



Multi-layered Security Technologies

for hyper-connected
smart cities

Cookbook

December 2020



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	Cookbook
Work Package	5
Submission date	December 2020
Deliverable lead	TST / KEIO
Authors	Jin Nakazawa (KEIO), Arturo Medela (TST)
Internal reviewer	F6S, WU, CEA
Dissemination Level	Public
Type of deliverable	
Version history	<ul style="list-style-type: none">- v00, 29/March/2020, Jin Nakazawa, Table of Contents, Full Draft- v01, June/2020, Arturo Medela, Initial contribution- v02, December/2020, Arturo Medela, Contents adapted, editing- v03, December/2020, final version

Worldline



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).





Table of Contents

Table of Contents	3
List of Tables	5
List of Figures.....	5
Glossary	9
Introduction.....	11
IoT Security	13
1.1 IoT devices with Increased Security	13
General Description	13
Components.....	14
Installation Instructions	21
1.2 Intrusion Detection System (IDS) for IoT devices	24
General Description of the Prototype	24
Components.....	24
Installation Instructions	26
Cloud and Data Level Security	27
1.3 Hardware based encryption	27
General Description	27
Installation Instructions	28
1.4 Software-based Threat Monitoring.....	31
General Description	31
Installation Instructions	32
1.5 Privacy Management Tool.....	34
General Description	34
Installation Instructions	36
P2P Level Security and M-Sec Blockchains.....	37
1.6 Blockchain Framework and Middleware Services	37
General Description	37
Components.....	38
Installation Instructions	52
1.7 IoT Marketplace.....	53





General Description	53
Components.....	55
Installation Instructions	57
1.8 Trust & Reputation Management.....	59
General Description	59
Components.....	60
Application Level Security	66
1.9 Crypto Companion Database (CCDB)	66
General Description	66
Components.....	67
Installation Instructions	78
1.10 Demonstration 2 – Security-analysis-tool	81
General Description	81
Components.....	82
Installation Instructions	87
End-to-End Security	89
1.11 Security Manager.....	89
General Description	89
Components.....	90
Installation Instructions	91
1.12 End-to-end Encryption Middleware for SOXFire.....	96
General Description of the Prototype	96
Components.....	96
Installation Instructions	97
1.13 sensiNact - Secured IoT Middleware.....	99
General Description	99
Components.....	100
Installation Instructions	102
Conclusion	106





List of Tables

Table 1: Environmental monitoring devices data frame	15
Table 2: Crowd counter data frame	18
Table 3: Connection layout between the MCU board and the TPM daughter-board.....	29
Table 4: Overview of M-Sec Token's functions	44
Table 5: Detailed presentation of functions and events of M-Sec Token	44
Table 6: Sensors Smart Contract details.....	46
Table 7: Demonstrators and its correlation with Use Case Pilots	106
Table 8: Demonstrators and its correlation with Use Case Pilots	107

List of Figures

Figure 1: Overall M-Sec topology	11
Figure 2: NB-IoT modules BC95 (left) and BC68 (right)	14
Figure 3: Environmental monitoring device circuitry	16
Figure 4: Environmental monitoring device appearance	16
Figure 5: Crowd counter devices circuitry	17
Figure 6: People counter IoT device appearance	18
Figure 7: Development board for the STSAFE-TPM	19
Figure 8: Measured boot process	20
Figure 9: Raspberry PI with the STPM4Raspi extension in White	21
Figure 10: External wiring for microprocessor developments	21
Figure 11: Secured Mobile Sensing Platform	24
Figure 12: IoT Gateway Device	25
Figure 13: Scheme for the hardware-encryption demonstrator.....	28
Figure 14: Location of the reset button on the Nucleo-L476RG board.....	29
Figure 15: Visualization Process	31
Figure 16: Client Dashboard - Alerts Overview	33
Figure 17: Client Dashboard - Events Overview	33





Figure 18: GANonymizer Test Results	34
Figure 19: GANonymizer Architecture	35
Figure 20: Representation of the architecture of our prototype based on Hyperledger	38
Figure 21. Hyperledger Explorer giving details about the underlying permissioned blockchain.....	38
Figure 22. Ganache Explorer allows us to examine all blocks, transactions, addresses and their balances	39
Figure 23: Details about blocks, transactions, addresses and smart contracts	40
Figure 24: The Trust Continuum	41
Figure 25. Alastria in the Trust Continuum	42
Figure 26. Item Manager Smart Contract.....	45
Figure 27. Sensors Smart Contract	46
Figure 28. Overview of the KYC process for the M-Sec Platform.....	49
Figure 29. Overview of the M-Sec IoT Marketplace and its components	53
Figure 30. Node-Red Flow for the simulation of IoT sensor data	55
Figure 31. Graphical User Interface enabling searching of sensors in the smart contracts running on blockchain.....	56
Figure 32. Graphical User Interface of the returned results after the query to the smart contract.....	57
Figure 33. Graphical User Interface of the Explorer	57
Figure 34: General steps followed in T&R models.	60
Figure 35: Calculation of the Trust Index of an actor	63
Figure 36: TRMSim-WSN.	64
Figure 37: Normal Network Comparison.....	65
Figure 38: Access to distributed data.	66
Figure 39. Crypto Companion Database Module components.	67
Figure 40. Sequence diagram. Enrolment in Crypto Module.	68
Figure 41. Sequence diagram. Data encryption in Crypto Module.	69
Figure 42. Sequence diagram. Data decryption in Crypto Module.	70
Figure 43. Sequence diagram. Disenrolment in Crypto Module.	71
Figure 44. Authentication API in Crypto Companion Database Module.....	72
Figure 45. Sequence diagram. Enrolment in CCDB Module.....	73
Figure 46. Sequence diagram. Disenrollment in CCDB Module	73
Figure 47. Sequence diagram. Read data in CCDB Module.....	74
Figure 48. Sequence Diagram. Save data in CCDB Module.	75



Figure 49. Sequence Diagram. Delete data in CCDB Module.	76
Figure 50. Sequence diagram. Authorize in CCDB Module.	76
Figure 51. Sequence diagram. Deauthorise in CCDB Module.	77
Figure 52. Sequence diagram. Request authorization in CCDB Module.	77
Figure 53. Authentication API in Crypto Companion Database Module.	79
Figure 54. Management API in Crypto Companion Database Module.	80
Figure 55. Screenshot of the Security analysis tool.	81
Figure 56. Conceptual model of software security knowledge.	83
Figure 57. Visualized software security knowledge base.	83
Figure 58. Top page of Software Security Knowledge Base system	84
Figure 59. Threat Type Screen	84
Figure 60. Threat Type registration form	85
Figure 61. Registered Identify Spoofing in Attack Pattern knowledge	85
Figure 62. Spoofing information in Threat Type Screen.	86
Figure 63. Edit form in Threat Type screen	86
Figure 64. Association of Treat Type knowledge "Spoofing" and Attack pattern knowledge "Identity Spoofing"	87
Figure 65. Stack for the M-Sec "Security Manager" with components described in blue	89
Figure 66. The first level of the directory service for the security manager	90
Figure 67. Users management page.	92
Figure 68. User edition	92
Figure 69. web user page for requesting a new certificate	94
Figure 70. Prompt to insert the CSR while requesting a new certificate	94
Figure 71. Publisher/Subscriber pattern with potential MITM in yellow and counter-measure in red	96
Figure 72. SOXFire setup screen	98
Figure 73. SOXFire database configuration	98
Figure 74. sensiNact service model	99
Figure 75. sensiNact service model mapping example	99
Figure 76. sensiNact gateway northbound and southbound connectivity	100
Figure 77. Secured connectivity	101
Figure 78. Sign the X bridge	101
Figure 79. Authorization code flow	102





Figure 80. Resource owner password credential flow	102
Figure 81. Login as adminTester	104
Figure 82. Login as anonymousTester	105





Glossary

AAA	Authorization, Authentication and Accounting
AES	Advanced Encryption Standard
API	Application Programming Interface
ARM	Advanced RISC Machines
ATF	ARM Trusted Firmware
CA	Certification Authority
CCDB	Crypto Companion Database
CoAP	Constrained Application Protocol
CPU	Central Processing Unit
CSR	Certificate Signing Request
CVE	Common Vulnerabilities and Exposures
DB	Database
DNS	Domain Name System
ECC	Elliptic Curve Cryptography
ECDSA	Elliptic Curve Digital Signature Algorithm
ECH	Elliptic Curve Diffie-Hellman
EVM	Extended Verification Module
GAN	Generative Adversarial Networks
GDPR	General Data Protection Regulation
GLCIC	Globally and Locally Consistent Image Completion
GPIO	General Purpose Input/Output
HAL	Hardware Abstraction Layer
HMAC	Hash-based Message Authentication Code
HSM	Hardware Security Module
HW	HardWare
I2C	Inter-Integrated Circuit
IDS	Intrusion Detection System
IETF	Internet Engineering Task Force
IMA	Integrity Measure Architecture
IoT	Internet of Things
ISO	International Organization for Standardization





JSON	Javascript Object Notation
LAN	Local Area Network
LDAP	Lightweight Directory Access Protocol
M(number)	Month (number of the month in the project)
MCU	Micro Controller Unit
MD5	Message-Digest Algorithm 5
MPU	Micro Processor Unit
MQTT	Message Queuing Telemetry Transport
NB-IoT	Narrow Band-IoT
NIST	National Institute of Standards and Technology
OP	Operating System
OPTEE	Open Portable Trusted Execution Environment
OSS	Open Source Software
PCR	Platform Configuration Register
PKI	Public Key Infrastructure
PPP	Point-to-Point Protocol
REST	Representational state transfer
RMF	Risk Management Framework
RPC	Remote procedure call
RSA	Encryption algorithm named after its inventors: Rivest, Shamir and Adleman
SASL	Simple Authentication and Security Layer
SPI	Serial Peripheral Interface
SSH	Secure Shell
SSL	Secure Sockets Layer
SW	SoftWare
TCG	Trusted Computing Group
TEE	Trusted Execution Environment
TLS	Transport Layer Security
TPM	Trusted Platform Module
UC	Use Case
WAN	Wide Area Network
WP	Work Package



Introduction

The main focus of this document is to introduce the M-Sec IoT security framework, which in turn will help to develop reliable and secure applications for the Smart City context. The goal here consists in looking for techniques, methods, and design and operating principles that minimize the risk of suffering critical vulnerabilities in a wide range of IoT devices, which could be leveraged by hackers to carry out a number of nefarious activities.

The overall M-Sec architecture is introduced and discussed in the M-Sec Whitepaper¹. This document provides a good list of the M-Sec components with their definition and ulterior implementation. In concrete, the following sections of this document are organized as follows, each of which focusing on a task shown in Figure 1: IoT security, cloud and data level security, P2P level security and blockchain, application-level security, and overall end-to-end security.

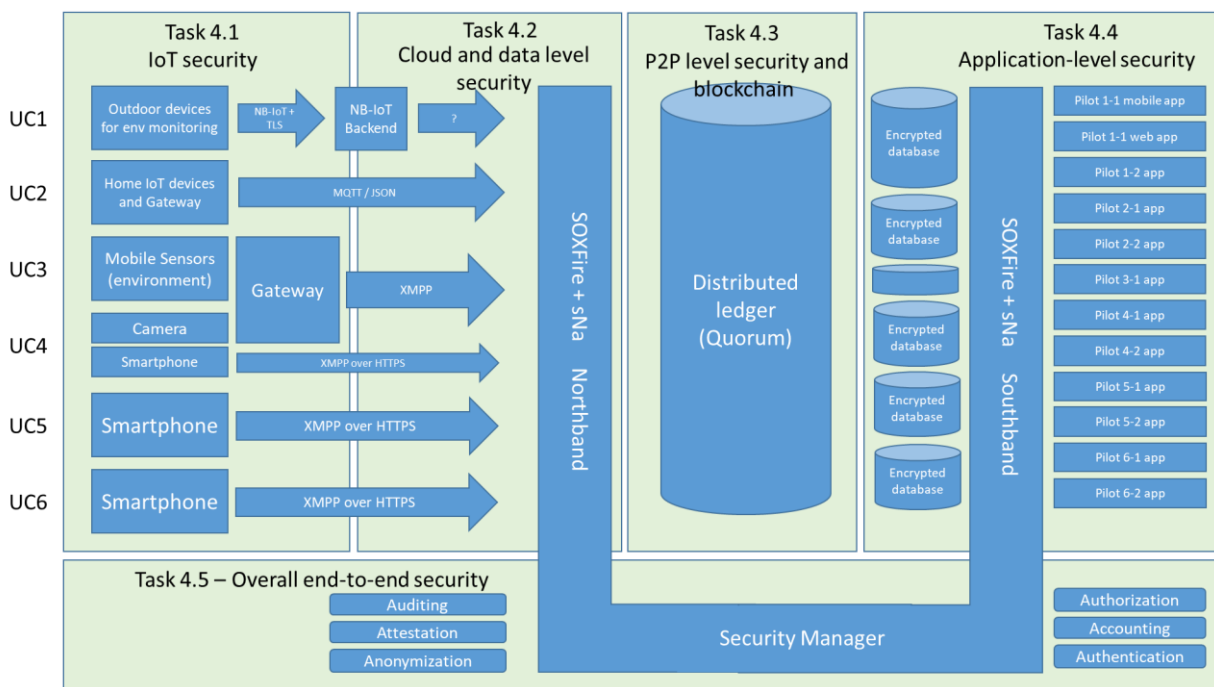


Figure 1: Overall M-Sec topology

First, the document addresses the main objectives of the M-Sec components strengthening the IoT layer, which will become one of the security layers in the overall Multi-layer Security (M-Sec) platform, providing the needed security and reliability for IoT devices as follows:

¹ https://www.msecproject.eu/wp-content/uploads/2020/10/M-Sec_WhitePaper_v5_CLEAN.pdf





- IoT devices with increased security, an asset that further strengthens the current state of the art Security provision in IoT devices on a hardware level.
- Intrusion Detection System (IDS), an asset that monitors communication between IoT devices and cloud in order to detect, prevent, and report any suspicious activity that may be a sign of an attack.

Then, the report presents the M-Sec cloud and data-level security. Three demonstrators are described:

- Hardware-based security, an asset which extends IoT device by bringing strong and complex encryption scheme as well as authentication capabilities to enforce the mutual verification of the connection between the device and the cloud
- Software-based “Visualization Tool for Security”, an asset that utilizes the security data from IoT based Intrusion Detection System (IDS) for security monitoring purposes.
- GANonymizer which processes video streams in order to remove sensitive/private data such as people walking on streets and also cars.

Afterwards the document presents three different demonstrators as well as the corresponding services and gives installation details. The main focus of the presented demonstrators and tools is to implement the M-Sec blockchain framework to facilitate the convergence of IoT security with blockchains in order to support an innovative smart city platform.

Later, the report consists of two different demonstrators as the main outcome of Task 4.4: Application level security. The two demonstrators provided in this deliverable contribute to the multi-layered security in different parts.

With the Crypto Companion Database we provide security for sensitive data that cannot be stored on Blockchain. The two main goals of this demonstrator are to encrypt and save the data and to secure it. In order to fulfil these requirements a secured API is provided. This API is also connected to a developed module called Crypto Module that allows the encryption and decryption of data per user.

With the Security analysis tool we provide security requirements that only the use case diagram cannot elicit. The main goal of this demonstrator is to create a misuse case diagram that enables the association of security knowledge and elicit threats and security requirements of the use case.

Finally, the document presents the reference design demonstrating M-Sec end-to-end security. End-to-end Security has a particular approach in M-Sec Project as it is a combination of the output from the 4 other tasks of the project, completed by a Security manager ensuring a secured and smooth interoperation of each element of the architecture. The functions provided by this security manager can be used directly by the assets of each task or can be interfaced using the two main middleware of the project.



IoT Security

1.1 IoT devices with Increased Security

The Internet of Things (IoT) has changed the way people interact with technology, and IoT security is a growing concern that is reaching a boiling point as of today.

People's connected devices are data collectors. The personal information collected and stored with these devices — such as user name, age, e-mail addresses, health data, location, and more — can aid criminals in stealing their identities.

At the same time, IoT is a growing trend, with a stream of new products hitting the market. But here is the problem: When you're connected to everything, there are more ways to access your information. That can make you an attractive target for people who want to make a profit from your personal data.

Every connected device you own can add another privacy concern, especially since most of them connect to your smartphone. But the more functionalities you add to your smartphone, the more information you store in the device. This could make smartphones and anything connected to them vulnerable to a multitude of different types of attacks.

In the particular scenario addressed by this deliverable, the authors will focus on the security incorporated to the IoT devices themselves, being Task 4.4, the one dealing with the techniques to apply in the application side of the equation in order to increase the security.

General Description

The prototype about to be described in this subsection will be in charge of deploying a series of novel IoT devices in selected locations in the city to both retrieve interesting environmental data along with a measurement of noise level while on the other hand will also be capable of sketching crowd heat maps, using as a source of information the number of mobile phones in the area.

These IoT devices will achieve its goals of implementing security measures through the integration of hardware components.

In this demonstrator, we present a technique to increase the security level of a physical object via an extension conforming to the "TPM" (Trusted Platform Module) profile standardized by the TCG (Trusted Computing Group), similar to a trust anchor. The security in question primarily relates to the integrity of the product to ensure that the product has not been compromised to extract sensitive information such as private keys or other authentication information with nuisance capability. These integrity checks can be done at different levels depending on the type of targeted platform: boot loader, Operating System (OS), and applications.





Components

Environmental monitoring devices (EnMon)

The environmental monitoring devices architecture is based in the HAL of the microprocessor STM32L4 and the programming environment is called System Workbench for STM32, based on Eclipse.



The devices use **Narrow Band-IoT** (NB-IoT) modules produced by Quectel for these narrowband experiences, specifically, BC95 and BC68, which are shown in Figure 2.



Figure 2: NB-IoT modules BC95 (left) and BC68 (right)

As of today, these IoT devices send environmental data related to temperature, humidity and noise and do so following the frame introduced in Table 1. The next steps in the prototype evolution will incorporate CO₂ and VOC sensors to help establishing comparisons among different areas of the location to deploy them, analyse the emissions levels and therefore derive strategies that may impact positively the city and its inhabitants.



Table 1: Environmental monitoring devices data frame

FIELD NAME	FIELD LENGTH	VALUE	TENTH VALUE	UNITS
TYPE	1B	01	1	-
LENGTH	2B	001c	28	-
N_SEND	4B	00000000	0	-
IMEI	16B	38363737323330333030343033333000	867723030040330	-
BATTERY_VOLTAGE	2B	0dd0	3536	mV
SONOMETER	2B	0040	64	dB
TEMPERATURE	2B	007c	124	°C x 10
HUMIDITY	2B	0235	565	% x 10

An example related to how this kind of **data frame** looks in real life trials is as follows:

```
01001c000000000383637373233303330303430333330000dd00040007c0235
```

At first sight, and knowing the characteristics of our developments, the way to interact with other components could consist in packing those developments (such as Wolf Secure Sockets Layer (SSL) library) in a library that could be integrated in EnMon's own architecture for the L4, created with HAL (Hardware Abstraction Layer) libraries following ST guidelines.

The initial prototypes for these environmental devices, relying on a STM32-L4 microprocessor went through an interaction with the TPM from the ST Microelectronics portfolio. Nevertheless, after thorough testing, the final prototypes rely on STM32-L422c bu6 Series microprocessor with data encryption capabilities.



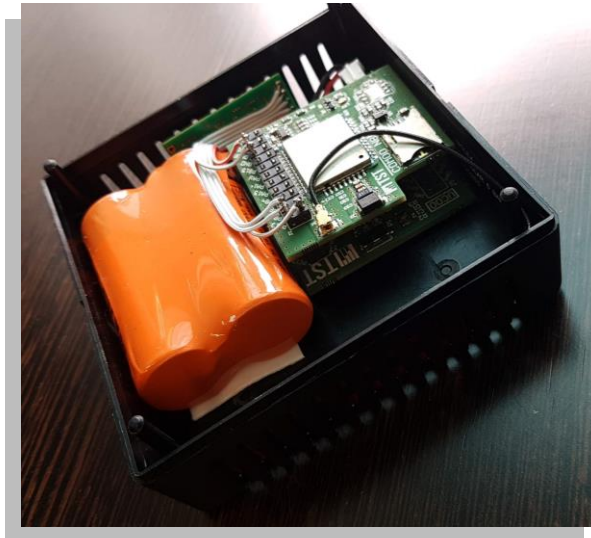


Figure 3: Environmental monitoring device circuitry

The current shape of this IoT device is the one featured in Figure 3 and in Figure 4. This prototype is encased in a proper IP casing to deploy it outdoors and substitutes the slot to plug into the power socket initially considered for a set of batteries to make it autonomous, as depicted in the pictures.



Figure 4: Environmental monitoring device appearance



Crowd counter devices (Crow)

On the other hand, the crowd counter IoT device works with a Raspberry Pi, relying on buildroot²; it should be compatible with Debian when the time of compiling comes.

Right now, Ethernet is only used during the device boot. Afterward, when the device is working, it employs Point-to-Point Protocol (PPP) and a communication module. Nevertheless, if we suppress that communication module, we could retort to Ethernet to proceed with the sending of data.

These crowd counting initial prototypes integrate **Raspberry Pi & TPM** to act securely during the initialization process and provide a measured boot. Final prototypes are also implementing advanced encryption mechanisms to prevent undesired external accesses that may affect the device functioning and thus the data it delivers. Both pictures in Figure 5 help to catch a glimpse of how these devices are built.

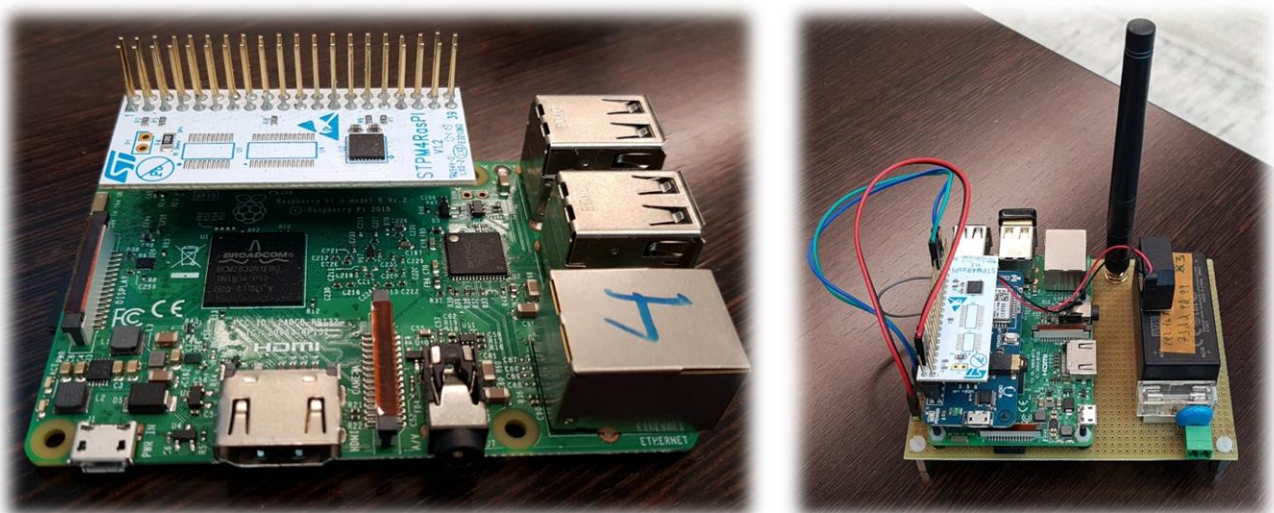


Figure 5: Crowd counter devices circuitry

² Buildroot, <https://buildroot.org/>



Referring to **data**, the information sent by this IoT device follows the structure depicted in Table 2, where it is easy to appreciate the different parts of every frame, and a real life example is also offered.

Table 2: Crowd counter data frame

SN@WIFI@\$DATE@\$TEMPERATURE@\$NUM_WIFI_PKTS@\$NUM_WIFI_DEVICE	MAC@RSSI
000000006956C35F@WIFI@20192208-22:41:55@65.53@12076@5	CC:4B:73:64:59:66@-41
	B8:27:EB:71:FF:F5@-57
	8C:F7:10:07:AE:C2@-71
	00:16:9D:F5:0B:80@-85
	44:07:0B:E9:4D:30@-35

As of today, the current design of this device presents the appearance depicted in Figure 6, including a slot to plug the battery charger. Nevertheless, the final goal implies preparing an autonomous version, which allows the Municipality services to treat it as a portable device and move it among different locations within the city.



Figure 6: People counter IoT device appearance



TPM2 device or equivalent

A TPM is the root component of this demonstration. It provides security primitives such as encryption algorithms, key storage, true random number generator, hash functions, etc. that runs in a secure way compared to traditional implementation.

There are many implementations of TPMs. The most classical one is a hardware component, which has the highest protection profile for hardware attacks (injection faults, side-channel attacks, etc.). Thus, other implementations have been deployed, such as the fTPM, which are firmware-based implementation.

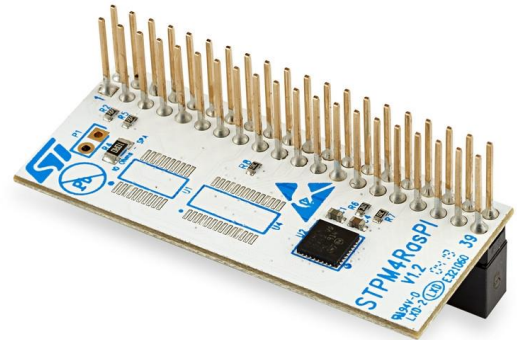


Figure 7: Development board for the STSAFE-TPM

These have been made popular by a manufacturer such as Intel. Finally, some software implementation also exists, whether they are simulators or applications targeted for Trusted Execution Environments. For this demonstration, we used the development board “STPM4Raspi” as shown in Figure 7.

Measured boot

We have developed a measured boot for both hardware targets, based on STM32L4 and the BCM2837 included in the Raspberry Pi. One of the difficulties is that those targets do not support natively secure boot as the boot loader is static or private. Instead of a secure boot, where each loader is verifying the N+1 loader prior to pass control to it, we have implemented a measured boot.

The goal of the measured boot is to ensure the integrity of the platform, making sure that it corresponds to what is expected. It prevents some attacks such as firmware tempering in which, for example, an attacker could perform eavesdropping on the OS kernel operation in order to gain knowledge about cryptographic operations.

This implementation relies on the TPM functions and, in particular, the PCR (Platform Configuration Registers), which are specific memory locations within the device. There are typically 24 PCR on a physical TPM2 device. These registers contain hashes, SHA-1 hashes on the first version, and SHA256 hashes on the second version. We do not write a data into a PCR, but we extend it with a new value. The value of a PCR depends on the previous value and the new one regarding the following function:

$$\text{PCR new value} = \text{Digest of } (\text{PCR old value} \parallel \text{data to extend})$$

It keeps a state of the system and its modifications during runtime. PCR can be used as an authentication policy to seal and unseal data in the non-volatile memory of the TPM and can also be used for attestation regarding the platform state.

Our usage of this function is to measure the system integrity and prevent to execute an application or release data if the system is not at a trusted state. Our approach is described in Figure 8. We have modified





the Raspberry Pi boot chain in order to add measurements at boot. During the boot stage, all data is encrypted on an SD Card, and the key is stored into the TPM bound to a policy based on PCRs states.

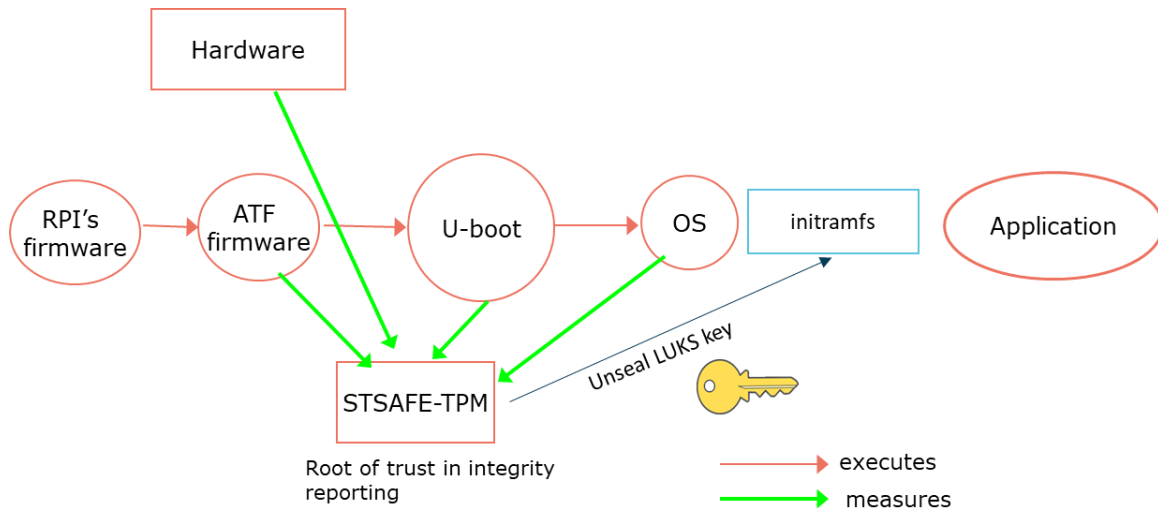


Figure 8: Measured boot process

If the measurements stored in the PCRs are equal to the trusted state, the decryption key can be unsealed by the kernel in order to mount the operating system. Otherwise, the decryption will not happen, and the operator will be asked to fill in manually the decryption key.

OS security

In complement to the measured boot, we have enabled some auditing features regarding the operating system's state, still using the secure element capabilities. Those auditing features rely on the IMA and EVM modules, namely Integrity Measure Architecture and Extended Verification Module.

This feature prevents any modification at runtime, in particular privilege escalation that can be gained from a remote attack. Many attacks on the Linux kernel occur and are subject to Common Vulnerabilities and Exposures (CVE)³ publications. These can be prevented and detected using such features.

At this stage, this component only reports suspicious activities, such as dynamic module loading, within the logging system (both local and remote using *rsyslog*). Thus, in a future version, we would like to extend it with a remote attestation platform in coordination with task T4.5 in order to manage this suspicious activity such as for example by placing the device into quarantine, preventing data and other connection from the device to spread to the system until it has been proven safe. This sanity of the device can be done at runtime using the "quote" capability of the TPM.

³ Common Vulnerabilities and Exposures website, <https://cve.mitre.org/>





Installation Instructions

This demonstrator will be part of the two pilots that constitute Use Case 1, to be conducted in Santander (Spain).

Required Tools and dependencies

Assuming the build machine is debian-based (debian, ubuntu, etc.), the following dependencies shall be installed:

```
apt-get install crossbuild-essential-arm64 fakeroot git kernel-wedge quilt ccache flex bison  
libssl-dev rsync libncurses-dev bc patchutils dh-python dh-exec libelf-dev device-tree-compiler
```

TPM physical connection

In production, the TPM would be directly soldered and routed on the board, but in our case, the devices to secure are legacy, or we are in a prototyping phase. The pinout of the board we have been using is designed for the Raspberry PI GPIO (General Purpose Input/Output) port. It can be plugged directly on top of the Raspberry, as shown in Figure 9.



Figure 9: Raspberry PI with the STPM4Raspi extension in White

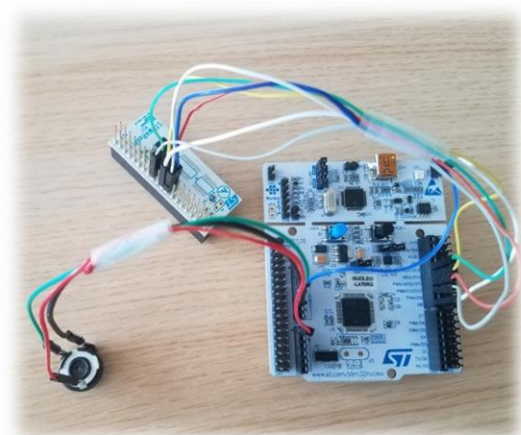


Figure 10: External wiring for microprocessor developments

For other devices, we can use either SPI (Serial Peripheral Interface) bus or I2C (Inter-Integrated Circuit) bus to connect the TPM to the micro-controller or processor. We have done a temporary wiring in order to conduct trials using an STM32L4 (Nucleo L476RG development board), as shown in Figure 10.



Bootchain

The bootchain is made of two components: the ARM trusted firmware and the U-Boot bootloader. In some cases, the Advanced RISC Machines (ARM) Trusted Firmware may be optional, its main functionality is to enable the TrustZone®, which is the ARM 's implementation of a Trusted Execution Environment (TEE).

U-Boot has been patched in order to support the physical TPM. Patches consist in:

- 1) declaring the TPM component on the SPI bus in the device tree
- 2) adding SPI support for the BCM2738 chipset.

A predefined configuration enabling the TPM commands, libraries, and associated security dependencies have been added. Optionally, the TrustZone® features can be enabled using the OPTEE OS in the secure world.

If Open Portable Trusted Execution Environment (OPTEE) is to be added, we can use this other specific command to do so.

Linux Kernel

In order to be fully available to application and to ensure the OS integrity, the Linux Kernel has to be recompiled with specific additional features such as:

- Declaring the TPM device within the device tree
- Enabling TPM2 driver (as a character device)
- Enabling the Integrity measurements and extended verification module (IMA and EVM)
- If needed, enabling the OPTEE driver for trusted applications

Linux Security

In order to monitor the execution of Linux and its application, we use the IMA/EVM module that we need to configure for our demonstration. The modules have been activated within the kernel configuration in the previous step, and then, we need to configure them with a list of files to monitor. We propose a script to deploy the parameters to be run after the installation of the tools.

```
apt-get install autoconf libtool libssl-dev libattr1-dev libkeyutils-dev asciidoc ima-  
evm-utils  
/opt/ima/deploy.sh
```

The policy regarding this module can be edited in `/etc/ima/ima_policy` while the measurements can be monitored in the `/sys/kernel/security/ima/ascii_runtime_measurements` file.

Download and Run Demonstrator

The following files compose the demonstrator:

- *arm-trusted-firmware.tar.bz2* which contains the ARM's Trusted Firmware, which acts as a hypervisor between the Linux OS and the TrustZone during runtime.
- *u-boot.tar.bz2* which contains the Linux boot loader with a patch to support the SPI bus and the proper configuration in order to enable the TPM2 (device tree) and the measurements (boot script)





- *optee.tar.bz2* (optional) which contains the OP-TEE system for the TrustZone® tailored for the Raspberry PI.
- *linux-4.19.12.tar.bz2* which is a patched version for the debian kernel for the Raspberry PI with TPM2, IMA/EVM, and OP-TEE driver. The device tree matched the TPM2 development board we have been using.
- *rpi-sdcrypt.tar.bz2* which is a set of files to be deployed on the operating system in order to manage the encryption of the partition and decryption using the TPM2 NVRAM based on a PCR policy. IT handles the provisioning phase as well as the *initramfs* generation for the decryption.

At the time of edition of this deliverable, the development files have not been published, and this publication is under review at CEA.

Licensing

Even if most of the developments have been made using open source code, the status of the development includes certain blocks of proprietary code (non-free). Nevertheless, the future steps will, for sure, imply developing open source code that could be made available.

Currently, some review process is being conducted in order to push some patches into mainstream repositories.



1.2 Intrusion Detection System (IDS) for IoT devices

General Description of the Prototype

As proof of concept for a software-based security solution, we have selected Use Case 3, which defines a common scenario that can be expected in a hyper-connected smart city, where information from sensors or IoT devices needs to be delivered without compromising the triads of information security, i.e., confidentiality, integrity, and availability. Since deliverable D4.5 will be addressing the threats and security elements for a hyper-connected smart city, and D4.3 will be addressing the cloud layer. Therefore, in this document, as stated previously, we will focus on the security components required for securing the IoT devices layer.

As it can be seen from Figure 11 below, the Mobile Sensing Platform being used in Use Case 3 is directly connecting to the internet without any security features. We need to secure it for delivering a reliable sensor data via the internet. Using research from IoT honeypot, we can secure this platform by using an intrusion detection system with customized signatures for preventing cyber-attacks in order to secure this mobile sensing platform.

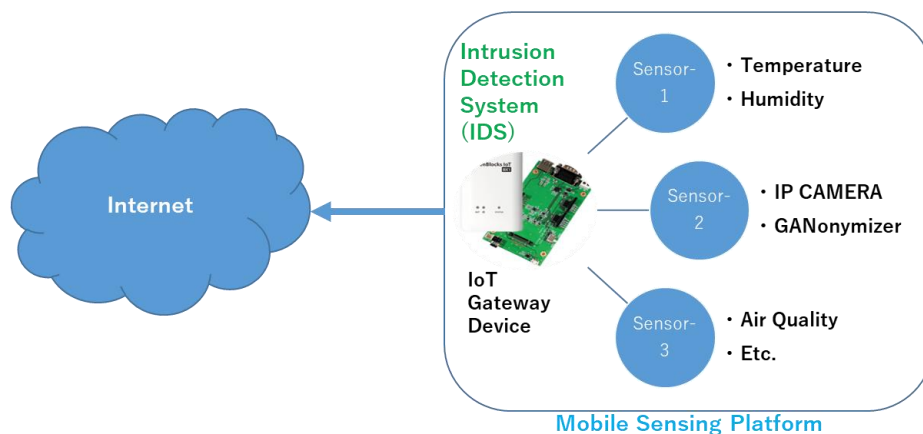


Figure 11: Secured Mobile Sensing Platform

Components

Internet connectivity exposes IoT devices to potential threats from bad actors (cybercriminals) with malicious intent. As a firewall is too heavy for IoT devices resources, therefore, we have adopted lightweight Intrusion Detection System (IDS) for providing security to the IoT devices layer along with OS hardening. OS hardening will help in reducing attack surface by closing all the ports that are not needed. This way, we can monitor threats as well as prevent known attacks without consuming too many resources of the IoT devices.



Honeypot (IoTPOT)

We first need to understand what weaknesses IoT devices may have that can be exploited and then mitigate them with appropriate security solutions. A honeypot can be one such analysis tool that can be used during the development and testing phase. Symantec defines a honeypot as a computer system that can be used to attract attacks and analyse how bad actors conduct such attacks. It can also be used to check the vulnerabilities in a device or system and strengthen security to mitigate vulnerabilities.

IoTPOT honeypot is an independent computer system that attracts attackers and helps with analysis of their attack techniques or patterns by exposing itself as an ordinary device connected directly to the internet. Unique patterns are obtained from such analysis that can be used for creating signature patterns for Intrusion Detection Systems (IDS) during the development stage. Furthermore, honeypot can be used in the testing phase for finding out vulnerabilities in internet connecting devices for security improvements.

IoT Gateway

IoT gateway has the embedded communication hardware in the IoT devices that has a communication module to connect to the internet, either via Local Area Network (LAN) / Wide Area Network (WAN), Wi-Fi, Bluetooth, or 3G/4G/LTE interfaces. Figure 12 offers a depiction of this device.



Figure 12: IoT Gateway Device

Intrusion Detection System (IDS)

IDS is the software module that is installed in the IoT device for monitoring and reporting purposes, along with the option for blocking malicious traffic matching known signatures. For more reliable signatures, we have also used honeypot (IoTPOT) for testing various IoT devices, analysed and extracted cyber-attack patterns or signatures. We will be using a network-based IDS. Therefore, this module can examine the network traffic, flag a packet if a known attack pattern match is detected, log the event for more analysis and then block/drop it based on configured rules to protect IoT devices from a potential attack.



Installation Instructions

For the sake of easiness, we have compiled the customized OS hardening commands and the IDS software together as a package for easy installation on various IoT gateway devices.

Required Tools and dependencies

The demonstrator has been designed for securing mobile sensing platform having the following specifications:

- Intel Atom Processor 500 MHz Dual Core, 1GB RAM, 4GB Flash, Debian GNU/Linux.

Download and Run Demonstrator

The installation file is named as “*installer.sh*” that can be downloaded securely over 3G or internet connection with the following command:

```
$ scp -P 64295 -i <key> rainforest@xxx.xxx.xxx.xxx:~/iot-k/installer.sh .
```

After downloading, check and confirm the integrity of the downloaded file using Message-Digest Algorithm 5 (MD5) hash, as follows:

```
$ md5sum installer.sh
MD5: 5cb38fb1754267c7d699565f00e3262c
```

Installation Steps

- 1) The “*installer.sh*” file should be downloaded into the “/root” directory and the directory should look something like this:
- 2) Launch installer shell as follows:
- 3) Wait for it to complete updates, download, and install the program. It will finish installation and return the prompt as follows:
- 4) Now check root directory contents and you should see as follows:

Run Demonstrator Guide

Demonstrator runs in the background, sending log events to the visualization tool in the cloud. On IoT gateway devices, IDS program will be initiated at startup with the following command:

```
root@iot:~# ./start_ips.sh
```

Licensing

The software solution developed as IDS for the IoT devices is based on Open Source Software (OSS) and, therefore, does not need any licensing. Whereas IoT POT is a proprietary asset that is used for study and analysis purposes only during the research and development phase.





Cloud and Data Level Security

1.3 Hardware based encryption

General Description

Data encryption is an operation that requires a lot of calculation steps. For an IoT product, generally constrained in computing resources and in energy, these operations are delicate because they can cause latency problems related to too-long computation time, or they can still use too much energy thus damaging a battery in a premature way. Similarly, encryption software implementations may be subject to vulnerabilities that can be exploited. A typical example of such vulnerability is the “heartbleed” (CVE 2014-0160⁴) software vulnerability which was introduced in OpenSSL and has affected many devices such as smartphones, core internet routers and hard-drives firmware.

A countermeasure for such vulnerabilities is to use a hardware component, such as an HSM, a crypto-accelerator or a TPM. With such a device, it is expected to protect not only the private keys of the encryption, but also the process of encryption/decryption itself so it cannot leak data to unauthorized software. Our demonstration aims to show this enhancement on constrained devices, where the employment of secure elements is emerging. It relies on the developments carried out in T4.1 and presented in deliverable 4.1 where the same secure element is used to provide system integrity.

Hardware target

This demonstration is made for typical IoT devices, powered by a micro-controller with energy limitation and using narrowband communication channel for direct connection to the cloud. Such a device is described in D4.1. It uses the STM32L4 microcontroller; this MCU plays the role of a connected object. For development purposes, we have used a NUCLEO-L476RG development board embedding such MCU. This device has a sensor attached to it, a potentiometer, simulating any analogue environmental sensor. It also has a STSAFE-TPM with an SPI connection to the MCU. These components are described in Figure 13.

⁴ <https://cve.mitre.org/cgi-bin/cvename.cgi?name=cve-2014-0160> / <https://en.wikipedia.org/wiki/Heartbleed>



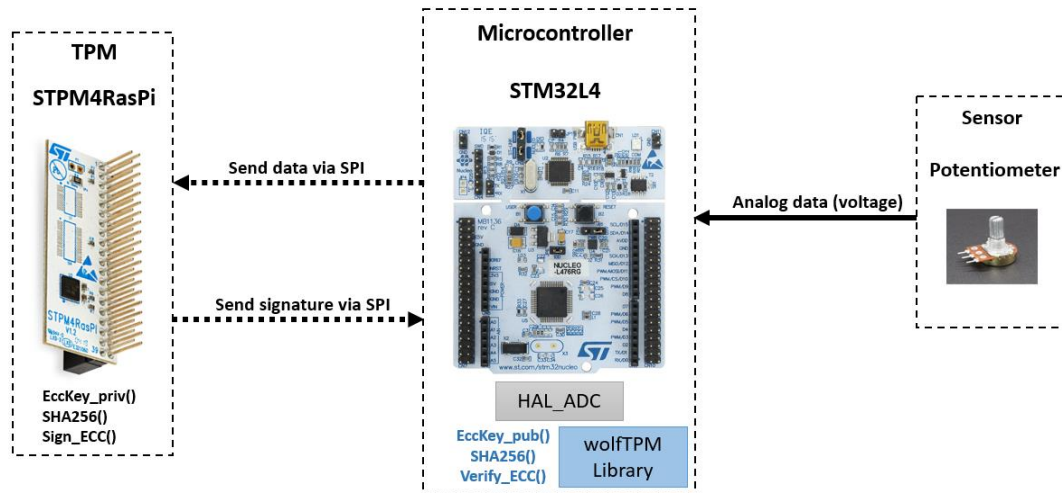


Figure 13. Scheme for the hardware-encryption demonstrator

Software library

In order to integrate the TPM at the STM32L4 level some steps are necessary. First, we use the wolfTPM library from wolfSSL to integrate the TPM features. WolfTPM library is a compliant TPM 2.0 stack, designed for embedded use. It is highly portable, due to having been written in native C, having a single I/O callback for SPI hardware interface with no external dependencies, and its compacted code with low resource usage. This uses the TPM Interface Specification to communicate over SPI and Includes wrappers for Key Generation, RSA encrypt/decrypt, ECC sign/verify and ECDH and Symmetric AES encrypt/decrypt.

Installation Instructions

Required Tools and dependencies

We have used the “TrueSTUDIO for STM32⁵” tool in order to develop and build the demonstrator’s firmware. Once the firmware is compiled from the source, it can be easily flashed to the device by simply copying the binary to the USB mass storage device which is mounted by plugging the device using a USB cable.

In order to run the demonstrator, the operator shall have the device, a USB cable with mini connectors and a computer with a serial terminal installed such as Putty under Windows or Minicom on Linux systems.

User Manual

The TPM is connected and powered with the STM32L4 via SPI and on the other part the potentiometer is plugged into via ADC pins. In particular, the TPM shall be wired as described in Table 3

⁵ <https://atollic.com/truestudio/>





Table 3. Connection layout between the MCU board and the TPM daughter-board

Function / signal	Pin on board	Pin on daughter board
SPI1_SCK	PA5	TPM_pin_23
SPI1_MISO	PA6	TPM_pin_21
SPI1_MOSI	PA7	TPM_pin_19
SPI1_CS	PB6	TPM_pin_24
rst_tpm	PA9	TPM_pin_18
	VCC	VCC
	GND	GND

To start the demonstration, we need to plug the board to a computer having a terminal using a USB-mini cable. The device will be powered-up directly from the USB port and will execute its firmware. Then, we can connect to its console using a terminal with the following parameters:

- Speed baud = 115200
- Data bits = 8
- Stop bits = 1
- Parity = None
- Flow control = None.

In order to have the initial output of the firmware we need to reset the device using the black push-button located on the board as illustrated in Figure 14.

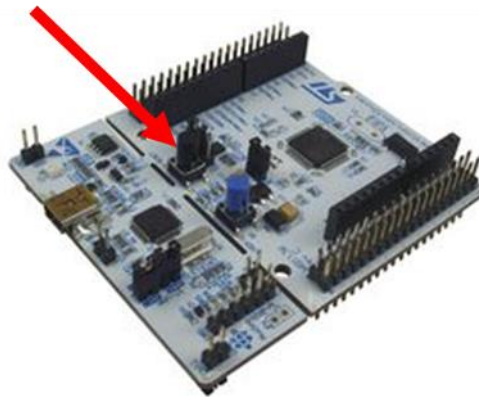


Figure 14. Location of the reset button on the Nucleo-L476RG board

At this stage, the device is unprovisioned and the first step is to provision the TPM with keys, keys for the session management and keys for the application.

First, we generate keys in order to preserve the privacy of the exchange between the TPM and the MCU. Indeed, as this exchange of information occurs on an SPI bus, an attacker can eavesdrop this communication. We prepare an authenticated session for the further commands from the CPU to the TPM which will encrypt most of the parameters in the next steps. To initiate authentication with the TPM, a session can be opened. When a session is started, the TPM processes the command and generates a session handle, computes a TPM nonce and calculates a session key. This key is used to generate HMACs, encrypt command parameters and decrypt response parameters. After the session is created, the session key remains the same for the lifetime of the session. The session handle and the TPM nonce are returned by the command.



The next step is to launch the creation of the primary key derived from the static seed Endorsement Key (EK). This new primary derivation key is transferred and stored in an NV (Non-volatile) ram in the TPM. Subsequently, all the keys that will be used for the security primitives will be derived from the primary key.

Then we generate a private key. To make a signature of the data, we use ECC (Elliptic-Curve Cryptography) asymmetric algorithm as a cryptographic scheme. This scheme begins to be widely used at the IoT level thanks to its energy performance. It has good efficiency in terms of implementation. Data coming from the sensor will be encrypted using this private key which will be kept into the memory of the component. This key is stored at the handle 0x80000004.

Once the provisioning phase is done, the application will loop over the following sequence

- 1) Read the analog value from the potentiometer
- 2) Send the analog value to the TPM for encryption
- 3) Retrieve the encrypted value from the TPM

The value retrieved via the ADC is sent via SPI in the TPM, and then a SHA256 value is calculated in the TPM. Once the hash fingerprint is generated, it is encrypted with the private key. In order to verify the validity of the previously signed data in the TPM, the public key is sent to the MCU, the signature is returned from the TPM to the STM32L4 and then decrypted using the public key generated in the TPM. A hash is calculated in the STM32L4 and compared with the one generated from the signature. If the two hashes are identical, the signature is validated. As a result, the integrity and authenticity of the data has been proven through the TPM.

Licensing

WolfSSL, the crypto library with TPM support used in this demonstration is available under two licensing models, either open-source or commercial. More information is available on the editor website: <https://www.wolfssl.com/license/>





1.4 Software-based Threat Monitoring

General Description

In order to stay vigilant and monitor threats to the IoT devices layer from anywhere in the cloud, an analysis tool is required that can translate the data into easy-to-understand graphical information. This visualization tool is a software-based solution that collects and examines activity from IoT layer or agents embedded in IoT gateway devices. This tool will not only help with the security health checks by providing insight into how the security is being maintained at IoT gateways, but also helps in further analysis of devices under attack. Thereby, providing 24/7 security threat monitoring and alerts.

Visualization Tool

The software solution consists of **four modules**:

1) Data Collection Agents

A software solution embedded in the IoT Gateways is used to collect logs generated by the IDS at the IoT security layer and forward the security data to the aggregator module.

2) Aggregator Module

Aggregator agent in the cloud organizes the data in real-time for further analysis.

3) Data Analysis Module

Data analysis engine examines all the data by matching with well-known attack patterns and forwards suspicious activity to the visual graphics module.

4) Visual Graphics Module

Visual graphics module converts the data to easy-to-understand information for further investigation at the security monitoring station.



Figure 15. Visualization Process

As shown above in Figure 15 the process is simple. IDS in IoT Gateways monitors the public interfaces by matching traffic with known signatures. These signatures are extracted from various testbeds, including research and analysis involving the honeypot (IoT POT). Embedded agent (*filebeat*) in IoT gateway uploads the logs to the aggregator module in the cloud for further analysis. The aggregator module collects all the data from various IoT gateways and passes it to the data analysis engine, which examines all the data with the help of the threat detection module. The results are flagged as alarms in the visual graphics module for further investigation and easy-to-understand security threat monitoring.



Installation Instructions

Required Tools and dependencies

Following are required for this demonstrator:

- Laptop/Desktop with CPU (i3 2.4GHz or above), Memory (16GB or above), Hard disk (512GB or above), and an internet connection.

User Manual

An installer has been compiled to download and install all the required open source software files along with customized settings for use case 3 and 4 pilots on the IoT gateway device. The software runs in the background and information is displayed on the monitoring screen.

The installation file is named as “installer.sh” that can be downloaded securely over any internet connection with the following command:

```
$ scp -P 64295 -i <key> rainforest@xxx.xxx.xxx.xxx:~/iot-k/installer.sh .
```

After downloading, check and confirm the integrity of the downloaded file using MD5 hash, as follows:

```
$ md5sum installer.sh  
MD5: 5cb38fb1754267c7d699565f00e3262c
```

Installation Guide

- 1) The “installer.sh” file should be downloaded into the “/root” directory.
- 2) Launch installer shell
- 3) Wait for it to complete updates, download, and install the program.
- 4) Confirm that filebeat program is running.

If not running, then start it with the following command:

```
$ /etc/init.d/filebeat start
```

Running Demonstrator

PC client can connect with the filebeat server from anywhere using secure pre-set credentials and shall look something like as shown in Figure 5 and 6. This is a **visual tool for security-related events or alarms**. User needs to login with the pre-set credentials and watch for security events/alarms and examine security reports. If further investigation is needed, then a security expert should examine the logs/alerts and take appropriate actions, ranging from further data analysis to updating threat signature patterns on the IDS.

Licensing

- This tool is based on Filebeat and Kibana open source code, customized according to M-Sec needs and requirements.





Figure 16: Client Dashboard – Alerts Overview

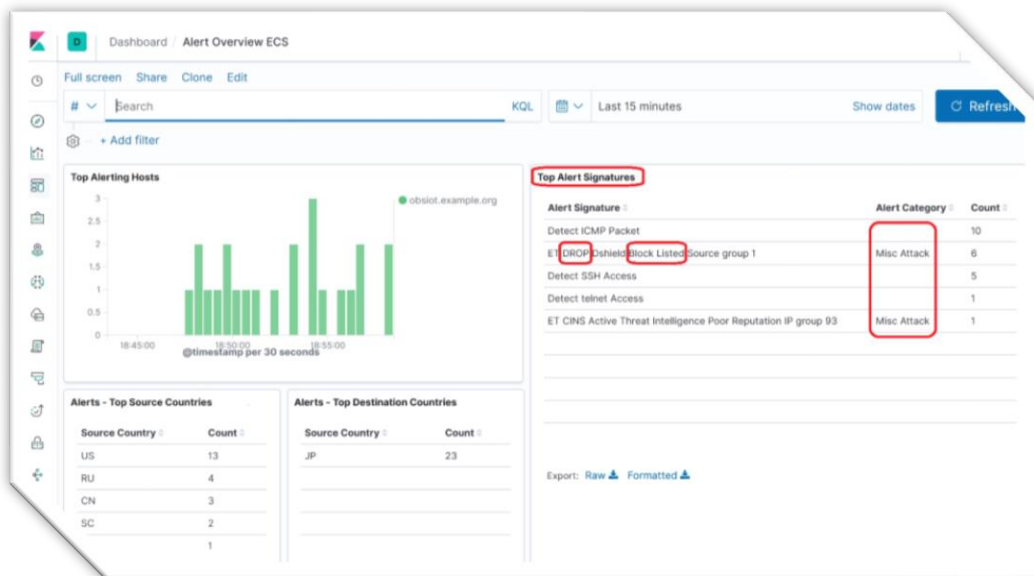


Figure 17: Client Dashboard – Events Overview





1.5 Privacy Management Tool

General Description

Use cases for handling video data in various situations in smart city solutions are increasing. For example, surveillance video for security, video monitoring of transport infrastructure and social infrastructure, people flow analysis, such as disaster prevention measures, etc. But in many cases, such video data includes privacy information that can identify an individual. It is, therefore, necessary to remove such personal information that can become a violation of General Data Protection Regulation (GDPR) and Japanese regulations. Hence, an application called "GANonymizer" is being developed as a solution that automatically removes objects related to personal information using Deep Learning.

Video Image Privacy - GANonymizer

GANonymizer is an imaging processing tool to remove (make transparent) the pedestrians and cars recorded in the driving record videos. Its objective is to avoid the privacy leakage when distributing and utilizing the videos. The GANonymizer is developed using deep learning-based object detection techniques and is currently designed to run securely on the KEIO's secured mobile sensing platform for use case 3 & 4, for addressing the privacy issue.



Figure 16: GANonymizer Test Results

The results of applying GANonymizer are shown in Figure 16.

- Upper row images are original video images
- Lower row images are after removing privacy objects, like cars and human objects.

GANonymizer consists of two parts of neural networks as shown in Figure 17.

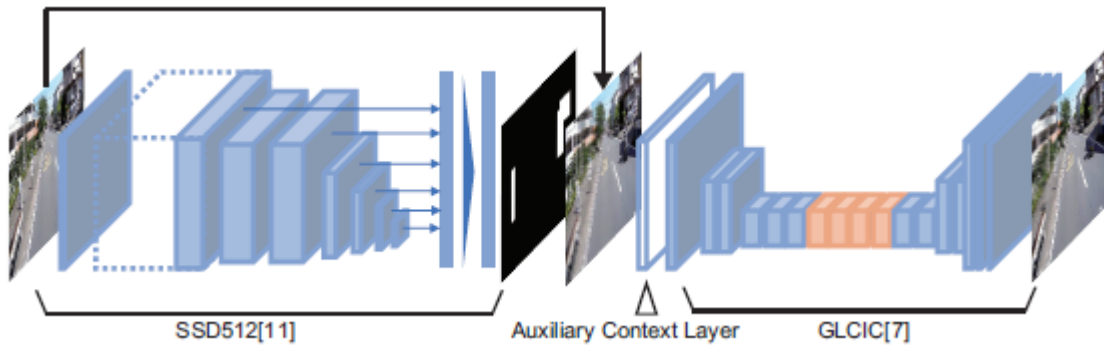


Figure 17: GANonymizer Architecture

In order to detect the target objects from the input image, which might violate the privacy, we adopt the deep neural networks: Single Shot Multibox Detector (SSD). And in order to generate a more natural image, we adopt Globally and Locally Consistent Image Completion (GLCIC) which is one of the most successful models in image completion.

Single Shot multibox Detector (SSD)

SSD is one of the popular models that can detect the object with high accuracy. Especially, we select SSD512 which is the variant SSD model and performs higher than any others. Since the target objects are general and those are contained in PascalVOC dataset, we use the model weights which are trained by PascalVOC.

Globally and Locally Consistent Image Completion (GLCIC)

After the target objects are detected, GANonymizer replaces the area where the target objects exist as if there are no objects. There are a lot of completion methods using computer vision technology. We adopt the in-painting methods which adopt the deep neural networks to succeed in generating the images more realistic and natural.

GLCIC is based on Generative Adversarial Networks (GAN) and consists of three networks: the completion network, a local discriminator network, and a global discriminator network. Since GLCIC requires an image and corresponding binary mask for its input, GANonymizer creates the mask based on the bounding boxes which are the outputs from SSD512. Then GLCIC reconstructs the mask part of the input image based on the whole image and is trained by the procedure of GAN. The local discriminator assesses the quality of the mask part of the input image which is completed by the completion network. Simultaneously, the global discriminator assesses the quality of the entire image which is completed by the completion network. The training is terminated when the discriminator networks cannot distinguish between the original input image and the image which is reconstructed by the completion network, which is when the completion network becomes able to reconstruct the mask part of the input image realistically and naturally.

In terms of object removal, it is significant to naturally reconstruct masks based on the various background of images, Hence, for our GLCIC, we apply the model trained with the place's dataset, which contains the pictures of the various place, so that it can reconstruct the mask more naturally.



Installation Instructions

Currently, the prototype has been made only for testing purposes. There is no installation package of GANonymizer yet, but we are planning some installation package in the future.

Required Tools and dependencies

The development environment for the current prototype of GANonymizer is as follows:

- CPU: Intel(R) Core(TM) i7-6950X @ 3.00GHz, GPU: NVIDIA GeForce GTX 1080 – 4 cores, RAM: 64GB, and OS: Ubuntu

Licensing

Currently, the GANonymizer includes some proprietary code, but open source is planned in the next phase.





P2P Level Security and M-Sec Blockchains

1.6 Blockchain Framework and Middleware Services

General Description

The main focus of this Prototype 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 project⁶, 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 language⁷, allowing anyone to write smart contracts and create DApps. Smart contracts are mainly written in the programming language Solidity⁸⁻⁹.

We have initially experimented with different Blockchain platforms, before concluding to Ethereum based blockchains such as Quorum and examined both public (unpermissioned) and private (permissioned) alternatives of the M-Sec blockchains. The most prominent among them was "Hyperledger", which is described in the next Section. 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.

⁶ J. Ray, "Ethereum Introduction," 11 12 2019. [Online]. Available: <https://github.com/ethereum/wiki/wiki/Ethereum-introduction>

⁷ "White Paper," [Online]. Available: <https://github.com/ethereum/wiki/wiki/White-Paper>

⁸ Ethereum, "What is Ethereum?," [Online]. Available: <http://www.ethdocs.org/en/latest/introduction/what-is-ethereum.html>

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





Components

Hyperledger blockchain framework

We used three different open source Hyperledger Platforms and an overview of the overall architecture is presented in the diagram that follows. The three projects are:

- Hyperledger Explorer: it provides details about the underlying blockchain such as the number of blocks, the transactions, the peers etc.
- Hyperledger Composer: it facilitates the development of smart contracts ("chaincode")
- Hyperledger Fabric: it provides the permissioned blockchain

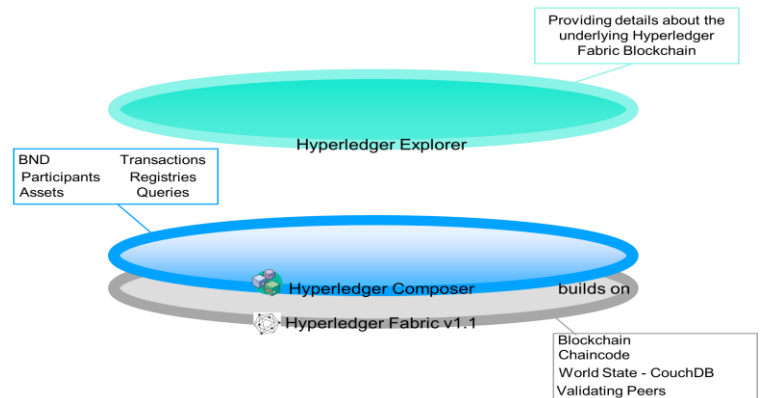


Figure 18: Representation of the architecture of our prototype based on Hyperledger

Our implementation was based on a permissioned blockchain, but we should note that by granting open access to all users, we would have a public-like blockchain.

Through Hyperledger Explorer (see Figure 19) we can inspect the Hyperledger Fabric Blockchain (Transactions, Blocks and others.)

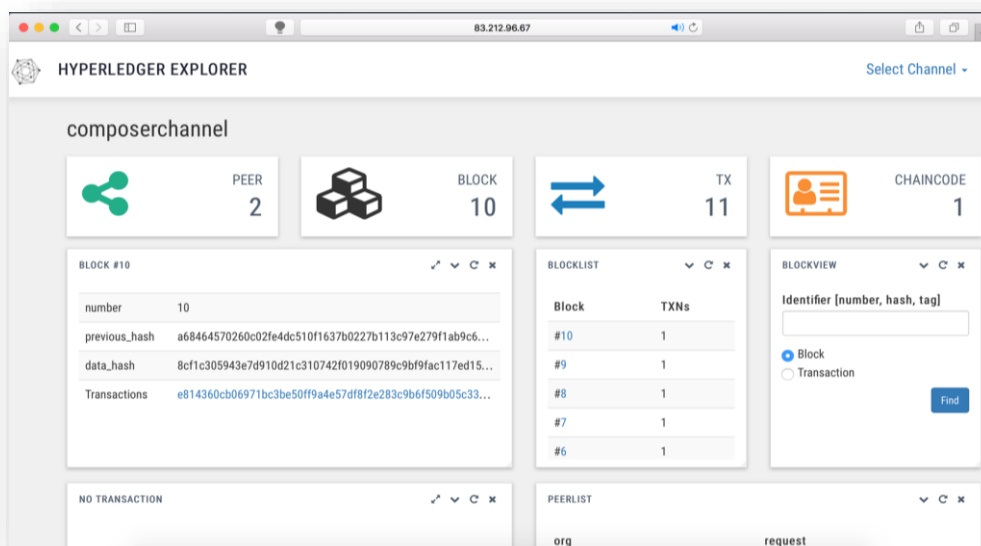


Figure 19. Hyperledger Explorer giving details about the underlying permissioned blockchain



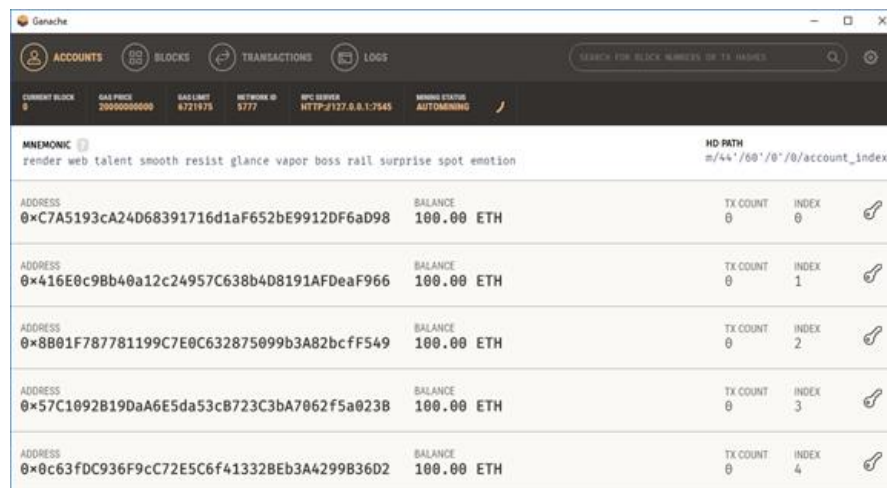


Ethereum & Quorum blockchain framework

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 Solidity¹⁰.

1. Private Ethereum Blockchain

During the development process we have used a local private blockchain named Ganache¹¹, 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 20 and allows us to quickly see the current status of all accounts, including their addresses, private keys, transactions and balances.



The screenshot shows the Ganache Explorer interface. At the top, there are tabs for ACCOUNTS, BLOCKS, TRANSACTIONS, and LOGS. Below the tabs, there is a search bar and a status bar showing current block, gas price, gas limit, network ID, RPC server, and mining status. The main section displays a list of accounts with their addresses, balances, transaction counts, and indices. The mnemonic phrase is also visible at the top.

ADDRESS	BALANCE	TX COUNT	INDEX
0xC7A5193cA24D68391716d1aF652bE9912DF6aD98	100.00 ETH	0	0
0x416E0c9Bb40a12c24957C638b4D8191AFDeaF966	100.00 ETH	0	1
0x8B01F787781199C7E0C632875099b3A82bcfF549	100.00 ETH	0	2
0x57C1092B19DaA6E5da53cB723C3bA7062f5a023B	100.00 ETH	0	3
0x0c63fDC936F9c72E5C6f413328Eb3A4299B36D2	100.00 ETH	0	4

Figure 20. Ganache Explorer allows us to examine all blocks, transactions, addresses and their balances

2. Public Ethereum Blockchain

Additionally, we deployed smart contracts on Public Ethereum Blockchain using browser IDE "Remix"¹². 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 blockchain¹³. 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.

¹⁰ Ethereum, "Solidity," Ethereum, [Online]. Available: <http://solidity.readthedocs.io>

¹¹ <https://www.trufflesuite.com/ganache>

¹² <https://remix.ethereum.org/>

¹³ <https://ropsten.etherscan.io/>





3. Quorum blockchain framework

Finally, the different smart contracts are written in the programming language Solidity¹⁴ and are deployed on Quorum blockchain framework¹⁵. Quorum is a permissioned implementation of Ethereum which allows certified members to build and run decentralized applications that run on blockchain technology. It is 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

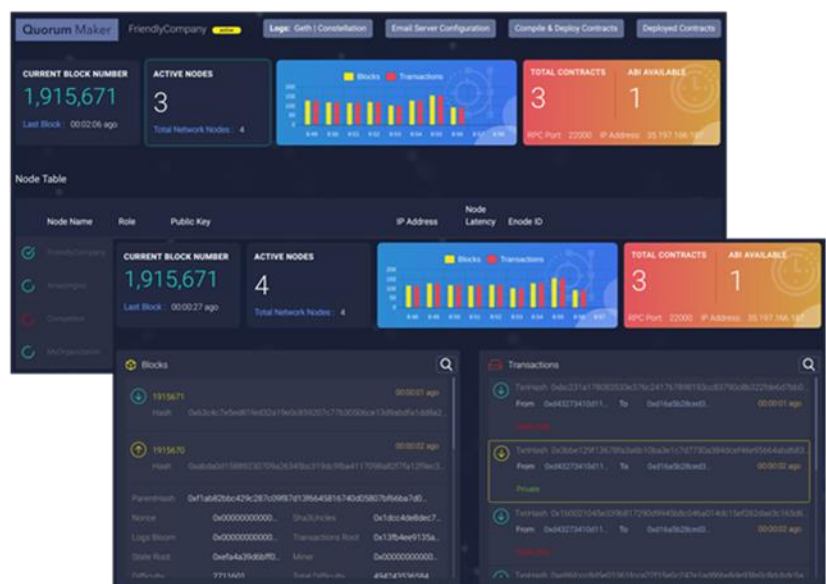


Figure 21: Details about blocks, transactions, addresses and smart contracts

¹⁴ Ethereum, "Solidity," Ethereum, [Online]. Available: <http://solidity.readthedocs.io>

¹⁵ <https://docs.goquorum.com/en/latest/>





thousands of transactions per second, due to its efficient consensus mechanism which belongs to the family of Byzantine Fault Tolerance (BFT) mechanisms¹⁶.

In order to develop and deploy the smart contracts to Quorum blockchain, we have used Truffle suite¹⁷. It is a development environment and testing framework using the Ethereum Virtual Machine (EVM). Additionally, we use Quorum Maker¹⁸, 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 nodes¹⁹.

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.

4. ALASTRIA

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

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.

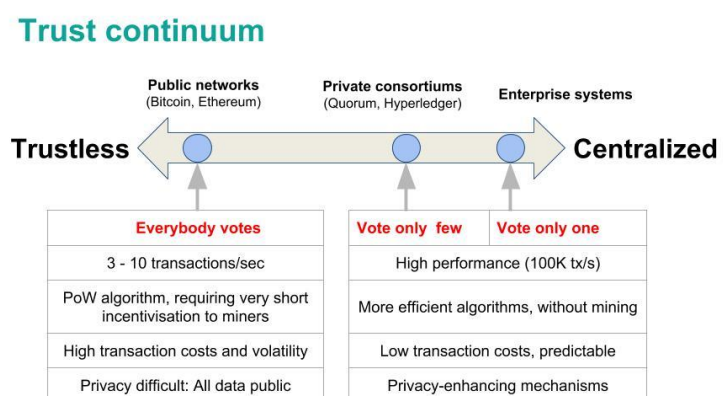


Figure 22: The Trust Continuum

¹⁶ 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

¹⁷ <https://www.trufflesuite.com/>

¹⁸ <https://github.com/synechron-finlabs/quorum-maker>

¹⁹ <https://github.com/jpmorganchase/quorum-examples/tree/master/examples/7nodes>





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*”²⁰, 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.

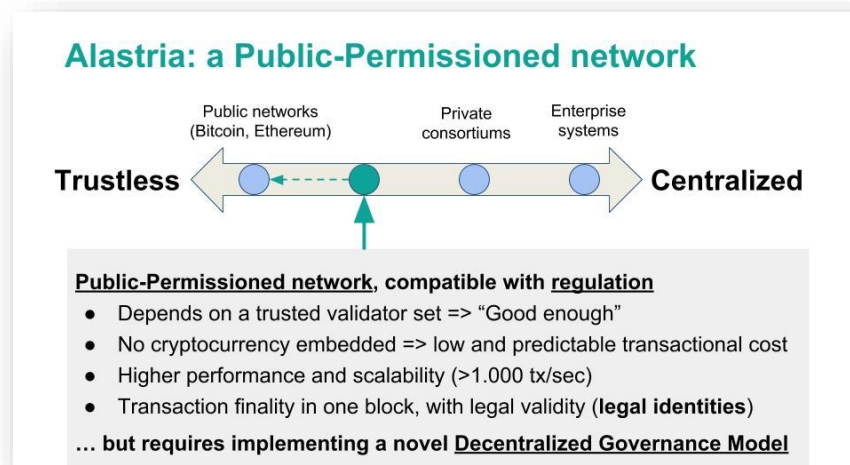


Figure 23. Alastria in the Trust Continuum

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

²⁰ https://medium.com/@aakash_13214/the-scalability-trilemma-in-blockchain-75fb57f646df





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.

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 Alastria²¹.

5. 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 or 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 ERC223²² 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

²¹ <https://github.com/alastria/alastria-platform/blob/master/en/Alastria-Core-Technical-Platform.md#alastria-core-technical-platform>

²² <https://github.com/ethereum/EIPs/issues/223>





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.

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 4: 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 5 more details are provided about the developed functions and events of the M-Sec Token:

Table 5: 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





NAME	INPUT	RESPONSE	DESCRIPTION
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 <i>tokenFallback</i> 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

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 24), 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 24. 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 ;  
}
```

Figure 25. Sensors Smart Contract

4) In the following Table 6 more details are provided regarding our Sensors Smart Contract:

Table 6: 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	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	Uint32 sensorID uint32 fromTime uint32 toTime		This function is called when a user wishes to buy data for a specific time interval and communicates with M-Sec Token to



NAME	INPUT	RESPONSE	DESCRIPTION
			perform this transaction

It is important to notice that functions like *changeSensorInfo* 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.

5) 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 Blockchain²³, 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.²⁴ 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.²⁵ 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

²³A. Anderberg et al., “Blockchain Now And Tomorrow,” 2019.

²⁴W. 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.

²⁵R. 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.





and Yin²⁶ 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 Tendermint²⁷ 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. Bhaskaran et al²⁸ described the design of smart contracts for consent-driven and double-blind data sharing on the Hyperledger Fabric blockchain platform²⁹ 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.³⁰ 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.

6) Smart City Data Smart Contract

This smart contract focuses on managing data from the smart cities of Santander and Fujisawa and supports the use cases. It directly communicates with other tools of M-Sec project such as encrypted data storage and off chain storage. It is an ongoing work and the final results will be described in details in the next iteration of the Deliverable D4.6 from T4.3. Additional flows are created as part of middleware services to support the interaction and integration with the rest of the platform.

²⁶ 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.

²⁷ Jae Kwon, "Tendermint: Consensus without Mining." 2014.

²⁸ 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.

²⁹ "Hyperledger Fabric," Hyperledger

³⁰ 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.





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.

1. Know Your Customer

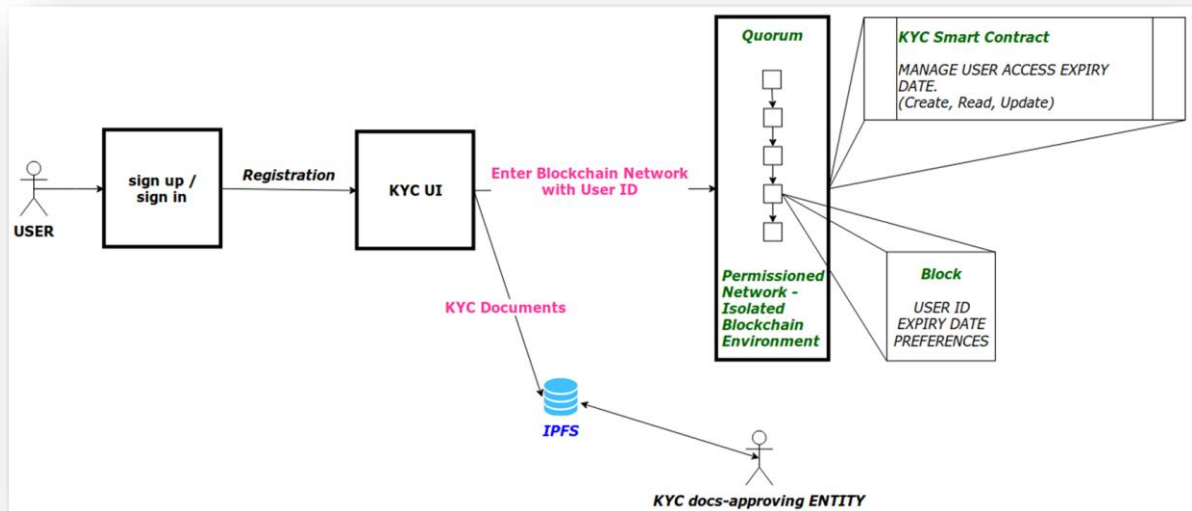


Figure 26. Overview of the KYC process for the M-Sec Platform

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 off-chain 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 26.

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.





2. 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 it³¹. 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 complementary component, settled exceptionally the SPOF problem, furnishing low latency and data distribution.

3. 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

4. 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.

³¹ Chen, Y., et al.: An improved P2P file system scheme based on IPFS and Blockchain. Big Data (Big Data), IEEE International Conference on (2017).





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.

5. 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:

- i. Title
- ii. Timestamp of start
- iii. Timestamp of end (whether applicable)
- iv. Owner/Creator name (or Address)
- v. Price in M-Sec Tokens
- vi. Description
- vii. Location (latitude and longitude)
- viii. 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:

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

6. 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.





Installation Instructions

Required Tools and dependencies

- Truffle Suite
- Solidity Programming Language
- Quorum Blockchain

Installation Guide

1. Install Truffle Suite

Truffle suite:

```
npm install truffle -g
```

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

2. Install Solidity

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

3. 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





1.7 IoT Marketplace

The goal is to create decentralized IoT ecosystems and validate their viability and sustainability. To this direction we defined and implemented a novel marketplace where smart objects can exchange information, energy and services through the use of virtual currencies allowing real-time matching of supply and demand enabling the creation of liquid markets with profitable business models of the IoT stakeholders. In this section we cover the basic technical implementation details of the M-Sec marketplace: market participants, from IoT devices to humans using mobile applications are able to exchange data and value through the M-Sec blockchain implementation.

General Description

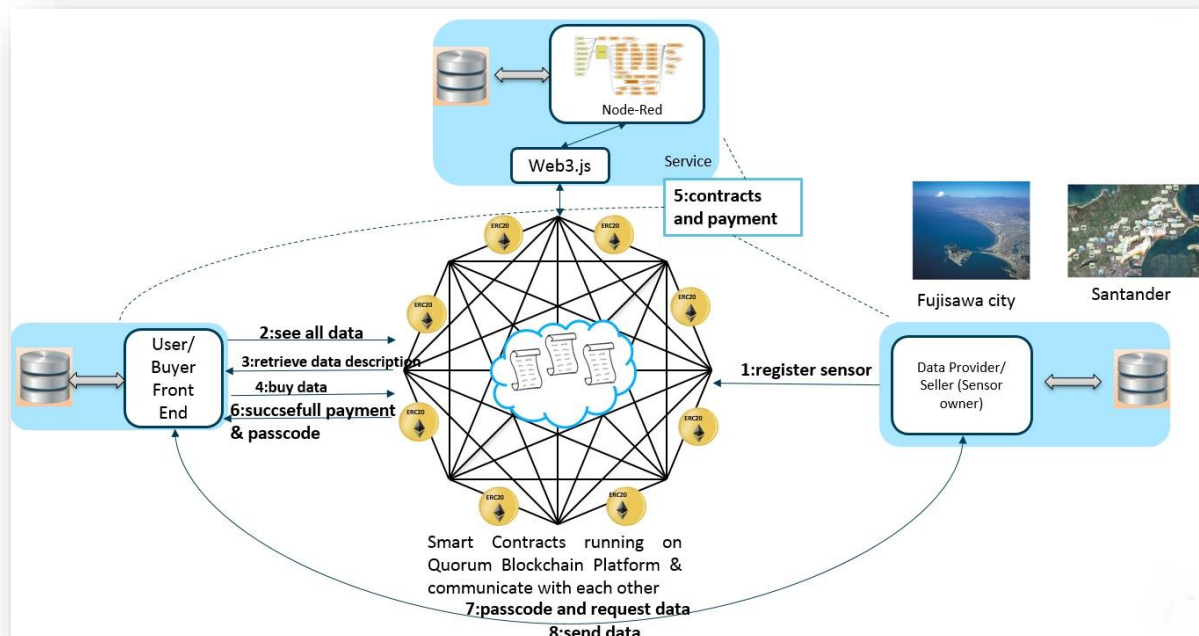


Figure 27. Overview of the M-Sec IoT Marketplace and its components

In Figure 27, we can see an overview of the developed marketplace and its components, explained in detail through for a specific example use of it.

1. The owner of a sensor/data source who wishes to make his data available for purchase or exchange register himself to the dedicated created smart contract providing information about the type of the data, their frequency, the price, the location etc.
2. A User of the M-Sec Platform who acts here as a potential buyer using our developed front end can see all the available sensors and their data
3. Upon finding some interesting data he/she can retrieve additional detailed descriptions about them and then
4. Buy the data of interest using M-Sec Tokens, which is a cryptocurrency in the form of a smart contract running in on blockchain presented in previous section





5. The deployed smart contracts communicate with each other to verify the sufficient funds of the buyer and complete the purchase by transferring funds from the balance of the buyer to the one of the data owner. The developed Node-Red flows also assist in this process connecting the different components of the system
6. In the case of successful payment, when the buyer has sufficient funds and after the tokens are transferred, a passcode is returned to the buyer necessary for accessing the purchased data
7. The buyer communicates with the platform and the API of the data owner and using the transactions details requests the data
8. The desired data is returned to the buyer in a predefined format such as JSON.



Components

Node-Red Flows

In order to orchestrate the different components and services we have used Node-Red and have developed several flows. Node-Red is a powerful visual tool for wiring together hardware devices, APIs and web-services, create flows that connect distributed components into a common IoT application³². Different flows for the different parts of the IoT Marketplace were developed.

During the development of the system we simulated the IoT weather sensors provided by public APIs and for this simulation we used an API provided by Dark Sky³³. Using Node-RED features we created flows that request current weather data for several locations from the Dark Sky API and then save these data (air temperature, relative humidity, pressure, visibility, wind speed and direction, sky cloud coverage, dew point, UV radiation and the columnar density of total atmospheric ozone layer) into a local database. We also exposed a RESTful API in order to serve the data to the users when requested. When a request is received, the API key is checked. If it is correct, the data responding to the specified time intervals is retrieved from the database and then sent to the requester.

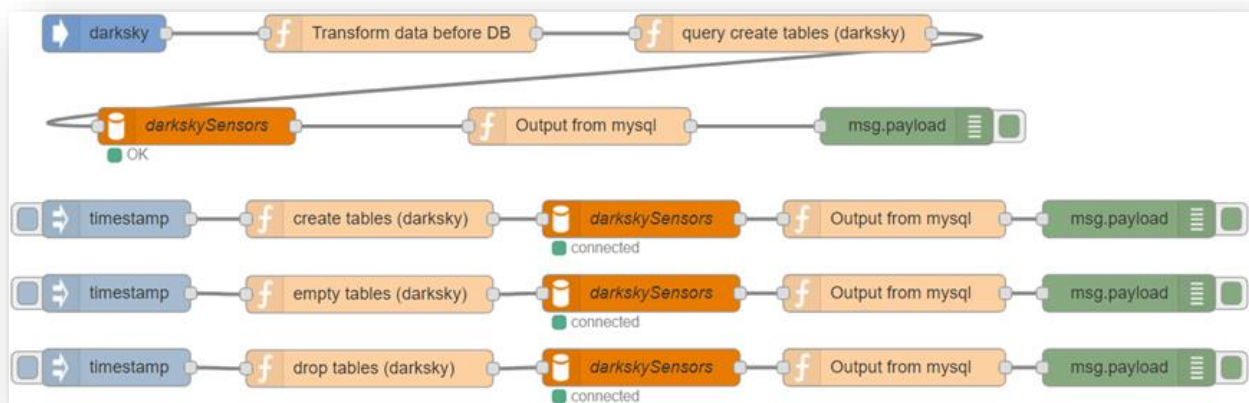


Figure 28. Node-Red Flow for the simulation of IoT sensor data

Blockchain Smart Contracts

Different smart contracts are implemented in the programming language Solidity to support the different use cases.

Some of the developed smart contracts were described in detail previously:

1. M-Sec Token: Digital cryptocurrency in the form of a smart contract in Solidity language running on blockchain.

³² <https://nodered.org/docs/>

³³ The Dark Sky Company, LLC, "Dark Sky," The Dark Sky Company, LLC, [Online]. Available: <https://darksky.net/dev>





2. Sensors Smart Contract: responsible for registering sensors and recording transactions for IoT sensor data.
3. Smart City Data Smart Contract: responsible for handling data from Smart Cities.

Web Application

This web application provides interfaces between the users and the blockchain. It provides functionalities helping users interact with the smart contracts deployed on Quorum Blockchain and access data they have bought. It also allows sending transactions to and reading data of transactions and smart contracts. It also “protects” users from misreading or mistyping info when sending a transaction.

We have used different languages and technologies to create these interfaces such as JavaScript, Bootstrap, HTML, jQuery, Nodejs. Some of the developed interfaces are described below with screenshots and details. We have used Web3.js to interact with the deployed smart contracts.

The user searches in all the available sensors registered in the Smart Contracts the sensors of interest specifying details in the corresponding fields such as the location, the type the data (temperature, starting date and time, frequency etc.), as shown in Figure 29.

The screenshot displays a web application interface for searching sensors. At the top, there's a search bar with a 'Show Search' button. Below it, a map of East Asia is shown with a red pin indicating a location. Below the map, there's a table listing available sensors.

Seller	Sensor ID	Sensor Type	Price (M-Sec Tokens)	First data at	Frequency	Map	Buy Data
0xf0f1b4ae53ae6079bf3c8da97119cf48893b7c3c	1	temperature	0.001	Wed Nov 22 2017 17:57:35 GMT+0200 (EET)	2	See map	Buy Data
0xf0f1b4ae53ae6079bf3c8da97119cf48893b7c3c	2	humidity	0.001	Wed Nov 22 2017 17:57:35 GMT+0200 (EET)	2	See map	Buy Data
0xf0f1b4ae53ae6079bf3c8da97119cf48893b7c3c	3	pressure	0.002	Wed Nov 22 2017 17:57:35 GMT+0200 (EET)	2	See map	Buy Data
0xf0f1b4ae53ae6079bf3c8da97119cf48893b7c3c	4	visibility	0.006	Wed Nov 22 2017 17:57:35 GMT+0200 (EET)	2	See map	Buy Data

Figure 29. Graphical User Interface enabling searching of sensors in the smart contracts running on blockchain

After specifying all the required information, a query is submitted to the smart contracts running on the Quorum blockchain and a list of all the available sensors is returned with information of the address of the data owner, the sensor type (temperature, pressure, visibility etc.), the frequency, a link opening a map and the option for the user to buy these data using M-Sec Tokens, as shown in Figure 30.





All the available sensors							
Show Search							
Search by location							
Seller	Sensor ID	Sensor Type	Price (NTUATok)	First data at	Frequency	Map	Buy Data
0x3898843762dcf4f8c9ae08f93e4355fdd4ed7daf	1	temperature	0.001	Wed Nov 22 2017 17:57:35 GMT+0200 (GTB Standard Time)	2	See map	Buy Data
0x3898843762dcf4f8c9ae08f93e4355fdd4ed7daf	2	humidity	0.001	Wed Nov 22 2017 17:57:35 GMT+0200 (GTB Standard Time)	2	See map	Buy Data
0x3898843762dcf4f8c9ae08f93e4355fdd4ed7daf	3	pressure	0.002	Wed Nov 22 2017 17:57:35 GMT+0200 (GTB Standard Time)	2	See map	Buy Data
0x3898843762dcf4f8c9ae08f93e4355fdd4ed7daf	4	visibility	0.006	Wed Nov 22 2017 17:57:35 GMT+0200 (GTB Standard Time)	2	See map	Buy Data

Figure 30. Graphical User Interface of the returned results after the query to the smart contract

An overview of the blockchain and the transactions included in each block is provided in our developed Explorer interface, as shown in Figure 31. The user is able to search for specific blocks, transactions, users, contracts and see the related activity.

All the transactions

Hide Search

Select TxHash

Tx Hash

Select block No

block No

Select Contract

Broker

Search transactions

Log Index	Tx Index	Tx Hash	Block Hash	Block Number	Contract Address (Contract Name)	Type	Event
0	0	0x7495f68c0235777110d953161ea04d11cb983c1a86d598a8e2a5ad2b3cf9ac	0xf4b7026b6205c2cf6f6ae6c0ebc48ee36fcc4e599883368fbaa8f6699a1bab2	3	0x3162447df38985e24d38cf3b67181a82b61de56c (Broker)	mined	SensorCreated
0	0	0x71fd3ea16ef8cfb20e82c805f19252b2599ef12400d230fd4fb59b150ab769	0xe97d52a0b02c4941868ef02ef21e8473a7b41feeb62687524ca842c3d1be686	4	0x3162447df38985e24d38cf3b67181a82b61de56c (Broker)	mined	SensorCreated
0	0	0xb68fec26e63dc9e7dccc8dc099f3b41484ed5c336406b76ca836634573a22	0x5ab708408359d0ba42160e421ca5f3353eb7d2ef3d31ed8d6e5890b557fc392d2	5	0x3162447df38985e24d38cf3b67181a82b61de56c (Broker)	mined	SensorCreated
0	0	0xb8c5548cc05c870a08c2aaa912f5545543e903e11cc67f5c6bc50cbd0f45cd	0xfca764fd0f2e6a63689903a418bab286dedcfa295fc3e15ef5e2c7b59af2053	6	0x3162447df38985e24d38cf3b67181a82b61de56c (Broker)	mined	SensorCreated

Figure 31. Graphical User Interface of the Explorer

Installation Instructions

Required Tools and dependencies

The following tools and dependencies are required to install and use the IoT Marketplace:

- NodeRed
- Nodejs
- MySQL
- Ethereum/Quorum Blockchain
- Nodejs and Javascript





Install NodeRed

- **Node-Red:** Node-RED is a powerful visual tool for wiring together hardware devices, APIs and web-services, create flows and connect distributed components into a common IoT application [<https://nodered.org/>].

Install Nodejs

- **Node.js:** Before installing Node-Red, a Node.js installation is required. We have installed Node.js version v8.9.3.

Install MySQL

- We used the MariaDB SQL, but any other SQL relational database can be used. Full instructions of how to install MariaDB database can be found here: <https://downloads.mariadb.org/> .

Install Front End

- **Front End:** We have developed a web front end, useful for end users of our application. It provides a Graphical User Interface
 - Based on HTML, Javascript, Vue Javascript framework and other libraries
 - Running: it is deployed on our server (and cloud servers as well) and accessible in <http://snf-755174.vm.oceanos.grnet.gr>

Install Java

- Most of the systems used are built on top of java engines so a Java distribution needs to be installed in the system before anything else.

Install Ethereum/Quorum Blockchain

Instructions are provided in the previous Section related to the Blockchain demonstrator.

Operating System

- We have tested the platform on Windows 10 and Ubuntu 18 but all of the software listed here is available in a large number of other distributions.

Okeanos

- **Okeanos:** We have deployed our Node-Red and Neo4j services to Okeanos cloud service for Greek Research and Academic Community (<https://oceanos.grnet.gr/home/>)

Licensing

Since ICCS/NTUA is a non-profit Academic Research Body, we will be releasing all related M-Sec results as open source contributions under Open Source licenses. Concretely, permissive licenses, as are not restrictive licenses and it can be used to create a proprietary good, allowing a commercial exploitation and ensuring high impact. Examples of those are: Apache, BSD, etc.





1.8 Trust & Reputation Management

General Description

Trust and Reputation (T&R) models have been proposed by many researchers 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 networks³⁴, ad-hoc ones³⁵, wireless sensor networks³⁶ or even multi-agent systems³⁷. 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-Molina³⁸ (Figure 35):

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.
3. **Selecting the most trustworthy or reputable entity** in the community providing a certain service and effectively having an interaction with it, assessing *a 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.

³⁴ 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.

³⁵ 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.

³⁶ Boukerche, L. Xu and K. El-Khatib, "Trust-based security for wireless ad hoc and sensor networks", Computer Communications 2007.

³⁷ J. Sabater and C. Sierra C, "REGRET: reputation in gregarious societies", Proceedings of the 5th International Conference on Autonomous Agents, Canada, 2001.

³⁸ S. Marti and H. Garcia-Molina, "Taxonomy of trust: categorizing P2P reputation systems", Computer Networks 2006.



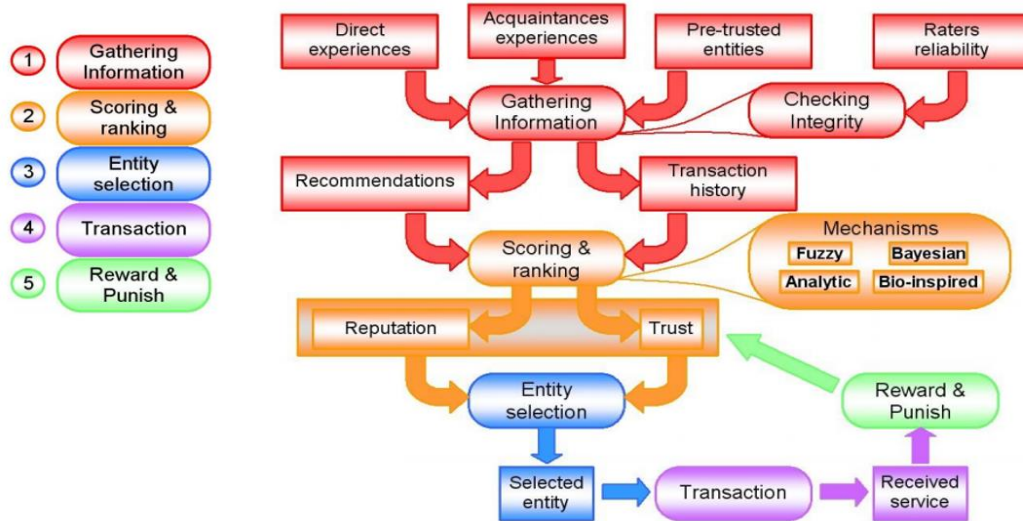


Figure 32: General steps followed in T&R models.

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.

Components

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 OASIS³⁹, **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.

³⁹ OASIS: <https://www.oasis-open.org/committees/orms>





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.

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 $D = a \cdot T + b \cdot R + c \cdot R' + d \cdot P$ where a , b , c and d (non-negative integers) are the weights of the measures and $a + b + c + d = 1$. 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). 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:

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

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:

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

Finally, we define Trust as:

$$T^k = \mu_t^k - \sigma_t^k \quad (4)$$

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.

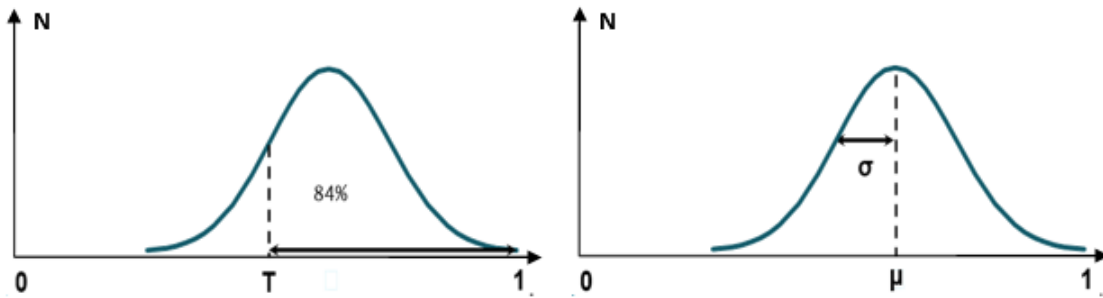


Figure 33: Calculation of the Trust Index of an actor

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



TRMSim-WSN

In order to test our T&R model, we used TRMSim-WSN⁴⁰, 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 template for the users to help them easily load new T&R models to the simulator⁴¹. 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.

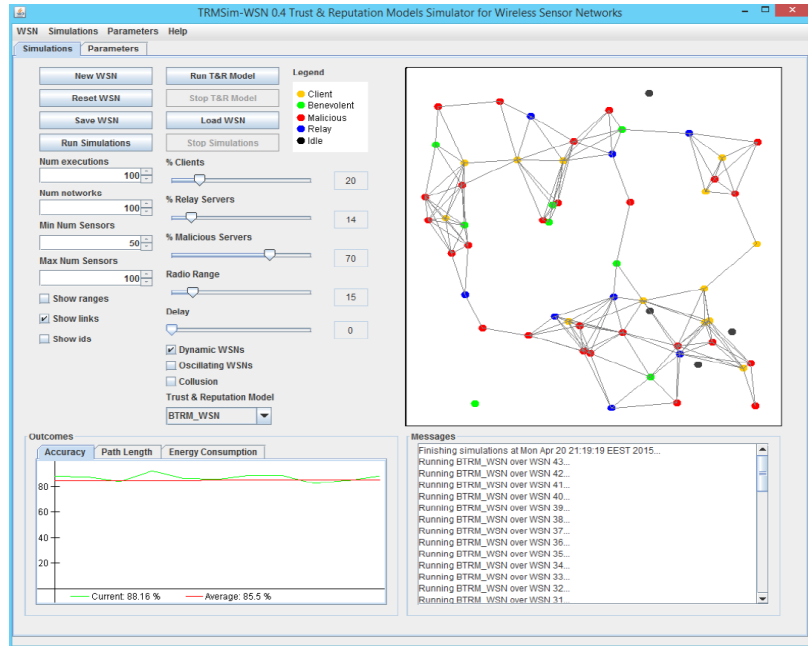


Figure 34: TRMSim-WSN.

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.

⁴⁰ F. G. Marmol and G. M. Perez, "TRMSim-WSN, Trust and Reputation Models Simulator for Wireless Sensor Networks"

⁴¹ F. G. Marmol, "Implementing and Integrating a new Trust and/or Reputation Model in TRMSim-WSN"



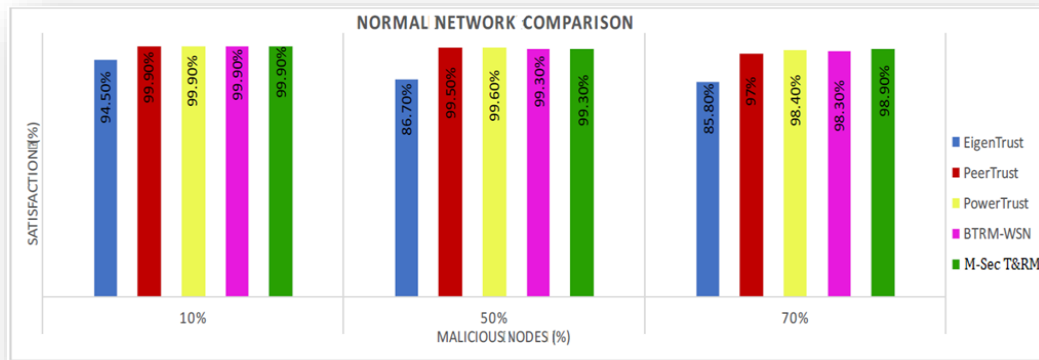


Figure 35: Normal Network Comparison

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

During Y2, the simulations for our T&R model continued, focused not only on the average satisfaction metric, but also on other important characteristics, such as scalability of the system. Once this level is completed, depending on the results, the model will be implemented as a mechanism on top of the M-Sec Blockchain and Marketplace and be tested accordingly.



Application Level Security

1.9 Crypto Companion Database (CCDB)

General Description

Since it is necessary that sensitive data stored has to be secured and private, the crypto companion database (CCDB) is proposed as a parallel system to the blockchain for the encrypted storage. The blockchain will save a hash created from the 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.

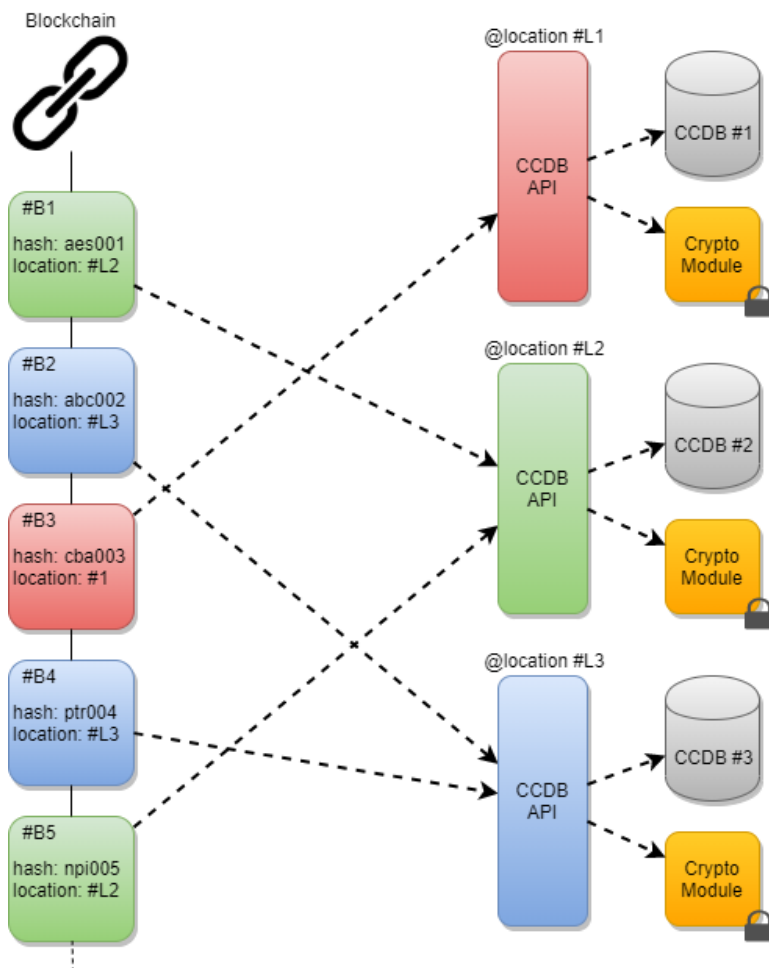


Figure 36: Access to distributed data.

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 a part of the CCBD 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 crypto companion database; 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 24-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.





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 DB**.

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 **Deliverable 4.5 section 2. Demonstration 1: Blockchain Framework and Middleware Services**.

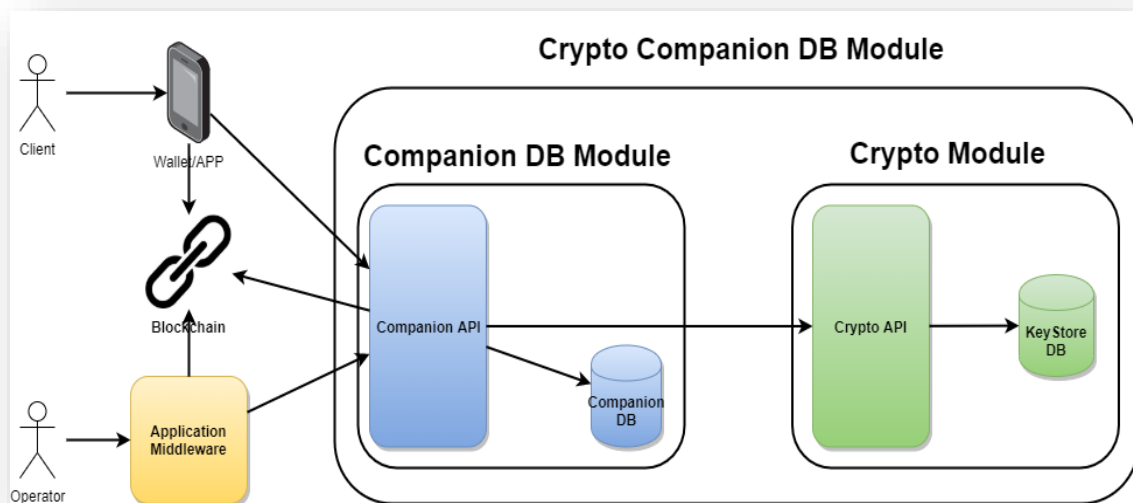


Figure 37. Crypto Companion Database Module components.

Crypto Module

This module allows a user to encrypt and decrypt data. It has two components:

- The Crypto API, that is in charge of encryption and decryption of 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 the following methods.



- A method to **create an asymmetric key pair**:

POST /crypto/enrol/{hash}

The creation of a public/private key pair can be made by scratch or by providing a 24-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 - Enrolment

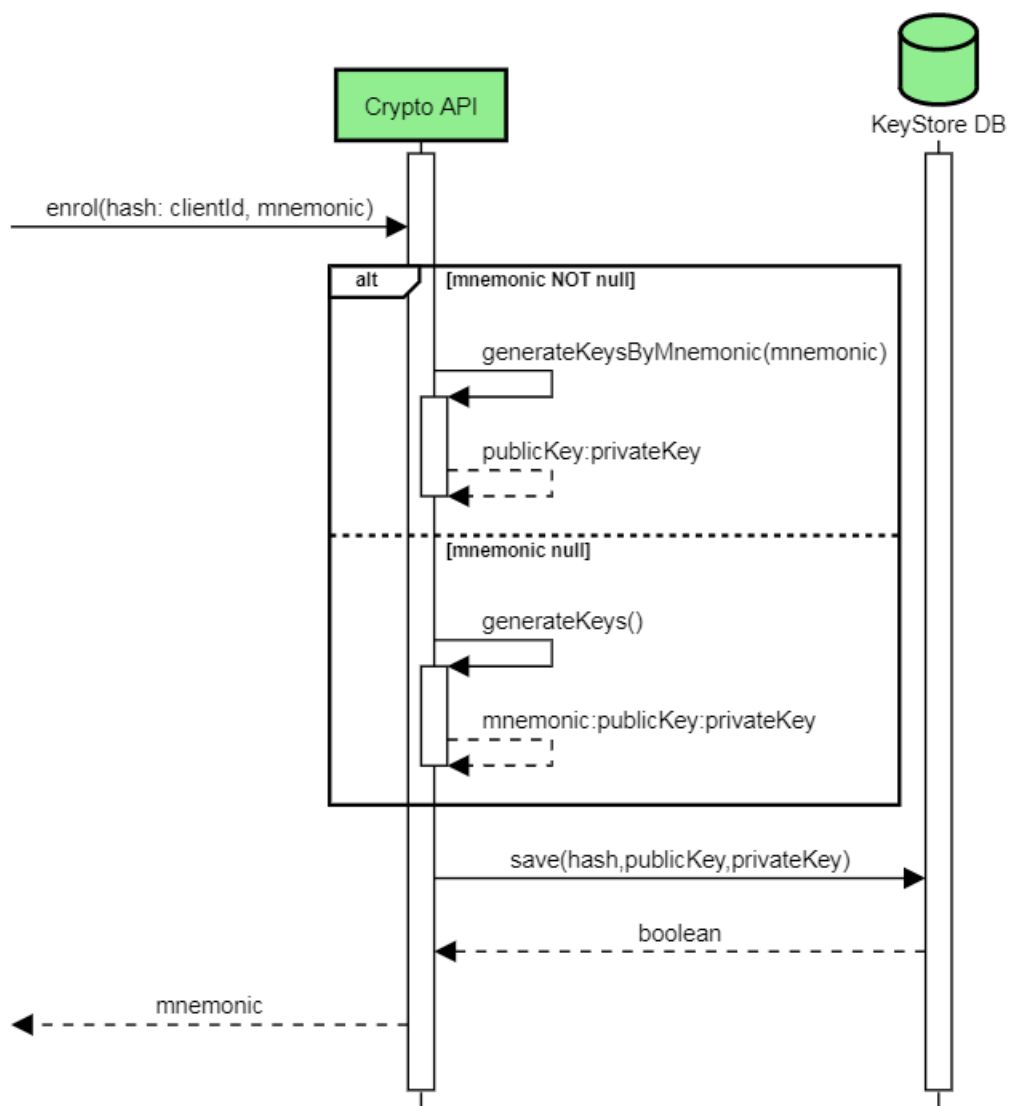


Figure 38. Sequence diagram. Enrolment in Crypto Module.



- A method to **encrypt data**:

GET /crypto/encrypt/{hash}

This endpoint will take the private key of the user with the hash provided, and encrypt the string with the data in the payload.

Crypto Module - Encrypt Data

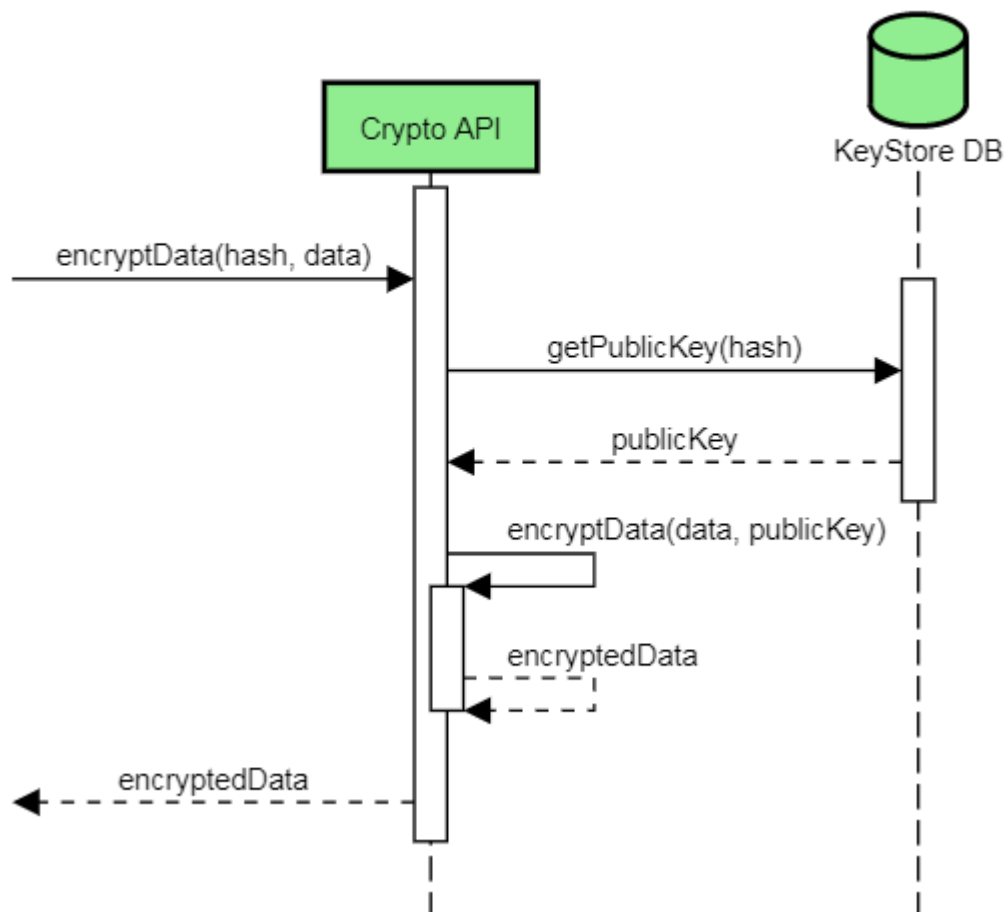


Figure 39. Sequence diagram. Data encryption in Crypto Module.



- A method to **decrypt data**:

GET /crypto/decrypt/{hash}

This endpoint will take the private key of the user with the hash provided, and decrypt the string with the data in the payload.

Crypto Module - Decrypt Data

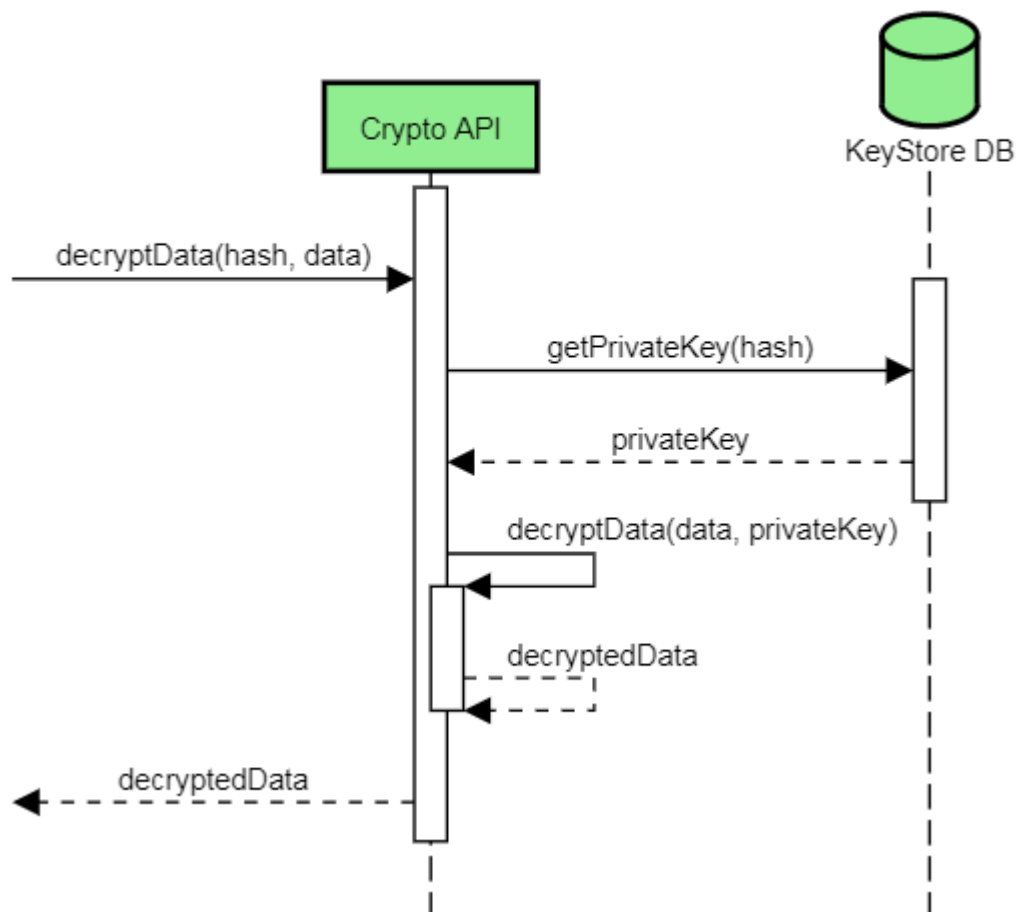


Figure 40. Sequence diagram. Data decryption in Crypto Module.



- A method to **delete the keys**:

DELETE /crypto/disenrol/{hash}

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

Crypto Module - Disenrolment

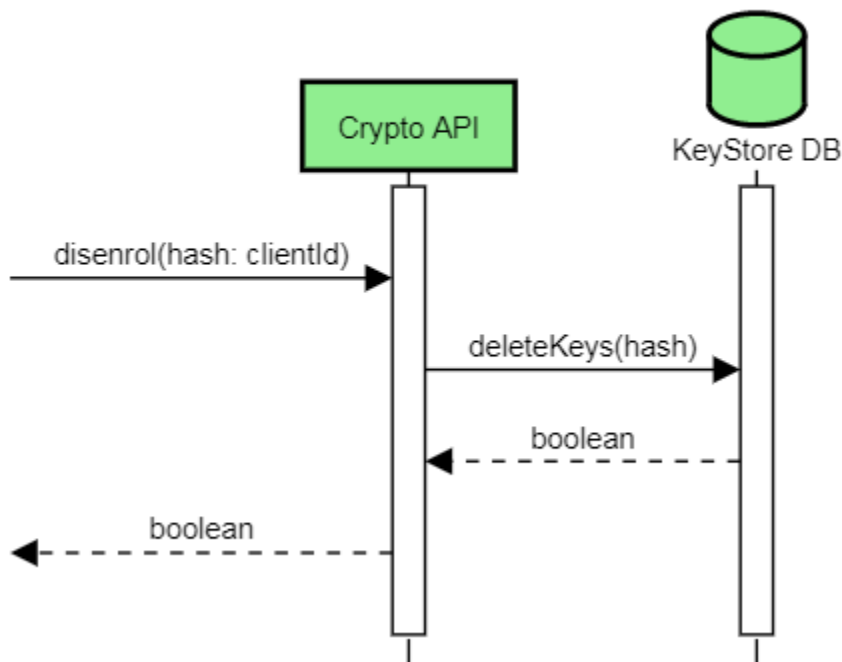


Figure 41. Sequence diagram. Disenrolment in Crypto Module.

The API of this module is intended to be private, so it does not provide any kind of security.

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 data stored in the database will look like:

- hash: 32-64 hexadecimal string.
- Public key: The Public key generated by an asymmetric key algorithm that matches the private key.
- Private key: The Private key generated by an asymmetric key algorithm that matches the public key.



Crypto Companion DB Module

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

This module has two components:

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

The Companion DB API provides the following methods.

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 42. 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/enrol



The creation of the public/private key pair can be made by scratch or by providing a 24-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.

CCDB Module - Enrolment

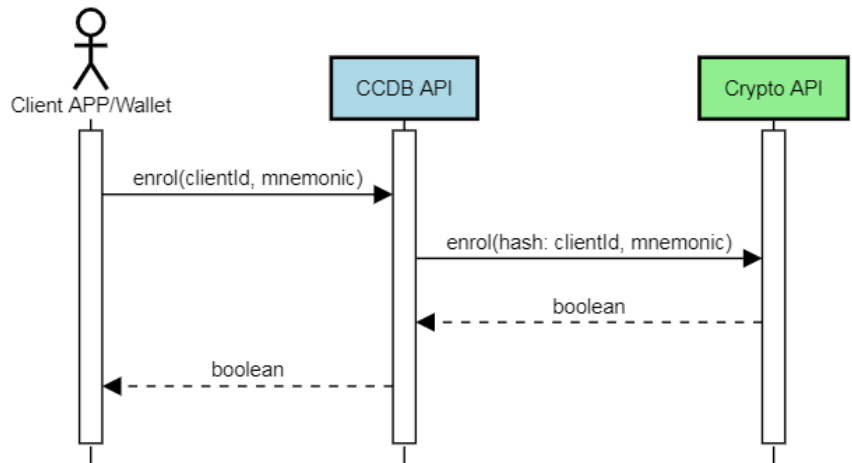


Figure 45: Sequence diagram. Enrolment in CCDB Module.

- A method to **disenroll**:

DELETE /companionDB/disenrol



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

CCDB Module - Disenrolment

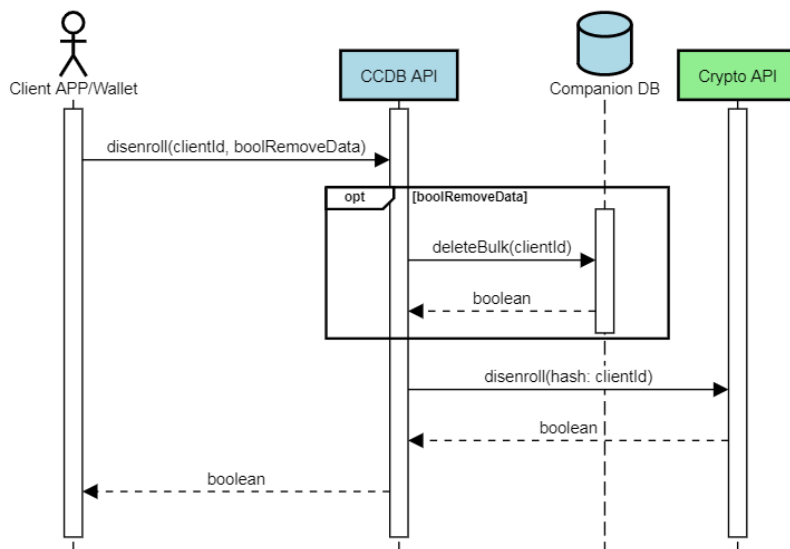
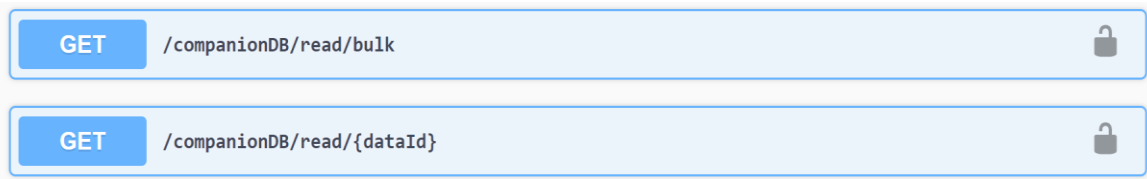


Figure 46: Sequence diagram. Disenrollment in CCDB Module.





- A method to **read data**:



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

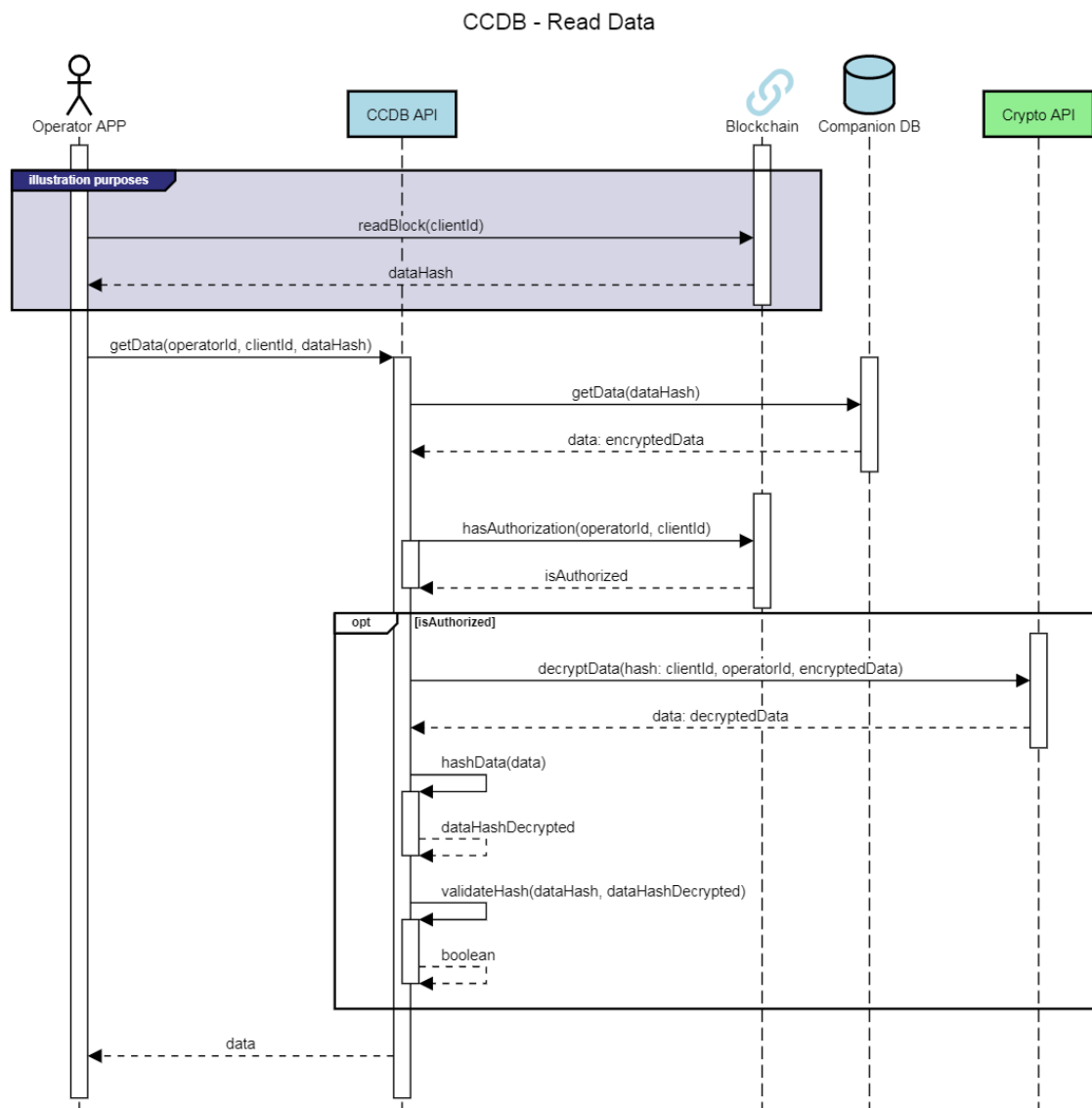


Figure 43. Sequence diagram. Read data in CCDB Module.





- A method to **save data**:

POST	/companionDB/save	🔒
POST	/companionDB/save/bulk	🔒

This endpoint will let an owner to save encrypted data.

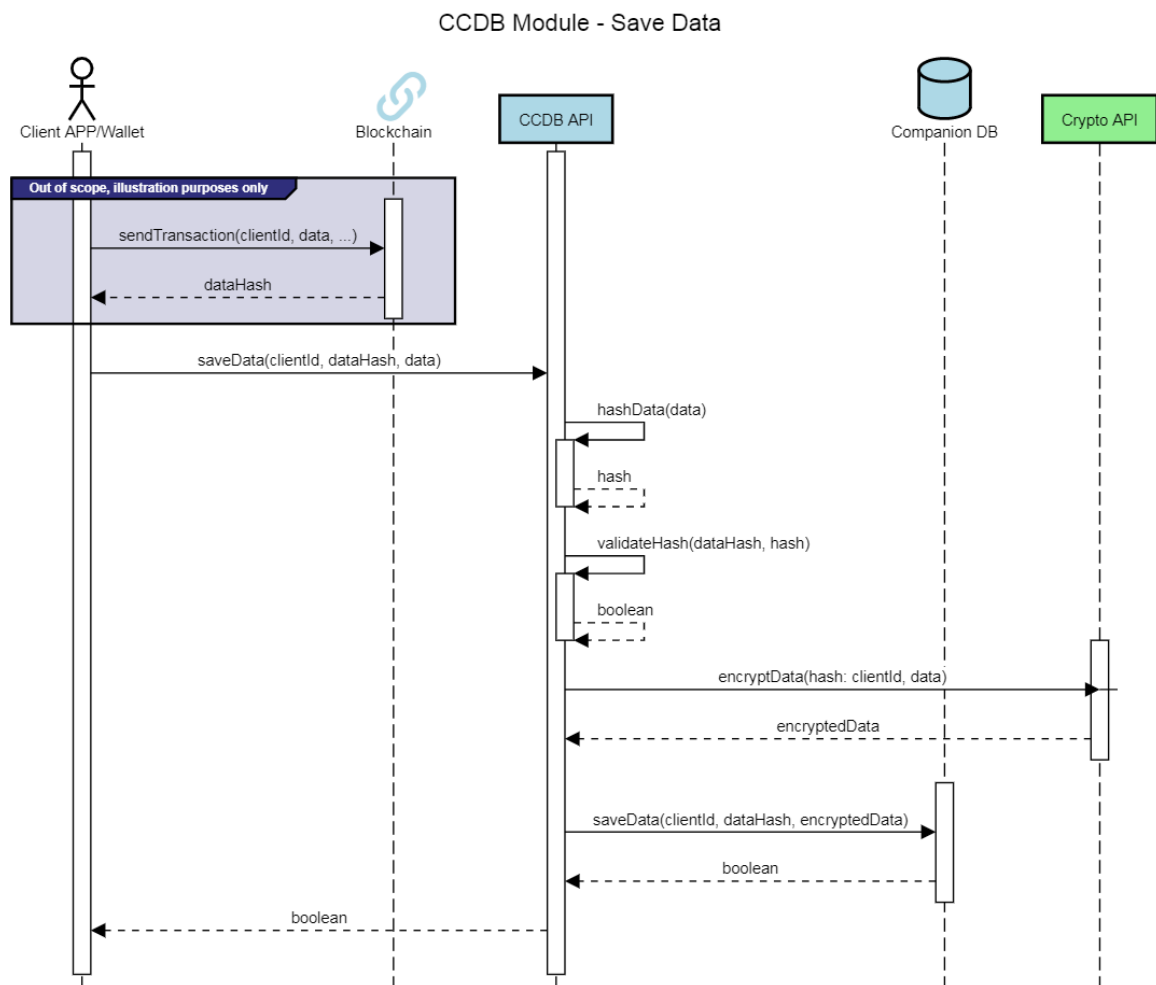


Figure 44. Sequence Diagram. Save data in CCDB Module.



- A method to **delete data**:

DELETE	/companionDB/delete/{dataId}	🔒
DELETE	/companionDB/delete/bulk	🔒

This endpoint will let the owner of the data to delete it.

CCDB Module - Delete Data

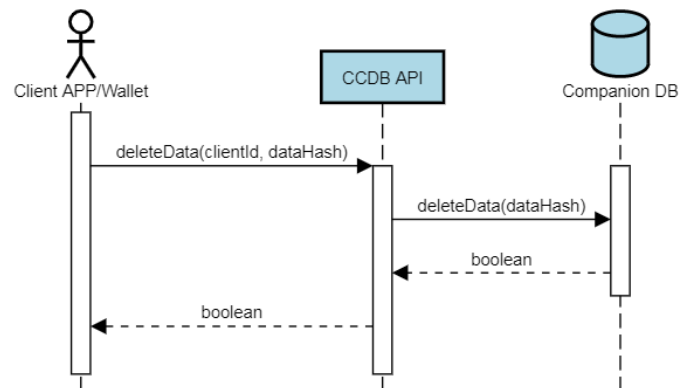


Figure 45. Sequence Diagram. Delete data in CCDB Module.

- A method to **authorize a user**:

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

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

CCDB Module - Authorise

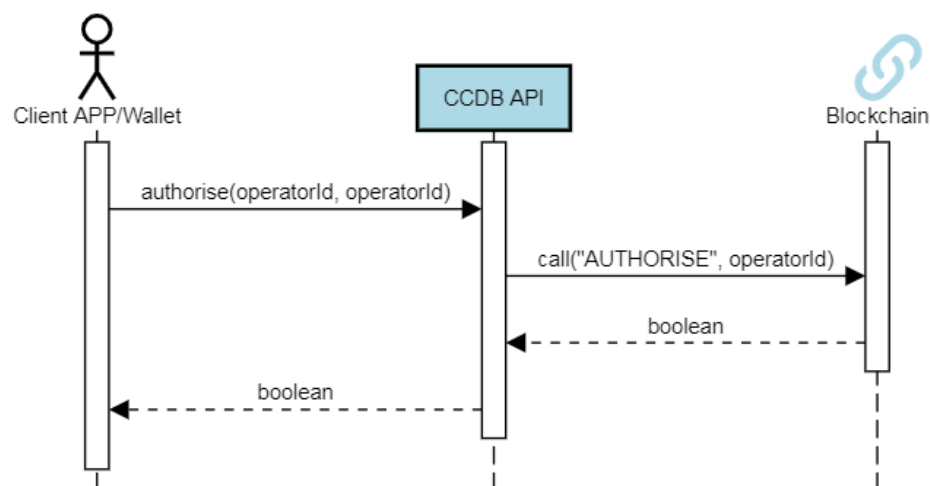


Figure 46. Sequence diagram. Authorize in CCDB Module.



- A method to **deauthorize a user**:

DELETE /companionDB/deauthorise/{hash}



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

CCDB Module - Deauthorise

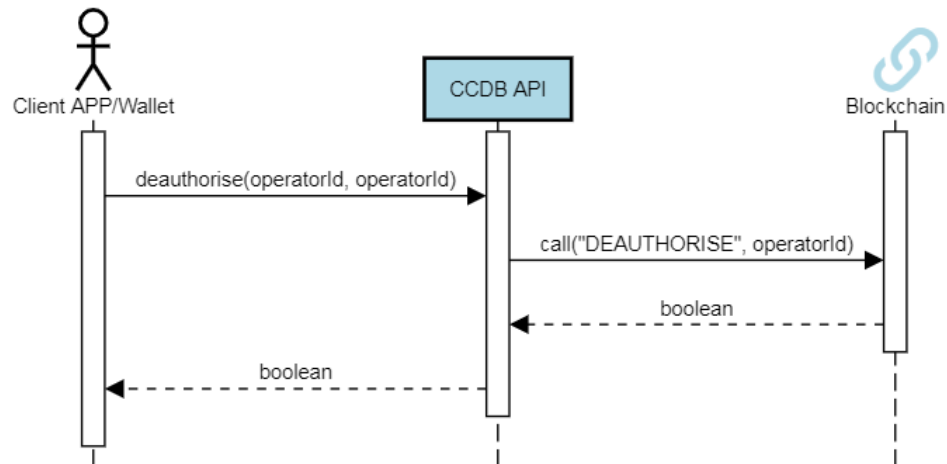


Figure 47. Sequence diagram. Deauthorise in CCDB Module.

- A method to **request authorization to a user**:

POST /companionDB/requestAuthorisation/{hash}



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

CCDB Module - Request Authorisation

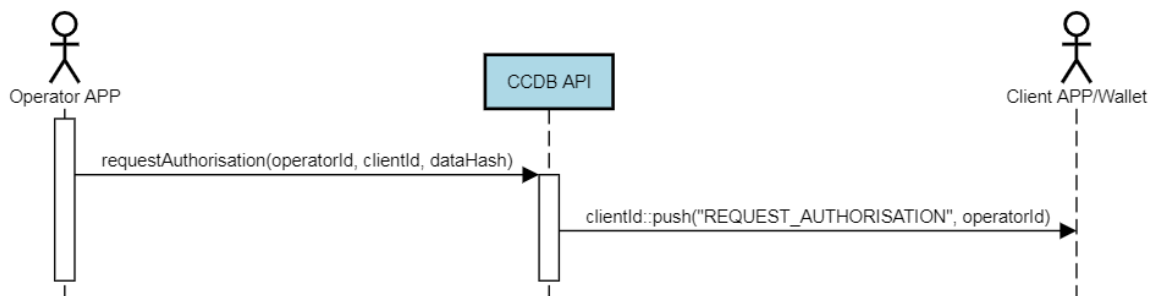


Figure 48. Sequence diagram. Request authorization in CCDB Module.





Installation Instructions

The following section provides a list of the tools and dependencies needed for the correct operation of the crypto companion database. It also provides a set of steps to install them as well as the demonstration itself. All installation processes have been taken from the official websites of the products and modified according to the needs of the demonstrator.

Required Tools and dependencies

The Following is the 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 a problem.

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/>.

Install Mongo DB

As docker has been 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/>.

Install Git

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

Download and Run the 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 following.

- **Authentication API**

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 49. Authentication API in Crypto Companion Database Module.





- **Data Management API**

POST	/companionDB/enrol	🔒
DELETE	/companionDB/disenrol	🔒
GET	/companionDB/read/bulk	🔒
GET	/companionDB/read/{dataId}	🔒
POST	/companionDB/save	🔒
POST	/companionDB/save/bulk	🔒
DELETE	/companionDB/delete/{dataId}	🔒
DELETE	/companionDB/delete/bulk	🔒
POST	/companionDB/authorise/{hash}	🔒
DELETE	/companionDB/deauthorise/{hash}	🔒
POST	/companionDB/requestAuthorisation/{hash}	🔒

Figure 50. Management API in Crypto Companion Database Module.

The Swagger UI provides enough information to let the developer know how to use this API.

Licensing

Licensing for all the components/software used:

- Docker is under Apache License 2.0 (<https://www.apache.org/licenses/LICENSE-2.0>) for 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>)





1.10 Demonstration 2 – Security-analysis-tool

General Description

It is crucial for secure software development to use various types of security knowledge. NII provides security requirements modelling support system (Security analysis tool) for a misuse case diagram that enables the association of security knowledge with elements that constitute the diagram.

The functions provided by the Security analysis tool are the following.

1. to draw diagrams
2. to associate security knowledge with elements of a diagram
3. to browse security knowledge in the knowledge base
4. review function
5. artifact management

The following diagram shows an overall screenshot of the Security Analysis Tool.

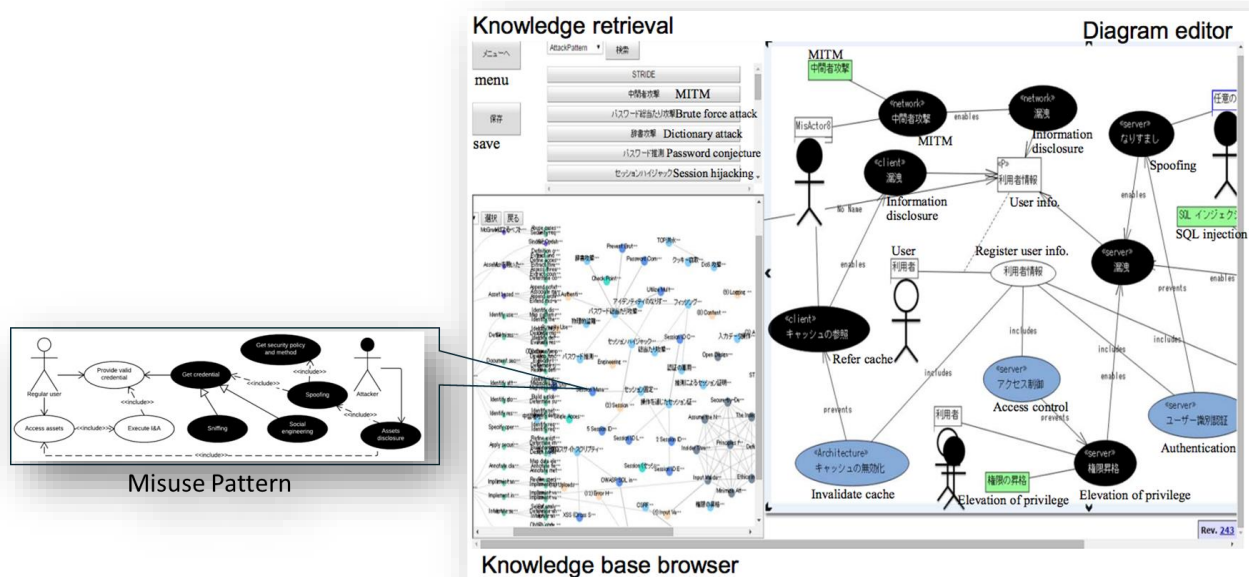


Figure 51. Screenshot of the Security analysis tool.

On the right side of this figure is an editor that creates a misuse case diagram. The lower left side of this figure represents the entire structure of the knowledge base. When a node is selected, the detailed information regarding the knowledge is shown in a sub-window.



Components

Software Security Knowledge Base

Software security knowledge base consists of several knowledge catalogues as follows:

- Principle is a statement of general security wisdom derived from experience.
- Attack pattern is developed by reasoning over large sets of software exploits.
- Guideline is a recommendation for things to do or to avoid during software development, described at the semantic level.
- Rule is a recommendation for things to do or to avoid during software development, described at the syntactic level.
- Vulnerability is the result of an analysis against a software that an attacker can use to gain illegal access to – or negatively affect the security of – a computer system.
- Exploit is a particular instance of an attack on a computer system that leverages a specific vulnerability or set of vulnerabilities.
- A historical risk is a risk identified in the course of an actual software development effort.
- Process/Methodology/Standard is a process, methodology, or standard regarding the development of secure software.
- Component is an element that consists of the aforementioned process, methodology, or standard. For example, the CLASP process consists of “Activity,” and/or “Sub-activity”.
- Security pattern is a solution to the recurring problem for security.

Software security knowledge base forms a highly complex graph structure, in which knowledge is represented by links. The knowledge base is visually represented in Figure 52. There are fourteen types of knowledge, and knowledge instances are identified by colours allocated to each knowledge type. When a node (knowledge instance) is selected, the detailed information is shown as in the bottom right side of Figure 53.

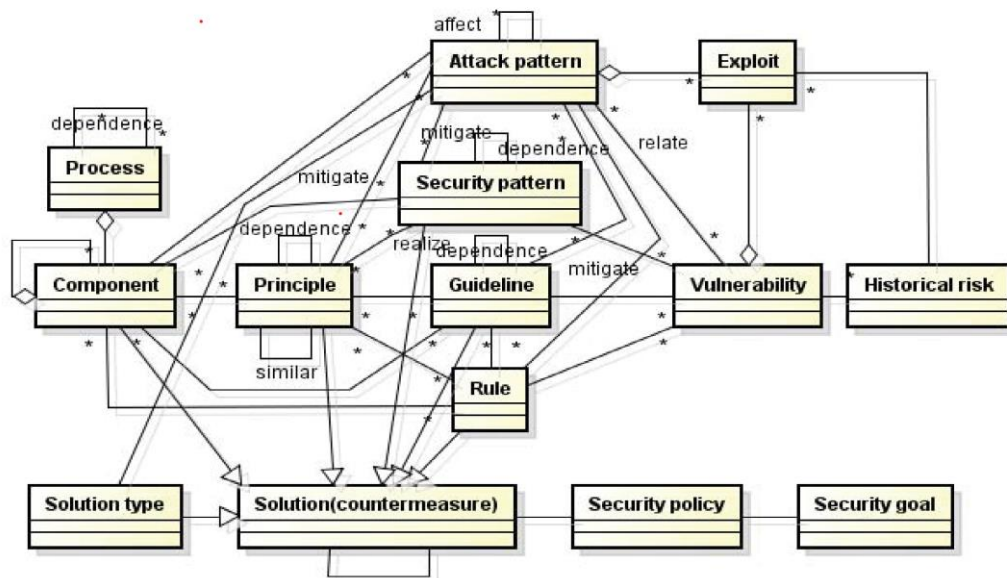


Figure 52. Conceptual model of software security knowledge.

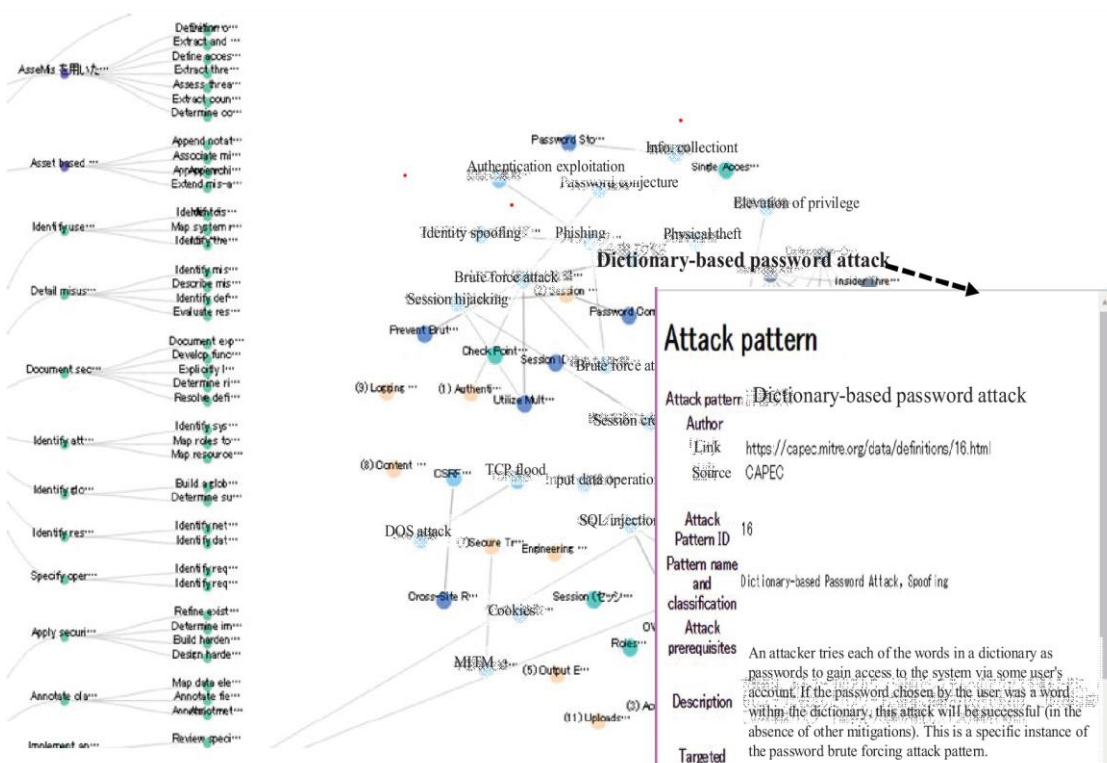


Figure 53. Visualized software security knowledge base.





We describe the operations for registering knowledge in the software security knowledge base system and associating the registered knowledge.

For example, we create “Spoofing” information in Threat Type knowledge catalogue: “Microsoft STRIDE”. And we create “Identity Spoofing” in “Attack Pattern” knowledge catalogue: “Common Attack Pattern Enumeration and Classification (CAPEC)” in the same way. We associate “Spoofing” information in Threat Type knowledge with “Identify Spoofing” in Attack Pattern knowledge.

1. At first, select “Threat Type” in the Top page of Software Security Knowledge Base.

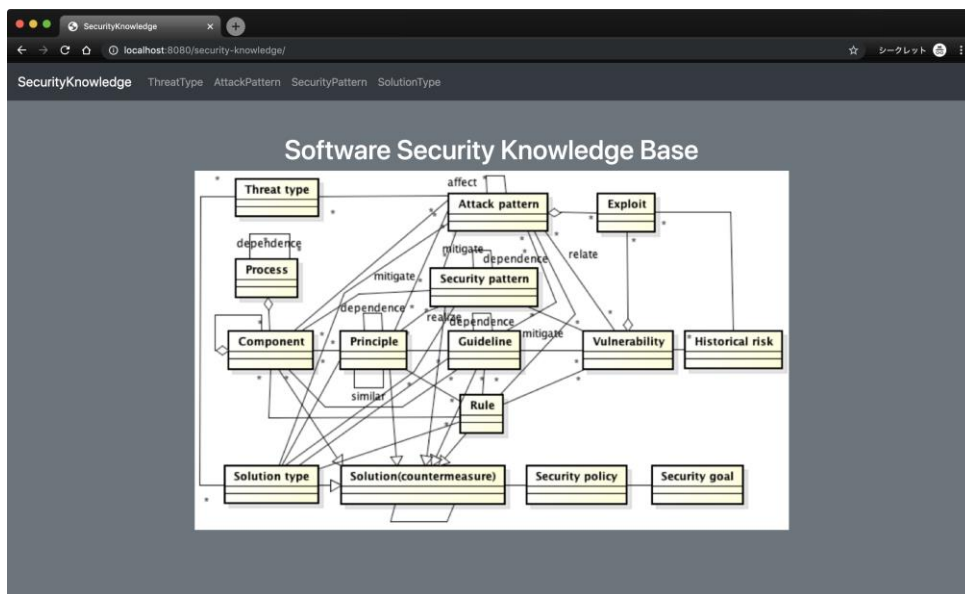


Figure 54. Top page of Software Security Knowledge Base system

2. Click “create” button and fill the Spoofing information on the registration form.

Figure 55. Threat Type Screen





SecurityKnowledge ThreatType AttackPattern SecurityPattern SolutionType

Threat Type

名前
Spoofing

説明
他のユーザーの認証情報 (ユーザー名、パスワードなど) に不正にアクセスし、それを使用する行為などです

create

Figure 56. Threat Type registration form

3. Next, create “Attack pattern” knowledge in the same way.

SecurityKnowledge ThreatType AttackPattern SecurityPattern SolutionType

Attack Pattern

パターン名
Identify Spoofing

説明
Identity Spoofing refers to the action of assuming (i.e., taking on) the identity of some other entity (human or non-human) and then using that identity to accomplish a goal. An adversary may craft messages that appear to come from a different principle or use stolen / spoofed authentication credentials. Alternatively, an adversary may intercept a message from a legitimate sender and attempt to make it look like the message comes from them without changing its content. The latter form of this attack can be used to hijack credentials from legitimate users. Identity Spoofing attacks need not be limited to transmitted messages - any resource that is associated with an identity (for example, a file with a signature) can be the target of an attack where the adversary attempts to change the apparent identity. This attack differs from Content Spoofing attacks where the adversary does not wish to change the

Figure 57. Registered Identify Spoofing in Attack Pattern knowledge

4. At last, we associate Threat Type knowledge with Attack pattern knowledge.

1) Select “*Spoofing*” from Threat Type List and Click “*update*” button.



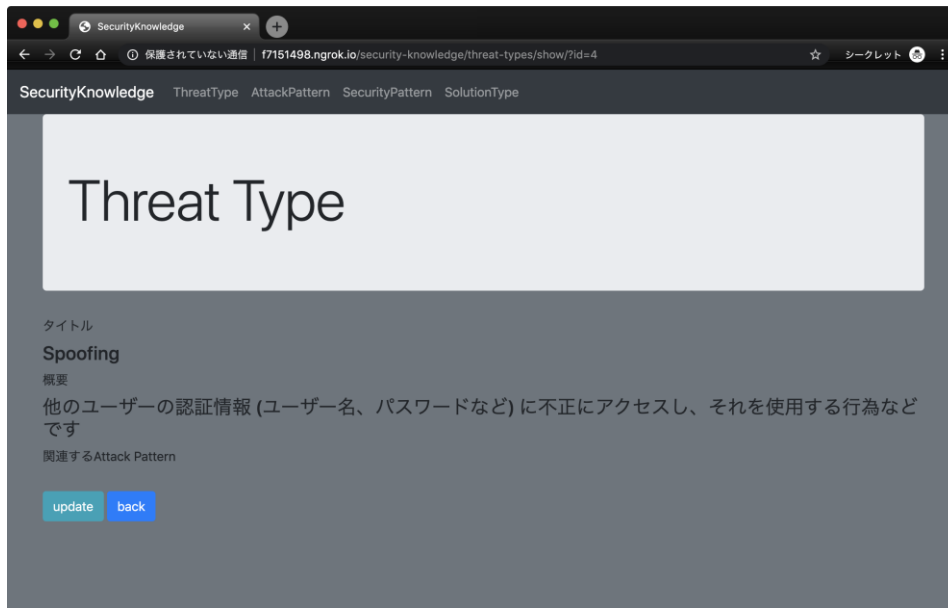


Figure 58. Spoofing information in Threat Type Screen

2) Chose “Identify Spoofing” from pull-down menu.

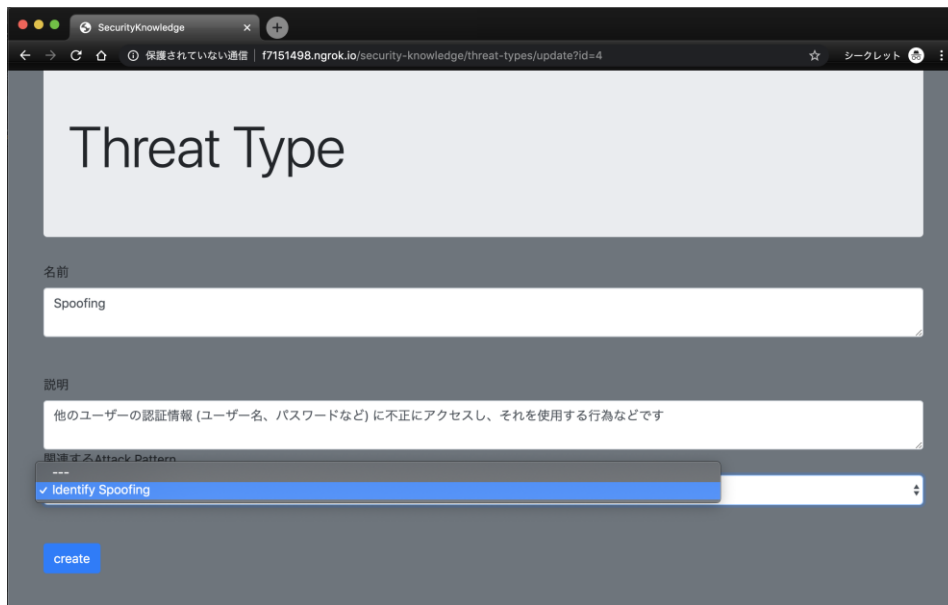


Figure 59. Edit form in Threat Type screen

3) “Spoofing” in Threat Type knowledge is associated with “Identity Spoofing” in Attack pattern knowledge.



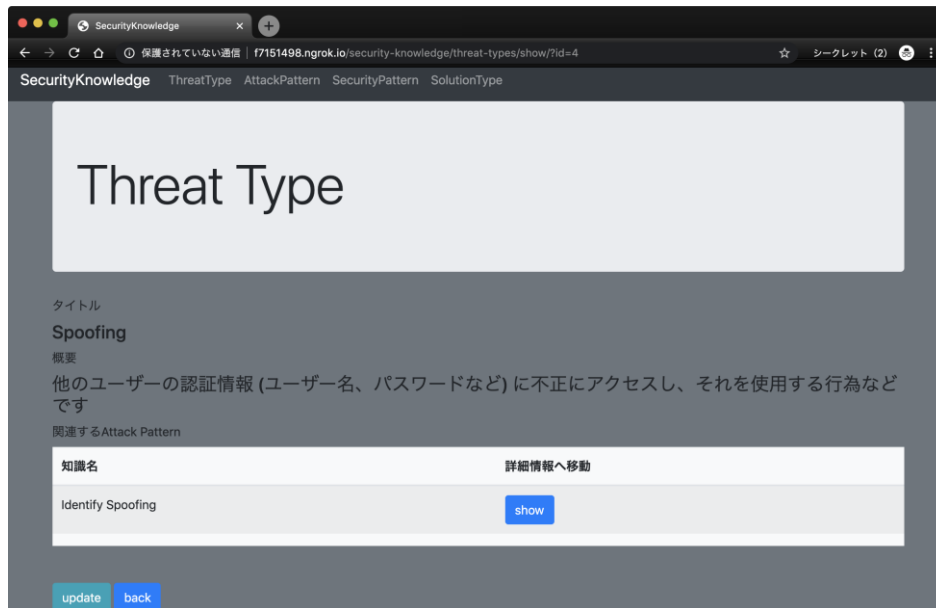


Figure 60. Association of Treat Type knowledge “Spoofing” and Attack pattern knowledge “Identity Spoofing”

Installation Instructions

The following section provides a list of the tools and dependencies for the correct operation of the Security analysis tool.

Required Tools and dependencies

The following tools and dependencies are required to use Security analysis tool:

The security analysis tool is under development and subject to change the following tools and dependencies.

- Docker
- MySQL

Install Docker

The installation can be found in the Docker’s webpage <https://docs.docker.com/v17.09/engine/installation/>, but in the section “Install Docker” in the “Crypto Companion Database (CCDB)” there is a list of the main steps and commands for Windows and Ubuntu.

Install MySQL

The installation can be found very detailed in MySQL’s webpage.

For Windows: <https://dev.mysql.com/doc/mysql-installation-excerpt/5.5/en/windows-installation.html>

For Linux: <https://dev.mysql.com/doc/mysql-installation-excerpt/5.5/en/linux-installation.html>

Download and Run Demonstrator

The security analysis tool is under development and it is not currently available to download and run.





A step-by-step guide will be provided after the Security analysis tool is completed.

User Manual

The security analysis tool is under development and some of its functionalities may change in the near future, so a user manual is not provided yet.

The user manual will be provided after the Security analysis tool is completed.

Licensing

Licensing for all the components/software used:

- Docker is under Apache License 2.0 (<https://www.apache.org/licenses/LICENSE-2.0>) for more detail, go to <https://www.docker.com/legal/components-licenses>.
- MySQL (GPL) is under GNU General Public License version 2.0 (<https://opensource.org/licenses/GPL-2.0>)





End-to-End Security

1.11 Security Manager

General Description

The security Manager is a set of centralized security functions that are necessary to ensure end-to-end security, privacy and therefore digital trust. It is designed to support several security functionalities aggregated in a single backend using the LDAP standard, as described in Figure 61.

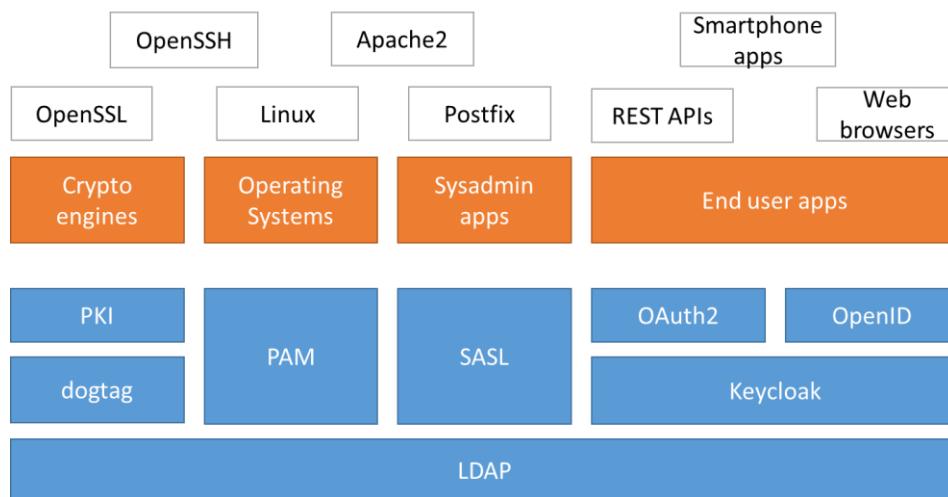


Figure 61. Stack for the M-Sec "Security Manager" with components described in blue

In terms of M-Sec implementation, this security manager is planned to be integrated into the project as follows:

- Within IoT devices in T4.1 for provisioning devices with initial manufacturer credential for firmware authentication
- In T4.2 between IoT devices and the cloud for communication authentication and encryption. This is most likely an encrypted channel between IoT devices and M-Sec middleware such as SOXFire and sensiNact. The initial steps of this integration are described in the next demonstrators.
- In T4.3 to provide marketplace authentication and authentication coordination between SOXFire/sensiNact and the blockchain implementation
- In applications within T4.4 in order to authenticate end-users in the system.



Components

Directory service

The central element for the security manager is a directory service containing all information to manage security services for clients, such services known as AAA for Authentication, Accounting and Authorization. In the implementation form, we use an LDAP directory as it is commonly accepted as a free and open standard from the IETF.

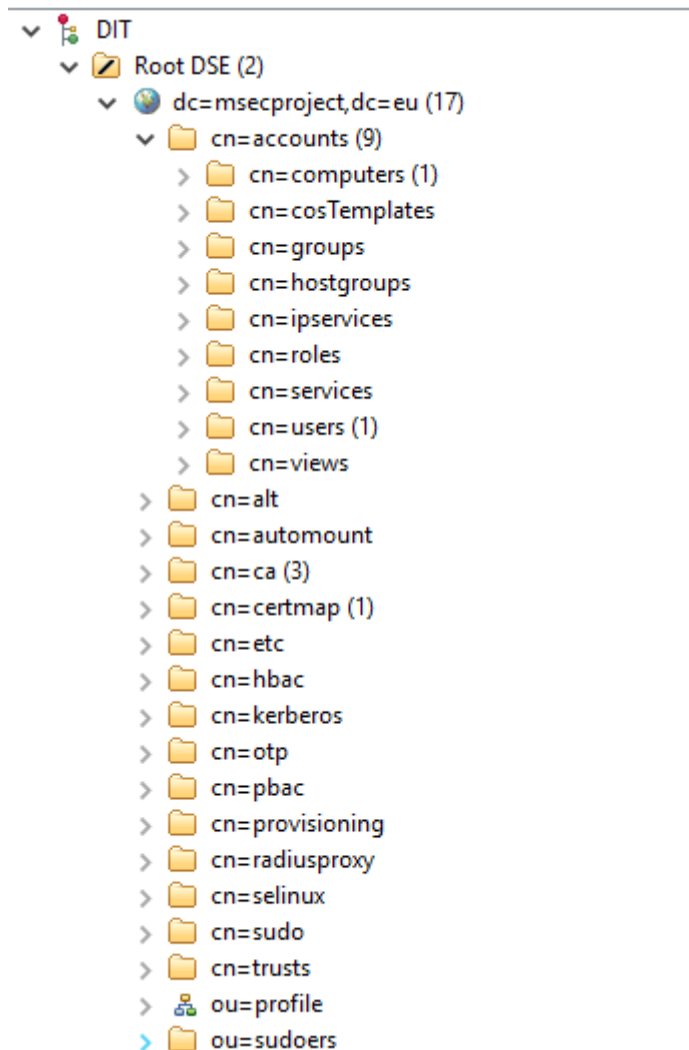


Figure 62. The first level of the directory service for the security manager

Public Key Infrastructure

Public key infrastructure is an essential component in modern communication patterns enabling asymmetric cryptography between untrusted parties. A public key infrastructure enables to bound public keys to identities (such as people or devices). It is usually managed by an authority associated with many roles such as validation authority, registering authority, etc.

We use the *dogtag* PKI solution, which has native capabilities to use LDAP as described earlier.





Installation Instructions

The demonstration in this first version aims to show how a client can talk to an endpoint with a secure channel managed by the Security Manager. It handles the configuration of the security manager, the enrolment of the user and the configuration of the endpoint.

Required Tools and dependencies

We assume that all three entities, the security manager, the client and the endpoint are running a Linux Operating System. In the demonstration, the client is a Raspberry Pi, the endpoint and the security manager are both virtual machines.

DNS

One of the first steps is to have a proper DNS configuration, even if services are running locally. To reflect this configuration, we force the name resolution in the `/etc/hosts` file as follows:

```
10.255.0.242    manager.msecproject.eu
10.255.2.168   client.msecproject.eu
10.255.2.150   endpoint.msecproject.eu
```

It is important for the security manager that these entries are placed at the very beginning of the file as only the first resolution is taken into account while resolving certificates domains.

FreeIPA

We use the FreeIPA toolkit which integrates many components, yet to be configured, such as an LDAP server, the dogtag PKI and Kerberos system. FreeIPA is available as a package on most Linux distributions. An installation script provides an interactive configuration tool.

```
apt-get install freeipa-server
/usr/sbin/ipa-server-install
```

Note that FreeIPA is not yet fully supported on Debian-based targets and require unstable packages (sid). Installation is most likely to fail and require few tweaks.

More information about installing FreeIPA is available at the following link:

https://www.freeipa.org/page/Quick_Start_Guide

Download and Run Demonstrator

The demonstrator shall be composed of three distinct machines: a device (Linux based), an IoT backend and the Security Manager. We describe here steps to deploy the security information. In this setup, we handle two kinds of flows:

- TLS encryption for HTTP based exchanges (REST)
- SSH tunnels for legacy/third party unencrypted data streams





Configuring FreelIPA

Once FreelIPA has been installed and initialized with the previous steps, one can log into the management page following the URL [https://\[domain name filled during installation\]](https://[domain name filled during installation]). Once logged in, we can manage the user on the webpage described in Figure 63.

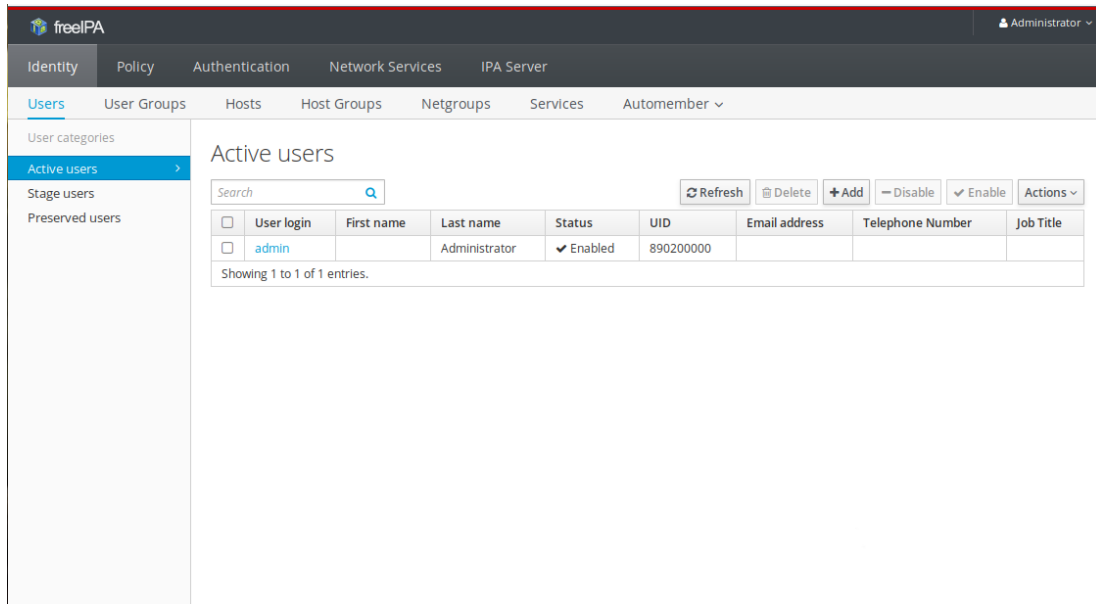


Figure 63. Users management page

From this webpage it is possible to add a user or to edit an existing user as described in Figure 64

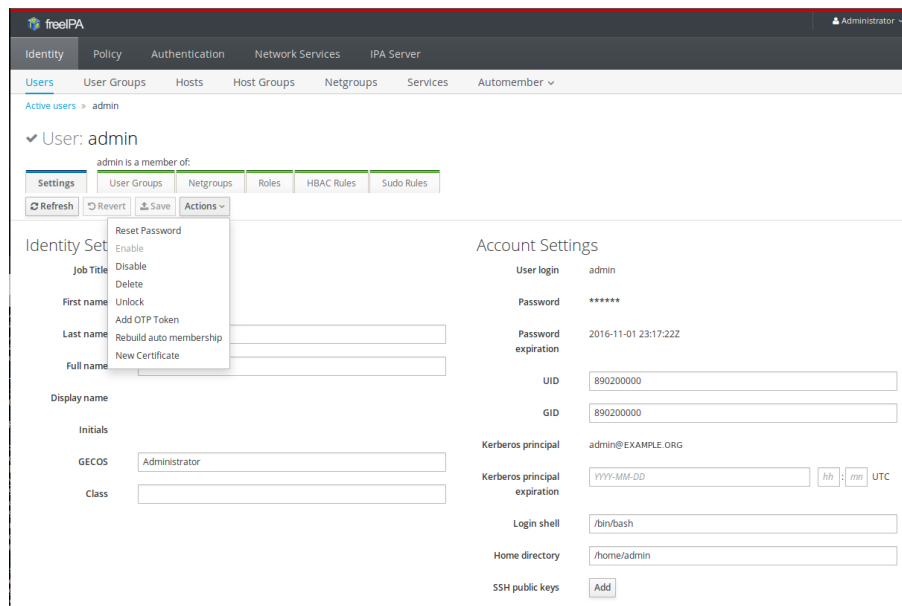


Figure 64. User edition

We assume that we have a user “demo” for the next steps. The user is just initialized and doesn’t have any SSH public key or any certificate attached to it.





In addition, we initiate a signing certificate authority for SSH. SSH uses its own key system and is not compatible with a PKI. At the end of this demo, each user will have two public keys: one for SSH and one bound to the PKI.

To generate an SSH CA, we use the following command which will generate two files: `ca.pub` as the public key and `ca` as the private key. This last key shall be stored in the safest place possible.

```
ca$ ssh-keygen -t rsa -N '' -C 'admin@msecproject.eu' -f ca
```

Configuring the Client

The client is bound to a user in FreeIPA. This user in our case is 'demo'. It will use this account bound to each outgoing connection, whether it's an SSH tunnel or a natively encrypted tunnel.

SSH Tunnel for the client

For the SSH tunnel, we need to generate a key with the FreeIPA account using the following command.

```
client$ ssh-keygen -t rsa -N '' -C 'demo@msecproject.eu' -f ~/.ssh/id_rsa
```

Then we can copy the generated file "`id_rsa.pub`" and copy it to the CA in order for the CA to sign it, making it trustful for other peers managed by this CA.

```
ca$ ssh-keygen -s ca -I 'demo@msecproject.eu' -n demo -O "clear" -O "permit-port-forwarding" id_rsa.pub
```

This command will output a file named "`id_rsa-cert.pub`" that can be inspected with the following command, especially to validate some extra options regarding the SSH capabilities (shell, X11 forwarding, etc).

```
ca$ ssh-keygen -L -f id_rsa-cert.pub
```

We can then copy back the signed certificate to the client and configure it properly to use this certificate and to trust the certification authority with the known hosts:

```
ca$ scp id_rsa-cert.pub client:~/.ssh/
ca$ echo "@cert-authority * $(cat ca.pub)" > known_hosts
ca$ scp known_hosts gateway:~/.ssh/
```

OpenSSL/Certificate management for the client

For other means than SSH, such as TLS tunnels, we can use OpenSSL to interact with the PKI. OpenSSL will generate a private key with an associated CSR (Certificate Signing Request) that will be sent to the PKI module of FreeIPA. FreeIPA will associate the public key to the user and make it available for remote parties.

We first start with the private key and CSR generation on the client using OpenSSL

```
client$ openssl req -out demo.csr -new -newkey rsa:2048 -nodes -keyout private.key
```





This command will output the private key in the file private.key (which shall be stored in the safest place possible) and the CSR in file demo.csr, containing the request as well as the public key. The content of the CSR can then be copied and associated with the user using the web interface as shown in Figure 65 and Figure 66.

The screenshot shows the 'User: demo' page in the MSEC Project Manager. The 'demo' user is a member of 'User Groups', 'Netgroups', 'Roles', 'HBAC Rules', and 'Sudo Rules'. The 'Identity Settings' section includes fields for Job Title, First name, Last name, Full name, Display name, Initials, GECOS, and Class. The 'Account Settings' section includes fields for User login, Password, Password expiration, UID, GID, Principal alias, Kerberos principal expiration, and Login shell. A 'New Certificate' button is visible in the 'Actions' menu.

Figure 65. web user page for requesting a new certificate

The screenshot shows the 'Issue new certificate for user 'demo'' dialog box. It contains the following instructions:

1. Create a certificate database or use an existing one. To create a new database:
`# certutil -N -d <database path>`
2. Create a CSR with subject `CN=<uid>,O=<realm>`, for example:
`# certutil -R -d <database path> -a -g <key size> -s 'CN=demo,O=MSECPROJECT.EU'`
3. Copy and paste the CSR (from `-----BEGIN NEW CERTIFICATE REQUEST-----` to `-----END NEW CERTIFICATE REQUEST-----`) into the text area below:

The text area contains the following CSR content:

```
-----BEGIN NEW CERTIFICATE REQUEST-----
MIIVODSR+o2R9be1AsyTYFnt5YboPwOoi3twyVbBzQnwrR5Mu3cjlH/96C+3pZdm
7MQMY7TiR8H/AFsyTpsFpg05mwv17OlgDc99YVQjGQicbQ55cdAnxQALe1+4ML6
BF/eXRi/RTMFQx7acovv2gTWY3ntnfv80izHcYUVXr/XPCcwHah8u8qDSfRAsDo
alku1pdBfAw2cGfCj+V9hxeFxbQ4DWiuRIVY6E+EFVR4N7YChhfzjk8Gnb0FAu3
YIANpADV5yZddN6cuxiaDuN78Mk+6z5WWOHsxhqzALaeLI=
-----END NEW CERTIFICATE REQUEST-----
```

Figure 66. Prompt to insert the CSR while requesting a new certificate

At this stage, the PKI has knowledge about the client but the client is not enrolled within the PKI. To do this enrolment, we need to provide the following command

```
client$ kinit demo
```





Configuring the Endpoint

SSH configuration

As for the client, the endpoint's SSH key must be signed by the CA. Assuming that the endpoint is accessible from the public network with 'endpoint' as a hostname, we can use the following command from the CA to extract the public key and to sign it.

```
ca$ ssh-keyscan -t rsa endpoint | sed "s/^[^ ]* //" > endpoint.pub
ca$ ssh-keygen -s ca -h -I endpoint endpoint.pub
```

The «*endpoint-cert.pub*» file will be generated and is to move back to the endpoint along with the CA's public key

```
ca$ scp endpoint-cert.pub endpoint:/etc/ssh/
ca$ scp ca.pub endpoint:/etc/ssh/
```

Then, we need to configure SSH on the endpoint so it recognizes and trusts the incoming SSH connections signed by the CA.

```
endpoint$ echo "HostCertificate /etc/ssh/endpoint-cert.pub" | sudo tee -a /etc/ssh/sshd_config
endpoint$ echo "TrustedUserCAKeys /etc/ssh/ca.pub" >> /etc/ssh/sshd_config
endpoint$ service ssh restart
```

OpenSSL configuration

From the endpoint, the configuration depends on the software used to collect data streams from the client. Documentation to use FreeIPA for a web-based application is proposed here: https://www.freeipa.org/page/Web_App_Authentication

Licensing

Most of the Security Manager relies on FreeIPA which is itself a combination of many open source softwares, which may be interchanged. Licenses are listed here: <https://www.freeipa.org/page/License>

We also have used some security clients on both the endpoint and the client, namely OpenSSH (BSD license) and OpenSSL. (<https://www.openssl.org/source/license.html>). Thus, other clients may be used in the future with potential commercial licenses.







Installation Instructions

SOXFire is developed based on Openfire. Therefore, SOXFire execution environment depends on Openfire's specifications. In addition, the algorithm has many variations to encrypt data, and you can choose your favorite algorithm. In this section, we describe only SOXFire.

Required Tools and dependencies

- Java
- MySQL(5.7.XX) 'XX' means any version.
- Ant
- SOXFire source code

Download and Run Demonstrator

1. Java
Download and install Java environment. Also set \$JAVA_HOME in your shell.
2. MySQL
Download and install MySQL. Then, create user and SOXfire database like this:

```
mysql> CREATE USER 'soxfire'@'localhost' IDENTIFIED BY 'YOUR_PASSWORD';  
mysql> GRANT ALL PRIVILEGES ON *.* To 'soxfire'@'localhost' WITH GRANT OPTION;  
mysql> CREATE DATABASE soxfiredb DEFAULT CHARACTER SET utf8mb4;
```

3. Ant
Install Ant.
4. SOXFire
Download SOXFire source code: http://sox.ht.sfc.keio.ac.jp/soxfire/soxfire_src-1.0.3.zip
5. Compile and start
Uncompress a downloaded zip file. Then move to build directory and build.

```
% cd soxfire_src/build  
% ant
```

You can find compiled files in soxfire_src/work and soxfire_src/target directory. To run the SOXFire server, go to target/openfire/bin directory and run openfire.sh.

```
% cd soxfire_src/target/openfire/bin  
% ./openfire.sh
```

6. Configuration
Open <http://localhost:9090> in your browser. You can see the following setup interface in Figure 68



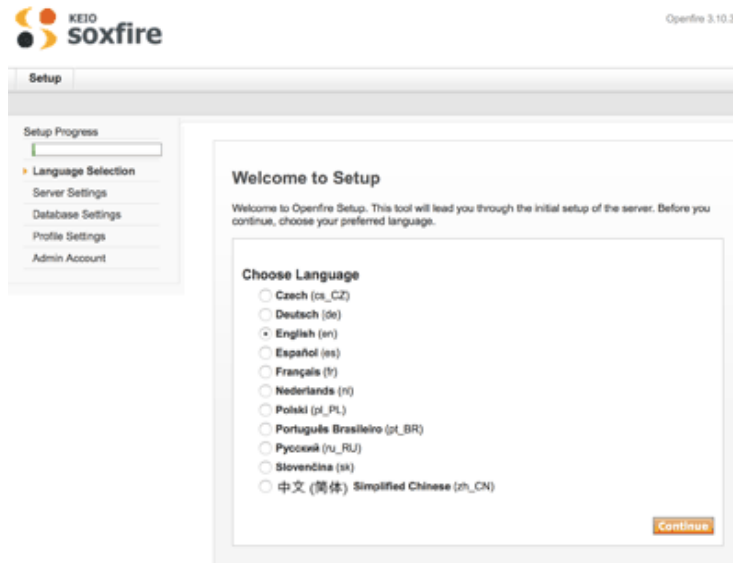


Figure 68. SOXFire setup screen

You can basically follow Openfire setup example as shown in the above link. Especially for SOXFire database, select MySQL.

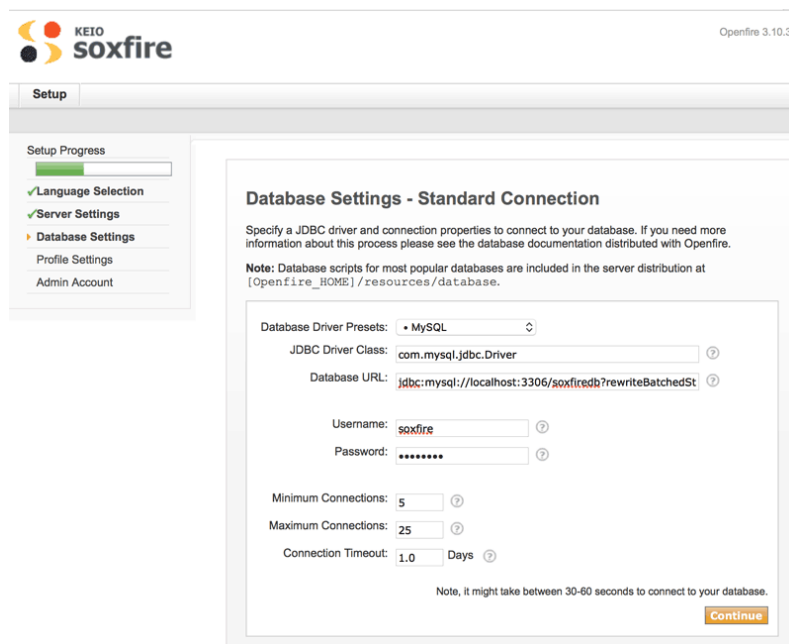


Figure 69. SOXFire database configuration

User Manual

The user manual is available on: <http://sox.ht.sfc.keio.ac.jp/soxfire/index.html>

This website shows the same content of 'Download and Run Demonstrator'.

Licensing

SOXFire license depends on Openfire license: Apache License 2.0





1.13 sensiNact - Secured IoT Middleware

General Description

The sensiNact Gateway implements the basic blocks for connectivity, service abstraction, device management, virtualization and remote access. The sensiNact Gateway allows the interconnection of different networks to achieve secured access and communication with embedded devices. The sensiNact platform defines a generic service model, described in Figure 70, and a set of generic access methods:

- the **Service provider**, **Service** and **Resource** triptychs the spine of the model:
 - the **Service provider** is attached to one location
 - the **Service** is attached to one business concept
 - and the **Resource** is attached to one business data
- A **Resource**, on which apply **Access Methods**, is a collection of **Attributes**, characterized by **Metadata**
- **Access Methods** allow reading (client/server or publish/subscribe model) to **write**, and to **actuate** when applicable

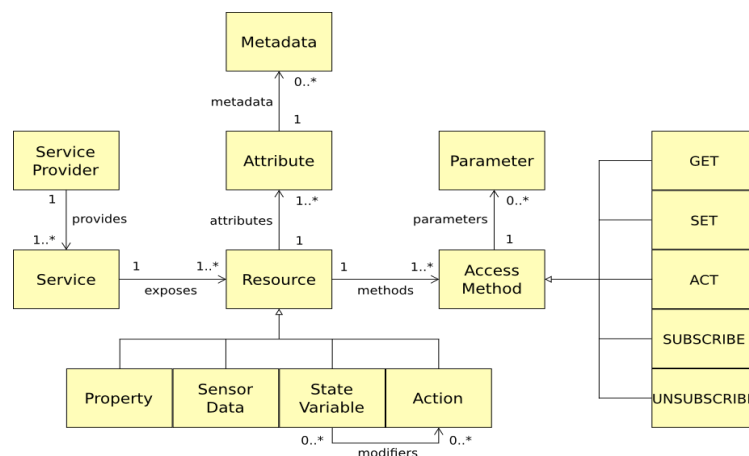


Figure 70. sensiNact service model

An example of the service model implementation is given in Figure 75.

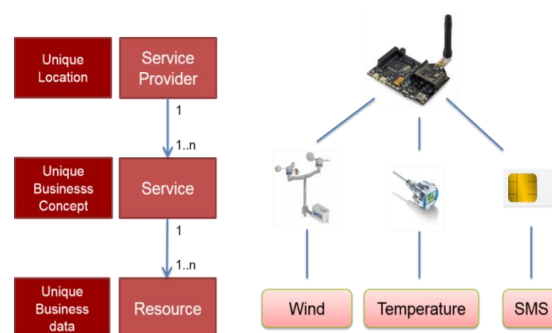


Figure 75: sensiNact service model mapping example



Components

Connectivity

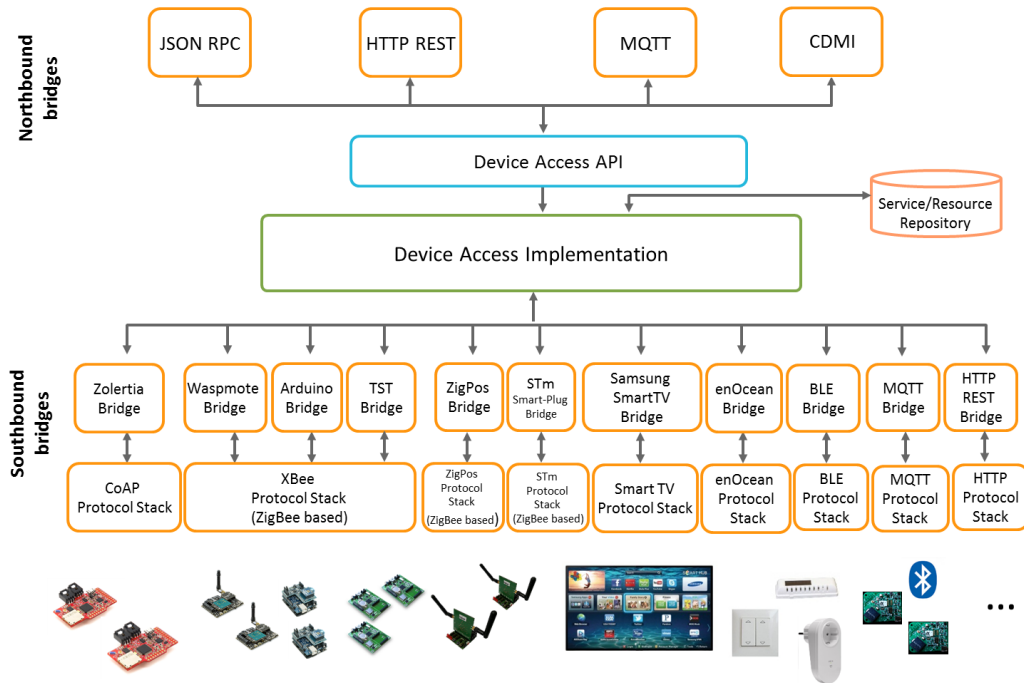


Figure 71. sensiNact gateway northbound and southbound connectivity

Figure 71 shows the southbound side the sensiNact gateway which allows to cope both with “physical device” protocols and “virtual device” ones, allowing a uniform and transparent access to an BLE network, or an HTTP Restful web service for example (pell-mell a non-exhaustive list of supported protocols: EnOcean, Bluetooth Low Energy, MQTT, CoAP, NGSI, Openhab, SoxFire, etc). On the northbound side the sensiNact gateway provides both client/server and publish/subscribe access protocols (MQTT, JSON-RPC, HTTP Restful, NGSI, CDMI, SoxFire, etc...)

Security Components

In sensiNact three sites allow providing a secured encapsulation of the data relayed by the platform as illustrated in Figure 72:

- The encryption of data coming from connected counterpart is easily implementable over each southbound bridge;
- The modules signature allows to define which module will be allowed to provide a feature, or a remote system/device connection;
- Finally an OAuth2 / OpenID intermediation service provides northbound access security

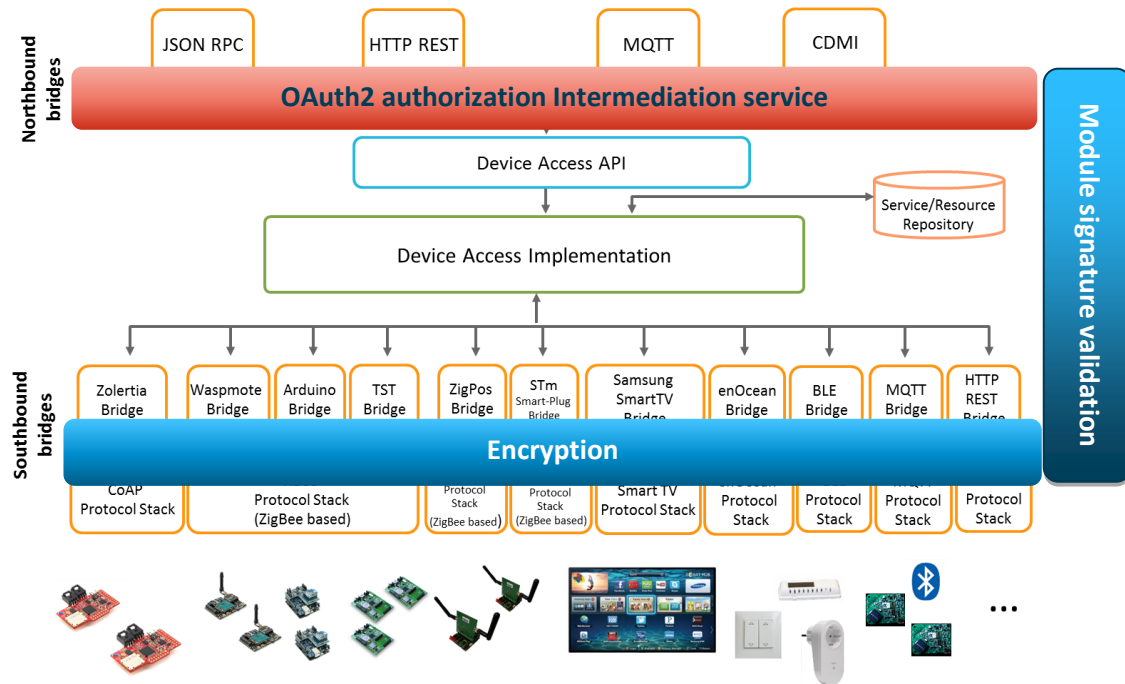


Figure 72. Secured connectivity

Encryption

The modular and generic southbound bridge template allows including a securitization process between the sensiNact and the connected counterpart, like keys sharing allowing encrypting exchange data without any extra implementation or update of the platform.

Module signature validation

The signature of the modules that compose the platform at build time permits validating that an installed module is allowed to connect to the others and to provide a new feature or a secure access to a connected counterpart. Figure 78 shows the deployment of a signed bridge within the container.

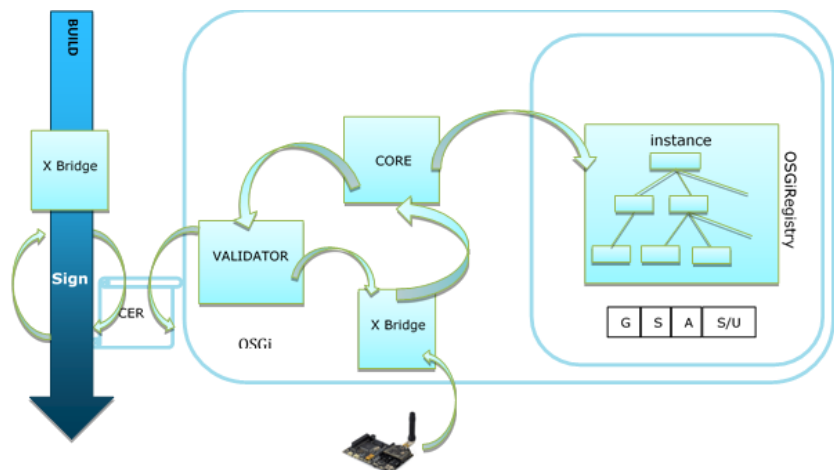


Figure 78: Sign the X bridge



OAuth2 authorization intermediation service

An oAuth2/OpenID filter, implementing for now Authorization code and Resource owner password credentials flow, offers a standard northbound secure access solution. **Figure 73** and **Figure 74** shows the authorization/verification flow.

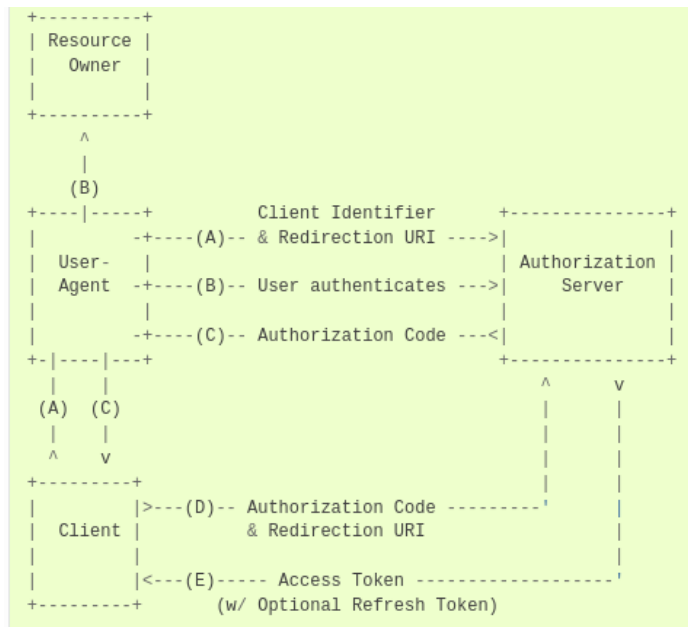


Figure 73. Authorization code flow

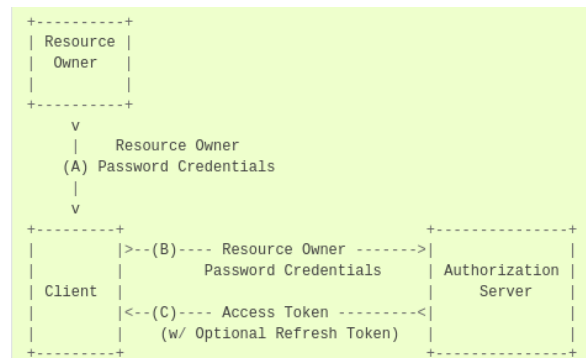


Figure 74. Resource owner password credential flow

This oAuth2/OpenID filter feature is the one presented in the demonstrator.

Installation Instructions

Required Tools and dependencies

In order to compile and deploy sensiNact, the following dependencies are required:

- Git
- Maven 3.5.3
- Java 1.8
- sensiNact source code
- Keycloak 7.0.1

Download and Run Demonstrator

JavaJDK 1.8

cf. <https://openjdk.java.net/projects/jdk8/>

When your environment has been installed define the JAVA_HOME targeting the JDK installation directory.





Git

cf. <https://git-scm.com/>

Maven

cf. <http://maven.apache.org/>

Keycloak

cf. <https://www.keycloak.org/index.html>

When the Keycloak server is installed you can ask for a standalone execution to configure it. You need now to configure client, roles, groups and users; A properly formatted JSON data structure help to proceed with the required configuration.

sensiNact

Clone the sensiNact's git source code repository:

```
git://git.eclipse.org/gitroot/sensinact/org.eclipse.sensinact.gateway.git
```

In the root directory of the newly created local repository build with maven, skipping tests to speed up the process:

```
mvn clean install -DskipTests
```

The `distribution/generator/target/sensinact` now contains a distribution that we will use for the demonstration. **Paths mentioned below refer to this location .**

Configuration

Clear the actual default configuration files directory: `rm cfigs/*`

Make sure that the script is executable: `chmod +x ./sensinact`

Create the configuration file defining the oauth2 mechanism `./cfigs/sensinact-security-oauth2.config` and define the content as below:

```
discoveryURL=http://localhost:8080/auth/realms/test/.well-known/openid-configuration
certsURL=http://localhost:8080/auth/realms/test/protocol/openid-connect/certs
client_secret=testClient
client_id=testClient
slider=admin:GET:/sensinact/slider/*
```

Configure the HTTP service provided by the felix framework in `conf/config.properties`:

```
org.osgi.service.http.port=8899
org.apache.felix.http.debug=true
org.apache.felix.http.jettyEnabled=true
org.apache.felix.http.whiteboardEnabled=true
```

Configure the OAuth2 configuration file in `conf/config.properties`:

```
org.eclipse.sensinact.security.oauth2.config=cfigs/sensinact-security-oauth2.config
```



Copy the necessary modules:

```
cp load/simulation/slider-2.0-SNAPSHOT.jar ./bundle/  
cp load/rest/*.jar ./bundle/  
cp ../../../../platform/sensinact-security/sensinact-security-oauth2/target/*.jar ./bundle/
```

You can now run sensiNact:

```
./sensinact
```

Now If you try to access to the definition of the simulated slider device using your browser (<http://localhost:8899/sensinact/slider>), you will be asked to authenticate, and if you do as adminTester (credentials adminTester:adminTester) that is the only one allowed to access to the slider, you will see the description of the targeted device. Figure 75 and Figure 76 shows the interfaces corresponding to those steps.

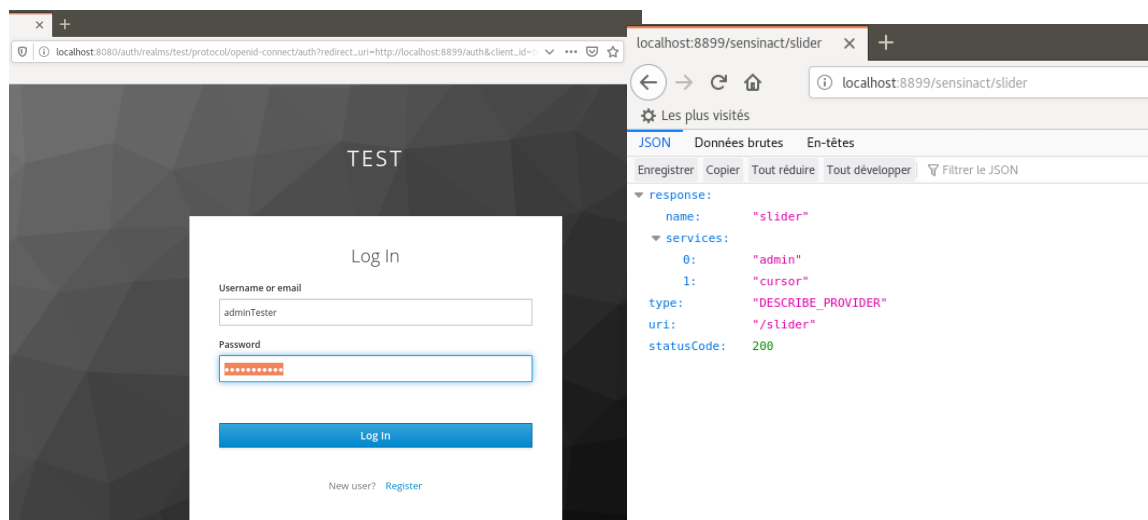


Figure 75. Login as adminTester



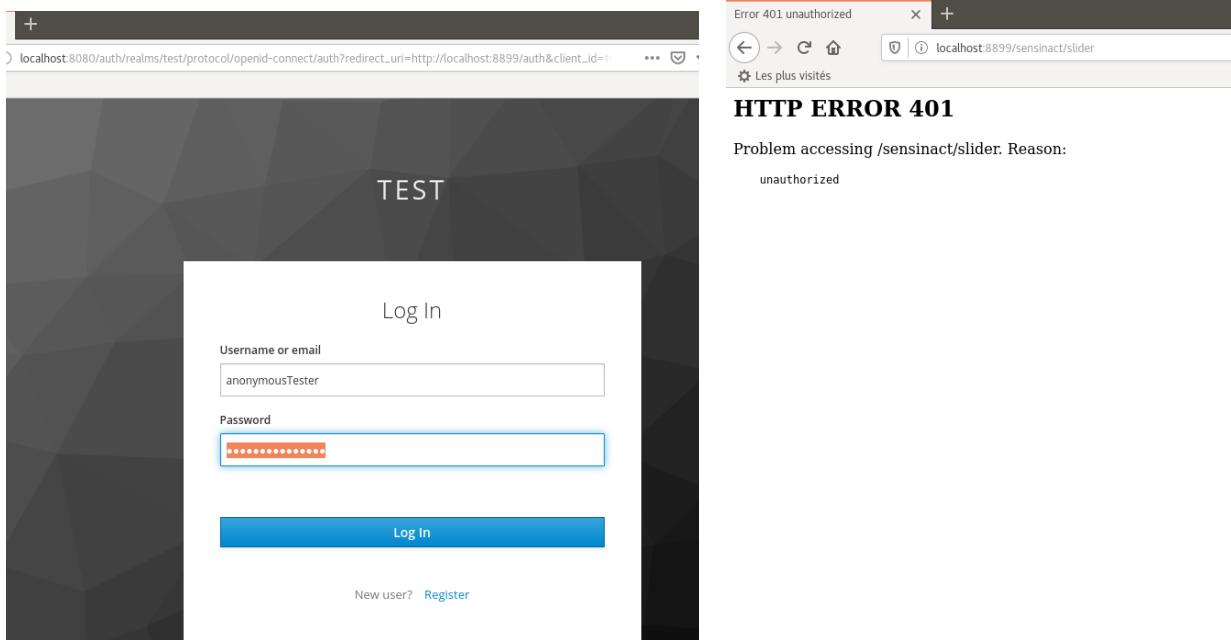


Figure 76. Login as anonymousTester

Licensing

sensiNact is an Eclipse project and is so under Eclipse licence.

- Eclipse Public License - v 1.0 cf. <https://www.eclipse.org/legal/epl-v10.html>



Conclusion

This document provided an introduction to M-Sec components from five different aspects, IoT security, cloud and data level security, P2P level security and blockchain, application-level security, and overall end-to-end security.

IoT security components, shown in Section 1, will help in addressing the security concerns and risks shown in the M-Sec Whitepaper. These components will be tested in real-life scenarios during the upcoming pilots and the integration with other components will be examined. Table 7 below summarizes the discussion reflected in this document.

Table 7: Demonstrators and its correlation with Use Case Pilots

DEMONSTRATOR	TYPE	USE CASE PILOTS	PURPOSE
SECURED IOT DEVICE	Hardware-based solution	Use Case 1 Pilot 1.1 Pilot 1.2	Provide embedded security layer to IoT devices
INTRUSION DETECTION SYSTEM (IDS)	Software-based solution	Use Cases 3 & 4 Pilot-3.1 Pilot-4.1	Secure IoT mobile sensing platform by monitoring and preventing cyber attacks

Further improvements will be made based on the pilot study results deploying the use cases. Therefore, the input received from end users, namely citizens and visitors from both smart cities involved in the trials and interacting with the IoT equipment, is crucial for the consortium to make decisions and improve the deployment along the course of the project execution. This will affect, on the one hand, the kind of data provided and/or its frequency, the applications, related to this IoT equipment, which are the way for end users to interact with the overall system. On the other hand, the moment the deployments are working, the consortium will keep an eye on the kind of cyber-attacks it suffers in order to conduct appropriate countermeasure and thus evolve the M-Sec platform as a whole.

Cloud and data level security includes three components that enhance the security of data between the devices and their respective backends in complementary ways. At first we show how encryption on IoT devices can be made in a safer manner in a domain in which the integration of complex algorithms is quite often neglected due to a lack of resources. Then, we have presented a tool that can monitor deployed devices and enables detection/responses facing attacks. Finally, we present a privacy-enhancing technology that removes any individual or cars on video streams that can be captured in the public space.

At this stage, the security technologies demonstrated in this deliverable are planned to be integrated with the use cases as described in Table 8.





Table 8: Demonstrators and its correlation with Use Case Pilots

Demonstrator	Type	Use Case Pilots	Purpose
Hardware-based encryption	Hardware-based solution	Use Case 1 Pilot 1.1 Pilot 1.2	Provide embedded security layer to IoT devices
Software-based Threat Monitoring	Software-based solution	Use Cases 3 & 4 Pilot-3.1 Pilot-4.1	Secure IoT mobile sensing platform by monitoring and preventing cyber attacks
Image Processing Tool for Video Privacy (GANonymizer)	Software-based solution	Use Case 3 Pilot-3.1	Enable privacy on the video feeds from IP cameras

P2P level security and blockchains presented three different demonstrators: blockchain framework and middleware services, IoT marketplace, trust & reputation management. In our next steps we are going to further explore the potential of smart contracts to support the different M-Sec Use cases and integrate middleware services with the rest of the components. Trust levels over a trustless IoT infrastructure will be further researched in order to allow the convergence of the specific implementations in a broader smart city context. Blockchains will be studied also as a technology foundation upon which values can be exchanged in an IoT infrastructure, enabling thus devices to buy services from other devices. Finally, a model that integrates Trust and Reputation model within IoT Marketplace and smart contracts will be further examined.

Application-level security establishes engineering foundations to support the development of secure smart city applications of the M-Sec system. We have presented the description of two prototypes, the Crypto Companion Database and the Security analysis tool. The Crypto Companion Database provides sensitive data security that cannot be stored on Blockchain. The Security analysis tool provides security requirements that the use case diagram cannot elicit. They contribute to establishing engineering foundations to support the development of secure applications of the M-Sec system.

Finally, overall end-to-end security consists of three demonstrations bound together to provide end-to-end security. At first we have presented a Security Manager which provides a core directory to store any “M-Sec” entity security information such as certificates, passwords and so on. This Security Manager is designed to provide security functions to the overall M-Sec component developed in other WP4 tasks such as IoT Devices, communication channels, blockchain system and applications. First integration of this Security Manager has started in the M-Sec middlewares such as SOXFire and sensiNact, enabling safer collection and distribution of data at a smart city level. Having a common security manager on both this middleware also provides interoperability for cross-border use-cases with the central management of credentials. The integration will continue in the next period and will be iterated with the Use-Case needs.