# D6.3

# Demonstrators Implementation Report – First Release

| | |
|---|---|
| **Project number:** | 779391 |
| **Project acronym:** | FutureTPM |
| **Project title:** | Future Proofing the Connected World: A Quantum-Resistant Trusted Platform Module |
| **Start date of the project:** | 1st January, 2018 |
| **Duration:** | 36 months |
| **Programme:** | H2020-DS-LEIT-2017 |

| | |
|---|---|
| **Deliverable type:** | Report |
| **Deliverable reference number:** | DS-06-779391 / D6.3/ 1.0 |
| **Work package contributing to the deliverable:** | WP 6 |
| **Due date:** | December 2019 – M24 |
| **Actual submission date:** | 6th April, 2020 |

| | |
|---|---|
| **Responsible organisation:** | S5 |
| **Editor:** | Sotiris Koussouris |
| **Dissemination level:** | PU |
| **Revision:** | 1.0 |

| | |
|---|---|
| **Abstract:** | Deliverable D6.3 provides a detailed documentation of the first round of experiments of the FutureTPM framework, in the context of the three envisioned use cases. It summarizes the operation of the FutureTPM demonstrators coupled with a comprehensive analysis of the integration and evaluation of the first release of the SW-based QR TPM environment enriched with lessons learnt and challenges that need to be taken into consideration during the next development, integration and evaluation cycle of the project. |
| **Keywords:** | Demonstrators, Implementation Report, Testing, Evaluation |

The project FutureTPM has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 779391.

**Editor**

Sotiris Koussouris (S5)

Thanassis Giannetsos (DTU)

**Contributors** (ordered according to beneficiary numbers)

Sofianna Menesidou, Dimitris Papamartzivanos (UBITECH)

Paulo Sérgio Alves Martins (INESC-ID), Luís Fiolhais (INESC-ID)

Roberto Sassu, Silviu Vlasceanu, Rahul Dulta (HWDU)

Fanis Sklinos, Stratos Moros (INDEV)

# Executive Summary

Deliverable D6.3 covers the main activities of the **evaluation, validation and refinement phase** related to setting up, executing and evaluating the three envisioned use cases; namely the "*Secure Mobile Wallet and Payments*", "*Personal Activity and Health Kit Data Tracking*" and "*Device Management*" reference scenarios. It reflects the pilot implementation and integration of the FutureTPM framework in three different use cases (called demonstrators), to test **the assumptions of the project, and the feasibility, the applicability and the overall acceptance of post-quantum TPM in specific business cases**, not only in terms of **security**, but also in terms of **performance, availability and of other business critical indicators**.

Within WP6, three discreet demonstrators have been set up for testing the integration of the core components of the FutureTPM platform (i.e., the *Risk Assessment*, the *Security Policy Enforcement* and the *QR TPM* modules.) into real-life business applications, and critically appraise the effectiveness of the overall platform for security and performance in those business settings. More information on the scope and specifics of these demonstrators can be found in deliverable D6.1 [1].

As the FutureTPM project adopts a two-cycle development, integration, demonstration and evaluation approach, the deliverable at hand provides a detailed documentation of the **first-cycle demonstrator results till M24** (, following the guidelines and the metrics set in the evaluation plan that was part of deliverable D6.1 "Technical Integration Points and Testing Plan"), while the final release (with their evaluation) will be delivered at M36. As such, the work performed for the "Secure Mobile Wallet and Payments", the "Personal Activity and Health Kit Data Tracking" and the "Device Management" demonstrators till M24 of the project is presented here. It needs to be noted that in order to facilitate smooth transition to the FutureTPM technology, all these use cases have worked initially to integrate the TPM2.0 characteristics that will be also test in FutureTPM, as to have a reference point for comparison of the post-quantum approach to the existing technological de facto standard.

Towards this direction, a detailed description of each reference scenario is given with the defined user stories of interest, and their requirements, as well as the conditions and the implementation, integration status of each demonstrator coupled with a detailed analysis of the extracted results. Recall that the goal of FutureTPM is to show case the use of TPMs towards providing enhanced security, privacy and trust while migrating into post-quantum era. Each one of these properties is demonstrated in separate reference scenarios, in order to avoid overlaps and to be able to progress with a more detailed evaluation and validation of the TPM operations needed for achieving a subset of these requirements.

Along this line, this deliverable captures the first experimental (demonstration) period of the project with many significant results drawn which will be fed back to the other technical WPs so as to continue research activities on further refining the FutureTPM solution. **The goal is to improve the current implementation, streamline the used algorithms and improve the technical backbone of the envisioned FutureTPM platform.** As such, it is expected that the next deliverable on the WP6 demonstrators, will provide revised figures for specific metrics that will be impacted by the ongoing work in the project, and will also provide a more complete and accurate view on the overall impact experienced in each demonstrator.

In the following chapters, we provide an in-depth analysis of the underpinnings of the performed experiments with the extracted results, and describe all open issues that need to be solved for further improving the performance of the overall framework. This ranges from the necessary improvements of a subset of the implemented QR crypto primitives (especially the Direct Anonymous Attestation algorithm) to the functional interfaces needed for supporting the required interactions with the other FutureTPM integral components, and especially the Control-flow Attestation Engine [3].

# Contents

## List of Figures

## List of Tables

# Chapter 1 Introduction and Overview of the First Experimental Period (M13-M24)

This deliverable covers the main activities of the **evaluation, validation and refinement phase** related to setting up, executing and evaluating the three envisioned use cases; namely the "*Secure Mobile Wallet and Payments*", "*Personal Activity and Health Kit Data Tracking*" and "*Device Management*" reference scenarios. As the FutureTPM project adopts a two-cycle development, integration, demonstration and evaluation approach, D6.3 provides a detailed documentation of the **first-cycle demonstrator results till M24**, while the final release (with their evaluation) will be delivered at M36. Focus, in this first release, is placed on the **performance evaluation of the SW-based QR TPM and the implemented Trusted Software Stack (TSS)** with timings of the sequences of TPM commands, for achieving the **security, privacy, and trust** properties of interest per reference scenario [1], being extracted. This also reflects the **execution overhead of the selected and integrated QR crypto algorithms** [2].

Towards this direction, a detailed description of each reference scenario is given with the defined user stories of interest, and their requirements, as well as the conditions and the implementation, integration status of each demonstrator coupled with a detailed analysis of the extracted results. Recall that the goal of FutureTPM is to show case the use of TPMs towards providing enhanced security, privacy and trust while migrating into post-quantum era. Each one of these properties is demonstrated in separate reference scenarios, in order to avoid overlaps and to be able to progress with a more detailed evaluation and validation of the TPM operations needed for achieving a subset of these requirements.

More specifically, the "*Secure Mobile Wallet and Payments*" reference scenario focuses on the provision of enhanced security properties in such complex e-Payment scenarios; the "*Personal Activity and Health Kit Data Tracking"* focuses on the strict user privacy issues that need to be met (through the use of the DAA protocol); and the "*Device Management*" scenario focuses on monitoring and the establishment of trust between devices that are managed by an NMS server.

| Reference Scenario | TPM Type | Security Property | Functionalities |
|---|---|---|---|
| **Secure Mobile Wallet and Payments** | Software TPM | Security | Sealing, Unsealing, Key Generation |
| **Personal Activity and Health Kit Data Tracking** | Software TPM | Privacy | DAA Join, DAA Sign, DAA Verify, DAA Attestation |
| **Device Management** | Software TPM | Trust | Remote Attestation, Device Management with Secure Key Identifiers |

Table 1: Reference Scenarios Overview during First Experimentation Cycle

As will be described in later sections, it is worth mentioning that the consortium decided to adjust the evaluation plan that had been put forth in D6.1 [1] by prompting to focus (in the first-cycle of experimentation) on the evaluation of the QR SW-based TPM environment that has been integrated in all demonstrators. This deviates from the initial plan considering also the QR HW-based and VM-based TPMs that are to be tested in the context of the e-Payment and Device Management reference scenarios, respectively. Thus, the final release of all QR TPM modules, alongside the other core FutureTPM framework components (providing the Risk Assessment, Security Policy Enforcement and Control-flow Attestation mechanisms), will be extensively tested and evaluated in the second development cycle (till M36) once the necessary updates have being performed based on the lessons learnt, as documented in this deliverable.

Table 1 above summarizes the main focus of the three reference scenarios, during this first experimentation cycle, regarding security, privacy or trust properties of interest and functionalities that were evaluated.

Overall, summarising this period, this deliverable captures the first experimental (demonstration) period of the project with many significant results drawn which will be fed back to the other technical WPs so as to continue research activities on further refining the FutureTPM solution. **The goal is to improve the current implementation, streamline the used algorithms and improve the technical backbone of the envisioned FutureTPM platform.** As such, it is expected that the next deliverable on the WP6 demonstrators, will provide revised figures for specific metrics that will be impacted by the ongoing work in the project, and will also provide a more complete and accurate view on the overall impact experienced in each demonstrator.

## 1.1 Evaluation, Validation and Refinement Methodology

As was described previously and is also specified in D6.1 [1], in each demonstrator a specific set of user stories and unit tests has been set up, and specific quantitative and qualitative KPIs have been designed to measure the impact of the FutureTPM framework. **This road map did not only focused on KPIs related to performance criteria but also taken into consideration the business value of the core FutureTPM services and functionalities provided.**

As decided by the consortium, during the first development cycle and with a view on the release of the first version of the QR TPM Software Stack, **all demonstrators have initially worked to integrate the TPM2.0 characteristics to their existing infrastructures.** This will enable the better evaluation of the implemented QR TPM modules by using the integration and performance evaluation of the current TPM architecture, as a starting point, when making the shift to QR trusted computing technologies. It allowed the engineers that worked on the demonstrators to acquire more knowledge on the TPM2.0 solutions (currently available) and get familiar with the overall stack, so as to be in a better position to absorb the knowledge required for plugin into their applications the QR TPM code that is developed by the project. Furthermore, **timings of the different TPM2.0 commands have been extracted, and used as reference points, for the QR TPM experiments to follow.** This allowed the consortium to measure the performance and impact of the FutureTPM approach, to the TPM2.0-enabled business applications, and better define some of the KPIs that were identified in the previous stages of implementation. As such, **this deliverable provides an initial evaluation report of the results gathered from the execution of the first demonstrator's phase**, following the scenarios and test cases that were defined for each demonstrator and summarized in Table 1.

It needs to be highlighted that during the first development and evaluation cycle, all three reference scenarios have experimented with the SW-based version of the FutureTPM solution, while in the next period the evaluation of all the QR TPM modules (SW-, HW-, and VM-based), alongside the other core framework components, will commence. *The motivation behind this action plan was to enable the consortium to extract a first set of results, which can be then used as a reference point for the later experiments.* Considering that the TSS, of the QR SW-based TPM, is also used for building the necessary interfaces to interact with the other QR TPM modules, it was imperative to first perform an extensive testing of the first software stack release so as to identify any open issues to be solved before progressing to the final evaluation of all TPM environments of the FutureTPM framework.

In the following chapters, we provide an in-depth analysis of the underpinnings of the performed experiments with the extracted results, and describe all open issues that need to be solved for further improving the performance of the overall framework. This ranges from the necessary improvements of a subset of the implemented QR crypto primitives (especially the Direct Anonymous Attestation algorithm) to the functional interfaces needed for supporting the required interactions with the other FutureTPM integral components, and especially the Control-flow Attestation Engine [3]

## 1.2 Environmental Setup of Deployed Scenarios

One of the main goals of this deliverable is to also provide a baseline setup for all future demonstrator experimentation activities. In the first evaluation cycle, the QR SW-based TPM was used as the underlying trusted component. As such, it is of upmost importance to design an environment which closely resembles a real-world scenario. The reasoning behind this approach is twofold: First, **we envision the usage of the methodologies proposed herein on architecture and application space exploration**, *i.e.*, a developer can benefit from these real scenarios by gaining greater insight into building their APIs, and "mock test" different frontend and backend configurations before deploying on a real TPM hardware. Second, we also propose a **baseline testing methodology** (Section 2.3) to better guide a developer on possible bottlenecks found in their application design; this can range from performance to memory footprint and communication latency, and forewarn them of bad design structures.

There are two recommended infrastructures when testing an application using the FutureTPM platform. The first one is called FutureTPM stack, and is comprised of two components: the **TSS** and the **SW-TPM**. The FutureTPM stack is a fully fledged out software emulator of a physical TPM (SW-TPM), based on IBM's open-source project, and a software library which implements the TPM's commands and its software stack. The SW-TPM was built to closely mimic its real-life counterpart, thus, providing certain memory and communication latency guarantees. The emulator makes no direct use of the heap, employing its memory in the .bss and .data program segments. The lack of heap usage ensures that the SW-TPM spends a very small amount of time inside system calls and the majority of its time is dedicated on executing the command code. Communication is handled strictly through a TCP layer which emulates the TPM's physical TCTI layer [4][5][6], providing some serialization and latency. Furthermore, the TSS library provided does not have to be used uniquely with the SW-TPM. New commands or functionalities added to the TSS can be first tested and prototyped using the software emulator, and then, when real hardware is available, the same code can be leveraged.

The second recommended infrastructure is libtpms. Libtpms is a wrapper of the software TPM and is meant to be used in conjunction with the swtpm component, which exposes TPM functionality through different interfaces (e.g. socket, device). swtpm also provides a dedicated interface that can be used directly by QEMU to provide TPM functionality to software inside a virtual machine.

### *1.2.1 IBM vs. Intel TSS Integration*

Several commodity TSS implementations exist, namely the Intel and IBM TSS implementation instances. While they share many similarities, Intel TSS [7] provides some additional functionalities (especially when it comes to resource management [8]) that have not been full incorporated yet in the IBM TSS [9]. On the other hand, the current IBM TSS version is at a more stable state. Based on these observations and in order to be able to perform a more holistic investigation in the context of FutureTPM, we decided to leverage both instances: *the Intel TSS was used as the baseline for the risk assessment analysis whereas the IBM TSS provided the cornerstone for the implementation and demonstration of the QR-based TPM* (WP5).

While the FutureTPM QR TPM stack is based on the enhancement of the IBM TSS, the Intel TSS was leveraged specifically for the vulnerability analysis and attestation of the **TPM Access Broker (TAB)** and the **Resource Manager (RM)** – two components that are of particular interest due to their inherent functionalities that may lead to sensitive data leakage (e.g., information about stored keys). The TAB controls multi-process synchronization to the TPM. Basically, it allows multiple processes to access the TPM without stomping on each other, while the RM acts in a manner similar to the virtual memory manager in an OS due to limited on-board memory [10]. TPMs generally have very limited memory and objects, sessions, and sequences need to be swapped from the TPM to and from memory to allow TPM commands to execute. RM must parse the command byte stream before the command is sent to the TPM and take any actions required to ensure that all transient objects used by that command are loaded into the TPM. This includes all sessions referenced in the authorization area and all objects, sessions, and sequences whose handles are in the command's handle area. Very recently, the Linux kernel 4.12 has included in-kernel RM [11] to provide isolation between objects & sessions created by different connections which is the core functionality required by applications. Eventually, all of the required features will end up in the kernel RM and it will become the default [11].

In the context of FutureTPM, the **Control-Flow Attestation tookit is used for tracing and attesting the correctness of the TAB and RM components**: *to hook eBPF in the in-kernel RM, trace all the TPM commands and identify possible object, sequence, session leakage and proven broken TPM commands* [12].

Furthermore, we also provide a port of Intel's open source tss implementation which is mostly used to interface with higher level applications, *e.g.*, openssl and is able to fully replace the TSS provided by the FutureTPM stack. In D5.1, Intel's TSS was used to replace openssl's cryptographic engine with the one available in the SW-TPM, thus, providing QR algorithms directly to a TLS connection.

Summarizing, the FutureTPM infrastructure provides a fast edit-debug-run cycle and direct deployment in real hardware, while guaranteeing sensible memory usage and communication latency. As far as emulation goes, the usage of the FutureTPM stack is the closest a developer gets to the hardware without actually deploying it.

## 1.3 Testing Methodology

Testing methodologies are the strategies and approaches used to test the FutureTPM platform to ensure it is fit for purpose. The focus is on testing that the QR TPM modules work in accordance to their specifications [13] and have no undesirable effects when employed in ways outside of their design parameters. Since each demonstrator will be executed in different hosts with various configurations, this section attempts to shine some light on the standardisation process used to assure comparable results between each demonstrator instance.

Modern processors found in commodity systems employ a plethora of techniques to improve the performance of all applications types. The TPM, on the other hand, does not offer such performance optimizations. As such, before describing the testing methodology, we need to understand what differentiates both architectures. Commodity processors rely on two major techniques to boost performance: **out-of-order execution** and **caching**. Out-of-order execution is used to exploit parallelism at the instruction level. Caching is achieved by applying multiple levels of small but fast memories between the slow external memory and the processor, hiding the large latency of the external memory. Since the TPM is implemented in an ASIC with tightly integrated domain specific accelerators (DSA) for most cryptographic operations, the usage of out-of-order execution in a testing platform can be safely ignored. However, caching cannot be so easily dismissed.

The OS uses time-slicing to share a single processor core between several processes. Therefore, it is possible that a process, other than the one we are measuring, evicts our process cached lines from the cache. As such, collecting measurements at different times results in completely different timings between the same applications. To diminish the effects of conflict-based evictions from the cache hierarchy, we must execute our measurements hundreds of times in a row. In doing so, we are avoiding cold accesses to the caches and possible spurious evictions. Further, this method closely resembles a TPM accessing its scratchpad memory. To offset the results from spurious evictions and cold accesses, we shall use the linearly weighted moving average (LWMA) in order to bias the most recent results from the oldest, i.*e.*, the measurements obtained using the warmed-up caches are preferred.

Measurements are performed differently depending on the infrastructure used (Section 2.2). When measuring application timings, using the FutureTPM stack, performance values are measured using bash's time command. While the TPM server is running and started up---using the startup command---, each command is measured with the TPM always in the same state. The aforementioned procedure measures command creation, communication, destruction, and the TPM's processing. The TPM processing can be generally thought of a five-stage operation: TCP reception, command validation and deserialization, command execution, response creation and serialization with results, and response dispatch. Note that when using authenticated sessions, the command validation operation is more involved and may require more time.

Given that demonstrators use the TSS in a different way, we found two alternative levels of the software stack, common to all demonstrators, from where performance measurements can be taken. **The first alternative level is the TSS library, which has been patched to measure the time elapsed between the beginning of TSS_Execute() and the end of the same function.** *Measurements from the TSS library take into consideration the time necessary to execute a command, the marshalling and unmarshalling of the buffers, and the time necessary to transmit the data between the TSS and libtpms*. The second alternative level from where performance measurements can be taken is **libtpms**. Doing performance measurements at this level is particularly interesting to compare the performance of non-QR algorithms versus QR algorithms.

Overall, within FutureTPM, we have prompted in identifying a robust testing methodology to be followed by all reference use cases. As will be depicted in the following chapters, for each demonstrator a detailed set of test cases were compiled (i.e., unit testing, integration testing and system testing) in order to measure the behaviour of the QR SW-based TPM in different conditions and scenarios, thus, evaluating whether the system can operate at the required response times for supporting the required security, privacy and trust properties.

# Chapter 2 Demonstrator #1 – *Secure Mobile Wallet and Payments*

## 2.1 Demonstrator Overview

The “INDEV Secure Mobile Wallet and Payments” use case, works on the security of mobile wallet and e-payment applications and more precisely on how the sensitive tokens are handled by both the mobile payment app and the corresponding backend server. The token correctness is fundamental to the overall security of the mobile payment transaction itself, making a quantum resistant TPM necessary to ensure both the integrity of sensitive data and the future proofing of the mobile payments application to resist quantum attacks. In this reference scenario, we will demonstrate a) the **sealing functionality** for the Bearer and Financial Tokens, b) the **unsealing** functionality for the tokens and c) the symmetric **key generation** to encrypt financial transaction history logs. At the same time all the aforementioned functionalities will be traced by the integrated risk assessment framework at the kernel level (kernel interceptor) and produce the quantified risk (second evaluation cycle – D6.5). Figure 1 below presents the high-level approach of this reference scenario introduced in D6.1 [1].

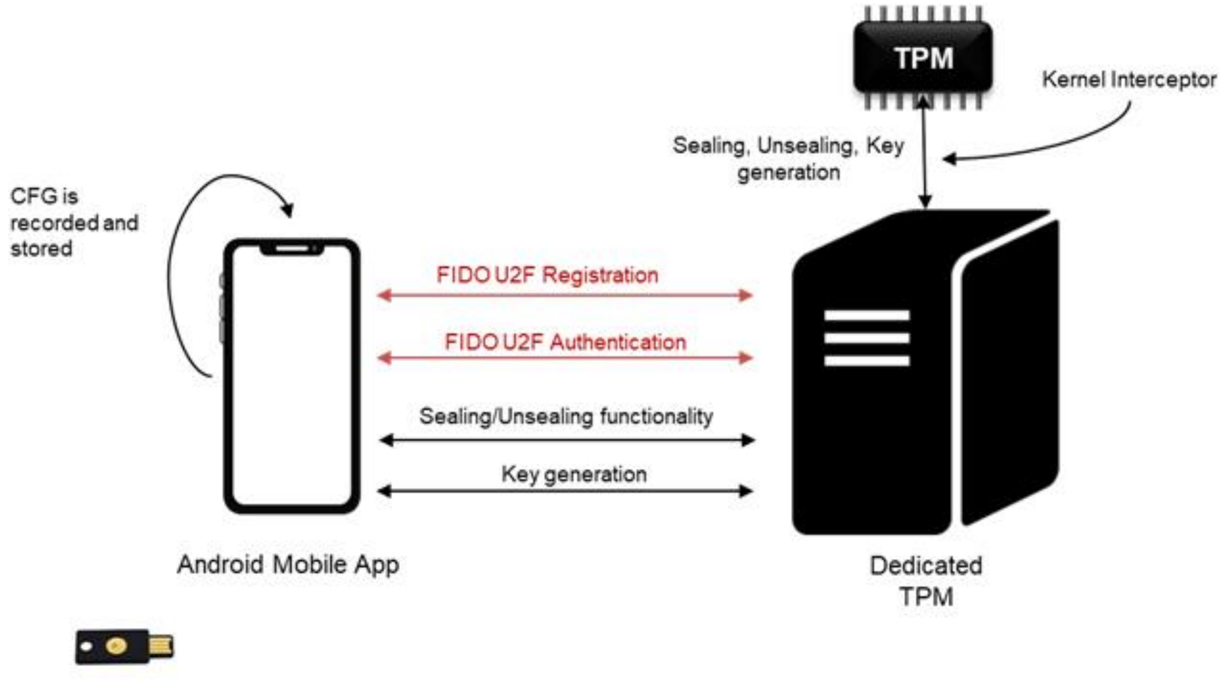


Figure 1: Secure Mobile Wallet and Payments High Level Approach

### *2.1.1 Demonstrator Needs and Challenges*

Mobile wallet and e-Payment received significant attention because it enables an easy payment mechanism and becomes an important complement to traditional payment means. However, using a mobile wallet over open devices and networks poses security challenges of a new dimension. The security is fundamental to the overall security of the mobile payment transaction itself. How the sensitive tokens are handled by the mobile payment app and the corresponding backend server are key security considerations. A quantum resistant TPM can help ensuring both the integrity of sensitive data and the future proofing of the mobile payments application to resist quantum attacks. The “INDEV Secure Mobile Wallet and Payments” use case, works on exactly this issue of making the sensitive data protected and tamper-proof, demonstrating how the use of FutureTPM project can benefit mobile wallet and payment applications to be secure. Below we summarize the updated and more detailed scenario user stories.

In the majority of the current Android devices, there is no TPM module attached, no recognized API definition available for Android TSS and most of the Java-based implementations such as jTSS are complex and error prone. In addition, as already introduced, this reference scenario will be demonstrated (during the second evaluation cycle) based on the use of the **hardware TPM**. In the context of FutureTPM, the hardware TPM will be released on an FPGA-based board exposed by TCP/IP. For that reason, we decided to adopt and architecture where the hardware TPM is hosted in a dedicated cloud server. The assumptions made in order to demonstrate this reference scenario are:

- An authenticated channel is established between the Android mobile app and the TPM server based on FIDO U2F signaling.
- User registers to the dedicated TPM Server (FIDO U2F Registration Phase)
- User authenticates to the TPM Server with FIDO webAuthN every time that needs to perform a TPM functionality (FIDO U2F Authentication Phase).
- The tokens are sealed based on the handle h created during the FIDO U2F Authentication Phase.

### *2.1.2 Demonstrator Architecture*

The demonstrator that is being designed and developed during the FutureTPM project is based on a refactored mobile application of the current INDEV application, bringing into the picture TPM methods to secure sensitive tokens.

Since, the hardware-based TPM will be released on an FPGA-based board exposed by TCP/IP, the consortium took appropriate measures and decided to use a TPM in a dedicated cloud server. *This dedicated TPM sever acts as an internal TPM that should be integrated in the Android device.* This approach brings the ability to further extend our solution and apply prominent authentication mechanisms such as FIDO Universal 2nd Factor (U2F) between the communication of the Android application and the dedicated TPM server.

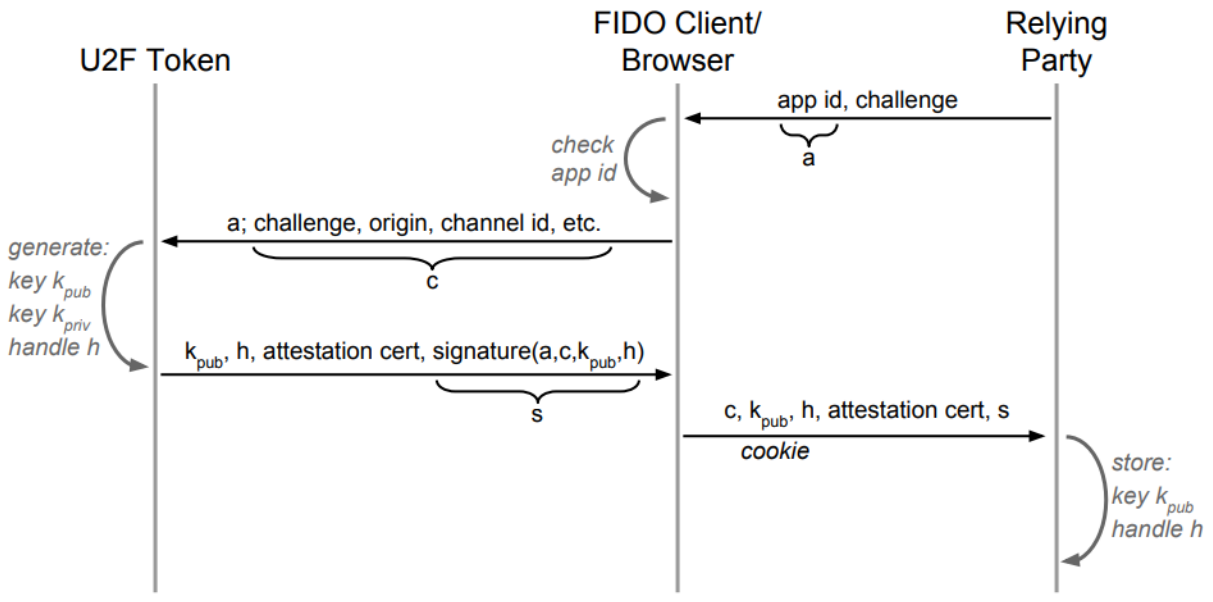

U2F Token
FIDO Client/ Browser
Relying Party
app id, challenge
a
check app id
a; challenge, origin, channel id, etc.
c
generate: key k_pub key k_priv handle h
k_pub, h, attestation cert, signature(a,c,k_pub,h)
s
c, k_pub, h, attestation cert, s
cookie
store: key k_pub handle h

Figure 2: U2F Registration

FIDO (Fast ID Online) is a set of technology-agnostic security specifications for strong authentication. FIDO specifications support multifactor authentication (MFA) and public key cryptography. FIDO U2F protocol is the state-of-the-art in the domain of authentication. U2F is an open authentication standard that enables internet users to securely access any number of online services with one single security key instantly and with no drivers or client software needed. U2F authentication requires a strong second factor such as a Near Field Communication (NFC) tap or USB security token. The user is prompted to insert and touch their personal U2F device during login (proof of presence). The user's FIDO-enabled device creates a new key pair, and the public key is shared with the online service and associated with the user's account. The service can then authenticate the user by requesting that the registered device signs a challenge with the private key. With this approach, no secrets are shared between service providers, and an affordable U2F Security Key can support any number of services. Both U2F Registration and Authentication Phases will be used with NFC-based Yubico HSM device. Figure 2 andFigure 3 present the aforementioned challenge-response flows for the Registration and Authentication phases respectively.

Note that, U2F authentication is an **extra layer of security** introduced in D6.1 and it was outside the scope the use case at first place. However, we will use it specifically in this reference scenario as extra security guarantees between the mobile and the dedicated TPM server. This extra layer does not change the nature of the application, since it will not be necessary when the Android device contains an attached TPM. Our approach, using this extra layer, is **more generic** and **covers also the Android devices without the support of the TPM**, by providing the ability to connect and use a dedicated TPM server.

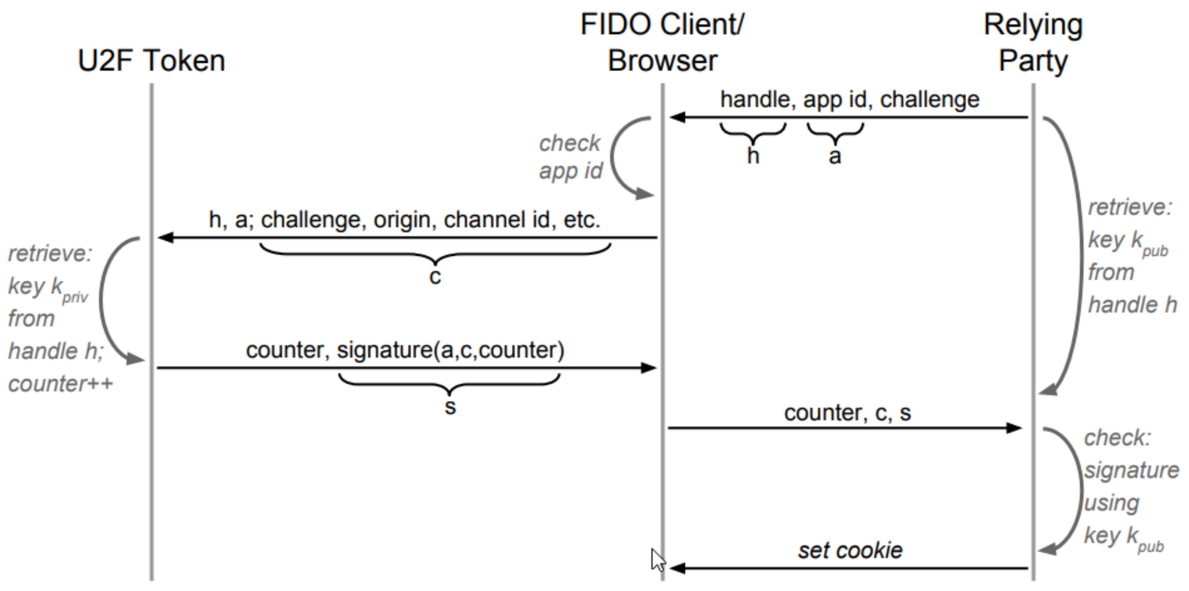

U2F Token
FIDO Client/ Browser
Relying Party
handle, app id, challenge
h
a
check app id
h, a; challenge, origin, channel id, etc.
c
retrieve: key $k_{priv}$ from handle h; counter++
retrieve: key $k_{pub}$ from handle h
counter, signature(a,c,counter)
s
counter, c, s
check: signature using key $k_{pub}$
set cookie

Figure 3: U2F Authentication

The implementation of the Android application needs to secure two discrete types of tokens. These two types of tokens are the Bearer Token and the Financial Token.

- **Bearer Token**: A security token with the property that any party in possession of this token (a "bearer") can use it in any way that any other party in possession of it can. When a user authenticates, the authentication server then generates the Bearer Token which is necessary to get an Access Token. This token is an OAuth token that is used for authentication between the client and the business logic.
- **Financial Token**: This token is created by a 3d party service, used to finalize a financial transaction and represents a user's credit card in a time frame.

To sum up, in this reference scenario we will demonstrate a) the **sealing functionality** for the Bearer and Financial Tokens, b) the **unsealing** functionality for the Bearer Token and c) symmetric **key generation** to encrypt financial transaction history logs. The aforementioned functionalities will be performed using the FutureTPM. In the context of this demonstration, the timing performance of the FutureTPM is compared to the one on the TPM2.0, in order to critically appraise the effectiveness of the FutureTPM framework for security and performance in the context of the Secure Mobile Wallet and Payments scenario. Note that, only after a secure authentication of the user and establishment of a secure channel, the mobile app will be able to use the TPM functionalities. Recall that Figure 1 offers a high-level approach of this reference scenario.

## 2.2 Emulated System Description

In order to concretely test the proposed architecture, the following scenario was emulated: all device instances (including the TPM) are executed without using any virtualisation mean, but instead the developed applications run directly on the hosts. For the TPM server, a machine with Kubuntu 18.04 OS was used, with CPU Intel Core i7-7700HQ@2.80GHz and 16GB of RAM. This emulation scenario allows us to measure the performance of the TPM variants without the interference of intermediate virtualisation layers.

For the client side, an Android App was used, running *WenAuthnAndroidLib* to register and authenticate with Public Key Credentials at *https://future-tpm.ubitech.eu,* running Android M (API level 23) or newer. The Android device supports NfcManager in order to be able to interact with the with the U2F Security Key. The specifications of the Android device do not affect the time measurements, as those are captured to the TPM server side.

Tests were executed for the TSS2.0 stack and the FutureTPM stack. In these instances, the emulated software QR TPM acts as a server which receives TCP requests from a client, a command from the TSS library. A dedicated TCP connection is built for each individual command, regardless of session and context. More details about the particulars of this setup can be found in D5.1 [14]. Regarding the interaction with the HW TPM2.0, the Intel TSS was used to fire commands directly to the hardware TPM of the above-mentioned TPM server.

All test results found herein for the demonstrator#1 are the weighted average (LWMA) of **100 consecutive runs**, in order to provide an objective performance measurement that spans through time and possible system conditions.

## 2.3 Implementation Path Report

During the 1st phase of the run of the demonstrator within the FutureTPM project, the user stories realised had to do mostly with the **implementation of the sealing and unsealing functionality of the sensitive tokens.** This has been achieved by integrating the TPM2.0 stack in a dedicated TPM server, where all the necessary TPM calls are proxied. The mobile application, only after a secure authentication of the user and establishment of a secure channel with the TPM and authentication server, will be able to use the TPM functionalities. Taking a step ahead, the current implementation considers the FutureTPM stack, which is used for deploying the new algorithms and libraries provided by the project in the Secure Mobile Wallet and Payments scenario.

The major challenges faced during this implementation had to do with the instrumentation of the SW FutureTPM stack for creating an approach of measuring the QR TPM performance by having the lowest possible interference to its operational profile. Towards this direction, minor modifications applied to the TSS engine in order to acquire the timestamps of TPM commands execution, so that to calculate the performance timings. Additionally, moving for the legacy TPM2.0 stack to the FutureTPM stack, minor modifications were needed to the utilised TPM commands for replicating the use case and demonstrating the sealing, unsealing and symmetric key generation functionalities.

### *2.3.1 User Stories Realisation*

Out of the User Stories and Test Cases described in D6.1 [1] which were scheduled for this period, the following has been executed:

| Description |
|---|
| **User Story Title:** *INDEV.AU.1 - As an Individual User I want to log in to the INDEV Service and keep safe the bearer token.* |
| **Workflow Developed:** A preliminary step of the workflow is the generation and storage of a Control-flow graph (CFG). Then, the workflow proceeds to the **registration** of the Android user to the TPM Server leveraging FIDO U2F (only the first time). The user registration process relies on a challenge/response protocol, as shown in Figure 2. Once the user is registered, she is **authenticated** to the TPM Server leveraging FIDO U2F when she wants to perform a TPM functionality, following the procedure shown in Figure 3. The Android application **seals** the Bearer Token in the dedicated TPM, based on the handle and the recorded CFG, by invoking the TSS stack on the dedicated TPM server. |
| **Issues Encountered:** No issues encountered. |
| **Status:** Completed |
| **Degree of Realisation:** Full |
| **Comments (if any):** N/A |

| Description |
|---|
| **User Story Title:** *INDEV.AU.2 - As an Individual User I want to use an external service to generate tokens for my credit card that go directly in the TPM and avoid revealing my credit card to the server.* |
| **Workflow Developed:** The Android user authenticates to the TPM Server leveraging FIDO U2F when she wants to perform a TPM functionality (see Figure 3). Then, she provides her credit card to a 3d party service to generate the necessary Financial Token for a financial transaction finalization. The user unseals the Bearer Token based on the recorded CFG state (INDEV.AU.1), and the Token is provided to the INDEV Server. The server forwards the token to the 3d Party service to generate the Financial Token. The 3d Party service forwards the generated Financial Token to the server and the server seals the Financial Token. |
| **Issues Encountered:** No issues encountered. |
| **Status:** Completed |
| **Degree of Realisation:** Full |
| **Comments (if any):** N/A |

### *2.3.2 Unit Test Results*

The following unit test, which correspond to the user stories mentioned above, have been implemented during this period.

| Test Case MWP1 | |
|---|---|
| **Reference Code** | MWP1 |
| **Components** | Mobile App lib |
| **Description** | This unit test extends the functionality of the FUTURETPM04 and aims at verifying the correctness of the sealing and unsealing functionalities of the Bearer Token, needed for the authorization of the device, based on the correct FIDO handle token reflected in the PCRs states. (INDEV.AU.1) |
| **Status** | Performed |
| **Unit Tests Results** | Bearer Token is successfully sealed and unsealed based on the correct PCR state. |

| Test Case MWP2 | |
|---|---|
| **Reference Code** | MWP2 |
| **Components** | Mobile App lib |
| **Description** | This unit test extends the functionality of the FUTURETPM04 and aims at verifying the correctness of the sealing and unsealing functionality of the Financial Token, needed for the completion of the financial transaction, based on the correct FIDO handle token reflected in the PCRs states. (INDEV.AU.2) |
| **Status** | Performed |
| **Unit Tests Results** | Financial Token is successfully sealed and unsealed based on the correct PCR state. |

### 2.3.3 KPIs Measured

During the first phase of the operation of the demonstrator, a set of KPIs that have to do with the sealing and unsealing functionalities has been tested. For these experiments, performance has been measured, by employing the “Kyber” algorithm in the 3rd mode (k=3).  More details are presented in the next KPIs, which have been used to measure the core processes of the reference scenario.

#### 2.3.3.1 Quantitative Metrics

Table 2 and Table 3 below show the time differences of the demonstrator between TPM 2.0 and QR-TPM. Entries in bold report the total time necessary to execute a demonstrator functionality, while time entries with regular style report the execution time of TPM commands for the demonstrator functionality.

Regarding the quantitative evaluation of the project, the acceptance criteria set initially in D6.1 [1] for the scenarios of the first release of the demonstrator have been met in their majority. It needs also to be noted that at the current state of the project the deployed application considers the current parameters as sufficient for the indicated scenario.

In general, the PCR commands are slower, since the QR-TPM supports more PCR banks with the SHA3 algorithm and because the kernel extends all allocated banks. In addition, the key creation commands cannot be compared because RSA key generation is not deterministic, while Kyber key generation is deterministic. Additionally, "Kyber" algorithm is deployed using the 3rd mode (k=3).

The next tables summarise the timings of the SW implementation of the QR TPM commands for this demonstrator, at the current released version, and are compared to the HW TPM2.0 command timings of the equivalent operations. Note that, a different TSS is used for measuring the performance of the SW QR TPM and the HW TPM2.0. For the former case, the IBM TSS is used to trigger the commands' execution to the SW-based TPM, while for the latter case the intel TSS is used for interacting with the HW-based TPM2.0 of the TPM server of the demonstrator. The decision for using the Intel TSS was made due to the previous developments of the project (Section 2.2.1) where the development of the TPM tracer and any interaction with the TPM2.0 was made thought the Resource Manager of the TPM2.0 using the intel TSS. Hence, in order to be aligned with the previous developments of the project and to adapt to the future ones, we proceeded to a demonstration which provides an overview of the previous setup and the latest one, which is based on the IBM TSS.

The timings focus on a) the **sealing functionality** for the Bearer and Financial Tokens, b) the **unsealing** functionality for the Bearer Token and c) symmetric **key generation** to encrypt financial transaction history logs. In addition, timings for the U2F Registration and Authentication processes are provided.

| HW TPM Command | Intel TSS Timings (sec) | TPM2-tools Command | Application Timings (sec) |
|---|---|---|---|
| **FIDO U2F Registration** | **0.032 + 0.031 [=0.063]** | | |
| **FIDO U2F Authentication** | **0.016 + 0.017 [=0.033]** | | |
| **Scenario Initialisation** | **0.000811237** | | **4.43236927** |
| **CC_CreatePrimary** | 0.000132 | **tpm2_createprimary** | 4.36693603 |
| **CC_ContextSave** | 0.000137747 | | |
| **CC_PCR_Extend** | 0.0000935 | **tpm2_pcrextend** | 0.01396532 |
| **CC_StartAuthSession** | 0.0000977 | **tpm2_createpolicy** | 0.05146792 |
| **CC_PCR_Read** | 0.00010853 | | |
| **CC_PolicyPCR** | 0.0001201 | | |
| **CC_PolicyGetDigest** | 0.00012166 | | |
| **Seal Bearer Token** | **0.000861264** | | **0.30136801** |
| **CC_ContextLoad** | 0.00011864 | **tpm2_create** | 0.06000505 |
| **CC_Create** | 0.000113 | | |
| **CC_ContextLoad** | 0.000125107 | **tpm2_load** | 0.12000210 |
| **CC_Load** | 0.00012154 | | |
| **CC_ContextSave** | 0.0001346 | | |
| **CC_ContextLoad** | 0.000120847 | **tpm2_evictcontrol** | 0.12136086 |

| HW TPM Command | Intel TSS Timings (sec) | TPM2-tools Command | Application Timings (sec) |
|---|---|---|---|
| **CC_Evictcontrol** | 0.00012753 | | |
| **FIDO U2F Authentication** | **0.018 + 0.020 [=0.038]** | | |
| **Unseal Bearer Token** | **0.00072306** | | **0.1307273** |
| **CC_StartAuthSession** | 0.00011721 | **tpm2_unseal** | 0.06087464 |
| **CC_PCR_Read** | 0.00010834 | | |
| **CC_PolicyPCR** | 0.00012113 | | |
| **CC_Unseal** | 0.00012099 | | |
| **CC_FlushContext** | 0.00011853 | | |
| **CC_Evictcontrol** | 0.00013686 | **tpm2_evictcontrol** | 0.06985266 |
| **FIDO U2F Authentication** | **0.015 + 0.016 [=0.031]** | | |
| **Seal Financial Token** | **0.000883173** | | **0.30647682** |
| **CC_ContextLoad** | 0.000127333 | **tpm2_create** | 0.05986540 |
| **CC_Create** | 0.00011825 | | |
| **CC_ContextLoad** | 0.000132587 | **tpm2_load** | 0.11995824 |
| **CC_Load** | 0.00011963 | | |
| **CC_ContextSave** | 0.000138973 | | |
| **CC_ContextLoad** | 0.00011724 | **tpm2_evictcontrol** | 0.12665318 |
| **CC_Evictcontrol** | 0.00012916 | | |

Table 2: Demonstrator #1 – Comparison of Timings between the TSS and the Application perspectives using TPM2.0 (HW).

Table 2 provides a side-by-side comparison of the timings for the HW TPM2.0, as those captured from two perspectives, namely at the TSS and the Application levels. As can be seen, a TPM command invocation at the Application layer may imply multiple command executions on behalf of the TSS. For example, the *tpm2_createprimary* command triggers the *CC_CreatePrimary* and *CC_ContextSave* commands of the Intel TSS.

As it can also be observed, the timings measured by the Intel TSS reflect the time needed to perform a command execution directly on the HW TPM2.0 of the TPM server. That is, the command execution occurs quite fast in contrast to the time measurements from the Application point of view. In fact, this behaviour is reflected in all the core functionalities. The "*Seal Bearer Token"* functionality takes 0.000861264 secs to complete by the Intel TSS, while 0.30136801 secs are needed for the Application to perform this operation. This is justified by the fact that, any application designed to interact with the HW TPM2.0 needs to actually trigger TPM commands through the TSS which acts as an intermediate entity between the application and the TPM. Hence, this additional execution overhead is justified as we need to consider the time needed for the application to interact with the TSS stack.

Regarding the *TPM2_CreatePrimary* execution time for the TPM2.0, this command lasted longer than expected. This is because in the context of the use case scenario the command aims to generate a secure RSA key-pair. However, this process is not deterministic and hence, the random generation of a secure key-pair may require a reasonable amount of time.

The timings for the FIDO U2F Registration and Authentication process are independent from the TPM operation. That is why, these performance timings are replicated in Table 3. In the context of this use case scenario, the Android user needs to register to the service. Every time a request is sent to the TPM dedicated server, it is authenticated in the background. Table 2 contains the timing for the aforementioned operations. Note that, the timings do not include the latency of the communication channel between the two entities, neither the time required for the user to interact with the U2F Security Key, as those measurements depend on the network specifications and the users' reflection respectively. That is, the captured timings refer to the server-side processes of handling the registration of a user and the authentication of each received request. Both the registration and the authentication timings consist of two measurements, which can be seen in Table 2. The one refers to the challenge handling process, and the other to the actual operations of the registration/authorisation operations, such as the creation of a new user in the database, lookup queries for registered users, signature checking etc. Overall, the Registration process lasts for twice the time of the Authentication process, which is reasonable.

Table 3 provides a comparison over the timings taken from the TSS and the Application perspectives for the SW-based QR TPM. As expected, the behaviour revealed by the timings of Table 2 for TPM2.0, is also reflected for the timings of the SW-based QR TPM. More specifically, the time measurements taken from the side of the TSS denote the command execution for SW-based QR TPM performs fast. The *"Seal Bearer Token"* functionality needs 0.0732692 secs to complete, while for the Application the same operation takes 1.027213278 secs. This time deference is reasonable as the TSS acts as an intermediate between the App and the TPM. It must be stated, that for QR TPM performance measurement the IBM TSS was utilised.

| QR TPM Command | TSS Timings (sec) | QR TPM Command | Application Timings (sec) |
|---|---|---|---|
| **FIDO U2F Registration** | **0.032 + 0.031 [=0.063]** | | |
| **FIDO U2F Authentication** | **0.016 + 0.017 [=0.033]** | | |
| **Scenario Initialisation** | **0.0715622** | | **0.986897155** |
| **CC_CreatePrimary** | 0.0102342 | **createprimary** | 0.120857779 |
| **CC_ContextSave** | 0.0105845 | **contextsave** | 0.263207519 |
| **CC_PCR_Extend** | 0.0101156 | **pcrextend** | 0.120562730 |
| **CC_StartAuthSession** | 0.0101682 | **startauthsession** | 0.120433829 |
| **CC_PCR_Read** | 0.010166 | **pcrread** | 0.120319939 |
| **CC_PolicyPCR** | 0.0101389 | **policypcr** | 0.120689600 |
| **CC_PolicyGetDigest** | 0.0101548 | **policygetdigest** | 0.120825759 |
| **Seal Bearer Token** | **0.0732692** | | **1.027213278** |
| **CC_ContextLoad** | 0.010632 | **contextload** | 0.131573730 |
| **CC_Create** | 0.0103062 | **create** | 0.120626840 |
| **CC_ContextLoad** | 0.010628 | **contextload** | 0.131922750 |

| QR TPM Command | TSS Timings (sec) | QR TPM Command | Application Timings (sec) |
|---|---|---|---|
| **CC_Load** | 0.0102251 | **load** | 0.120615570 |
| **CC_ContextSave** | 0.0105145 | **contextsave** | 0.263207519 |
| **CC_ContextLoad** | 0.0106861 | **contextload** | 0.131504579 |
| **CC_Evictcontrol** | 0.0102773 | **evictcontrol** | 0.127762290 |
| **FIDO U2F Authentication** | **0.018 + 0.020 [=0.038]** | | |
| **Unseal Bearer Token** | **0.0610737** | | **0.733673946** |
| **CC_StartAuthSession** | 0.0102885 | **startauthsession** | 0.125108659 |
| **CC_PCR_Read** | 0.0101359 | **pcrread** | 0.120378019 |
| **CC_PolicyPCR** | 0.0102411 | **policypcr** | 0.120288340 |
| **CC_Unseal** | 0.0101065 | **unseal** | 0.120404149 |
| **CC_FlushContext** | 0.0101262 | **flushcontext** | 0.120281479 |
| **CC_Evictcontrol** | 0.0101755 | **evictcontrol** | 0.127213300 |
| **FIDO U2F Authentication** | **0.015 + 0.016 [=0.031]** | | |
| **Seal Financial Token** | **0.0727406** | | **1,147778298** |
| **CC_ContextLoad** | 0.0105624 | **contextload** | 0.132756830 |
| **CC_Create** | 0.0101690 | **create** | 0.120763869 |
| **CC_ContextLoad** | 0.0105212 | **contextload** | 0.131351670 |
| **CC_Load** | 0.0101926 | **load** | 0.120285510 |
| **CC_ContextSave** | 0.0105149 | **contextsave** | 0.384044250 |
| **CC_ContextLoad** | 0.0105534 | **contextload** | 0.131452940 |
| **CC_Evictcontrol** | 0.0102271 | **evictcontrol** | 0.127123229 |

Table 3: Demonstrator #1 – Comparison of Timings between the TSS and the Application perspectives using SW-based QR TPM.

For performing a cross-comparison between the HW TPM2.0 and SW QR TPM, one needs to take a look over the results of both tables. As can be observed based on the TSS timings, the HW TPM2.0 performs faster in contrast to the SW QR TPM. In fact, this can be justified since HW TPM is a dedicated chip destined to perform cryptographic operations and, on the other hand, the SW QR TPM operates on generic hardware by utilising the CPU of the server. This difference is reflected in the timing difference of the "*Seal Bearer Token*" functionality, where HW TPM needed 0.000861264 secs, while SW QR TPM completed the task in 1.027213278 secs. It must be stated, that the comparison between the HW TPM and SW QR TPM for this use case is not straight forward, as the timings are taken using different TSS variants (Intel/IBM TSS). However, the notable performance difference cannot be attributed to this fact, but it is the intrinsic difference of the HW and SW which affects the performance. Regarding the performance from the Application perspective, the discrepancy of the timings between the HW TPM and SW QR TPM is not as evident as is from the TSS perspective, but still, the HW-based TPM application performs faster, apart from the initialization phase, where the non-deterministic RSA key generation process adds a performance overhead.

Concluding, the timings of Table 2 and Table 3 offer a performance overview from multiple perspectives. The timings are acceptable from the business point of view for the current demonstrator.

The next table showcases the KPIs corresponding to the implemented use cases, as identified in D6.1 and measured in this deliverable. Note that the lower performance timings, i.e., the timings taken from the application perspective, were used in the next table.

| Id | Metric | Target Value | Acceptance criteria | (M)andatory / (G)ood to Have / (O)ptional | Measured by M24 | Comments |
|---|---|---|---|---|---|---|
| 1 | Amount of sealed objects | >=2 | =2 | M | With TPM2.0: **100%** With FutureTPM: **100%** | Target Achieved. Successfully sealed both Bearer and Financial Tokens. |
| 2 | Performance of sealing functionality within the domain of ms | <=1000 ms | <=2000 ms | M | With TPM2.0: **306.48 ms** With FutureTPM: **1027.21 ms** | Target Achieved. The sealing performance is below the acceptance threshold. |
| 4 | Performance of the FIDO Registration | <=2 sec | <=3 sec | M | With TPM2.0: **0.063 ms** With FutureTPM: **0.063 ms** | We consider only the server-side processes for user registration, excluding network latency and user's interaction with the U2F Security Key. Target achieved. |
| 5 | Performance of the FIDO Authentication | <=1.5 sec | <=2 sec | M | With TPM2.0: **0.0038 ms** With FutureTPM: **0.0038 ms** | We consider only the server-side processes for authentication, excluding network latency and user's interaction with the U2F Security Key. Target achieved. |

Table 4: Demonstrator #1 – Quantitative Metrics by M24

#### 2.3.3.2 Qualitative Metrics

The protection of sensitive tokens has been achieved with the current version of the software-based implementation of FutureTPM, which has been released by the project in order to kick start the demonstrators, and it covered the main scenarios that have been defined for the first version of the demonstrators. Note that, the 4th qualitative metrics regarding the "User authentication through the use of TPM", is shifted to the 2nd release of the demonstrators. This is because this metric is related to the *INDEV.AU.5* user story, which is destined to be deliver in the 2nd release.

| Id | Metric | Target Value | (M)andatory / (G)ood to Have / (O)ptional | Measured by M24 | Comments |
|---|---|---|---|---|---|
| 1 | Protection of sensitive tokens | Supported | M | With TPM2.0: Yes<br>With FutureTPM: Yes | Successfully sealed both Bearer and Financial Tokens. |
| 4 | User authentication through the use of TPM | Supported | M | With TPM2.0: Not tested yet<br>With FutureTPM: Not tested yet | Testing for this KPI has been shifted to the 2nd phase of the demonstrator. |

Table 5: Demonstrator #1 – Qualitative Metrics by M24

### *2.3.4 Plan for the next Period*

During the next period, the rest of the user stories as defined in deliverable D6.1 [1] will be executed, by leveraging the HW-based TPM, including the integrity verification history log and operational correctness of the Android device and the FIDO U2F.

## 2.4 Conclusions

Sealing and unsealing have been successfully implemented in this demonstrator using the software implementation of the FutureTPM. We contacted a thorough comparison on the performance of both HW TPM2.0 and SW-based FutureTPM, in order to provide deep insights on their operational behaviour in the context of the demonstrator. The results of Table 2 and Table 3 advocate that the performance of the software implementation of the FutureTPM meets the performance KPIs. The time discrepancies among the contacted measurements are justified by the nature of the TPMs (Software/Hardware) and the interception placement (TSS/Application) for capturing the timings. Further experiments will be conducted for the 2nd release of the demonstrator, in order to evaluate the performance for the rest of the user stories under the distributed nature of the overall architecture of the dedicated TPM server and the resources needed to work with QR algorithms and schemes. Overall, the performance of FutureTPM meets the goals of the demonstrator.

# Chapter 3 Demonstrator #2 – *Activity Tracking Demonstrator*

## 3.1 Demonstrator Overview

The S5Tracker demonstrator is based around the infrastructure build by S5 that is called S5Tracker. The S5Tracker is a cloud-based analytics engine developed by S5 acting as a data handling information environment of personalised and interlinked data streams related to activities performed mostly by individuals. The S5Tracker can be used for creating information-rich user profiles, based on activities recorded in diverse ICT communication channels and devices, pulled automatically, or inserted into the system in a semi-automatic manner by users themselves. The current information entry sources supported include APIs of specific IoT devices (e.g. Apple Health, Fitbit, Nike+, Garmin, Smart devices, etc.), Web2.0 social platforms that record users activity (such as Facebook, Twitter, etc.), as well as other smart devices that could be connected to the platform such as Smart Home kits, etc.

As in any cloud-based data analytics engine, the development, expansion and the deployment of the service suffers from a set of systemic challenges that require continuous integration and testing efforts, as well as big time investments to undertake strategic decisions guaranteeing the service's performance and availability. In more detail, the main challenges faced at the moment, as the service resides in a public cloud provider operating as a centralised application, have to do with:

- Data sharing, privacy, confidentiality and security considerations, both at the cloud-based infrastructure as well as in the upcoming S5Tracker mobile application service;
- Data volume handling and scalability issues;
- Data processing power and system performance optimisation over the cloud-based offering.

As such, a strong, but also pain point of the S5Tracker is the Data Anonymization and Privacy preservation service that can be used to either secure the data and the details of each user to not be accessible from other parties accessing the platform, and also the generation of aggregated "User Personas" which are fictional representative users, that can be globally accessible by analysts, in order to create reference cases.

### *3.1.1 Demonstrator Needs and Challenges*

By utilizing the infrastructure to be made available by FutureTPM, the Activity Tracking demonstrator will be in a position to include into the overall ecosystem of its operation trusted devices. They are used at the edge of the infrastructure (e.g. at the data generation and collection points, as well as the data analysis points), which in turn will provide guarantees regarding privacy and security. These are considered highly important for the data that is being exchanged over the suggested infrastructure in order to avoid data forging incidents and data leaks, and at the same time care for privacy preservation and anonymized data delivery, while such features will be able to provide an extra layer of trust with regards to the mandates of GDPR, allowing data owners and data collectors to trust even more the entities that take part in the overall information exchange.

As such, the use of FutureTPM allows the trusted communication and information sharing between entities of the overall ActivityTracker and will provide an extra layer of privacy and trust for the users of the platform, as well as the security primitives necessary to safeguard that data uploaded to the platform is genuine and comes from the authenticated endpoints.

### *3.1.2 Demonstrator Architecture*

The demonstrator that is being designed and developed during the FutureTPM project is based on a refactored architecture of the current S5Tracker infrastructure of the company, bringing into the picture TPM methods that allow for highly privacy-preserving information exchange. In this frame, the demonstrator has three main actors and three different components where each one of these actors operates one component.

The actors identified, which play significant roles in the data value chain of the use case, and have security and privacy considerations, are the following:

- An **Individual User**, who is a user that collects his own data from specific sensors and social media accounts;
- A **Data Analyst**, who gets access to the data (anonymised data or access to personal data) to perform certain analyses;
- The **S5Tracker Analytics Engine** which is not an actual user but a system role that is responsible for the operation of the S5Tracker Analytics Engine.

The different components are the following:

- **S5PersonalTracker** - A device on the side of the "individual user" which is used primary for data collection and data push to the S5Tracker Analytics Engine;
- **S5Tracker Analytics Engine** – A central cloud-based service, which gets data from the S5PersonalTracker and performs some analyses online, managing individuals' data;
- **S5DataEdgeAnalysis** – A computer interface used by the Data Analyst, that connects to the S5Tracker to fetch data and run online queries

As shown in the next figure, both the S5PersonalTracker and the S5DataAnalysis interfaces connect and exchange data with the S5Tracker Analytics Engine. The core focus of the use case will be to utilise **software TPM methods**, both at the S5PersonalTracker and at the S5DataAnalysis sides, to realise a holistic environment of privacy preservation and trust generation.

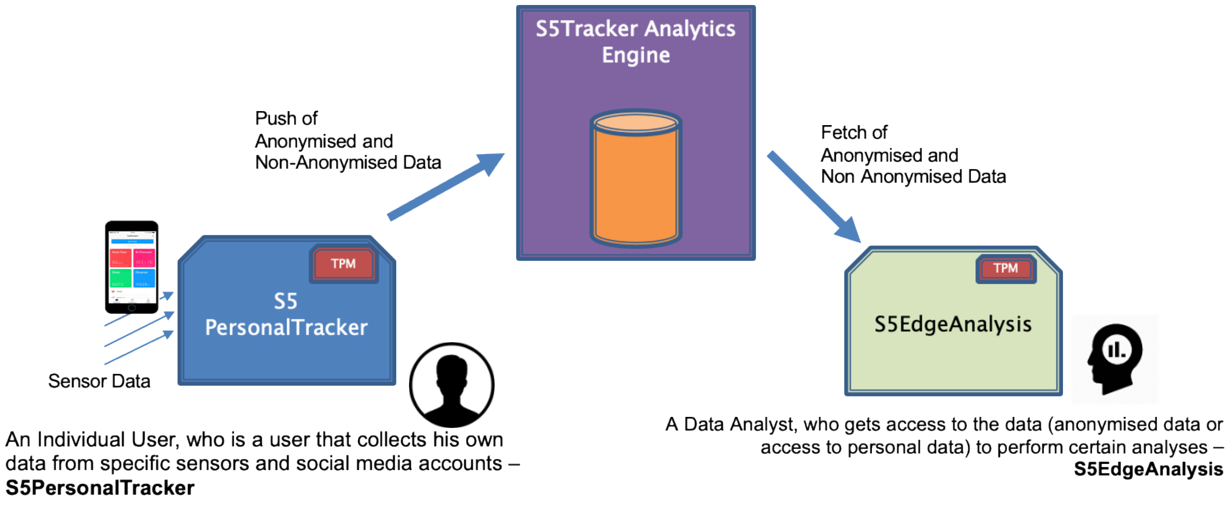

S5Tracker Analytics Engine
Push of Anonymised and Non-Anonymised Data
Fetch of Anonymised and Non Anonymised Data
TPM
S5 PersonalTracker
TPM
S5EdgeAnalysis
Sensor Data
An Individual User, who is a user that collects his own data from specific sensors and social media accounts – **S5PersonalTracker**
A Data Analyst, who gets access to the data (anonymised data or access to personal data) to perform certain analyses – **S5EdgeAnalysis**

Figure 4: Demonstrator #2 – Main Actors and Entities

In this context, privacy regarding the data owner could be achieved by enabling interconnection between the S5PersonalTracker and the S5Tracker Analytics Engine through Direct Anonymous Attestation, while at the same time, data sharing modalities towards the S5DataAnalysis side would be safeguarded, by providing access only to trusted devices for data fetching and analysis, which would be configured according to the data sharing principles of the overall platform (so that for example data cannot be exported to a storage medium.

During this period, implementation focused on the DAA part between the S5PersonalTracker and the Analytics Engine infrastructure, with the focus of allowing the former to sign and send payload to the latter, which verifies the payload and stores it in the appropriate database, depending whether the payload sent is anonymous (thus contributing to building anonymised "personas"), or eponymous, by using specific basenames, which then is stored to the personal bucket of a user in the database. The exact architecture of the overall infrastructure, as revised to fit the TPM modules is shown in the next figure.

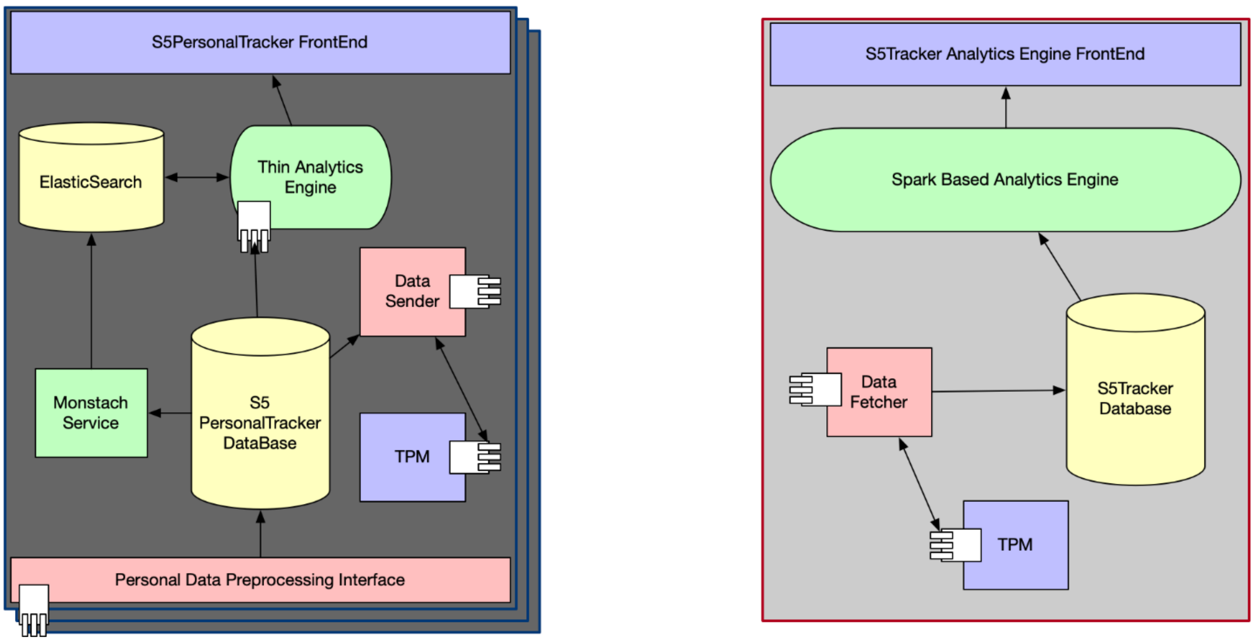

S5PersonalTracker FrontEnd
ElasticSearch
Thin Analytics Engine
Data Sender
Monstach Service
S5 PersonalTracker DataBase
TPM
Personal Data Preprocessing Interface
S5Tracker Analytics Engine FrontEnd
Spark Based Analytics Engine
Data Fetcher
S5Tracker Database
TPM

Figure 5: Demonstrator #2 – Architecture showing the 2 entities concerned for the use cases till M24

## 3.2 Emulated System Description

In order to concretely test the proposed architecture, both the S5PersonalTracker and the S5Personal tracker Engine have been executed within docker containers.

For both machines (including the TPM host), a machine running macOS 10.15.3, using an Intel(R) Core(TM) i7-8850H CPU operating at a clock frequency of 2.60GHz, having also 16GB of DDR4 RAM running at 2400 MHz.

The Docker running Fedora 31, and the docker engine version is 19.03.5; utilising all 6 physical cores (with 2 threads each) and occupying 6GB of memory with 1.5GB swap.

As in the other use cases, tests were executed for the TSS2.0 stack and the FutureTPM stack. In these instances, the emulated software QR TPM acts as a client which uses the LDAA method to contact a server for being identified as an attested machine that can push some data.

All test results found herein for the Demonstrator#2 are the average of 100 consecutive runs, in order to provide an objective performance measurement that spans through time and possible system conditions.

## 3.3 Implementation Path Report

During the 1st phase of the run of the demonstrator within the FutureTPM project, the user stories realised had to do mostly with implementing the LDAA protocol that concerns the joining of the S5PersonalTracker to the network, the signature of payload packages and the verification of those by the S5Tracker Analytics Engine, for storing them in the appropriate buckets (or dropping them in case these were not verifiable). As such, the whole process that deals with LDAA has been implemented between those entities, and the according user stories have been successfully implemented. Initially, this has been achieved by integrating the TPM2.0 stack in the existing infrastructure, which was then replaced with the FutureTPM stack, by using the new algorithms and libraries provided by the project.

The major challenges faced during this implementation had to do with certain delays that caused runtime errors and sync errors between the two different entities, with the main reason for those being the size of the payload and the delays imposed by the TPM in the signing and verifying the data. Therefore, it was necessary to implement a mechanism that truncated the payload into smaller packages, that were faster to sign and verify, and overcome this obstacle. Moreover, when shifting to the FutureTPM stack, severe delays were experienced in the execution of the TPM commands, which was a logical consequence of the number of computations necessary for the QR algorithms to get configured and executed. To overcome this challenge, a specific parameter in the QR FutureTPM stack has been used, which selects the weakest security parameters to use in the LDAA, in an effort to boost performance.

### 3.3.1 User Stories Realisation

Out of the User Stories and Test Cases described in D6.1 [1] which were scheduled for this period, the following has been executed:

| Description |
|---|
| **User Story Title:** *S5.IU.1 - As an Individual User I want to provide authenticated data to the S5Tracker Analytics Engine, so that I can be served with user-specific services such as notifications send by the analysts.* |
| **Workflow Developed:** For this use case, the S5PersonalTracker had to acquire the TPM credentials by using the Join() command, and then select the payload to Sign(). The signed packets were sent to the S5Tracker Analytics Engine, which performed the Verify() command to check the signature and either store the payload in the bucket of the designated user, or drop it. |
| **Issues Encountered:** The issues encountered had to do with timeouts that resulted in messages not able to be signed. The workaround was to reduce the payload to smaller packages and use the -weak parameter in the sign() protocol. |
| **Status:** Completed |
| **Degree of Realisation:** Full |
| **Comments (if any):** N/A |

| Description |
|---|
| **User Story Title:** *S5.IU.2 - As an Individual User I want to provide anonymous and privacy-preserving data to the S5 Analytics Engine, so that data analysts can have a rich repository of activity data for exploration.* |
| **Workflow Developed:** For this use case, the S5PersonalTracker had to acquire the TPM credentials by using the Join() command, and then select the payload to Sign() by using a basename that has been common amongst all other clients. The signed packets were sent to the S5Tracker Analytics Engine, which performed the Verify() command to check the signature and either store the payload in the bucket of the "persona" user (thus anonymous), or drop it. |

| |
|---|
| **Issues Encountered:** The issues encountered in this user story were similar to S5.IU.1 as the only difference was the "base name" used. These had to do with timeouts that resulted in messages not able to be signed. The workaround was to reduce the payload to smaller packaes and use the -weak parameter in the sign() protocol. |
| **Status:** Completed |
| **Degree of Realisation:** Full |
| **Comments (if any):** N/A |

| **Description** |
|---|
| **User Story Title:** *S5.DA.1 - As a Data Analyst, I want to verify the integrity of the S5Tracker Analytics Engine Database, so that I can get data which is not tampered with.* |
| **Workflow Developed:** LDAA has been set up also in the S5Tracker Analytics Engine, and signing the snapshot of the database is pending. |
| **Issues Encountered:** LDAA has been set up also in the S5Tracker Analytics Engine, and signing the snapshot of the database is pending. |
| **Status:** OnGoing |
| **Degree of Realisation:** Zero |
| **Comments (if any):** This finalisation of this User Story has been shifted to the next period |

### *3.3.2 Unit Test Results*

The following unit test, which correspond to the user stories mentioned above, have been implemented during this period.

| **Test Case ATRACK01** | |
|---|---|
| **Reference Code** | ATRACK01 |
| **Components** | S5 PersonalTracker, Issuer |
| **Description** | This unit test aims at verifying that the S5 Personal Tracker correctly executed the Join() phase of the DAA protocol. The unit test checks the validity of the TPM of the host S5 PersonalTracker and the created DAA key. |
| **Status** | Performed |
| **Unit Tests Results** | The S5 PersonalTracker successfully acquires and can activate its TPM credentials for LDAA after communication with the Issuer |

| Test Case ATRACK02 | |
|---|---|
| **Reference Code** | ATRACK02 |
| **Components** | S5 PersonalTracker |
| **Description** | This unit test aims at verifying the signature (SIGN phase) of user's bunch of data using the DAA key. This unit receives the data from the users and then checks how the TPM forwards back the signed data, either anonymously or non-anonymously based on the use of a unique base-name. |
| **Status** | Performed / On-Going / Not Executed Yet / Skipped |
| **Unit Tests Results** | Payload signed by the S5 PersonalTecker is successfully signed, either on the anonymous or not modes |

| Test Case ATRACK03 | |
|---|---|
| **Reference Code** | ATRACK03 |
| **Components** | S5 Personal Tracker, S5 Analytics Engine |
| **Description** | This unit test aims at verifying the received signed data by the S5 Analytics Engine. It will validate the DAA VERIFY() phase, based on the use of the DAA key. |
| **Status** | Performed |
| **Unit Tests Results** | Payload received by the S5 Analytics Engine, that is generated and signed by the S5 PersonalTracker is verified. |

| Test Case ATRACK06 | |
|---|---|
| **Reference Code** | ATRACK06 |
| **Components** | S5 PersonalTracker, S5 Analytics Engine |
| **Description** | This unit test aims at verifying that the S5 Analytics Engine is able to unwrap and store (in a trusted manner) data that is sent by the S5 Personal Tracker to the database. |

| Test Case ATRACK06 | |
|---|---|
| **Reference Code** | ATRACK06 |
| **Components** | S5 PersonalTracker, S5 Analytics Engine |

| Test Case ATRACK06 | |
|---|---|
| **Description** | This unit test aims at verifying that the S5 Analytics Engine is able to unwrap and store (in a trusted manner) data that is sent by the S5 Personal Tracker to the database. |
| **Status** | Performed |
| **Unit Tests Results** | Once the payload is verified, it is unwrapped and is stored as necessary in the database of the Analytics Engine |

### *3.3.3 KPIs Measured*

During the first phase of the operation of the demonstrator, a set of KPIs that have to do with the establishment of a DAA scheme between the PersonalTracker interface and the Analytics Engine has been tested, using a simulated environment where data has been fabricated and send from the one end to the other to check the performance of the protocol.

For these experiments, performance has been measured, while in the case of the FutureTPM DAA implementation, the experiment has been conducted by employing its "weak" state, as this has allowed to retrieve the fastest possible responses from the TPM.

More detail is presented in the next KPIs, which have been used to measure the DAA performance.

#### 3.3.3.1 Quantitative Metrics

In terms of the quantitative evaluation of the project, the acceptance criteria set initially in D6.1 for the scenarios of the first phase of the demonstrator (M24) have been met in their majority using the -LDAA1 flag, aka "weak" parameter of the current software QR-TPM implementation. The -LDAA1 flag selects the weakest security parameters to use in the LDAA. The parameters are: q = 3329 (12 bits); cyclotomic polynomial of 256; k = 3; etc. This is faster because it foregoes security in favour of performance due to the inefficiency of the implemented LDAA algorithm. In case stronger security parameters are set, then performance is significantly reduced, as at the typing the FutureTPM implementation is under prototyping. It needs also to be noted that ate the current time, the concerned current business application considers the current parameters as enough for the indicated scenarios.

Unlike the applications that can replace RSA and ECC functionality with similar QR counterparts, the presented LDAA results and commands should not be interpreted in a similar way. Due to its memory requirements the current LDAA implementation is not deeply integrated in the TPM. The commands provided were implemented as a possible interface for a quantum-resistant accelerator. As such, there is not a one to one mapping to the non-quantum-resistant TPM. The integration of LDAA into the standard TPM commands was foregone because of backwards-compatibility concerns. Its addition would be disruptive to the standard commands, given the magnitude of the data that LDAA has to operate over, and break previous TSS compatible programs. In order to reduce the impact of the current LDAA implementation, we have decided to separate the commands such that we can test the current interface without interfering with other applications.

The next tables summarise the timings of the SW implementation of TPM commands for this demonstrator at the current version released in M21 of the project. The following table presents the timings of the complete sequence of commands for applying the DAA method with the use of the Software implementation of TPM2.0, measured at the application level of the Activity Tracker demonstrator.

| TPM Command | TPM2.0 Timings |
|---|---|
| **Application Timing** | |
| **Initialise and Join ()** | **1.190250sec** |
| TPM2_ReadPublic | 0.10071649sec |
| TPM2_GetCapability (TPM_PT_REVISION) | 0.099378898sec |
| TPM2_Create | 0.0024465sec |
| TPM2_Load | 0.094564203sec |
| T1 Host prepares | 0.097215602sec |
| T2 Issuer challenges | 0.0002133sec |
| TPM2_Activate_Credential | 0.099243sec |
| TPM2_Commit | 0.1003126sec |
| TPM2_Hash | 0.099998492sec |
| TPM2_Sign (ECDAA) | 0.1090904sec |
| T3 Host responds | 0.07920188sec |
| T4 Issuer verifies response | 0.0011676sec |
| T5 Issuer creates credential | 0.0024837sec |
| TPM2_Activate_Credential | 0.097036602sec |
| T6 Host verifies credential | 0.1101832sec |
| T7 Host checks pairings | 0.096997998sec |
| **Sign ()** | **1.116446383sec** |
| TPM2_GetCapability (TPM_PT_PERSISTENT) | 0.1094627 |
| TPM2_GetCapability (TPM_HT_PERSISTENT) | 0.12004409 |
| TPM2_ReadPublic | 0.1005071 |
| TPM2_GetCapability (TPM_PT_REVISION) | 0.099512797 |
| TPM2_Load | 0.073159998 |
| TPM2_Commit P1 (s2,y2) | 0.095290492 |
| T8B Host commits | 0.099133 |
| TPM2_Hash | 0.1102931 |
| TPM2_Sign (ECDAA) | 0.099247906 |
| T9 Host signs | 0.2097952 |

| TPM Command | TPM2.0 Timings |
|---|---|
| **Verify()** | **0.0382752sec** |
| T12B verify signature | 0.0079627sec |
| T13 Verifier checks pairings (S,C,Q) | 0.0111749sec |
| T12B verify signature | 0.0079627sec |
| T13 Verifier checks pairings (S,C,Q) | 0.0111749sec |

Table 6: Demonstrator #2 –Timings at Application Level using the TPM2.0 (SW)

Having as a reference point the timings with the current TPM2.0 implementation, the same amount of payload has been selected to perform the equivalent DAA operations (signing and verifying) with the QR software implementation of FutureTPM. In essence, LDAA has been used, utilising the same computational resources, and the timings at application and TSS level are the ones presented in the next table.

| QR FutureTPM Timings | | | |
|---|---|---|---|
| **Application Timing** | | **TSS Timing** | |
| **Initialise and Join ()** | **1.23763sec** | **Initialise and Join ()** | **0,097600915sec** |
| **New issuer ldaa** | 0.008677sec | | |
| **New host** | 0.000012sec | | |
| **Startup** | 0.157466sec | CC_Startup | 0,012818105 sec |
| **Createprimary** | 0.162345sec | CC_CreatePrimary | 0,012704095 sec |
| **Create** | 0.201239sec | CC_Create | 0,017840957 sec |
| **Load** | 0.282727sec | CC_Load | 0,027992065 sec |
| **Ldaa join** | 0.158229sec | CC_LDAA_Join | 0,013578031 sec |
| **Ldaa sign proceed** | 0.156717sec | CC_LDAA_SignProceed | 0,012667655 sec |
| **Join** | 0.110551sec | | |
| **Sign ()** | **38.909526** | **Sign ()** | **7,52866176 sec** |
| **Ldaa commit token link** | 0.167817sec | CC_LDAA_CommitTokenLink | |
| **Ldaa SignCommit (multiples)** | 34.926421 sec | CC_LDAA_SignCommit (multiples) | 6,89314087 sec |
| **Host sign proceed** | 0.603191sec | | |
| **Host generate challenge** | 0.189532sec | | |
| **Ldaa sign proof (multiple)** | 3.0220348 sec | CC_LDAA_Sign-Proof | 0,63552089 sec |
| **Sign merge** | 0.004312sec | | |
| **Verify ()** | **1.205691sec** | | |
| **Start verify** | 1.205691sec | | |

Table 7:Demonstrator #2 –Timings at Application and TSS Level using the FutureTPM QR Implementation (SW)

Even with the -weak parameter activated, there was a noticeable delay in specific TPM operations at the level of the Application, with the most severe being in the Sign() protocol that takes 38 times more than the current implementation. The other noticeable delays concerns the Verify() protocol that takes 33 times more than in the TPM2.0, however as the time required for this operation is lower than 1.5 seconds, they are acceptable from the business point of view for the current demonstrator.

The main justification for these delays has to do with the fact that LDAA signature (Sign()) is a multi-step process and there are certain steps which take longer than others. The first one is the required shared matrix between the host and the TPM. Since this matrix is very large, hundreds of MB, it would take longer to transfer it to the TPM than to regenerate it, so it was decided to regenerate the matrix using a pre-determined seed. This slows down the processing immensely because every time a call to a sign command is made, this matrix will have to be regenerated. The reason behind not using a cache is because the TPM doesn't possess any cache and in the Software implementation we wanted to be as true to the physical device as possible.

Another important point is the fact that the commitment scheme doesn't suit the TPM, i.e., the commitment scheme requires a vector matrix multiplication where the matrix is very large. Finally we have to be conscientious that we are emulating the TPM and thus every time we issue a command our OS needs to spawn the process, setup the TCP connection, run the required code by the TPM, transfer the data, wait until the SW-TPM responds, and finally kill the process and destroy all objects.

The next table showcases the KPIs corresponding to the implemented use cases, as identified in D6.1 and measured in this deliverable.

| Id | Metric | Target Value | Acceptance criteria | (M)andatory / (G)ood to Have / (O)ptional | Measured by M24 | Comments |
|---|---|---|---|---|---|---|
| 1 | Allowing only for trusted S5 PersonalTracker interfaces to interact with the S5Tracker Analytics Engine | 100% | 100% | M | With TPM2.0: 100%<br>With FutureTPM: 100% | Target Achieved.<br>Packets that have not be signed, are automatically dropped |
| 2 | Performance evaluation of process of sending and analysing an average set of daily collected personal data | -35% | -45% | M | With TPM2.0: 1,5 seconds<br>With FutureTPM: 40,1 seconds | Target not achieved Amount corresponds to 10Mb of data analysed, measuring in principle the sign and verify and unbundling |
| 3 | Performance evaluation of the | 800 ms | 2.000 ms | G | With TPM2.0: | Target not achieved but within the |

| Id | Metric | Target Value | Acceptance criteria | (M)andatory / (G)ood to Have / (O)ptional | Measured by M24 | Comments |
|---|---|---|---|---|---|---|
| | infrastructure during the Join() phase | | | | 1,190250 seconds<br>With FutureTPM: 1.23763 seconds | acceptable space |
| 4 | Improved perception of Individual Users' trust to S5PersonalTracker as a data hub[1] | 100% | 60% | G | With TPM2.0: 100%<br>With FutureTPM: 90% | Target not achieved but still acceptable<br>Users commented on the small delay experienced, which impacted negatively their perception of trust. |
| 5 | Performance evaluation of checking the integrity of S5 Tracking Engine and Data Analyst to avoid potential exploitation attempts | - 10% | - 25% | O | With TPM2.0: Not Tested yet<br>With FutureTPM: Not Tested yet | Testing for this KPI has been shifted to the 2nd phase of the demonstrator |

Table 8: Demonstrator #2 – Quantitative Metrics by M24

### 3.3.3.2 Qualitative Metrics

Support for DAA has been achieved with the current version of the software-based implementation of FutureTPM, which has been released by the project in order to kick start the demonstrators, and it covered the main scenarios that have been defined for the first version of the demonstrators.

| Id | Metric | Target Value | (M)andatory / (G)ood to Have / (O)ptional | Measured by M24 | Comments |
|---|---|---|---|---|---|
| 1 | Support DAA for enhanced privacy S5PersonalTracker | Supported | M | With TPM2.0: Yes | DAA support has been |

[1] To be measured with the use of structured Saaty scale questionnaires, addressed to a set of 25 selected users of the S5 Activty tracker users that will be introduced to the advantages brought by the TPM technology

| | | | | | |
|---|---|---|---|---|---|
| | | | | With FutureTPM: Yes | successfully implemented |

Table 9: Demonstrator #2 – Qualitative Metrics by M24

### *3.3.4 Plan for the next Period*

During the next period, the rest of the user stories as defined in deliverable D6.1 will be executed, while the existing user stories might be re-run in case an improved implementation of the LDAA is implemented in the course of the project.

## 3.4 Conclusions

As indicated above, LDAA has been successfully implemented in this demonstrator using the software implementation of the FutureTPM, however there have been some performance issues which are inherited by the nature and the overall architecture of the TPM, the resources needed to work with QR algorithms and schemes and also the business logic of the current demonstrator which worked with packaging, sending and unbundling data close to real-time.

In general, however, the results (except the signature process) are acceptable, even if not very close to the set targets. To mitigate the delay witnessed during the signature process there is an idea to alter a bit the business logic of the demonstrator, scheduling the signature and the sending of the payload to happen at off-peak times, using pre-programmed daemons that will run in the background on the S5 PersonalTracker instance. This however will result in users losing the ability to see their data in real-time on the cloud-based infrastructure (S5 Analytics Engine).

# Chapter 4 Demonstrator #3 – *Device Management Demonstrator*

## 4.1 Demonstrator Overview

The network device management demonstrator intends to show how system integrity challenges can be solved, at scale, in the scenario of a distributed telecommunications infrastructure composed of many network devices that are centrally managed, as described in Figure 6. In the demonstrator, network routers equipped with a QR-TPM are required to prove their hardware identity and software integrity to a Network Management System (NMS). The process is integrated with the usual management operations that the NMS is performing across the entire lifecycle of the router, from deployment stage through regular operation until their decommissioning, by leveraging the concept of Remote Attestation. Based on the outcome of this process, the NMS can decide whether any given router can be trusted for routing user traffic or, if it cannot be trusted, whether it should be avoided, e.g. by adjusting the routing policy on its neighbouring routers. The demonstrator pushes the industry state of the art by introducing new technologies and methods to address several of the challenges identified in the following section.

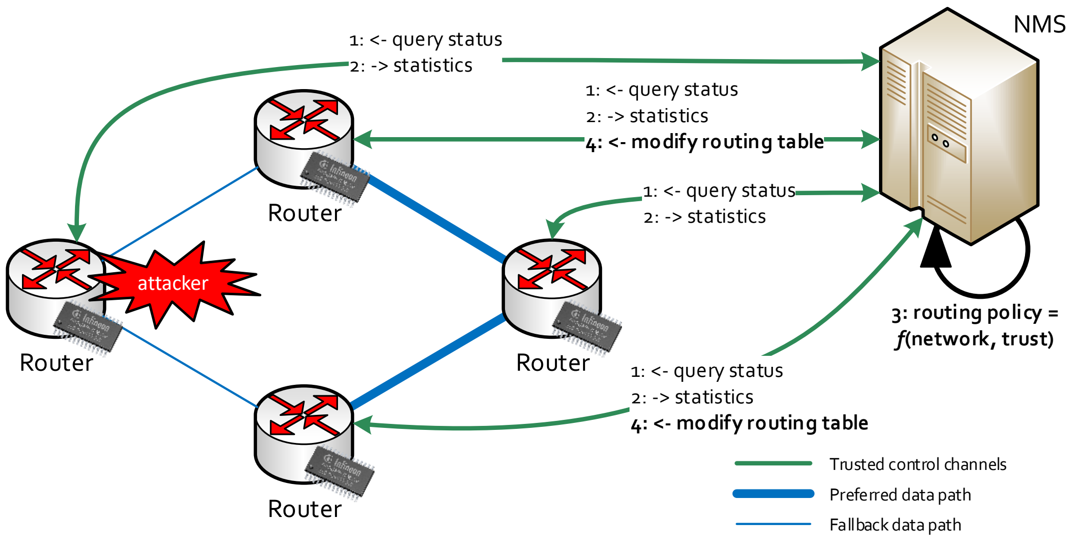


Figure 6: Demonstrator #3 – overall architecture and main entities

### *4.1.1 Demonstrator Needs and Challenges*

System integrity is a fundamental security aspect. It cannot be simply assumed that a certain security policy is enforced on a given system without having evidence that the part of the system responsible to enforce the policy, called the Trusted Computing Base (TCB), is trustworthy. The trusted computing paradigm promoted by the TCG addresses the need of verifiable evidence about a system and the integrity of its TCB and, to this end, the TPM and related TCG specifications provide both the foundational concepts, such as Measured Boot and Remote Attestation, as well as the necessary building blocks, such as the TPM and the TSS, to provide trusted computing capabilities to a wide range of ICT systems.

Still, there remain several challenges for the wide scale adoption of trusted computing and the telecommunication industry is a particular case. Often the adoption is not reaching its true potential due several aspects such as incomplete support infrastructures, lack of standard protocols, flexibility in the platform specifications, scalability, performance and availability concerns, and adoption in virtual infrastructures, to name a few. There is also a perceived aspect of incompleteness of integrity measurements or guarantees, due to the traditional focus of trusted computing on the system boot time or, at most, the load-time of applications, without covering system integrity beyond these stages, during system execution, which is especially important for high-availability systems that have months or years between reboots.

A different type of challenge is related to the long expected lifecycle of telecom routers, ranging from 10 to 15 or even 20 years. This means that the underlying cryptographic primitives of roots of trust such as the TPM need to remain trustworthy also beyond the horizon for practical quantum computer cryptanalysis. Using a QR TPM will provide insights into transitioning from classical cryptography to QR cryptography, with respect to performance and integration impact.

### *4.1.2 Demonstrator Architecture*

The entities in the demonstrator are:

- the **routers**, that route user traffic;
- the **NMS**, which manages the routers over TLS channels;
- the **RA Server,** which is responsible for attesting the routers.

The NMS augments the decision on the routing policy that is to be sent to the routers in the network, by factoring in the trust state of each router, in addition to the usual network-related parameters. The trust state is the result of Remote Attestation (RA), in which the measurements of the software loaded on a router is verified by an RA Server against reference values that characterize known (and thus trusted) software versions and configurations. If all routers are in respective trusted states, meaning that all the software running on the router is known to be good, the routing policies calculated by the NMS for the network will only depend on the network parameters. If a given router does not attest successfully, meaning that not all the software running on it is known, the NMS will push to the neighboring routers policies that divert traffic away from the untrusted router. This is done to the extent allowed by the network service level agreement, as some routers might be a single point of failure for a certain part of the network and avoiding them completely might break the network availability.

Each of the entities above interface with each other through standard REST APIs, as depicted in the user story diagrams in section 4.3.1. Each router is modeled as a virtual machine (VM) which uses a dedicated QR software TPM instance running on the hypervisor and exposed by qemu.

Compared to the architecture described in D6.1, the demonstrator introduces a new capability called Secure Zero Touch Provisioning (S-ZTP), which allows the automatic and secure establishment of trust, called enrolment, between a new router connected to the network and the NMS, without human intervention (other than plugging-in the router). S-ZTP eliminates the need of trust on first use or out-of-band trust establishment schemes, which, in practice, can be very unreliable from the perspectives of trust model, organization and cost. The result of successful enrolment of a router is materialized by the issuance of a TLS certificate that can be used to securely communicate with the NMS or with other routers.

## 4.2 Emulated System Description

The tests have been performed in a virtualized environment. The hardware used is an Intel i7-6700 CPU and 16 GB of RAM. The operating systems used are Ubuntu 18.04 in the host and Fedora 30 in the virtual machine. The hypervisor used is KVM.

To expose a virtual TPM in the virtual machine, libtpms and swtpm (both the non-QR and the QR version) have been installed in the host. Initial provisioning of the virtual TPM has been manually done with swtpm_setup.sh (for TPM 2.0) and with TSS utilities (for QR TPM).

Router software for remote attestation has been installed in the virtual machine, while the RA Server and the server endpoint of the TLS connection have been installed in the host.

The tests results have been obtained by running 100 times the binaries that implement the four main functionality of the demonstrator (AK creation, TLS key creation, TLS connection, and TPM quote), by collecting the results and by calculating both the non-weighted and the weighted (LWMA) average.

## 4.3 Implementation Path Report

In terms of design, the Device Management Demonstrator attempts to address the challenge of doing remote attestation in a complex software stack like that of a router, as described in section 4.1.1. Once the OS kernel is loaded, processes and files are loaded in parallel, driving an explosion of loading order paths that are almost impossible to match to a reference. Also, many processes often create their own files on the system (e.g. state, configuration files, logs etc.), files for which there can be no initial reference as for the executables. In addition, various processes can directly or indirectly interact with each other, such as through IPC or through successive file writes-reads, making it hard to evaluate the impact of an unknown process on the others. To this end, we have introduced the Comprehensive Integrity Verification (CIV), an architecture that allows to assess and/or preserve the integrity of the operating system TCB, at load time and during system execution, while ensuring predictability of the PCR values regardless of the order of loading of applications and reducing performance impact by dramatically reducing the number of TPM PCR extend.

CIV is building on the IMA and EVM features of the Linux kernel and introduces a new Linux Security Module (LSM) called *Infoflow,* which implements the Clark-Wilson integrity model [1]. It monitors the information flows between TCB processes and those outside the TCB and can prevent violations or record them in the TPM-protected IMA measurement list. CIV introduces a concept of *digest lists* to limit the reporting of measured software only to the case when that software is unknown (not added to the digest list). This approach ensures predictable PCR values and reduced usage of the TPM and, consequently, reduced performance impact. It also introduces *Simple Remote Attestation (Simple RA)*, to minimize the effort of integrating Remote Attestation in existing distributed architectures, by using implicit attestation over existing secure protocols (e.g. TLS), while addressing the lack of dedicated standard attestation protocols and thus mitigating interoperability concerns. The CIV overview is depicted in **Figure 7**.

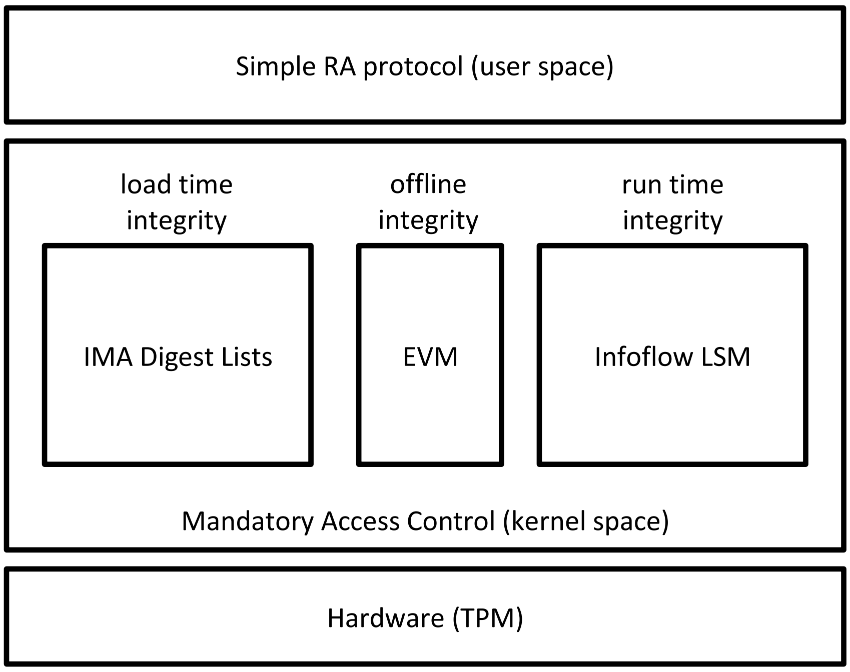


**Figure 7:** Demonstrator #3 – CIV architecture

The workflow is the following: CIV verifies immutable files by searching for a file digest in the digest lists provided by the software vendor. Alternatively, CIV detects/prevents offline attacks on mutable files by verifying the HMAC and detects/prevents online attacks by restricting through the Infoflow LSM the processes that are able to modify those files. Measurements produced by CIV (only if the verification failed) are used by the Simple Remote Attestation Protocol for reporting the integrity status of the router to the NMS.

The following changes have been made to the system architecture compared to D6.1 and are reflected in the revised user stories in section 4.3.1:

- introduce the ZTP client on the router, to support secure zero touch provisioning;
- the RA Server pushes the trust state of a router to the NMS, during S-ZTP (previously the NMS was pulling it from the RA server);
- the TLS CA has been moved from the RA server to the NMS.

Initially, the demonstrator used the regular TPM 2.0 and switched later to the QR software TPM 2.0 once it became available. This has required modifications across to all software stacks that use the TPM, in order to accommodate for larger key, buffer or command sizes due to the QR algorithms: TPM driver, TSS, SeaBIOS, qemu.

A major implementation challenge was how to modify existing software to perform implicit remote attestation. We wanted to use one of the most widely adopted library for implementing secure protocols, OpenSSL, without modifying it. Integrating our code in OpenSSL would have required a lot of effort, without the guarantee that upstream developers accepted it.

We opted for a more efficient approach, by implementing the implicit (and explicit) remote attestation functionality in a separate software called attest-tools. attest-tools consists of several components: RA lib (verifiers) to verify the integrity of the system attested from attestation data, RA lib (enrolment), to perform initial steps necessary for the subsequent remote attestation process, RA lib (skae), to parse attestation data from a X.509 extension. The API exposed by RA lib (skae) is suitable to use together with OpenSSL. In particular, it exposes a function that can be used as an additional method during the verification of the peer’s certificate.

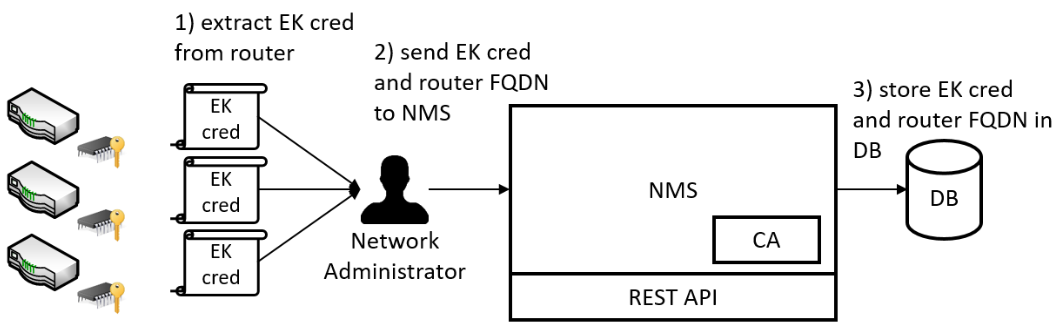
The additional verification method checks the Subject Key Attestation Evidence (SKAE) certificate extension, standardized by TCG. Implicit RA consists in verifying additional guarantees for the key used in the secure communication (e.g. TLS): the key is securely stored in the TPM and is associated with a good software configuration.

OpenSSL however, although it allows developers to specify a callback function called during the peer’s certificate verification, it does not give the possibility to specify additional parameters for that callback. This is a significant limitation because RA lib (skae) needs as input the requirements from the remote attestation verifier for the software configuration associated to the TLS key and additional information necessary to verify the SKAE (e.g. the certificate of the Attestation Key used to sign the TLS key).

We solved this issue by implementing a mechanism to load the data necessary for the SKAE verification at a different time than the time of the verification itself. Developers using attest-tools create a new data context and add to that context attestation data received from the peer, before the TLS connection is established. The data context is stored in a global variable, so that it is accessible by the callback function invoked by OpenSSL when the SKAE of the peer’s certificate is being verified. The result of the SKAE verification is stored in another structure called verifier context, which can be accessed by the application, to check if the SKAE verification was successful or which errors have been encountered by attest-tools.

Another implementation challenge was to modify the software which exposes the software TPM to the virtual machine. In particular, this software uses a fixed length buffer that was deemed by developers sufficient to store any TPM command and response. However, with the introduction of quantum resistant algorithms such as Kyber and Dilithium, the buffer length became insufficient due to the larger key size. Usage of the newly introduced algorithms became possible only after doubling the length of the buffer in many of these software components (i.e. Linux kernel, SeaBIOS, qemu).

### 4.3.1 User Stories Realisation

| Description |
|---|
| **User Story Title:** *HWDU.NA.1 – As a Network Administrator, I want to enrol the router with the NMS so that it is accepted in the network infrastructure.* |
| **User Story Confirmations:**<br>➢ *The router appears in the list of devices managed by the NMS based on its TPM-based identity* |
| **TPM Functionalities:**<br>➢ *NVRAM access* |
| **User Story Implementation:**<br> 1) extract EK cred from router<br>EK cred<br>EK cred<br>EK cred<br>2) send EK cred and router FQDN to NMS<br>Network Administrator<br>NMS<br>CA<br>REST API<br>3) store EK cred and router FQDN in DB<br>DB<br>**Figure 8:** Router registration |
| **Components:**<br>➢ **ZTP Agent**: Agent running on each router, responsible to initiate the enrolment process and to respond to implicit RA requests from the NMS.<br>➢ **RA Server**: Remote Attestation Server that exposes a REST API to routers for device enrolment and explicit RA.<br>➢ **RA Client**: Remote Attestation Client running on each router to generate TPM keys, quotes and CSRs and to send certificate requests to RA Server.<br>➢ **RA Lib (enrolment):** Library running on RA Server to perform enrolment of each router.<br>➢ **RA Lib (verifier):** Library running on RA Server to verify CSRs (for implicit RA) and quotes (for explicit RA).<br>➢ **NMS**: Network Management System. |
| **Workflow:** |

| |
|---|
| **1.** Extract EK cred from routers to be enrolled<br>➢ The Network Administrator accesses the router and extracts the EK credential from the TPM<br>**2.** Send EK cred and router FQDN to NMS<br>➢ The Network Administrator sends the extracted EK credential and the desired router FQDN to the NMS<br>**3.** Store EK cred and router FQDN in DB<br>➢ The NMS stores the EK credential and router FQDN in the NMS DB |
| **Issues encountered:** see Section 5.2 for the generic implementation challenges |
| **Status:** Completed |
| **Degree of realisation:** Full |

| **Description** |
|---|
| **User Story Title:** *HWDU.NA.2 – As a Network Administrator I want to define a trusted routing policy on the NMS so that the traffic is processed according to the trust states of routers.* |
| **User Story Confirmations:**<br>➢ *A routing policy depending, among others, on the trust state of routers is defined in the NMS.* |
| **Issues encountered:** - |
| **Status:** OnGoing |
| **Comments:** Planned for the second round of experimentation |

| **Description** |
|---|
| **User Story Title:** *HWDU.NA.3 – As a Network Administrator I want to enforce the trusted routing policy in the network to reduce the risk of traffic leaking by untrusted routers.* |
| **User Story Confirmations:**<br>➢ *Routing tables on adjacent routers are modified when the trust state of a given neighbouring router changes* |
| **Issues encountered:** - |

**Status:** OnGoing

**Comments:** Planned for the second round of experimentation

**Description**

**User Story Title:** *HWDU.NA.4 – As a Network Administrator I want to monitor the overall trust state of the network infrastructure.*

**User Story Confirmations:**

- *The NMS displays the trust state and routing table for each router in the network*

**TPM Functionalities:**

- Key storage, signing, decryption, platform configuration

**User Story Implementation:**

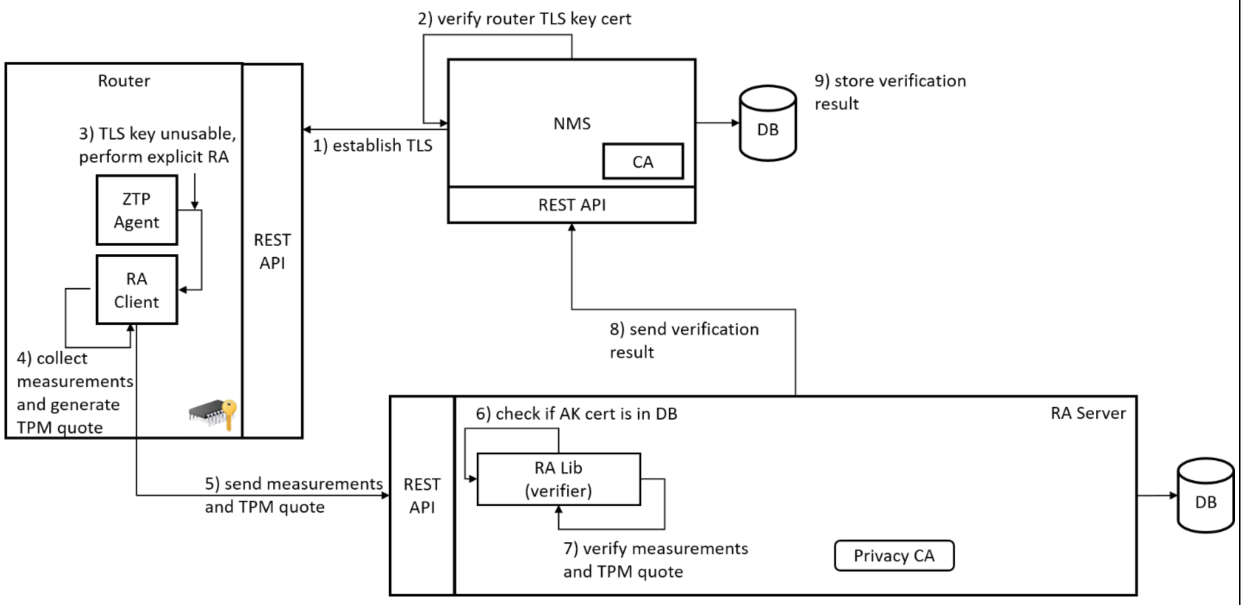

2) verify router TLS key cert
Router
3) TLS key unusable, perform explicit RA
ZTP Agent
RA Client
4) collect measurements and generate TPM quote
REST API
1) establish TLS
NMS
CA
REST API
DB
9) store verification result
8) send verification result
5) send measurements and TPM quote
REST API
6) check if AK cert is in DB
RA Server
RA Lib (verifier)
7) verify measurements and TPM quote
Privacy CA
DB

**Figure 9**: Router runtime verification

**Components:**

- **ZTP Agent**: Agent running on each router, responsible to initiate the enrolment process and to respond to implicit RA requests from the NMS.
- **RA Server**: Remote Attestation Server that exposes a REST API to routers for device enrolment and explicit RA.
- **RA Client**: Remote Attestation Client running on each router to generate TPM keys, quotes and CSRs and to send certificate requests to RA Server.

- **RA Lib (enrolment):** Library running on RA Server to perform enrolment of each router.
- **RA Lib (verifier):** Library running on RA Server to verify CSRs (for implicit RA) and quotes (for explicit RA).
- **NMS**: Network Management System.

**Workflow:**

1. Establish TLS connection
   - The NMS establishes a TLS connection with managed routers.
   - The router replies to the NMS and sends the certificate associated to the generated TLS key.
2. Verify router TLS key cert
   - The NMS queries the DB to verify the router TLS key certificate.
3. TLS key unusable, perform explicit RA
   - If implicit RA fails (TPM key unusable in the router due to configuration change), ZTP Agent asks RA Client to perform explicit RA
4. Collect measurements and generate TPM quote.
   - RA Client collects measurements from the system and asks the TPM to perform the quote operation.
5. Send measurements and TPM quote
   - RA Client sends measurements and TPM quote to RA Server.
6. Check if AK cert is in DB
   - RA Lib (verifier) checks whether the TPM quote has been signed by a TPM AK for which a certificate was released by RA Server.
7. Verify measurements and TPM quote
   - RA Lib (verifier) verifies the measurements and TPM quote sent by RA Client in the router.
8. Send verification result
   - RA Server sends the result of router integrity verification to the NMS so that it can be seen by the Network Administrator.
9. Store verification result
   - The result of the router integrity verification is stored in the NMS DB.

**Issues encountered:** it was not known in the concept phase where the CA used to sign router certificates should be placed. During the software architecture phase, we chose to have different CAs depending on the purpose: the likely existing NMS CA for TLS certificates (since the NMS contacts the routers), and a new Privacy CA (included in the RA Server) for Trusted Computing specific functionality.

**Status:** OnGoing

**Degree of realisation:** Partial

**Comments**: The routing policy functionality and the UI will be delivered with the 2nd release.

**Description**

**User Story Title:** *HWDU.NO.1 – The Network Operator connects the router to the network and is able to verify the device integrity based on a whitelist.*

**User Story Confirmations:**

- *A TPM key is generated on the router for use to establish trusted channels.*
- *The TPM key is validated by the NMS (i.e. it can be used only with software and integrity policy approved by the Network Administrator).*
- *A trusted management channel is established between the NMS and the router (on the router the TPM enforces the validated TPM key policy).*
- *An LED light on the router case indicates that the router has connected to the NMS.*

**TPM Functionalities:**

- Key storage and certification, identity verification, signing, decryption.

**User Story Implementation:**

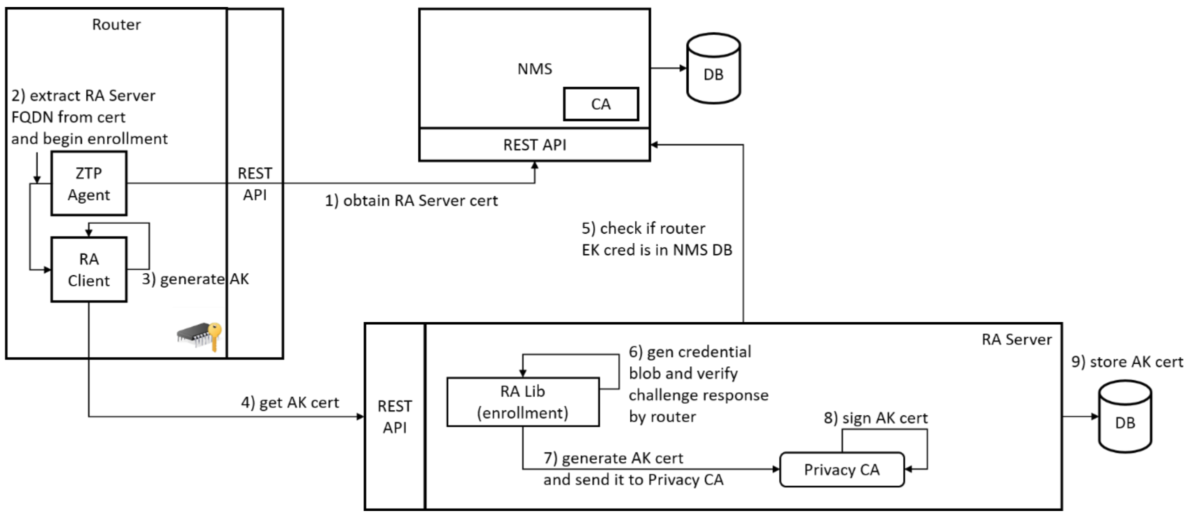

Router
2) extract RA Server FQDN from cert and begin enrollment
ZTP Agent
RA Client
3) generate AK
REST API
1) obtain RA Server cert
NMS
CA
REST API
DB
5) check if router EK cred is in NMS DB
4) get AK cert
REST API
RA Server
RA Lib (enrollment)
6) gen credential blob and verify challenge response by router
8) sign AK cert
7) generate AK cert and send it to Privacy CA
Privacy CA
9) store AK cert
DB

**Figure 10**: Router AK certificate generation

**Workflow (AK certificate):**

1. Obtain RA Server cert
   - ZTP Agent obtains RA Server certificate from the NMS.
2. Extract RA Server FQDN from cert and begin the enrolment
   - ZTP Agent extracts RA Server FQDN from the certificate and passes it to RA Client.
3. Generate AK
   - RA Client generates an AK that will be used to certify the TLS key and sign TPM quotes.
4. Get AK cert
   - RA Client asks RA Server to issue a certificate for the AK it generated.
5. Check if router EK cred is in NMS DB
   - RA Server asks the NMS if the EK credential of the router requesting an AK certificate has been added to the NMS DB by the Network Administrator; this prevents any router from getting an AK certificate.
6. Generate credential blob and verify challenge response by router
   - RA Lib (enrolment) generates a credential blob and asks RA Agent in the router to prove that the router possesses the EK.
7. Generate AK cert and send it to Privacy CA
   - RA Lib (enrolment) generates a certificate for the router AK and asks Privacy CA in RA Server to sign the certificate
8. Sign AK cert
   - Privacy CA signs the AK certificate; RA Server sends it to the router.
9. Store AK cert
   - RA Server stores the signed AK certificate in the DB

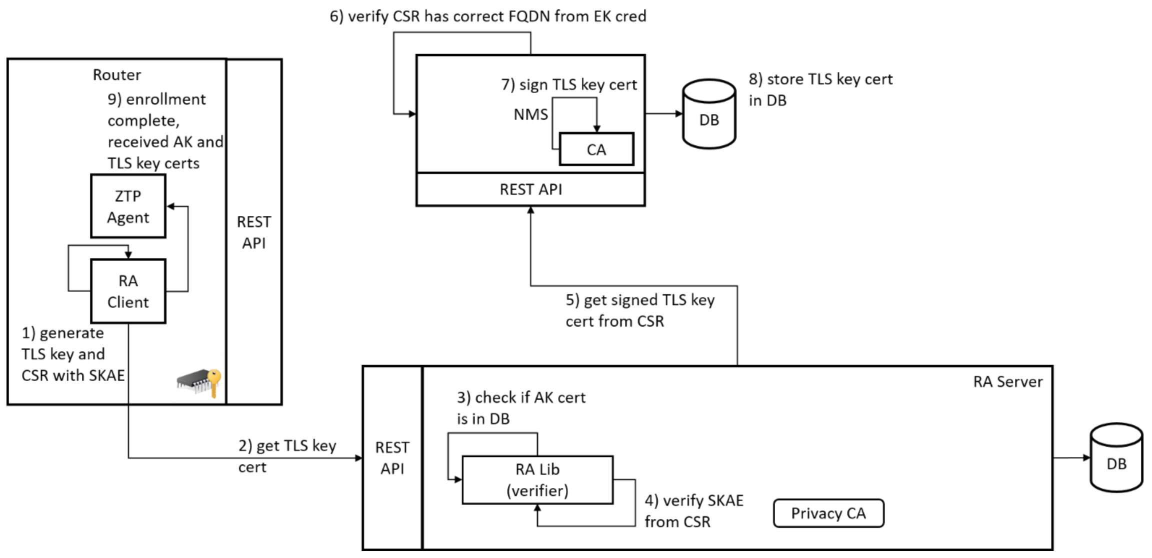

6) verify CSR has correct FQDN from EK cred
Router
9) enrollment complete, received AK and TLS key certs
ZTP Agent
RA Client
1) generate TLS key and CSR with SKAE
REST API
7) sign TLS key cert
NMS
CA
REST API
DB
8) store TLS key cert in DB
5) get signed TLS key cert from CSR
2) get TLS key cert
REST API
RA Server
3) check if AK cert is in DB
RA Lib (verifier)
4) verify SKAE from CSR
Privacy CA
DB

**Figure 11:** TLS key certificate generation

**Workflow (TLS key certificate):**

1. Generate TLS key and CSR with SKAE

   ➢ RA Client generates a TPM key for TLS (the key policy is specified as a parameter of TPM2_Create(); the policy should specify the correct software configuration for which the TPM will allow the key to be used).

   - A malicious router can specify a bad policy (e.g. for an incorrect/insecure software configuration) but cannot convince the RA Lib (verifier) that the policy was good (the generated key and the specified key policy are signed internally by the TPM, so the router has no control over this process).

   ➢ The TPM signature is made with an Attestation Key (AK), which can be reliably associated by the RA Lib (verifier) to a router with the EK credential of that router.

   ➢ RA Client also creates a CSR for the generated key and includes the TPM signature in a certificate extension called Subject Key Attestation Evidence (SKAE) defined by TCG.

2. Get TLS key cert and begin the enrolment

   ➢ RA Client asks RA Server to issue a certificate for the router TLS key

3. Check if AK cert is in DB

   ➢ RA Lib (verifier) first checks if there is a certificate for the AK the router used for signing the TLS key

4. Verify SKAE from CSR

- ➢ RA Lib (verifier) verifies that the CSR is signed with a TPM key, that the TPM key is signed with an AK belonging to the given router and that the signed policy is correct (i.e. the router has a good software configuration).

5. Get signed TLS key cert from CSR
   - ➢ RA Server sends the CSR with the verified SKAE to the NMS, so that the NMS CA can sign it.
6. Verify CSR has correct FQDN from EK cred
   - ➢ The NMS verifies that the FQDN the router included in the CSR is the same that was sent by the Network Administrator during the router registration phase.
7. Sign TLS key cert
   - ➢ The NMS CA signs the TLS key certificate
8. Store TLS key cert in DB
   - ➢ The NMS stores the TLS key certificate of the router in the NMS DB; the TLS key certificate is delivered to RA Client.
9. Enrolment complete, received AK and TLS key cert
   - ➢ RA Client informs ZTP Agent that it successfully received the AK and TLS key certificates.

After the enrolment is complete, ZTP Agent tries to establish a connection with the NMS to verify whether the enrolment was successful.

**Issues encountered:** implementing the enrolment logic was particularly complex due to lack of existing TCG guidance on using the TSS for this purpose. We used the IBM Attestation Client Server from Ken Goldman as reference for implementing this feature in attest-tools.

**Status:** Complete

**Degree of realisation:** Full

### 4.3.2 Unit Test Results

| Test Case DEVMAN1 | |
|---|---|
| **Reference Code** | DEVMAN1 |
| **Components** | RA lib (enrolment) |
| **Description** | This unit test aims at verifying the correctness of the router identification. The unit test checks that the library is able to generate and validate challenges required to identify a router. |
| **Status** | Performed |

| | |
|---|---|
| **Unit Tests Results** | The library correctly generates and encrypts the challenges on RA Server, which, in turn, can be successfully decrypted by the RA Client. |

| **Test Case DEVMAN2** | |
|---|---|
| **Reference Code** | DEVMAN2 |
| **Components** | RA lib (verifier) |
| **Description** | This unit test aims at verifying the correctness of the integrity verification done by RA lib (verifier). The unit test receives attestation data, performs integrity verification and checks the result from reference data. |
| **Status** | Performed |
| **Unit Tests Results** | The library is able to recognize missing or tampered information and fully verify the integrity of the router from the provided data. |

| **Test Case DEVMAN3** | |
|---|---|
| **Reference Code** | DEVMAN3 |
| **Components** | RA Client (prev. called RA Agent) |
| **Description** | This unit test extends the functionality of the FUTURETPM02. This unit test aims at verifying the correctness of the TPM key and certificate generation. This unit tests verifies that a TPM key is created with the given policy and that the generated certificate is associated to that TPM key. |
| **Status** | Performed |
| **Unit Tests Results** | The library successfully extracts the attestation policy from the certificate signing request and verifies the correctness of the software configuration associated to the TLS key. |

| **Test Case DEVMAN4** | |
|---|---|
| **Reference Code** | DEVMAN4 |
| **Components** | RA Client, RA Server |
| **Description** | This unit test aims at verifying the communication between the RA Client and the RA Server. The unit tests verify that both components are able to correctly generate requests and parse responses. |
| **Status** | Performed |
| **Unit Tests Results** | The library successfully performs the expected functionality. |

### 4.3.3 *KPIs Measured*

#### 4.3.3.1 Quantitative Metrics

The table below shows the differences in performance when the demonstrator uses TPM 2.0 and QR-TPM. Entries in bold report the total time necessary to execute a demonstrator functionality. The time was take from the virtual machine. Entries with regular style report the list of TPM commands executed for the demonstrator functionality in the previous row (not exhaustive, for brevity reasons). Only for the router boot phase detailed measurements are not shown, as the TPM commands are sent by the kernel and not by the TSS.

The first and the third column of the table report the TPM command executed by the demonstrator. The third column contains information only if the algorithm used is different. The second and fourth column report the time necessary to execute a TPM command and it has been taken between the beginning and the end of TSS_Execute() function in the TSS.

From the detailed performance measurement we can conclude that the QR-TPM is slower than the unmodified SW-TPM (TPM 2.0). Higher execution times can be explained by the increased size of the data being transmitted between the TSS and the TPM (500 bytes for TPM 2.0 and about 4000 bytes for QR-TPM). Another reason that applies for the PCR commands is that the number of allocated PCR banks in the QR-TPM (7) is higher than the number of PCR banks in TPM 2.0 (4). Also, NVRAM operations are slower due to the different amount of data to fetch (the public key in the EK credential is bigger). Key creation commands cannot be compared because RSA key generation is not deterministic, while Kyber and Dilithium key generation is deterministic. TPM operations that require asymmetric cryptography (e.g. TPM2_Load(), TPM2_ActivateCredential(), TPM2_Certify(), TPM2_Sign()) are seven to ten times slower in the QR-TPM.

From the application perspective, the performance degradation is not as high. The AK creation for example is only about three times slower in the QR TPM. The difference is more significant for the other functionalities of the demonstrator.

| TPM 2.0 Command | TPM 2.0 Timings (TSS) | FutureTPM Command | FutureTPM Timings (TSS) |
|---|---|---|---|
| **Router Boot** | **6.159** | | **6.466** |
| TPM2_ReadClock | N/A (kernel) | | N/A (kernel) |
| TPM2_SelfTest | N/A | | N/A |
| TPM2_GetCapability | N/A | | N/A |
| TPM2_PCR_Extend (SHA1,SHA256,SHA384,SHA512) | N/A | TPM2_PCR_Extend (SHA1,SHA256,SHA384,SHA512,SHA3-256,SHA3-384,SHA3-512) | N/A |
| TPM2_StirRandom | N/A | | N/A |
| TPM2_GetRandom | N/A | | N/A |
| TPM2_HierarchyChangeAuth | N/A | | N/A |
| TPM2_PCR_Read (SHA1) | N/A | TPM2_PCR_Read (SHA1) | N/A |
| TPM2_Load (sealed blob under rsa 2048) | N/A | TPM2_Load (sealed blob under kyber security=3) | N/A |

| TPM 2.0 Command | TPM 2.0 Timings (TSS) | FutureTPM Command | FutureTPM Timings (TSS) |
|---|---|---|---|
| TPM2_StartAuthSession | N/A | | N/A |
| TPM2_PolicyPCR (SHA1) | N/A | TPM2_PolicyPCR (SHA256) | N/A |
| TPM2_Unseal | N/A | | N/A |
| TPM2_FlushContext | N/A | | N/A |
| **AK Creation** | **0.300** | | **0.834** |
| TPM2_NV_ReadPublic (EK credential length) | 0.000921 | | 0.01377 |
| TPM2_GetCapability | 0.000590 | | 0.013580 |
| TPM2_NV_Read (EK credential) | 0.004778 | | 0.01802 |
| TPM2_Create (AK, rsa 2048) | 0.004779 | TPM2_Create (AK, dilithium mode=2) | 0.031657 |
| TPM2_CreatePrimary (EK, rsa 2048) | 0.011244 | TPM2_CreatePrimary (EK, kyber security=3) | 0.020212 |
| TPM2_Load (AK, rsa 2048) | 0.002805 | TPM2_Load (AK, dilithium mode=2) | 0.030117 |
| TPM2_StartAuthSession | 0.000799 | | 0.013721 |
| TPM2_PolicySecret | 0.000592 | | 0.013733 |
| TPM2_ActivateCredential | 0.002394 | | 0.018827 |
| TPM2_FlushContext | 0.000471 | | 0.013273 |
| **TLS Key Creation** | **0.194** | | **0.655** |
| TPM2_PCR_Read (SHA1) | 0.000789 | TPM2_PCR_Read (SHA256) | 0.013633 |
| TPM2_Create (TLS, rsa 2048) | 0.004865 | TPM2_Create (TLS, dilithium mode=2) | 0.032031 |
| TPM2_Load (TLS, rsa 2048) | 0.002942 | TPM2_Load (TLS, dilithium mode=2) | 0.030333 |
| TPM2_Load (AK, rsa 2048) | 0.002779 | TPM2_Load (AK, dilithium mode=2) | 0.030129 |
| TPM2_Certify | 0.002279 | | 0.023121 |
| TPM2_FlushContext | 0.000492 | | 0.013544 |
| TPM2_ReadPublic (SRK, rsa 2048) | 0.002016 | TPM2_ReadPublic (SRK, kyber security=3) | 0.018828 |
| TPM2_StartAuthSession (SRK used as salt key) | 0.001963 | | 0.018708 |
| TPM2_PolicyPCR (SHA1) | 0.000601 | TPM2_PolicyPCR (SHA256) | 0.013880 |
| TPM2_RSA_Decrypt | 0.003242 | TPM2_Sign | 0.022728 |
| **TLS Connection** | **0.073** | | **0.331** |

| TPM 2.0 Command | TPM 2.0 Timings (TSS) | FutureTPM Command | FutureTPM Timings (TSS) |
|---|---|---|---|
| TPM2_ReadPublic (SRK, rsa 2048) | 0.002401 | TPM2_ReadPublic (SRK, kyber security=3) | 0.018779 |
| TPM2_StartAuthSession(SRK used as salt key) | 0.002068 | | 0.018585 |
| TPM2_Load (TLS, rsa 2048) | 0.003677 | TPM2_Load (TLS, dilithium mode=2) | 0.030866 |
| TPM2_PolicyPCR (SHA1) | 0.000623 | TPM2_PolicyPCR (SHA256) | 0.013606 |
| TPM2_RSA_Decrypt | 0.003241 | TPM2_Sign | 0.022806 |
| TPM2_FlushContext | 0.000492 | | 0.013335 |
| **Quote** | **0.066** | | **0.381** |
| TPM2_Load (AK, rsa 2048) | 0.003126 | TPM2_Load (AK, dilithium mode=2) | 0.029669 |
| TPM2_Quote | 0.002785 | | 0.022542 |
| TPM2_FlushContext | 0.000531 | | 0.013034 |

Table 10: Demonstrator #1 – Comparison of Timings between TPM2.0 (SW) and FutureTPM (SW)

Regarding KPIs 1 and 2, to the best of our knowledge, the Simple RA introduced in the demonstrator is applicable to all types of routers and/or compute devices running Linux. In the case of highly customized Linux versions, it might be possible to require minor adaptations, while keeping the concept unchanged.

For KPIs 3 and 5 there was no known industry solution at the time of starting the project to achieve the target values. Therefore, we developed the new CIV architecture that allowed us to fill the gap.

| Id | Metric | Target Value | Acceptance criteria | (M)andatory / (G)ood to Have / (O)ptional | Measured by M24 | Comments |
|---|---|---|---|---|---|---|
| 1 | Amount of routers whose integrity is monitored by NMS | 100% | 100% | M | With TPM2.0: 100%<br>With FutureTPM: 100% | |
| 2 | Amount of routers hiding their integrity status | 0% | 0% | M | With TPM2.0: 0%<br>With FutureTPM: 0% | No enrolled router can hide its status. However, due to limitations of dynamic routing protocols, a router whose identity is not known to the NMS might still operate in the network. |

| Id | Metric | Target Value | Acceptance criteria | (M)andatory / (G)ood to Have / (O)ptional | Measured by M24 | Comments |
|---|---|---|---|---|---|---|
| 3 | Amount of detected integrity attacks on routers | 80% (with integrity models) | 60% (standard IMA) | M | With TPM2.0: 80%<br>With FutureTPM: 80% | Besides attacks detected by standard IMA, we additionally cover attacks on:<br>- mutable files;<br>- non-regular files (e.g. IPC, socket etc.).<br>Not covered:<br>- control flow attacks;<br>- file path protection. |
| 4 | Amount of traffic diverted to alternative paths when a router is compromised | 75% | 55% | G | With TPM2.0: N/A<br>With FutureTPM: N/A | Planned for 2nd release. |
| 5 | Amount of files whose integrity can be verified | 100% (with integrity models) | 99% (standard IMA) | G<br>M | With TPM2.0: 100%<br>With FutureTPM: 100% | All files can be verified. |

Table 11: Demonstrator #3 – Quantitative Metrics by M24

#### 4.3.3.2 Qualitative Metrics

TPM-based secure channels can be implemented by following existing specifications and several examples exist in the industry. However, it has not practical so far to bind the TPM keys to the complete software configuration, due to the traditional Measured Boot concept which is not suitable for complex operating system scenarios, where several processes are executed in parallel. Introducing CIV enables to overcome this limitation and achieve the below qualitative KPIs.

| Id | Metric | Target Value | (M)andatory / (G)ood to Have / (O)ptional | Measured by M24 | Comments |
|---|---|---|---|---|---|
| 1 | Traffic routing based on router trust state | Supported | M | With TPM2.0: N/A<br>With FutureTPM: N/A | Planned for 2nd release. |
| 2 | Trusted channels between NMS and each router in the network | Supported | M | With TPM2.0: Supported<br>With FutureTPM: Supported | |

| Id | Metric | Target Value | (M)andatory / (G)ood to Have / (O)ptional | Measured by M24 | Comments |
|---|---|---|---|---|---|
| 3 | Device authentication key for trusted channel protected by TPM | Supported | M | With TPM2.0: Supported<br>With FutureTPM: Supported | |
| 4 | Integrity protection of router configuration data using a TPM key | Supported | M | With TPM2.0: Supported<br>With FutureTPM: Supported | |

Table 12: Demonstrator #3 – Qualitative Metrics by M24

### 4.3.4 *Plan for the next Period*

During the next period, the rest of the user stories as defined in deliverable D6.1 will be executed. Once the virtual QR TPM becomes available, the demonstrator will switch to it and re-run the performance measurements for its respective set of QR algorithms.

## 4.4 Conclusions

The Device Management demonstrator implementation is proceeding according to plan. There have been a number of design and implementation issues that have been overcome, new features have been added (S-ZTP) and new technologies have been introduced (CIV architecture) to solve existing industry challenges.

The performance measurements are reasonable and do not dramatically impact the router system, enabling to achieve the set KPIs.

In the first period, the demonstrator has focused on the enrolment and remote attestation parts. In the subsequent period, the demonstrator will add support for the regular device management logic according to the remaining user stories.

# Chapter 5 Summary and Conclusion

The current deliverable aims to cover main activities of the **evaluation, validation and refinement phase** related to setting up, executing and evaluating the three envisioned use cases; namely the "*Secure Mobile Wallet and Payments*", "*Personal Activity and Health Kit Data Tracking*" and "*Device Management*" reference scenarios. It reflected on the pilot implementation and integration of the FutureTPM framework in three different use cases (called demonstrators), to test **the assumptions of the project, and the feasibility, the applicability and the overall acceptance of post-quantum TPM in specific business cases**, not only in terms of **security**, but also in terms of **performance, availability and of other business critical indicators**.

The key outputs have been the: (i) **definition of a set of tests** for the list of core, integral components plus the technologies to be leveraged towards carrying on with such tests, paying special attention to the integration plan, and (ii) analysis of the first set of results related to the **performance evaluation of the SW-based QR TPM and the implemented Trusted Software Stack (TSS)** with timings of the sequences of TPM commands, for achieving the **security, privacy, and trust** properties of interest per reference scenario

Towards this direction, the work performed for each one of the aforementioned demonstrators till M24 of the project was presented here.

- ➢ In the context of the <u>Secure Mobile Wallet and Payment</u> use case, sealing and unsealing operations have been successfully implemented using the software implementation of the FutureTPM. We contacted a thorough comparison on the performance of both HW TPM2.0 and SW-based FutureTPM, in order to provide deep insights on their operational behaviour in the context of the demonstrator. The results advocate that the performance of the software implementation of the FutureTPM meets the performance KPIs. The time discrepancies among the contacted measurements are justified by the nature of the TPMs (Software/Hardware) and the interception placement (TSS/Application) for capturing the timings. Further experiments will be conducted for the 2nd release of the demonstrator, in order to evaluate the performance for the rest of the user stories under the distributed nature of the overall architecture of the dedicated TPM server and the resources needed to work with QR algorithms and schemes. Overall, the performance of FutureTPM meets the goals of the demonstrator.
- In the context of the <u>Personal Activity and Health Kit Data Tracking</u> use case, the LDAA has been successfully implemented using the software implementation of the FutureTPM, however, there have been some performance issues which are inherited by the nature and the overall architecture of the TPM, the resources needed to work with QR algorithms and schemes and also the business logic of the current demonstrator which worked with packaging, sending and unbundling data close to real-time. In general, however, the results (except the signature process) are acceptable, even if not very close to the set targets. To mitigate the delay witnessed during the signature process there is an idea to alter a bit the business logic of the demonstrator, scheduling the signature and the sending of the payload to happen at off-peak times, using pre-programmed daemons that will run in the background on the S5 PersonalTracker instance.
- In the context of the <u>Device Management</u> use case, the performance measurements are reasonable and do not dramatically impact the router system, enabling to achieve the set KPIs

The final version of the overall documentation and the validation of the demonstrators will be part of deliverable D6.5 and D6.6 of the project, to be delivered in M33 and M36 respectively, as this is the point that will mark the successful implementation of all demonstrator activities and the full evaluation of the FutureTPM platform as a whole.

# Chapter 6 List of Abbreviations

| Abbreviation | Translation |
|---|---|
| **AK** | Attestation Key |
| **CFA** | Control Flow Attestation |
| **CFG** | Control Flow Graph |
| **CFP** | Control Flow Path |
| **CISQ** | Consortium for IT Software Quality |
| **DH** | Diffie-Hellman |
| **eBPF** | enhanced Berkeley Packet Filter |
| **FIDO** | Fast ID Online |
| **KPI** | Key Performance Indicators |
| **KVM** | Kernel-based Virtual Machine |
| **MFA** | Multifactor Authentication |
| **NFC** | Near Field Communication |
| **PCR** | Platform Configuration Register |
| **PDP** | Policy Decision Point |
| **PE** | Policy Enforcement |
| **PEP** | Policy Enforcement Point |
| **QEMU** | Quick Emulator |
| **RA** | Risk Assessment |
| **SKAE** | Subject Key Attestation Evidence |
| **WP** | Work Package |

# Chapter 7 Bibliography

[1] T. F. Consortium, “D6.1 – Technical Integration Points and Testing Plan,” 2019.

[2] T. F. Consortium, “D2.1 – Second Report on New QR Cryptographic Primitives,” 2019.

[3] T. F. Consortium, “D4.2 – FutureTPM Risk Assessment Framework – First Release,” 2019.

[4] Trusted Computing Group (TCG), *TCG Glossary (Version 1.1, Revision 1.00),* 2017.

[5] Trusted Computing Group (TCG), *Trusted Platform Module Library - Part 1: Architecture (Family 2.0, Revision 01.38),* 2016.

[6] Trusted Computing Group (TCG), *TCG TSS 2.0 TPM Command Transmission Interface (TCTI) API Specification,* 2018.

[7] C. Yue, B. Boehm and L. Sheppard, “Value driven security threat modeling based on attack path analysis.,” in *40th Annual Hawaii International Conference on System Sciences (HICSS 2007)*, 2007.

[8] V. Saini, Q. Duan, and V. Paruchuri, “Threat modeling using attack trees,” *Journal of Computing in Small Colleges,* vol. 23, no. 4, pp. 124-131, 2018.

[9] ETSI TS 102 165-1, “Methods and protocols; Part 1: Method and proforma for Threat, Risk, Vulnerability Analysis,” in *Telecommunications and Internet converged Services and Protocols for Advanced Networking (TISPAN)*, 2014.

[10] Trusted Computing Group (TCG), “TCG TSS 2.0 TAB and Resource Manager Specification,” 2018.

[11] TPM2-abrmd authors, “TPM2 Access Broker & Resource Manager,” [Online]. Available: https://github.com/tpm2-software/tpm2-abrmd.

[12] T. F. Consortium, “D4.1 – Threat Modelling & Risk Assessment Methodology,” 2019.

[13] T. F. Consortium, “D1.1 - FutureTPM Use Cases and System Requirements,” 2018.

[14] T. F. Consortium, “D5.1 – First Version of Implementation,” 2019.