SOFIE - Secure Open Federation for Internet Everywhere

779984

DELIVERABLE D4.2

Testbed and Emulation Environment Design and Setup

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	28.2.2019
Author(s)	Dmitrij Lagutin (AALTO), Pekka Nikander (AALTO), Santeri Paavolainen (AALTO), Vasilios Siris (AUEB-RC), Nikos Fotiou (AUEB-RC), Spyros Voulgaris (AUEB-RC), George Polyzos (AUEB-RC), Mikael Jaatinen (LMF), Petri Laari (LMF)
Responsible person	Dmitrij Lagutin (AALTO), dmitrij.lagutin@aalto.fi
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

1. Introduction

The main aim of the SOFIE project is to enable interoperability between IoT silos using distributed ledger technologies. The SOFIE solution will be tested in four real-world pilots within three diverse topic areas: energy, food chain, and mixed reality gaming.

The main objective of SOFIE's evaluation work package (WP4) is the qualitative and quantitative evaluation of SOFIE's architecture and framework. The results of the validation and evaluation work conducted in WP4 and the recommendations based on these results will be fed into the architecture and framework design in WP2, and the business integration in WP3 (Business Platforms Integration). The plan for SOFIE's evaluation work has been described in deliverable "D4.1 - Validation and Evaluation Plan" [Vas2018].

This deliverable describes the design and setup of SOFIE’s testbed and emulation environment. The purpose of this work is twofold:

- SOFIE's testbed spans multiple project partners and allows testing various distributed ledger technologies and their interaction with IoT devices on a wider scale.
- SOFIE's emulation environment emulates certain aspects of SOFIE pilots and related more general use-cases, which allows realistic testing of various solutions without deploying them yet in pilot environments.

The SOFIE testbed and emulation environment will be used in the evaluation work during the rest of the project, with the first evaluation results to be reported in deliverable "D4.3 - First Architecture and System Evaluation Report", due in June 2019.

2. SOFIE Testbeds

This section provides an overview of SOFIE's testbeds. The purpose of the testbeds is dual: (a) to enable testing various distributed ledger technologies (DLTs) without worrying about the transaction costs and latency, (b) to protect the confidentiality of our work in progress, (c) to allow testing interconnection between DTLs and IoT devices. The interconnected SOFIE testbed is shown in Figure 1 below. Subsequent subsections describe local testbeds of partners, various options for interconnecting local testbeds, and how access to the KSI blockchain is provided.

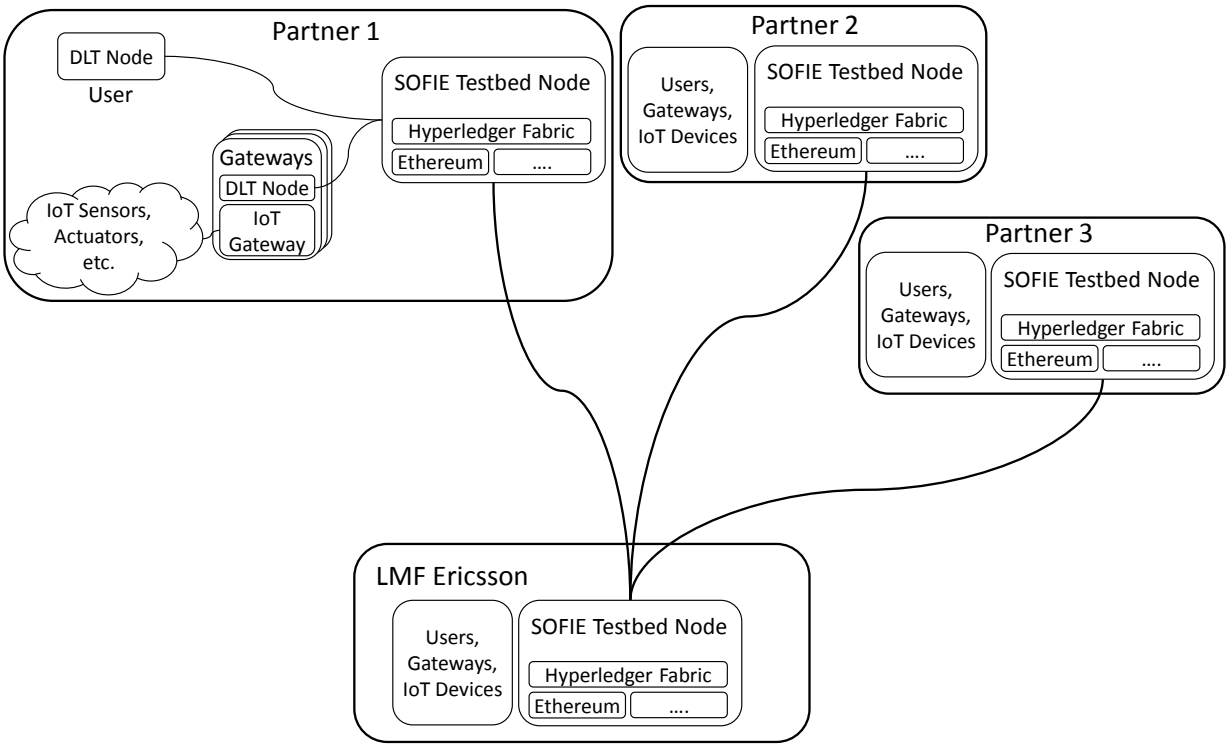

Figure 1. Detailed overview of the current SOFIE testbed

The current SOFIE testbed is composed of local partners' testbeds (currently AALTO, AUEB, and LMF Ericsson), which are connected with each other through the LMF Ericsson testbed node. The local testbed deployments include nodes of the selected DLTs (Ethereum, Hyperledger Fabric, etc.), IoT devices, IoT gateways, authentication and authorization services, as well as tools that facilitate development and deployment.

User wanting to use the testbed would run an own DLT node and should possess necessary tokens for the used DLT. Afterwards, the user can interact with IoT devices through DLTs and smart contracts.

2.1 Local Testbeds

This section provides a detailed overview of each partner's local testbed.

2.1.1 AALTO

Aalto's local testbed is mainly used for research and it may also be used for teaching in the future. The testbed will contain the following DLT components:

- An Ethereum miner, which operates in the context of the Nanopool mining pool. Ether gained from mining can be used for smart contract experimentation on the real Ethereum network.

- An archival node on the main Ethereum network for research purposes. An archival node, in addition to storing the blockchain history, also stores additional information such as the blockchain state history information normally retained by full nodes only for a limited number of blocks. Such information is useful for analysing the Ethereum network.
- Access to Guardtime's KSI Blockchain.
- An Ethereum node on a SOFIE private Ethereum network.
- A Hyperledger Fabric node on a SOFIE private network.
- A Hyperledger Indy node on a SOFIE private network.

In addition, the testbed will contain multiple IoT-related components located in working area (shown in Figure 1) of the Aalto's SOFIE group.

Figure 2. Location of Aalto's IoT testbed

The IoT setup at Aalto includes IoT gateways (such as Raspberry Pis), which are connected to sensors (temperature, humidity, etc.), actuators, and other devices such as LED lights. The gateways also run instances of relevant DLT nodes, which allows interaction between the DLTs and actual IoT devices.

2.1.2 AUEB-RC

The AUEB local testbed is currently focused on experimentation with IoT device integration, and user authentication and authorization. It consists of the following elements:

- A private Ethereum chain implemented using:
  - Two Virtual Machines (VMs) acting as miners and RPC servers.
  - One VM that implements auxiliary functions (bootstrap node, monitoring tool).

  - One Android smartphone running the Geth[1] Ethereum client.
- Local Ethereum nodes connected to the Rinkeby and Ropsten Ethereum test networks
- An instance of the WS02 identity server[2], which is based on open standards and open source principles and provides identity and access management functions.
- An instance of the OAuth2.0 server[3], which implements an open authorization framework that provides delegated authorization to protected resources.
- A VM running Mozilla's Thing Gateway[4], which allows monitoring and controlling devices through a single secure web interface, using the Web of Things standard[5].
- A Hyperledger Fabric network.
- A Hyperledger Indy node.
- Hash-lock based interledger functionality between the public Etherium test networks and the private Ethereum deployments and between Ethereum and Hyperledger Fabric.
- Access to Guardtime's KSI Blockchain.

AUEB is additionally planning to provide:

- A Hyperledger Quilt deployment.

Furthermore, AUEB will deploy Raspberry Pis connected to sensors (temperature, humidity, etc.), actuators, and other devices such as LED lights. Some of these devices will run Mozilla's Thing Gateway.

2.1.3 LMF Ericsson

LMF Ericsson's local testbed is used for SW integration and validation before delivery to evaluation and pilot deployment. It consists of the following elements:

- Access to Guardtime's KSI Blockchain.
- One Ethereum node participating in the SOFIE private Ethereum network.
- One Ethereum node participating in the Rinkeby Ethereum test network.
- One Hyperledger Fabric node participating in the SOFIE private Hyperledger Fabric network.
- One Hyperledger Indy node participating in the SOFIE private Hyperledger Indy network.
- Supporting components that are used to manage and monitor the validation environment (logging, monitoring, access control, backups, etc.).
- The Staging and "production" environments that will be used to host the local testbed components, as well as all tools required by WP3 Continuous Integration and Continuous Delivery.

LMF Ericsson will replicate the node configurations from Aalto or AUEB for the nodes that participate in SOFIE private DLTs.

2.2 Testbed Interconnection

Some of the local deployments mentioned in the previous section will be interconnected, forming a federated testbed among partners. This interconnection can be achieved using various means, each of which induces performance-security tradeoffs. In the following we discuss the various interconnection options as well as their tradeoffs. We have chosen the second option "Interconnection using public IPs over the Internet firewall" for the current

[1] https://geth.ethereum.org/downloads/
[2] https://wso2.com/identity-and-access-management/
[3] https://github.com/bshaffer/oauth2-server-php
[4] https://iot.mozilla.org/gateway/
[5] https://www.w3.org/WoT/

SOFIE testbed. If necessary, we may modify the testbed to use another solution for interconnection during the course of the project.

The following table summarizes the possible interconnection methods and their properties.

Table 1. Testbed interconnection design choices

Interconnection Method	Advantages	Disadvantages
Public IPs	• Very easy to setup/extend	• Requires reachable IPs/ports • Low security
Public IPs with firewall	• Easy to setup/extend • Prevents 3rd parties from joining the blockchain network	• No confidentiality • Some management overhead
VPN	• Secure • Public IPs are not required, except for VPN server	• Management overhead

2.2.1 Interconnection Using Public IPs over the Internet

This is the simplest form of interconnection. The nodes in the local deployments are configured with a publicly accessible IP address and they are interconnected using the peer-to-peer (P2P) mechanisms provided by each ledger technology. Using this approach, the testing network can be easily set up and extended. However, this approach requires reachable IP/port pairs, which is not always easy to achieve (especially in corporate networks). Furthermore, this approach introduces security risks: in addition to the traditional security risks that the communication over the public Internet entails, there are some new, DLT-specific risks. For example, the official implementation of the Ethereum blockchain cannot be easily configured in a way that prevents 3rd parties to monitor the local chain, or even participate in its mining process. It should be noted here that the blockchain specific P2P protocols provide integrity protection.

2.2.2 Interconnection Using Public IPs over the Internet with Firewall

This method is similar to previous one with the difference that it uses a firewall (e.g., iptables tool in Linux kernel) in order to prevent outside parties from joining the blockchain network. In a nutshell this method can be used for turning a permissionless blockchain technology into persmissioned. However, this method does not provide confidentiality, hence all DLT-specific messages can be monitored by an outside party. Still, using this method the testbed can be easily set up, although it entails some management overhead when extending the testbed (e.g., updating firewall rules). Furthermore, and similarly to the previous method, it requires reachable IP/port pairs.

2.2.3 Interconnection over Virtual Private Network (VPN)

The final method under consideration is the use of a VPN service (some of the consortium members are experienced in setting up testbeds over OpenVPN). This method is the most secure of all, as it provides confidentiality and access control. Furthermore, assuming a star topology, only the IP address/port of the VPN server must be reachable. On the other hand this is the hardest method to configure and manage, since it requires certificate generation, node configuration, etc. Furthermore, the VPN server must have high availability to assure adequate overall testbed availability.

2.3 KSI Blockchain Access

KSI signatures are server based, meaning that signing data requires online access to the KSI service. The verification of the signatures can be done both offline and online. There are two options for access to KSI:

- KSI Software Development Kit (SDK)
- Catena middleware

The KSI SDK provides the lowest level of integration. It enables "full access" to the KSI functions (signing, extending, verifying) and lets the integrator fine-tune all possible options. As a consequence, it leaves many common challenges, such as signature storage and extension, to the integrator to solve.

Catena is middleware that is meant to address common integration challenges, such as asynchronous signing, signature persistence, and automatic extension. It provides the integrators with higher-level functionality, such as annotating signatures and linking signing events (data provenance), in order to reduce the effort for building a complete solution. Catena internally uses the aforementioned SDK to perform low-level KSI operations. The functionality of Catena is grouped and packaged into logical applications (Catena-KSI, Catena-DB, Catena-Prov) so that the integrator can choose which ones to deploy and use.

KSI SDK and Catena are not mutually exclusive: they can be used in combination if needed. This depends on the application type and the requirements for signing and verifying data.

2.3.1 Integration Resources

Depending on which integration option one wants to use, the following is needed to sign data, extend and verify signatures with KSI:

- access to KSI Gateway and KSI SDK to communicate with it; or
- access to Catena middleware and HTTP client library to communicate with it.

2.3.2 Try-out Servers

Catena

For the list of available API endpoints and reference documentation see https://tryoutcatena.guardtime.net. It is a Swagger UI that also allows direct execution of requests. This is the quickest option to get started with signing.

KSI Gateway

The KSI Gateway has two endpoints, one for aggregation/signing and one for signature extension. For tryout these are:

- http://tryout.guardtime.net:8080/gt-signingservice
- http://tryout-extender.guardtime.net:8081/gt-extendingservice

The KSI Gateway uses HMAC-based authentication built in the KSI protocol. The KSI Gateway endpoints are expected to return "Bad URI" when just used in web browser.

In addition, the KSI publications file URL is needed for signature extension and verification with the KSI SDK. For Guardtime provided KSI services, this is https://verify.guardtime.com/ksipublications.bin.

KSI SDK

The SDK is available for Java, .NET and C. The description and documents on how to access are provided to the try-out user account.

2.3.3 Interconnection with Public Testbeds

The SOFIE project will use and combine public/private ledgers, in order to evaluate basic blockchain features with our implementations. We will examine consensus protocols, such as Proof-of-Work (PoW), Proof-of-Stake (PoS), and Proof-of-Authority (PoA), and the use of Smart Contracts instead of legacy databases.

Ethereum is an open-source decentralized platform with the ability to deploy and execute smart-contracts. It also implements state and value-awareness. The online platform uses PoW, like Bitcoin. Though, we will utilise public testbeds, through the following clients:

- ***Parity-Ethereum*** software is developed in Rust programming language. It gives access in two public test networks. **Ropsten** for nodes with PoW and **Kovan** for nodes with PoA consensus. Parity allows us to analyse GAS consumption in executed transactions with a tracing tool.
- ***Go-Ethereum*** is developed in Go programming language. We are able to connect to **Rinkeby** testnet for PoA consensus. Go-Ethereum also allows us to investigate GAS consumption.

Both clients support the deployment and usage of smart-contracts. These contracts are created through **Solidity**. Solidity is a turing-complete language, designed to target the Ethereum virtual machine. So far, it is considered as the best choice for this task.

Apart from the public testbeds, we will setup private Ethereum networks to test various consensus mechanisms. Parity allows us to change the consensus mechanism, according to our needs. Go-ethereum does not provide this feature, but we are able to extend the client, implementing our own ideas for consensus.

We already use Javascript language (*web3.js*[6]), in order to retrieve statistics from the public Ethereum networks. The first version of our clients is also implemented Javascript. These clients are able to interact with Parity and Go-Ethereum respectively. Our plan is to switch to Python (*web3.py*[7]), for the same purposes.

[6] https://github.com/ethereum/web3.js
[7] https://github.com/ethereum/web3.py

3. Emulation Environment

In SOFIE evaluation work package (WP4) we plan to design, implement, and experiment with emulated (or simulated in some cases) versions of all four SOFIE pilots and the related more general use cases. The emulations will help us develop a better understanding of the interaction and interfacing of involved entities, as well as potential functionality and performance shortcomings that need to be addressed. Most importantly, they will allow us to experiment with a number of diverse configurations and more general settings, to assess their advantages and disadvantages, and to perform more educated decisions on the selection and future recommendations of appropriate components and parameters.

More specifically, our pilot emulations will allow us to address the issue of data management, that is, what data is stored where. This is far from trivial, as data will cross multiple databases, ledgers, and IoT devices, which may be private, public, or shared between specific entities. Deciding on which data is stored where is of key importance to the SOFIE project, as it will determine the level and tradeoffs between privacy, accountability, and performance of the proposed framework.

Focusing on the blockchain-related aspects of our pilots, we will have the opportunity to experiment with different ledgers, such as (public and/or private) Ethereum, Hyperledger Fabric, KSI, etc. This gives us a multitude of diverse configurations, as different blockchains have different transaction latency, transaction size, transaction cost, block generation time, and more. Notably, our emulations will allow us to experiment with and assess the pros and cons of different interledger setups, which is a crucial element in any framework employing multiple interacting ledgers.

In the following sections we lay out a generic roadmap for the implementation and use of an emulation environment for each of the four SOFIE pilots.

3.1 Food Chain Pilot – From Field to Fork

The food chain pilot emulation will provide an environment for experimenting with the use case's alternative solutions and parameters. This pilot involves five main stages: (i) a farmer producing table grapes, (ii) a transportation company, (iii) a storage and distribution center, (iv) a second transportation company, and (v) a supermarket. Grapes are packaged in *smart boxes* at the very first stage, i.e., directly by the farmer, and they remain in these smart boxes all the way till they are displayed on supermarket shelves and purchased by customers. That is, smart boxes are the end-to-end transportation unit across the entire food chain pilot.

Each smart box has an *RFID tag* with a unique *box ID*. Every person (or entity, location…) involved in the food chain is equipped with an *RFID reader* and his/her unique *person ID* (or *entity ID* if the system is fully automated). When a person starts handling a box, he/she scans its tag, and a record is created associating a *person ID* with a *box ID* and a timestamp.

In addition to data collected through RFID readers, each stage associates additional metadata with each smart box, collected through sensors on their premises. For example, on the farmer's side, *SynField* sensors (flagship product of SOFIE partner *Synelixis*) contribute relevant historical precipitation, humidity, and temperature data. Refrigerator trucks are equipped with digital thermometers recording the history of temperatures during transportation. Temperature, humidity, and location information are gathered at the storage and distribution centre, while similar measurements are collected at the supermarket site as well.

The ultimate goal in the food chain pilot is to provide provenance information and accountability, while respecting different sites' privacy policies. Indeed, some of the data collected at a site may be made public, while some may be classified as proprietary to the site.

To fulfil this goal, there is a need to generate some sort of digital "receipt" at each handover of smart boxes. More specifically, when a site A hands a smart box over to a site B, the following could happen:

- A's public data related to the box should be signed by A and handed to B, and B should sign that data and return the signature to A.
- A's private data related to the box should be stored by A, but a *hash* of that data should be signed by A and provided to B, and B should sign that hash and return the signature to A. Although this data remains private at A's site, in case of dispute signed hashes can be used to prove the authenticity of complete handover information.

Sites A and B may opt to exchange that data and signatures in a peer-to-peer fashion, and to store them in private databases, each being responsible for their copy of the data. Alternatively, they may opt to utilize a common ledger. In the general case, the ledger storing handover data between two stages need not be the same as the ledger employed by two other stages, neither do the two ledgers be based on the same ledger technology. For instance, the farmer and the first transportation company could be recording handovers on a private Ethereum deployment, while the first transportation company and the storage and distribution centre may be using a Hyperledger Fabric installation. In the extreme case, four separate ledgers can be used to store handover data across all five stages.

We expect stages to opt for deploying private ledgers, to avoid the transaction costs associated with public ledgers, such as Ethereum. However, private ledgers do not provide as strong immutability guarantees as established public ledgers, given that the latter typically involve a substantially larger user base. To address this issue, it may be selected to frequently store anchors of private-ledger block hashes on some highly trusted public ledger, such as KSI or public Ethereum, to provide this extra immutability guarantee. To further lower costs, an extra intermediate ledger *L* may be introduced between small private ledgers and the highly-trusted, expensive public ledger. In that scenario, all private ledgers would frequently store their current block's hash into the intermediate ledger *L*, while only *L*'s block hashes being submitted to the expensive highly-trusted ledger.

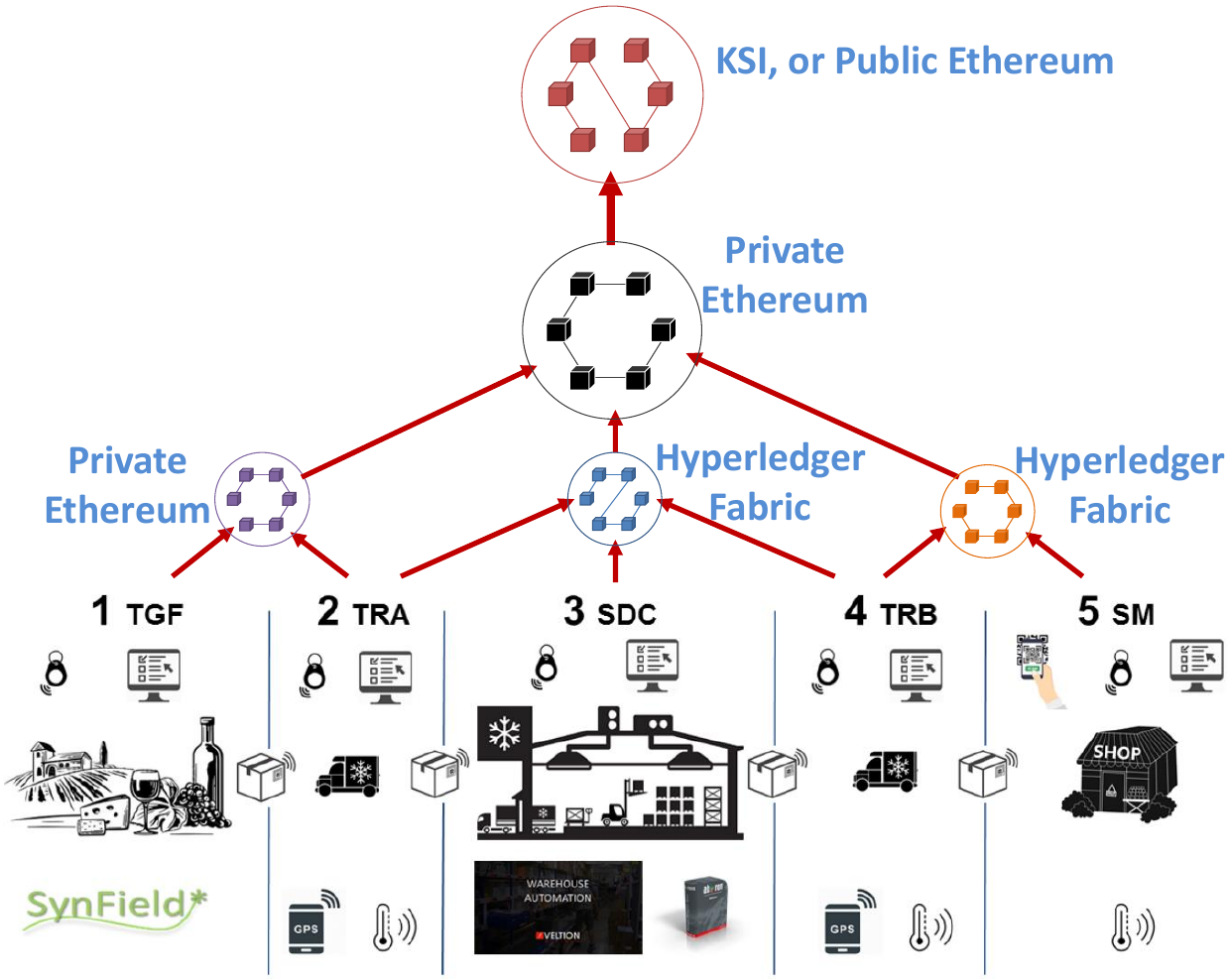

Figure 3. Emulation environment of SOFIE Food Chain Pilot

Figure 3 illustrates a sample configuration, where smart box handover data are recorded in private instances of Ethereum or Hyperledger Fabric, which periodically store their block hashes into an intermediate level private Ethereum instance. The latter, in turn, periodically stores its block hashes into a top-level ledger, which can be either KSI or the public Ethereum blockchain, in order to provide higher immutability guarantees.

The food-chain pilot emulation will implement and test the efficiency and cost of the various types of configurations described above. In terms of implementation, we will deploy our own instances of Ethereum and Hyperledger Fabric on VMs (e.g., use the SOFIE testbeds or interconnected testbed). We will use Python with the *web3.py* library for accessing Ethereum, and the Hyperledger bindings for accessing Hyperledger Fabric[8]. Each actor (i.e., each distinct farmer, transportation truck, storage employee, and supermarket staff member will be modelled as a separate process written in Python, interacting with each other and participating in the corresponding ledgers. The use of interledger transactions will be tested for interactions across different ledgers.

A number of parameters will be explored. Most notably, the choice of specific blockchain technologies will be assessed, as well as the potential issues regarding their interoperation. Performance parameters to test will include the frequency and type of data being fed into the system, the frequency of transaction submissions, the frequency of block hash submissions to a higher-trust ledger in the hierarchy, and the volume of data stored on private databases. The

[8] https://github.com/hyperledger/fabric-sdk-py

latency and the cost of transactions will be measured. In addition, the steps that need to be taken to resolve a dispute will be assessed.

3.2 Energy Pilot – Optimized Demand Response and Electricity Marketplace

The ultimate goal of the electricity marketplace pilot is to use micro-contracts and micro-payments to reduce or even eliminate reverse flows in the electricity grid. Due to increased generation of electricity by distributed renewable sources, such as photovoltaic (PV) cells or wind, electricity is being generated at the "consumption" end in increasing amounts. The goal of the local electricity marketplace is to use pricing and other incentives to increase demand at those places where more production is anticipated, thereby eliminating reverse flows created by the "excess" electricity generated at PV cells.

In the SOFIE pilot, this "excess" electricity is marketed to the owners or managers of electrical vehicles, incentivising them to charge their cars at places and sites where such cheap or maybe even negatively priced electricity will be available.

The goal of the emulated version described here is twofold:

1. Test the technical feasibility of the planned approach.
2. Explore user reaction to different user experiences w.r.t. the transactions.

From the technical feasibility point of view, the emulation environment will set up an early version of the "Flexibility marketplace", as described in SOFIE deliverable D5.1 [Oik2018], in an emulated environment. The necessary datasets for the emulation environment include: typically available excessive energy in kWh and time period, the GPS locations of the charging stations associated with each period, and preferably also the general availability of the vehicles. Having additional representative datasets that would describe the pilot in a much larger setting, e.g. in a setting emulating a large city, a county, or even country, would make the results more interesting, but at this writing it is unclear if such data can be generated.

The datasets defined in SOFIE Data Management Plan [Laa2018] will be used or not used, in the following manner:

1. Topology and asset description (JSON) is currently not planned to be used.
2. Measurement data from EVs (format open) is currently not planned to be used.
3. Log and access data (JSON) is currently not planned to be used.
4. Prediction, forecasting and planning data (JSON) will be used to simulate the planning of demand response campaigns, using the kWh and time period information.
5. Demand response data (JSON) will be used to simulate the actual charging of EVs.
6. Whether Energy or Power forecast data for PV generation will be used is currently open.
7. Positioning and location data of EVs (JSON) will be used to generate anonymised, simulated availability of the vehicles, possibly together with the Energy or Power forecast data.

In addition to these, it is expected that the simulations will need the GPS locations of the charging stations.

The emulated environment will consists of the following components:

1. Underlying DLTs to:
    a. Store the forecasts and actual charging data.
    b. Store offers, bids, and deals (SOFIE marketplace)
2. An early version of an interledger component, as developed by Aalto University.

3. An updated, energy-specific version of the decentralized marketplace, most probably based on the source code released by SOFIE project in 2018[9].

From the user experience point of view, the goal is to emulate and study the following functionality. This corresponds to the Italian Energy Pilot scenario outline in SOFIE deliverable D5.1. The exact relationship to the scenarios needs to be worked out during the actual task.

1. Provide offers of cheap or negatively priced energy to be available. This may happen automatically, based on rules and incoming forecasts. This corresponds to Flexibility request use case of SOFIE deliverable D5.1.
2. Take bids from drivers to promise to charge at the designated time and place. Pull and Push offer use cases from D5.1.
    a. Test with alternative fleet manager experiences, e.g. auction, immediate pricing.
    b. Test with alternative incentive structures, e.g. different offer structures.
3. Collect the actual charging data in a secure manner (Fleet monitoring use case).
4. In real life, this data is expected to be extracted from the charge MQTT data stream, illustrated in Figure 13 of D5.1. For the emulation, this data will be artificially generated, either in a semi-random manner or based on the data about the general availability of the vehicles, if feasible.
5. Match accepted bids with actual charging behaviour. This is not defined as a use case in D5.1, but is still needed for the full functionality.
    a. Pay benefits to bids that charged as agreed.
    b. Charge “fines” for bids that failed to fulfil promises.

Later on, the emulation environment is planned to be reused to generalise the approach to also other major electricity consumer classes in addition EV users and fleet managers. It may also be reused for the upcoming EU H2020 PHOENIX project.

3.3 Energy Pilot – Smart Meters

The goal of the smart meters data access pilot is to enable controlled, secured, and privacy preserving smart meter data sharing among smart meter operators, users and third party service providers. Sharing smart meter data entails privacy risks. Furthermore, with the activation of the EU General Data Protection Regulation (GDPR), this task is impacted by strict legal restrictions (currently resulting in halting this activity in Estonia, i.e., the location of this pilot). This emulation based study related to this pilot will pursue the demonstration of smart contract-based access control mechanisms that will provide transparency, will support user approval, as well as consensus withdrawal, and will enable user rights to data access. At the same time, these mechanisms will also be used for dispute resolution among the involved parties, as well as for privacy preserving statistics using techniques such as differential privacy.

The pilot will leverage the IoT resource access business platform described in D2.3 [Paa2018] and it will use existing authorization standards, such as OAuth2, combined with blockchain-based smart contracts. Furthermore, it will consider the limitations, in terms of computational power, trust, access to code, of the smart meter devices and will use auxiliary services and mechanisms, sidechain constructions, and micropayment techniques (such as those described in [Fot2018][Fot2019]).

The goals of the emulated version of the pilot are:

1. Test the technical feasibility of the planned approach.
2. Measure the impact of various privacy-performance trade-offs.
3. Estimate the performance and cost of large-scale deployments as a function of the costs introduced by the underlay blockchain technology.

[9] https://github.com/SOFIE-project/offer-marketplace

4. Evaluate possible trade-off (in terms of performance, cost, and security) of different distributed ledgers interconnected with interledger mechanisms and micropayment techniques.
5. Evaluate the security properties of the pilot by performing security attacks.
6. Propose extensions/modifications to the existing devices and platforms.

The emulated environment will consist of the following components:

1. "Underlying," company-specific DLTs as follows:
    a. KSI blockchain used for timestamping as well as for recording data hashes.
    b. Ethereum blockchain used for implementing the corresponding smart contracts.
    c. Permissioned or private distributed ledgers used for storing data hashes.
2. Emulated smart meters. The emulated devices will be capable of:
    a. Generating the same type of data as the real devices.
    b. Encoding data using protocols supported by the real devices.
    c. Providing the same access mechanisms (e.g., direct access, access over the Internet using HTTPS), as well as cryptographic primitives (i.e., public key encryption and emulated physical unclonable functions) as the real devices.
3. Emulated "Data hubs". These are the entities responsible for collecting data from the smart meters.
4. Applications that will emulate the involved actors, i.e., (and as they are specified in SOFIE deliverable D5.1):
    a. Smart meter system operators.
    b. Smart meter owners.
    c. Energy service providers.
5. "Authorization servers" supporting OAuth2 (and related protocols). The emulated environment will support multiple granularities of access control (e.g., access to the data of a specific user, access to the data of the users of a specific operator), hence it will support various deployment strategies for the authorization servers (e.g., servers owned by the users, by an operator, etc).
6. Differential privacy mechanisms.

3.4 Mixed Reality Gaming Pilot

The focus of the emulation of this pilot will be to investigate what is a suitable DLT setup for this pilot, experimenting with all the possible configurations at the same time. Eventually, we will provide an environment for experimenting with the scenarios presented on this pilot. One of the main challenges, in this pilot, is to provide scalable DLTs to cost effectively support millions of active users per day with thousands of transactions per second. This pilot consists of the following:

- Core game
- Mini game(s)
- Mini IoT game(s)

The core game that will be implemented in this pilot will allow users to collect, buy, and trade assets from other users leveraging DLTs to provide player ownership of an asset, a marketplace, transparency, and consistency of asset attributes and transactions. In the emulation of the core game, in order to meet the aforementioned requirements we can use several solutions and configurations. First of all, to emulate this pilot we will use an implementation of Ethereum. We will experiment with both public and private instances of the Ethereum blockchain, to see the differences between these two implementations, assessing their pros and cons. Furthermore, we will test how the system will respond to different consensus algorithms (PoS, PoA, etc.). In order to support payments both in cryptocurrencies

and in fiat currencies, we plan to investigate a version of the Interledger Protocol[10]. Figure 4 illustrates a sample configuration of core game.

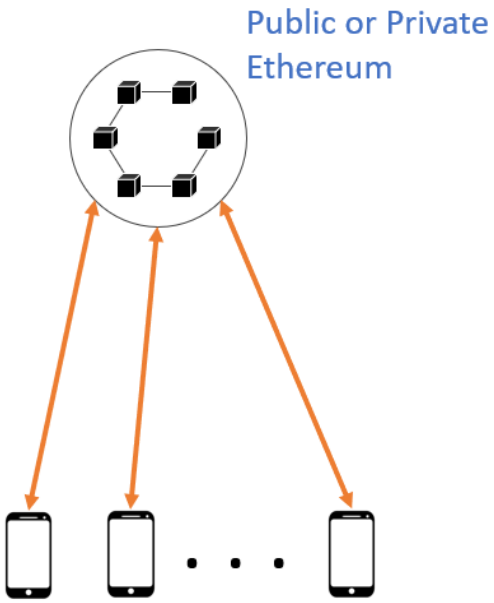

Figure 4. Basic overview of SOFIE Mixed Reality Gaming Pilot emulation environment

Players could be emulated as JavaScript applications with the use of the *web3.js* library. The creation of new assets can be done via smart contracts. Players or even applications will be able to purchase and trade objects with the use of smart contracts. Therefore, users will have to interact with the DLT to perform such actions. The assets and their ownership will be recorded on the DLT.

Mini games are built on top of the assets created on the core game and they constitute an extension of the core game. Developers will be able to use the assets from the core game to build new games, based on the core game. For example, if the assets that were created on the core game are weapons, then a mini game could be a duel game between the players and theirs weapons respectively. The winner will earn a reward (e.g., a shield), which will be used only for the mini game. The emulation of a mini game will be on top of core game content that exists on the blockchain. In this scenario, there are several possible configurations. Firstly, we will use a single blockchain, used both for the core game and for the mini game. Another setup will be to use different blockchains for the core game and each of the mini games. This configuration is useful for scenarios, where mini games provide additional assets. In this setup, we will also utilise the Interledger protocol. Figure 5 provides an overview of one of the many possible configurations for the emulation of the mini game.

[10] https://interledger.org/

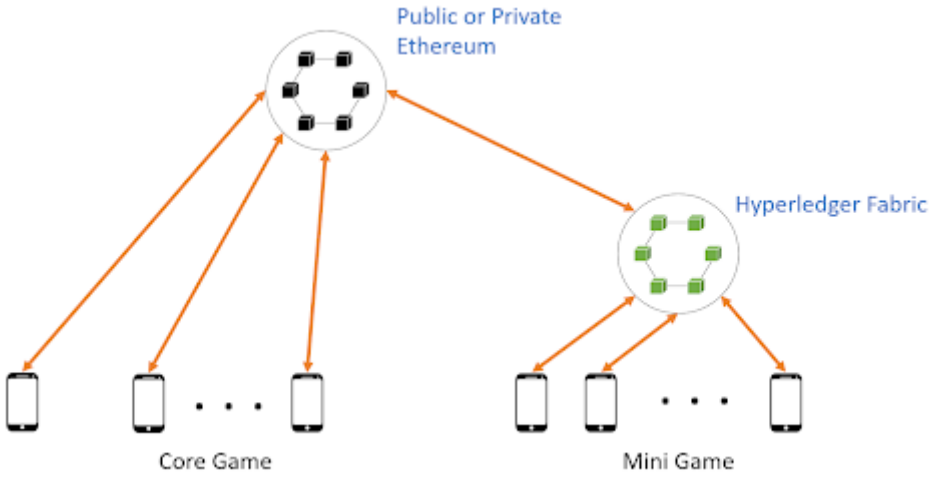

Figure 5. SOFIE Mixed Reality Gaming Pilot emulation environment with Mini game

The last part of this pilot are mini IoT games. These games would leverage IoT devices to enhance the gameplay. For example, players will interact with an IoT device (e.g., a beacon) to gain an asset. These games are also built on top of the assets created on the core game. For the emulation of these games, we will use virtual Things and we will record the actions to the blockchain (e.g., that a user has reached a target geolocation). The configurations on these IoT games are the same as the configurations on the mini game.

We can also investigate through emulation or simulation more general scenarios and use cases involving real mixed-reality gaming, i.e., both cyber and real-world assets and both sensors and actuators, for example in a smart mall environment.

4. Conclusions

This deliverable describes the SOFIE testbed and emulation environment. The SOFIE testbed spans multiple project partners and allows testing various distributed ledger technologies and their interaction with IoT devices on a wider scale. SOFIE's emulation environment emulates certain aspects of SOFIE pilots and related more general use-cases, which allows realistic testing of various solutions without deploying them yet in pilot environments. The SOFIE testbed and emulation environment will be used in the evaluation work during the rest of the project, and the further progress of SOFIE's evaluation work package will be reported in deliverable "D4.3 - First Architecture and System Evaluation Report", due in June 2019.

References

[Fot2018] N. Fotiou, V. A. Siris, and G.C. Polyzos, "Interacting with the Internet of Things Using Smart Contracts and Blockchain Technologies", Proc. of Security, Privacy, and Anonymity in Computation, Communication, and Storage 2018 (SpaCCS 2018), Melbourne, Australia, 2018.

[Fot2019] N. Fotiou, V. A. Siris, S. Voulgaris, G.C. Polyzos, and D. Lagutin, “Bridging the cyber and physical worlds using blockchains and smart contracts”, Proc. of the NDSS Workshop on Decentralized IoT Systems and Security (DISS), 2019.

[Laa2018] P. Laari et al., “SOFIE Deliverable D6.5 - Data Management Plan”, June 2018. Available at: https://media.voog.com/0000/0042/0957/files/SOFIE_D6.5-Data_Management_Plan_v1.00.pdf.

[Oik2018] I. Oikonomidis et al., “SOFIE Deliverable D5.1 - Baseline System and Measurements”, June 2018. Available at: https://media.voog.com/0000/0042/0957/files/SOFIE_D5.1-Baseline_System_and_Measurements.pdf.

[Paa2018] S. Paavolainen et al., “SOFIE Deliverable D2.3 - Federation Framework, 1st version”, October 2018. Available at: https://media.voog.com/0000/0042/0957/files/SOFIE_D2.3-Federation_Framework_1st_version_v1.00.pdf.

[Sir2018] V. A. Siris et al., “SOFIE Deliverable D4.1 - Validation and Evaluation Plan”, October 2018. Available at: https://media.voog.com/0000/0042/0957/files/SOFIE_D4.1-Validation_and_Evaluation_Plan-v1.00.pdf.