot skip a challenge by paying in cryptocurrency, because Fabric does not support cryptocurrencies. The UML class diagram of this scenario that shows the actions, the function, etc., is presented in the following figure.

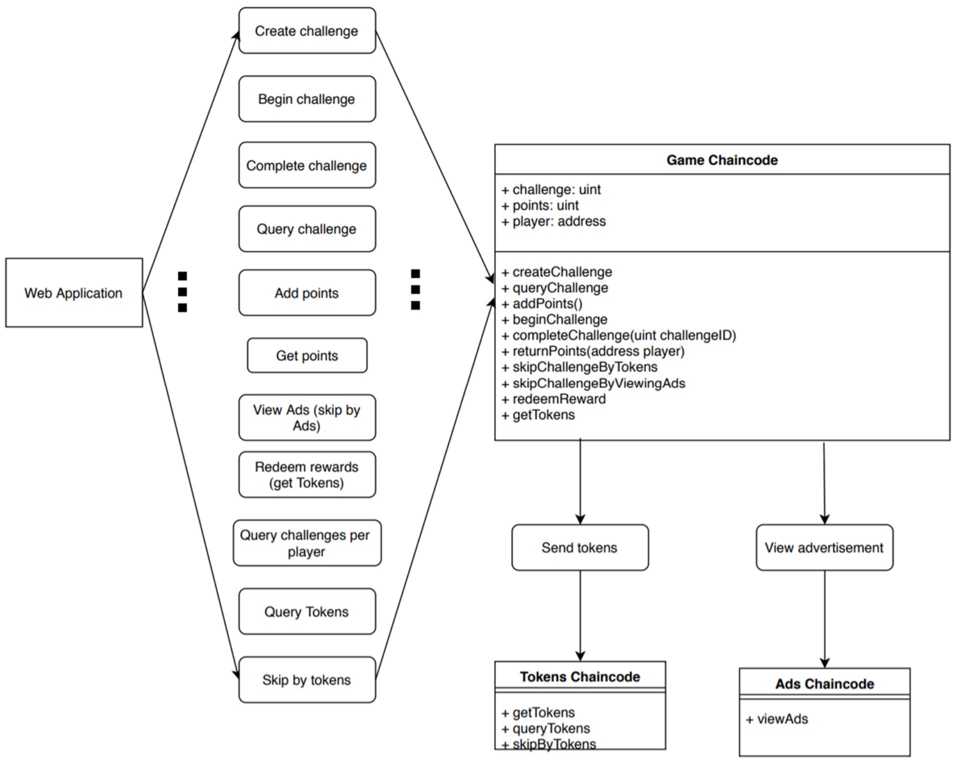
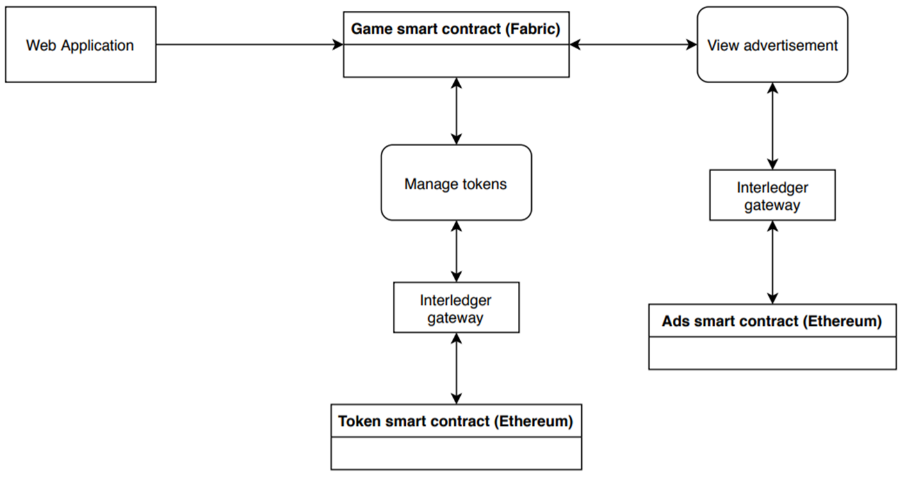


Figure 40: UML component diagram for the third MRMG scenario

6.4.3.4 Scenario 4

Finally, the last scenario that will be evaluated utilizes two (different) blockchains instead of one. The first one is the public Ethereum, while the second one is Hyperledger Fabric. The game smart contract is deployed on Fabric, while the ads and tokens smart contracts are deployed on the public Ethereum. In this scenario we need an ILG for the communication between the two ledgers. The ILG implements both the Fabric SDK, as well as the communication with the Ethereum blockchain, using the web3 library. The class diagram of this scenario is shown below. The smart contracts of this scenario are the same as in the other scenarios, thus the diagram does not show the functions and the parameters of each smart contract.



Web Application
Game smart contract (Fabric)
View advertisement
Manage tokens
Interledger gateway
Interledger gateway
Ads smart contract (Ethereum)
Token smart contract (Ethereum)

Figure 41: UML component diagram for the fourth MRMG scenario

6.4.4 Evaluation results

In Deliverable 4.3, we presented some preliminary results from the evaluation of the MRMG pilot. Specifically, we presented the execution cost in terms of gas for Scenarios 1 and 2 of our emulation. Moreover, we showed how the emulation of the pilot meets the defined requirements. In this deliverable, we present a more comprehensive system performance evaluation of the mobile game, based on the KPIs we have defined, for all four scenarios.

The first KPI involves the cost for executing operations on a public ledger. All the aforementioned scenarios involve the invocation of a smart contract. This invocation is achieved through a transaction. Transactions on a public blockchain incur a transaction cost, which in Ethereum is expressed as the cost of gas for executing transaction on the Ethereum Virtual Machine (EVM). On the other hand, transactions on a private/permissioned blockchain (private Ethereum and Fabric) do not entail an execution cost. The execution cost for the scenarios is shown in the following table.

Table 36: EVM execution cost for MRMG scenarios

Function	Scenario 1	Scenario 2	Scenario 3	Scenario 4
Add challenge	47050	N/A	N/A	N/A
Begin challenge	52432	N/A	N/A	N/A
Complete challenge	53529	N/A	N/A	N/A
Skip challenge by paying	61867	N/A	N/A	N/A
Skip challenge by paying in In-App tokens	63877	33438	N/A	33438
Skip challenge by viewing advertisements	53926	21462	N/A	21462
Get tokens by paying	44199	35274	N/A	35274
Get tokens by viewing advertisements	37981	56736	N/A	56736
Redeem rewards	36618	35274	N/A	35274

The transaction cost (gas) for the Scenarios 1, 2 and 4 is shown in the following figure. Note that the execution cost for the functions of the Scenario 2 and 4 is exactly the same, because the smart contracts are the same.

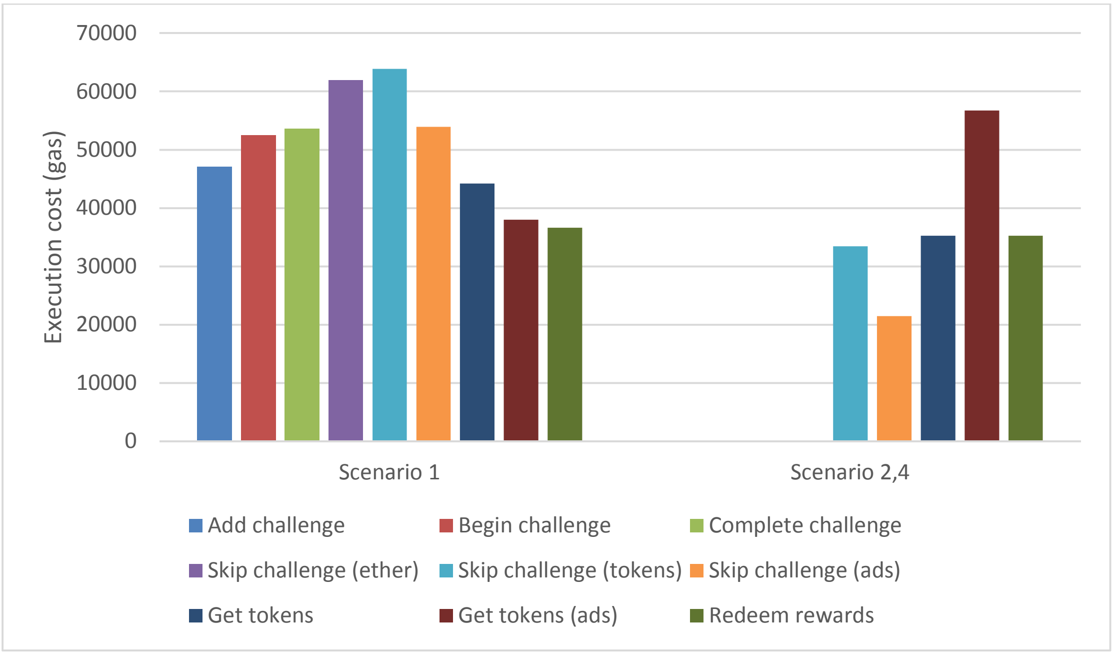


Figure 42: EVM execution cost for the MRMG scenarios

From the figure above, we observe that for all the functions except one, the execution cost is smaller in the second and fourth scenario, which involve a public and private ledger. That is happening, because these actions involve the interaction of the game smart contract with the ads or the token smart contract. On Scenarios 2 and 4, these two smart contracts are deployed in the private blockchains, so they impose no execution cost. For this reason, the total amount of cost required for these specific actions is the gas consumed only by the public ledger.

The function that has a higher execution cost on the 2nd and 4th scenario is for getting in-app tokens by viewing advertisements. This function involves the invocation of all the three smart contracts. The player first calls the function of the game smart contract to alert the system that he wants to watch an advertisement in order to get tokens, then the game smart contract will invoke the ads smart contract and finally when the ads smart contract finishes the execution, the game smart contract will invoke the token smart contract to give the tokens to the player. In the first scenario, all these invocations need only one transaction for invoking the game smart contract; the game smart contract will invoke the other smart contracts internally, without the need of a new transaction. In the other scenarios, that cannot happen because the smart contracts are on different ledgers. Thus, we need more transactions. First, we invoke the game smart contract, which will trigger an event that will be “caught” by the ILG. Then, the ILG will send a transaction to the ads smart contract and when the execution finishes, it will send a transaction back to the game smart contract to inform it that the user watched the advertisement and it can proceed with the payment. This procedure will happen with the token smart contract too. So, for this particular action these two scenarios add an overhead of two more transactions to the game smart contract.

The most common performance metric of any system is the response time required by the system to execute read and write requests. In our case, where the gaming system utilized blockchains, the response time metric corresponds to the time that the system performs read or write transactions. The second KPI refers to the time for the system to respond to game sate altering transactions. In addition to gas, the use of Ethereum blockchain incurs a transaction delay, which depends on the block mining. The average time for mining a new block on Ethereum is ~ 15 seconds. Therefore, the response time for write requests on Ethereum is ~ 15 seconds, so with the first and the second scenario we cannot achieve the desirable result, which is 3 seconds. However, we have performed experiments in the Rinkeby public testnet to measure the transaction delay. These results depend on factors such as the load of the Ethereum network. The results in the table are the average from 10 executions for each of the actions. For a better understanding of the results, we calculated the 95% confidence interval. The results and the 95% confidence interval (in parenthesis) are shown in the table below.

Table 37: Mean response time (s) for write requests (confidence intervals in parentheses)

Function	Scenario 1	Scenario 2	Scenario 3	Scenario 4
Add challenge	14.55 (2.48)	14.55 (2.48)	2.209 (0.029)	2.209 (0.029)
Begin challenge	15.38 (3.14)	15.38 (3.14)	2.195 (0.009)	2.195 (0.009)
Complete challenge	14.14 (3.57)	14.14 (3.57)	2.187 (0.011)	2.187 (0.011)
Skip challenge by ads	15.14 (3.39)	15.14 (3.39)	2.182 (0.016)	15.14 (3.39)
Get tokens (redeem reward)	11.12 (2.05)	11.12 (2.05)	2.187 (0.015)	11.12 (2.05)
Skip challenge by tokens	12.96 (2.90)	12.96 (2.90)	2.176 (0.014)	12.96 (2.90)

The above table shows that the response time for all the write requests, for the Scenario 3, is less than 3 seconds, thus this scenario achieves the target for the specific KPI. On the other hand, Scenario 4 achieves the target, only for the functions that are executed on the Fabric blockchain. The response time for altering transactions of all scenarios is shown in the following figure.

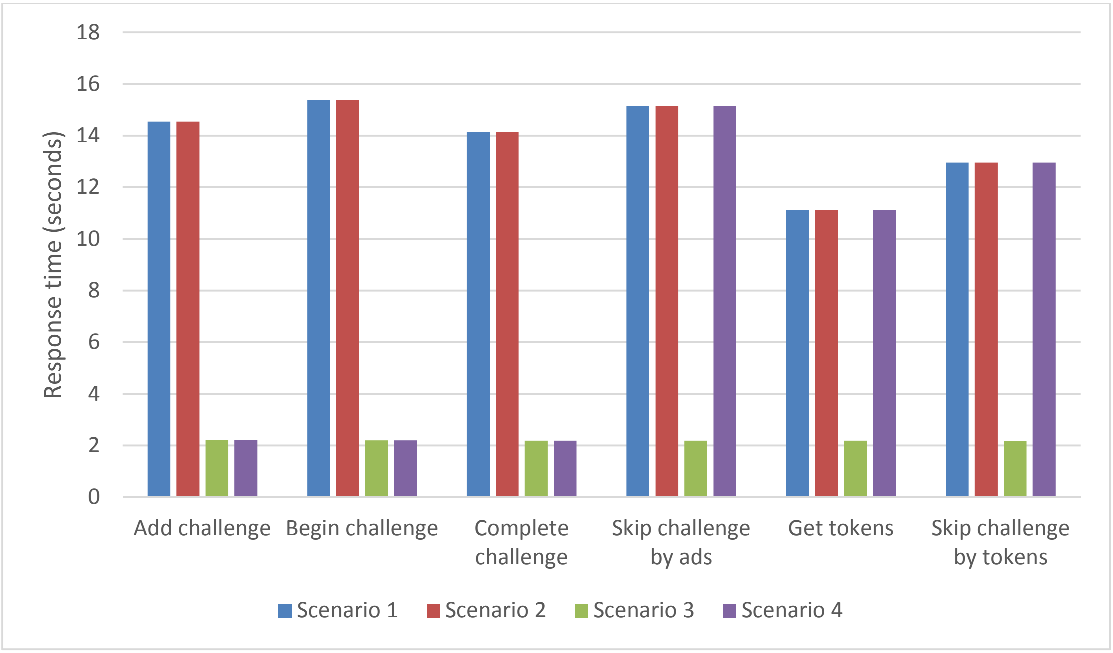


Figure 43: Response time for write requests

On the other hand, the third KPI refers to the response time for read requests, namely non-altering transactions. Read requests in Ethereum do not broadcast or publish anything on the blockchain, and the response is returned instantaneously, since the requests are local, and they do not need the Ethereum network. Thus, for this KPI, we can achieve the desirable target for all scenarios, even for the scenarios that involve Ethereum. The results with the 95% confidence interval are shown in the table below. Again, we performed each experiment 10 times to take the average value.

Table 38: Mean response time (s) for read requests (confidence intervals in parentheses)

Function	Scenario 1	Scenario 2	Scenario 3	Scenario 4
Query points	1.106 (0.053)	1.106 (0.053)	0.0241 (0.0019)	0.0241 (0.0019)
Query challenge	1.085 (0.036)	1.085 (0.036)	0.0256 (0.0062)	0.0256 (0.0062)
Query tokens	1.124 (0.058)	1.124 (0.058)	0.0211 (0.0012)	1.1239 (0.0583)

As we can see from the above table, the response time in scenarios that utilize Ethereum is not negligible. That is happening because in our emulation, we do not run an Ethereum (full or light) node. So, to connect with the Ethereum blockchain we communicate with another node of the Ethereum network. For this reason, we have an addition overhead introduced by the network. However, we did some experiments with local Ethereum, in order to find out the exact value for the response time in non-altering transactions, which is 0.0450 second. The following figure presents the response time for read requests for the four scenarios.

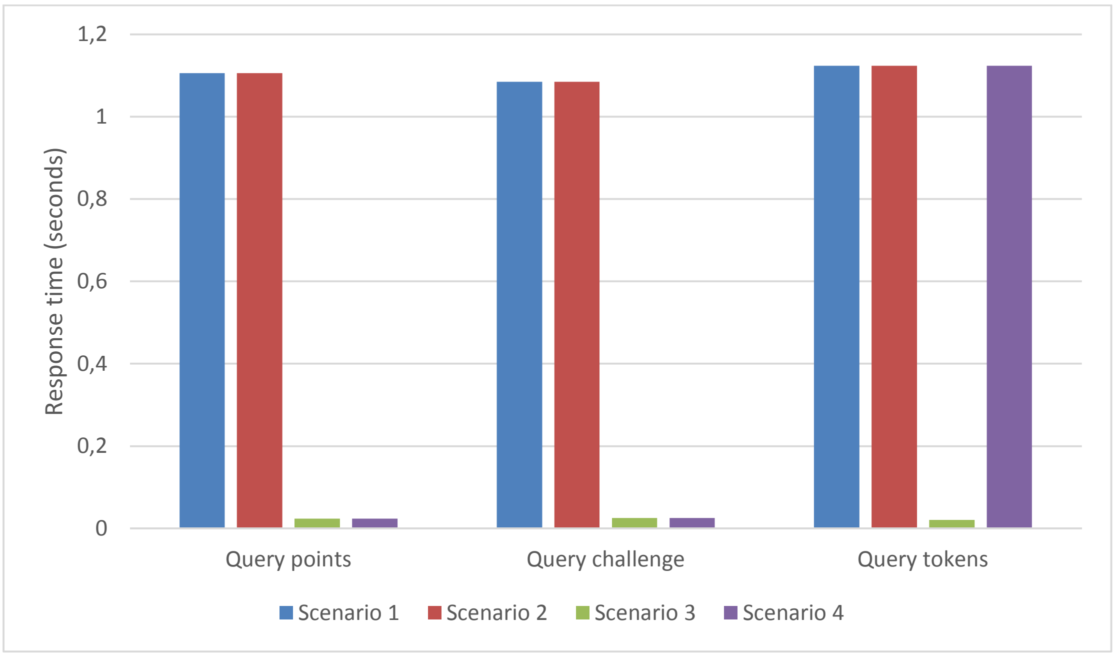


Figure 44: Response time for read requests

The fourth KPI is the BLE beacon detection time. As we have already mentioned above, our emulation does not consider IoT related metrics, thus we do not have results about this specific KPI. The next KPI is about throughput, which is defined as the maximum number of transactions per time unit. The target for this KPI is 222 read transactions per second and 133 write transactions per second.

It is clear that for the first and the second scenario we cannot achieve the desired results, due to the time constraints that the Ethereum blockchain introduces. However, for the other two scenarios the target is achievable.

To measure the throughput in Fabric, we set the number of transactions per block equal to 20, with a maximum transaction size equal to 99 MB. From the experiments, we observed that a block is added on the ledger in ~ 100 ms. It takes ~ 30 ms to validate a block and ~ 70 ms for the multiversion concurrency control mechanism and the mechanism that commits the block to the ledger.

In our case, the write transaction size is ~ 500 KB, thus for a write request we need one transaction. To store 133 transactions, we need 7 blocks. As we have already mentioned above, 1 block is added on the ledger in ~ 100 ms, so 7 blocks need 0.7 seconds. Thus, the system can support 190 write transactions per second. Therefore, we can easily achieve the desirable target for the write requests, which is 133 write transactions per second.

On the other hand, for the read requests, the process is slightly different. The transaction flow in Hyperledger Fabric is:

1. The client sends a transaction proposal.
2. The endorsing peers execute the transaction and send back as a proposal response the signed result. No updates are made to the ledger.
3. The client broadcasts the transaction proposal and response to the ordering service.
4. The ordering service orders the transactions and creates blocks.
5. The blocks are broadcasted to all peers, are validated and each peer appends the new block to the chain.

However, if the transaction is a query (read transaction), the flow ends on step 2. So, the throughput for the read requests depends only on the network and not on the blockchain itself.

To measure the throughput for the read requests in Fabric, we performed some experiments. The experimental results show that for 222 read transactions, we need 1.9053137 seconds, so the system can support 117 transactions per second. However, note that these transactions are sent sequentially, thus if we send the requests in parallel, we can achieve the desirable target.

The remaining KPIs refer to scalability. In particular, the sixth KPI is the cost scalability. This KPI is applied only to the scenarios that utilize the Ethereum public blockchain, that incurs a transaction cost. We defined the cost scalability as the ratio of gas cost over the number of challenges. The results are shown in the following figure.

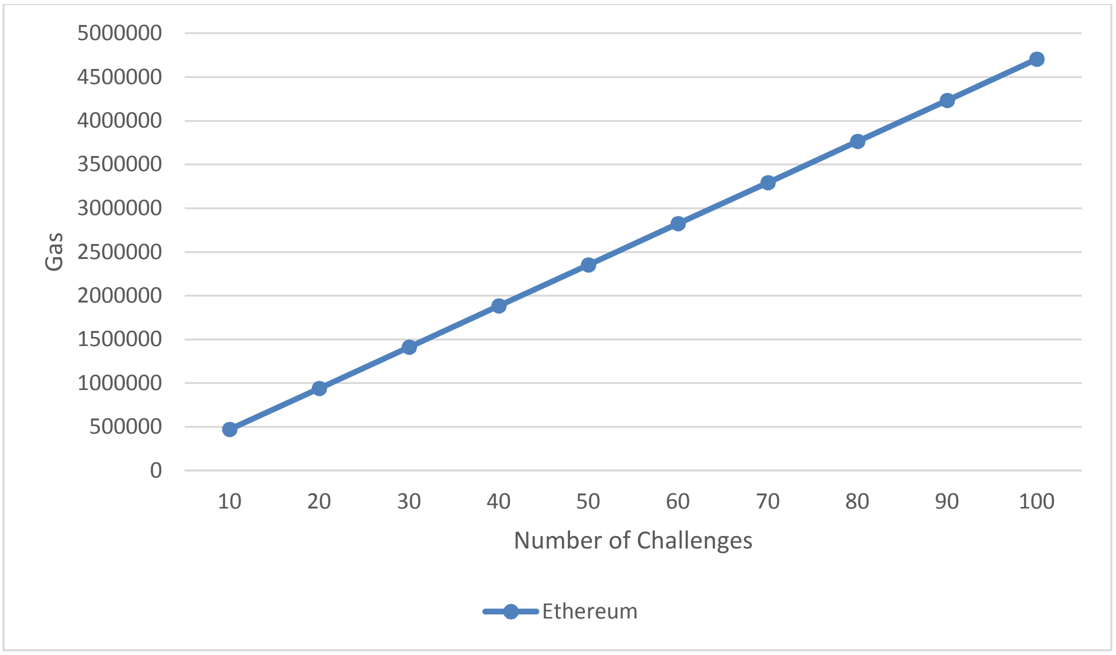


Figure 45: Cost scalability

As we can see from the figure above, cost scalability is linear, which is desirable. The cost scalability will be remained linear in all the scenarios that involve the public Ethereum blockchain. The last KPI is the time scalability. In Ethereum, to record a new challenge on the blockchain we send a transaction. A block in private Ethereum can store 74 transactions related to challenges. Each block is mined in ~ 15 seconds. So, the time scalability for the scenarios that involve private Ethereum is sub-linear. On the other hand, in public Ethereum, we cannot find out the exact number of transactions that can be stored in a block, because it depends on many external factors, such as the gas and the transactions sent by other nodes in the network. However, the scalability will remain sub-linear. In Fabric, we have already mentioned that a block can store 20 transactions and a block is added on the ledger on ~ 100 milliseconds. The time scalability for the scenarios is shown in the figures below.

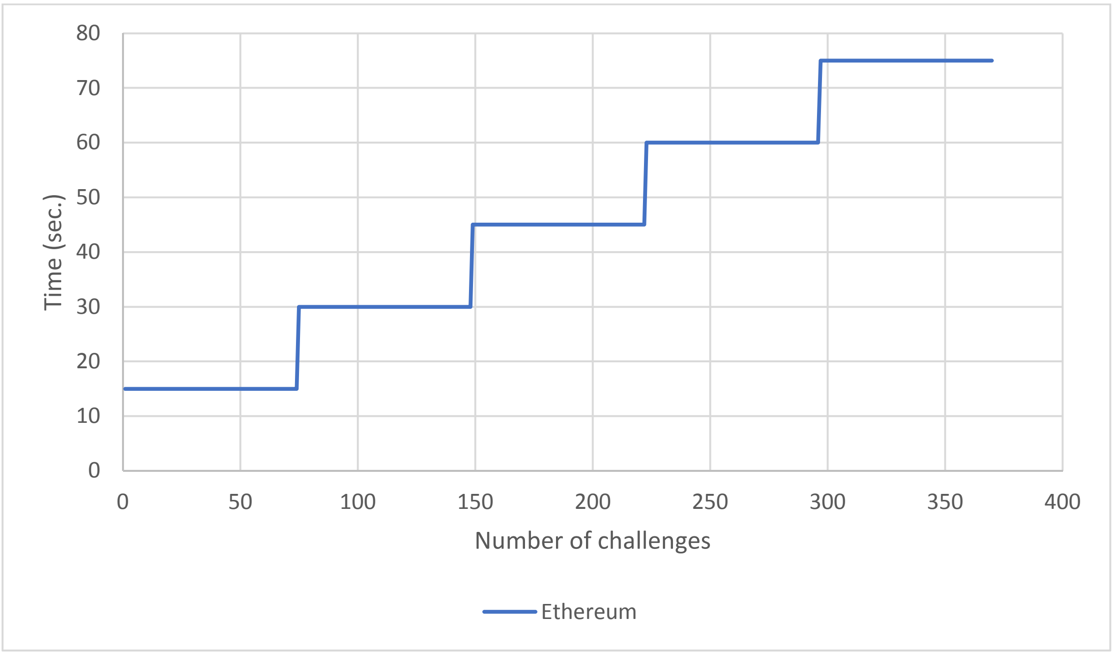


Figure 46: Time scalability for scenarios utilizing Ethereum

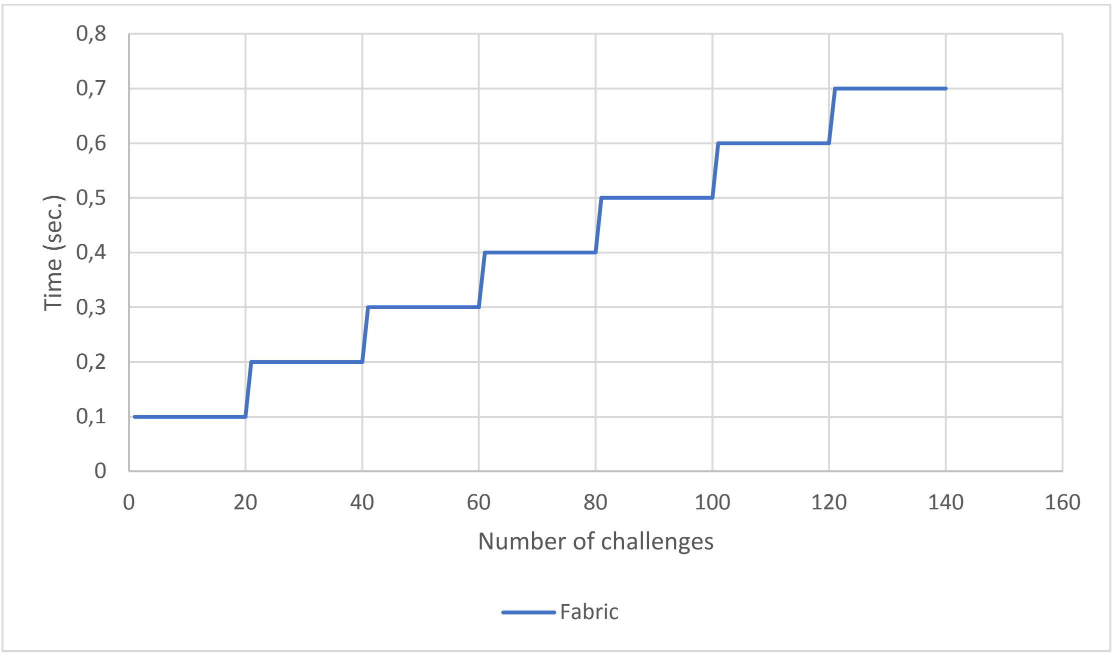


Figure 47: Time scalability for scenarios utilizing Hyperledger Fabric

6.4.5 Pilot requirements in emulation scenarios

The following table lists the requirements for the MRMG pilot from Deliverable 5.2 and presents how the emulation scenarios address these requirements. Some of the presented requirements refer to the final pilot application and are out of scope of the emulation. Such requirements are marked as “Not Applicable”.

Table 39: MRMG requirements and emulation scenarios

ID	**Name**	**Description**	**Emulation Scenarios**
REQ_ MRMG0.1	Unique identifiers for every actor	Each person interacting with the game or web application should have a unique identifier.	Each actor has a unique identifier (wallet) All scenarios
REQ_ MRMG1.1	Game Mobile application	Game challenges are accessible using the Android application	The mobile application is emulated as a React or JS application and every challenge is accessible through it All scenarios
REQ_ MRMG1.2	Joining any game challenge	Players can join any challenge by scanning the QR code or manually entering challenge ID.	Any player can join any challenge by entering the challenge unique ID All scenarios
REQ_ MRMG1.3	Unique identifier for challenges	Each challenge should have a unique identifier	Each challenge has a unique integer identifier All scenarios
REQ_ MRMG1.4	Record the time taken to complete a challenge.	Time should be recorded for each player, starting after joining the challenge till the player completes it.	Not Applicable
REQ_ MRMG1.5	Receive Clues / tasks	Players should receive unique clues / task when near the IoT beacons based on their challenge.	Not Applicable
REQ_ MRMG1.6	Skip any task	Players should be able to skip any task and receive location of next IoT beacon using the In-App tokens.	Players can skip any task by paying with In-App tokens All scenarios
REQ_ MRMG1.7	Purchase In-App tokens	Players can buy an unlimited amount of In-App token using the fiat currency	Players can buy In-App tokens using fiat currency In Scenario 1,2.4
REQ_ MRMG1.8	Points calculation	System should automatically calculate the points based on the time taken to complete any challenge	When a player completes a challenge, a fixed amount of points is automatically added to his account All scenarios
REQ_ MRMG2.2	Rewards distribution	System should automatically add the rewards to the players account after the challenge ends.	Players get rewards based on their collected points All scenarios

REQ_ MRMG3.1	In-App Advertisement video	Player should be given the option to view advertisements while playing a challenge.	A player can view an advertisement at any time All scenarios
REQ_ MRMG3.2	Advertisement reward	Player should receive tokens for viewing the advertisement.	A player receives In-App tokens when he watches an advertisement All scenarios
REQ_ MRMG3.3	Advertising viewability data	Every ad viewability data should be recorded as a transaction on the blockchain.	Advertisements are emulated as smart contracts, thus viewability data is recorded on the blockchain All scenarios
REQ_ MRMG4.1	Assert marketplace	Players can buy and sell In-App asserts on the blockchain	Not Applicable
REQ_ MRMG4.2	Assert trading data	Every asset traded on the platform should be recorded as a transaction on the blockchain.	Not Applicable
REQ_ MRMG5.1	Web Application	Web application for designing new challenges and uploading advertisements.	In the emulation, there is a client application where an actor can upload a new challenge and new advertisements All scenarios
REQ_ MRMG5.2	Access control to the web services	Access control to the web services, based on the role of the user.	Not Applicable
REQ_ MRMG7.1	Offer rewards	Rewards can be offered to the players through challenges and advertisement videos.	Players get rewards, when they complete a challenge or watch an advertisement All scenarios
REQ_ MRMG7.2	Rewards data	Rewards should be added and recorded on the blockchain.	Every reward for every player is recorded on the blockchain All scenarios
REQ_ MRMG8.1	Publish new advertisements	Ads manager should publish any ad video using the web application	An ads manager can publish an advertisement at any time All scenarios

6.4.6 Pilot system performance KPIs in emulation scenarios

The results for all the KPIs and scenarios are summarized on the following table.

Table 40: System performance KPIs for the MRMG emulation scenarios

KPI	Name	Target	Scenario 1	Scenario 2	Scenario 3	Scenario 4
KPI_MRMG_1	Public ledger execution cost	As low as possible	Achieved. 50164 gas used on average.	Achieved. Zero cost for actions in private Ethereum and 36437 gas used on average for actions in public Ethereum.	Achieved. Zero cost.	Achieved. Zero cost for actions in Fabric and 36437 gas used on average for actions in public Ethereum.
KPI_MRMG_2	Response time for write requests	< 3 s	Not achieved. 15 s on average due to the use of public Ethereum, where a block is generated within 15 s.	Not achieved. 15 seconds on average due to the use of public Ethereum, where a block is generated within 15 seconds.	Achieved. 2.1993 s on average.	Achieved partially. 2.1692 s on average for actions in Fabric and 15 s for actions in Ethereum.
KPI_MRMG_3	Response time for read requests	< 1 s	Not achieved[7] 1.5836 s on average, due to the communication with an RPC server.	Not achieved. 1.5836 s on average, due to the communication with an RPC server.	Achieved. 0.0242 s on average.	Achieved partially. 0.0255 s on average for actions in Fabric and 1.4738 s for actions in Ethereum.
KPI_MRMG_4	BLE beacon detection time	< 4 s	Not Applicable	Not Applicable	Not Applicable	Not Applicable
KPI_MRMG_5	Throughput	> 222 read and > 133 write transactions per second	Not achieved	Not achieved	Achieved for both read and write transactions.	Achieved partially. Only for actions in Fabric.
KPI_MRMG_6	Scalability – cost	Linear or sublinear	Achieved. Linear	Achieved. Linear	Not applicable	Not applicable
KPI_MRMG_7	Scalability – time	Linear or sublinear	Achieved. Sublinear	Achieved. Sublinear	Achieved. Sublinear	Achieved. Sublinear

6.4.7 Conclusions

As in other pilot-inspired scenarios, the experimental evaluation of the MRMG scenarios shows the gains, in terms of transaction cost and time, that can be achieved for the mobile gaming use case when a private ledger is used instead of a public ledger or when public and private ledgers

[7] Not achieved for the current implementation. However, if we had a node in Ethereum the average time would be 0.0450 second

are combined. Nevertheless, we should also consider the properties of each ledger type, in terms of trust, privacy, security and transparency. Therefore, using only one private ledger is better for implementing the main functionality of the mobile game, while utilizing two ledgers is better for managing tokens, redeeming rewards and processing advertisements, because these actions require a higher level of trust and transparency, which the public ledger can support.

7 Analysing SOFIE business platforms with System Dynamics

7.1 Introduction

The term platform, as in *digital business platform,* was first introduced to economics by Rochet and Tirole in 2006 [RT06]. *A business platform* is a business model that creates value by facilitating interactions between two or more interdependent groups, usually consumers and producers. The term digital business platform was actually introduced before the term business platform. Therefore, they do not have generally distinct meanings.

An example of such digital business platforms is the modern smart phone with its added function of the (per platform) application store. In smartphones, the more consumers use a certain platform, say Apple's iPhone, the more interesting and valuable it is also to the app producers, and vice versa. In order to make this value creation happen, platforms harness and create large, scalable networks of users and resources that can be accessed on demand.

SOFIE aims to provide a secure open federation of IoT systems so that those systems can exchange data under their own rules and voluntarily cooperate to achieve their goals. Many of the rules and impediments, but also incentives and opportunities, are about business aspects, rather than about technical issues. Therefore, a business perspective evaluation of the SOFIE architecture, and the mechanisms being developed in its context, is prudent. In addition to the aspects discussed earlier, such as those of a distributed system and an open (business) platform, additional business aspects are also discussed and evaluated here.

Key questions to understand include whether and under what conditions a Federated Open Platform (System) can successfully be bootstrapped, grown, and sustained. In addition, whether the constituting subsystems also grow and prosper and how potential gains are sustainably distributed across the constituents.

The remainder of this section deals with these questions in the following order:

1. What are the key forces that determine digital business platform adoption, and sustainable growth for each SOFIE pilot (or use-case more generally)? This is answered with pilot-specific models in each business platform model, as part of section 7.2.
2. Why federation is difficult to achieve, even if the value proposition of the federation is notable? This question is explored in section 7.1.2.
3. How the security provided by the SOFIE federation model affects platform adoption?

The System Dynamics [Ste00] approach is used to investigate these questions. In section 7.2, a brief introduction on how to read System Dynamics diagrams is provided.

This deliverable cannot and will not present a general model of a digital business platform success factors and forces, because SOFIE provides a framework architecture and not a specific platform architecture. Thus, the business platforms are always business area specific. Such generalized models could be *sketched* in deliverable D4.5.

7.1.1 Business platform stages of maturity

This report proposes to view the lifecycle of the business platform in three main phases. Most notably, this life cycle is not linear, but new phases are added as the platform matures and new versions of the platform are produced iteratively [RBM09]:

1. The *R&D stage* is about studying and applying research methodologies and conceptualizing the digital business platform, explaining general information about it, defining system boundaries, determining the prominent entities and key factors, detecting the causal

relationships and underlying plausible processes among them, studying regulatory conditions, consulting with technology providers, and finally, depicting the system dynamics regarding technical issues and commercial strategies.

2. The *implementation stage* is about launching the small-scale pilot to evaluate its functionality with actual data. In this phase, the network paths between entities are examined to detect the barriers and potential sources of failures. Further, a generalised model of a federated business platform is presented.
3. The *commercialization stage*: In this stage, commercial, monetary logic dominates, and technical activities consist of maintenance and customer support. Owners of the platform try to retain their market share and improve the number of customers and producers.

In a modern commercialized stage business platform, all the stages – R&D, implementation, and commercial activity – exist simultaneously and are ongoing. Thus, later stages are typically added to the activity of the entity providing the business platform and none of the earlier stages are completely removed.

7.1.2 On how a federation creates value and the difficulties of federating

We have defined and discussed business platforms federation and *IoT systems federation* in Section 2.1.3. In a *federation of IoT systems* (silos), which in the SOFIE sense are considered to be DLT based business platforms, they are connected together via combining the DLTs so that agreed upon transactions are valid and trusted also from the point of view of other ledgers in the whole federation.

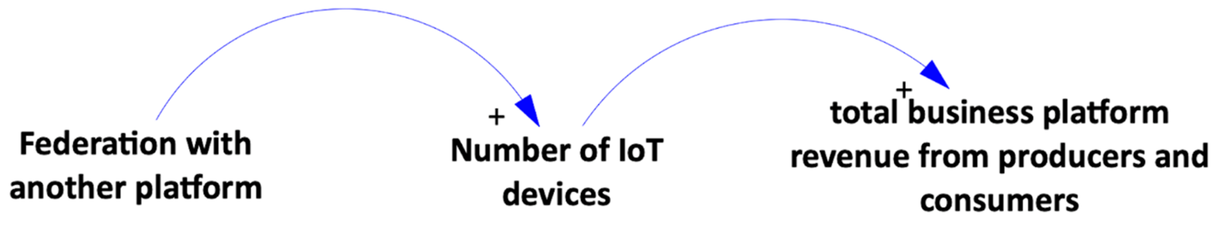

Federation with another platform
+
Number of IoT devices
+
total business platform revenue from producers and consumers

Figure 48. Federation adds to the revenue potential within an IoT business platform by increasing the number of devices connected to it.

Figure 48 demonstrates the straightforward economic logic of how the federation can scale up the ecosystem and potentially also the value generation with basically a flip of a federation switch.

On the other hand, Figure 49 illustrates the difficulty in actually enabling such a collaborative effort from a business platform owner point of view. While competitive logic can directly capture the value of revenue as money in double entry bookkeeping, the cooperative logic of a federation requires that *a perception of value* acts as an input flow to a monetary stock, which is formed via accounting rules subject to interpretation. This hypothesis is described with the System Dynamics stocks and flow model in Figure 49. In this figure, the value chain that produces monetary value via straightforward value capture as money from the competitive logic of the markets, is shorter and (more) deterministic. While the value chain producing value from cooperation, (the topmost chain of stocks and flows), is lengthier, and more importantly, subject to unclear interpretative rules, thus uncertain. This diagram shows why federation is difficult, even though it provides potentially enormous value for the federating platforms.

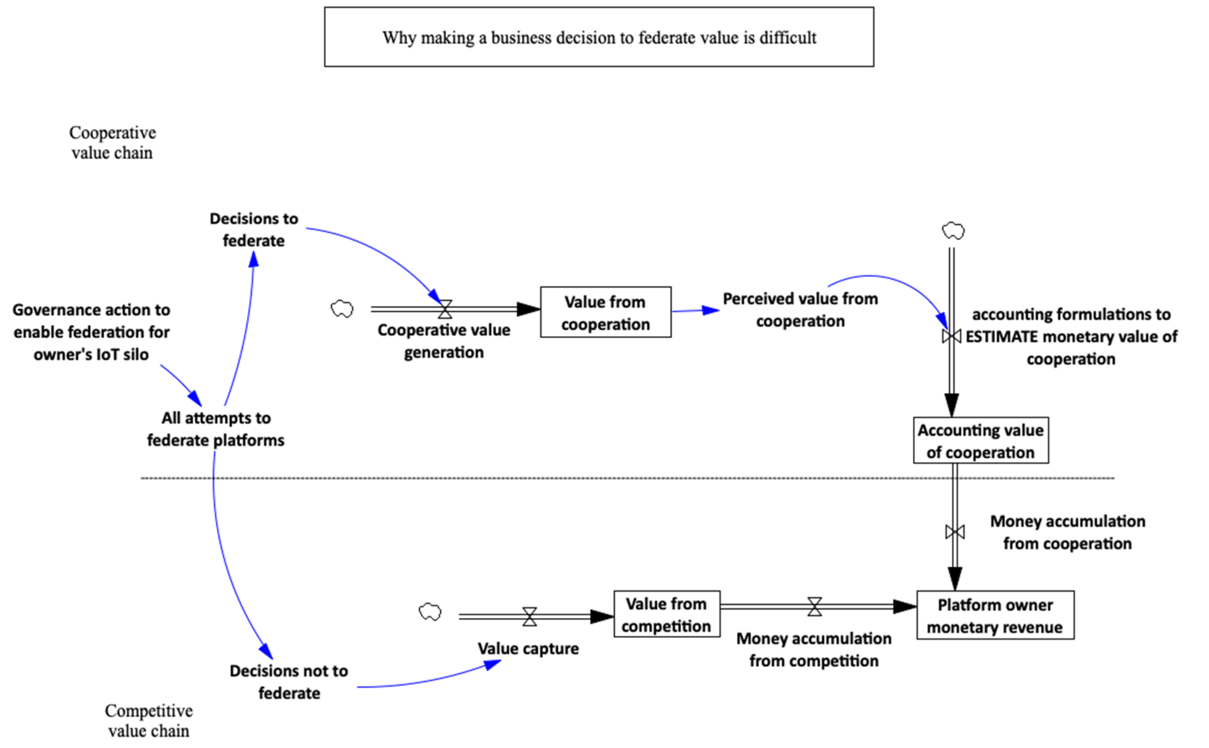

Why making a business decision to federate value is difficult
Cooperative value chain
Decisions to federate
Governance action to enable federation for owner's IoT silo
All attempts to federate platforms
Cooperative value generation
Value from cooperation
Perceived value from cooperation
accounting formulations to ESTIMATE monetary value of cooperation
Accounting value of cooperation
Money accumulation from cooperation
Value capture
Value from competition
Money accumulation from competition
Platform owner monetary revenue
Decisions not to federate
Competitive value chain

Figure 49. Two distinct value chains combined to produce the revenue ("bottom line", bottom right) of a business platform: the cooperative value generation chain at the top of the diagram and the competitive value generation chain at the bottom.

In addition, the time dimension differs for these two value chains [Kap00] due to the accounting interpretation rules expressing *perceived value* from cooperation as monetary value to be transferred to the bottom line. Our position is that if we had better accounting units of *non-rival currencies* [NE19], the demonstration of such value would be easier than it is now. Currently, we are forced to rely on transforming the abundance-based value created in the cooperation value chain, to the units of the competitive value chain, which has scarcity-based units of account and economic logic.

7.2 Models of individual SOFIE Business Platforms

This section describes the SOFIE Business platforms in terms of System Dynamics. The aim is to provide a high-level business-oriented evaluation, which is based on a scientifically rigorous modelling methodology. We start by introducing System Dynamics and its core concepts briefly.

System Dynamics (SD) is a modelling methodology, which can be used to model and understand the dynamic, typically time dependent, behaviour of complex systems. It recognises the interconnected, circular relationship in any complex system. It can thus model properties about the whole system, which do not easily emerge by looking at the systems' individual elements. Systems can be presented as *Causal Loop Diagrams (CLD)*, where elements of the system are presented as nodes of a graph, and causalities connecting them as arrows. These can be extended to *stock and flow diagrams,* which make quantitative analysis and thus simulations possible.

One noteworthy factor is that the success of a business platform typically depends on other systems or platforms. The methodology used can take into account other involved ecosystems, such as, in our case, the decentralised Ethereum network or social consensus such as reputation, which can impact the success of the SOFIE framework, and thus the SOFIE business platforms. The business platforms studied in the following sections are based on the SOFIE framework utilising business platforms serving as the pilots of the SOFIE project.

7.2.1 Food Supply Chain

The goals for this platform are:

1. To accelerate automation, digitalization and secure information sharing among the participants in the food supply chain.
2. To enhance food supply chain accountability by adding to the integrity of the workflows and relationships, and by engaging food consumers throughout the value chain.
3. To provide food business networks and consumers the means to verify product quality and safety, as those are transferred from the field to the market.
4. To build trust and support collaboration between companies in the supply chain, each with its own processes and operations, without the need of a centralized authority.

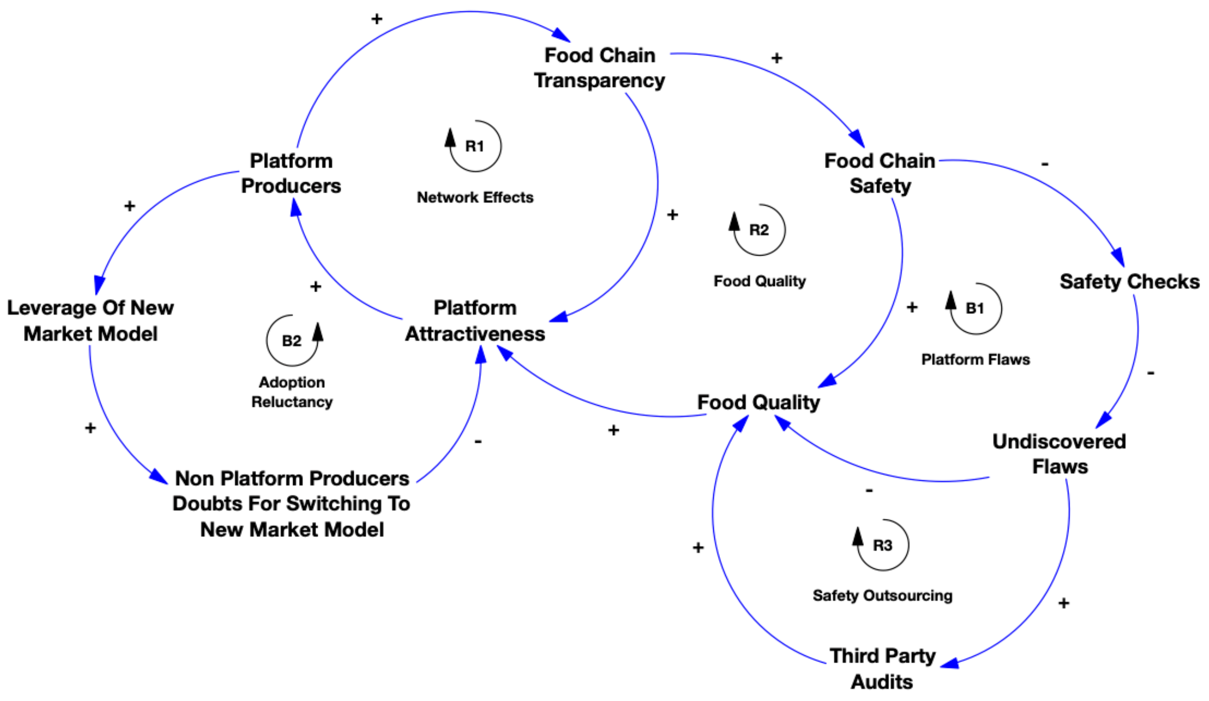


Figure 50. CLD of FSC pilot business platform

The workings of the food supply chain platform are summarized in the causal loop diagram in Figure 50. A transparent food chain provides the means for ensuring accountability and integrity of the workflows and for the verification of food quality and safety. Food quality and safety can be additionally safeguarded with third-party audits.

Network effects (reinforcing loop R1): The larger the platform's producer base, the higher the food chain's transparency, as a result of the producers' profile of openness[8]. A transparent food chain in turn adds to the platform's attractiveness.

[8] We note that such causality will require *governance action* from the consortium organization: there should be rules on how openness is practiced and what are the associated benefits. SOFIE expects IoT devices to massively proliferate. If these devices are connected and measure important quality attributes in the food supply chain and then *automatically and massively* write them into a sufficiently open trustworthy accounting ledger, such as a DLT, radical

Food quality (reinforcing loop R2): A transparent food chain is incentivising producers and retailers to follow a rigorous approach with respect to the products that they decide to release to the market. Therefore, transparency preserves the safety of the food chain and the quality of the food that is being produced and supplied to consumers, adding to the platform's attractiveness.

Platform flaws (balancing loop B1): A transparent and safe food chain may lessen the motivation for safety checks. However, the system will always present flaws, irrespective of its transparency. Flaws in the FSC may impact the food's quality and, in turn, the platform's attractiveness.

Safety outsourcing (reinforcing loop R3): The more flaws in the system, the higher the need for third party audits. These will, finally, offer an additional layer of protection for the quality of the food.

Adoption reluctance (balancing loop B2): The more producers in the platform, the more they will promote its new market model. However, a new market model always comes with uncertainty, which may raise a reluctance to follow, especially from non-adopters with a settled business in agriculture.

In summary, for the FSC, trustworthy transparency in the food chain is a crucial determinant for its success. Transparency is vital for the food's quality and safety, through which reinforcing loops R1 and R2 will dominate the balancing loops. Reinforcing loop R3 is complementing the platform's transparency by adding an extra layer of control. On the other hand, dominance of the balancing loops implies a platform sustainability failure. We also note that creating such a radical transparency platform as envisioned in this pilot may best start with a group of roughly equal peer producers, who have high mutual trust. This would require a level of confidentiality, which a DLT approach coupled with federation can provide.

7.2.1.1 **Simulation of the FSC platform as a Stock and Flow model**

Figure 51 extends the model of Figure 50 to a more detailed simulation model. The new model can demonstrate and simulate some of the hypothesized key forces affecting the success of the food supply chain platform. The simulation model depicted in Figure 51 is an augmentation and modification of the CLD depicted in Figure 50, also introducing stock variables. The most important stock in the model, is the stock *Platform Producers S* in the upper left-hand corner. This is the number of producers (farmers) who have joined the platform, and thus one of the main success metrics for the FSC platform from the business point of view.

openness may become a good business strategy. Regulatory oversight is likely required to utilize such transparency to systematically benefit the overall quality.

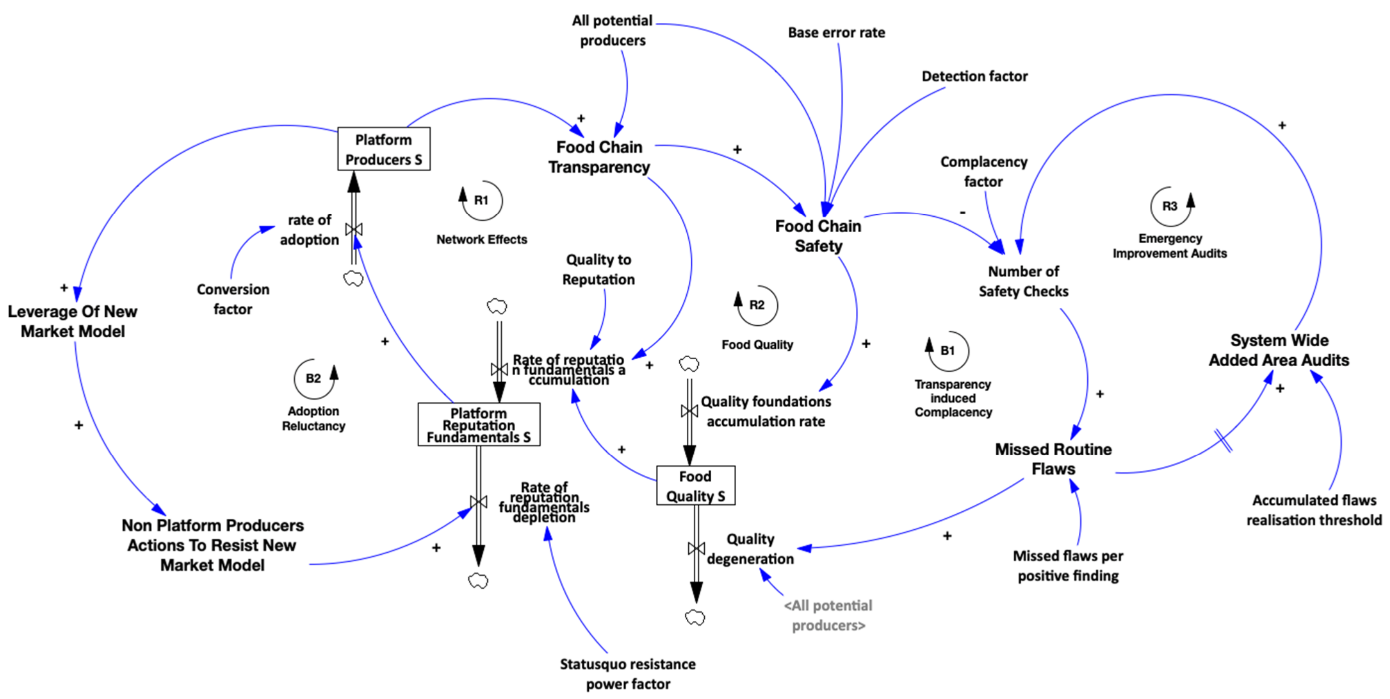

All potential producers
Base error rate
Detection factor
Platform Producers S
Food Chain Transparency
R1
Network Effects
Food Chain Safety
Complacency factor
R3
Emergency Improvement Audits
rate of adoption
Conversion factor
Quality to Reputation
Number of Safety Checks
Leverage Of New Market Model
R2
Food Quality
B2
Adoption Reluctancy
Rate of reputation fundamentals accumulation
B1
Transparency induced Complacency
System Wide Added Area Audits
Quality foundations accumulation rate
Platform Reputation Fundamentals S
Missed Routine Flaws
Food Quality S
Rate of reputation fundamentals depletion
Accumulated flaws realisation threshold
Non Platform Producers Actions To Resist New Market Model
Quality degeneration
Missed flaws per positive finding
<All potential producers>
Statusquo resistance power factor

Figure 51. A simulation model of the food supply chain platform, presenting the core forces affecting the adoption and sustainability of the platform.

Via loop R1, *Network Effects*, the *Platform Producers S* (stock) drives and produces greater *Food Chain Transparency* [9] for those who have joined. This added *Food Chain Transparency* increases the reputation in feedback loop R1, and also to added *Food Chain Safety*, which in turn leads to self-reinforcing dynamics of accumulating *Food Quality S* (R2). Both loops R1 and R2 contribute to *Rate of reputations fundamentals accumulation*, which together determine the inflow of stock *Platform Reputations Fundamentals S*, which is the accumulated reputation of the platform. This can be deflated via the separate outflow of *Rate of reputation fundamentals depletion*, which is regulated by the exogenous action of *Statusquo resistance power factor* (we discuss this below).

The created greater *Food Chain Safety,* via the complacency of all participants, leads to reduced *Number of Quality Checks* in balancing loop B1, *Transparency Induced Complacency.* This leads to *Missed Routine Flaws*, which deplete the quality via *Quality degeneration* completing the endogenous success mitigating loop B1. A typical governance action attempting to mitigate the B1 dynamic is to tailor order tests with a significant reaction delay, thus creating the R3 loop, *Emergency Improvement Audits.*

Adoption Reluctancy, or the success hindering loop B2, is kickstarted when the *Platform Producers S* via publicity (or otherwise) engage the *Leverage of New Market Model*; this causes those who have invested in status quo business models (both in real terms and in psychological terms) to oppose the new business platform. This is depicted in *Non-Platform Producers Actions To Resist New Market Model* causing *Rate of reputation fundamentals depletion*. The power of such action is regulated by the *Status quo resistance power factor*, which can be modelled via the ratio of negative publicity to all publicity.

[9] We assume governance rules in support of such behaviour.

7.2.1.2 The simulation runs and the status quo resistance power factor

The variable which controls the effect of the B2 loop, to exit the valve depleting *Platform Reputation Fundamentals S*, and thus negatively affecting the reputation accumulation, is named *Status quo resistance power factor*. It describes the ratio of negative publicity of the platform to all publicity of the platform. This parameter can be quantitatively approximated by surveying the major news outlets, e.g., at least in the past, the major newspapers. The number of all articles mentioning the platform is observed, as well as the number of the articles classified as being "negative" by a human observer. This way a ratio can be calculated. We assume that this variable is exogenous, not directly controlled by the other "forces" described in this model.

It is well known that many large organisations seek to understand and affect the amount of negative and positive publicity with respect to their business area. In the established area of FSC, a large retailer may already be engaged in it, while the regulator may also be engaged in such publicity observation. Those are some of the existing players within an existing business ecosystem. And whoever seeks to disrupt such established markets, by means of radical transparency through the studied platform, for example, would easily face resistance to building a reputation for the new business model and its corresponding platform.

We studied the impact of the *Status quo resistance power factor* parameter through simulation. Figure 52 shows results for a simulation run with *Status quo* 0.2, and Figure 53 with *Status quo* 0.03. We see that when varying this parameter by a factor of about 10, we can affect the *Platform Producers S* stock (i.e. the number of farmers adopting and contributing to the platform), which is a direct reflection of the success of the platform, quite drastically in the long run. With *Status quo resistance power factor* 0.2, as shown in Figure 52, the platform fails, while with 0.03, as shown in Figure 53, it succeeds in capturing the market. Note that the failure in the first case would happen no matter the technical merits, quality and safety improvements, and regardless of their exponential accumulating nature. In the latter case, the effect of the decrease in negativity in the publicity is large enough to the point of making the platform successful, as the reputation is not depleted.

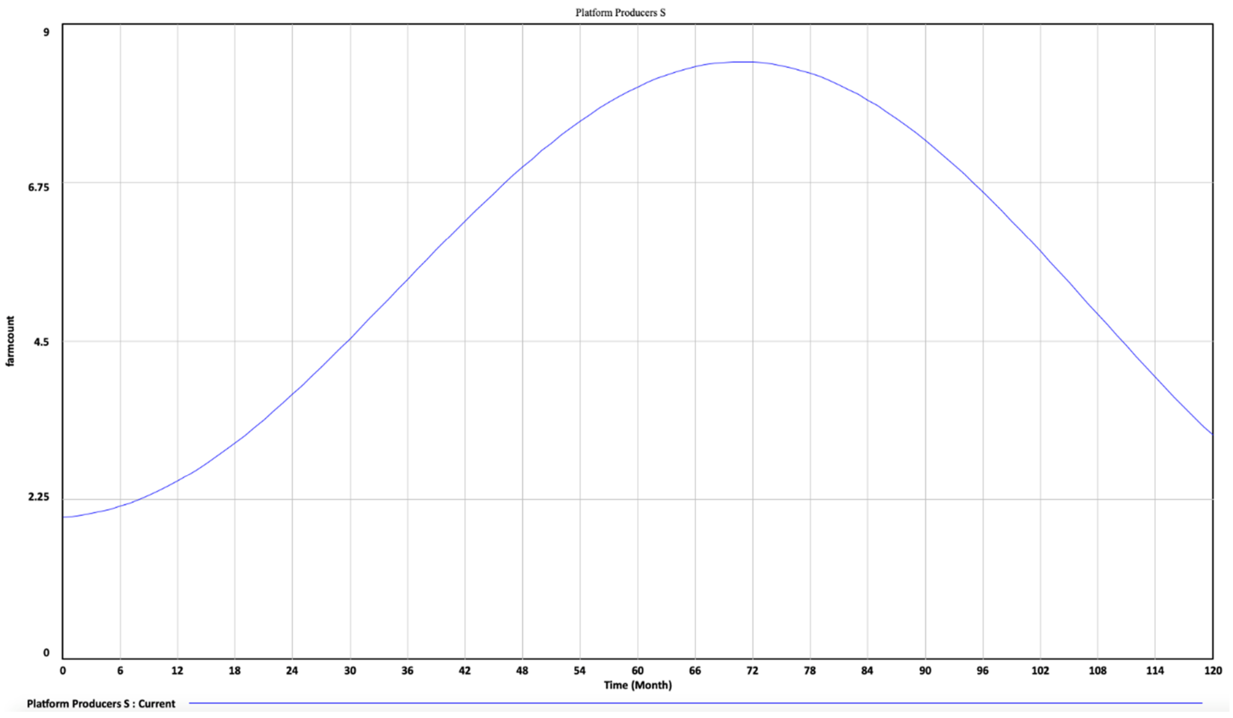


Figure 52. Number of farmers in the platform (Status quo resistance power factor set to 0.2).

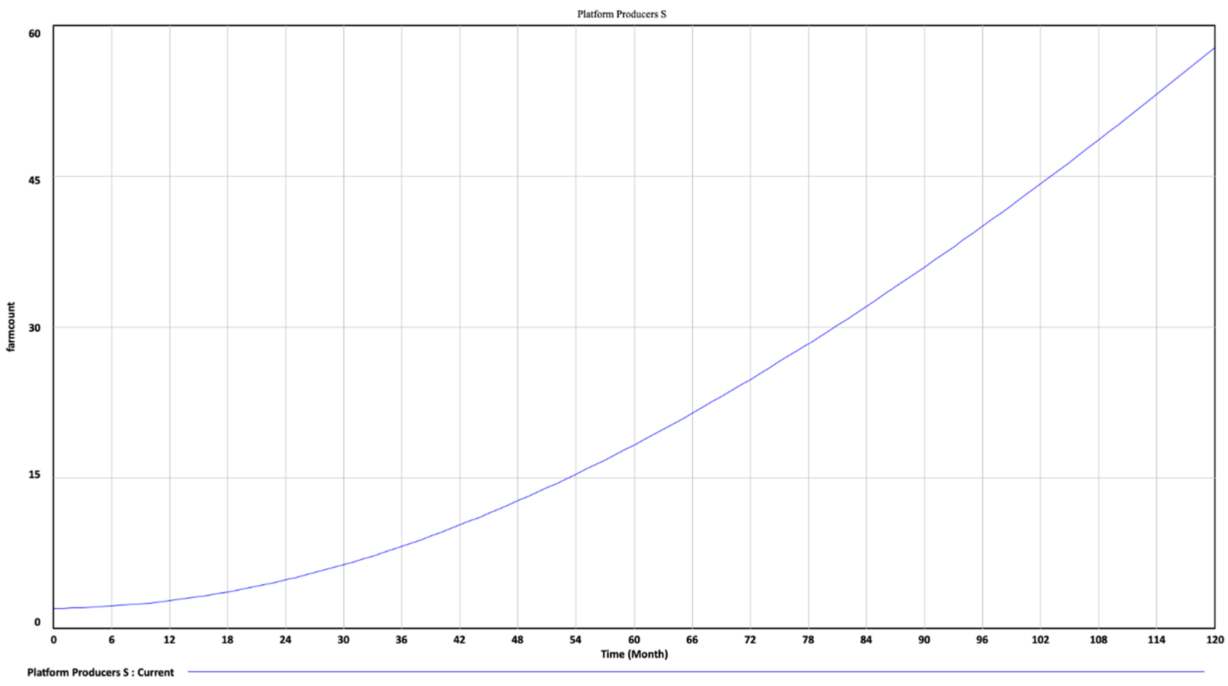


Figure 53. Number of farmers in the platform (Status quo resistance power factor set to 0.03).

In order to more thoroughly validate our model, we need to feed it with enough empirical data of important variables and run enough simulation runs, so that we can see if the model, with properly calibrated constants, can reproduce the observed time dependent behaviour of the other variables. The results observed here demonstrate the power and potential significance of the approach in general and of this study in particular, but are only indications since the employed data is synthetic. Expert interviews (conducted by the System Dynamics study team during the January 2020 SOFIE plenary and expanded later) have validated the model qualitatively to some extent, but empirical calibration via simulations remains as required future work.

Future work for this platform will concentrate on how the bootstrapping and accumulation of the reputation of the platform could be designed to be as resistant to such exogenous inputs as possible. Note that many aspects have been abstracted out, including DLTs, federation, and interledger aspects. Such aspects will be considered in the future and, even if not explicitly introduced in the model, they could be accounted for through proxy variables and different parameter choices.

However, already at this stage, it can be said that exponential real-world growth feedback should be engaged as early as possible, to create a publicity dynamic which is self-sustaining. Intuitively, the earlier in the growth process the status quo power affects such growth loops exogenously, the more efficiently it can hinder the growth of the endogenous reputation fundamentals. A key question to be answered via simulations with synthetic and real-world data is, what time dependent distributions make the exogenous publicity parameter least impacting the long-term reputation.

Another practical observation from studying this platform in detail through SD is the following: If one seeks to make such platforms dominant in the market, an appropriate strategy would be to make every effort for the new business model to be *complementary*, not directly competing with the business of those who decide *not to join* early.

7.2.2 Decentralized Energy Data Exchange

The vision for business in this pilot is to enable seamless access to data with a few clicks of the (citizen) data owner, regardless of where the person lives, or what existing energy networks are in place [Sof20]. The pilot exploitation plans and activities are directed towards smart meter data operators (TSOs/DSOs). The idea is to make a novel digital infrastructure available that will allow TSOs/DSOs to grant access to data, track the process of who provides and receives data through their platform and create immutable evidence for auditing and security purposes.

The business model under investigation focuses on the DSOs/TSOs operating the access control of energy consumption data and providing them with SOFIE based digital infrastructure on an annual licence fee. The solution would add value to existing, running platforms, so that the DSOs/TSOs can make a shortcut into sharing data and skip the planning/development phase on their existing platform. The second step of the business plan consists of getting various service providers to start using SOFIE adapters and components to get consumer data through the DSOs/TSOs and sell, e.g., flexibility services. The business model with the service providers would be based on sharing the revenue stream based on the new customer base and their data that they get through the digital infrastructure enabled through the SOFIE adapters and components.

The SD model of the DEDE pilot is under development and it and its results are planned to be reported in D4.5.

7.2.3 Decentralized Energy Flexibility Marketplace

The business vision [Sof20] in this pilot is that via the IoT and sensors proliferation it is possible to obtain useful information to achieve accurate power forecasts, avoid reverse power flows, and otherwise balance the grid loads based through mitigating actions.

Specific intermediate goals of this platform are:

1. Flexibility management of Distributed Energy Resources (DERs). A key energy resource exploited in the pilot is Electrical Vehicles (EVs). A particular property exploited is the fact that EVs are mobile and can (within limits) balance loads at various points of the grid.
2. To accelerate automation and automated communications between *Distribution System Operators (*DSOs) and DERs. Another specific and exploited feature of the pilot is the hierarchical communication, command and control model enabled by the fleet of EVs.

An initial causal loop diagram for the DEFM pilot has been developed and is shown in Figure 54.

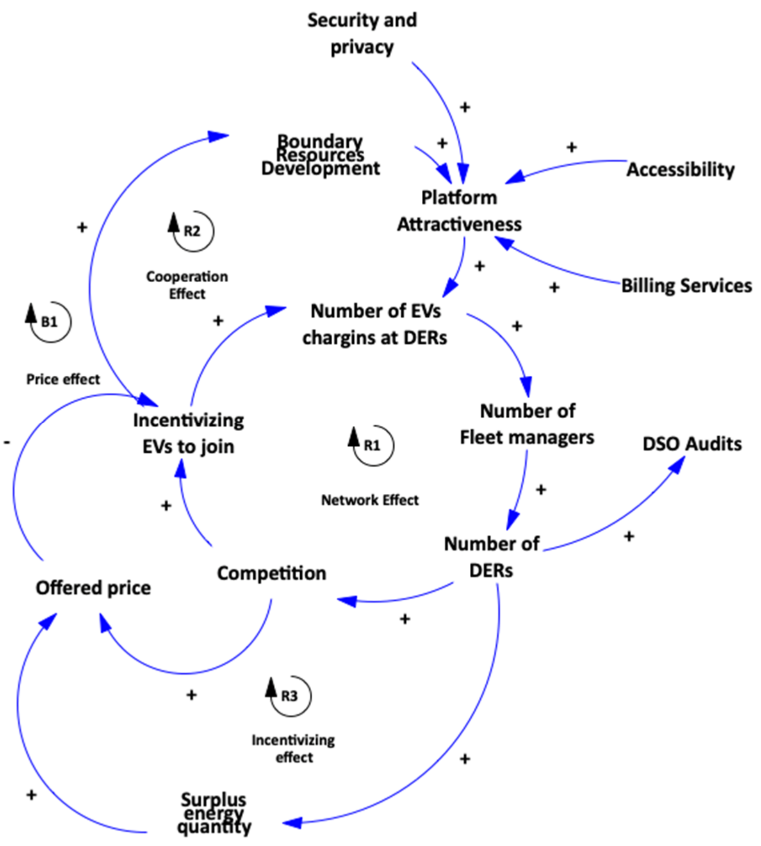


Figure 54. DEFM platform CLD.

For this diagram, the definitions of and the relations between components of the platform are described as follows:

Network Effect (Reinforcing loop R1): More EVs enhance the number of fleet managers which, in turn, increase the number of localized renewable energy producers, DERs, to satisfy the electricity demand in the market. In fact, DERs are producers in the electricity producer-consumer digital business platform. Due to the increase of the number of DERs, the retail competition in the market also increases. The DERs compete to react to a demand-response signal and generate more local electrical energy which is supplied in the auction market through a recommendation system with incentives. These flows of causality form a network effect for EVs and DERs in Figure 54. Similarly, the growth on the DERs side, results in an increased number of EVs to join the platform.

Cooperation Effect (Reinforcing loop R2): In addition to incentivizing EVs to join the platform via the recommendation system, the cooperation loop leads to improvement of other boundary resources [Sti+20], which are either useful technologies or cooperative agreements. Those resources can be used to promote system effectiveness and attractiveness, for example, automation for smart grids, fixed-term smart contract agreement, using the fleet of batteries, machine learning to predict patterns, etc.

Price Effect (Reinforcing loop R3): The profit price in the auction market recommended by the system should be reasonable enough to be accepted by both the buyer and the seller. If the asking price is lower than the marginal cost, the market fails [Hag09].

Feedback loop (reinforcing R4) indicates that the more localized and renewable energy DERs produce, the more the platform can invest in attracting an increased number of DERs.

In addition to platform attractiveness, factors which are presented by the technical foundation of SOFIE for either of DERs or EVs, for instance, accessibility and billing services, the price strategy is another vital factor in this pilot. The business process in this pilot is the bidding market. The offered price by DERs should be properly set so it is both profitable and able to attract EVs to join to the platform.

The SD Stock and Flow model for the DEFM pilot is also under development and will be reported together with its results in D4.5.

7.2.4 Mixed Reality Mobile Gaming

This platform seeks to achieve the following business goals [Sof20]:

1. Game discovery: How to reach a mass market given mobile application ecosystem restrictions.
2. Player Value: Making the gaming experience fun and valued by the player.
3. Revenue opportunity: New business case with potential revenues from real-world elements.
4. IoT device scale: Identify devices fit for gaming use cases that can easily grow to a global scale.

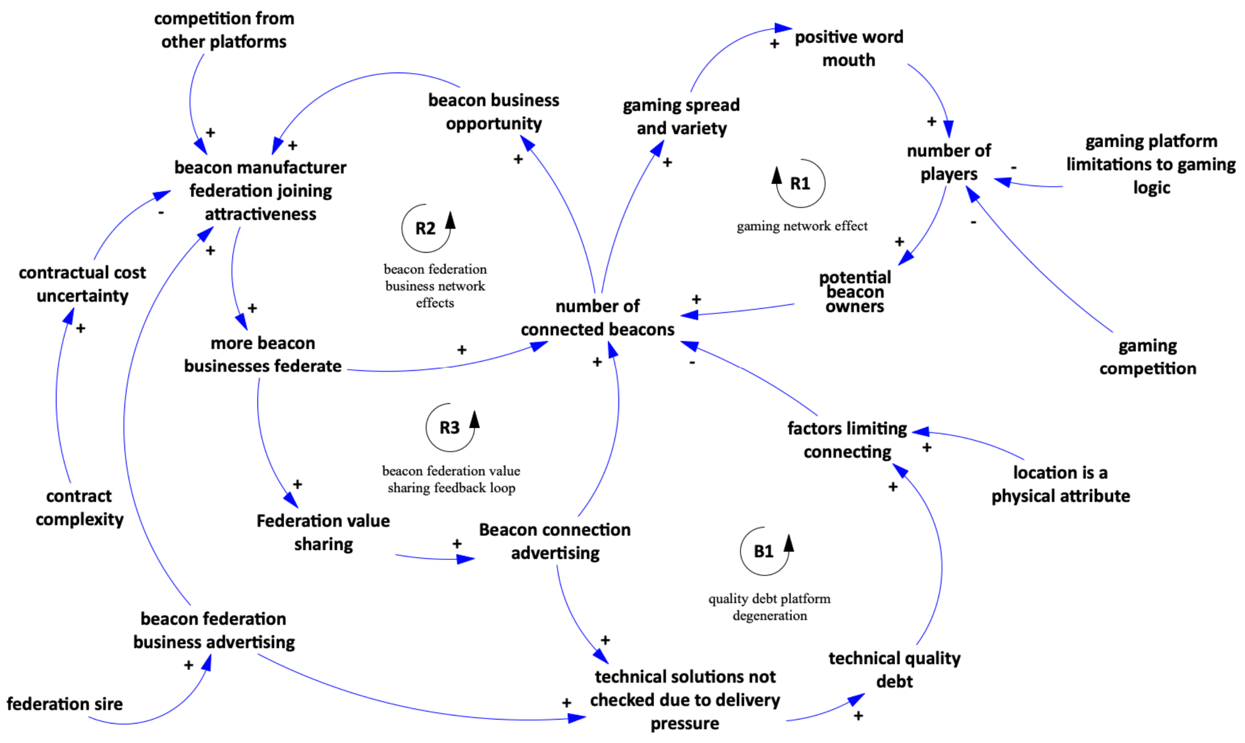

competition from other platforms
beacon business opportunity
gaming spread and variety
positive word mouth
number of players
gaming platform limitations to gaming logic
R1
gaming network effect
beacon manufacturer federation joining attractiveness
R2
beacon federation business network effects
contractual cost uncertainty
potential beacon owners
number of connected beacons
gaming competition
more beacon businesses federate
R3
beacon federation value sharing feedback loop
factors limiting connecting
location is a physical attribute
contract complexity
Federation value sharing
Beacon connection advertising
B1
quality debt platform degeneration
beacon federation business advertising
technical quality debt
federation sire
technical solutions not checked due to delivery pressure

Figure 55. CLD model of key determinants of platform growth via IoT-beacons, initiation by the parent company, and the technical quality degeneration as growth-limiting feedback loop (B1).

Figure 55 presents a CLD model of how IoT-beacons and gamers interact in the MRMG platform. Beacons are local network elements, which have a specific and distinct location and can, at the minimum, provide an anchor for the mobile game ensuring that the user has physically been close to the IoT-beacon. This makes mobile gaming interesting because it combines strong physical location checks with the gaming logic.

The logic of Figure 55 can be described as follows. Increasing *the Number of connected beacons* increases the *gaming variety and geographical spread* for the players, which increases the positive reputation among potential players (*positive word of mouth*). This, in turn increases the *number of players* of the game, who are the first early adopters of beacons, thus increasing the *number of potential beacon owners.* This will of course increase the number of connected beacons again. Together these elements and causalities form a reinforcing *gaming network effects* (R1) loop.

The description of the leftmost reinforcing loop of *beacon federation business network effects* (R2) is as follows: As the *number of connected beacons* increases, the *number of beacon business providers* also increases*,* which has the positive effect of some of those manufacturers joining this specific (federation) platform via the *number of beacon business federations*. This will naturally increase the *number of connected beacons* in this business platform ecosystem.

Loop B1, *quality debt platform degeneration,* has its main start in the element *technical solutions not checked due to delivery pressure.* This variable feeds on all the promises the advertising pushes to market. Without sufficient quality checks, *technical quality debt* increases, increasing all kinds of *factors limiting connecting* (e.g., user-interface not being good enough, subcontractor not knowledgeable, ineffective installation guide, connectivity problems, etc.). This will start to decrease the number of connected beacons, a loop detrimental to the success of the platform.

Loop R3, *beacon federation value sharing feedback loop,* depicts the notion that the beacons are included in the game via federation. Thus, federation rules can establish an entry fee or operating fee type of payment. A portion of these payments can be used to fund advertising, thus bringing more attention and connections to the federation.

The CLD models are not detailed enough for simulation; so, at best we can offer qualitative insights at this stage. Successfully bootstrapping and maintaining a two-sided market, as depicted in Figure 55, will create a growing business. The main self-reinforcing dynamic is loop R1, which depicts the main growth cycle of the gaming company's successful game. If the negative exogenous factors in R1, *gaming platform limitations* and *gaming competition*, or the business depleting loop B1 dominate, the platform might never properly enter the exponential self-reinforcing growth cycle.

We note that in this platform federation is an explicit tool to promote the success of the platform by making it possible from the ground up to build *a meta-platform*—a federation based platform of IoT silos—which does not in any way prevent beacons being connected also to other business platforms (or performing other functions). To answer the question of what makes the growth of the federation difficult in this case, the answer is that some initial investment is needed to properly kick-start the federation.

7.3 Conclusions from the System Dynamics study

One of our models was developed all the way to a simulation model: the FSC business platform model. From the early simulation results based on synthetic data, we were able to determine that negative publicity actions by those who invested in earlier, competing, non-transparent business models, can have a detrimental effect to platform growth, to the point of "killing" the platform. Also, that transparency makes the regulator's job more cost effective.

The other presented (CLD) models offer qualitative insights. In general, bootstrapping and reinforcing growth loops of the most important value generative properties of any platform will also lead to a thriving platform. The balancing loops can dampen or overcome the reinforcing dynamics, thus creating hindered growth or even demise of the platform business.

Federation potentially multiplies the ecosystem size fast. But enabling federation on a business platform of a significant commercial size, requires careful balancing of competitive and cooperative forces and value generation. Having non-rival goods as key units of account is hypothesized to help such efforts.

8 Conclusion

This deliverable contains the evaluation results from WP4's second evaluation cycle and its submission (April 2020/M28) coincides with the second evaluation cycle completion milestone (MS14). Since the first evaluation deliverable D4.3 (initial submission June 2019, revised in December 2019), the deliverables containing the second version of the SOFIE federation architecture (D2.4, initially submitted July 2019 and revised December 2019), the second version of the federation framework (D2.5 – August 2019), as well as the initial platform validation (D5.2 – June 2019) and business platforms – pilot release (D3.3 – September 2019) have been submitted. These deliverables contain a more stable description of the SOFIE architecture and federation framework components and of the pilot systems, including the pilot scenarios and their objectives. This has allowed the evaluation and validation results contained in the current deliverable to be more concrete and focused.

Section 2 presented a high-level architecture evaluation, focusing more on the style and desired properties of the architecture, rather than very specific architectural components and structure. The text is based on the previous deliverable D4.3, confirming that the features identified in the previous deliverable remain relevant. Its content has been updated to take into account the validation and evaluation work and results that have been obtained up to this point.

Section 3 focused on the validation of the SOFIE approach, including the architecture, component, and pilot validation. The results and current validation status were presented in three validation matrices, corresponding to each of the three validation directions.

Section 4 focused on component evaluation. Compared to the first evaluation deliverable D4.3, further evaluation results were presented for the Interledger, Privacy and Data Sovereignty (PDS), and Identification, Authentication, and Authorization (IAA) components. A discussion and new results are also presented for the three other components: Semantic Representation (SR), Marketplace (MP), and Provisioning and Discovery (PaD).

Section 5 contained further results on IoT resource access evaluation. The new results include decentralized authorization for constrained environments, where Ethereum is interconnected with Hyperledger Fabric, access control for multi-tenant IoT systems, and decentralized interledger gateway architectures for enhancing the reliability of the interledger functionality.

Section 6 on evaluation scenarios followed the SOFIE pilots, generalizing them into pilot inspired use cases by including alternatives that are not necessarily selected in the SOFIE pilots, and uses emulation and simulation to consider the various tradeoffs of many potential alternative design decisions. The investigations include alternatives for a hierarchy and interconnection of different types of blockchains (public/private, or permissionless/permissioned, etc) and the impact of these choices. The new results included in this deliverable compared to the previous evaluation deliverable D4.3 include new evaluation scenario involving multiple ledgers (which include public/private Ethereum, Hyperledger Fabric, and Hyperledger Indy) and interledger technology, as well as end-to-end performance (execution cost, response time), throughput, and overall scalability results based on the defined system performance KPIs. The experience and results from the emulation scenarios will provide guidance to the pilot validation work in WP5.

Section 7 addressed business platform evaluation using the System Dynamics methodology. We first provided models of platform federation illustrating the benefits, but also explaining the difficulty in federating. Then we presented the results of our efforts to date to model the platforms inspired from the four SOFIE pilots through this methodology. For the FSC platform, we have been able to refine the models all the way to the point to obtain simulation-based investigations of key parameters using synthetic data and demonstrate the sensitivity of the model and the platform to a key exogeneous parameter. Our next most advanced model is for the MRMG platform, for which we still have qualitative results, but they are much more detailed and include the notion of federation. Modelling for the platforms will continue with the goal of having reasonably detailed models at least for the key aspects of the platforms validated through parameters obtained through the pilots and thoroughly investigated in deliverable D4.5.

The WP4 work that will be performed in the remainder of the project includes the completion of the validation activities and further extension of the evaluation results, which will take into account ongoing pilot validation work in WP5. This will include extending and introducing new scenarios alongside those considered in the evaluation work up to now. The key research work will focus on the joint analysis and comparison of emulation results and pilot validation results. This will verify the performance gains of the SOFIE platform in real conditions and validate its advantages. Also, it will make the final evaluation results available in a single comprehensive document, which includes the component requirements and the pilot requirements and KPIs considering both the emulation/simulation results of WP4 and the pilot validation results of WP5. This will be useful for the final overall architecture evaluation and the final version of the federation framework.

9 References

[AEN19] E. Arzoglou, T. Elo, and P. Nikander, "The Case of iOS and Android: Applying System Dynamics to Digital Business Platforms," Proc. ICCS 2019. In: J. Rodrigues et al. (eds) *Computational Science – ICCS 2019*, Lecture Notes in Computer Science, Vol. 11540, Springer, Cham, 2019.

[AJM14] R. Azarderakhsh, K.U. Järvinen, and M. Mozaffari-Kermani, "Efficient Algorithm and Architecture for Elliptic Curve Cryptography for Extremely Constrained Secure Applications," *IEEE Transactions on Circuits and Systems*, Vol. 61, No. 4, pp. 1144–1155, 2014.

[Ber06] D.J. Bernstein, "Curve25519: New Diffie-Hellman Speed Records," Proc. 9th International Conference on Theory and Practice of Public-Key Cryptography (PKC 2006), pp. 207–228, New York, NY, USA, April 2006.

[Ber+12] D.J. Bernstein et al., "High-speed high-security signatures," *Journal of Cryptographic Engineering*, Springer, Vol. 2, No. 2, pp. 77–89, 2012.

[But16] V. Buterin, "Chain Interoperability," R3 Report, September 2016. Available online: https://www.r3.com/wp-content/uploads/2017/06/chain_interoperability_r3.pdf

[Cai+18] W. Cai, Z. Wang, J.B. Ernst, Z. Hong, C. Feng and V.C.M. Leung, "Decentralized Applications: The Blockchain-Empowered Software System," *IEEE Access*, Vol. 6, pp. 53019-53033, 2018.

[Cle+14] R. de Clercq, L. Uhsadel, A. Van Herrewege, and I. Verbauwhede, "Ultra low-power implementation of ECC on the ARM Cortex-M0+," Proc. 51st ACM/EDAC/IEEE Design Automation Conference (DAC), San Francisco, CA, USA, June 2014.

[RBM09] M. de Reuver, H. Bouwman, and I. MacInnes, "Business model dynamics: a case survey," *Journal of theoretical and applied electronic commerce research*, Vol. 4, No. 1, pp. 1–11, April 2009.

[Del+16] A. Delignat-Lavaud, C. Fournet, M. Kohlweiss, and B. Parno, "Cinderella: Turning Shabby X.509 Certificates into Elegant Anonymous Credentials with the Magic of Verifiable Computation," Proc. IEEE Symposium on Security and Privacy (SP), May 2016.

[Dül+15] M. Düll et al., "High-speed Curve25519 on 8-bit, 16-bit, and 32-bit microcontrollers," in *Designs, Codes and Cryptography*, Springer, Vol. 77, No. 2-3, pp. 493-514, 2015.

[Fot+16] N. Fotiou, T. Kotsonis, G.F. Marias, G.C. Polyzos, "Access control for the Internet of Things," Proc. ESORICS International Workshop on Secure Internet of Things, pp. 29–38, 2016.

[Fot+19] N. Fotiou, I. Pittaras, V.A. Siris, S. Voulgaris, G.C. Polyzos, "Secure IoT access at scale using blockchains and smart contracts," Proc. 8th IEEE WoWMoM Workshop on the Internet of Things: Smart Objects and Services (IoT-SoS), Washington DC, USA, June 2019.

[Fot+20] N. Fotiou, I. Pittaras, V.A. Siris, S. Voulgaris, G.C. Polyzos, "OAuth 2.0 authorization using blockchain-based tokens," NDSS Workshop on Decentralized IoT Systems and Security (DISS), San Diego, CA, USA, February 2020.

[FSP18] N. Fotiou, V.A. Siris, G.C. Polyzos, "Interacting with the Internet of Things using Smart Contracts and Blockchain Technologies", Proc. Security, Privacy, and Anonymity in Computation, Communication, and Storage (SpaCCS 2018), Melbourne, Australia, December 2018.

[Ger+18] S. Gerdes et al., "An architecture for authorization in constrained environments," IETF Draft, October 2018.

[GH01] J.F. Gubrium and J.A. Holstein, *Handbook of interview research: Context and method*, Sage Publications, 2001.

[Glo18] Global Platform, “TEE System Architecture v1.2,” December 2018. Available online: https://globalplatform.org/specs-library/tee-system-architecture-v1-2/.

[Hag09] A. Hagiu, “Two-Sided Platforms: Product Variety and Pricing Structures,” *Journal of Economics & Management Strategy*, Vol. 18, No. 4, pp. 1011–1043, 2009. Available online: https://www.onlinelibrary.wiley.com/doi/full/10.1111/j.1530-9134.2009.00236.x/

[Haa+18] J. Haapola et al., “Peer-to-Peer Energy Trading and Grid Control Communications Solutions' Feasibility Assessment Based on Key Performance Indicators,” Proc. 87th IEEE Vehicular Technology Conference (VTC Spring), 2018.

[Har+12] D. Hardt et al., “The OAuth 2.0 Authorization Framework,” RFC 6749, Standards Track, IETF, October 2012.

[HS13] M. Hutter and P. Schwabe, “NaCl on 8-Bit AVR Microcontrollers,” Proc. International Conference on Cryptology in Africa, published in *Progress in Cryptology – AFRICACRYPT 2013*, A. Youssef, A. Nitaj, and A.E. Hassanien (eds), Lecture Notes in Computer Science, Vol. 7918, Springer, 2013.

[JBS15a] M. Jones, J. Bradley, and N. Sakimura, “JSON Web Token (JWT),” RFC 7519, Standards Track, IETF, May 2015.

[Kor+19] Y. Kortesniemi, D. Lagutin, T. Elo, N. Fotiou, “Improving the Privacy of IoT with Decentralised Identifiers (DIDs),” *Journal of Computer Networks and Communications*, Hindawi, 2019.

[Kov+19] M. Kovatsch et al., “Web of Things (WoT) Architecture,” retrieved August 2019. Available at: https://www.w3.org/TR/wot-architecture/.

[KL08] J. Katz and A.Y. Lindell, “Aggregate message authentication codes,” Proc. The Cryptographers’ Track at the RSA conference on Topics in Cryptology (CT-RSA), 2008.

[KM00] R. Kaplinsky and M. Morris, “A handbook for value chain research,” University of Sussex, Institute of Development Studies, 2000. Available online:
http://www.fao.org/fileadmin/user_upload/fisheries/docs/Value_Chain_Handbool.pdf

[KSD18] S. Kalra, R. Sanghi, and M. Dhawan, “Blockchain-based real-time cheat prevention and robustness for multi-player online games,” Proc. 14th International ACM Conference on emerging Networking EXperiments and Technologies (CoNEXT), pp. 178–190, Heraklion, Greece, December 2018.

[Lag+19] D. Lagutin, Y. Kortesniemi, N. Fotiou, V.A. Siris, “Enabling Decentralized Identifiers and Verifiable Credentials for Constrained Internet-of-Things Devices using OAuth-based Delegation,” NDSS Workshop on Decentralized IoT Systems and Security (DISS), San Diego, CA, USA, February 2019.

[Nak08] S. Nakamoto, “Bitcoin: A Peer-to-Peer Electronic Cash System,” White Paper, October 2008. Available online: https://bitcoin.org/bitcoin.pdf

[NE19] P. Nikander and T. Elo, “Will the data markets necessarily fail? A position paper,” Proc. 30th International Telecommunications Society (ITS) European Conference, Espoo, Finland, June 2019.

[PD16] J. Poon and T. Dryja, “The Bitcoin Lightning Network: Scalable off-chain instant payments,” White Paper, January 2016.
Available online: https://lightning.network/lightning-network-paper.pdf

[Ree19] D. Reed et al., “Decentralized Identifiers (DIDs) v0.13: Data Model and Syntaxes for Decentralized Identifiers,” Draft Community Group Report, W3C, June 2019. Available online: https://w3c-ccg.github.io/did-spec/

[RT06] J.-C. Rochet and J. Tirole, "Two-Sided Markets: A Progress Report," The RAND Journal of Economics, Vol. 37, No. 3, pp. 645-667, 2006.

[Sho01] V. Shoup, "A proposal for an ISO standard for public key encryption," Cryptology ePrint archive, Report 112, 2001. Available online: https://eprint.iacr.org/2001/112

[Smart16] Deliverable D3.2, "Key performance indicators for p2p energy trading communications," H2020 Project Peer to Peer Smart Energy Distribution Networks (P2P-SmarTest), 2016. Available online:

http://www.p2psmartest-h2020.eu/deliverables

[Sir+19a] V.A. Siris, D. Dimopoulos, N. Fotiou, S. Voulgaris, G.C. Polyzos, "Interledger Smart Contracts for Decentralized Authorization to Constrained Things," Proc. 2nd Workshop on Cryptocurrencies and Blockchains for Distributed Systems (CryBlock 2019), in conjunction with IEEE INFOCOM 2019, Paris, France, April–May 2019.

[Sir+19b] V.A. Siris, D. Dimopoulos, N. Fotiou, S. Voulgaris, G.C. Polyzos, "OAuth 2.0 meets Blockchain for Authorization in Constrained IoT Environments," Proc. 5th IEEE World Forum on Internet of Things, Limerick, Ireland, 2019.

[Sir+19c] V.A. Siris, D. Dimopoulos, N. Fotiou, S. Voulgaris, G.C. Polyzos, "IoT Resource Access utilizing Blockchains and Trusted Execution Environments," Proc. Global IoT Summit, Aarhus, Denmark, 2019.

[Sir+19d] V.A. Siris, P. Nikander, S. Voulgaris, N. Fotiou, D. Lagutin, G.C. Polyzos, "Interledger Approaches," *IEEE Access*, Vol. 7, pp. 89948-89966, 2019.

[Sir+20] V.A. Siris, D. Dimopoulos, N. Fotiou, S. Voulgaris, G.C. Polyzos, "Decentralized authorization in constrained IoT environments exploiting interledger mechanisms," *Computer Communications*, Vol. 152, pp. 243-251, February 2020.

[Sof20] Project Use Cases, SOFIE website. Accessed: 15.4.2020. Available online: https://www.sofie-iot.eu/about/project-use-cases

[SMS07] B. Sunar, W.J. Martin, and D.R. Stinson, "A Provably Secure True Random Number Generator with Built-In Tolerance to Active Attacks," *IEEE Transactions on Computers*, Vol. 56, No. 1, pp. 109-119, January 2007.

[Spo+19] M. Sporny et al., "Verifiable Credentials Data Model 1.0: Expressing verifiable information on the Web," Draft Community Group Report, W3C, September 2019.

[Ste00] J.D. Sterman, *Business dynamics: systems thinking and modeling for a complex world*, McGraw-Hill Education, 2000.

[Sti+20] K. Still et al., "Platform Economy - Interactions & Boundary Resources: Checklist for Companies," Technical Report, Tampere University of Technology, 2017. Available online: https://tutcris.tut.fi/portal/files/13267284/INTERACTIONS_BOUNDARY_RESOURCES_CHECKLIST_2017_10_27.pdf

[SysFl19] Deliverable 10.1, "Report on the selection of KPIs for the demonstrations," H2020 Project EU-SysFlex, February 2019.

[VV15] V. Vogelsteller and B. Vitalik, "ERC-20 token standard," Tech. Rep., 2015. Available online: https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20.md

[Wik20a] "Federation," *Wikipedia*. Accessed: 15.4.2020. Available online: https://en.wikipedia.org/wiki/Federation

[Wik20b] "Metcalfe's law," *Wikipedia*. Accessed: 15.4.2020. Available online: https://en.wikipedia.org/wiki/Metcalfe%27s_law

[Woo18] D.G. Wood, "Ethereum: A Secure Decentralised Generalised Transaction Ledger," Ethereum Yellow Paper, December 2018. Available online: https://ethereum.github.io/yellowpaper/paper.pdf