Multi-layered Security Technologies

for hyper-connected smart cities

D4.1: IoT Security

December 2019

Grant Agreement No. 814917

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

Project acronym	M-Sec
Deliverable	D4.1 IoT Security
Work Package	WP4
Submission date	December 2019
Deliverable lead	TST/YNU
Authors	Arturo Medela (TST), Aamir Bokhari (YNU), Mathieu Gallisot (CEA)
Internal reviewer	ICCS/WU
Dissemination Level	Public
Type of deliverable	DEM
Version history	- v00, 13/November/2019, Arturo Medela, Table of Contents, Full Draft - v01, 26/November/2019, Aamir Bokhari, Initial contribution - v02, 26/November/2019, Arturo Medela, Mathieu Gallissot, Initial contribution - v03, 10/December/2019, Keio contribution - v04, 15/December/2019, Aamir Bokhari, expanded contribution - v05, 16/December/2019, Arturo Medela, second round of contribution - v06, 17/12/2019, Mathieu Gallissot, latest updates - v07, 18/12/2019, Arturo Medela, final round of contribution, edition - v08, 20/12/2019, WU, ICCS, internal review

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

List of Tables

List of Figures

Glossary

ARM	Advanced RISC Machines
ATF	ARM Trusted Firmware
CVE	Common Vulnerabilities and Exposures
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
HW	HardWare
I2C	Inter-Integrated Circuit
IDS	Intrusion Detection System
IMA	Integrity Measure Architecture
IoT	Internet of Things
ISO	International Organization for Standardization
LAN	Local Area Network
MD5	Message-Digest Algorithm 5
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
PPP	Point-to-Point Protocol
RMF	Risk Management Framework
SPI	Serial Peripheral Interface
SSL	Secure Sockets Layer
SW	SoftWare
TCG	Trusted Computing Group

TEE	Trusted Execution Environment
TLS	Transport Layer Security
TPM	Trusted Platform Module
WAN	Wide Area Network

1. Introduction

The main focus of this task is to implement 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.

Retorting to the M-Sec architecture already introduced and discussed in previous reports such as Deliverable 3.3 [D33], the layer addressed by this task is clearly visible in the lowest part as well as its connection to Task 4.2 topics. Figure 1 below shows this situation.

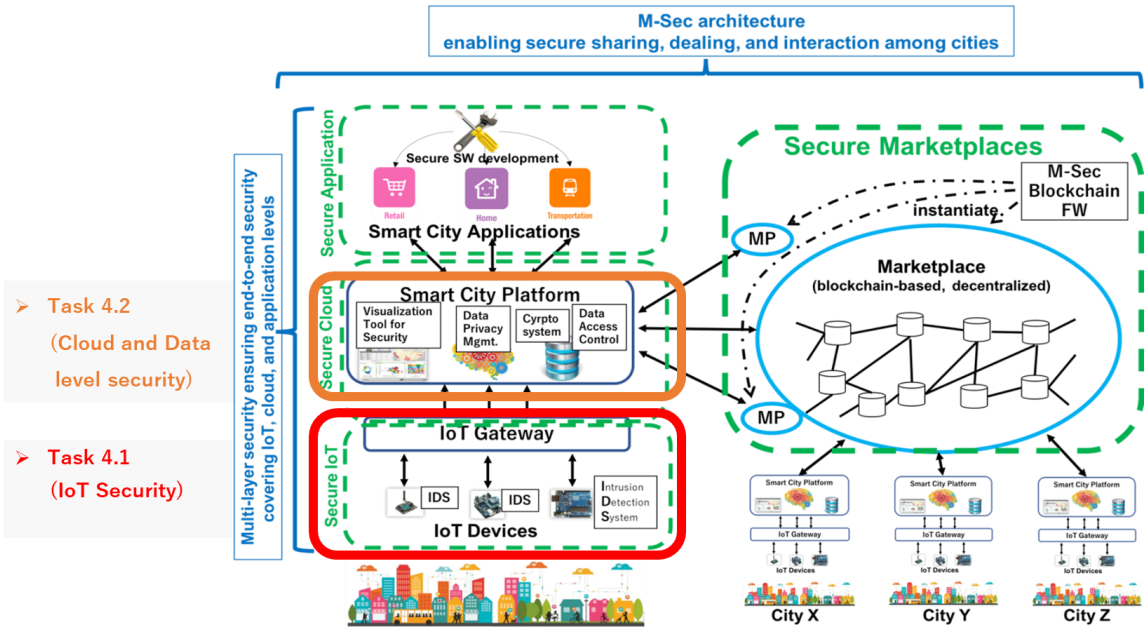

Figure 1: IoT Security in the M-Sec architecture

All in all, this task has as its main objective the definition and ulterior implementation of the M-Sec IoT security layer.

1.1 Scope of the document

This document addresses the main objectives of this task establishing 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:

- 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.

This set of assets will act as the foundations coming out of Task 4.1 and feeding project pilots.

1.2 Relationship to other work packages and tasks

Task 4.1 relates to Work Package 2 since the IoT devices there described as part of the diverse use cases are in need of an increase in their security features to provide users with a safe and reliable service. This is where Work Package 4 comes into action and delivers the techniques to bring an answer to those needs.

On the other hand, there is a relation to Work Package 3, where a series of assets related to the IoT concept is listed and identified for their use in M-Sec demonstrators. From that WP3 list, the exercise to pick the assets directly related to the IoT is derived and thus this report feeded.

Inside this very same work package, it is a fact the close relation to Task 4.2, where the cloud/data security layer is discussed, given that all information produced by the IoT devices will travel through it, and also to Task 4.4, dealing with the application level security, since data provided by IoT devices will be employed by the apps offered to users.

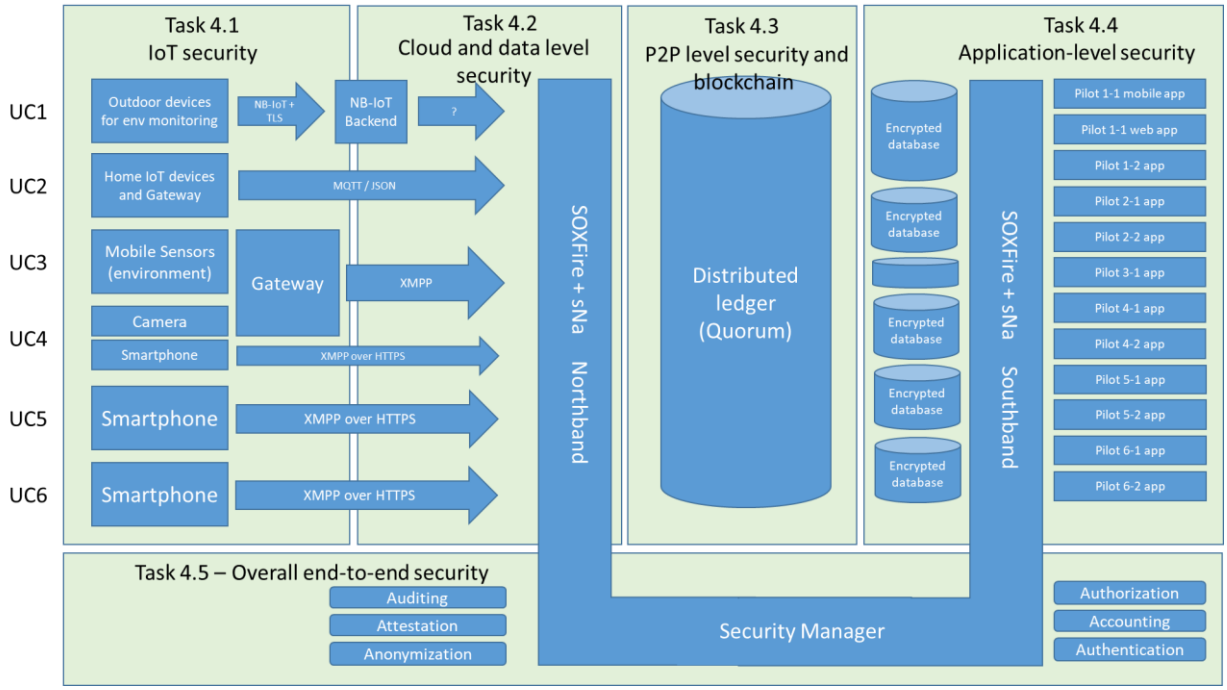

Figure 2: Overall M-Sec topology

1.3 Methodology followed

In a hyper-connected smart city, IoT devices such as sensors, IP cameras, etc. will primarily be communicating via the internet and, therefore, needs to be properly secure. The security solutions can be either hardware-based or software-based. In order to make M-Sec scalable and provide both kind of options to future users, we have adopted the approach to develop both types of solutions and test them in one or more of the use cases being piloted in M-Sec project as proof of concept.

The methodology used for securing the data at IoT security layer consists of

i. Understanding the data path
ii. Threat and risk analysis
iii. Developing solution for securing IoT devices

Furthermore, examples will be taken from certain relevant external risks assessment guides such as the Risk Management Framework (RMF) [RMF], which includes a set of information security policies and standards for federal government developed by The National Institute of Standards and Technology (NIST) [NIST] in its search to establish common assessment procedures to assess the effectiveness of security controls in federal systems. This particular risk-based approach provides a process that integrates security and risk management activities into the system development life cycle. Its architecture follows a series of steps:

- Prepare
- Categorize
- Select
- Implement
- Assess
- Authorize
- Monitor

Another reference is the ISO framework for risk management [ISO], in the shape of the standard ISO 31000:2018, *Risk management – Guidelines,* which provides principles, framework, and a process for managing risk. It can be used by any organization regardless of its size, activity, or sector.

Last but not least, the STRIDE [STR] model of threats developed at Microsoft for identifying computer security threats provides a list of six categories:

- **S**poofing
- **T**ampering
- **R**epudiation
- **I**nformation disclosure (privacy breach or data leak)
- **D**enial of service
- **E**levation of privilege

2. Demonstrator 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's 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 off of 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.

2.1 General Description of the Prototype

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.

2.2 Components

Component Module 1 – EnMon & Crow

Environmental monitoring devices (EnMon)

The environmental monitoring devices architecture is based in the HAL of the microprocessor STM32L4 (using various different microprocessors of that family). 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 3 below.

Figure 3: 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.

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:

```
01001c0000000038363737323330333030343033333000dd00040007c0235
```

The current shape of this IoT device is the one featured in Figure 4. This prototype is yet to be encased in a proper IP casing to deploy it outdoors and will substitute the slot to plug into the power socket for a set of batteries to make it autonomous.

Figure 4: Environmental monitoring device appearance

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.

Crowd counter devices (Crow)

On the other hand, the crowd counter IoT device works with a Raspberry Pi, relying on buildroot [BLD]; 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.

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 5, 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 5: People counter IoT device appearance

Component Module 2 - 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.

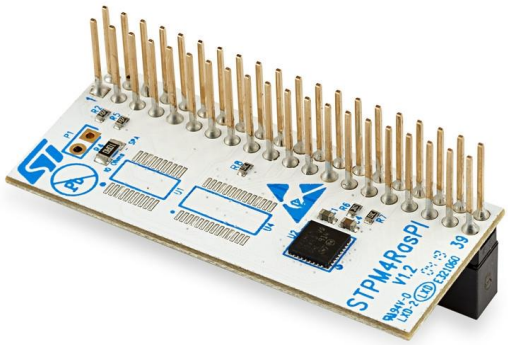

Figure 6: Development board for the STSAFE-TPM

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. 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 6.

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 at 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:

```
PCR new value = Digest of (PCR old value || 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 7. 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.

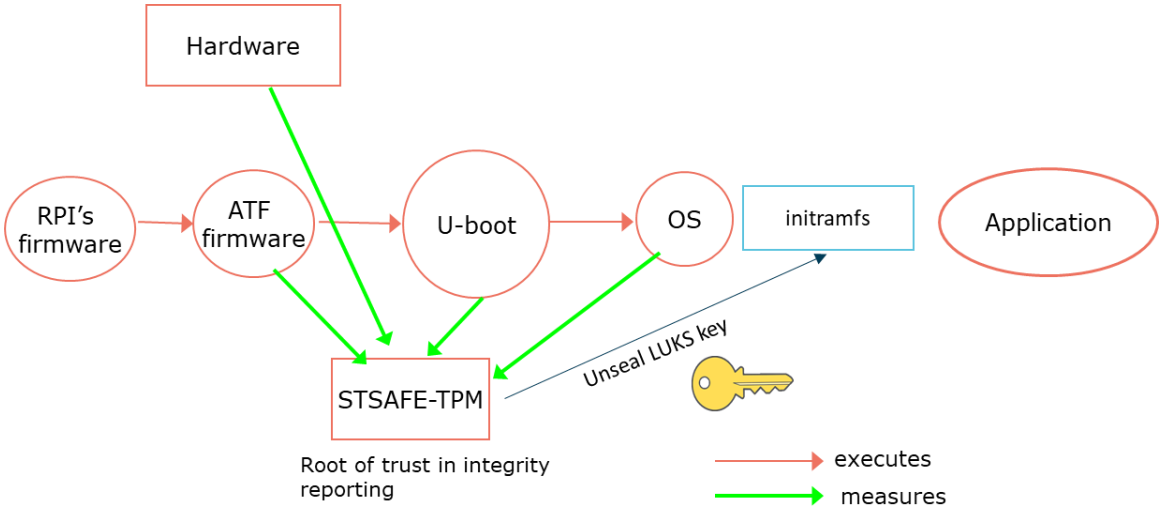

Figure 7: Measured boot process

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) [CVE] 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.

2.3 Package Information & 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 Rasbperry, as shown in Figure 8.

Figure 8: Raspberry PI with the STPM4Raspi extension in White

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 9.

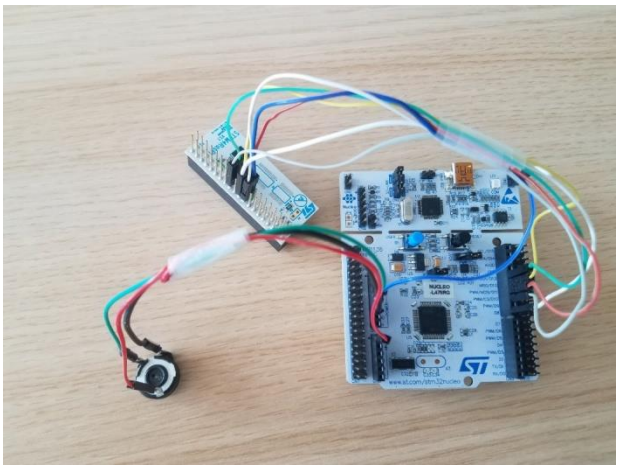

Figure 9: External wiring for microprocessor developments

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. Once this code has been retrieved, the compilation occurs as follows

```
make CROSS_COMPILE=aarch64-linux-gnu-
```

Optionally, the TrustZone® features can be enabled using the OPTEE OS in the secure world. OP-TEE can be compiled with the following command:

```
make -C ../optee_os O=out/arm CFG_ARM64_core=y \
CROSS_COMPILE="aarch64-linux-gnu-" \
CROSS_COMPILE_core="aarch64-linux-gnu-" \
CROSS_COMPILE_ta_arm64=aarch64-linux-gnu- \
CROSS_COMPILE_ta_arm32=arm-linux-gnueabihf- \
CFG_TEE_CORE_LOG_LEVEL=3 \
DEBUG=1 CFG_TEE_BENCHMARK=n PLATFORM=rpi3
```

In order to compile the ARM Trusted Firmware (ATF), sources can be retrieved from the mainline repository and compiled with the following command:

```
make   PLAT=rpi3 DEBUG=0   CROSS_COMPILE=aarch64-linux-gnu- \
 BL33=../u-boot/u-boot.bin \
 RPI3_PRELOADED_DTB_BASE=0x01000000 all fip
```

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

```
make PLAT=rpi3 DEBUG=0  CROSS_COMPILE=aarch64-linux-gnu-  NEED_BL32=yes \
 BL32=../optee/optee_os/out/arm/core/tee-header_v2.bin  \
 BL32_EXTRA1=../optee/optee_os/out/arm/core/tee-pager_v2.bin \
 BL32_EXTRA2=../optee/optee_os/out/arm/core/tee-pageable_v2.bin \
 BL33=../u-boot/u-boot.bin CRASH_REPORTING=1   SPD=opteed \
 RPI3_PRELOADED_DTB_BASE=0x01000000  all fip
```

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

On debian-based systems, the kernel can be recompiled with the following suite of commands:

```
ARCH=arm64
FEATURESET=none
FLAVOUR=arm64
CROSS_COMPILE=aarch64-linux-gnu-

export $(dpkg-architecture -a$ARCH)
export PATH=/usr/lib/ccache:$PATH
export DEB_BUILD_PROFILES="cross nopython nodoc pkg.linux.notools"
export MAKEFLAGS="-j$(($(nproc)*2))"
export DEBIAN_KERNEL_DISABLE_DEBUG=
[ "$(dpkg-parsechangelog --show-field Distribution)" = "UNRELEASED" ] &&
  export DEBIAN_KERNEL_DISABLE_DEBUG=yes

fakeroot make -f debian/rules cleanfakeroot make -f debian/rules orig
fakeroot make -f debian/rules source
fakeroot make -f debian/rules.gen setup_${ARCH}_${FEATURESET}_${FLAVOUR}
fakeroot make -f debian/rules.gen binary-arch_${ARCH}_${FEATURESET}_${FLAVOUR}
fakeroot make -f debian/rules.gen binary-libc-dev_arm64
```

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.

2.4 User Manual

In order to successfully run this demonstrator, no specific user manual is required since the deployment of the secured IoT devices will be rather straightforward, and the user will just need to employ a mobile application to interact with them.

2.5 Licensing (if applicable)

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.

2.6 Reference use case

As suggested before, this demonstration will feed Use Case 1, "*Reliable IoT devices*", and its two pilots, where the environmental measurements and the figures related to the number of people present in certain spots will be delivered by the IoT devices secured from the hardware standpoint discussed in this section.

2.7 Summary

All in all, the main goal of this demonstrator consists of providing an innovative way to secure different IoT devices that are exposed to diverse attacks. This solution will be tested in a real environment in the city of Santander in Spain.

2.8 Next Steps

The actual HW interaction between the IoT environmental device and the Secure Element is the next item in line in the development schedule of this asset. It will be complemented by running SW tests of IoT devices inside the IoTPOT that will help to determine the kind of threats they are subjected to.

Afterward, the development of the IoT crowd counting devices and its integration with the corresponding HW security components will be addressed prior to the deployment in Las Llamas Park as part of Use Case 1. Then, both pilots will be up an running in a real-world scenario where the equipment will be put to a test and requirements coming from end users will arise that will help the consortium better understand which are the real needs that the IoT equipment must comply with, therefore impacting over the next stages in its evolution.

3. Demonstrator 2 - Intrusion Detection System (IDS) for IoT devices

3.1 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 10 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.

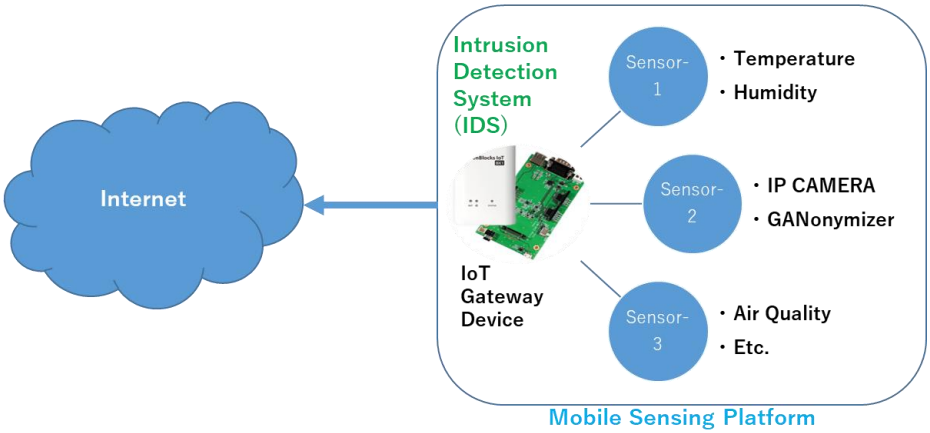

Figure 10: Secured Mobile Sensing Platform

3.2 Components

Internet connectivity exposes IoT devices to potential threats from bad actors (cybercriminals) with malicious intent. As 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 much 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 analyze 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 11 offers a depiction of this device.

Figure 11: 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, analyzed 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.

3.3 Package Information & 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:

```
root@iot:~# ls -l
total 40712
-rw-r--r-- 1        root root         0 Sep 27  2017 1
-rwxr-xr-x 1        root root         41678664 Nov 26 16:59 installer.sh
-rw------- 1        root root         1675 Nov 26 16:32 key.pem
root@iot:~#
```

2) Launch installer shell as follows:

```
root@iot:~# chmod +x installer.sh
root@iot:~# ./installer.sh
installer/
installer/start_ips.sh
installer/suricata-4.1.5.tgz
installer/root.tgz
:        :        :        :
```

3) Wait for it to complete updates, download, and install the program. It will finish installation and return the prompt as follows:

```
:        :        :        :
Index setup finished.
Loading dashboards
Loaded dashboards
Loaded machine learning job configurations
Loaded Ingest pipelines
root@iot:~#
```

4) Now check root directory contents and you should see as follows:

```
root@iot:~# ls -l
total 40724
-rw-r--r-- 1      root root         0 Sep 27  2017 1
-rwxr-xr-x 1      root root         41678664 Nov 26 16:59 installer.sh
-rw------- 1      root root         1675 Nov 26 16:32 key.pem
-rw-r--r-- 1      root root          801 Nov 26 17:50 readme.txt
-rwxr-xr-x 1      root root         187 Nov 26 17:56 start_ids.sh
-rwxr-xr-x 1      root root         343 Nov 26 17:58 start_ips.sh
```

The "*start_ids.sh*" is for monitoring mode, whereas, "*start_ips.sh*" is for preventing attacks mode.

Run Demonstrator Guide:

Demonstrator runs in the backgroud, 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
```

3.4 Licensing (if applicable)

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, IoTPOT is a proprietary asset that is used for study and analysis purposes only during the research and development phase.

3.5 Reference use case

The reference use case for this solution is use case 3 and 4. Following the above installation guide, the IoT gateways for each mobile sensing platform will be installed with the IDS security package and run in the background on reboot, monitoring, and preventing potential attacks.

3.6 Summary

This demonstrator will provide IoT layer security to the mobile sensing platform that was lacking security in use case 3 and 4 scenarios. The software has been customized with OS hardening to reduce attack surface, secured communication using Transport Layer Security (TLS), and signature patterns obtained from the testing and analysis in the IoT honeypot, besides other well-known up-to-date attack signature patterns provided by open source resources.

3.7 Next Steps

The next step is to install this demonstrator in other mobile sensing platforms in preparation for the Pilots 3.1 & 4.1. Also, look into ways to further improve it, depending on the results of these pilots.

4. Conclusion

The prototypes developed will help in addressing the security concerns and risks shown in WP2 & WP3. These prototypes will be tested in real-life scenarios during the upcoming pilots and the integration with other WP4 elements will be examined. Table 3 below summarizes the discussion reflected in this document.

Table 3: 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 in the M18 – M30 period, 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.

Acronyms

[BLD] Buildroot, https://buildroot.org/

[CVE] Common Vulnerabilities and Exposures website, https://cve.mitre.org/

[D33] Deliverable 3.3 "*M-Sec Architecture: Functional and technical specifications – first version*", June 2019

[ISO] ISO 31000 Risk Management, https://www.iso.org/iso-31000-risk-management.html

[NIST] National Institute of Standards and Technology, https://www.nist.gov/

[RMF] Risk Management Framework Overview, https://csrc.nist.gov/projects/risk-management/risk-management-framework-(RMF)-Overview

[STR] STRIDE chart, https://www.microsoft.com/security/blog/2007/09/11/stride-chart/