Multi-layered Security Technologies

for hyper-connected smart cities

D4.6 P2P level security and M-Sec blockchains

March 2021

Grant Agreement No. 814917

Multi-layered Security technologies to ensure hyper-connected smart cities with Blockchain, BigData, Cloud and IoT

Project acronym	M-Sec
Deliverable	D4.6 P2P level security and M-Sec blockchains
Work Package	WP4
Submission date	31 March 2021
Deliverable lead	Georgios Palaiokrassas (ICCS)
Authors	Georgios Palaiokrassas (ICCS), Xavier Cases Camats (WLI), Orfefs Voutyras (ICCS)
Internal reviewer	Xavier Cases Camats (WLI) / Aamir Bokhari (YNU)
Dissemination Level	Public
Type of deliverable	DEM

The M-Sec project is jointly funded by the European Union's Horizon 2020 research and innovation programme (contract No 814917) and by the Commissioned Research of National Institute of Information and Communications Technology (NICT), JAPAN (contract No. 19501).

Version history

#	Date	Authors (Organization)	Changes
v0.1	01 February 2021	Georgios Palaiokrassas (ICCS)	Full ToC and assignments
v0.2	22 February 2021	Georgios Palaiokrassas (ICCS)	Content for FGs Components
v0.3	01 March 2021	Xavier Cases Camats (WLI)	CCD Section
v0.4	17 March 2021	Georgios Palaiokrassas (ICCS)	FGs & Integrations of FGs updates
v0.5	18 March 2021	Xavier Cases Camats (WLI)	new CCD figures
v0.6	18 March 2021	Georgios Palaiokrassas (ICCS)	Components API & Figures
v0.7	22 March 2021	Xavier Cases Camats (WLI)	Interaction with other FGs
v0.8	23 March 2021	Georgios Palaiokrassas (ICCS)	FGs' APIs and Common API
v0.9	24 March 2021	Georgios Palaiokrassas (ICCS)	Rephrasing and updating
V0.10	25 March 2021	Xavier Cases Camats (WLI)	Rephrasing and corrections.
v0.11	25 March 2021	Georgios Palaiokrassas (ICCS)	Integrating different parts
v0.12	27 March 2021	Georgios Palaiokrassas (ICCS)	Formatting, corrections
v0.13	28 March 2021	Orfefs Voutyras (ICCS)	Version for internal review
V0.14	31 March 2021	Xavier Cases Camats (WLI)	Internal Review
v0.15	31 March 2021	Aamir Bokhari (YNU)	Internal Review
v1.0	31 March 2021	Georgios Palaiokrassas (ICCS)	Version ready for submission

Table of Contents

List of Tables

List of Figures

Glossary

Acronym	Description
API	Application Programming Interface
IPFS	InterPlanetary File System
CCDB	Crypto Companion Database
KYC	Know Your Customer
Dx.y	Deliverable y of WP x
FG	Functional Group
Tx.y	Task y of WP x
P2P	Peer-to-peer
UC	Use Case
WP	Work Package
REST	Representational state transfer
T&R	Trust and Reputation
IoT	Internet of Things

Executive Summary

The work described in this deliverable (D4.6) was carried out in the framework of WP4 – “Multi-layered Security Technologies”, and more specifically, in the framework of T4.3 – “P2P Level Security and Blockchains”. This report presents the updated and final version of the document (the first version being D4.5), providing the technical details of the Functional Group and Functional Components related to the Task.

All technical partners involved in this task collaborated and developed the appropriate tools to meet the objectives set out in the project, especially with regard to novel security aspects in IoT contexts. Every partner focuses on the individual modules that they are responsible for during the implementation phase of WP4 and supports the integration activities of WP2, while following the common Architecture framework set by WP3 in D3.4.

All of the updated versions of the WP4 technical deliverables (D4.2, D4.4, D4.6, D4.8, D4.10) follow the same approach and have the same structure. Section 1 provides an introduction to the scope of this document and its relation with other WPs and Tasks. Section2, which aggregates all the main outcomes of the Task, presents extensively the FG and the Functional Components covered by the Task, by providing an extensive description of the corresponding functionalities, and details related to the API of the FG and its interactions with other FGs of the M-Sec solution. Finally, Section 3 concludes the document.

Regarding the differences between ‘D4.5 M-Sec P2P Level Security and Blockchains – first version’ and ‘D4.6 M-Sec P2P Level Security and Blockchains – final version’:

- **Section 1** has remained more or less the same, but includes an extra subsection identifying the M-Sec Risks linked to this specific Task.
- **Section 2** as a whole provides a more integrated view of the Components, as it focuses on their presentation from an FG perspective. As such, the CCDB component is also described in this deliverable (moved from T4.5).
- **Section 2.2** corresponds to Sections 2 and 3 of the previous version of the document (D4.5). The Package Information, Installation Instructions, and the Licensing Information are omitted in the new version, but they are replaced by **Section 2.4**.
- **Section 4** corresponds to Section 4 of the previous version of the document.

All in all, the deliverable is considered to have provided all the information required to expose the M-Sec technical solutions related to T4.3 as well as the results of the integration and demonstration related activities.

1. Introduction

1.1 Scope of the document

The current document, deliverable 'D4.6 P2P level security and M-Sec blockchains', provides the second and final version of M-Sec developments related to blockchain technology and P2P level security and also the second version of the Crypto Companion Database. In detail, it presents four different demonstrators as well as the corresponding services and gives installation details. It also provides details about developments since the previous iteration of this deliverable, namely D4.5 on M18, and also part of D4.7. The main focus of the presented demonstrators and tools is to implement the M-Sec blockchain framework together with the CCDB to facilitate the convergence of IoT security with blockchains in order to support an innovative smart city platform.

1.2 Relation to other work packages and tasks

The following figure summarises the relations of this deliverable (and the corresponding task) to other tasks and WPs.

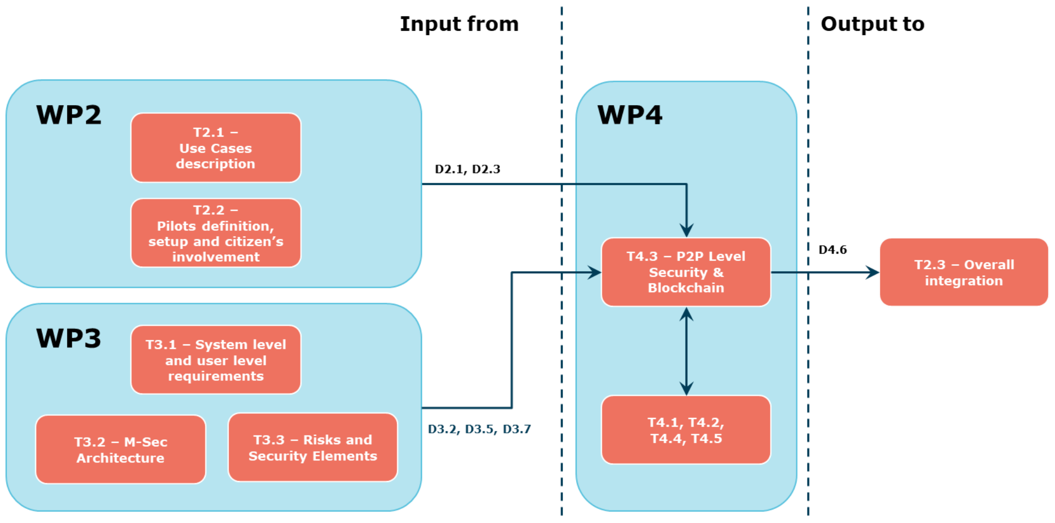

Input from
Output to
WP2
T2.1 – Use Cases description
T2.2 – Pilots definition, setup and citizen's involvement
WP3
T3.1 – System level and user level requirements
T3.2 – M-Sec Architecture
T3.3 – Risks and Security Elements
D2.1, D2.3
D3.2, D3.5, D3.7
WP4
T4.3 – P2P Level Security & Blockchain
T4.1, T4.2, T4.4, T4.5
D4.6
T2.3 – Overall integration

Figure 1. T4.3 and D4.6 relation with other WPs and Tasks

The Task is directly related to WP3. T4.3 receives as input system and user requirements from T3.1 and Risks- and Threats-related information from T3.2. Moreover, it follows the common Architectural framework that has been identified in T3.2 for the coordination of all the technical activities. Similarly, the Task receives input from WP2 related to the coverage of the needs of the UCs and the pilots.

Furthermore, T4.3 has dependencies with the rest of WP4 "Multi-layered Security technologies" Tasks and more specifically with T4.1 for IoT security and related services based on blockchain technology, T4.2 "Cloud and data level security", T4.4 "Application level security" and T4.5 "Overall end-to-end security" focusing on the integration with respective implementations for encrypted data storage.

Finally, the results of this report are directly provided as input to T2.3 which is focusing on the overall integration activities. Together with the other final deliverables of WP4, D4.2 and D4.6 provide all the information and functionalities required for an integrated security solution.

1.3 Methodology followed

In order to enable the M-Sec paradigm, we researched different technologies and approaches of blockchain technology. We initially started the analysis with a detailed description of different blockchain frameworks, such as Ethereum, Hyperledger and Quorum, and tools we experimented with. After analysing thoroughly all the different available technologies, we selected Ethereum-based blockchain framework for our implementation.. More details are presented in Section 2. The same section provide details about other assets of this FG, including Middleware Services, Trust and Reputation Management and Companion Database. In the last section, the interactions among the components and other Functional Groups are given, along with the installation and demonstration instructions.

Data security and protection is one of the main goals of any application, and one of the main goals of the M-Sec project. In order to be compliant with the GDPR, a parallel system to Blockchain has been developed: a "Companion Database", that allows saving data linked to Blockchain with a hash of the same. This solution has been adopted, but with some additions. The data stored in the database will be encrypted per user, meaning that users will not share keys between them. The problem of a database usually is that it is centralized, so with this development the data can be distributed.

Therefore, the main motivations of this development were:

- A user can save data.
- A user can delete data.
- A user can read data.
- A user can modify data.
- A user can delete all data owned.
- A user can read all data owned.
- The data will be encrypted when saved.
- The data will be decrypted when read. (Conditional with the next statement)
- The data will only be accessible by the owner or an authorized user.
- The data will be distributed.

1.4 Relation to M-Sec Risks

The list of the main potential risks and threats that may affect M-Sec's Secured and Trusted Storage FG is provided in the following table, as extracted from Task 3.3, D3.5.

Table 1. M-Sec T4.3 related Risks and Threats

Threat #	Description	STRIDE Threat Class	M-Sec Asset	Source	Probabi-lity	Critica-lity	Ra-ting	Comments/ Mitigation
Thr.Com.9	Disclosure of encryption parameters for the communication channels	I	IoT Gateway, Caburn	Use Case 2, 3	3	5	15	Security Manager
Thr.Com.21	51% attack over blockchain	R, T	Quorum Blockchain	All	1	5	5	Avoid PoW blockchain and use instead permissioned - delegated consensus mechanisms
Thr.Com.22	Private key security - Public key encryption scheme	R, E	Quorum Blockchain	All	1	3	3	Permissioned blockchains can mitigate the specific threat more easily
Thr.CD.1	Impersonation: A third party uses a false ID to gain access to the cloud	S	SoxFire, Companion DB	Use Case 2, 3	3	3	9	Strong Authentication
Thr.CD.2	An attacker may install a malware to access the data and whole cloud system	I, T, D	SoxFire, Companion DB	Use Case 2, 3	1	3	3	Protected in Keio's network
Thr.CD.3	Accidental or intentional physical damage to any cloud part may cause cloud service failure	D	SoxFire	Use Case 3	3	1	3	Backup server
Thr.CD.4	Disruption of a global service (e.g. attack on power management)	D	SoxFire	Use Case 3	3	1	3	Backup power
Thr.CD.5	Data (raw & processed, personal data) stored in the cloud can be read by an intruder	I	SoxFire, Companion DB	Use Case 2, 3	3	5	15	Encryption
Thr.CD.6	An unauthorized party gets access to device configuration information	I	SoxFire	Use Case 3	1	3	3	Protected in Keio's network
Thr.CD.7	Attacker denies legitimate users access to infrastructure services	D	N/A	-	-	-	-	No Infra services running

Thr.CD.8	Attacker can poison cloud database and/or alters outgoing information	T	SoxFire, Companion DB	Use Case 2, 3	3	5	15	Encryption
Thr.App.16	Vulnerabilities-flaws in smart contracts	T,R,I,D	Blockchain app / Smart contract	All Use Cases	3	5	15	Flaws in smart contracts can cause unforeseen security breaches. Thorough lab testing before going into production. Continuous code review.
Thr.App.17	Under-optimized smart contracts	T,R,I,D	Blockchain app / Smart contract	All Use Cases	3	3	9	Dead code, loop fusion, repeated computation can cause denial of service on the long run. Thorough lab testing before going into production. Continuous code review.
Thr.App.18	Transaction privacy leakage	T,R,I,D	Blockchain app / Smart contract	All Use Cases	3	3	9	Once identity is revealed the whole history of transactions is exposed. Thorough lab testing before going into production. Continuous code review.
Thr.Com.9	Disclosure of encryption parameters for the communication channels	I	IoT Gateway, Caburn	Use Case 2, 3	3	5	15	Security Manager
Thr.Com.21	51% attack over blockchain	R, T	Quorum Blockchain	All	1	5	5	Avoid PoW blockchain and use instead permissioned - delegated consensus mechanisms
Thr.Com.22	Private key security - Public key encryption scheme	R, E	Quorum Blockchain	All	1	3	3	Permissioned blockchains can mitigate the specific threat more easily

2. Secured and Trusted Storage FG

2.1 General Description of the FG

The following figure presents the core components of the Secured & Trusted Storage FG. As the name implies, this FG is focused on providing tools and mechanisms that enhance the security of the M-Sec overall solution at the Storage level. To achieve that, M-Sec exploits both a blockchain-focused approach and an encrypted database one. By storing and encrypting the main core of the data off-chain on the Cloud and storing the corresponding metadata and interactions-related data on-chain, M-Sec couples the benefits of both the P2P and Cloud solution.

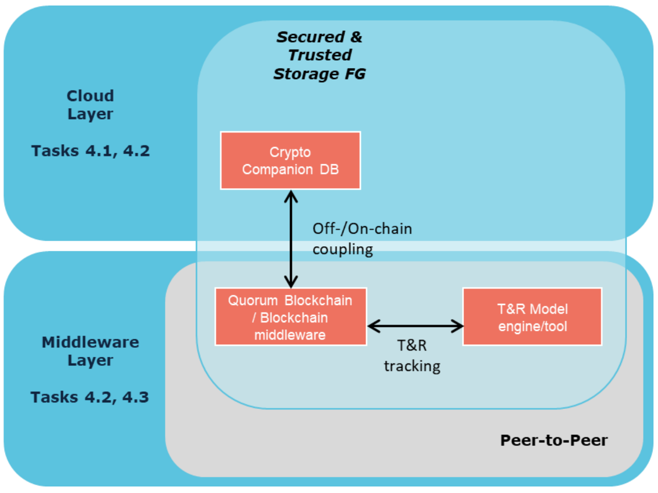


Figure 2. Secured & Trusted Storage FG

The core components that relate to this FG are:

- Blockchain & Blockchain Middleware Services
- T&R Management
- Crypto Companion Database

In the following section, these components are described in detail. Section 2.3 presents the interactions of these components with components of other M-Sec FGs. Finally, the Annex presents the position of the Secured & Trusted Storage FG within the whole M-Sec Architecture.

2.2 Components of the FG

The Secured and Trusted Storage FG consists of three main components the Crypto Companion Database, the Quorum Blockchain/Blockchain middleware and the Trust & Reputation Management. One of the most important features of this FG is the decoupling and handling of data based on their types from components of the FGs, namely the CCDB and the Quorum Blockchain/Blockchain middleware. The main distinction is whether personal data, metadata or data for interactions and transactions are handled. In the following subsections the different components are described in detail as well as the interactions among them and the interactions with other FGs.

First description of the components was provided in the previous deliverable D4.5. In the current deliverable, more details, additions and further improvements are also provided, as well as details about the interactions among the components and among this FGs and the other FGs defined.

Blockchain Framework

General Description of the Component

The main focus of this Component is to implement the M-Sec blockchain framework, and to facilitate the convergence of IoT security with blockchains in order to support an innovative smart city platform. We used Ethereum-based blockchains as the basic foundation of M-Sec blockchain as it enables not only the exchange of value (M-Sec tokens) but also the enforcement of smart contracts, which provides an additional feature for the implementation and validation of the selected M-Sec use cases.

A milestone for the course of blockchain technology was the development of Ethereum project[1], offering new solutions by enabling smart contracts' implementation and execution. It is a suite of tools and protocols for the creation and operation of Decentralized Applications (DApps), "*applications that run exactly as programmed without any possibility of downtime, censorship, fraud or third-party interference*".

It also supports a contract-oriented, high-level, Turing-complete programming language[2], allowing anyone to write smart contracts and create DApps. Smart contracts are mainly written in the programming language Solidity[3,4].

We have initially experimented with different Blockchain platforms, before concluding to Ethereum-based blockchains, such as Quorum and examined both public (permissionless) and private (permissioned) alternatives of the M-Sec blockchains. The most prominent among them were "Hyperledger" and "Quorum", which are described in the next Sections. Hyperledger implementation was considered as it can enable, through specific channels, the implementation of flexible blockchains with different permissions and authorization schemes.

The peer group management service is also part of the work covered in this task, as research will be pursued for defining how the blockchain networks are going to be self-organized and structured in the context of service provisioning so that they can form and operate the multi-layered architectures.

[1] J. Ray, "Ethereum Introduction," 11 12 2019. [Online]. Available: https://github.com/ethereum/wiki/wiki/Ethereum-introduction

[2]"White Paper," [Online]. Available: https://github.com/ethereum/wiki/wiki/White-Paper

[3] Ethereum, "What is Ethereum?," [Online]. Available: http://www.ethdocs.org/en/latest/introduction/what-isethereum.html

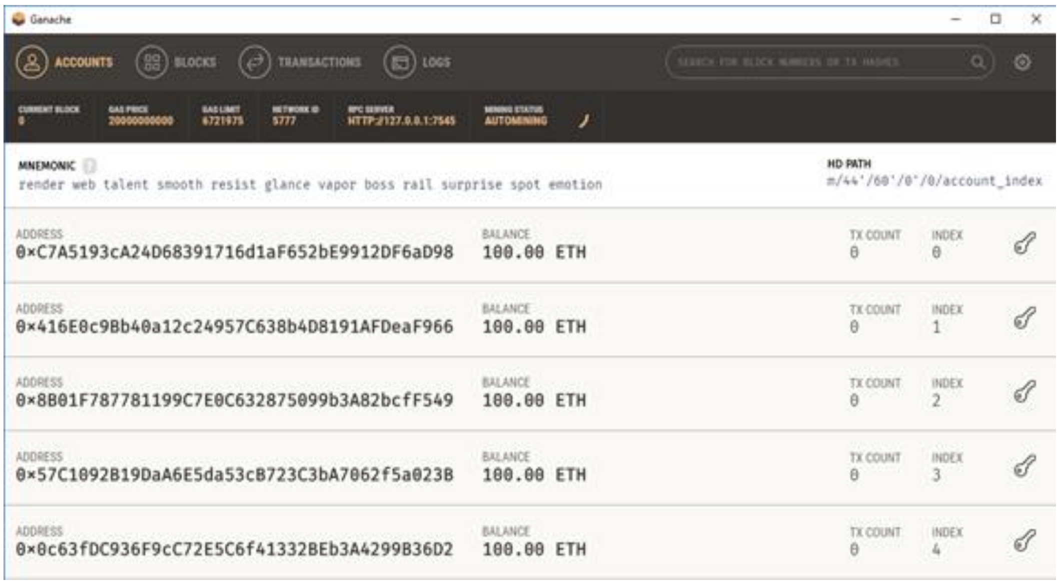
[4] Ethereum, "Solidity," Ethereum, [Online]. Available: http://solidity.readthedocs.io

Blockchain framework modules

In this Section we present the details regarding the blockchain platform, in which we develop the smart contracts that support the different use cases. As mentioned before, the different smart contracts are written in the programming language Solidity[5] .

Private Ethereum Blockchain

During the development process we have used a local private blockchain named Ganache[6], which allowed us extensive testing of the developed smart contracts. It provides a personal Ethereum blockchain which we can use to run tests, execute commands, and inspect the state while controlling how the chain operates. It provides a built-in explorer as shown in the following Figure 3 and allows us to quickly see the current status of all accounts, including their addresses, private keys, transactions and balances.



CURRENT BLOCK 0 | GAS PRICE 20000000000 | GAS LIMIT 6721975 | NETWORK ID 5777 | RPC SERVER HTTP://127.0.0.1:7545 | MINING STATUS AUTOMINING

MNEMONIC
render web talent smooth resist glance vapor boss rail surprise spot emotion

HD PATH
m/44'/60'/0'/0/account_index

ADDRESS	BALANCE	TX COUNT	INDEX
0×C7A5193cA24D68391716d1aF652bE9912DF6aD98	100.00 ETH	0	0
0×416E0c9Bb40a12c24957C638b4D8191AFDeaF966	100.00 ETH	0	1
0×8B01F787781199C7E0C632875099b3A82bcfF549	100.00 ETH	0	2
0×57C1092B19DaA6E5da53cB723C3bA7062f5a023B	100.00 ETH	0	3
0×0c63fDC936F9cC72E5C6f41332BEb3A4299B36D2	100.00 ETH	0	4

Figure 3. Ganache Explorer allows us to examine all blocks, transactions, addresses and their balances

Public Ethereum Blockchain

Additionally, we deployed smart contracts on Public Ethereum Blockchain using browser IDE "Remix"[7]. Remix is an open-source tool that supports smart contracts development on the browser and facilitates the deployment on local or public Ethereum-based blockchain platforms. We used Ropsten public Ethereum blockchain[8]. It is important to extensively test the smart contracts before we deploy them to the Quorum blockchain network (see next section), since the code can't be changed after deployment. To this direction, extensive testing was carried out on blockchains, by using these two testing solutions: Ganache Cli, as well as the Ropsten Test Net.

[5] Ethereum, "Solidity," Ethereum, [Online]. Available: http://solidity.readthedocs.io

[6] https:// www.trufflesuite.com/ganache
[7] https://remix.ethereum.org/
[8] https://ropsten.etherscan.io/

Quorum blockchain framework

Finally, the different smart contracts are written in the programming language Solidity[9] and are deployed on Quorum blockchain framework[10]. Quorum is a permissioned implementation of Ethereum which allows certified members to build and run decentralized applications that run on blockchain technology. It is an open-source platform and supports smart contract privacy. Both private and public smart contracts are validated by every node within the blockchain network. Additionally, Quorum provides privacy and transparency, both at transaction-level and network wide.

In each Quorum node consensus is achieved with the Raft or Istanbul BFT consensus algorithms instead of using Proof-of-Work. The P2P layer has been modified to only allow connections to/from permissioned nodes. In Ethereum the notion of Gas was introduced (the fee or pricing value required to successfully conduct a transaction or execute a contract on Ethereum blockchain platform), while in Quorum the pricing of Gas has been removed, although Gas itself remains.

One of the features of Quorum that are of great value for the component is the network and peer to peer permission management. This feature enables only the validated and authorized users to have access and be a part of the network. Also, Quorum provides enhanced transaction and smart contract privacy features. Permission-based nature of Quorum enables the constitution of private and public transaction getting the best of both worlds, open transactions are analogous to Ethereum but when it comes to the private transaction then it is confidential, and the data is not exposed to the public. Quorum adds privacy functions that allow for private transactions that are only visible to the transacting parties, while the other parties in the network would only see a hash. Finally, Quorum is considered to be very fast and being able to process even thousands of transactions per second, due to its efficient consensus mechanism which belongs to the family of Byzantine Fault Tolerance (BFT) mechanisms[11].

In order to develop and deploy the smart contracts to Quorum blockchain, we have used Truffle suite[12]. It is a development environment and testing framework using the Ethereum Virtual Machine (EVM). Additionally, we use Quorum Maker[13] (see Figure 4 below), which facilitates the deployment of smart contracts, offering visualization features to monitor the Quorum blockchain network and related blocks and transactions. Before we deploy the smart contracts to the blockchain network, we extensively tested them on Ethereum blockchains using two testing solutions: Ganache Cli, as well as the Ropsten Test Net. Additionally, before being deployed on a larger Quorum Network, we used a Quorum test network consisting of seven nodes[14].

[9] Ethereum, "Solidity," Ethereum, [Online]. Available: http://solidity.readthedocs.io

[10] https://docs.goquorum.com/en/latest/

[11] Vukolić M. (2016) The Quest for Scalable Blockchain Fabric: Proof-of-Work vs. BFT Replication. In: Camenisch J., Kesdoğan D. (eds) Open Problems in Network Security. iNetSec 2015. Lecture Notes in Computer Science, vol 9591. Springer, Cham

[12] https://www.trufflesuite.com/
[13] https://github.com/synechron-finlabs/quorum-maker
[14] https://github.com/jpmorganchase/quorum-examples/tree/master/examples/7nodes

Smart contracts are written in Solidity so it is feasible to migrate from Quorum permissioned Blockchain Framework to public Blockchain Frameworks (e.g. Public Ethereum Network), since Solidity is the common programming language to Ethereum-based blockchain frameworks.

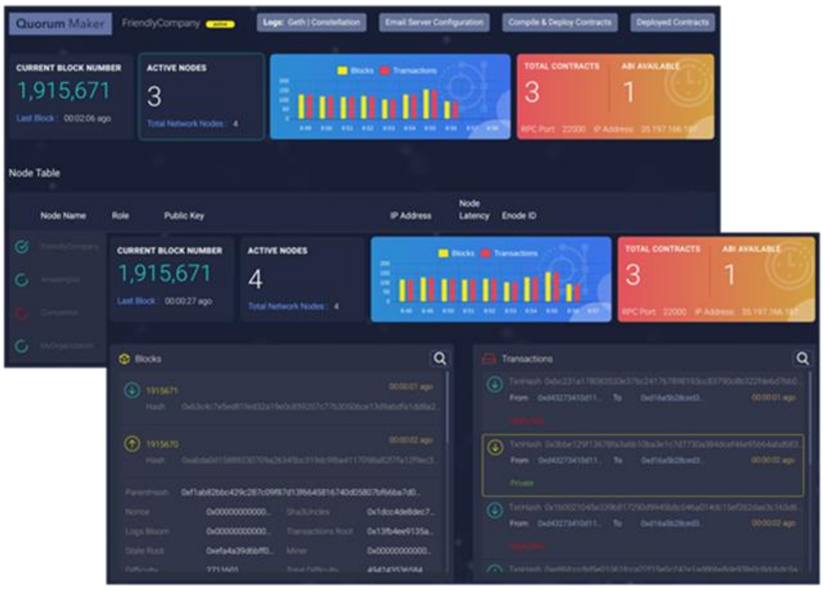
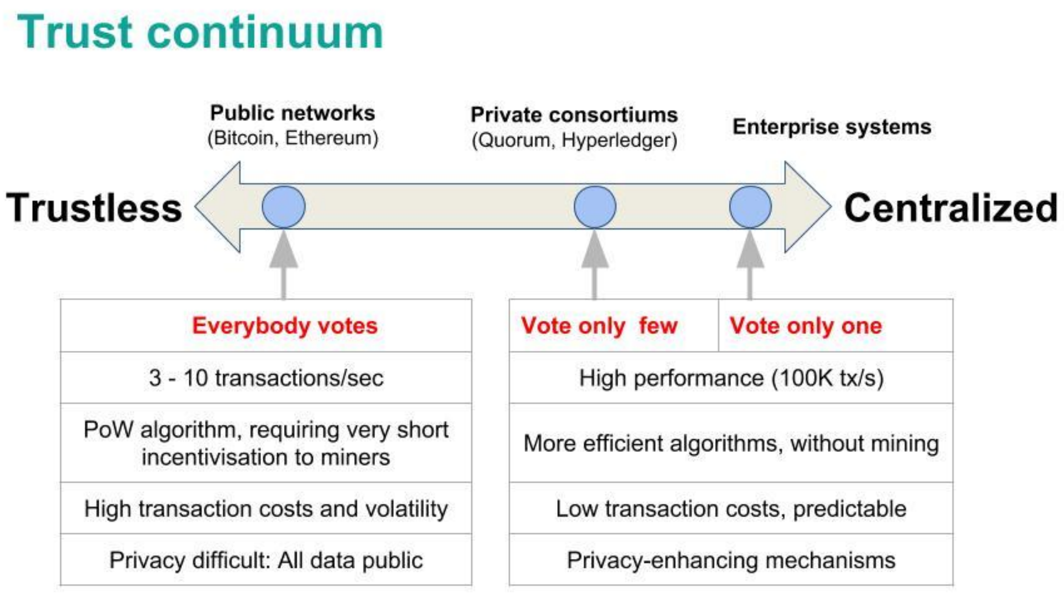


Figure 4. Details about blocks, transactions, addresses and smart contracts

ALASTRIA

Alastria is neither a public-permissionless network nor a private consortium, it is a Public-Permissioned network. It shares some of the properties of both types of networks, and it also has some requirements of its own.

Figure 5 describes the prevailing public-permissionless blockchain networks currently in production like Bitcoin or Ethereum have the very desirable property of being “*Trustless*”. However, mainly due to the characteristics of the consensus algorithms used to achieve that property, they suffer from very well documented scalability problems. There are a lot of efforts being made to solve or alleviate the scalability problem, but as of today, the problem still exists and permissioned networks will always have several orders of magnitude better performance.



Everybody votes	Vote only few / Vote only one
3 - 10 transactions/sec	High performance (100K tx/s)
PoW algorithm, requiring very short incentivisation to miners	More efficient algorithms, without mining
High transaction costs and volatility	Low transaction costs, predictable
Privacy difficult: All data public	Privacy-enhancing mechanisms

Figure 5. The Trust Continuum

The Problems of the Public-Permissionless blockchain networks are:

- Scalability: The networks choose Decentralization and Security over Scalability.
  Taking into account the words of Vitalik Buterin describing the "*Blockchain trilemma*[15]", the trilemma claims that blockchain systems can only at most have two of the following three properties:
  - Decentralization, defined as the system being able to run in a scenario where each participant only has access to *O(c)* resources, where *c* refers to the size of computational resources available to each node (i.e. a regular laptop or small VPS "Virtual Private Server").
  - Scalability, defined as being able to process *O(n)* > *O(c)* transactions, where *n* refers to the size of the ecosystem in some abstract sense.
  - Security (or Safety), defined as being secure against attackers with up to *O(n)* resources
- Transaction Costs: High and Volatile
- Privacy: By default, in public blockchains like Bitcoin or Ethereum, transactions are executed by all nodes in the network, transactions are globally published and state data is not encrypted in most applications, so all participants have access to all data stored in the ledger without any restriction.

[15] https://medium.com/@aakash_13214/the-scalability-trilemma-in-blockchain-75fb57f646df

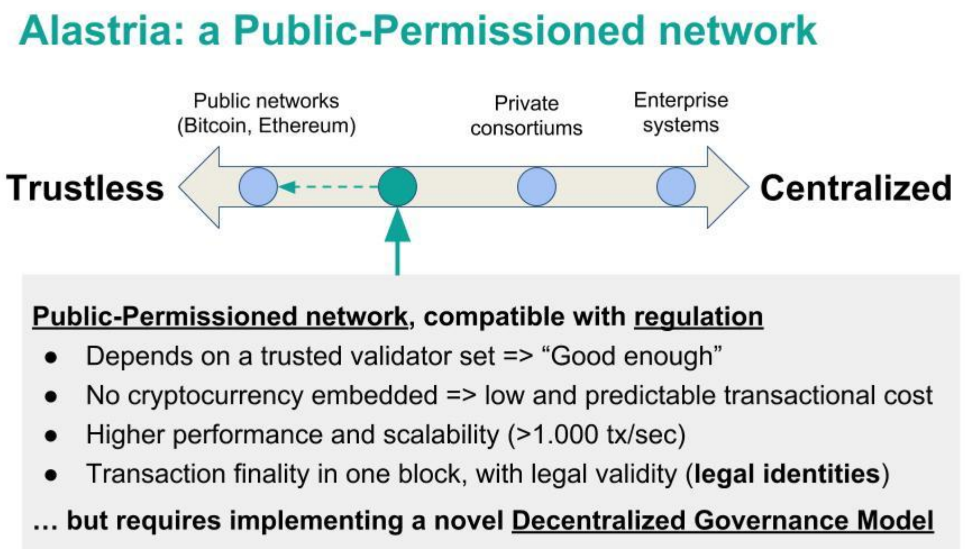

Alastria: a Public-Permissioned network
Public networks (Bitcoin, Ethereum)
Private consortiums
Enterprise systems
Trustless
Centralized
Public-Permissioned network, compatible with regulation
● Depends on a trusted validator set => "Good enough"
● No cryptocurrency embedded => low and predictable transactional cost
● Higher performance and scalability (>1.000 tx/sec)
● Transaction finality in one block, with legal validity (legal identities)
… but requires implementing a novel Decentralized Governance Model

Figure 6. Alastria in the Trust Continuum

In Public-Permissioned networks, the objective is to maximize decentralization and safety, even if this goes to the detriment of scalability. In this context, decentralization typically means the ability to transact anonymously but safely among individuals without the need for any intermediary acting as trusted party. It is often the case that the requirement to eliminate third parties is stronger than the requirement that the system be high-performance so it could be used as a general purpose transaction mechanism.

In Private Consortiums the objectives are generally different, and instead of trying to eliminate third parties at all costs, they try to use blockchain technology to improve efficiency and reduce costs of transaction among the partners composing the consortium. In many private consortiums, they want a shared database and transaction system so they can eliminate frictions and reduce costs of reconciliation.

As Figure 5 and Figure 6 describe, Alastria tries to be as public as possible, but without the disadvantages associated with public-permissionless networks.

As mentioned before, Alastria is not a Private Consortium but a Public-Permissioned network compatible with regulation instead. At a very high-level, the characteristics of Alastria are the following:

- It's permissioned, so every participant node has to be identified before it can participate in the network.
- No cryptocurrency embedded.
- A more efficient consensus algorithm, enabling higher performance and scalability.
- Transaction finality in one block, enabling legal validity of executed transactions.
- Implements legal identities of all participants.

For further information check the GitHub page of Alastria[16].

[16] https://github.com/alastria/alastria-platform/blob/master/en/Alastria-Core-Technical-Platform.md#alastria-core-technical-platform

Smart Contracts

Smart Contracts are an instance of a computer program that runs on blockchain. In the case of permissioned blockchain such as Quorum, where only authorized users are able to interact with the ledger, an authorized user can create a contract by posting a transaction to the blockchain. It is important to notice that its code is fixed and cannot be changed after deployment. The code's execution is provoked by a received message either from a user of another contract and could provide utility to other contracts or require assistance from other Smart Contracts.

In this section we describe the different smart contracts developed to support the M-Sec use cases as well as some of the functionalities they provide.

1. M-Sec Token

A custom token was created specifically for research purposes. It is actually a cryptocurrency in the form of a smart contract running on Quorum Blockchain. It follows the ERC223[17] token standard. Preliminary implementations followed the ERC20 token standard but ERC223 is a superset of the previous standard offering security improvements and more usability and backwards compatibility with any services and functionalities designed and developed for ERC20. As fully compliant with ERC223, it implements a set of functions and events, such as *name()*, *transfer()*, *totalSupply()* and *Transfer event* which is emitted to the blockchain when an amount of Tokens is transferred from a user to another. Some indicative developed methods are presented in the following Table 2.

This Token has different applications in the use cases. It is firstly used as a payment currency to exchange value among the users of the Marketplace. Another implementation and configuration of the M-Sec Token allows us to use it as a "*Social Token*". Users of the platform have an initial balance and particular users are rewarded with more token based on specific criteria such as for example:

i) the most active user,

ii) the most social user,

iii) the user who uploaded the most popular content.

This Token acts as a mean to tokenize a loyalty points program with rewards.

Table 2: Overview of M-Sec Token's functions

```
contract ERC223 {
  uint totalSupply;
  function balanceOf();
  function name();
  function symbol();
  function decimals();
  function totalSupply();
  function transfer(to, value);
  function transfer(to, value, data);
  function transfer(to, value, data, custom_fallback);
  event Transfer(from, to, value, data);
}
```

In the following table more details are provided about the developed functions and events of the M-Sec Token:

[17] https://github.com/ethereum/EIPs/issues/223

Table 3: Detailed presentation of functions and events of M-Sec Token

Name	Input	Response	Description
totalSupply (function)	-	uint256 totalSupply	Get the total token supply
Name (function)	-	string _name	Get the name of token
Symbol (function)	-	bytes32 _symbol	Get the symbol of token
Decimals (function)	-	-	Get decimals of token
balanceOf (function)	address _owner	uint256 balance	Get the account balance of an account with address: address _owner
transfer (function)	address _to, uint _value	boolean	Transfer tokens, compatibility with ERC20
transfer (function)	address _to, uint _value, bytes _data	boolean	function that is always called when someone wants to transfer tokens. This function must transfer tokens and invoke the function *tokenFallback* if _to is a contract.
Transfer (event)	address indexed _from, address indexed _to, uint256 _value, bytes _data	-	Triggered when tokens are transferred and is emitted to the blockchain network
tokenFallback (function)	address _from, uint _value, bytes _data	-	A function for handling token transfers, which is called from the token contract, when a token holder sends tokens
TransferForSensor Data(event)	address indexed _from, address indexed _to, uint256 _value, uint32 _sensorID, string _name, uint32 _fromTime, uint32 _toTime		Triggered when tokens are transferred related to sensor data and is emitted to the blockchain network

TransferForMedia Data(event)	address indexed _from, address indexed _to, uint256 _value		Triggered when tokens are transferred related to media items and is emitted to the blockchain network
approve(function)	address _spender , uint256 _value)	bool success	O user (address A) allows to another entity (address B) to spent M-Sec tokens on his/hers behalf

2. Item Manager Smart Contract

The Item Manager Smart Contract allows the interaction of item/content creators (e.g. photos, multimedia items, sensor data etc.) with the platform and the blockchain. A user is able to upload all the information and metadata related to an item. To this direction, we have created dedicated "structs" (Figure 7), which are a special feature of Solidity contract-oriented programming language, in order to store for each item, the details (e.g. tags, information, metadata) and the unique address of its owner.

```
struct item {
      address owner;
      string URI;
      uint256 price;
      string tag;
      string info;
}
```

Figure 7. Item Manager Smart Contract

3. Sensors Smart Contract

This smart contract records all the registered IoT sensors. It gives the possibility to register a sensor and to change its information afterwards as well. Dedicated Solidity structures were created to store this information and functions to allow its retrieval.

A structure that allows the storing of the information is the following:

```
struct sensor {
      address sensor-Owner ;
      uint8 type-of-Sensor ;
      uint MSec-Token-Price ;
```

```
            uint32 timestamp-of-start ;
            uint16 frequency ;
            int32 latitude ;
            int32 longitude ;
            string url ;
            string name
      }
```

Figure 8. Sensors Smart Contract

In the following Table 4, more details are provided regarding our Sensors Smart Contract:

Table 4: Sensors Smart Contract details

Name	Input	Response	Description
registerSensor (function)	address sensor-Owner uint8 type-of-Sensor uint MSec-Token-Price uint32 timestamp-of-start uint16 frequency int32 latitude int32 longitude string url	Boolean success	Registration of a sensor to the dedication structure of the smart contract with the related information. Upon registration a verification of registration is returned
changeSensorInfo(function)	uint8 type-of-Sensor uint MSec-Token-Price uint32 timestamp-of-start uint16 frequency int32 latitude int32 longitude string url	Boolean success	The owner of the sensor changes some of the fields for example the price in M-Sec Tokens or its position
BuySensorData (event)	Uint32 sensorID uint32 fromTime uint32 toTime		Triggered when the data of a sensor is purchased and is emitted to the blockchain network
SensorCreated (event)	address indexed seller, uint32 indexed sensorID,uint8 sensorType, uint price, uint32 startTime, uint16 frequency, int32 latitude, int32		Triggered when a sensor is created and registered and is emitted to the blockchain network

	longtitude,string url, string name		
SensorChangedSeller (event)	uint32 sensorID, address seller		Triggered when the owner of a sensor is changed and is emitted to the blockchain network
SensorChangedPrice (event)	uint32 sensorID, uint price		Triggered when the price of a sensor is changed and is emitted to the blockchain network
SensorChangedUrl (event)	uint32 sensorID, string url		Triggered when the url of a sensor is changed and is emitted to the blockchain network
CompletedTransaction (event)	uint32 transID, address indexed buyer, uint32 indexed sensorID, uint32 fromTime, uint32 toTime, uint amount		Triggered when a transaction about a sensor is completed and is emitted to the blockchain network
changeSensorSeller (function)	uint32 sensorID1, address seller1	Boolean success	The owner of the sensor the ownership of a sensor
changeSensorPrice (function)	uint32 sensorID1, uint price1	Boolean success	The owner of the sensor changes the price in M-Sec Tokens or its position
changeSensorUrl (function)	uint32 sensorID1, string url1	Boolean success	The owner of the sensor changes the url of the sensor's data
getId (function)			The unique id of the sensor is returned
buyData (function)	address indexed buyer , uint32 sensorID, uint32 fromTime, uint32 toTime	Boolean success	A user purchases data of a sensor for a specific time period

It is important to note that functions like *changeSensorInfo, changeSensorSeller, changeSensorPrice, changeSensorUrl* succeed only when the owner of the sensor (specific address) attempts to change the fields, otherwise the access is denied.

The function *BuySensorData* directly communicates with M-Sec Token smart contract, when a user wishes to buy data for a specific period. If the user has sufficient funds and the information is correct then the transaction will be successfully completed. Upon success the event *Transfer* is emitted to the network informing the users who watch the smart contracts that this transaction took place.

4. Know Your Customer Smart Contract

A huge number of financial banking transactions takes place every day. It is indicative that in July 2019 the Society for Worldwide Interbank Financial Telecommunication (SWIFT) recorded an average of approximately 32 million transactions per day. Blockchain can enable parties with no particular trust in each other to exchange digital data on a peer-to-peer basis with fewer or no third parties or intermediaries. In the recent report Scientific and Technical Research Report of European Commission on Blockchain[18], the need for *Know Your Customer* mechanisms is highlighted: "*the obligation of cryptocurrency exchanges and custodian wallet providers within the scope of EU regulation to implement mechanisms to counter money laundering and terrorist fundraising, such as 'know your customer' (KYC)* ".

It is evident that previously mentioned works involve value exchange through blockchain transactions and dedicated created smart contracts, making Know Your Customer process necessary. In this direction, we are presenting an approach which blends smart contracts for exchanging value in the IoT domain on a decentralized manner, integrating a KYC process handling *on chain* and *off chain* data.

Recent works have tried to tackle the problem of data management and KYC for blockchain applications. Shabair et al.[19] introduced a blockchain-based KYC proof of concept system and an orchestration tool for managing private blockchain environments over large scale test beds. In their work they highlight the need for additional research on security and privacy issues of blockchain applications. Norvill et al.[20] presented a demo of a system that allows automation and permissioned document sharing in order to simplify and reduce the work required by the KYC process, while Zhang and Yin[21] conducted a research on a digital copyright management system based on blockchain technology. They focused mostly on PBFT (Practical Byzantine Fault Tolerance) consensus mechanism improved by Tendermint[22] replacing original Ethereum POW (Proof of Work), digital signatures and smart contracts to design user account management strategies, copyright review and applications for the needs of digital rights management. In our work we further explore the design and implementation of smart contracts for the KYC process on a decentralized approach.

Blockchain is beginning to transform industries and there is an increasing interest in exploring its potential for various production use cases, especially for supporting multi-party processes where members don't necessarily trust each other. However, there are many challenges that remain to be addressed such as trade-offs between respecting privacy and supporting transparency. Bhsaskaran et al[23] described the design of smart contracts for consent-driven and double-blind data sharing on the Hyperledger Fabric blockchain platform[24]

[18]A. Anderberg et al., "Blockchain Now And Tomorrow," 2019

[19] W. Shbair, M. Steichen, and J. François, "Blockchain orchestration and experimentation framework: A case study of KYC," in The First IEEE/IFIP International Workshop on Managing and Managed by Blockchain (Man2Block) colocated with IEEE/IFIP NOMS 2018, 2018

[20] R. Norvill, M. Steichen, W. M. Shbair, and R. State, "Blockchain for the Simplification and Automation of KYC Result Sharing," in 2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC), 2019, pp. 9–10

[21] X. Zhang and Y. Yin, "Research on Digital Copyright Management System Based on Blockchain Technology," presented at the 2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC), Chengdu, China, 2019

[22] Jae Kwon, "Tendermint: Consensus without Mining." 2014

[23] K. Bhaskaran et al., "Double-blind consent-driven data sharing on blockchain," in 2018 IEEE International Conference on Cloud Engineering (IC2E), 2018, pp. 385–391

[24] "Hyperledger Fabric," Hyperledger

into a KYC application, where the data are submitted, validated and kept within the ledger supporting different consent rules and privacy levels.

Vishwa et al.[25] presented a decentralized data management system for data privacy and control focusing on multimedia files. In their solution they use an external data lake, namely a centralized data storage solution on a cloud to store the transaction details of all the data added on the blockchain. In order to access the blockchain, a user signs up by broadcasting his identity and will be accepted by the consent of the majority of the nodes and will be provided his new identity and access permissions. In our approach we additionally use IPFS leading to a decentralized application and have successfully implemented smart contracts and software components, leveraging blockchain to automate tasks related to KYC process.

Our process of developing the smart contract to support KYC process is described through its use in the middleware services section.

Table 5. KYC process exposed methods

Name	Input	Response	Description
CreateNewUser (function)	Address, info	Boolean success	Creation of a new user, approve request
updateUser (function)	Address, info	Boolean success	Update the information of an existing user
suspendUser (function)	Address, info	Boolean success	Suspend the activity of a specific user
approveUser(function)	Address, info	Boolean success	Approve requested user
ExtendKYCduration (function)	Address, info	Boolean success	Update the information of a user and extend the duration of his/hers approval

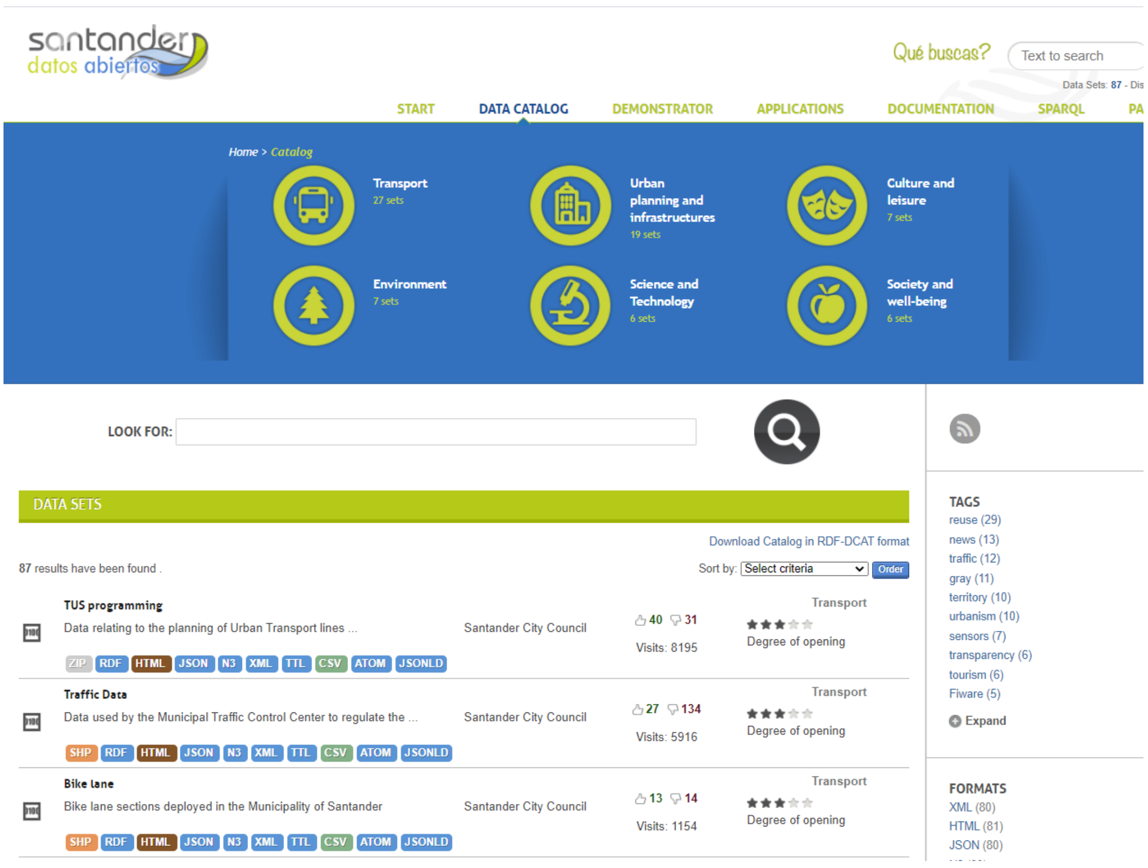
5. Smart City Data Smart Contract

This smart contract focuses on managing data from the smart cities of Santander and Fujisawa and support the use cases. It directly communicates with other tools of M-Sec project such as encrypted data storage and offchain storage. Additional flows are created as part of middleware services to support the interaction and integration with the rest of the platform.

This smart contract constitutes an extension of the Sensor Smart Contract oriented to better handle datasets provided by smart cities. Among others, this smart contract was used to integrate the data and datasets from Santander Open Data Platform[26]. Different datasets are provided about transport, urban planning &

[25] A. Vishwa and F. K. Hussain, "A Blockchain based approach for multimedia privacy protection and provenance," in 2018 IEEE Symposium Series on Computational Intelligence (SSCI), 2018, pp. 1941–1945
[26] www.datos.santander.es

infrastructure, culture & leisure, environment, science & technology, society, well-being and more. A view of the datasets is also shown in the following figure.



Data Sets: 87 - Dis

Home > Catalog

Transport 27 sets

Urban planning and infrastructures 19 sets

Culture and leisure 7 sets

Environment 7 sets

Science and Technology 6 sets

Society and well-being 6 sets

LOOK FOR:

DATA SETS

Download Catalog in RDF-DCAT format

87 results have been found .

Sort by: Select criteria Order

TUS programming
Data relating to the planning of Urban Transport lines ...
Santander City Council
40 31
Visits: 8195
Transport
Degree of opening
ZIP RDF HTML JSON N3 XML TTL CSV ATOM JSONLD

Traffic Data
Data used by the Municipal Traffic Control Center to regulate the ...
Santander City Council
27 134
Visits: 5916
Transport
Degree of opening
SHP RDF HTML JSON N3 XML TTL CSV ATOM JSONLD

Bike lane
Bike lane sections deployed in the Municipality of Santander
Santander City Council
13 14
Visits: 1154
Transport
Degree of opening
SHP RDF HTML JSON N3 XML TTL CSV ATOM JSONLD

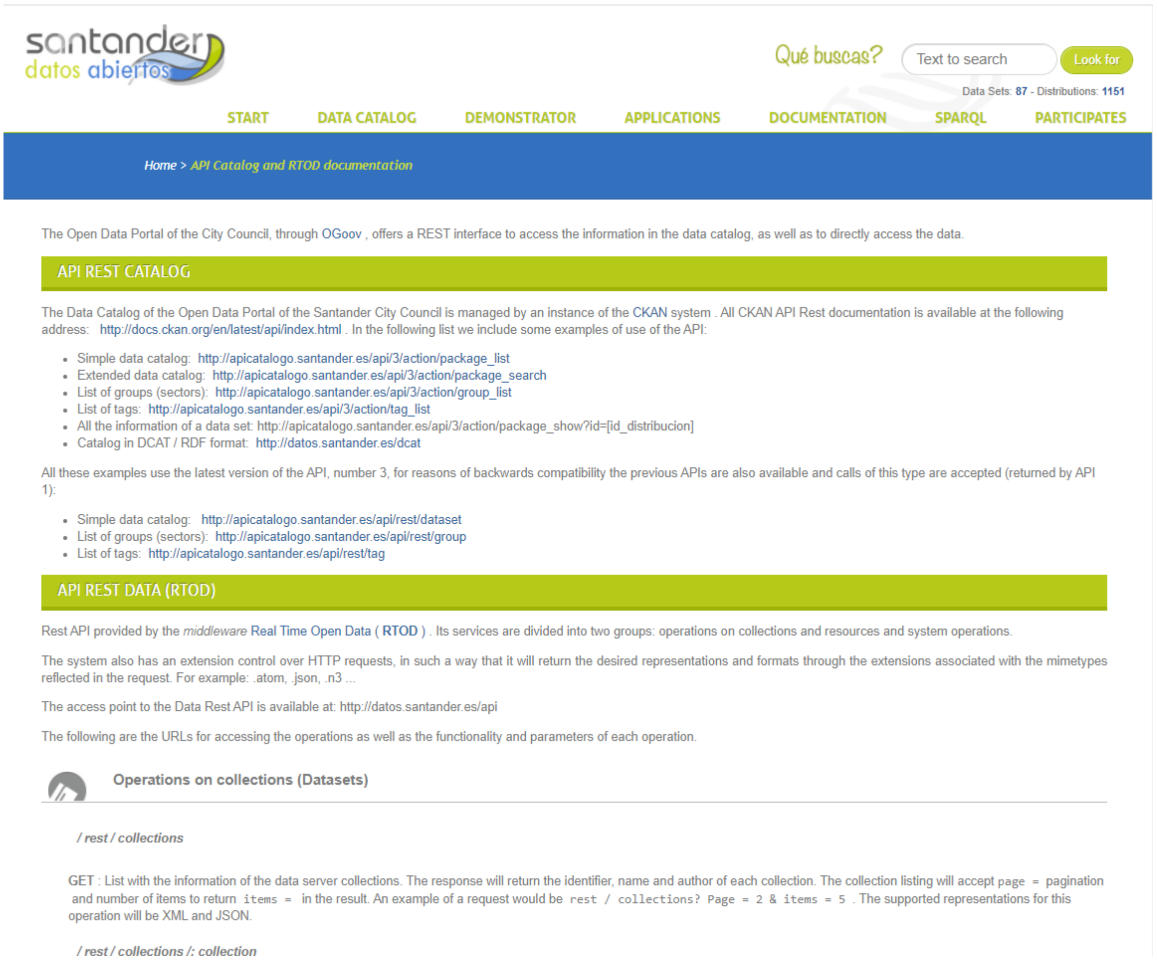
TAGS
reuse (29)
news (13)
traffic (12)
gray (11)
territory (10)
urbanism (10)
sensors (7)
transparency (6)
tourism (6)
Fiware (5)
Expand

FORMATS
XML (80)
HTML (81)
JSON (80)

Figure 9. Example of datasets from Santander Open Data Platform

As part of the integration, where the open data API is also utilized[27], this smart contract stores all the available metadata, making available the datasets to the marketplace. A view of the documentation of the Open Santander Platform API is shown in the following figure.

[27] http://datos.santander.es/documentacion-api/



The Open Data Portal of the City Council, through OGoov , offers a REST interface to access the information in the data catalog, as well as to directly access the data.

API REST CATALOG

The Data Catalog of the Open Data Portal of the Santander City Council is managed by an instance of the CKAN system . All CKAN API Rest documentation is available at the following address: http://docs.ckan.org/en/latest/api/index.html . In the following list we include some examples of use of the API:

- Simple data catalog: http://apicatalogo.santander.es/api/3/action/package_list
- Extended data catalog: http://apicatalogo.santander.es/api/3/action/package_search
- List of groups (sectors): http://apicatalogo.santander.es/api/3/action/group_list
- List of tags: http://apicatalogo.santander.es/api/3/action/tag_list
- All the information of a data set: http://apicatalogo.santander.es/api/3/action/package_show?id=[id_distribucion]
- Catalog in DCAT / RDF format: http://datos.santander.es/dcat

All these examples use the latest version of the API, number 3, for reasons of backwards compatibility the previous APIs are also available and calls of this type are accepted (returned by API 1):

- Simple data catalog: http://apicatalogo.santander.es/api/rest/dataset
- List of groups (sectors): http://apicatalogo.santander.es/api/rest/group
- List of tags: http://apicatalogo.santander.es/api/rest/tag

API REST DATA (RTOD)

Rest API provided by the *middleware* Real Time Open Data (RTOD) . Its services are divided into two groups: operations on collections and resources and system operations.

The system also has an extension control over HTTP requests, in such a way that it will return the desired representations and formats through the extensions associated with the mimetypes reflected in the request. For example: .atom, .json, .n3 ...

The access point to the Data Rest API is available at: http://datos.santander.es/api

The following are the URLs for accessing the operations as well as the functionality and parameters of each operation.

Operations on collections (Datasets)

/ rest / collections

GET : List with the information of the data server collections. The response will return the identifier, name and author of each collection. The collection listing will accept `page =` pagination and number of items to return `items =` in the result. An example of a request would be `rest / collections? Page = 2 & items = 5` . The supported representations for this operation will be XML and JSON.

/ rest / collections /: collection

Figure 10. A view of the documentation of the Open Santander Platform API

6. Value Handler Smart Contract

This smart contract is responsible for handling values in different formats such as hashed and is used to enhance the security aspects among the different assets and facilitate convergence of IoT security with blockchains to support an innovative smart city platform.

Some of the functions and events of this smart contract are presented in the following Table 6.

Name	Input	Response	Description
retrieveAll (function)	-	stored values	All stored values, depending on permissions
retrieveValue (function)	Address, value, info	stored value	stored value, depending on permissions
retrieveValuesForUser (function)	Address, value, info	stored values	stored values for a user, depending on permissions

uploadValue (function)	Address, value, info	Boolean Success	Stores a value, or bunch of values
Uploading (event)	Address, value		
Retrieving (event)	Address, value		

Table 6. Functions and Events of Values Handler Smart Contract

API: Blockchain framework exposed methods

In order to allow the communication and integration with other components, services, assets, several methods were developed. These methods are exposed via a RESTful API, while respective clients have been developed to facilitate the integration process and documentation with examples and indicative architecture figures and snippets. In the table that follows, some of the methods are presented, while we could note that part of them have a final form, while others are still updated to facilitate the integration with other assets, better support the use cases based on the feedback or improve security aspects of the provided services.

Method "getMSecTokenBalance"

Name	Input	Response	Description
getMSecTokenBalance ()	String publicKey	{"publicKey": "0x…", "balance": 10.5}	This is a GET method, which returns the balance of a specific account

Method "addFreeMSecTokens"

Name	Input	Response	Description
addFreeMSecTokens ()	String publicKey	{"publicKey": "0x….", "message": "added token with success"", "balance": 10.5}	This is a GET method, which for testing purposes adds 1 token to the balance of the given address

Method "insertNewValue"

Name	Input	Response	Description
insertNewValue ()	value	transaction details such as hash	This is a POST method, which stores a value to the

Method “getValuesByPublicKey”

Name	Input	Response	Description
getValuesByPublicKey ()	Public key	Json with values	This is a GET method, which returns values of a specific user

Method “getValueByTransactionHash”

Name	Input	Response	Description
getValueByTransactionHash ()	Transaction Hash	json with value	This is a GET method, which returns the value corresponding to a specific transaction hash

Middleware Services

This component refers to all the implemented basic blockchain services that include services such as search and indexing of the P2P network resource, advertising & discovery services, and messaging services for exchanging messages between the peers.

Know Your Customer

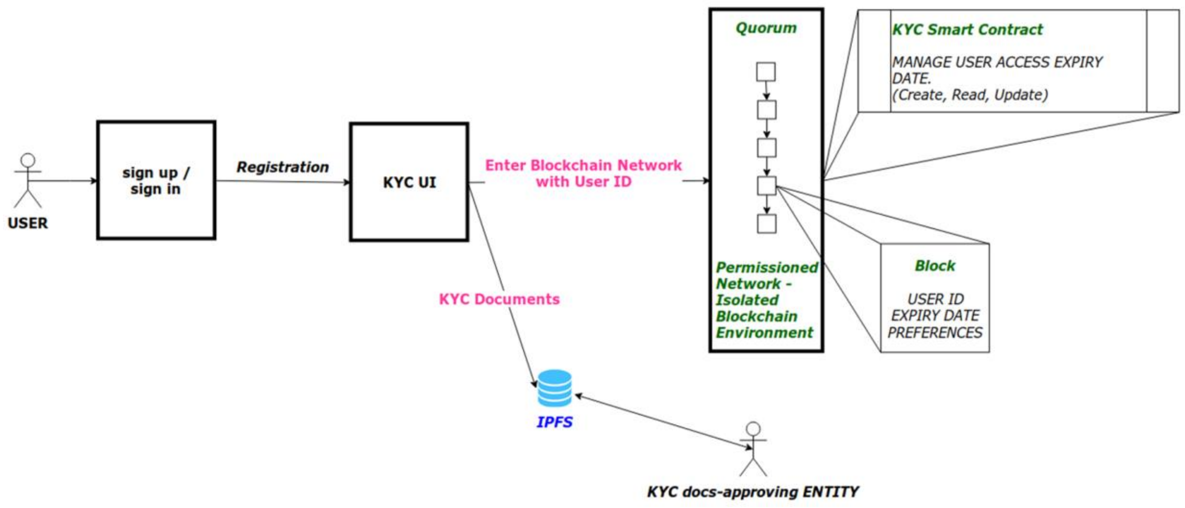

USER
sign up / sign in
Registration
KYC UI
Enter Blockchain Network with User ID
KYC Documents
IPFS
KYC docs-approving ENTITY
Quorum
Permissioned Network - Isolated Blockchain Environment
KYC Smart Contract
MANAGE USER ACCESS EXPIRY DATE.
(Create, Read, Update)
Block
USER ID
EXPIRY DATE
PREFERENCES

Figure 11. Overview of the KYC process for the M-Sec Platform

The KYC service as part of M-Sec Blockchain Middleware Services allows us to have a system where anonymity is maintained among user choices. The user identification has already been done outside the blockchain network, while no one inside the blockchain network is aware of the user's real identity. In this way M-Sec Platform will use KYC service to have services been delivered to users while hiding their true identity from their service provider or other users.

The M-Sec KYC Solution Concept includes the storage of personal data on an offchain database while the user is able to connect to the M-Sec Platform (and the blockchain network) using a special ID not relevant to his real identity. So the User Verification is conducted by an External Certificate Authority before accessing the system while the user uses the hash ID provided to him to interact with the System, as shown in the Figure 11.

Additionally, we have integrated the feature of the Expiration date. The System maintains an Expiry Date of users in the blockchain network. This information is stored within the smart contracts not in an external centralized database.

IPFS: InterPlanetary File System

Aiming to a more decentralized design we integrated blockchain with IPFS, a peer-to-peer version-controlled protocol and filesystem, run by multiple nodes, storing files submitted to it[28]. It combines distributed Hash Tables, Block Exchanges and Merkle Trees.

Using middleware, users are able to upload content to IPFS and place its unique hash code (address of the file) to the smart contracts running on Quorum blockchain. If we use a central database for storage, we benefit from the high throughput but this centralization does not coincide with the decentralized nature which blockchain advocates leading to a Single Point of Failure (SPOF) of the whole application. Facing the aforementioned drawback, IPFS being a peer-to-peer (p2p) file sharing system and Blockchain's

[28] Chen, Y., et al.: An improved P2P file system scheme based on IPFS and Blockchain. Big Data (Big Data), IEEE International Conference on (2017).

complementary component, settled exceptionally the SPOF problem, furnishing low latency and data distribution.

On-chain, off-chain data and access control

One of our goals is to design a blockchain-based decentralized content marketplace, which enables trustless disintermediation between sensor owners (and more generally data owners) and consumers. Using a dedicated created cryptocurrency (M-Sec Token) for payments, a consumer can buy data on the marketplace without involving a marketplace intermediary. This refers to the research and development of data privacy-enhancing mechanisms along with data access control and privacy policies that are necessary for the M-Sec framework. Moreover, it deals with the separation of data, meaning to identify what needs to be pushed on blockchain and what to remain off-chain, a decision that is always critical when designing blockchain platforms

Transaction Handler

One of the main and most important features of the Quorum is the private transaction mechanism. Transaction privacy is achieved by using the Ethereum Transaction Model and enhancing it with new parameters that specify the nodes in which the transactions should be published. The Constellation layer of Quorum that contains the transaction Manager and Enclave module is responsible for the private transaction handling. All the public transactions follow the already established p2p Ethereum network flow.

Additional mechanisms are implemented that:

i. allow only authorized users to commit a transaction and have access to the blockchain,
ii. verify the identity of user using cryptography algorithms,
iii. in case he is about to receive some data/service in exchange of M-Sec Tokens it is verified that he has already made the purchase.

The Transaction Handler could be regarded as a flow providing a layer before blockchain that performs a first process of the potential transactions to formulate them and optimize and verify the content to be inserted in the blockchain.

Upload Handler

This part of the Middleware Services provides functionalities for efficiently performing actions related to Assets. As an asset we could consider a file, a multimedia item, a dataset that could be described with a predefined set of fields such as:

- Title
- Timestamp of start
- Timestamp of end (whether applicable)
- Owner/Creator name (or Address)
- Price in M-Sec Tokens
- Description
- Location (latitude and longitude)

- URL related to the storage of the asset

All these functionalities are related to Smart Contracts in which we have defined Solidity structs keeping record of uploaded assets/items and we have additionally define related fields for metadata. These functionalities include:

- Uploading of an item by providing its details, as specified previously so it can be registered to the Item Manager Smart Contract.
- Browsing through all the available items registered in the smart contract.
- After specifying some criteria, the user is able to view an asset and its metadata.

Write/Update Metadata of Asset

This service is strongly connected to the Transaction Handler. As an indicative case, only the authorized users are allowed to update the metadata of an item. The user who has the right to update is the owner of the item or a user with a specific permission. The service handles the communication with the smart contracts and checks the rights of a user.

Package Information & Installation Instructions

Required Tools and dependencies

- Truffle Suite
- Solidity Programming Language
- Quorum Blockchain

Install Truffle Suite

Truffle suite:

```
npm install truffle -g
```

More installation instructions could be found in the following link: https://www.trufflesuite.com/truffle

Install Solidity

Ethereum, "Solidity," Ethereum, [Online]. Available: http://solidity.readthedocs.io.

Install Quorum Blockchain Network

https://github.com/synechron-finlabs/quorum-maker

https://github.com/jpmorganchase/quorum-examples/tree/master/examples/7nodes

Licensing

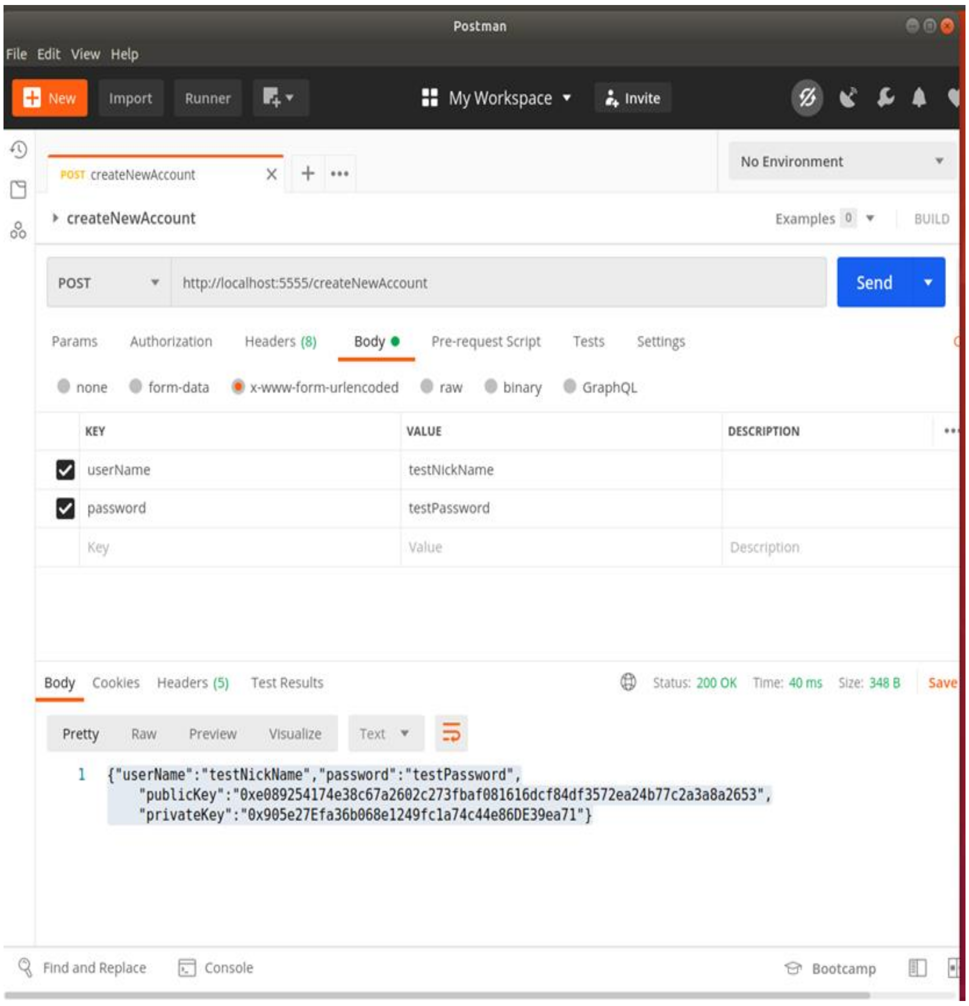
Quorum, the go-ethereum library (i.e. all code outside of the cmd directory) is licensed under the GNU Lesser General Public License v3.0

Solidity is licensed under GNU General Public License v3.0

API: Middleware Services exposed methods

Method “createNewAccount” M

Name	Input	Response	Description
createNewAccount()	String userName, String password	{"userName":"testNickName", "password":"testPassword", "publicKey":"0x…", "privateKey":"0x…"}	This is a POST method, which allows the creation of a new account provided the username and password



POST createNewAccount

POST http://localhost:5555/createNewAccount

Body: x-www-form-urlencoded

KEY	VALUE	DESCRIPTION
userName	testNickName	
password	testPassword	

Status: 200 OK Time: 40 ms Size: 348 B

```
{"userName":"testNickName","password":"testPassword",
    "publicKey":"0xe089254174e38c67a2602c273fbaf081616dcf84df3572ea24b77c2a3a8a2653",
    "privateKey":"0x905e27Efa36b068e1249fc1a74c44e86DE39ea71"}
```

Figure 12. Example call of createNewAccount function, using Postman

Method “uploadPhotoToIPFS”

Name	Input	Response	Description
uploadPhotoToIPFS ()	String publicKey, photo	json	This is a POST method, which uploads a photo to IPFS

Method “uploadDataToIPFS”

Name	Input	Response	Description
uploadDataToIPFS ()	String publicKey, datatype, data	json	This is a POST method, which uploads data to IPFS

Method “freezeMySensor”

Name	Input	Response	Description
freezeMySensor ()	publicKey, sensorid, authorization details	Boolean success	This is a POST method, which allows suspension of sensors owned by users, only when having sufficient authorization details

Method “freezeMyAccount”

Name	Input	Response	Description
freezeMyAccount ()	publicKey, authorization details	Boolean success	This is a POST method, which allows suspension of a users’ account, only when having sufficient authorization details

Method “freezeAccountByAdmin”

Name	Input	Response	Description
freezeAccountByAdmin ()	publicKey, authorization details	Boolean success	This is a POST method, which allows suspension of a users’ account, only when having sufficient authorization details

Method “freezeSensorByAdmin”

Name	Input	Response	Description
freezeSensorByAdmin ()	filters [location, time]	json (details, urls to IPFS)	This is a POST method, which allows suspension of sensors owned by users, only when having sufficient authorization details

Trust & Reputation Management

Introduction

Trust and Reputation (T&R) models have been proposed by many researches as an innovative solution for guaranteeing a minimum level of security between two entities of a distributed system that want to have a transaction or interaction. Thus, many studies, works and models have been designed, carried out and developed in this direction, leading to a current solid research field on which both academia and industry are focusing their attention. Many methods, technologies and mechanisms have been proposed in order to manage and model trust and reputation in systems such as P2P networks[29], ad-hoc ones[30], wireless sensor networks[31] or even multi-agent systems[32]. Such methods have been used in many environments like P2P networks, Wireless Sensor Networks (WSN), Vehicular Ad-hoc Networks (VANETs), Identity Management Systems, Collaborative Intrusion Detection Networks (CIDN), Cloud Computing Systems, Application Stores and of course the IoT.

T&R management is a very useful and powerful tool in environments where a lack of previous knowledge about the system can lead participants to undesired situations, specifically in virtual communities where users do not know each other at all or, at least, do not know everyone. It is in those cases where the application of trust and reputation mechanisms is more effective, helping a peer to find out which is the most trustworthy or reputable participant to have an interaction with, preventing thus the selection of a fraudulent or malicious one. Most of the current T&R models in the literature follow four general steps which are described by Marti and Garcia-Molina[33] (Figure 15):

1. **Collecting information** about a certain participant in the community by asking other users their opinions or recommendations about that peer.
2. **Aggregating all the received information** properly and somehow computing a score for every peer in the network.

[29] F. Almenarez, A. Marin, C. Campo, C. Garcia, “PTM: a pervasive trust management model for dynamic open environments”, First workshop on pervasive security and trust, Boston, USA; 2004.
[30] M. Moloney, S. Weber, “A context-aware trust-based security system for ad hoc networks”, Workshop of the 1st International Conference on Security and Privacy for emerging areas in communication networks, Greece; 2005, pp. 153–60.
[31] Boukerche, L. Xu and K. El-Khatib, “Trust-based security for wireless ad hoc and sensor networks”, Computer Communications 2007.
[32] J. Sabater and C. Sierra C, “REGRET: reputation in gregarious societies”, Proceedings of the 5th International Conference on Autonomous Agents, Canada, 2001.
[33] S. Marti and H. Garcia-Molina, “Taxonomy of trust: categorizing P2P reputation systems”, Computer Networks 2006.

3. **Selecting the most trustworthy or reputable entity** in the community providing a certain service and effectively having an interaction with it, assessing posteriori the satisfaction of the user with the received service.
4. **Punishing or rewarding** according to the satisfaction obtained, adjusting consequently the global trust (or reputation) deposited in the selected service provider.

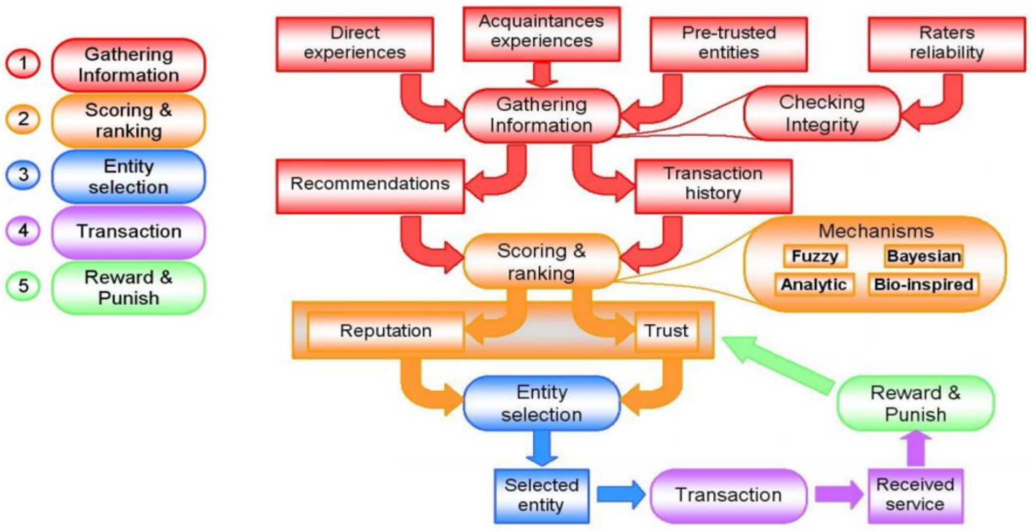


Figure 13: General steps followed in T&R models.

Currently, the idea of using a T&R engine on top of the Blockchain Middleware Services and the IoT Marketplace (already described in the previous sections) is being investigated. Such an engine would enhance the security mechanisms of M-Sec and make it possible to evaluate the actual content being shared through the Blockchain and the Marketplace, thus ensuring the trustworthiness of the several actors participating in the exchange or sharing of information, data and services.

The M-Sec T&R model (M-Sec T&RM)

Different models manage concepts such as Trust or Reputation in many different ways. Although there are some generic data structures for the domain of T&R provided for example by the Open Reputation Management Systems (ORMS) of OASIS[34], **there are no standards** for concepts like Trust and Reputation. In this subsection we try to provide some clear definitions of the main concepts that build up the M-Sec T&R model, and the main features that characterize it. In M-Sec T&RM we define Trust and Reputation as follows:

- **Trust:** The expectation that an interaction will be satisfactory based on *our personal experience*.
- **Reputation:** The belief that an interaction will be satisfactory based on the experience of *our social circle*.

Node A will have a high Trust index for Node B if the services provided from Node B to Node A have been evaluated from Node A positively. Node A will have a high Reputation index for Node B if the services provided from Node B have been evaluated from the social circle of Node A positively.

Definitions

The distinction between a trust and a reputation model is not always clear. However, in our opinion, those models making an explicit use of other participants' recommendations could be categorized as reputation models while the rest could be considered just as trust models.

[34] OASIS: https://www.oasis-open.org/committees/orms

Let's assume that actor-1 wants to find out some social characteristics of actor-2 for a specific service offered. The following terms can then be defined:

- **Popularity (P):** A counter which monitors how many times actor-2 has received or may receive a request (how many "hits" it has). The Popularity Index is an accumulative and comparative indicator, and is used to determine the stability of Reputation and Trust.
- **Trust (T):** The belief of actor-1 that actor-2 is going to deliver the correct service based on previous interactions of actor-1 with actor-2. The Trust Index of actor-2 provided by actor-1 is a property which states how many times actor-2 has successfully shared its services with actor-1. Trust is "subjective", because it is estimated from perspective of the individual trustor (actor-1 in this case).
- **Reputation (R):** The belief of actor-1 that actor-2 is going to deliver the correct service based on previous interactions of other actors. The Reputation Index can be calculated from the Trust that other actors (apart from actor-1) have on actor-2. In other words, this metric determines the belief of others on an actor and is useful especially when actor-1 does not have enough data to extract a Trust Index for actor-2 (because e.g. there are no interactions between the two actors yet).
- **Reliability (R'):** An absolute indicator of the performance of the actor that quantifies its efficiency to offer successfully its services relatively to its ideal or normal operation. The Reliability Index should be based on criteria like: response time upon request, ability to communicate, quality of service provided, etc.
- **Dependability (D):** A social measure combining all the above social measures. It can be simply derived by the expression $\boldsymbol{D = a \cdot T + b \cdot R + c \cdot R' + d \cdot P}$ where *a, b, c* and *d* (non-negative integers) are the weights of the measures and $\boldsymbol{a + b + c + d = 1}$. For this calculation, Popularity has to get normalized. By selecting the appropriate weights, we can provide the expression of the Dependability Index that we want. For example, when there are only a few interactions between actor-1 and actor-2, then the Trust Index should have a low weight and the Reputation Index should have a high weight. This means that the weights should change dynamically and be set according to the users or developers preferences.

General Features

Reputation connects closely to the concept of Trust, but there is a clear difference, which can be illustrated by the following two scenarios:

- Actor-1 trusts actor-2 because Actor-2 has a good Reputation. This reflects that Reputation can be used to build Trust.
- Actor-1 trusts actor-2 despite the bad Reputation of Actor-2. This reflects that even if actor-1 knows the Reputation of actor-2, actor-1 has its own private knowledge (e.g. direct experience with actor-2) which is considered to be more important.

Generally, an actor can be evaluated only by information gathered from other actors. Its Dependability can be calculated by each and every other actor of the community (a subjective estimation) or by the whole system (a more, but not totally, objective estimation). Depending on its (subjective or objective). Both big and small time-windows are used to quickly detect malicious or unsatisfactory behaviour and avoid the fast redemption of blacklisted actors. Moreover, feedback from recent interactions has a higher weight than this of older actions.

Benevolent actors should have more opportunities than newcomers. As a result, newcomers with 0 interactions with other actors will have Reputation equal to 0. However, an extra rule has to be applied to the model we have designed to give the opportunity to newcomers that have a low Reputation (because of the small number of interactions with other actors) to be chosen as service providers at some point and start building their Reputation. For example, 10% of the recommendations from the platform should introduce newcomers to the rest of the community. The same applies for actors which have low Reputation due to malicious or unsatisfactory behaviour in the past. In other words, this rule enables the **social integration** and **reintegration** of the actors to the system. Moreover, this rule is necessary for the first moments of the social community that may be born from M-Sec, as the network, at its **initial state**, will not have any actors with high Reputation.

It should be noted that, in contrast with many T&R models, we choose to use **different** Trust and Reputation **scores** for different services provided by the members of the network. This feature helps as face quite many security threats. For example, abuse of a high achieved Reputation is easily avoided.

Calculation of Trust & Reputation

In M-Sec T&RM, only the idea of subjective Trust is modelled, as we claim that subjectiveness is embedded in Trust's meaning. Strong Trust on an actor cannot and should not be affected by claims of a third party. In order to model Trust, the experiences based on which the Trust is calculated need to be modelled. Thus, we need memory. For that purpose, the M-Sec Blockchain can be used to store the "social" interactions between actors and the evaluations of the corresponding services. Some crucial attributes that have to be stored in these Log Files are:

- **Satisfaction (s):** This value is essentially a subjective QoS indicator. The Satisfaction is automatically derived by the absolute values of the service based on their correctness. For example, a sensor that suddenly reports a really high temperature will be assigned a satisfaction rating based on the correctness of this report. If there is a fire, the Satisfaction is high, but if the is not, the Satisfaction is zero. Since an actor that regulates the alarms can consult more than one sensors, a malicious or faulty sensor will quickly lose any trust. If the sensor is fixed, the Social Reintegration part of the system will allow it to build trust again.
- **Weight (w):** This is a value indicating how crucial the service is for the well-being of the actor. It is used in order to prevent a malicious actor from providing a minor service well and then exploiting the built Trust and providing a crucial service poorly. Due to this value, it is difficult for the Trust index to increase just because of minor services, whereas it can drop quickly in case of a crucial service with low quality.
- **Fading factor (f):** When new interactions take place, the importance of older ones should decrease. The fading factor addresses this issue and forces peers to stay consistent with their previous behaviour. Old interactions have lower fading factor values, so an actor cannot misbehave relying on its good history. The fading factor makes the Social Reintegration of ex-malicious nodes possible, meaning that if they become benevolent, it is possible for them to get a second chance and form new ties with the network. Of course multiple incidents of misbehaviour can get an actor permanently black-listed. This fading factor can be set by the system administrator so that the actors take under consideration the last N interactions with any other actor.

When an actor wants to calculate the **Trust Index** of another one, it looks into the appropriate Log Files in the Blockchain and calculates the trust value as the weighted average of the log entries using:

$$\mu_t^k = \frac{\sum_{i=1}^{N}(s_i \cdot w_i \cdot f_i)}{W} \qquad (1)$$

where W is the normalization co-efficient which ensures that the trust value will be between [0,1] and is calculated by:

$$W = \sum_{i=1}^{N} (w_i \cdot f_i) \quad (2)$$

The **mean value (μ)** is a measure of the overall observed behaviour of the actor and indicates the expected satisfaction value of the next interaction. However, it is needed to know how confident we can be about the value of μ i.e. how much the satisfaction from the service may actually deviate from μ. Thus, the **standard deviation (σ)** of the behaviour is also calculated. To reduce the computational overhead, the calculation of the later occurs simultaneously with the calculation of the mean value following the formula:

$$\sigma_t^k = \frac{\sqrt{\sum_{i=1}^{N}(s_i{}^2 \cdot w_i \cdot f_i) \cdot W - \left(\sum_{i=1}^{N}(s_i \cdot w_i \cdot f_i)\right)^2}}{W} \quad (3)$$

Finally, we define Trust as:

$$\boldsymbol{T^k = \mu_t^k - \sigma_t^k} \quad (\mathbf{4})$$

To sum up, μ shows the satisfaction that actor-1 should expect from actor-2, while σ shows how predictable the behaviour of actor-2 is. This means that if $T = 0.5$ then there is an 84% probability that the satisfaction for the service will be 0.5 or greater. That way the service providers that are not consistent and have an ever changing and oscillating behaviour will have lower Trust indexes even if their μ value is higher.

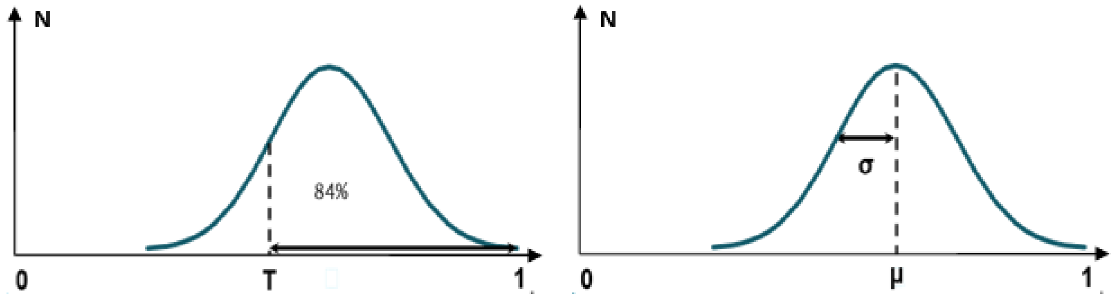


Figure 14: Calculation of the Trust Index of an actor.

Similar approach is being followed to calculate the **Reputation Index**, although this metric needs to extract more interactions logs from the Blockchain.

Testing

TRMSim-WSN

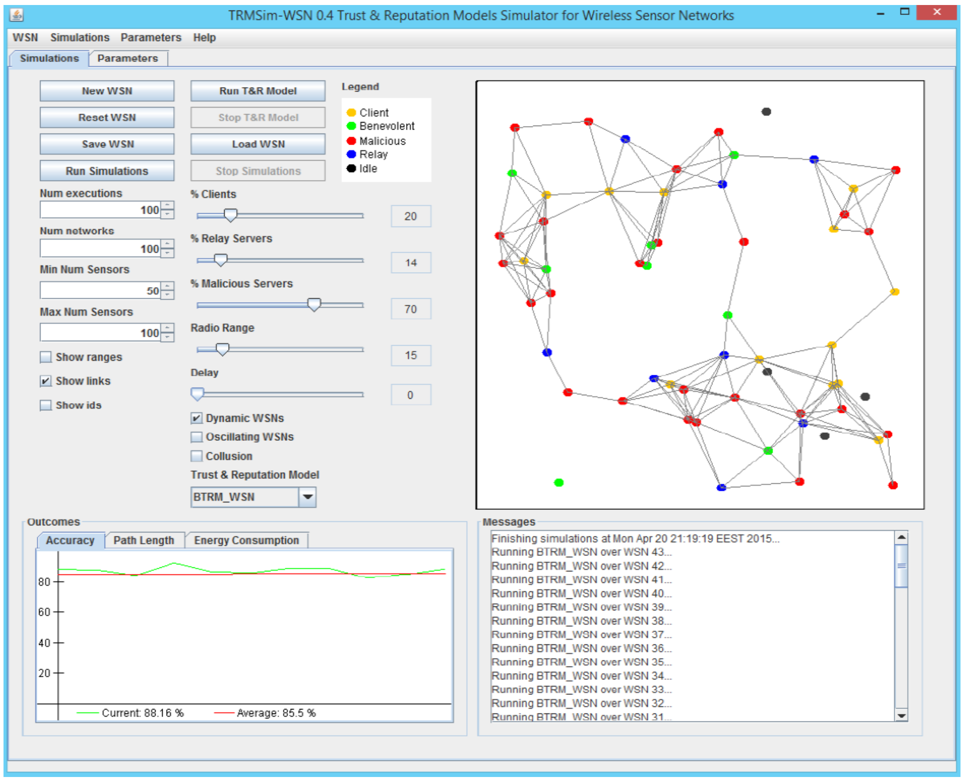
In order to test our T&R model, we used TRMSim-WSN[35], a simulator for T&R models. The TRMSim-WSN is a Java-based T&R models simulator aiming to provide an easy way to test a trust and/or reputation model over WSNs and to compare it against other models.

The TRMSim-WSN is, as far as we know, the state-of-the-art simulation platform for confidence-renowned systems. It is aimed at simulating algorithms for reputation and trust management in WSN systems, but the same principles can apply to IoT systems in general. The simulation can be run over a single randomly generated WSN or over a set of networks. The user is able to define parameters of the network, such as the percentage of clients and that of malicious nodes. Network topologies may also be loaded from and saved to XML files. Sample trust and reputation models have been included and an API is offered which provides a

[35] F. G. Marmol and G. M. Perez, "TRMSim-WSN, Trust and Reputation Models Simulator for Wireless Sensor Networks".

template for the users to help them easily load new T&R models to the simulator[36]. For the tests, parameters that can be configured are: number of executions, number of random networks to be tested, % of Malicious Actors, Collusion between Malicious Actors, Oscillating behaviour of actors, etc.

To evaluate our T&R model we compared it with three predominant (as of today) T&R models (Eigentrust, PeerTrust and PowerTrust) as well as with a relatively new system known as BTRM (Bio -Inspired Trust and Reputation Model) that applies a biological algorithm known as Ant-Colony System.



Num executions: 100
Num networks: 100
Min Num Sensors: 50
Max Num Sensors: 100
☐ Show ranges
☑ Show links
☐ Show ids
% Clients: 20
% Relay Servers: 14
% Malicious Servers: 70
Radio Range: 15
Delay: 0
☑ Dynamic WSNs
☐ Oscillating WSNs
☐ Collusion
Trust & Reputation Model: BTRM_WSN

Current: 88.16 %
Average: 85.5 %

```
Finishing simulations at Mon Apr 20 21:19:19 EEST 2015...
Running BTRM_WSN over WSN 43...
Running BTRM_WSN over WSN 42...
Running BTRM_WSN over WSN 41...
Running BTRM_WSN over WSN 40...
Running BTRM_WSN over WSN 39...
Running BTRM_WSN over WSN 38...
Running BTRM_WSN over WSN 37...
Running BTRM_WSN over WSN 36...
Running BTRM_WSN over WSN 35...
Running BTRM_WSN over WSN 34...
Running BTRM_WSN over WSN 33...
Running BTRM_WSN over WSN 32...
Running BTRM_WSN over WSN 31...
```

Figure 15: TRMSim-WSN.

We run simulations both in simple networks and in networks with dynamic entry or oscillating behaviour of actors. Measurements of the average satisfaction were made at various percentages of malicious actors (10%, 50% and 90%). The results are given in the next figure.

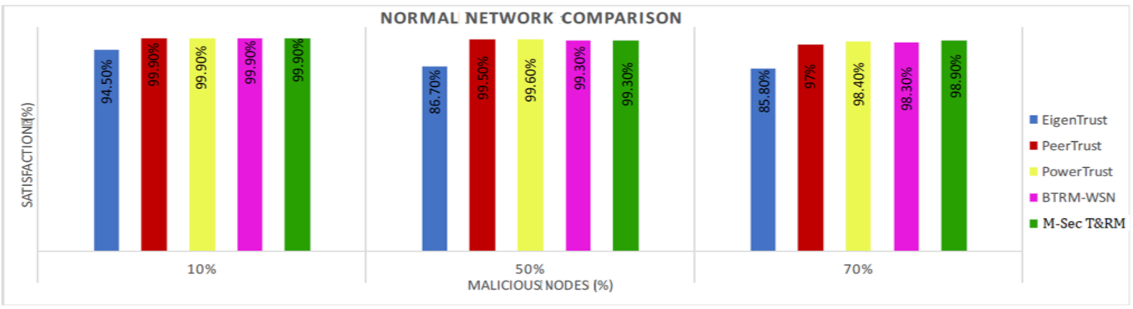


Figure 16: Normal Network Comparison

[36] F. G. Marmol, "Implementing and Integrating a new Trust and/or Reputation Model in TRMSim-WSN".

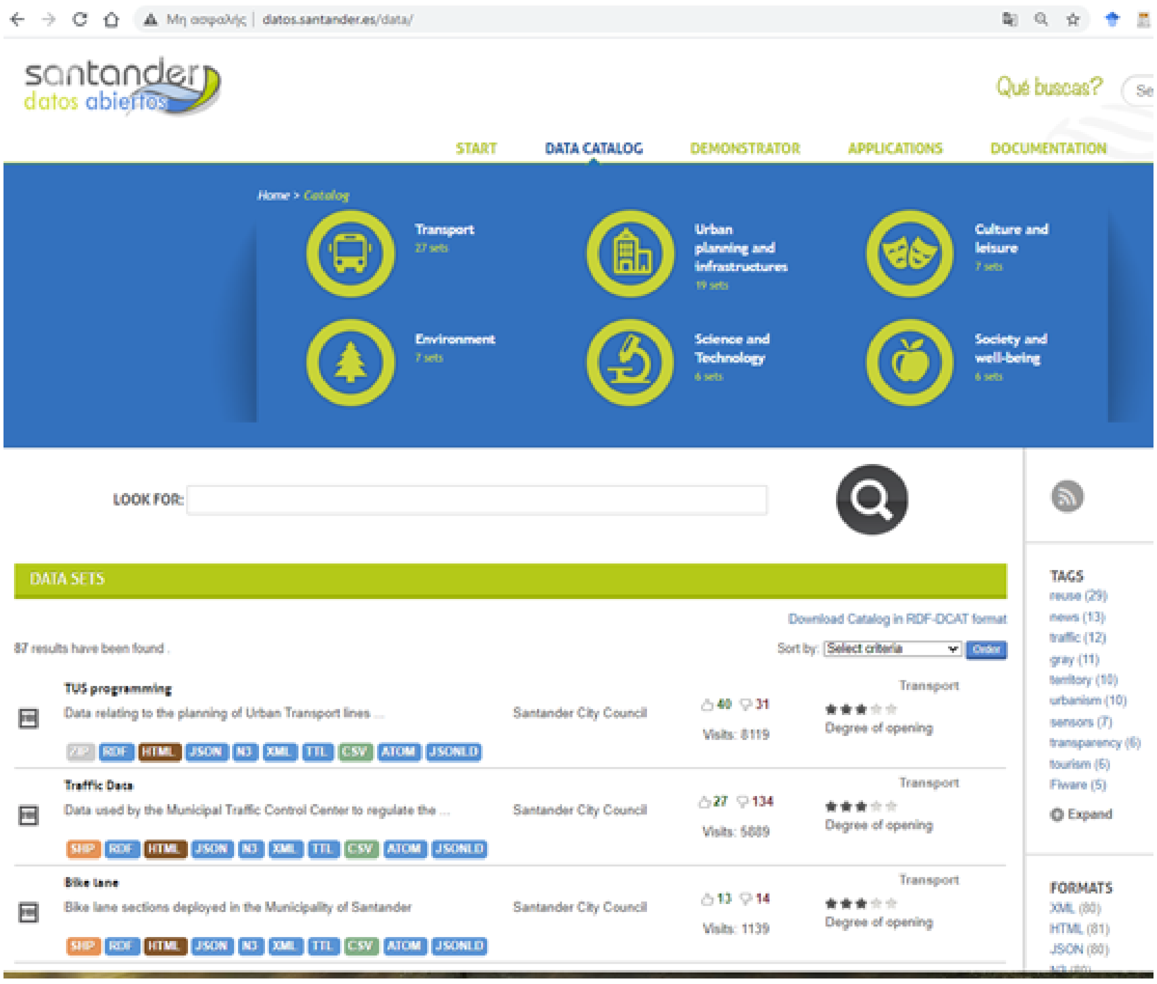
From the above, it can be seen that our models performance is comparable to that of the other models (and in some cases better).

Applying T&R in M-Sec with Santander Open Data Platform

The model can be implemented as a mechanism integrated into the M-Sec Blockchain Middleware. The main requirement for the mechanism to provide results is for an evaluation mechanism to also be in place, close to the point where the services are first provided.

As it can be seen in the figure below, such an evaluation mechanism does exist in platforms such as the Santander Open Data Platform. Features that enable end-users to evaluate a service (through a like-dislike option, a 5-stars system, or other), can be used to identify the Satisfaction metric that was recognized in the previous subsections. This value is essentially the subjective QoS indicator needed to “feed” the T&R model and extract the higher level metrics that identify the Trust over a service provider or a specific service.

At this level, tests with real evaluation data can be initiated to enable the functionality of the T&R engine and initiate its validation in the given scenarios.



87 results have been found .

TUS programming Data relating to the planning of Urban Transport lines ... ZIP RDF HTML JSON N3 XML TTL CSV ATOM JSONLD	Santander City Council	40 31 Visits: 8119	Transport ★★★☆☆ Degree of opening
Traffic Data Data used by the Municipal Traffic Control Center to regulate the ... SHP RDF HTML JSON N3 XML TTL CSV ATOM JSONLD	Santander City Council	27 134 Visits: 5889	Transport ★★★☆☆ Degree of opening
Bike lane Bike lane sections deployed in the Municipality of Santander SHP RDF HTML JSON N3 XML TTL CSV ATOM JSONLD	Santander City Council	13 14 Visits: 1139	Transport ★★★☆☆ Degree of opening

Figure 17. Evaluations from the Santander Open Data Platform as input to T&RM

Crypto Companion Database

General Description of the Prototype

Since it is necessary that sensitive data stored has to be secured and private, the CCDB is proposed as a parallel system to the blockchain for the encrypted storage. The blockchain will save a hash created from the encrypted sensitive data, and the CCDB will store the sensitive data encrypted together with the hash. The hash will be used to have a connection between the transaction in the blockchain and the data stored in the database.

In order to be compliant with the GDPR, the hash stored in the Blockchain is not going to be created from original but the encrypted data.

The CCDB will encrypt the data with an asymmetric public/private key pair.

This data could only be accessed by the owner, which will have to be authenticated, and the authorized operators allowed by the owner. The authorization is not part of the CCDB as it will be carried by the Blockchain itself, so the component will ask the Blockchain for it.

With this database insertion, deletion and consultation of the information will be possible. The modification process will be a bit more complex as the hash that holds the link between the blockchain and the database will change if the information changes, so if a modification is needed it will be done by deleting the old information and inserting the new one.

Each application in the ecosystem can have its own CCDB; therefore data will always be distributed. In order to make it accessible and replicated if wanted, the key pair can be replicated on any system by providing the 12-word mnemonic. To reduce the amount of data held by a single database, the location of specific information can be stated in the blockchain transaction.

The following figure shows how data can be accessed:

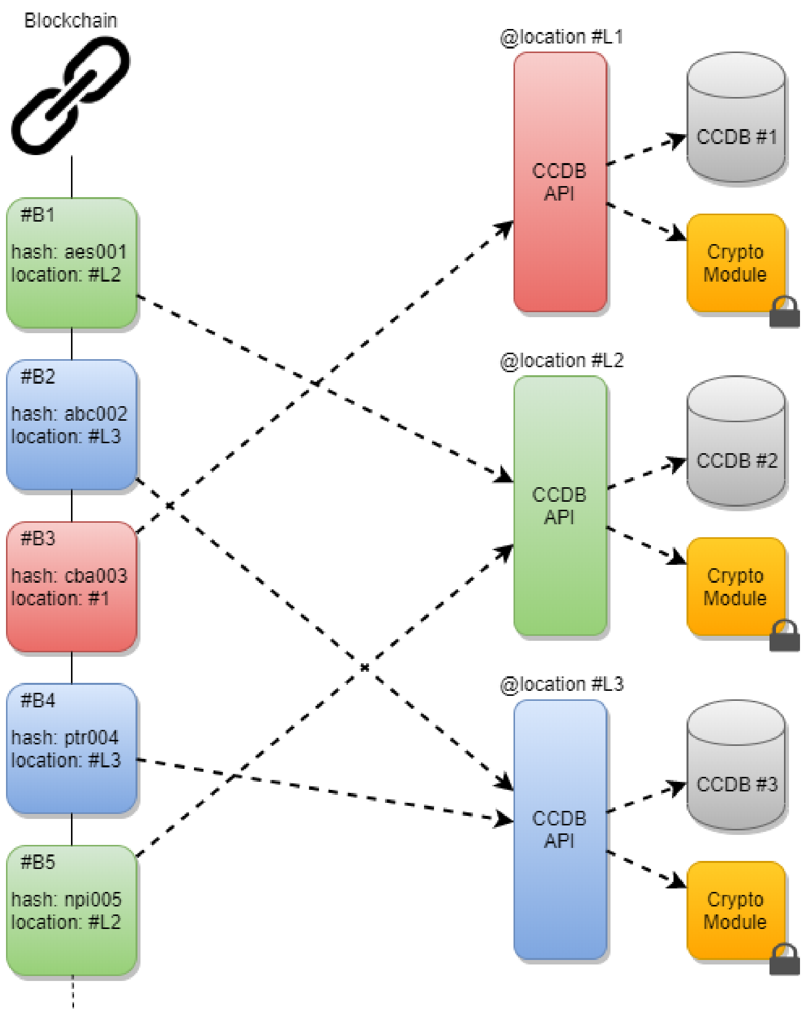


Figure 18. Access to distributed data.

Components

The components used to create the CCDB are the following:

- Crypto Module
- Companion DB Module

The evolution of the Companion DB Module with the Crypto Module makes a secured database, as the data will be encrypted by an asymmetric key pair, so it will be called **Crypto Companion Database**.

The Crypto Module will be used independently on any type of database (currently only supports MongoDB) and in any software because it provides an API to encrypt/decrypt data. The API of this module is designed as a private API with no access to the internet, so it does not provide any security.

The Companion DB Module has a public API that can be used to save, delete and query data. It also provides an authentication layer in order to secure the users that access the data. This module also provides an authorization layer in order to know if the owner of the data allows an operator or external user to see it. This authorization layer will make use of the Smart Contracts on Blockchain described in the previous sections.

In the following diagram an overview of the components is shown:

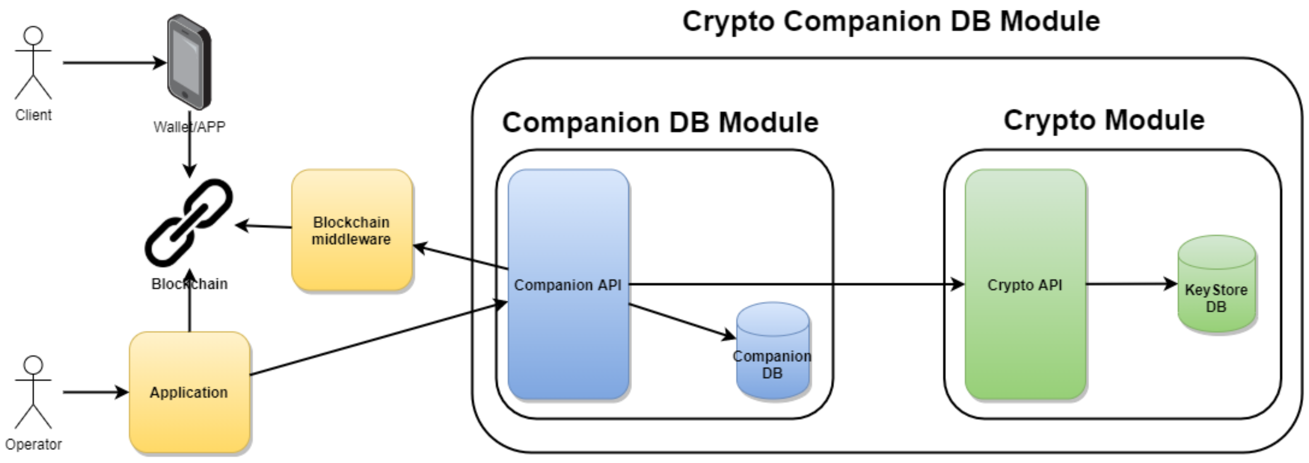


Figure 19. Crypto Companion Database Module components.

Crypto Module

This module allows a user to encrypt and decrypt data.

This module has two components:

- The Crypto API, that is in charge of encrypts and decrypts the data with the keys stored in the KeyStore DB.
- The KeyStore DB, that is a MongoDB that holds the key pairs to encrypt and decrypt data by the users.

The Crypto API provides:

- A method to create an asymmetric key pair:

```
enroll(ownerHash: string, mnemonic: string, blockchainOwnerKeys: object)
```

The creation of the public/private key pair can be made by providing a 12-word mnemonic, allowing replicating the keys in other applications. It will be useful if the user wants to authorize always with the same public/private key, and also will allow a distributed system to be able to decrypt data in a distributed way.

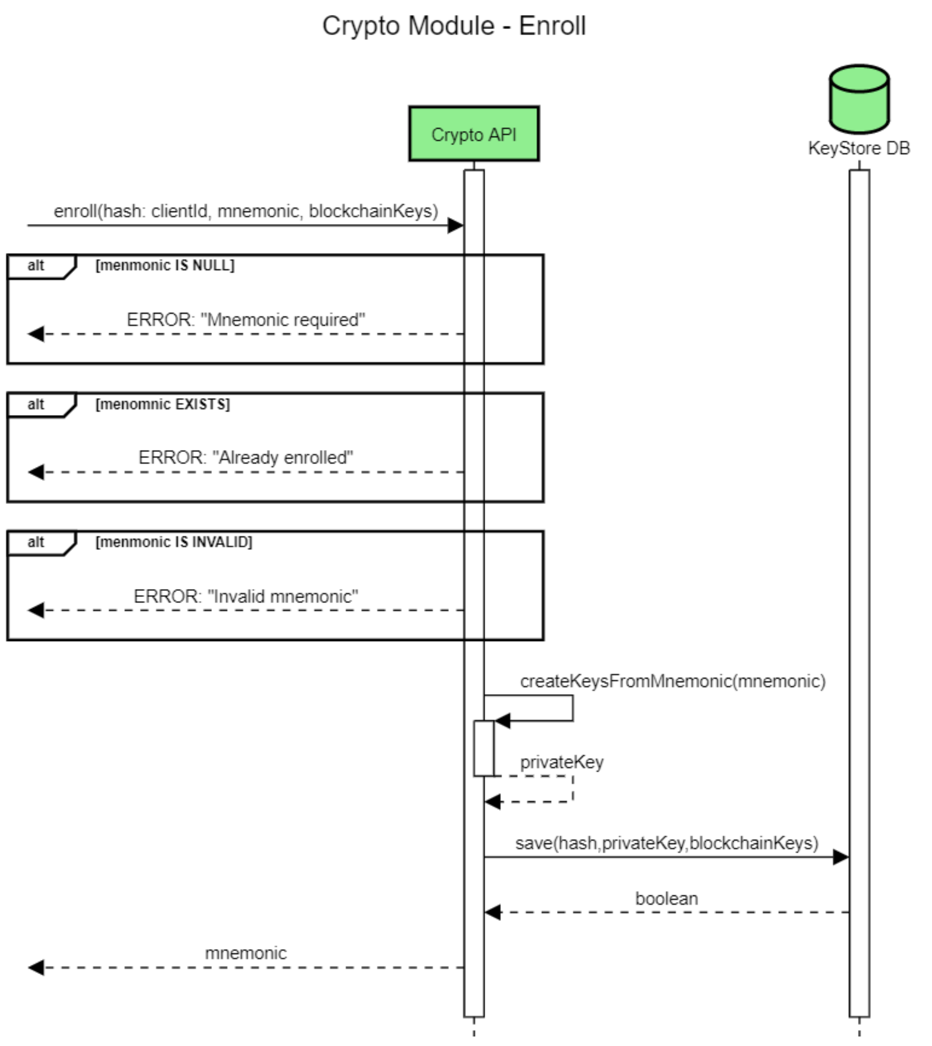

Crypto Module - Enroll
Crypto API
KeyStore DB
enroll(hash: clientId, mnemonic, blockchainKeys)
alt
[menmonic IS NULL]
ERROR: "Mnemonic required"
alt
[menomnic EXISTS]
ERROR: "Already enrolled"
alt
[menmonic IS INVALID]
ERROR: "Invalid mnemonic"
createKeysFromMnemonic(mnemonic)
privateKey
save(hash,privateKey,blockchainKeys)
boolean
mnemonic

Figure 20. Sequence diagram. Enrolment in Crypto Module.

- A method to encrypt data:

```
encrypt(hash: string, data: string):
```

This endpoint will take the private key of the user with the hash provided and encrypt the string with the data in the payload. If the user does not exists it will return the data sent as it is.

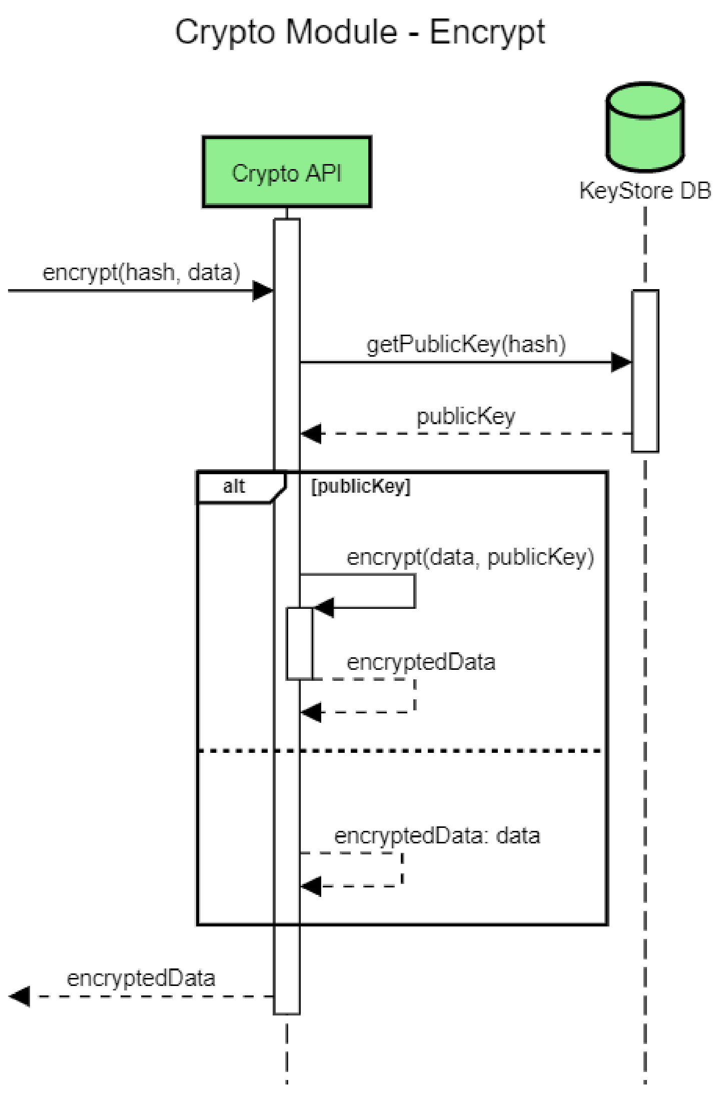


Figure 21. Sequence diagram. Data encryption in Crypto Module.

- A method to decrypt data:

```
decrypt(hash: string, data: string)
```

  This endpoint will take the private key of the user with the hash provided and decrypt the string with the data in the payload. If the user does not exist, it will return the data sent as it is.

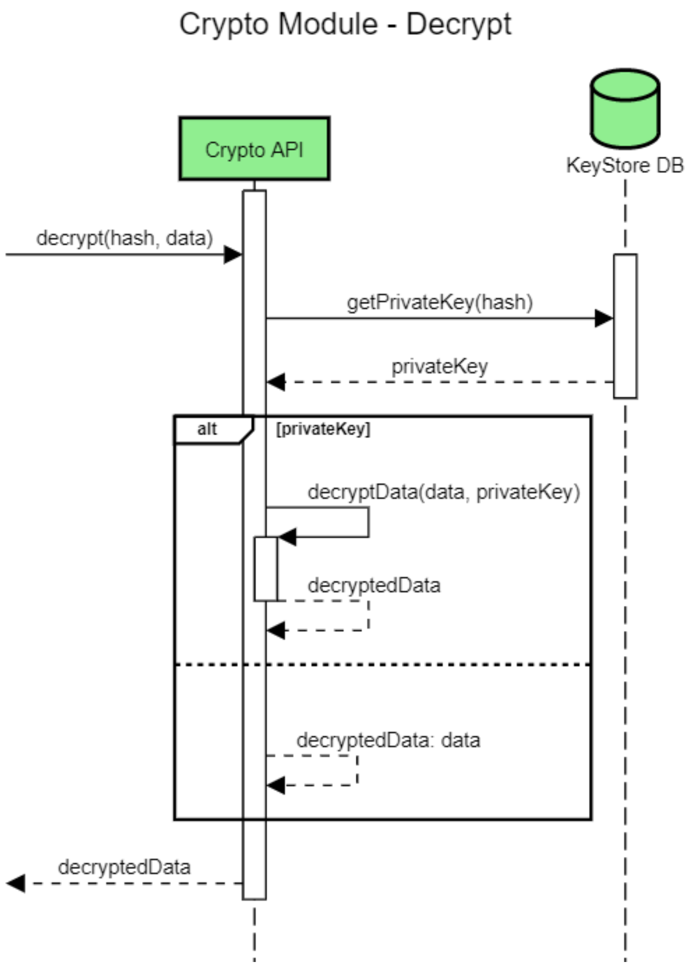


Figure 22. Sequence diagram. Data decryption in Crypto Module.

- A method to delete the keys:

```
disenroll(hash: string)
```

This endpoint will delete the public/private keys associated with the hash provided.

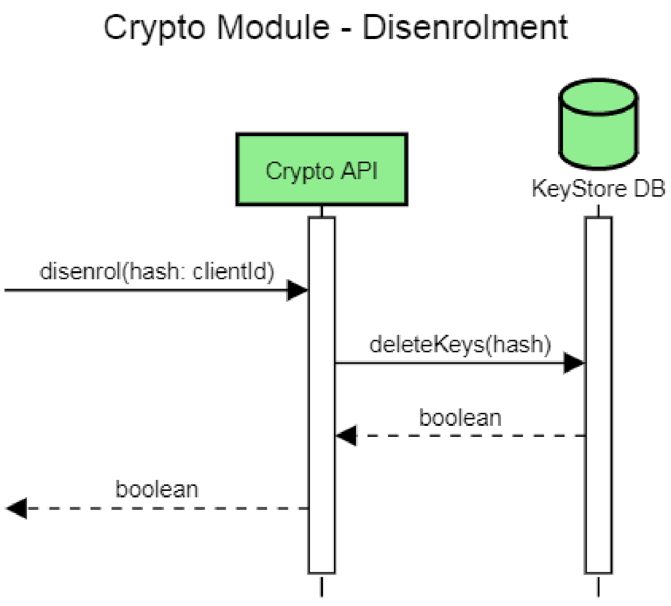


Figure 23. Sequence diagram. Disenrollment in Crypto Module.

The API of this module is intended to be private and only provide mechanisms to encrypt and decrypt data, so it does not provide authentication.

The KeyStore DB will store the public and private keys created by the Crypto API with a hash that will act as an identifier.

So the keys stored in the database will look like:

- hash: 32-64 hexadecimal string identifying the user.
- privateKey: The Private key generated by an asymmetric key algorithm that matches the public key.
- mnemonic: a set of 12-word that is used to create an account into the blockchain and to generate the privateKey.
- blockchainOwnerKeys: Object provided by the creation of a user in the blockchain.

Crypto Companion Database Module

This module allows a user to have an authentication system and save data encrypted. It also provides other users with the possibility to read data from a user if authorized.

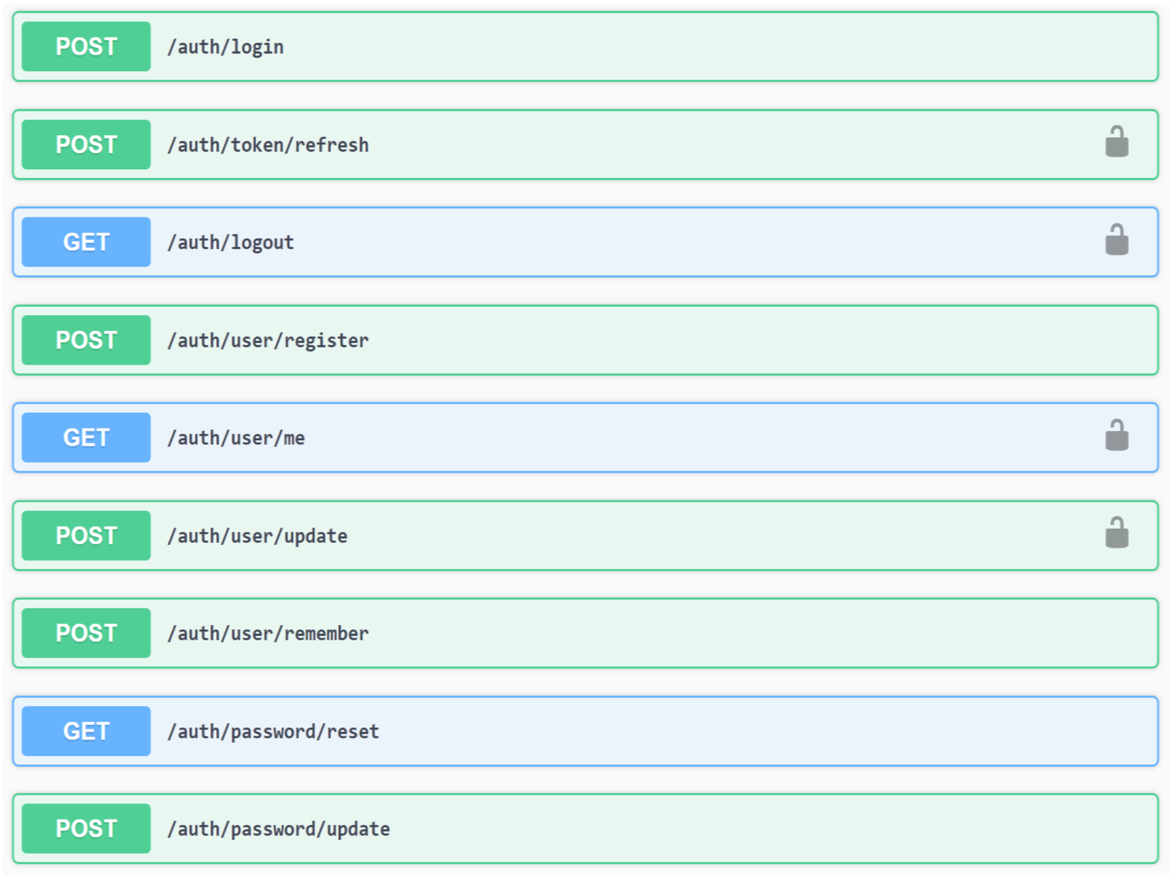
This module has two components:

- The CCDB API, is in charge of authentication and managing all the data providing methods to save, read and delete data in the database.
- The CCDB, is a MongoDB that stores the encrypted data.

The CCDB API provides:

Authentication API.

- A set of methods to register, update user information and recover a password.


POST /auth/login

POST /auth/token/refresh

GET /auth/logout

POST /auth/user/register

GET /auth/user/me

POST /auth/user/update

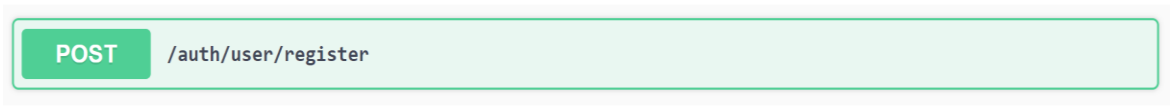
POST /auth/user/remember

GET /auth/password/reset

POST /auth/password/update

Figure 24. Authentication API in Crypto Companion Database Module.

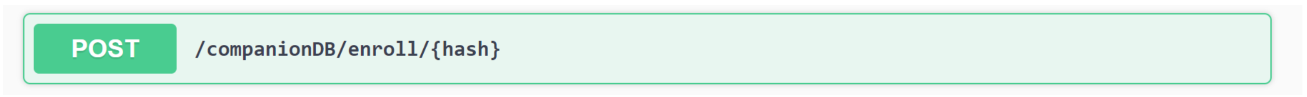
- A method to register:


POST /auth/user/register

The registration of a user will also trigger the enrolment on the Crypto Module, so the keys will be created during the registration.

Data Management API.

- A method to enrol:


POST /companionDB/enroll/{hash}

The creation of the public/private key pair can be made by providing a 12-word mnemonic, allowing replicating the keys in other applications. It will be useful if the user wants to authorize always with the same public/private key, and also will allow a distributed system to be able to decrypt data in a distributed way.

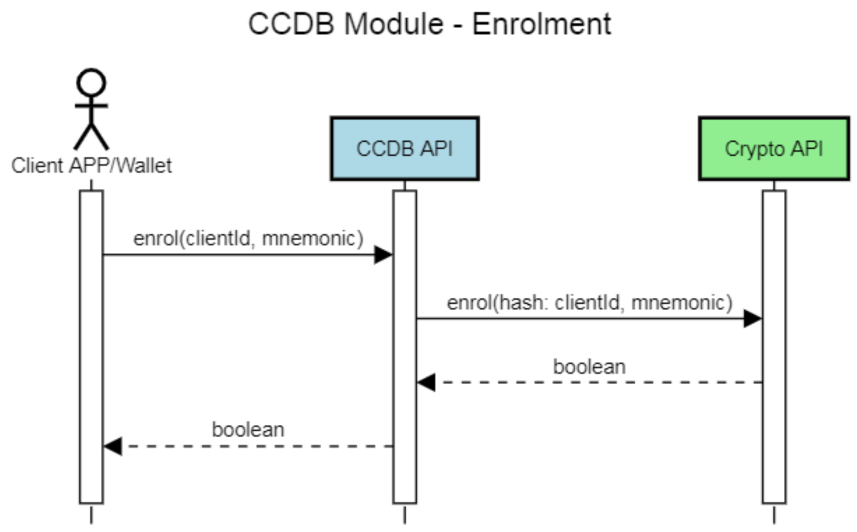


Figure 25. Sequence diagram. Enrolment in CCDB Module.

- A method to disenroll:

DELETE `/companionDB/disenroll/{hash}`

This endpoint will delete all data associated with the user along with its public/private keys.

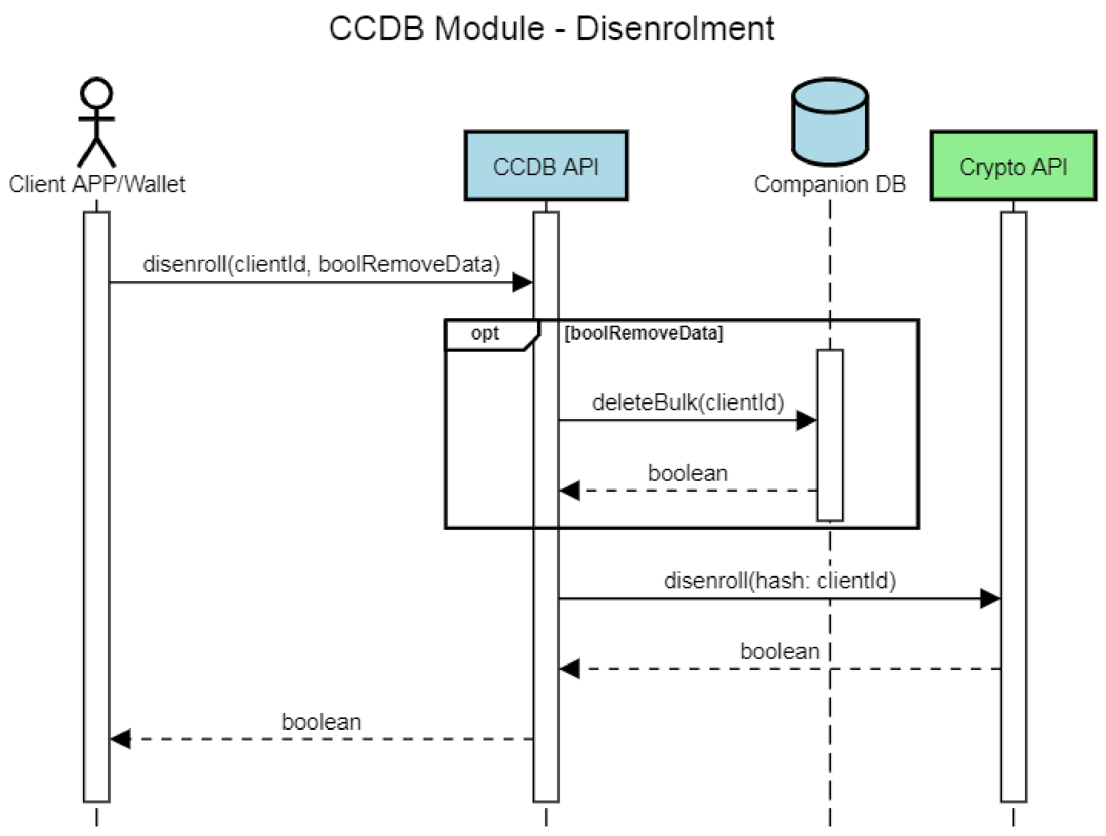
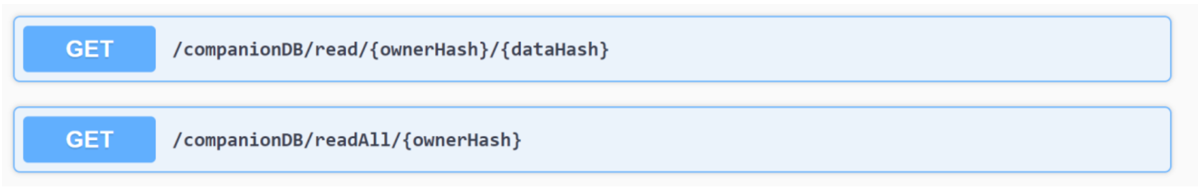


Figure 26. Sequence diagram. Disenrollment in CCDB Module.

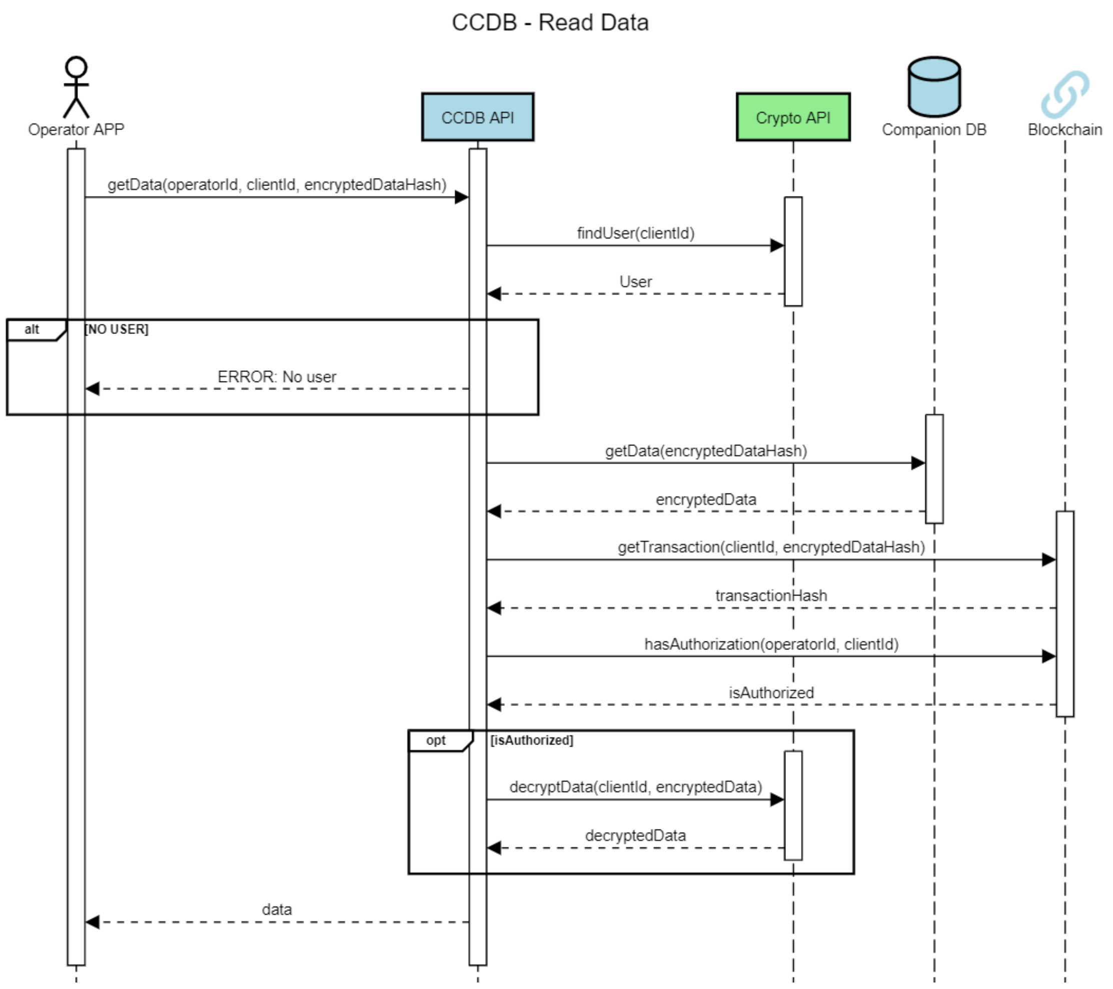
- A method to read data:



GET /companionDB/read/{ownerHash}/{dataHash}

GET /companionDB/readAll/{ownerHash}

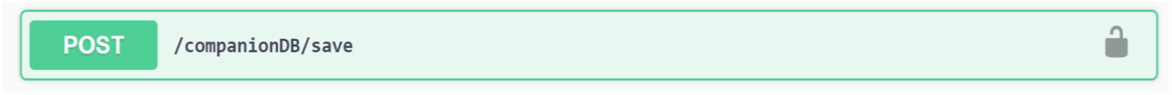
These endpoints will let an owner or an authorized user to read the encrypted data.



CCDB - Read Data
Operator APP
CCDB API
Crypto API
Companion DB
Blockchain
getData(operatorId, clientId, encryptedDataHash)
findUser(clientId)
User
alt
[NO USER]
ERROR: No user
getData(encryptedDataHash)
encryptedData
getTransaction(clientId, encryptedDataHash)
transactionHash
hasAuthorization(operatorId, clientId)
isAuthorized
opt
[isAuthorized]
decryptData(clientId, encryptedData)
decryptedData
data

Figure 27. Sequence diagram. Read data in CCDB Module.

- A method to save data:



POST /companionDB/save

This endpoint will let an owner to save encrypted data.

This method has evolved in order to be more compliant with the GDPR.

Before the user should call the blockchain outside and provide a hash in order to link the information between the CCDB and the blockchain. Now, the companion database will take care of the encryption and the hash generation, making it more secure and having a hash in the blockchain that will be generated from encrypted sensitive data, not the raw sensitive data.

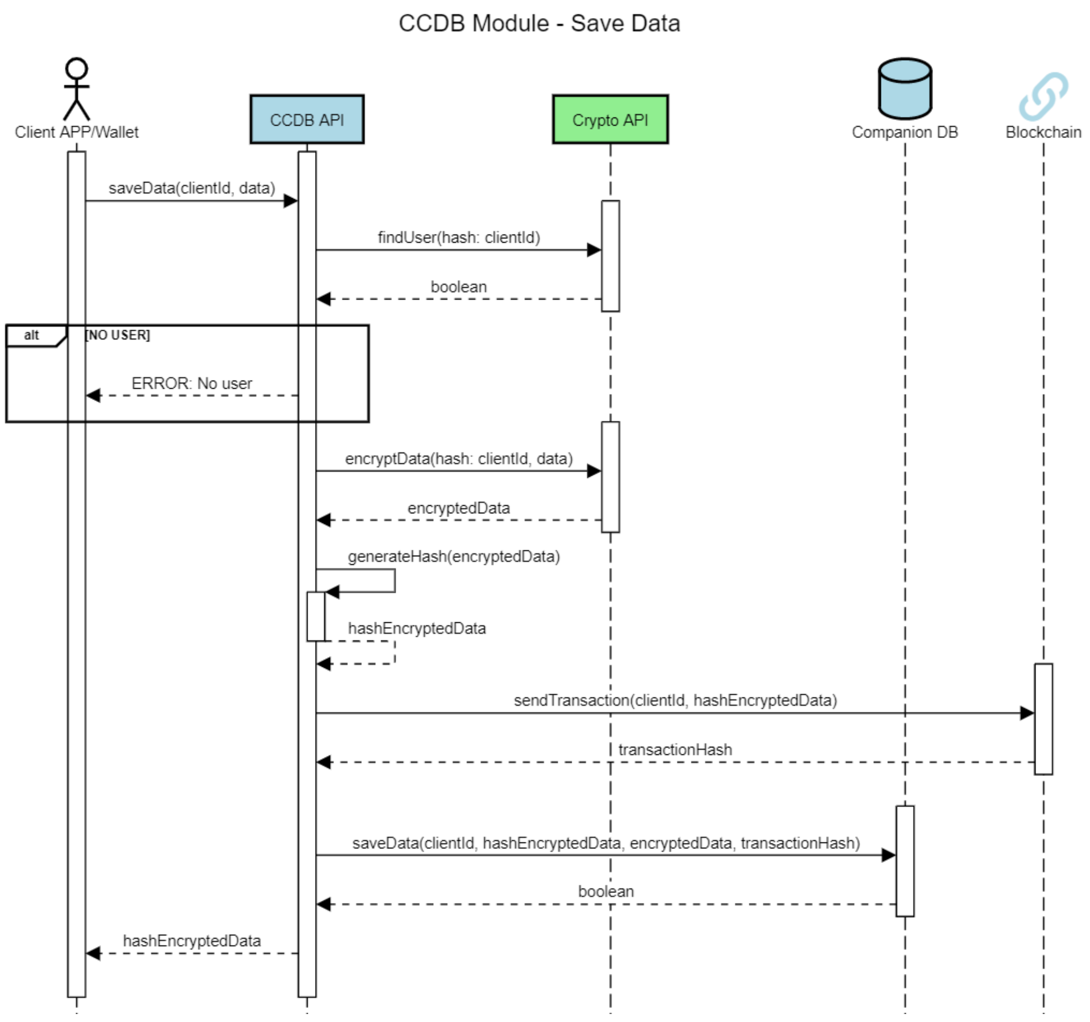
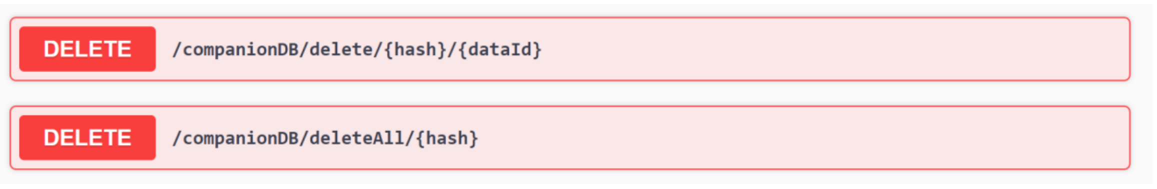


Figure 28. Sequence Diagram. Save data in CCDB Module.

- A method to delete data:



DELETE `/companionDB/delete/{hash}/{dataId}`

DELETE `/companionDB/deleteAll/{hash}`

These endpoints will let the owner of the data to delete it.

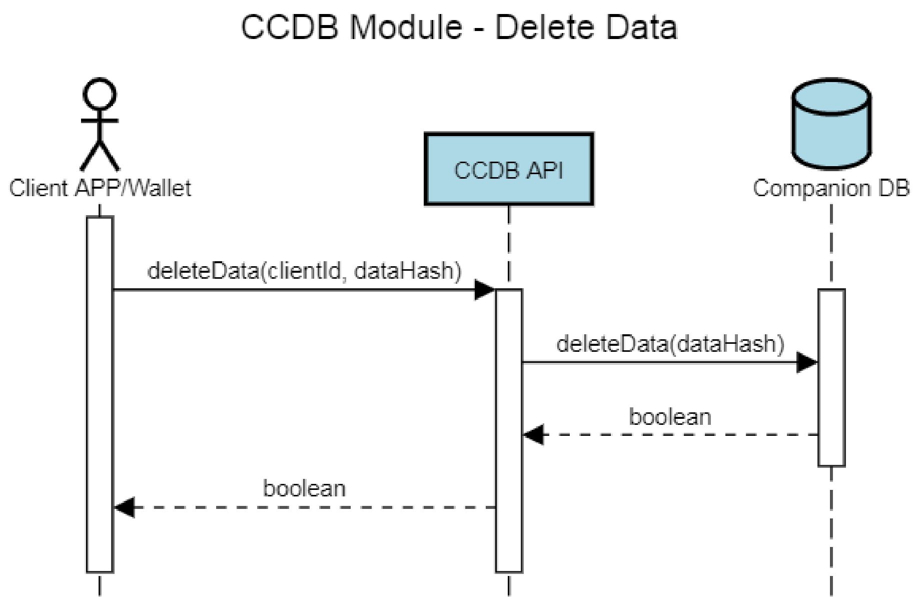
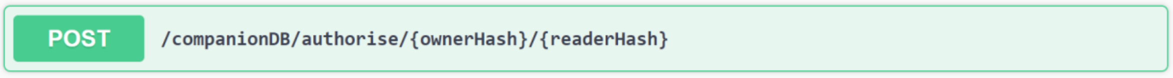


Figure 29. Sequence Diagram. Delete data in CCDB Module.

- A method to authorize a user:


POST /companionDB/authorise/{ownerHash}/{readerHash}

This endpoint will let an owner to authorize another user to decrypt its data.

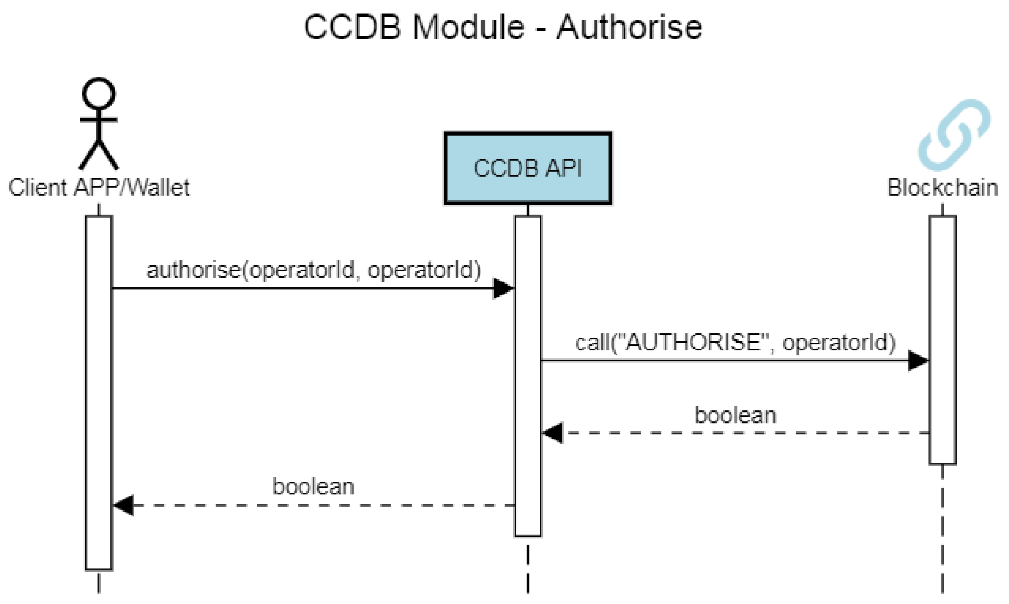
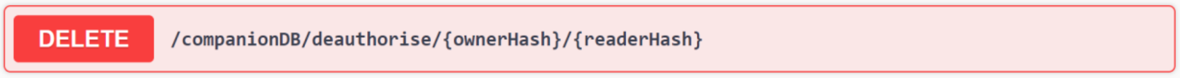


Figure 30. Sequence diagram. Authorize in CCDB Module.

- A method to de-authorize a user:


DELETE /companionDB/deauthorise/{ownerHash}/{readerHash}

This endpoint will let an owner to de-authorize another user to decrypt its data.

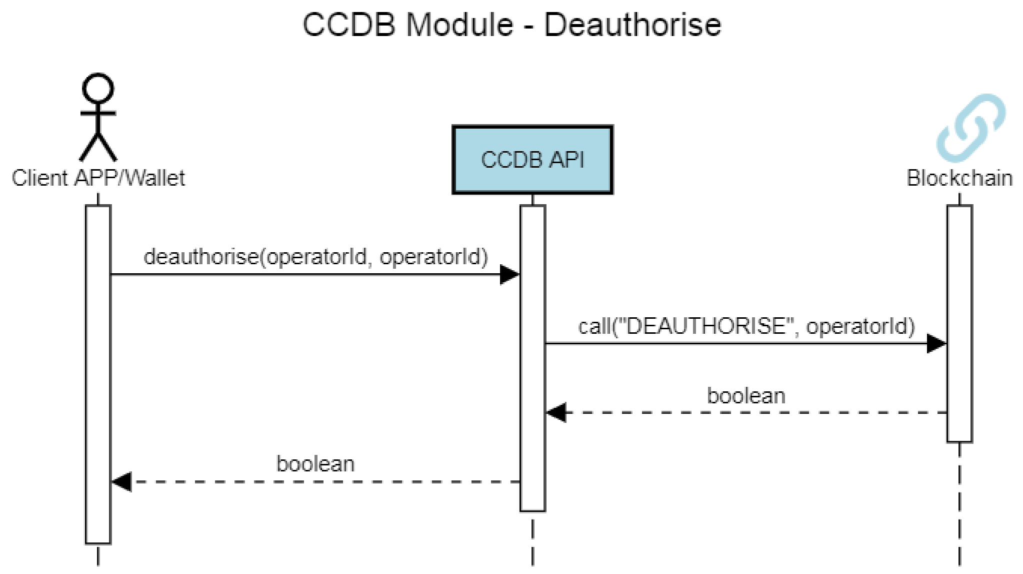

CCDB Module - Deauthorise
Client APP/Wallet
CCDB API
Blockchain
deauthorise(operatorId, operatorId)
call("DEAUTHORISE", operatorId)
boolean
boolean

Figure 31. Sequence diagram. Remove authorization in CCDB Module.

- A method to request authorization to a user:

POST `/companionDB/requestAuthorisation/{dataHash}/{readerHash}`

This endpoint will let an external user to request authorization to access data to the owner.

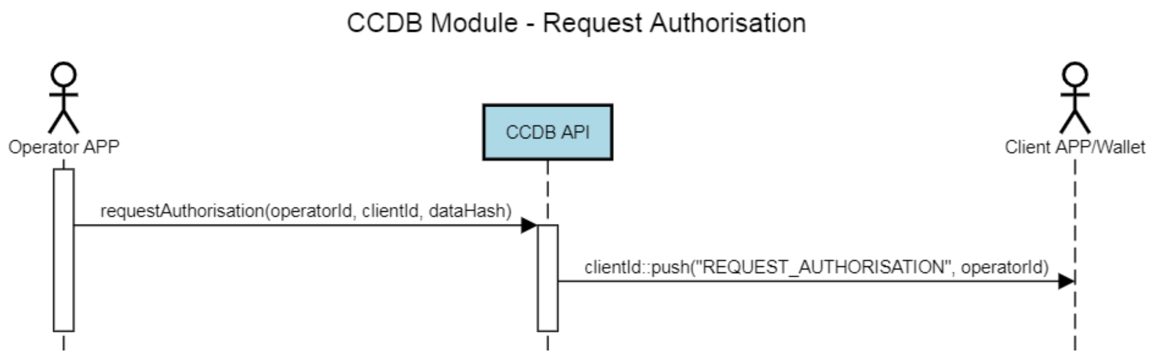

CCDB Module - Request Authorisation
Operator APP
CCDB API
Client APP/Wallet
requestAuthorisation(operatorId, clientId, dataHash)
clientId::push("REQUEST_AUTHORISATION", operatorId)

Figure 32. Sequence diagram. Request authorization in CCDB Module.

Package Information & Installation Instructions

Required Tools and dependencies

Following a list of the required tools and dependencies of the modules:

- Docker (it comes with, Kubernetes, Kitematic, Docker Manager, …)
- Docker Quickstart Terminal
- Docker Toolbox (for Windows Users only)
- Mongo DB
- Oracle VM Virtualbox
- Nodejs v10.17.0
- NPM 6.11.3
- Git

The version indicated in some tools/dependencies are important for compatibility. If the versions are not these, it might raise some problems.

In order to ease the installation, proceed with the established order in the list.

Install Docker

The installation can be found in the Docker's webpage https://docs.docker.com/v17.09/engine/installation/, but following there is a list of the main steps and commands for Windows and Ubuntu.

Windows 10: (Source: https://docs.docker.com/v17.09/docker-for-windows/install/#start-docker-for-windows)

In order to install Docker, we have to follow the next steps:

1. Download Docker from the Docker Hub:
   https://download.docker.com/win/stable/Docker%20for%20Windows%20Installer.exe
2. Double-click Docker for Windows Installer.exe to run the installer.
3. Follow the instructions on the installation wizard to accept the license, authorize the installer, and proceed with the install.
   When prompted, authorize the Docker Desktop Installer with your system password during the install process. Privileged access is needed to install networking components, links to the Docker apps, and manage the Hyper-V VMs.
4. Click Finish on the setup complete dialog and launch the Docker Desktop application.
5. Docker will not start automatically. To start it, search for Docker, select the app in the search results, and click it (or hit Return).

Ubuntu Xenial 16.04 LTS: (Source: https://docs.docker.com/v17.09/engine/installation/linux/docker-ce/ubuntu/)

A. Set up the repository

1. Update the `apt` package index:

```
sudo apt-get update
```

2. Install packages to allow apt to use a repository over HTTPS:

```
sudo apt-get install apt-transport-https ca-certificates curl software-
properties-common
```

3. Add Docker’s official GPG key:

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add
-
```

Verify that you now have the key with the fingerprint 9DC8 5822 9FC7 DD38 854A E2D8 8D81 803C 0EBF CD88, by searching for the last 8 characters of the fingerprint.

```
sudo apt-key fingerprint 0EBFCD88

pub   4096R/0EBFCD88 2017-02-22
      Key fingerprint = 9DC8 5822 9FC7 DD38 854A  E2D8 8D81 803C 0EBF CD88
uid                  Docker Release (CE deb) <docker@docker.com>
sub   4096R/F273FCD8 2017-02-22
```

4. Use the following command to set up the stable repository.

```
sudo add-apt-repository \
   "deb [arch=amd64] https://download.docker.com/linux/ubuntu \
   $(lsb_release -cs) \
   stable"
```

B. Install Docker

1. Install the ***linux-image-extra*** kernel package:

```
sudo apt-get update -y && sudo apt-get install -y linux-image-extra-
$(uname -r)
```

2. Install Docker:

```
sudo apt-get install docker-engine -y
```

3. Start Docker:

```
ludo service docker start
```

4. Verify Docker:

```
sudo docker run hello-world
```

Install Mongo DB

As the Docker was installed in the section above, the installation of the Mongo DB will be as easy as executing the following command for any operating system:

```
docker run -p 27017:27017 --name mongo-nest -d mongo:4
```

Install Node.js and npm

The installation of these two tools is done together and it can be found in the Node.js webpage https://nodejs.org/en/ , but following there are a list of the main steps and commands for Windows and Ubuntu.

Windows 10:

1. Download the binary from: https://nodejs.org/dist/v10.17.0/
2. Install the *msi* or *exe* file by double-click.
3. Follow the instructions.
4. Check that Node.js is installed with the command:

```
node -v
```

5. Check that npm is installed with the command:

```
npm -v
```

Ubuntu Xenial 16.04 LTS: (check https://github.com/nodesource/distributions/blob/master/README.md#debinstall for further information)

1. Add the NodeSource package signing key:

```
curl -sL https://deb.nodesource.com/setup_10.x | sudo -E bash -
```

2. Install Node.js

```
sudo apt-get install -y nodejs
```

Install Git

Extracted from https://git-scm.com/book/en/v2/Getting-Started-Installing-Git

Windows 10:

1. Download and install it from https://git-scm.com/download/win

Ubuntu Xenial 16.04 LTS:

1. Execute the following command:

```
sudo apt install git-all
```

Download and Run Demonstrator

In order to download and run the demonstrator the following steps have to be performed:

1. Clone the GitHub project in a selected folder:

```
git clone https://github.com/jordiescudero/wl-bc-cs/
```

2. Execute the command from the installation of Mongo DB:

```
docker start mongo-nest
```

3. Go to the root of the project:

```
npm run start
```

4. The base URI for all the interface will be: http://localhost:3000/api/
5. The Swagger UI can be found at: http://localhost:3000/api/docs/#/

User Manual

As stated in the section “Companion DB Module” the APIs that will be published and used will be the Authentication API and the Data Management API.

The Swagger UI provides enough information to let the developer know how to use this API, but some examples were put together as a starting point.

- **GET /*companionDB*/read/{dataId}**

```
curl -X GET
"http://localhost:3000/api/companionDB/read/hashhashhashhashhash" -H
"accept: application/json"
```

- **POST /*companionDB*/save**

```
curl -X POST "http://localhost:3000/api/companionDB/save" -H "accept:
application/json" -H "Content-Type: application/json" -d "{ \"name\":
\"Name\", \"email\": \"email@email.com\", \"birht_date\": \"01/01/2001\",
\"gender\": \"Other\", \"city\": \"Barcelona\"}"
```

The json beautified:

```
{
  "name": "Name",
  "email": "email@email.com",
  "birht_date": "01/01/2001",
  "gender": "Other",
  "city": "Barcelona"
}
```

- **DELETE /*companionDB*/delete/{dataId}**

```
curl -X DELETE "http://localhost:3000/api/companionDB/delete/hashhashhash"
-H "accept: application/json"
```

- **POST /*companionDB*/authorise/{hash}**

```
curl -X POST
"http://localhost:3000/api/companionDB/authorise/hashhashhash" -H "accept:
application/json" -H "Content-Type: application/json" -d "{ \"authHash\":
\"authorisedHash\"}"
```

The json beautified:

```
{
  "authHash": "authorisedHash"
}
```

Licensing

Licensing for all the components/software used:

- Docker is under Apache License 2.0 (https://www.apache.org/licenses/LICENSE-2.0) form more detail go to https://www.docker.com/legal/components-licenses.
- Mongo DB is under Server Side Public License (https://www.mongodb.com/licensing/server-side-public-license)
- NPM is under Artistic License 2.0 (https://www.npmjs.com/policies/npm-license)
- Git is under GNU General Public License version 2.0 (https://opensource.org/licenses/GPL-2.0)
- Oracle VirtualBox is under GNU General Public License, version 2 (https://www.gnu.org/licenses/old-licenses/gpl-2.0.html)
- Software developed is under MIT (https://github.com/jordiescudero/wl-bc-cs/blob/master/LICENSE)

2.3 Interactions with other FGs

The following figure presents all the interactions of the Secured & Trusted Storage FG with other components of the M-Sec solution. The Annex also presents a more "global" view, with the positioning of the FG within the whole M-Sec system.

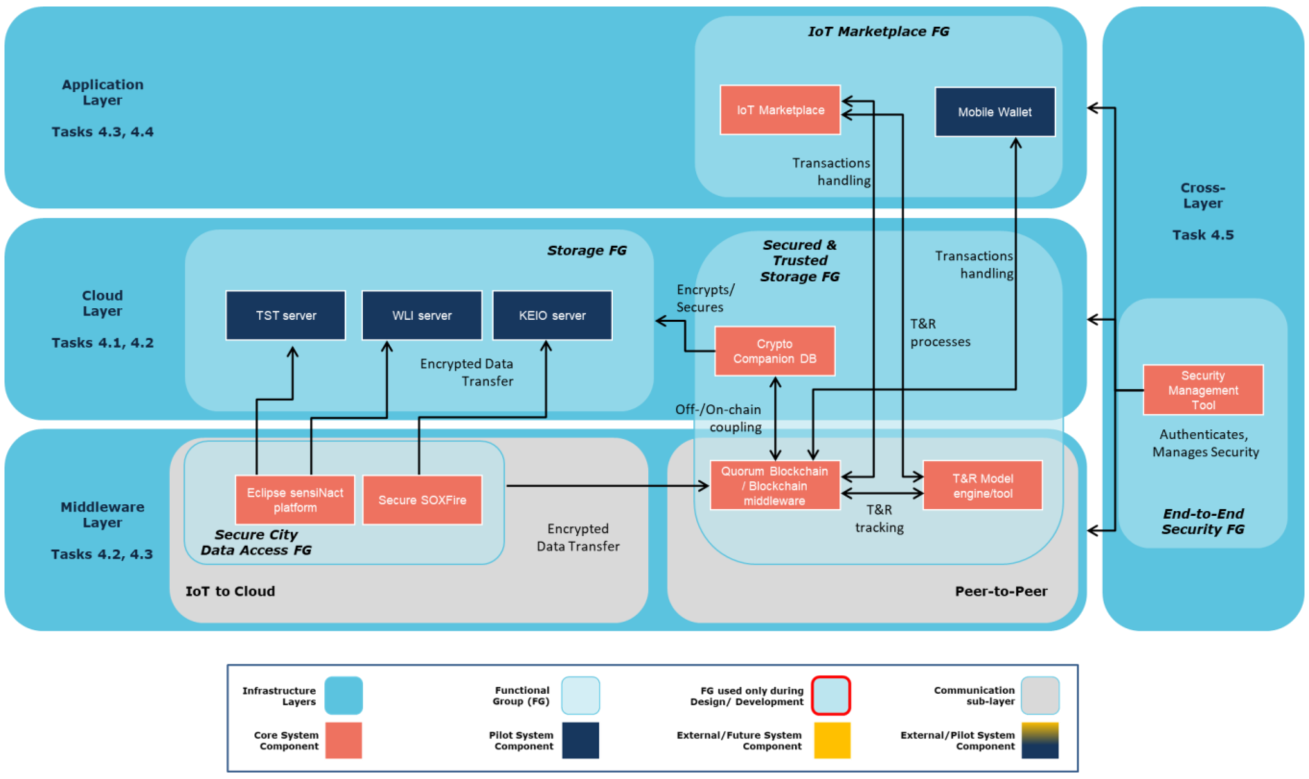


Figure 33. Interaction of the Secured & Trusted Storage FG with other FGs

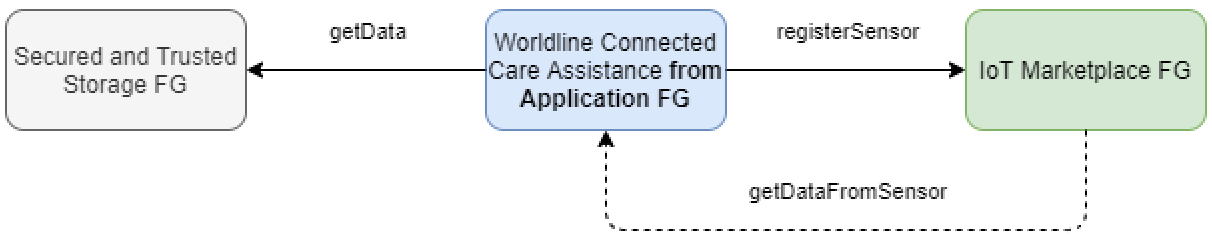
Interaction with IoT Marketplace FG

The Secured and Trusted Storage FG does not generate data from itself, it is used as a storage system to store sensitive data, and the same happens with the IoT Marketplace FG, it is used to store/gather sets of anonymized data in order to sell it.

So, both FGs having the need to be fed from external applications implies that the integration between them is not direct, and they need another component to link them together.

Although, taking as an example the Use Case 2, we can see that the Worldline Connected Care Assistance application save the activities of all IoT sensors into the Secured & Trusted Storage FG, then the application registers all sensors to the IoT Marketplace FG providing a link to retrieve data.

In the following figure, the flow of the data can be seen, showing the integration of both FGs having as a communication component the Worldline Connected Care Assistance.



Secured and Trusted Storage FG
getData
Worldline Connected Care Assistance from Application FG
registerSensor
IoT Marketplace FG
getDataFromSensor

Figure 34. Example of data flow between Secured & Trusted Storage FG and IoT Marketplace FG

Interaction with Security city data Access FG

As mentioned before in the section ‘Interaction with IoT Marketplace FG’, the ‘Secured & Trusted Storage FG’ does not generate data from itself, it is used as a storage system to store sensitive data.

The Security city data Access FG is used to gather information from IoT sensors, do some modifications, standardizations and send it elsewhere.

The data sent by the Security city data Access FG could be sent directly to the Secured & Trusted Storage FG, but as the Security city data Access FG does not have as a purpose to act as a data manager, some other application have to be the link between them.

Taking as an example from Use Case 2, the Worldline Connected Care Assistance application save the activities of all IoT sensors coming from the Security city data Access FG into the Secured and Trusted Storage FG.

In the following figure the integration from Devices FG to Secured & Trusted Storage FG can be seen.

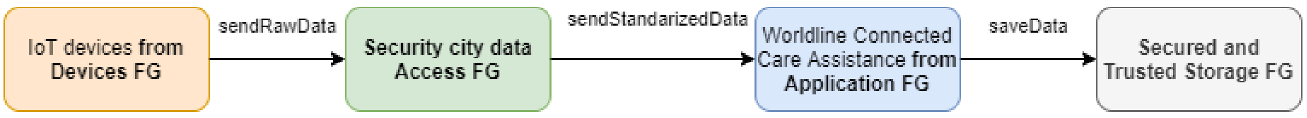

IoT devices from Devices FG
sendRawData
Security city data Access FG
sendStandarizedData
Worldline Connected Care Assistance from Application FG
saveData
Secured and Trusted Storage FG

Figure 35.Example of data flow between Secured & Trusted Storage FG and Security city data Access FG

Another point of integration with Security city data Access FG was implemented to facilitate different use cases (mainly Use Case 3, Use Case 4 and Use Case 5) and end users of the pilots. To this direction, a new component was developed, namely “SOXFire – Blockchain – IoT Marketplace Bridge” allowing registration of sensors, purchase/exchange of data and visualization of data. An overview of this component is shown in the following figure.

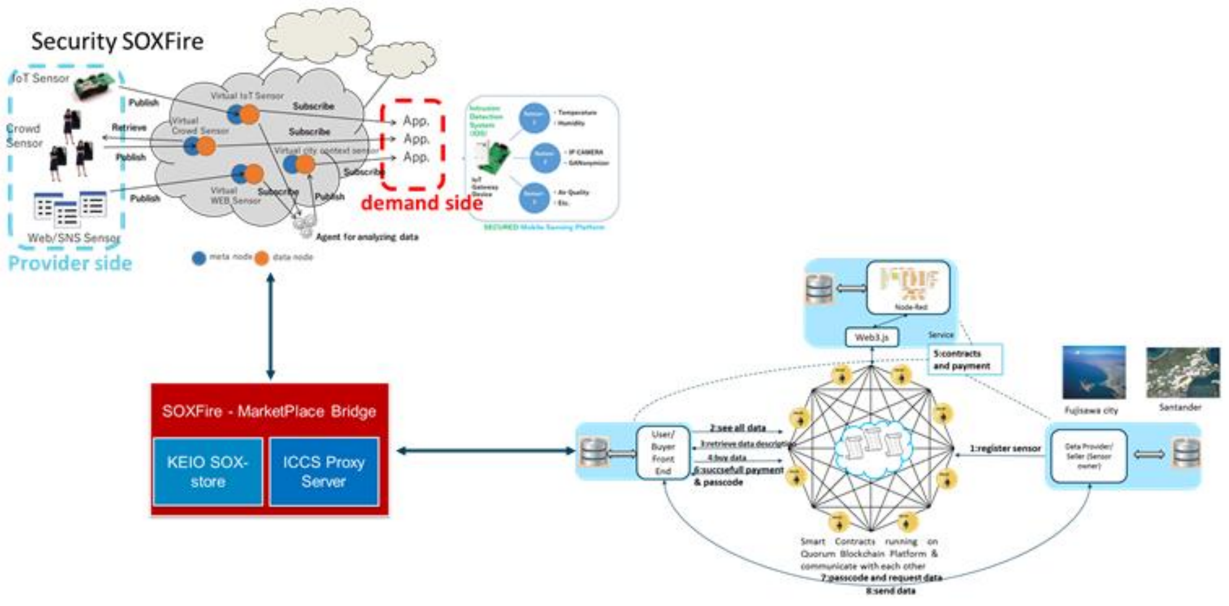
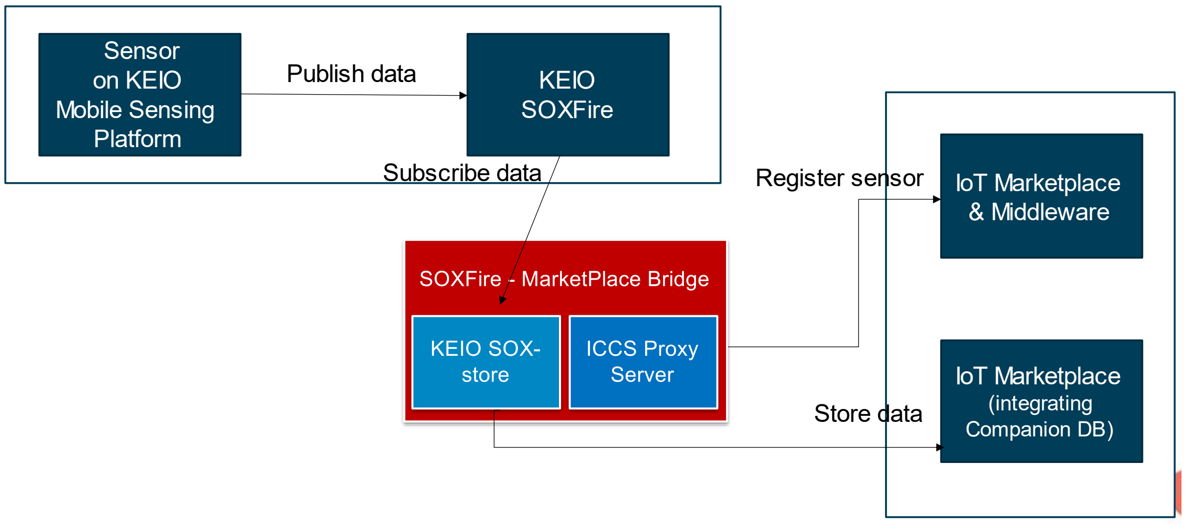


Figure 36. Overview of SOXFire – Blockchain - IoT Marketplace Bridge

A more technical figure is provided below, displaying details of this integration.



Sensor on KEIO Mobile Sensing Platform
Publish data
KEIO SOXFire
Subscribe data
Register sensor
IoT Marketplace & Middleware
SOXFire - MarketPlace Bridge
KEIO SOX-store
ICCS Proxy Server
Store data
IoT Marketplace (integrating Companion DB)

Figure 37. Technical overview of SOXFire – Blockchain – IoT Marketplace Bridge

Interaction with Application FG

Following the example of the Use Case 2 and the application Worldline Connected Care Assistance, the Secured & Trusted Storage FG is fully integrated with the Application FG. The application uses this FG to store all the information of the activities from the IoT devices. To have a clear picture, the information from the IoT devices (Devices FG) is sent to the broker of the company (Caburn); this information is gathered, filtered and standardized by the Eclipse sensiNact Platform (Secure City Data Access FG); and finally the application saves it to the Secured & Trusted Storage FG.

In the following figure the flow of all the data and the integrations can be seen.

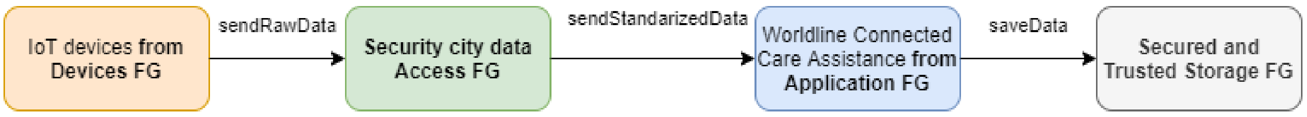

IoT devices from Devices FG
sendRawData
Security city data Access FG
sendStandarizedData
Worldline Connected Care Assistance from Application FG
saveData
Secured and Trusted Storage FG

Figure 38. Example of data flow between Application FG and Secured & Trusted Storage FG

Interaction with End-to-End Security FG

The integration between these two functional groups will be to make use for the Authentication of the End-to-End Security FG in order to be able to use the same user for all the FGs.

Investigations to perform the integrations have been made. The goal is to integrate the End-to-End security at the level of the Functional Group API, so it will not be needed to have authentication in each component inside, easing the implementation and integration of inner components.

As described in detail in D4.8, the integration of IoT Marketplace FG and End-to-End Security FG enable clients of the marketplace to be authenticated either as the owner of devices, owner of data, or simply a consumer. Middleware and Blockchain components facilitate this integration and indirectly support it, leaving no potential for unauthorized access and enhancing overall security.

2.4 Common API

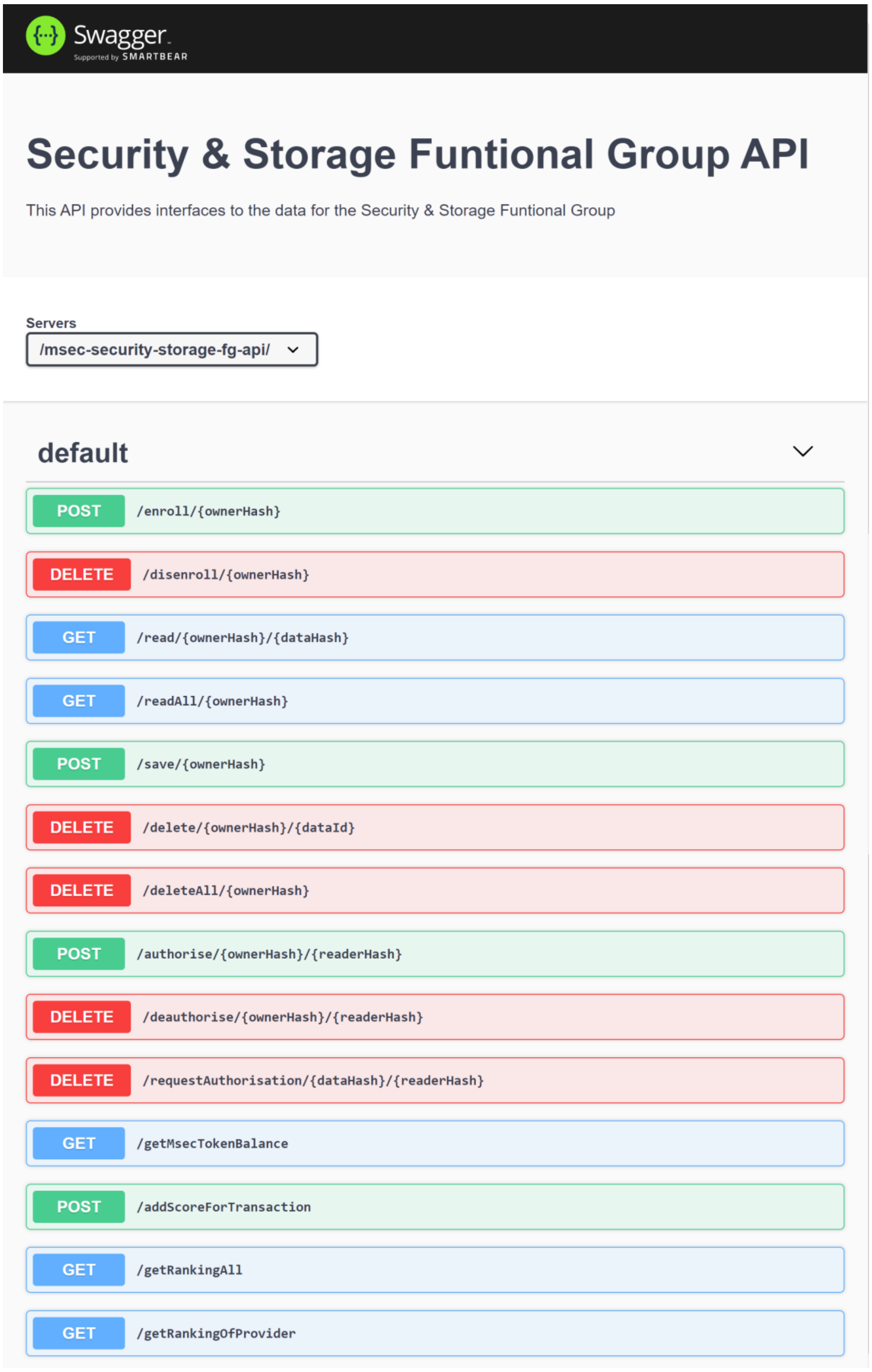
In order to allow the communication and integration with other FGs, services, assets, several methods were developed. These methods are exposed via a RESTful API, while respective clients have been developed to facilitate the integration process and documentation with examples and indicative architecture figures and snippets.

To this direction, a common API was developed and all relevant endpoints have been added in a common URL in order to ease the use of it.

After a series of steps, we ended in some considerations for a method to be useful on the common API:

- The use of the method in a high level usage for a final user.
    - For example, using the save method/endpoint from the Crypto Companion Database will save data and link it with the blockchain.
- Forbid to bypass steps.
    - It is not allowed to send information directly to the blockchain, bypassing the CCDB. Saving to the blockchain directly will bypass the steps to encrypt and save the data into the Crypto Companion Database and will not be compliant with the GDPR.
- Only one method/endpoint can do one action/functionality.
    - If two methods/endpoints of save are provided, one from CCDB and one from blockchain, it will confuse the user, and one of them will bypass some needed steps.

To clarify, if the user wants to use the Secured and Trusted Storage FG, it will not use calls directly to the blockchain, because with the changes made in order to be compliant with the GDPR, the interaction with the blockchain will be managed by the Crypto Companion Database, meaning that reading or saving information, creating an account and other more specific endpoints are not shown in the common API. In the following figure, the summary of Security & Storage Functional Group API is presented.



Swagger
Supported by SMARTBEAR

Security & Storage Funtional Group API

This API provides interfaces to the data for the Security & Storage Funtional Group

Servers
/msec-security-storage-fg-api/

default

```
POST    /enroll/{ownerHash}
DELETE  /disenroll/{ownerHash}
GET     /read/{ownerHash}/{dataHash}
GET     /readAll/{ownerHash}
POST    /save/{ownerHash}
DELETE  /delete/{ownerHash}/{dataId}
DELETE  /deleteAll/{ownerHash}
POST    /authorise/{ownerHash}/{readerHash}
DELETE  /deauthorise/{ownerHash}/{readerHash}
DELETE  /requestAuthorisation/{dataHash}/{readerHash}
GET     /getMsecTokenBalance
POST    /addScoreForTransaction
GET     /getRankingAll
GET     /getRankingOfProvider
```

Figure 39. Security & Storage Functional Group API

3. Conclusion

This deliverable D4.6 is the final version of the deliverables related to the task T4.3. Following the original M-Sec architecture already introduced and discussed in previous reports such as Deliverable 3.3, it can be identified that the Secured and Trusted Storage FG covered in this document is one of the core FGs required to facilitate the functionalities of several of the UCs covered by the project.

The components of the FG, its functionalities, as well as its interactions and integrations with other components as parts of a unified architecture is also documented in detail. Similarly, enough information is provided related to the usage of the available tools, as well as of a common API that exposes all the capabilities of the FG

Table 7: Demonstrators and their correlation with Use Case Pilots

Demonstrator	Type	Use Case Pilots	Purpose
Blockchain Framework	Software-based solution (Solidity)	Use Case 2	Support encryption and security features by interacting with other components
Middleware Services	Software-based solution (Javascript)	Use Cases 2,5	Support end users and other components
Trust & Reputation Management	Software-based solution (Javascript)	Use Case 5	Evaluate transaction and media assets
Crypto Companion Database	Software-based solution (Javascript)	Use Cases 2	Store encrypted sensitive data with a link to the Blockchain.

Annex

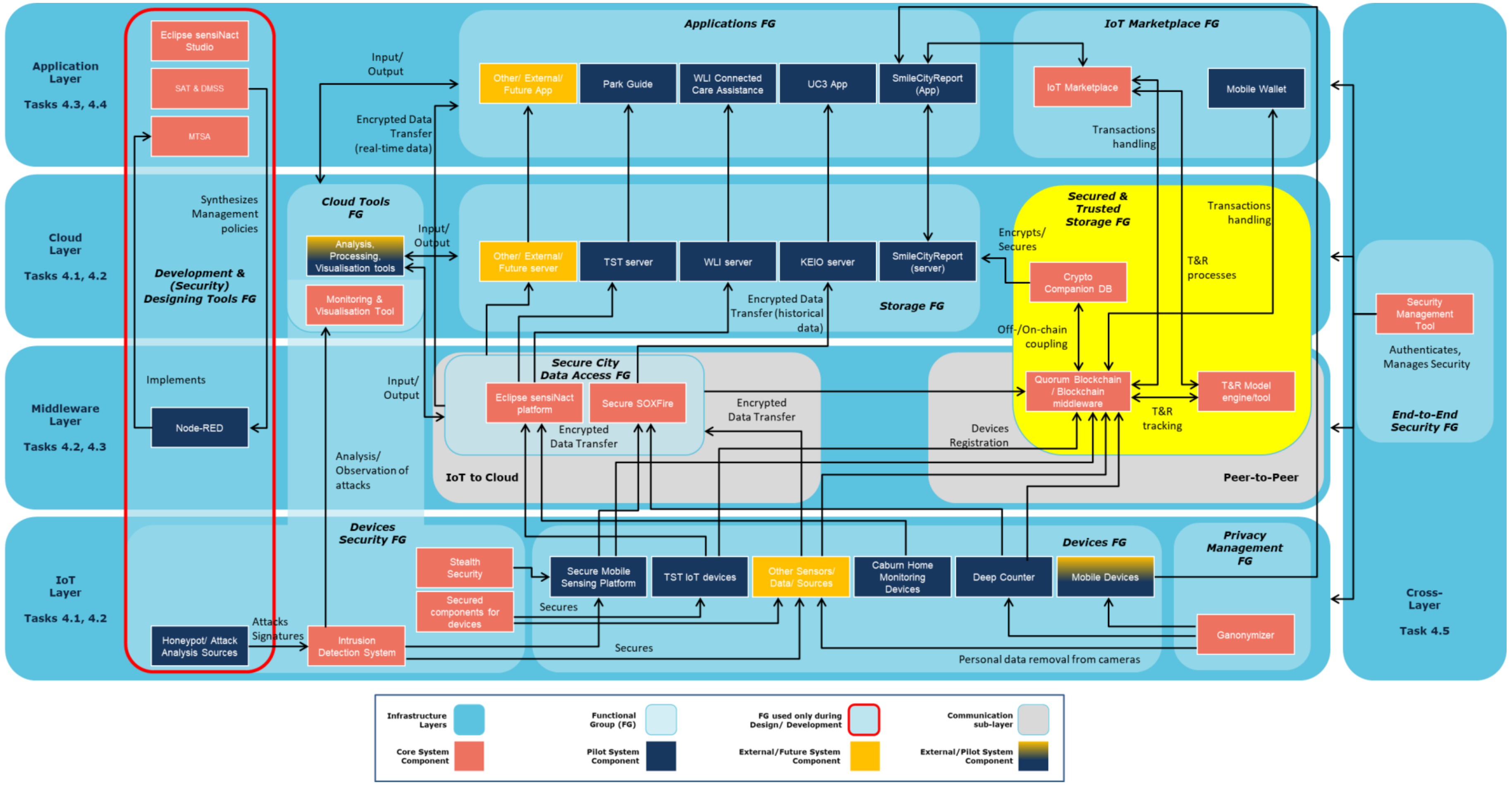


Figure 40. The M-Sec Architecture (T4.3 FG in yellow)