D6.5

Final Demonstrators Implementation Report

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:	October 2020 – M34
Actual submission date:	February 3rd, 2021

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

Abstract:	Deliverable D6.5 provides the final reporting on the second round of experiments of the FutureTPM framework, in the context of the three envisioned use cases. It summarizes the operation of the core FutureTPM artefacts (QR algorithms implemented in the HW-, SW, and VM-based TPM variants, Configuration Integrity Verification, and Risk Assessment) in the demonstrators coupled with a comprehensive analysis of the integration and evaluation of the second release of the overall framework. This analysis is also enriched with challenges that were taken into consideration when migrating to such QR Root-of-Trusts. The latter also acts as a preliminary documentation of the general adoption guidelines and lessons learnt, throughout the project, which will be put forth in Deliverable D6.6.
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)

Liqun Chen (SURREY)

Sofianna Menesidou, Dimitris Papamartzivanos (UBITECH)

Rogério Paludo, Luís Fiolhais, Leonel Sousa (INESC-ID)

Christine Wright (RHUL), Daniele Sgandurra (RHUL), Harry Lockyer (RHUL)

Roberto Sassu, Silviu Vlasceanu, Rahul Dulta (HWDU)

Fanis Sklinos, George Evangelogeorgos (INDEV)

George Bikas (S5)

Executive Summary

Deliverable D6.5 documents and wraps up the main activities of the **evaluation, validation and refinement phase** related to setting up, executing and evaluating the three envisioned use cases of the FutureTPM project, as identified in the previous WP6 deliverables and in the project's DoA; namely, *Secure Mobile Wallet and Payment*, *Activity Tracking* and *Device Management*. As the goal of FutureTPM is to show-case the use of TPMs, as decentralized roots-of-trust, towards providing enhanced **security, privacy and trust** while transitioning in the post-quantum era, the core property of interest is the overhead added when executing the various QR cryptographic primitives that have been analysed in the context of WP2; covering all of the needed security functionalities ranging from **digital signatures, and symmetric crypto to asymmetric crypto algorithms, key management and the integration of advanced privacy-preserving L-DAA mechanisms**. Each one of these primitives 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 all of the defined requirements without, however, affecting the applicability of this new generation of TPM chips (in continuation of the current TPM 2.0) in a variety of application domains with security and privacy considerations.

The deliverable is the <u>second and final step</u>, compiled by the consortium, towards **testing the assumptions of the project, and the feasibility, the applicability and the overall acceptance of post-quantum TPMs in specific business cases**, not only in terms of **security**, but also in terms of **performance, availability and of other business critical indicators**.

In this context, the deliverable at hand provides a detailed documentation of the **second-cycle demonstrator results till M35 of the project** (following the results extracted during the first cycle that lasted till M24) and provides the findings in guidance with the processes and the indicators set in the project's evaluation plan (D6.1 "Technical Integration Points and Testing Plan"),

Building on top of the results and findings of the previous evaluation cycle, D6.5 provides a high-level description of each **reference scenario**, evaluated in this second testing cycle, accompanied by the different **user stories** of interest, the **configuration parameters** and the **implementation, integration status** of each demonstrator and of course provides a detailed analysis of the extracted results. The latter are, in many cases (depending on the use case, the scenario and the metric examined), linked to the results of the first evaluation cycle, as the approach chosen by the demonstrators' work package was not to simply test the FutureTPM framework, but to also provide continuous feedback. This has been achieved by setting up the demonstrators testbeds in such a way so that the different algorithms employed in FutureTPM, as well as the technical backbone of the envisioned FutureTPM platform, could be evaluated and improved in iterative rounds.

In the latter chapters of the deliverable, we also provide an in-depth analysis of the underpinnings of the performed experiments with the extracted results and describe all issues that need to be solved for further improving the performance of the overall FutureTPM framework. This summary of all key performance indicators from the QR algorithms developed and tested, as well as the new remote attestation enablers, will set the scene for the critical appraisal of all the project's artefacts towards securing both extremes of a network, namely the edge and the cloud.

Contents

Chapter 1 Introduction 1
1.1 Evaluation, Testing and Validation Methodology 2
1.2 Harmonized Test Guidelines 3
1.3 Document Structure 4
Chapter 2 Demonstrator #1 – Secure Mobile Wallet and Payment 5
2.1 Demonstrator Overview and Final Architecture 5
2.1.1 Overview of the FIDO U2F Registration and Authentication Phases 7
2.1.2 Overview of Remote Attestation Schemes: Attestation by-proof and Attestation-by-Quote 9
2.1.3 Strong Authentication by integrating the use of TPMs and DAA in the FIDO U2F Protocol 10
2.1.4 Demonstrator Needs and Challenges 19
2.2 Implementation Path Report for the 2nd Experimentation Period 20
2.2.1 User Stories Realisation 20
2.2.2 Unit Test Results 25
2.2.3 KPIs Measured 27
Chapter 3 Demonstrator #2 – *Activity Tracking* 34
3.1 Demonstrator Overview and Final Architecture 34
3.1.1 Demonstrator Needs and Challenges 38
3.2 Implementation Path Report for the 2nd Experimentation Period 38
3.2.1 Activity Tracker Demonstrator – Experimentation with LDAA-v1 and LDAA-v2 39
3.2.2 LDAA-v2 Experimentation as a Standalone Application 52
3.3 Use Case Evaluation 54
Chapter 4 Demonstrator #3 – *Device Management* 55
4.1 Demonstrator Overview 55
4.1.1 Demonstrator Needs and Challenges 55
4.1.2 Demonstrator Architecture 55
4.2 Emulated System Description 57
4.3 Implementation Path Report for the 2nd Experimentation Period 58
4.3.1 User Stories Realisation 59
4.3.2 QR Virtual Trusted Platform Module Experimentation 75
4.4 Use Case Evaluation 78
Chapter 5 Conclusions 79
Chapter 6 List of Abbreviations 80
Chapter 7 Bibliography 82

List of Figures

Figure 1: Secure Mobile Wallet and Payment High Level Approach...5

Figure 2: Mobile App environment. (a) User login, (b) FIDO Registration/Authentication and execute financial transaction functionalities, (c) Interaction with YubiKey, (d) Successful Configuration Integrity verification for transaction execution, (e) Failed Configuration Integrity verification for transaction execution. 6

Figure 3: U2F Registration 8

Figure 4: Mo: U2F Authentication...8

Figure 5: Workflow of system Configuration Integrity Verification: Attestation by Proof (Left) and Attestation by Quote (Right) 10

Figure 6: System Model 12

Figure 7: State diagram showing the states for generating a Service Attestation key. Blue takes place on the host and green in the TPM...13

Figure 8: Diagram showing the states for generating an authentication key. Blue is provided by the host, green is states with relation to the authentication key and red is in regards to the service attestation (parent) key 13

Figure 9: State diagram showing the states for generating an ECC Key for Direct Anonymous Attestation. Blue actions occur on the host, and orange in the TPM. 14

Figure 10: State diagram showing the states necessary for signing an Authentication Key for Registration with a Relying Party. 15

Figure 11: State diagram showing the necessary states for signing a host-provided dataset for authentication to a Relying party. 15

Figure 12: The adapted registration protocol. New items are highlighted in yellow, line-through items have been removed. 17

Figure 13: The adapted authentication protocol. New items are highlighted in yellow, line-through items have been removed 19

Figure 14: INDEV.AU.3 workflow. Attestation by Quote for the integrity verification of transactions DB 22

Figure 15: INDEV.AU.4 workflow. Attestation by Proof for the verification of the operational correctness of the mobile application 24

Figure 16: Boxplot of CC_Create of BLISS key pair 28

Figure 17: Demonstrator #2 – Main Actors and Entities 35

Figure 18: Demonstrator #2 – Revised Architecture showing entities concerned in the demonstrator for the use cases till M36 36

Figure 19: Demonstrator #2 – Screenshot of the S5Personal Tracker Interface 37

Figure 20: Demonstrator #2 – Screenshot of Sharing Selection and Execution at the S5Peronal Tracker Side 37

Figure 21: S5.IU.1 sequence diagram...40

Figure 22: Comparison Graphs for different timings at application level 43

Figure 23: Comparison Graphs for different timings for 5kB and 25MB files.................................45

Figure 24: Comparison Graph 2 for different timings for 5kb and 25Mb files 45

Figure 25: Comparison Graphs for different timings for 5kB and 25MB files.................................46

Figure 26: Comparison Graph for different timings for 5kb LDAA-v1 vs LDAA-v2 48
Figure 27: Comparison Graphs for different timings for 5kb and 25kb files for the Initialise and Join phase 49
Figure 28: Total Execution Time for the Join and Sign phase at the TSS level 50
Figure 29: Demonstrator #3 – overall architecture and main entities 56
Figure 30: Demonstrator #3 – Software stack diagram 58
Figure 31: Router runtime verification 61
Figure 32: Router AK certificate generation 63
Figure 33: Router TLS cert generation 65
Figure 34: Network graph with all routers healthy 70
Figure 35: traceroute output with all routers healthy 70
Figure 36: Network graph with one router compromised 71
Figure 37: Network capture from the time the NMS detected the attack on Router 2 71
Figure 38: Network capture from the time packets are diverted to Router 3 72
Figure 39: Wireshark statistics 72
Figure 40: ping output from the client 72
Figure 41: traceroute output after the attack is detected on Router 2 73
Figure 42: Virtual Machine Manager output before and after the attack 73
Figure 43: V-TPM Architecture 76
Figure 44: Timing Comparison of SW-TPM and V-TPM for Dilithium and Kyber 77
Figure 45: Timing Comparison of all V-TPM Commands for Rainbow, SPHINCS+ and Dilithium .. 77

List of Tables

Table 1: Reference Scenarios Overview during Second Experimentation Cycle 1

Table 2: CC_Create command statistics over 100 experimental results for the BLISS key creation 29

Table 3: Demonstrator #1 – Comparison of Timings among the App, TSS and the Network perspectives using HW QR-TPM (on FPGA board) for the realisation of user story INDEV.AU.3.. 29

Table 4: Demonstrator #1 – Comparison of Timings among the App, TSS and the Network perspectives using HW QR-TPM (on FPGA board) for the realisation of user story INDEV.AU.4. with successful signature verification. 30

Table 5: Demonstrator #1 – Comparison of Timings among the App, TSS and the Network perspectives using HW QR-TPM (on FPGA board) for the realisation of user story INDEV.AU.4. with failed signature verification. 31

Table 6: Demonstrator #1 – Quantitative Metrics by M24 and M35 31

Table 7: Demonstrator #1 – Qualitative Metrics by M24 and M35 33

Table 8: Comparison of DAA/LDAA Timings at Application level........ 42

Table 9: Demonstrator #2 –Timings at TSS Level using the FutureTPM QR Implementation (SW) of LDAA-v1 44

Table 10: Demonstrator #2 – Timings at TSS Level using the FutureTPM QR Implementation (SW) of LDAA-v2 46

Table 11: Demonstrator #2 – Timings at TSS Level using the FutureTPM QR Implementation (SW) of LDAA-v1 versus LDAA-v2........ 47

Table 12: Demonstrator #2 – Quantitative Metrics by M36........ 50

Table 13: Demonstrator #2 – Qualitative Metrics by M36 51

Table 14: Execution timings for LDAA-v2 in comparison with comparison with LDAA-v1. 52

Table 15: Public Key, Private Key and Sigature Sizes for LDAA-v2 in comparison with comparison with LDAA-v1........ 53

Table 16: Maximum memory requirements for LDAA-v2 in comparison with comparison with LDAA-v1........ 53

Table 17: Demonstrator #3 – Comparison of Timings between TPM2.0 (SW) and FutureTPM (SW) 68

Table 18: Demonstrator #3 – Quantitative Metrics by M36........ 74

Table 19: Demonstrator #3 – Qualitative Metrics by M36 75

Table 20: Results of V-TPM Tests for Dilithium and Kyber........ 76

Chapter 1 Introduction

Deliverable D6.5 comes as the final deliverable, of the implementation and evaluation work-package, covering the demonstrators' experimentation lifespan and documents the main activities conducted towards the **evaluation, validation and refinement phase** related to setting up, executing and testing the three envisioned use cases of the FutureTPM project.

As the goal of FutureTPM is to enhance security at all levels of future systems, embedding trust at both extremes of a network, namely and the edge and the cloud is crucial. Indeed, in the era when "*service is everything and everything is a service*", there is an emerging trend for intelligent edge computing – comprising of heterogeneous devices with various security and privacy concerns – to work in tandem so as to provide flexible design choice that best meet business and operational goals. However, this evolution brings a number of new challenges with **security, resilience, trust and operational assurance**, in the ecosystem of quantum computing, being some of the major concerns that FutureTPM tries to resolve.

Compounding this issue, the approach followed when defining the experimentation scenarios aimed at demonstrating each one of these properties in separate reference cases, which also allowed the consortium to progress with a more detailed evaluation and validation of the different TPM operations needed for achieving a subset of these requirements; *always within the business context provided by each demonstrator*. As such, the purpose and results of this deliverable do not cover an exhaustive set of all available TPM operations, but demonstrate the core ones needed for achieving the main vision of FutureTPM towards enhanced operational assurance of "Systems-of-Systems", linking them to real business world needs and evaluating their usability, feasibility and end-user acceptance.

These use cases, as described in deliverable D6.1 of the project are the following:

- "*Secure Mobile Wallet and Payments*" with core properties of interest being on **confidentiality and integrity of financial transactions, user authentication and secure key management**;
- "*Personal Activity and Health Kit Data Tracking*" where the primary interest is on ensuring the **privacy of the participating users** by enabling them to control the level of **anonymity when sharing their data**;
- "Device Management" focusing on services embedded in all of today's business ecosystems, towards the establishment of trust between network devices, including **secure device identification, software integrity and zero-touch configuration integrity verification**.

Table 1: Reference Scenarios Overview during Second Experimentation Cycle

Reference Scenario	TPM Type	Security Property	Functionalities
Secure Mobile Wallet and Payments	Hardware TPM	Security	Sealing, Unsealing, Key Generation, Attestation by Quote, Attestation by Proof
Personal Activity and Health Kit Data Tracking	Software TPM	Privacy	DAA Join, DAA Sign, DAA Verify, DAA Attestation
Device Management	Software TPM in Virtualized Environments	Trust	Remote Attestation, Device Management with Secure Key Identifiers

As the FutureTPM project adopts a two-cycle development, integration, demonstration and evaluation approach, D6.5 provides a detailed documentation of the **second-cycle demonstrator results till M35**. Building on top of the results and the evaluation of the first-cycle which finished in M24 of the project (see deliverable D6.3 [22]), this final implementation report focus 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.

We have to highlight that in some demonstrators, these timings concern new user stories which were realised, according to the consortium's evaluation plan, in the second evaluation cycle towards completing the overall demonstrator scenario, while in others these timings are measured compared to timings acquired during the first cycle aiming to substantiate the performance gains and improvements that have been implemented following the suggestions springing out of the evaluation results as presented in D6.3.

In this context, an updated description per demonstrator is provided, and reference to the executed scenarios and user stories is done in each demonstrator, identifying also their requirements, as well as the conditions and the implementation and integration status, followed by the analysis of the extracted results.

Furthermore, an additional interesting direction is being put forth based on the comments received by the project's Advisory Board: *to investigate the integration of standardized user authentication schemes, as have been proposed by the FIDO Alliance, in the context of the e-Payment use case for better protecting the security and privacy of all involved stakeholders and especially the user conducting their financial transactions*. In this context, the consortium leveraged the **FIDO Universal Two Factor (U2F)** that is based on the use of multiple factors for authenticating user based on "proof-of-ownership" of some secret. It extended this U2F protocol by designing **new models based on the integration of Direct Anonymous Attestation (DAA)** so as to be able to provide enhanced **user privacy, operational assurance and functional safety**; properties that were not achieved by the existing protocol since, for instance, the focus is not on user privacy and the vendor is left with the responsibility to provide anonymous attestation.

1.1 Evaluation, Testing and Validation Methodology

The evaluation approach has been specified in D6.1 [1] and consists of user stories and unit tests, concluding with specific quantitative and qualitative KPIs for measuring the impact of the FutureTPM framework in the business context of each demonstrator. As such, **the evaluation is not only concerned about the timings of the KPIs but is more concerned to evaluate the application of the FutureTPM framework on the existing demonstrators and to measure the rationale and the business value of introducing QR-TPM methods to those**.

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. Within this deliverable, the demonstrators worked with the envisioned release of the QR-TPM modules, (HW or SW – see Table 1) alongside the other core FutureTPM framework components (providing the Risk Assessment, Security Policy Enforcement and Configuration Integrity Verification mechanisms), which allowed to test and evaluate both the FutureTPM platform as a whole but also the different algorithms in the context of each use case.

Therefore, the HW-based QR TPM was integrated in the "*Secure Mobile Wallet and Payments*" scenario while the other two use cases leveraged the SW-based QR TPM. In the context of the "*Device Management*" application domain, the consortium also tested this software-based TPM variant in virtualized environments to better emulate the use of a VM-based TPM. This action was based on the initial set of results acquired in the first evaluation cycle, which have been used as a reference point for the upcoming experiments. It is reminded that during that phase, all demonstrators initially worked to integrate the TPM2.0 characteristics to their existing infrastructures, followed by the conduction of QR-TPM experiments, delivering the first evaluation report in the context D6.3.

In the following chapters, we provide an in-depth analysis of the underpinnings of the performed experiments with the extracted results and describe all issues that were encountered and dealt with which allowed the consortium to improve the performance of the overall framework. For the sake of completeness, we provide in the following lines of this sub-section some information about the testing and the different layers of testing, which have been part also of deliverable D6.3

When it comes to testing, the focus was on examining that the QR-TPM modules work in accordance with their specifications [13] and have no undesirable effects when employed in ways outside of their design parameters. Since each demonstrator was executed in different hosts with various configurations, in what follows we attempt to shed some light on the harmonization process used to assure comparable results between each demonstrator instance.

1.2 Harmonized Test Guidelines

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 through its Trusted Software Stack (TSS). As such, before describing the testing methodology, we need to understand what differentiates the TPM architecture from other commodity controller. 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. On the other hand, is achieved by applying multiple levels of small but fast (-mirrored) memories between the slow external memory and the processor, hiding the large latency of the external memory. Since the HW-based QR TPM is implemented in an ASIC [23]. 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.

In such processors, 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 time instances, results in completely different performance 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. When measuring application timings, using the FutureTPM stack, performance values are measured using bash's time command. While the TPM component is running and started up---using the startup command---, each command is measured with the TPM always been in the same state to make sure that no overhead is measured from internal TPM operations, like platform configuration register reset, that may take place after rebooting the TPM. 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 underlying 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.

1.3 Document Structure

The structure of this document is formed in such a way so that each demonstrator is described in a holistic manner under a dedicated section.

As such, the document starts with the first introductory section (Chapter 1) where the evaluation and the testing methodology are highlighted as a reminder to the reader of how these have been used during the duration of the demonstrators.

The second section (Chapter 2) is dedicated to the experimentation of the "Secure Mobile Wallet and Payments" scenario that focuses on the Security and Integrity Verification aspects that TPMs can offer, dealing with the application of FutureTPM on a business case in the financial services market and business ecosystems.

The third section (Chapter 3) deals with the "Activity Tracking" Demonstrator, which focuses on how User Privacy and Data Anonymization can be achieved in the domain of healthcare and personal activity data management with the introduction of direct anonymous attestation methods that are part of the FutureTPM framework

Chapter 4, is dedicated to the "Device Management" demonstrator, showcasing how FutureTPM can be used to increase Trust between devices and systems focusing on a scenario with a centrally managed distributed telecommunications infrastructure composed of various devices.

Finally, Chapter 5 of the document at hand concludes the deliverable.

Chapter 2 Demonstrator #1 – Secure Mobile Wallet and Payment

2.1 Demonstrator Overview and Final Architecture

This demonstrator focuses on a wide range of technological features that foster innovation in the financial landscape. More specifically, the e-payment use case demonstrates 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 the integrity of mission critical data. In addition, as aforementioned, through the realisation of the user stories, the e-payment scenario integrates the competitive characteristics of the FIDO U2F Protocol for user Registration and Authentication, while it incorporates the developments of the project regarding the runtime tracing techniques and the designed remote attestation schemes, namely Attestation-by-Proof and Attestation-by-Quote. The aforementioned functionalities work in tandem with the QR-TPM for future proofing the mobile e-payment application to resist quantum attacks.

In D6.3, we provided the demonstration of: a) the sealing functionality for the Bearer and Financial Tokens, and b) the unsealing functionality for the tokens. The focal point of this deliverable is the demonstration of: c) the key generation for **encrypting financial transaction history logs** and **attesting the integrity** of the database (*INDEV.AU.3*), d) the v**erification of the operational correctness of the mobile device** using the attestation-by-proof schema (*INDEV.AU.4*) and finally, e) a thorough analysis on **how a TPM can be used in synergy with FIDO U2F Protocol** (*INDEV.AU.5*).

At the same time, all these functionalities – aiming at enhanced operational correctness and functional safety - are supported by the multi-level detailed tracing techniques developed in the context of the risk assessment framework for the efficient monitoring of the configuration and execution behavioural properties to be attested [24]. Note that in the previous experimental phase, such tracing techniques were deployed at the kernel level (kernel interceptor) in order to produce the evidence for the risk quantification. Due to implementation limitations in the HW QR-TPM, which is based on the ASIC FPGA board, we proceeded to the reengineering of the FutureTPM eBPF

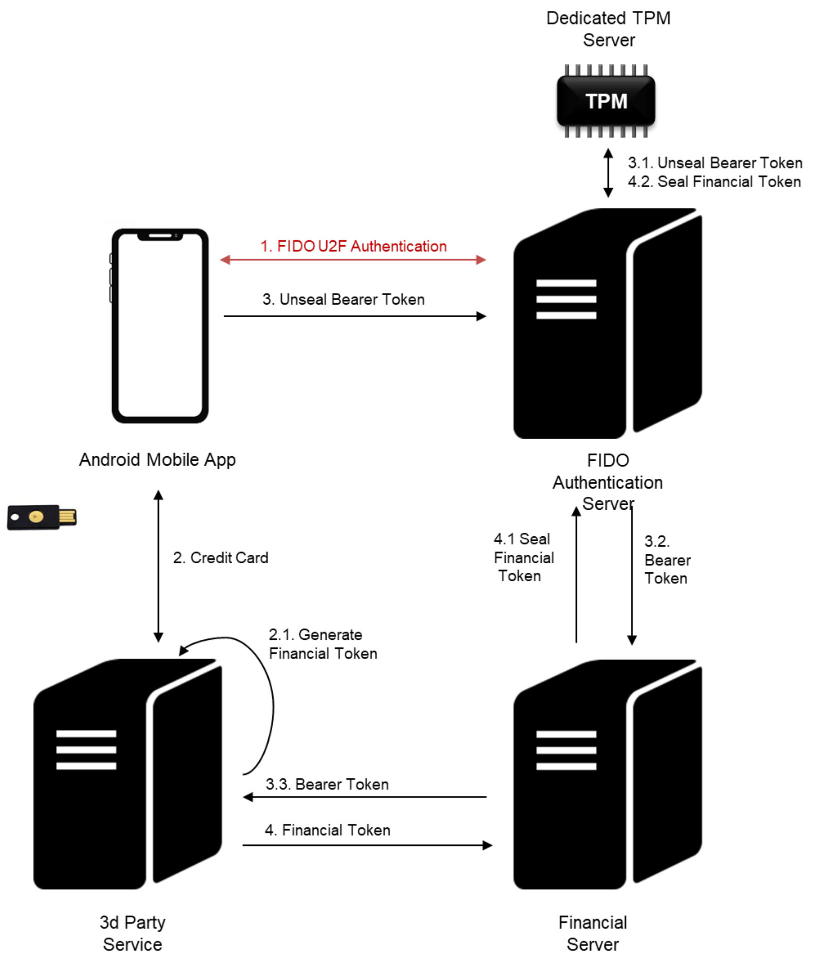
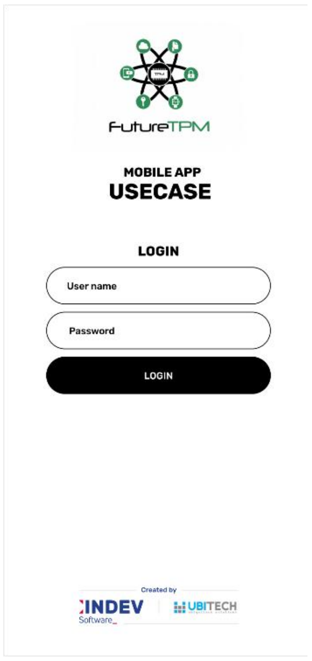
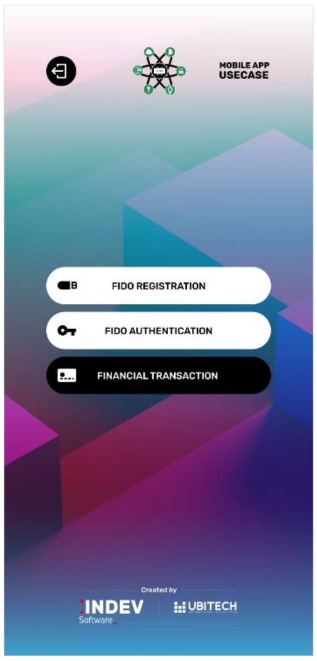
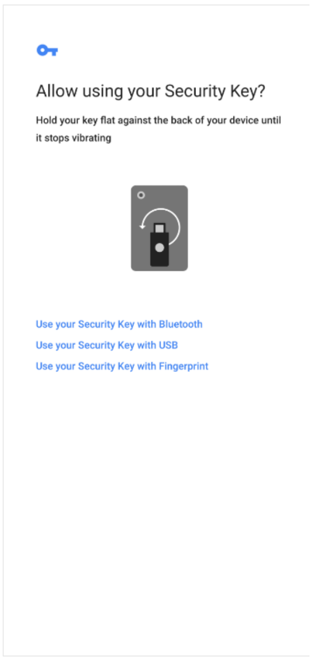


Figure 1: Secure Mobile Wallet and Payment High Level Approach

tracer [3, 24] in order to be able to intercept the TPM commands execution over the network. Figure 1 above presents the high-level approach of this reference scenario introduced in D6.1.





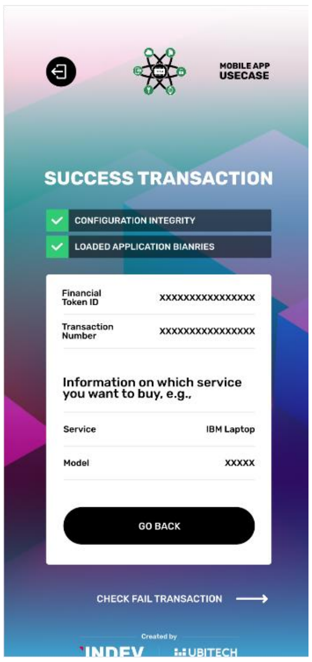

Allow using your Security Key?

Hold your key flat against the back of your device until it stops vibrating

Use your Security Key with Bluetooth
Use your Security Key with USB
Use your Security Key with Fingerprint

(b)

(a) (c)



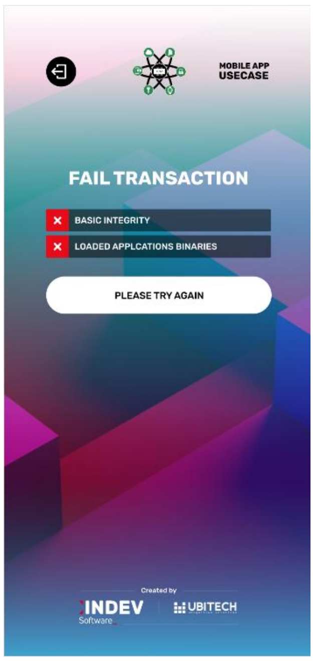
SUCCESS TRANSACTION

✓ CONFIGURATION INTEGRITY
✓ LOADED APPLICATION BIANRIES

Financial Token ID	xxxxxxxxxxxxxxxx
Transaction Number	xxxxxxxxxxxxxxxx

Information on which service you want to buy, e.g.,

Service	IBM Laptop
Model	xxxxx



(d) (e)

Figure 2: Mobile App environment. (a) User login, (b) FIDO Registration/Authentication and execute financial transaction functionalities, (c) Interaction with YubiKey, (d) Successful Configuration Integrity verification for transaction execution, (e) Failed Configuration Integrity verification for transaction execution.

The demonstrator which has been 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 and facilitate the remote attestation functionalities. This approach brings the ability to further extend our solution and apply prominent authentication mechanisms, such as FIDO Universal 2nd Factor (U2F), and attest the operational state of the mobile device, by establishing a communication channel between the Android application and the dedicated TPM server. Figure 2 illustrates the mobile application environment used to facilitating the necessary functionalities for the realisation of the 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. That is, this reference scenario is demonstrated based on the use of the **hardware TPM**, which is released on an FPGA-based board exposed by TCP/IP. For that reason, we decided to adapt the architecture and host the hardware TPM in a dedicated cloud server. The assumptions made in order to demonstrate this reference scenario are:

- The FPGA-based Hardware QR-TPM is connected to a dedicated TPM server but acts as an integral component of the mobile device;
- An authenticated channel is established between the Android mobile app and the TPM server based on FIDO U2F signalling;
- User register 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;
- The developed eBPF-based tracer has been refactored for TPM command interception on the network level (instead of the Kernel level) due to the FPGA-based implementation of the HW QR-TPM.

Even if the aforementioned assumptions have been made to facilitate the integration of the QR trusted component to the mobile device, it has to be stated that -in parallel- these assumptions highlight the need to foster the research and standardization actions for creating trust enablers for mobile environments. **The design and adoption of such trust anchors for mobile devices will enable the provision of secure and trusted functions, which can enhance the security posture of security-sensitive business domains, such as the financial technologies.** Thus, apart from the benefits of ensuring trust by integrating a QR-TPM in the Secure Mobile Wallet and Payments use case, FutureTPM project highlights the need for the community to investigate for viable solutions and proceed to standardisation actions based on Trusted Computing architectures.

2.1.1 Overview of the FIDO U2F Registration and Authentication Phases

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 and Figure 3 present the aforementioned challenge-response flows for the Registration and Authentication phases respectively.

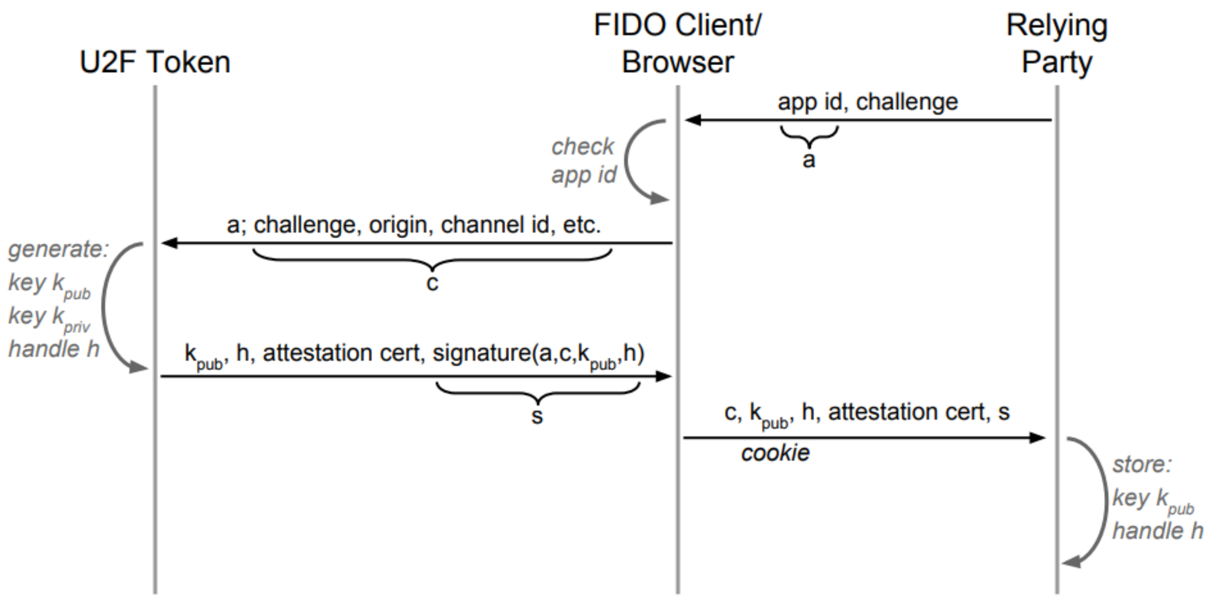

U2F Token
FIDO Client/ Browser
Relying Party
app id, challenge
a
check app id
a; challenge, origin, channel id, etc.
c
generate: key k_pub key k_priv handle h
k_pub, h, attestation cert, signature(a,c,k_pub,h)
s
c, k_pub, h, attestation cert, s
cookie
store: key k_pub handle h

Figure 3: U2F Registration

Note that, U2F authentication is an **extra layer of security** introduced in D6.1 and it was outside the scope of this demonstrator at first place. However, based on the comments received from the project's Advisory Board, we will use it explicitly 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.

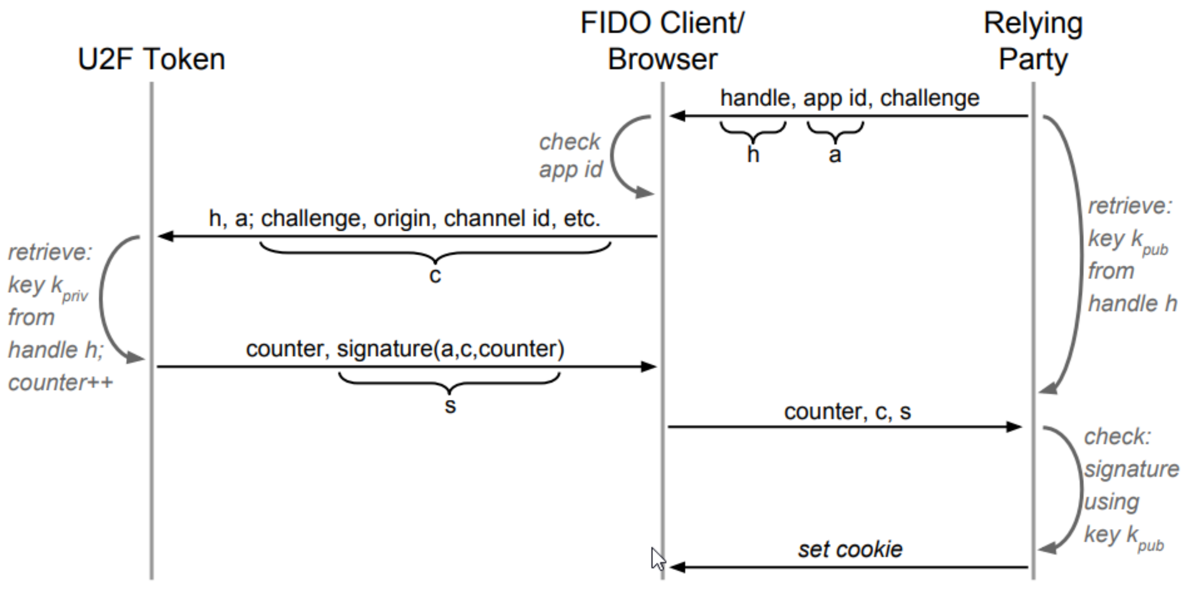

U2F Token
FIDO Client/ Browser
Relying Party
handle, app id, challenge
h
a
check app id
retrieve: key k_pub from handle h
h, a; challenge, origin, channel id, etc.
c
retrieve: key k_priv from handle h; counter++
counter, signature(a,c,counter)
s
counter, c, s
check: signature using key k_pub
set cookie

Figure 4: Mo: 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 the a) sealing functionality for the Bearer and Financial Tokens, and b) the unsealing functionality for the Bearer Token, have been demonstrated in D6.3. However, in order to realise the rest of the user stories in the 2nd experimental phase, a user needs to be registered and authenticated to the service in order to interact with the e-payment service. In addition, the registration and authentication process, described above, will be used as the basis for the analysis conducted for INDEV.AU.5 and how a TPM can be used in synergy with FIDO U2F Protocol in Section 2.1.3.

In addition, in the context of this use case, we do not solely focus on meeting the trust and operational assurance requirements of the field by using the QR trust enablers, but we also contribute to the privacy preservation of users through the integration of Direct Anonymous Attestation (DAA) protocol in the FIDO U2F Registration and Authentication phases. In this way, FutureTPM project aims to go beyond the provision of QR-TPM, as trust enabler in the financial domain, but aims to enhance FIDO protocol with trust and privacy preserving qualities. The FutureTPM consortium aims to push the updated FIDO models, which are describe in Section 2.1.3 in detail, to the FIDO standardization bodies for consideration to the future releases of the technical specifications. In fact, the latest FIDO working group has already identified DAA as a privacy preserving primitive that can benefit the FIDO protocol; however they have not released the technical details to achieve it. FutureTPM consortium has identified this gap, and our endeavor aims to address this limitation.

The next section elaborates on our actions towards enhancing the security posture of the Fintech application domain by integrating and showcasing the Configuration Integrity Verification of the mobile device using the two attestation schemes, namely Attestation by-proof and Attestation-by-Quote, as have been introduced in the FutureTPM project in D4.4 [25].

2.1.2 Overview of Remote Attestation Schemes: Attestation by-proof and Attestation-by-Quote

As aforementioned, part of the Secure Mobile Wallet and Payments use case will be the demonstration of two attestation schemes namely, **Attestation by Proof and Attestation by Quote** (as depicted in the left and right side of *Figure 5*, respectively), for enabling the automatic, or upon request, secure establishment of trust between the Mobile App and the backend banking system. Note that, in the generic representation of *Figure 5*, the Attestation software agents, i.e., the Verifier (Vrf) and Prover (Prv), correspond to the banking server and Mobile App (that works in synergy with the HW QR-TPM), respectively.

The evidence of the integrity state of the mission critical resources or functionalities on the mobile device are authenticated by the attached HW QR-TPM. Thus, the Attestation by Quote and Attestation by Proof processes, are used in the realisation of the INDEV.AU.3 and INDEV.AU.4, respectively.

More specifically, the Attestation by Quote enables the integrity verification of the mobile device without conveying additional or unnecessary information of the underlying host to the remote verifier. In the context of INDEV.AU.3, the aim is to attest a local database containing the financial transactions history. Every time a transaction is made, a new entry is appended in the database. To ensure the database integrity, a database digest is stored in the PCRs of the TPM. Upon an attestation request of the Verifier, the mobile APP interacts with the QR-TPM, which constructs a quote structure comprising the current values of the chosen PCRs, and signs it with a key

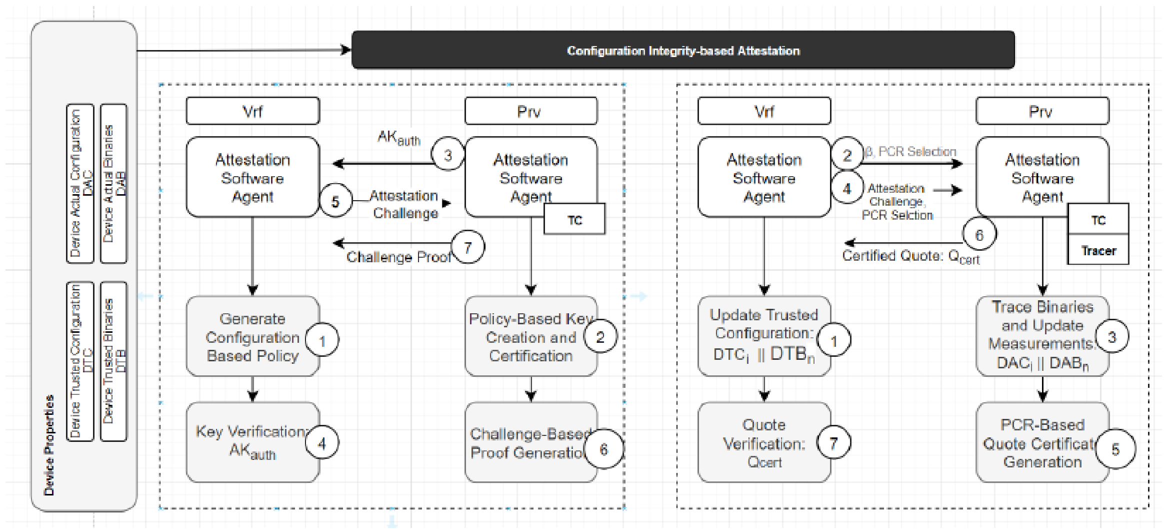

Configuration Integrity-based Attestation
Device Properties
Device Actual Configuration DAC
Device Actual Binaries DAB
Device Trusted Configuration DTC
Device Trusted Binaries DTB
Vrf
Prv
Attestation Software Agent
AK_auth
Attestation Challenge
Challenge Proof
TC
Generate Configuration Based Policy
Policy-Based Key Creation and Certification
Key Verification: AK_auth
Challenge-Based Proof Generation
β, PCR Selection
Attestation Challenge, PCR Selction
Certified Quote: Q_cert
Tracer
Update Trusted Configuration: DTC_i || DTB_n
Trace Binaries and Update Measurements: DAC_i || DAB_n
Quote Verification: Qcert
PCR-Based Quote Certificate Generation

Figure 5: Workflow of system Configuration Integrity Verification: Attestation by Proof (Left) and Attestation by Quote (Right)

generated by the QR-TPM using the BLISS signature scheme. The quote certificate and signature are then sent to the Verifier. Following this approach, the correct state of the recorded transaction history on the mobile device can be attested by providing the necessary evidence to the server.

The Attestation by Proof schema allows for attestation without disclosing any information that can infer identifiable characteristics about the individual configurations of the attested system. This scheme is utilised in the context of INDEV.AU.4, where the Verifier attests the sequence of QR-TPM commands executed on the mobile device (i.e., the extracted CFG) for the realisation of the INDEV.AU.3. Upon a Verifier’s attestation request which includes a nonce n and a policy digest which reflects the reference value of the operational state of the mobile App, the latter presents a signed nonce to the Verifier as an indisputable evidence that the App’s execution has resulted the correct measurement.

Section 2.2.1 offers a more detailed description of the developed workflows for the aforementioned attestation schemes in the user stories. The interested reader can refer to D6.4 and D4.4 for more details on the two schemes.

2.1.3 *Strong Authentication by integrating the use of TPMs and DAA in the FIDO U2F Protocol*

This section elaborates on the modelling of integrating the TPM in the FIDO U2F Protocol and how the DAA algorithm is used for achieving user-controlled unlikability in the financial services domain. By enabling both FIDO authentication and DAA services, we achieve authenticated and anonymous verification of Yubico credentials in the identity management process of financial transactions. More specifically, in this section we present the overall system model that shows its internal components. After introducing the components, we look into the required trust modelling.

Note that, this thorough analysis is presented as the realisation of the INDEV.AU.5 user story, which poses a significant research challenge.

2.1.3.1 System Model

Figure 6 overall system model and the different actors of the proposed system. This setup is an extension of the standard U2F setup, while we replace the roaming authenticator with a TPM. In addition, the Issuer, i.e., the manufacturer of the module, is presented as an additional, but not active, asset in the U2F specification. For a roaming authenticator such as Yubikey, the Issuer would be Yubico. The different entities which are engaged in the systems are the followings:

- **Service Provider:** The Service Provider is the entity that the user wishes to authenticate with. The service provider also acts as a verifier, as it verifies that the TPM is valid, using anonymous signatures created based on the DAA protocol.
- **Host:** The host contains the client and is also called the platform (the host and TPM). The host could be any device that integrates a TPM, such as the smartphone in the case of the e-payment use case. It is responsible for handling communication with the service provider, as in the FIDO U2F protocol. The host further provides information to the TPM regarding the service provider it is talking to, again as in the original protocol.
- **Trusted Platform Module:** The TPM resides in the platform and is physically bound to it. The TPM provides secure cryptographic functionalities and is used to generate keys and signatures. The TPM also has the responsibility of measuring the integrity of the host, i.e., by hashing applications, to ensure device integrity. The TPM encrypts and decrypts Service Attestation Keys, which are unique to a user/service-provider pair.
- **Service Attestation Key**: A service attestation key (SAK) encrypts and decrypts multiple authorization keys. Such a Service Attestation Key is unique to a specific service provider and a single user. The key can only be used when a request comes from the service provider it is linked to.
- **Authentication Key**: The authorization keys are used for signing the challenges received from the service provider. Those keys are protected by policies which virtually ensures that only the right user, under the right circumstances, can gain access to the key. The authentication key is equivalent to the keys described in the FIDO U2F Protocol, but in our case, these are a product of the TPM functionality and, thus, they are better protected.
- **DAA Key:** This is a unique key only accessible to the TPM. The DAA key provides enables the anonymous attestation. This can be compared to the attestation certificate in the FIDO U2F protocol.
- **Issuer**: The Issuer is the manufacturer of the TPM. In the FIDO U2F protocol, this is the relying party.
- **User**: The user(s) are the last actor and are responsible for providing authentication data such as passwords, biometric data, etc. to the host to be able to unseal keys. The user interacts with the host device.

Note that, the designed system takes advantage of the key hierarchy. The keys that are higher in the hierarchy are used to wrap, and thus protect, other keys lower in the hierarchy. In this direction, a primary key wraps the service attestation key, and that key wraps authentication keys. A service attestation key is unique to a user and service provider and wraps all the authorization keys used to authenticate the user to that given service provider.

2.1.3.2 Requirements

A requirement of paramount importance is that we must ensure that the system will only operate when the host is in a trusted state. There may be several aspects of the deployed system that need to be trusted in order to ensure that the system in its entirety is in a legitimate operational state, so that to have this requirement valid and to uphold. That is, one need to ensure the integrity of the systems during the creation of the service attestation keys or DAA keys.

In the developed concept of TPM integration in the FIDO U2F protocol we consider that the underlined system supports Configuration Integrity Verification (CIV), and it is measured both during load-time (before and immediately after boot) and run-time. For the overall system, this means that in case an attacker executes attacks on the system post-boot, the attestation process could reveal discover in cases where the attack changes the system’s behavioural profile.

To be able to achieve trust in the dynamic environment of the platform dynamic measurements are needed, such as Control-Flow Attestation techniques. In the models, we assume that universal standards for device integrity are in place and are being checked, meaning that any operation can only happen if and only if the device meets the integrity requirements. These requirements are met by the measurements gathered during boot and during run-time and guarantee that the system configurations are as expected.

In addition, apart from the trust requirements that drive the design of the models to be presented in the next section, there is a strong requirement in protecting user privacy. In this regard, the utilisation of DAA in the context of FIDO protocol and the utilisation of DAA keys and protect users' privacy against service provider that could potentially link a user's activity among different services.

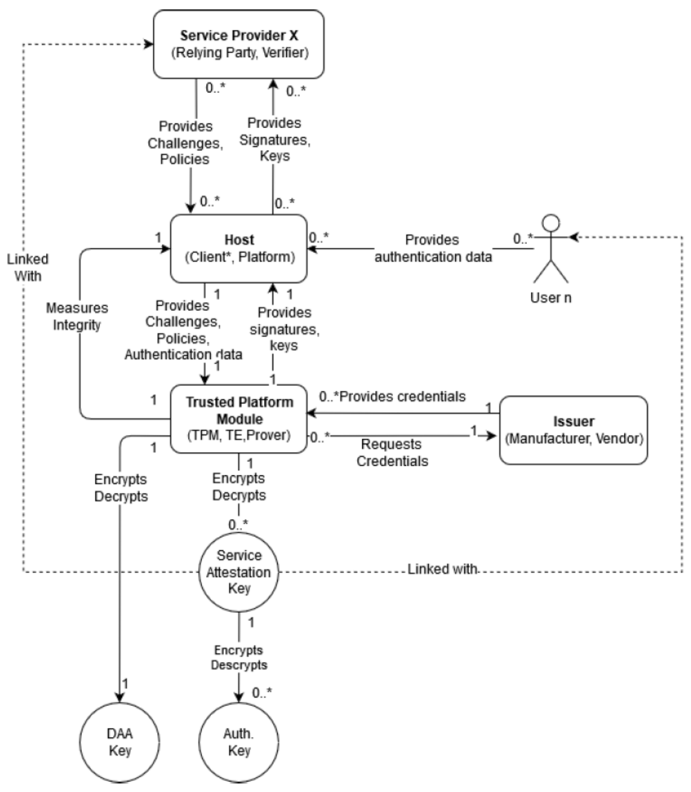


Figure 6: System Model

2.1.3.3 Trust Models

The models are designed to capture the assumptions and relationships among assets and entities needed for the system to operate in a trusted manner. They should capture not only the use-cases of the system, but also the potentially untrusted entities and assets, such as objects, data, etc.

The final goal of these models is to provide a formally verifiable trust model, that could be verified with tools such as ProVerif [19] or Tamarin [20]. These models can be used to generate attestation policies that can protect mission critical operations and guarantee operational assurance. The formal verification of these models is out of the scope of this deliverable and has been identified as future work after the completion of the project.

Towards this direction, we make use of diagrams to model the trust relations between entities in the system. **These diagrams depict a sequence of states that reflect assets, entities, code-execution, etc., which the underlying trust assumptions and requirements**. If any of the states do not meet the necessary requirements, they cannot be trusted; hence, the overall security cannot be guaranteed. Using these models, we can describe a generic enhanced version of the FIDO U2F Protocol that operates in tandem with a TPM.

We limit the modelling only to those diagrams that depict the core attestation and key management functionalities and trust requirements of the system:

A. Create Service Attestation Key

The Service Attestation Key is used to wrap the user authentication keys, as extracted by the attached Yubiko. They are unique to a specific user and a specific service provider. During registration, this key is created and meets the requirements of origin-specific keys, since the authentication keys that this key wraps, can only be unlocked if the correct service provider is present.

As shown in Figure 6, the key is encrypted by the TPM. In reality, it is encrypted by a primary key that lies within the TPM. If this model succeeds, a key is provided to the host where the public part is available, and the private part is encrypted by the primary key, and can only be decrypted in the TPM, if the correct AuthValue is provided. Assuming all states are successfully executed, a key is created and sealed to an authorization value that represents the origin.

- **Precondition**: Primary key created.
- **Postcondition**: Service Attestation key wrapped by primary key, and the key blob is returned.

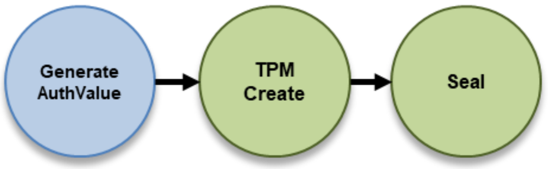


Figure 7: State diagram showing the states for generating a Service Attestation key. Blue takes place on the host and green in the TPM

Generate AuthValue: This AuthValue is required to protect the overall system of MITM-attacks by using origin-specific data. It combines the information of a user-provided password, the relying party as seen from the host, and a public known randomness. This value is used as an authorization value (password) for the key.

TPM Create: Creates the attestation key that is used to wrap authentication keys. The AuthValue is set as the host-provided value.

Seal: Seals the newly created key to the authorization value provided.

B. Create Authentication Key

The authorization key (pair) is the key that is used during authentication. When created, the public part of the key can be sent to the relying party. The key is created with a set of policies that may be required by the service provider. This process is exemplified in the model of Figure 8 with a configuration whitelist. This requires a service attestation key to be loaded (as we defined in the previous model). This can only be done if the origin of which we are creating authentication keys is the same as when the service attestation key was created. As with the Service Attestation Key, we are getting data output to the host, where the public key is readable for the host, but the secret part is not exposed out of the TPM.

- **Precondition:** Primary and Service Attestation key created.
- **Postcondition:** Authentication key wrapped by attestation key.

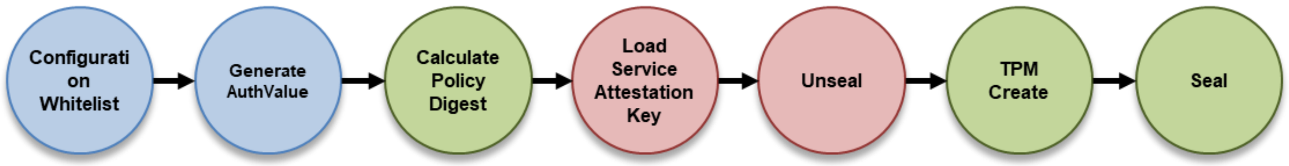


Figure 8: Diagram showing the states for generating an authentication key. Blue is provided by the host, green is states with relation to the authentication key and red is in regards to the service attestation (parent) key

Configuration Whitelist: The provider may require one or more configuration whitelists to be considered in the generation of an authentication key in order to add as an extra factor to achieve trust. Such configuration whitelists are used to provide a trust-baseline for unsealing the key.

Generate AuthValue: The process in which the unique authorization value needed to unseal the service attestation key, is provided. An example could be a SHA256 hash of some application-specific data, user-specific data (password), and some randomness to ensure uniqueness. This value is to ensure that an adversary intercepting communication is not providing data to the system.

Calculate Policy Digest: In an authenticated session, the policy digest is calculated to be able to unseal the key. The digest is calculated by executing different policy commands on the TPM, followed by getting the newly created digest. Such a digest can be any combination of the policies to support the requirements of multi-factor authentication.

Load Service Attestation Key: Loads the attestation key to act as the parent of the authentication key under creation.

Unseal: This process unseals the Service Attestation Key (inside the TPM, the private key is not available to the user)

TPM Create: Creates the Authentication Key. This state must provide a unique key that cannot be linked to any other authorization key, created under the same service attestation key.

Seal: Seals the Authentication Key to the configuration whitelist and the policy digest value generated.

The whitelist is by design the foundation of the policy, as it should represent the desired state of the configurations in the system. The registers of the TPM can be extended but not overwritten. Because of this, the actual policy-value can be generated by extending that value with the secondary policies. This model shows how the TPM can fulfil requirements to generate keys that are only readable in the TPM by design. Due to the nature of the policies, any number of policies could be used to seal the key.

C. Create DAA Key

The DAA key is used in the DAA protocol for the purpose of attesting the validity of the TPM. The DAA key is created in order to comply with the requirement of attestation. This key is encrypted by the TPM, while a primary key must be created. Since the DAA key is used to attest the validity of the TPM during authentication, it must be protected by a platform password.

- **Precondition:** Parent key created.
- **Postcondition:** DAA Key created and sealed.

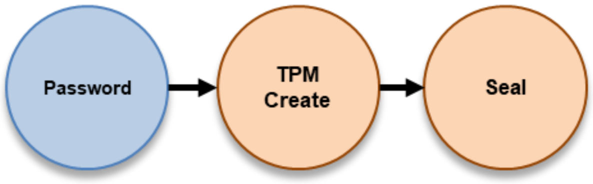


Figure 9: State diagram showing the states for generating an ECC Key for Direct Anonymous Attestation. Blue actions occur on the host, and orange in the TPM.

Password: This is the password the key is to be sealed with. The password is a secret value and should, therefore, only be known to the owner of the platform.

TPM Create: Creates the ECC DAA Key with the password send with as authValue.

Seal: Seals the newly created key to the authorization value provided.

The DAA key creation process follow the same approach as the one for the attestation key shown in Figure 7. To ensure that nobody else than the platform owner can create the DAA Key, the key is sealed using a password.

D. Create Attestation Signature

To be able to attest to the validity of the trusted module, we need to provide a signature from a DAA key. This signature establishes a guarantee at the service provider that the trusted module, i.e., is a TPM and that it is valid.

- **Precondition**: DAA Setup and Join protocols completed successfully.
- **Postcondition**: Signature created

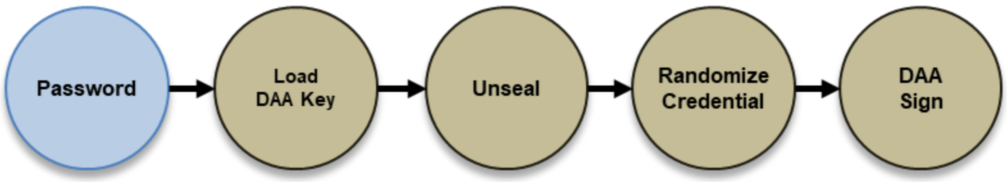


Figure 10: State diagram showing the states necessary for signing an Authentication Key for Registration with a Relying Party.

Password: The password is required to unseal the DAA key. This is provided untampered and is known only to the platform owner.

Load DAA Key: Load the DAA Key into the TPM.

Unseal: Unseal the DAA key with the password (AuthValue) provided by the host.

Randomize Credential: Before using the credential, it is necessary to randomize it such that the signatures are unlinkable to meet requirement of establishing a protocol that must support multiple unlinkable keys. If linkability is desired between signatures, a basename can be provided here.

DAA Sign: The signing operation with the DAA key. The signing operation is done on the marshalled public key. This is done by using the TPM's commit function and further calculations on the host.

The goal of the use case is to provide a signature to a public key that can be verified as only being able to be produced on a valid TPM. Since it is the public part of an authentication key that is to be signed, no special operations are needed to load this. The public key is being signed to convince the verifier that the public key originated from a valid TPM. Indeed, the key could be protected by one or more policies, though working under the assumption that the password is only known to the platform owner, this is can be considered sufficient. By using Direct Anonymous Attestation, the ability to link different attestation signatures with each other is eliminated.

E. Create Authentication Signature

This is a core functionality for the authentication phase of the protocol. The private key is used to sign the data. The corresponding public key lies on the service provider. This is a multi-stepped process since we need to unseal not only the service attestation key but also the authorization key.

- **Precondition:** Authentication Key, Service Attestation Key and Primary Key are created.
- **Postcondition:** Signature is created.

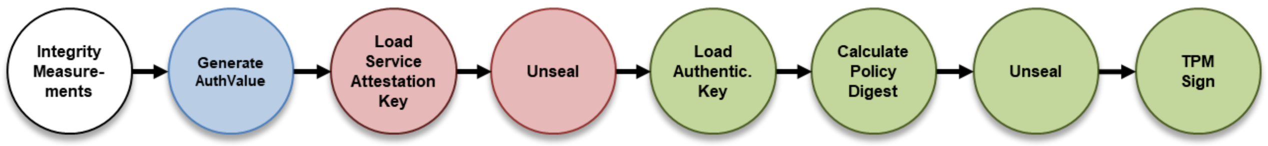


Figure 11: State diagram showing the necessary states for signing a host-provided dataset for authentication to a Relying party.

Integrity Measurements: Integrity Measurements are being done during boot and create a set of digest values stored within the TPM. These values represent the current configurations of the platform and range from low-level configurations such as the BIOS, MBR, etc., all the way to post-boot measurements of installed applications and their configurations.

Calculate AuthValue: The value based on application and user data, and predefined random data. It is used to gain access to the Service Attestation key and protect against man in the middle attacks.

Load Service Attestation Key: Load the service attestation key into the TPM.

Unseal: Unseal the Service Attestation Key, which gives the TPM access to the private parts of the key. The private part is never revealed to the user.

Load Authentication Key: Loads the selected authentication key into the TPM, but the private part of the key is still sealed.

Calculate Policy Digest: In an authenticated session, the policy digest is calculated to use the authentication key. This must be done by executing the required policies correctly and in the correct order. If a configuration whitelist digest is provided by the relying party during setup, the first policy to execute must be the PolicyPCR followed by any policies required.

Unseal: The unsealing of the authentication key based on the policy digest calculated before.

TPM Sign: The signing operation with the Authentication Key on the host provided data. This succeeds if and only if the policy-digest matches that of which was generated during creation, and if the correct authvalue for the parent is provided.

Assuming all the steps in model for the creation of the Authentication Key (Figure 8) are executed correctly, then it is only possible to sign the data provided if both the service attestation key can be unsealed (based on the client's perspective of the verifier and a password) and the platform's integrity measurements meet the expectations of the relying party and any other negotiated policies. This ensures the necessary functional security requirement that keys can be generated and exercised under specific circumstances, since any (intentional of unintentional) change in the state of the system will result to a failure of the unsealing process a key.

2.1.3.4 Adapted design of the registration and authentication protocols

Given the models described in the previous section that guarantee the trust requirements for the functionalities key generation and management delegated to the TPM, this section elaborates on how the registration and authentication protocols of the FIDO U2F can be adapted to integrate the trust assurance thought the TPM and the competitive advantages of DAA for achieving enhanced privacy of the end-user.

2.1.3.4.1 Registration protocol

The FIDO U2F protocol does not focus on users' privacy, but only on providing a second factor, while the user is authenticated using a username and password during registration. The aim of the adapted protocol is to ensure users privacy by exploiting the benefits of DAA. The adaptation of the protocol aims to be as generic as possible. That is, we do not specifically define the authentication method to be used. Instead, we change this to a more generic term *basic authentication*. Basic authentication implies that the host (or user of the host) can provide evidence that it is eligible for registration. By doing this, we ensure that this setup can be used to obtain both full- and partial anonymity. A user could use other forms of authentication, e.g., password, group signature, or any other way of authentication that lets the relying party verify that registration is allowed.

On the basic authentication request, the relying party responds with a challenge. Since we aim to perform local multi-factor authentication, the process must meet policy requirements to comply with. These requirements are sent back with the challenge and can, for example, contain a configuration whitelist, requirements to password, biometric data, and so forth.

The next step engages a challenge-response message, i.e., the response from the host to the challenge during registration. This message originally contains an attestation certificate that allows the relying party to validate the type of U2F Device. If the relying party does not link this with the to-be-stored public key, then the unlinkability requirement can still be met among future authentication requests.

To achieve full anonymity, we need to provide attestation without revealing our identity. To do so, Direct Anonymous Attestation (DAA) has been promoted as the prominent methodology to meet the privacy requirements in the context of the project.

Direct Anonymous Attestation (DAA) is a protocol that offers anonymous attestation of a platform without providing information about the platform itself. It still relies on a common trusted party such as the TPM Manufacturer, which we call Issuer. The protocol consists of two main phases: *Join and Sign*. In the first phase, the TPM (the Prover) proves to the Issuer its identity and sends along a DAA Key (an ECC key). The Issuer then issues a credential based on that key to the Prover. When the platform needs to provide its identity to a party, it scrambles the credential and creates a signature with the modified credential. It is up to the platform whether the signatures provided can be linked to each other, by introducing a *basename* into the scrambled credential.

When the Verifier receives the signature, it can verify that it is legit based on a Zero-Knowledge proof - without talking to the Issuer. With this approach, the Issuer is only contacted when it needs to issue a credential, and a verifier can verify a signature independently of the Issuer.

Given the advantages of the DAA, the proposed approached for the adapted registration protocol replaces the attestation certificate with a DAA signature over the public key. By doing so, the host proves to the relying party that a valid TPM is used, and the public key is acknowledged as a key derived from a valid TPM.

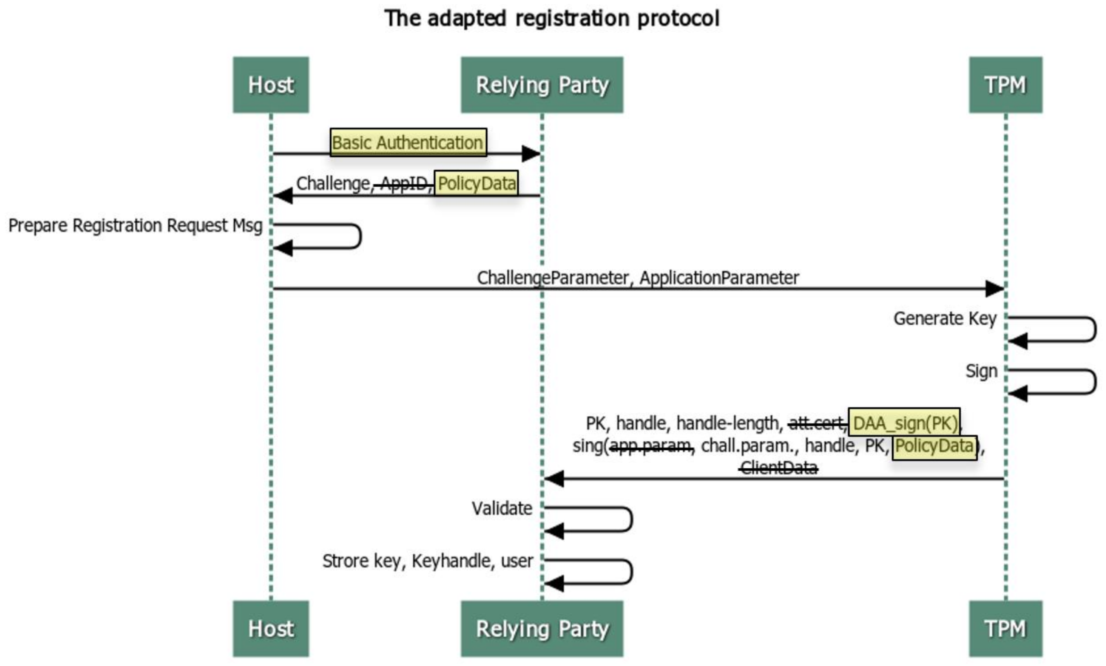

The adapted registration protocol
Host
Relying Party
TPM
Basic Authentication
Challenge, ~~AppID,~~ PolicyData
Prepare Registration Request Msg
ChallengeParameter, ApplicationParameter
Generate Key
Sign
PK, handle, handle-length, ~~att.cert,~~ DAA_sign(PK),
sing(~~app.param,~~ chall.param., handle, PK, PolicyData),
~~ClientData~~
Validate
Strore key, Keyhandle, user
Host
Relying Party
TPM

Figure 12: The adapted registration protocol. New items are highlighted in yellow, line-through items have been removed.

Figure 12 presents the adapted protocol as a result of the abovementioned analysis. Overall, by utilizing DAA, the host can anonymously register to a party without revealing its identity. The proposed approach eliminates the requirement for the username and password combination and leaves it to the relying party to decide the authentication scheme. In addition, a new parameter is introduced, namely the *PolicyData*, to instruct a set of policies the key must be sealed with. The AppID factor is no longer needed as the origin of the host is attested thought the DAA signature.

Basic Authentication: A relying party shall perform a basic registration/authentication in order to grand access to a service. Authentication is necessary but can be of different levels. To support requirements, this basic authentication must be able to be unlinkable to a specific user or person.

Challenge: The challenge is a simple nonce, which is essential to protect against replay attacks. As a new parameter, the adapted protocol considers the PolicyData, which is a (potentially negotiated) set of policies used to seal the key. This can be requirements of different authentication-schemes such as passwords, biometrics or an application whitelist.

Prepare Registration Request Message: The host receives the challenge parameter and immediately prepares the required client data, which is a hash of the challenge and the application ID seen from the host perspective - the hashed version of client data is called the

ChallengeParameter. The application parameter is vital, as it is used to prevent phishing and man in the middle attacks. Instead of the ApplicationParameter being just the digest of the AppID, we introduce it as the digest of the AppID and application ID seen from the host (e.g., AppID: Email, Application ID: https://google.com). We do this to protect the key from MITM attacks but support multiple applications on a single Relying party.

Registration Request: Both the ChallengeParameter and the ApplicationParameter is sent to the TPM and the latter is used for generating a signature and a key.

Generate and Sign: The TPM then creates a keypair and seals it to the origin described by the application parameter. It then returns a signature on those parameters along with the public key and the UUID (key handle). Note that, the key handle in the case of the e-payment use case is the one provided through the interaction of the Yubico. Further, the TPM returns a DAA signature over the PK. The ClientData is not sent with the message as it is not needed.

Validate: When the relying party receives the message, it should first confirm the DAA Signature, and if that holds, it can check the signature provided. In the original protocol, the ClientData is sent along for the Relying party to verify the signature and validate the data. In our case, we have no interest in knowing what went wrong (i.e., the origin is incorrect due to MITM), but simply if something is wrong. The Relying Party can simply reconstruct the ClientData as it expects it to look like, and then check the signature. If this process results to an unexpected outcome, it discards the key; otherwise, the key is stored but not linked with a user.

2.1.3.4.2 Authentication protocol

Analysing the authentication, the same issue arises as with registration: the first step includes the host authenticating using a username and password. Since the aim is to protect the host's privacy only proof that the host is registered is sufficient. That is, request for a challenge is sent. Obviously, since the Relying Party does not know who the user is, it cannot send the corresponding key handle. Therefore, the Relying Party simply provides the challenge-parameter. The platform locates a key registered with the service provider, which is then used to sign the challenge. In the response message to the host, there is no need for a counter since the TPM is by design un-cloneable, and the platform is verified when loading the key. The control-byte is no longer necessary as user's presence can be established with policies, e.g., by biometrics. The key handle (UUID) is sent along, such that the relying party can locate the key.

As is depicted in *Figure 13*, the adapted design has reduced the data needed to be transmitted, and no personal information is exchanged, but the Relying Party can be convinced that interfaces with a genuine user, since it has registered at an earlier state.

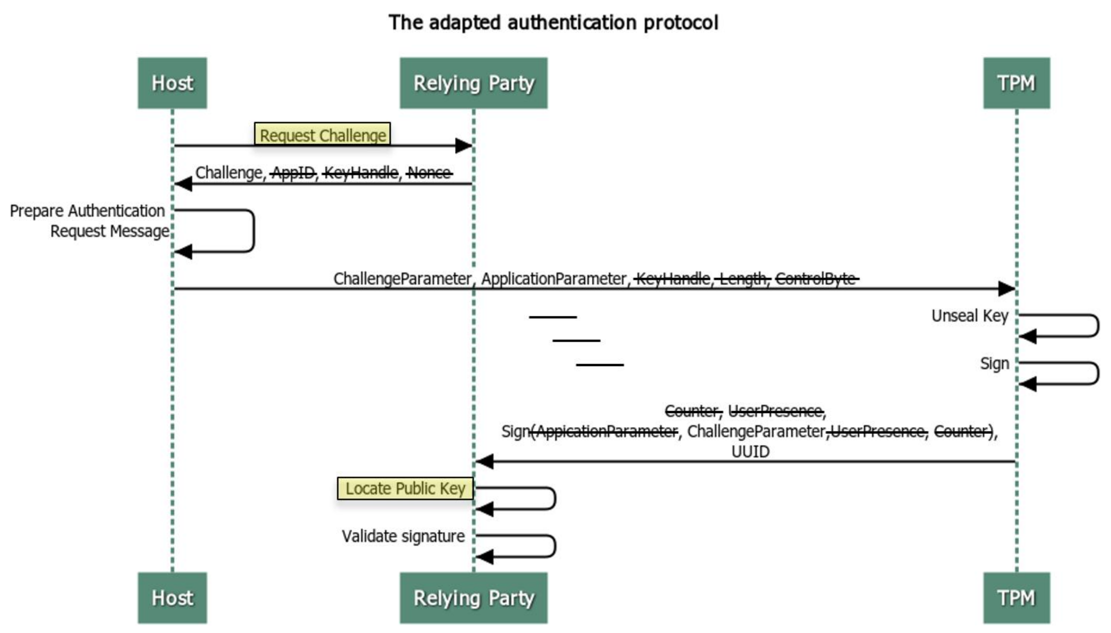

The adapted authentication protocol
Host
Relying Party
TPM
Request Challenge
Challenge, ~~AppID~~, ~~KeyHandle~~, ~~Nonce~~
Prepare Authentication Request Message
ChallengeParameter, ApplicationParameter, ~~KeyHandle~~, ~~Length~~, ~~ControlByte~~
Unseal Key
Sign
~~Counter~~, ~~UserPresence~~,
Sign(~~AppicationParameter~~, ChallengeParameter, ~~UserPresence~~, ~~Counter~~),
UUID
Locate Public Key
Validate signature
Host
Relying Party
TPM

Figure 13: The adapted authentication protocol. New items are highlighted in yellow, line-through items have been removed

Request: Since a username and password can reveal to the relying party who the user is, the adapted design simply requests a challenge.

Challenge: In the original protocol, the relying party would identify the user and provide a challenge, AppID, and a Key Handle. However, in the adapted design we argued in the registration phase why why we discarded the App ID, and since the relying party has no information for the user it cannot provide a Key Handle. Due to this, the relying party simply returns a challenge.

Prepare: Originally the host had to check the origin on the facet list fetched with the help of the AppID provided to see if the origin provided is legit. Instead, we simply create a ClientData structure and hash it to create the ChallengeParameter and hash the origin to provide the ApplicationParameter.

Unseal and Sign: If, and only if, the application parameter is the same as when the authorization key was created, then the authorization key can be loaded. In this case, the key needs to be unsealed with respect to the policies negotiated. If that is successful, then the TPM can sign the challenge parameter and send it. Note that, since the App ID is eliminated, it is not included in the signature. The UUID of the public key is also included in this message; otherwise, the Relying Party cannot locate the key. The key handle is essential to privacy: it is the host and not the relying party that decides what key to use. Since the user-presence can be defined in a protocol, this item is not returned, and neither is the counter due to the fact that the key could not be unlocked on any other platform (assuming configuration integrity verification and the fact that the trusted environment by design is assumed un-cloneable).

Validation: It is not before this step that the Relying Party can locate the public key. When the public key is located, the signature can be verified, and the user is authenticated.

2.1.4 Demonstrator Needs and Challenges

The 1st experimentation period, which resulted to the compilation of D6.3, was conducted using the Software-based variation of the QR-TPM and the implementation was focused on the realisation of the first two user stories of this use case. The 2nd experimentation period is conducted by leveraging the Hardware-based QR-TPM. Since, the HW TPM comes on an FPGA board, this means that for the demonstration purposes we had to make a set of reconfigurations to the reference implementation to interface the board. The major challenge faced in this transition was the necessary reengineering of the eBPF tracer which was designed in earlier stages of the project. More specifically, the deployed interception hook was modified in order to be able to intercept the TCP traffic generated on the network controller, as the FPGA is connected to the dedicated TPM server via an ethernet cable. In addition, as reported in D4.4 and D6.4, the eBPF tracer is aware of the internal structure of the TPM command codes in order to be able to decode them. That is, given that the HW QR-TPM comes with the extensions of the NewHope and BLISS QR schemes, we extended the TPM command decoder in order to capture the TPM inner features.

In parallel to the developments of WP4, and more specifically with the definition of the Attestation-by-Proof and Attestation-by-Quote schemes in D4.4, the developed user stories integrate the remote attestation artifacts of the FutureTPM project in order to provide a holistic evaluation approach. As such, the use of the FutureTPM enables the trusted execution of financial transactions for the Secure Mobile Wallet and Payments by providing evidence via eBPF tracing and the generation of CFGs and attesting the operational behaviour and the configuration integrity of the mobile device.

It has to be stated that the implementation of the INDEV.AU.5 poses a significant research challenge in the field of Authentication and financial technologies. Our research and developments led to the definition of the protocols and the models that have to be adopted in order to achieve the security and privacy requirements of this research challenge, as those have been presented in Section 2.1.3.

2.2 Implementation Path Report for the 2nd Experimentation Period

The focal point of the 2nd phase of the demonstrator is the **realisation of the INDEV.AU.3, INDEV.AU.4 and INDEV.AU.5 user stories**. 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. More specifically, the demonstrator has integrated the HW implementation of the QR-TPM, using the FPGA board connected to the dedicated TPM sever over an ethernet connection. The HW QR-TPM comes with the implementation of the NewHope Key Exchange and the BLISS digital signature schemes. Both schemes are utilised to support the functionalities of the aforementioned user stories.

The major challenges faced during this implementation had to do with the integration of the HW QR-TPM and creating an approach of measuring the QR-TPM performance by having the lowest possible interference to the operational profile of the FPGA board. 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, the eBPF tracer developed and documented in the context of WP4 was reengineered in order to be able to intercept the communication between the host machine (dedicated TPM server) and the FPGA board over the established TCP/IP connection. Thus, the interception timestamps between the request and response packets that encapsulate the TPM commands sent from/to the TSS residing in the host, can give us a quite accurate performance measurement for the execution of the QR algorithms on the PFGA. It has to be stated that due to implementation limitations, the network packet interception was chosen as the approach that offers the "closest proximity" to the FPGA. Note that, these measurements include the time for establishing the TCP/IP connection and the time for transmitting the TPM command packets. This time frame can be considered negligible.

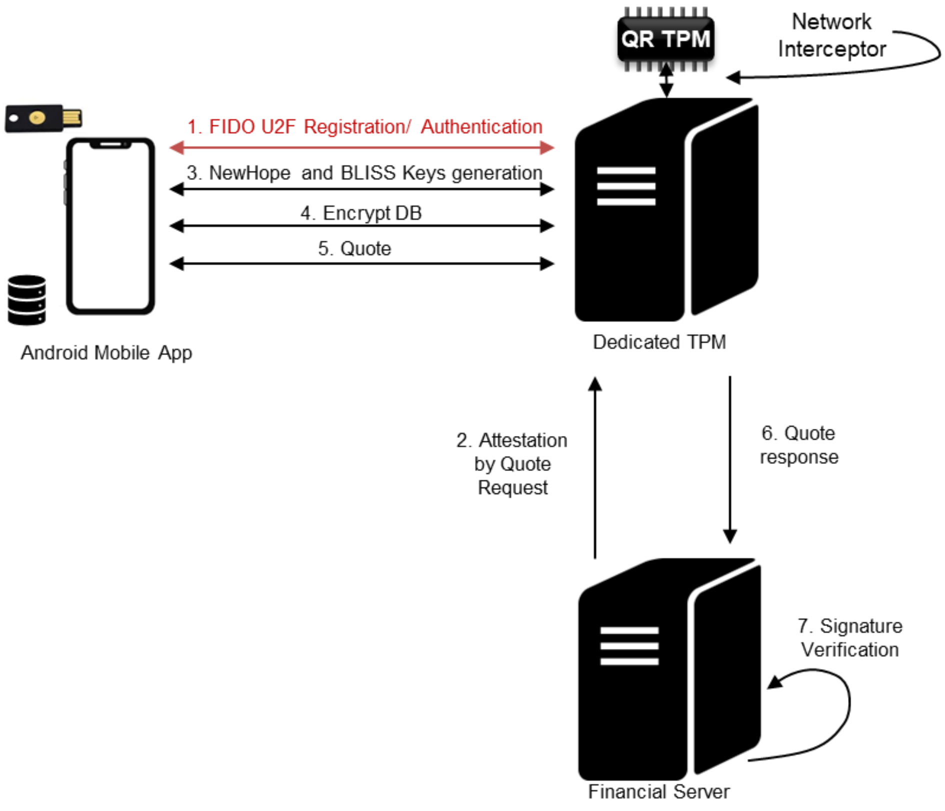
2.2.1 User Stories Realisation

Out of the User Stories and Test Cases described in D6.1, the INDEV.AU.1 and INDEV.AU.2 have been demonstrated in the context of D6.3, while the INDEV.AU.3, INDEV.AU.4 and INDEV.AU.5 have been scheduled for this period. The tables below describe the developed workflows for the corresponding user stories.

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): Completed in the context of D6.3

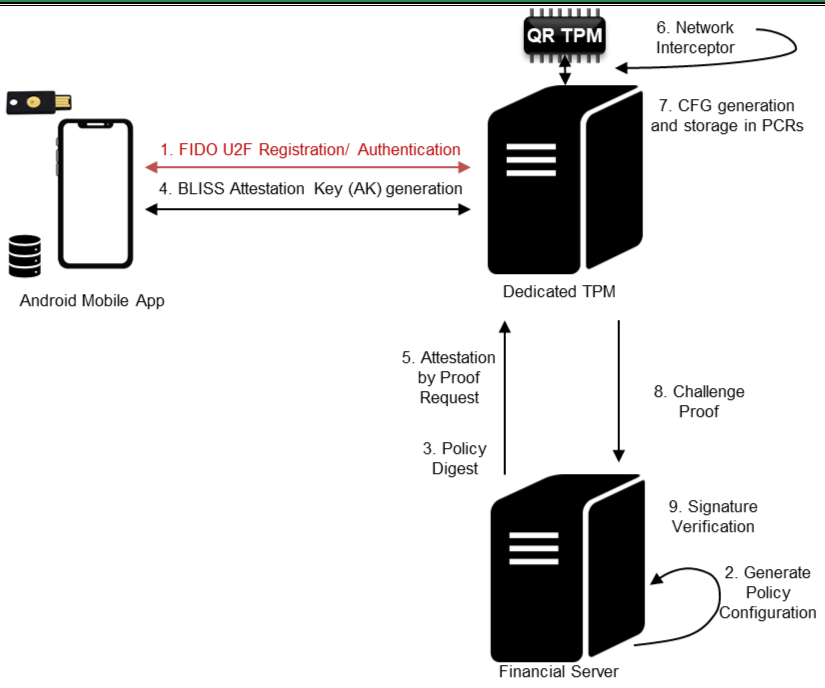
Description
User Story Title: *INDEV.AU.2 - As an Individual User I want to use an external service to generate tokens for my credit card that go directly in the TPM and avoid revealing my credit card to the server.*
Workflow Developed: The Android user authenticates to the TPM Server leveraging FIDO U2F when she wants to perform a TPM functionality (see *Figure 13*). 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): Completed in the context of D6.3

Description
User Story Title: *INDEV.AU.3 - As an Individual User I want to ensure that my financial transactions history is secure and not tampered with*   ***User Story Confirmations:***   • *The local database containing financial transaction history logs is encrypted with a key generated by the TPM and also their integrity is verified using the PCRs of the TPM.*

Description
Workflow Developed: The Android Mobile App, which is used for performing the financial transactions, maintains a local database containing the financial transactions history. Every time a transaction is made, a new entry is appended in the database. To ensure the database confidentiality, the transactions history is encrypted using a NewHope key, generated by the QR-TPM. The hash of the database is then stored in the PCRs of the TPM in order to ensure its integrity. An external entity (e.g., a financial service provider) acts as a verifier and initiates a Remote Attestation using the Attestation-by-Quote method. In the context of this use case this entity is the TPM server since the server is connected to the unique QR HW TPM. Thus, the Verifier sends a nonce *n* (used for the freshness of the interaction) and a selection of PCRs to attest, ***I***. The mobile APP passes these arguments to the QR-TPM, which constructs a quote structure comprising the current values of the chosen PCRs, and signs it with the attestation key generated by the QR-TPM using the BLISS signature scheme. In this way, it is certified that the quote structure has been generated internally by the QR-TPM. The quote certificate and signature are then sent to the Verifier. The quote and its signature are successfully verified by the Verifier, if and only if they are valid, and if the PCR values correspond to the artificial reference values already calculated by the Verifier. Given the above, the correct state of the recorded transaction history on the mobile device can be attested by providing the necessary evidence to the external entity. In this way, the e-Payment service provider attests the correct state of the transaction's history on the mobile phone and ensure its integrity.  QR TPM Network Interceptor 1. FIDO U2F Registration/ Authentication 3. NewHope and BLISS Keys generation 4. Encrypt DB 5. Quote Android Mobile App Dedicated TPM 2. Attestation by Quote Request 6. Quote response 7. Signature Verification Financial Server Figure 14: INDEV.AU.3 workflow. Attestation by Quote for the integrity verification of transactions DB
Issues Encountered: No issues encountered.
Status: Completed

Description
Degree of Realisation: Full
Comments (if any): We assume that the service provider is aware of the transactions' history of the client application. That is, the provider knowns the correct state of the PCRs to be attested. Using the Attestation-by-Quote method, the provider attests the correct state of the transactions' history on the mobile phone and ensures its integrity. By having the attestation result, the provider can decide whether new transactions are allowed to be performed by the mobile app, or to forbid financial transactions in case the database has been tampered. Note that, the encryption of the database on the mobile device occurs only for meeting the confidentiality requirement of the financial data on the device. The PCRs of the mobile device store the state to the unencrypted database to achieve the integrity verification.

Description
User Story Title: *INDEV.AU.4 - As an Individual User I want to verify the integrity of the systemic environment setup of the device used to connect to the service* *User Story Confirmations:* • *User can verify the operational correctness of the host device environment based on the CFG generated beforehand and reflect the normal behaviour of the device*
Workflow Developed: The realisation this user story implies the attestation of a critical function of the system in order to ensure the correct status of the device used to connect to the service. In this direction, we proceed to the attestation of the sequence of QR-TPM commands executed on the mobile device for the realisation of the INDEV.AU.3. Thus, this workflow requires the generation and storage of a Control-flow graph (CFG) generated by the deployed eBPF tracer which monitors the execution of the TPM commands at the Linux Kernel level. To achieve this goal, the developed workflow is based on the Attestation-by-proof schema. Initially, the Verifier sends a nonce ***n*** to the Mobile App. The mobile App presents a signed nonce to the Verifier as an indisputable evidence that the App's execution has resulted the correct measurement. To do so, the Verifier sends a policy digest which reflects the correct reference value of the run-time App's behaviour. This policy digest is used as a template policy which is used for the creation of a BLISS key, used for signing the nonce. The prover, i.e., the Mobile App, proceeds to the execution of the AU.3 and a tracing measurement is taken by the eBPF tracer and stored to the PCRs of the TPM. A Policy_PCR command is used to check the values of the PCRs and provide the proof that the captured tracing is the one which was provided by the Verifier at first place. If the Policy_PCR matches the current PCRs content with the policy digest, then the prover proceeds to the correct signing of the nonce provided by the prover and sends the corresponding response to the latter. In this way, the Prover, i.e., the server is in position to tell that the correct execution path, i.e., the CFG was followed on the prover during the execution of the AU3 realisation.

Description
 QR TPM 6. Network Interceptor 7. CFG generation and storage in PCRs 1. FIDO U2F Registration/ Authentication 4. BLISS Attestation Key (AK) generation Android Mobile App Dedicated TPM 5. Attestation by Proof Request 3. Policy Digest 8. Challenge Proof 9. Signature Verification 2. Generate Policy Configuration Financial Server Figure 15: INDEV.AU.4 workflow. Attestation by Proof for the verification of the operational correctness of the mobile application
Issues Encountered: No issues encountered.
Status: Completed
Degree of Realisation: Full
Comments (if any): It has to be stated the server is aware of the public part of the Attestation Key of BLISS, which is used to confirm the correct signature of the nonce.

Description
User Story Title: *INDEV.AU.5 - As an Individual User I want to perform the two-factor authentication with the Financial service through the TPM*
Workflow Developed: We approached the realisation of this user story from a research perspective as was initially the plan described in D6.1. More specifically, instead of an implementation of the actual required testbed, we proceed to an investigation on the challenges and models required for Integrating the use of TPMs in the FIDO U2F Protocol in order to provide a strong authentication scheme via Trusted Platforms. That is, the realisation of this user story has taken the form of a thorough analysis of the models and the adapted design of the FIDO U2F Protocol, as described in section 2.1.3. Crucially, our analysis is not solely focused on using the QR-TPM to the FIDO protocol, but on top of that, our modelling integrates the DAA in the FIDO protocol to deliver an updated authentication protocol with trust and privacy preserving qualities.
Issues Encountered: No issues encountered.
Status: Materialised as a thorough analysis of open research topic.
Degree of Realisation: Full.
Comments (if any): N/A

2.2.2 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.

Test Case MWP3	
Reference Code	MWP3
Components	Mobile App lib
Description	This unit test extends the functionality of the FUTURETPM02 and aims at verifying the correctness of the symmetric key generation. This unit test verifies the correctness of a TPM key creation. (INDEV.AU.3)
Status	Performed
Unit Tests Results	Instead of the generation of a symmetric key the consortium took the decision to generate the asymmetric key pair Generation based on NewHope QR algorithm for evaluating this newly deployed QR scheme. The

Test Case MWP3	
	key creation functionality has been successfully implemented and integrated in the HW QR-TPM. Encryption and decryption functionalities have been verified.

Test Case MWP4	
Reference Code	MWP4
Components	Mobile App lib
Description	This unit test aims at verifying the correctness of the integrity verification of the transaction’s history log. The unit test encrypts the history transactional logs. (INDEV.AU.3)
Status	Performed
Unit Tests Results	Instead of the generation of a symmetric key, the consortium took the decision to generate the asymmetric key pair Generation based on NewHope QR algorithm for evaluating this newly deployed QR scheme. Encryption and decryption functionalities have been verified. The integrity verification has been tested following the attestation by quote scheme and completed successfully.

Test Case MWP5	
Reference Code	MWP5
Components	Mobile App lib
Description	This unit test aims at verifying the operational correctness of the Android device that is connecting to the TPM server. This requires the generation of the CFG reflecting the normal behaviour of all application components (this will be created through the RA framework) and the subsequent run time monitoring and tracing of the CFPs and the verification against the generated CFGs. (INDEV.AU.4)
Status	Performed
Unit Tests Results	The generation of CFGs has been performed successfully based on the outcome of eBPF tracer developed as part of the RA framework in WP4. The verification has been performed using the Attestation by Proof schema. The verification of the attestation outcome is performed successfully.

Test Case MWP6	
Reference Code	MWP6
Components	Mobile App lib
Description	This unit test aims at verifying the operational correctness of the FIDO U2F

Test Case MWP6	
	two factor authentication, to be supported only through a TPM. (INDEV.AU.5)
Status	Not Executed, as the corresponding action realised by the analysis conducted in section 2.1.3.

2.2.3 KPIs Measured

During the second phase of the demonstrators, we proceed to the evaluation of the entire set of KPIs that was identified in D6.1. For these experiments, we measured the performance of the triggered TPM commands used for the realisation of the user stories. Note that, the timings in Tables *Table 3*, *Table 4* and *Table 5*, correspond to the average execution time after 100 executions of each experiment. In addition, specifically for the e-payment use case, we performed the performance evaluation for three deferent perspectives, namely, the Application, the TSS Layer and Network Layer perspectives.

More specifically, the application perspective provides the timings from the standpoint of the backend application of the financial transactions' server. The TSS Layer implies the instrumentation of the TSS stack by placing the proper hook at the TSS_Execute() function of the TSS. The Network Layer perspective measures the performance of the TPM command execution on the network layer based on the packet capturing achieved using the eBPF tracer developed in the context of WP4.

As aforementioned, the eBPF tracer developed and documented in the context of WP4 was reengineered in order to be able to intercept the communication between the host machine (dedicated TPM server) and the FPGA board over the established TCP/IP connection. Thus, the interception timestamps between the request and response packets that encapsulate the TPM commands sent from/to the TSS residing in the host, can give us a quite accurate performance measurement for the execution of the QR algorithms on the PFGA. It has to be stated that due to implementation limitations, the network packet interception was chosen as the approach that offers the "closest proximity" to the FPGA. Note that, these measurements include the time for establishing the TCP/IP connection and the time for transmitting the TPM command packets. This time frame can be considered negligible.

In addition, through the experimental testbed for the realisation of the INDEV.AU.3 and INDEV.AU.4 user stories, we are in position to measure the performance of the NewHope Key Exchange and the BLISS digital signature schemes, while we also measure the performance of the attestation by Quote and Attestation by Proof schemes. The aforementioned experiments are performed having the HW QR-TPM integrated to use case reference implementation.

2.2.3.1 Quantitative Metrics

Regarding the quantitative evaluation of the project, the acceptance criteria set initially in D6.1 [1]. The acceptance criteria of both the first and the second release of the demonstrator have been met in their vast majority.

Table 3, *Table 4* and *Table 5* show the time differences of the demonstrator among the different chosen standpoints. *Table 3* corresponds to the evaluation of INDEV.AU.3, and *Table 4* and 5 to the INDEV.AU.4. The entries of the FIDO U2F registration and Authentication are those measured during the first experimental period but are reported again in the tables to ease reference. 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 the following tables.

As a general statement, which applies to all commands in the realised use stories, the timings of the Application layer are greater than those from the TSS layer, and in turn, the latter are greater than those of the Networking stack. This is a justifiable evidence, as the application layer is at the highest level on the utilised operational stack. The performance between the Application and the TSS perspective is negligible. One can notice a difference of around 0.01 sec between the timings. However, the timings of the Network layer are considerably faster that the Application and TSS layers. This is justified by the fact that the Application and TSS layers incorporate also the time needed for the marshalling and unmarshalling operations. As aforementioned, the timing of the network layer can be seen as the execution timing of the TPM commands on the HW QR-TPM on the FPGA board. In cases, where a TPM command engages the execution of a QR algorithm the captured measurement can offer an approximation of the performance of the algorithm.

NewHope key creation and encryption/decryption operations: More specifically, regarding the NewHope performance, as can be seen in *Table 3*, CC_Create command is used for the creation of a key pair. The NewHope key generation (as a child object) is a quite fast process which requires, on average, 1.53 secs to complete. The CC_Create refers to the creation of a child object, which requires the creation of a primary object first, under one of the hierarchies using the CC_CreatePrimary. The latter completes in a similar time frame of around 1.53 secs on average.

After creating the NewHope key pair, the encryption and decryption operations are performed using the CC_NEWHOPE_Enc and CC_NEWHOPE_Dec, respectively. The commands have been tested in the context of INDEV.AU.3 for the encryption and decryption for the transactions' database. Both operations required 1.53 secs to complete.

Overall, the aforementioned operations of NewHope can be performed within a reasonable time frame, considering also that the HW QR-TPM is implemented on an FPGA board and the implementation used in not optimise for the ARM architecture. In addition, the FPGA includes a simple scheduler which consumes ~50% of the available CPU time. Given these facts, the performance of NewHope can be considered satisfactory.

BLISS key creation and signing operations: In both user stories INDEV.AU.3 and INDEV.AU.4, the BLISS signature scheme has been used for acting as the attestation key for attestation by quote and by proof approaches. Thus, the CC_Create command triggers the process of the key creation of BLISS. As can be seen Tables *Table 3*, *Table 4* and*Table 5*, the BLISS key creation needs considerable time to complete. More specifically, in all cases this operation converges to around 41 secs for the key creation in average of the 100 execution of each user story scenario. This notable behaviour motivated us to search in more detail its behaviour and investigate the distributional characteristics of the 100 execution results. More specifically, for the case on the Network layer measurements of *Table 3*, we have extracted the statistic shown in *Table 2*. We can infer that BLISS has a stochastic behaviour that make the time performance to deviate significantly among the collected results. This is advocated by the range (Max-Min) of 255.6 seconds, while the standard deviation of the results is 42.63 seconds.

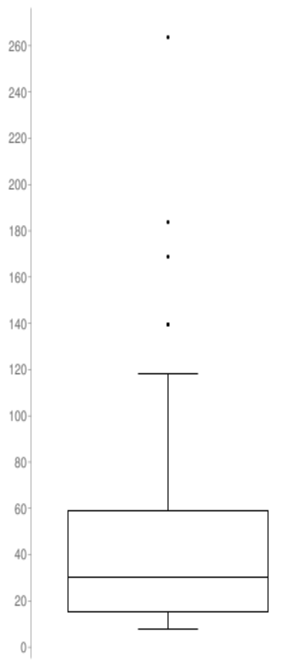


Figure 16: Boxplot of CC_Create of BLISS key pair

Figure 16 reveals the distributional characteristics of the results. The Median is placed at around 30 secs, the 1st and 2nd quartiles being rather concise, but the 3rd and 4th being rather expansive, and thus, affecting the average performance to converge approximately to 40 secs.

The BLISS implementation which was integrated in the HW QR-TPM can be found in the GALACTICS repository [21]. Given this implementation one can see that during key generation there are multiple steps, where randomness of the primitives may be rejected, and the generation process is initiated again. That is, given this implementation approach, the deviation in the performance of the BLISS key generation is justified, as the process tries to maximise the randomness and several iterations may occur to achieve this goal.

Table 2: CC_Create command statistics over 100 experimental results for the BLISS key creation

CC_Create (BLISS) statistics	
Max	263.54 seconds
Min	7.94 seconds
Range	255.6 seconds
St. Deviation	42.63 seconds
Coefficient	0.9822

The BLISS scheme is used to derive the attestation keys leveraging in the configuration integrity verification protocols. In this regard, the signing and signature verification operations are important to evaluate the overall efficiency of the attestation schemes. More specifically, the CC_Quote and CC_VerifySignature are used in the attestation by quote scheme for signing the quote and verifying its signature respectively. The CC_Quote command aims at providing a quote and signature for a given list of PCRs. Thus, the average of 2.32 secs requires both accessing the PCRs and signing their content. The signature verification is a rather nimble process and takes around 1.26 secs.

Table 4 and *Table 5* include the execution of the CC_Sign, which is used for signing the nonce ***n*** used in the attestation by proof scheme [24]. The two tables represent the sequence of TPM commands used for the realisation of same scenario of INDEV.AU.4 user story, but the one in *Table 5*, leads to a failed attestation outcome, due to a policy discrepancy. That is, the CC_Sign command in the case of the wrong policy needs ~1.024 secs, as it fails to complete the signing process due to a policy non-compliance; the BLISS key creation performed using a policy which cannot be verified. As a result, the signing process terminates in a shorter time frame. On the other hand, in the case of *Table 4*, the CC_Sign is executed normally under a valid policy matching and requires ~2.29 secs to complete. The CC_Sign performance is considered reasonable given the current implementation of the HW QR-TPM.

Table 3: Demonstrator #1 – Comparison of Timings among the App, TSS and the Network perspectives using HW QR-TPM (on FPGA board) for the realisation of user story INDEV.AU.3.

HW QR-TPM Command	Application Timings (sec)	TSS Layer (sec)	Network Layer interception using eBPFs (sec)
FIDO U2F Registration	**0.032 + 0.031 [=0.063]**		
FIDO U2F Authentication	**0.016 + 0.017 [=0.033]**		
CC_Startup	1.02253222466	1.01330438375	0.267365820408
CC_CreatePrimary	1.53199502707	1.52219373465	0.774837913513
CC_Create (newhope)	1.53315597534	1.52315644741	0.781616315842
CC_Create (bliss)	43.9633174801	43.9522656822	43.4051845002
CC_Load	1.24923972845	1.24015207529	0.615240006447
CC_NEWHOPE_Enc	1.53221192122	1.51612931967	0.763236413002
CC_PCR_Extend	1.01708666325	1.00894747257	0.258343296051

HW QR-TPM Command	Application Timings (sec)	TSS Layer (sec)	Network Layer interception using eBPFs (sec)
CC_NEWHOPE_Dec	1.53350549459	1.52393599272	0.76504529953
CC_Load	1.19869992733	1.1885679388	0.584811768532
CC_Quote	2.31629784346	2.30664509058	1.57253108263
CC_FlushContext	1.02460578918	1.01436275005	0.259500694275
CC_LoadExternal	1.02329705477	1.01380927324	0.256406946182
CC_VerifySignature	1.26514987469	1.25572157145	0.626230025291
Total	**60.21109500411**		

Table 4: Demonstrator #1 – Comparison of Timings among the App, TSS and the Network perspectives using HW QR-TPM (on FPGA board) for the realisation of user story INDEV.AU.4. with successful signature verification.

HW QR-TPM Command	Application Timings (sec)	TSS Layer (sec)	Network Layer interception using eBPFs (sec)
FIDO U2F Registration	**0.032 + 0.031 [=0.063]**		
FIDO U2F Authentication	**0.016 + 0.017 [=0.033]**		
CC_Startup	1.02175130129	1.01202268839	0.261343362331
CC_PCR_Extend	1.02312913656	1.01314315557	0.25795977354
CC_StartAuthSession	1.02397012711	1.01540932417	0.264666309357
CC_PolicyPCR	1.02413032532	1.01493059874	0.25902463913
CC_PolicyGetDigest	1.02474012852	1.01523864508	0.258535227776
CC_Startup	1.02400782108	1.0152261591	0.258282940388
CC_PCR_Extend	1.02451362133	1.01625168562	0.259160575867
CC_StartAuthSession	1.02481968164	1.01443250418	0.258640646935
CC_PolicyPCR	1.02350260735	1.0142342329	0.258288798332
CC_CreatePrimary	1.53854681015	1.52879444122	0.775353782177
CC_Create(bliss)	40.7815047693	40.7722033358	40.2426240754
CC_Load	1.23451404572	1.22659765482	0.609657390118
CC_Sign	2.2854979682	2.27603115797	1.53549443722
CC_LoadExternal	1.02415616501	1.01391834325	0.256568736281
CC_VerifySignature	1.25534365438	1.24173166243	0.601121025282
Total	**57.33412816296**		

Table 5: Demonstrator #1 – Comparison of Timings among the App, TSS and the Network perspectives using HW QR-TPM (on FPGA board) for the realisation of user story INDEV.AU.4. with failed signature verification.

HW QR-TPM Command	Application Timings (sec)	TSS Layer (sec)	Network Layer interception using eBPFs (sec)
FIDO U2F Registration	**0.032 + 0.031 [=0.063]**		
FIDO U2F Authentication	**0.016 + 0.017 [=0.033]**		
CC_Startup	1.04219357967	1.033020401	0.283529734612
CC_PCR_Extend	1.02371020317	1.01462564468	0.258074879646
CC_StartAuthSession	1.02457458973	1.01488978863	0.258891987801
CC_PolicyPCR	1.02409098148	1.01493635178	0.259133982658
CC_PolicyGetDigest	1.02450957298	1.01527831554	0.258982896805
CC_Startup	1.02513589859	1.0150737524	0.258654332161
CC_PCR_Extend	1.02335813046	1.00944423676	0.258316397667
CC_StartAuthSession	1.02413773537	1.0151296854	0.258891010284
CC_PolicyPCR	1.02583520412	1.01545004845	0.258951711655
CC_CreatePrimary	1.53579690456	1.52652206421	0.773678135872
CC_Create	41.8168566227	41.8077156544	41.3133294582
CC_Load	1.17736263275	1.16880648136	0.577876329422
CC_Sign	1.02373678684	1.02264732865	0.258264088631
Total	**54.7912988424**		

The next table gives a summary of the KPIs corresponding to the implemented reference scenarios, as identified in D6.1, and measured in the second round of experimentation. As can be seen, all of the KPIs have been achieved with a success rate close to 100%, thus, further justifying the benefits of integrating decentralized Roots-of-Trust in such heavily regulated environments as the ones met in the Fintech application domain.

Table 6: Demonstrator #1 – Quantitative Metrics by M24 and M35

Id	Metric	Target Value	Acceptance criteria	(M)andatory / (G)ood to Have / (O)ptional	Measured by M36	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 SW FutureTPM: **1027.21 ms** With HW FutureTPM: **1024.00 ms**	**Target Achieved.** The sealing performance is below the acceptance threshold. Using either the SW or the HW implementation of the FutureTPM

Id	Metric	Target Value	Acceptance criteria	(M)andatory / (G)ood to Have / (O)ptional	Measured by M36	Comments
3	Performance of the FIDO Registration	<=2 sec	<=3 sec	M	With TPM2.0: **0.063 ms** With FutureTPM: **0.063 ms**	**Target achieved.** We consider only the server-side processes for user registration, excluding network latency and user's interaction with the U2F Security Key. Target achieved.
4	Performance of the FIDO Authentication	<=1.5 sec	<=2 sec	M	With TPM2.0: **0.0038 ms** With FutureTPM: **0.0038 ms**	**Target achieved.** We consider only the server-side processes for authentication, excluding network latency and user's interaction with the U2F Security Key. Target achieved.
5	Performance of the control flow property-based **attestation** toolkit for the operational correctness	<=7 sec	<=10 sec	M	With HW FutureTPM: **Attes.By.Quote** [CC_Quote + CC_VerifySignature] = 3.59 sec **Attes.By.Proof** [CC_Sign + CC_VerifySignature] = **3.55 secs**	**Target achieved**. For this KPI we consider the time needed to perform the core attestation by Quote of Proof schemes. That is we sum the timings of CC_Quote, CC_Sign and CC_VerifySignature, as shown in the previous column. The attestation key creation is addressed by the following KPI.
6	Performance of key generation functionality within the domain of ms	<=20 ms	<=30 ms	M	With HW FutureTPM: **NewHope**: 780 ms **BLISS**: 43405 ms	The timings given here for the key creation of NewHope and BLISS represent the average of 100 executions of the corresponding command. In addition, we report here the timings captured from the network perspective as the closest point to HW TPM that gives the most accurate result.

The KPIs numbered 1, 2, 3 and 4 were tested during the 1st experimentation period. However, the 2nd KPI regarding the performance of the sealing process has been updated in order to provide a timing measurement during the 2nd experimentation period with the use of HW TPM.

The KPIs numbered 5 and 6 were planned for the 2nd experimentation period. For the 5th KPI, it must be stated that we focus on the performance of the core cryptographic operations of the attestation scheme (Quote, Proof), per se, and we do not refer to the time needed to perform the tracing and the generation of the control flow graph. If fact, this process has been evaluated in the context of WP4 and more specifically in D4.5; there the eBPF and IntelPT tracing techniques, in the context of the FutureTPM multi-level detailed monitoring techniques, have been throughly evaluated and compared for different orders of software complexity.

The performance of key generation functionality in the 6th KPI may not meet the target values set in D6.1, however we need to state that the target was rather optimistic. Considering that the HW QR-TPM is implemented on an FPGA board and the implementation used is not optimised for the ARM architecture, and its scheduler consumes ~50% of the available CPU time, the acquired performance timings of the NewHope case can be considered satisfactory. The performance of the BLISS key creation process is strictly related to its intrinsic behaviour as explained before.

2.2.3.2 Qualitative Metrics

During the 2nd experimental phase, all the mandatory qualitative metrics have been achieved. More specifically, the protection of sensitive tokens was a target which achieved during the 1st experimentation period. The successful KPIs related to the integrity and confidentiality of the history logs come as a result of the realisation of the INDEV.AU.3 user story. The user authentication through the use of TPM metric is achieved through the design and extensive analysis of the trust models given in section 2.1.3. Regarding the optional metric for the creation of a TPM-based wallet that can support TPM migration functionality, the consortium left this target as a future work that can play a crucial role in future applications.

Table 7: Demonstrator #1 – Qualitative Metrics by M24 and M35

Id	Metric	Target Value	(M)andatory / (G)ood to Have / (O)ptional	Measured by M35	Comments
1	Protection of sensitive tokens	Supported	M	With TPM2.0: Yes With FutureTPM: Yes	Successfully sealed both Bearer and Financial Tokens.
2	Confidentiality of local history logs	Supported	M	With FutureTPM: Yes, using NewHope QR scheme	Successfully performed the encryption and decryption commands of NewHope
3	Integrity of local history logs	Supported	M	With FutureTPM: Yes, using BLISS QR scheme to enable the attestation by Quote method.	Successfully performed the integrity verification of the history logs through attestation by Quote and the use of BLISS signature scheme.
4	User authentication through the use of TPM	Supported	M	Target Achieved Documentation is given in Section 2.1.3.	We approached this KPI as a research topic and we developed the required trust models in order to achieve user authentication through the use of TPM and DAA protocol. The evaluation is given in Section 2.1.3.
5	The creation of a TPM-based wallet that can support TPM migration functionality throughout the user's devices	Supported	O	Not implemented. Left for future work as a crucial function for future applications also outside of the fintech scope.	

Chapter 3 Demonstrator #2 – *Activity Tracking*

3.1 Demonstrator Overview and Final Architecture

As described in the previous deliverables, the S5Tracker demonstrator is based around the infrastructure build by S5 (Suite5 Data Intelligence Solutions Ltd), that is called S5Tracker, and targets the activity tracking and personal health data collection and analysis domain, offered as a service to healthcare institutions and professionals who need to track the physical activity as well as health vital signals of individuals that they curate.

The main usage of the S5Tracker is that 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.

In terms of technical infrastructure, the S5Tracker is a set of solutions which are the following:

- a **cloud-based analytics engine** (S5Tracker Analytics Engine) acting as a data handling information environment of personalised and interlinked data streams related to activities performed mostly by individuals, and
- various **personal applications** (called S5PersonalTracker, where each one corresponds to one and only one individual) that are used to retrieve data from wearable devices and other data sources that are residing at the individual's side and push these data to the cloud-based engine.

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, using an application call S5PersonalTracker (See below);
- 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

The following picture highlights both the S5PersonalTracker and the S5DataAnalysis parts of the infrastructure, while also the current dataflow directions are shown.

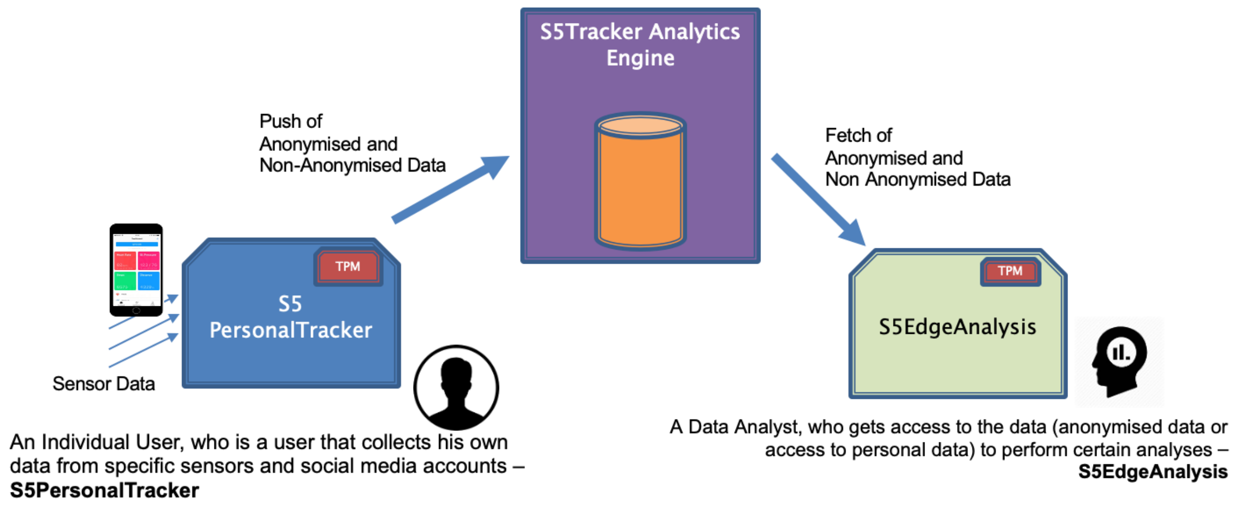

S5Tracker Analytics Engine
Push of Anonymised and Non-Anonymised Data
Fetch of Anonymised and Non Anonymised Data
TPM
S5 PersonalTracker
Sensor Data
TPM
S5EdgeAnalysis
An Individual User, who is a user that collects his own data from specific sensors and social media accounts – S5PersonalTracker
A Data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

Figure 17: Demonstrator #2 – Main Actors and Entities

As in any client-server infrastructure, which handles sensitive data, 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 level of the cloud-based infrastructure (S5Tracker Analytics Engine) as well as in the S5PersonalTracker side
- Data volume handling and scalability issues
- Data processing power and system performance optimisation over the cloud-based offering.

Out of the aforementioned pain points, the most important which has been in the focus of the FutureTPM demonstrator, is that of Data Anonymization and Privacy preservation that can be used to both secure the data and the details of each user to not be accessible from other parties accessing the platform, and also for the generation of aggregated "User Personas" which are fictional representative users, that can be globally accessible by analysts, in order to create reference cases.

During this demonstrator, the core focus lay on the how the utilisation of **Software TPM**, both at the S5PersonalTracker and at the S5DataAnalysis sides could be used to realise a highly trusted and secure environment for sharing personal data in a trusted and privacy preserving manner, that can guarantee data integrity and anonymity as well in case the latter is chosen.

The demonstrator that was 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, as depicted in the previous figure, the demonstrator has three main actors and three different components where each one of these actors operates one component.

The exact architecture of the overall infrastructure, as revised to fit the TPM modules, both in the first and in the second experimentation period is shown in the next figure. As indicated in the figure, the westbound component is that of the S5Personal Tracker, which consists of a frontend interface and in the back end the different sub-modules are integrated that are used to retrieve the data, store it locally, perform thin analytic methods and prepare the data to be sent over to the Cloud Based Engine which resides eastbound. In more detail, data is retrieved from different sensors (southbound in the figure), which do not possess TPM capabilities as they have very low computational resources. The data is cleansed, curated and homogenised, and is stored in the S5 PersonalTracker Database. There, a Monstach services is used to synchronise the data with an Elasticsearch component which is used to feed the Thin Analytics Engine, alongside with the DB.

In case data needs to be shared to the S5Tracker Analytics Engine, a choice is made in the FrontEnd by the user of whether the data will be stored as is or anonymised, and following this data goes through a Data Selector & Bundler, which is tasked to select and truncate the payload into sizes optimised for transfer.

The eastbound component is that of the S5Tracker Analytics Engine, which resides in a cloud infrastructure (or alternatively is hosted by an organisation that offers S5PersonalTrackers to its clients). A Data Check-in module is listening for incoming data streams and resolves in case a payload is received whether it should be stored in the S5Tracker Users Data Store or it should pass through the Persona Builder to be stored in the Anonymous Persona Data Store. Both of these databases are available to a Spark-based Analytics Engine, that can be used through the Frontend by Data Scientists, and the results can be saved in the S5Tracker Insights Store. Also, all databases in the Analytics Engine component are available through an Export API to serve Data Scientists with those data as needed.

The TPM comes in this picture in order to attest the S5PersonalTracker to the S5Tracker Analytics Engine. As such each S5PersonalTracker executes first a Join() command in order for its TPM to join the network (step 1 in the figure below). Then upon deciding to share the data the Commit() (Step 2) command is executed, and then they payload is signed using the Sign() command (step 4). Upon arrival to the S5Tracker Analytics Engine, the payload's signature is checked using the Verify() (step 4) command, and in case there is a failure, the payload is dropped, else It is stored either in the User Data Store, or is passed to the Persona Builder (by resolving the base name selected).

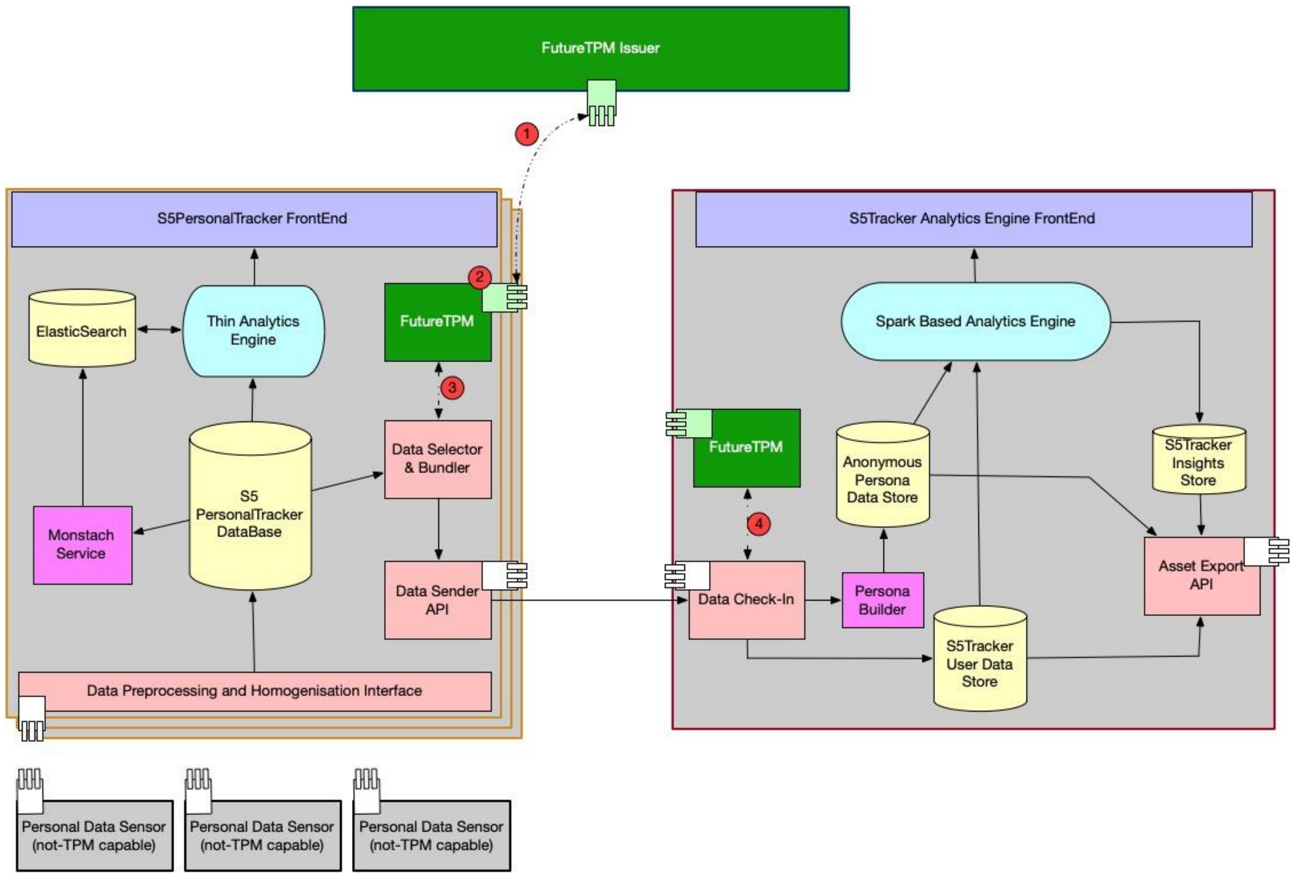
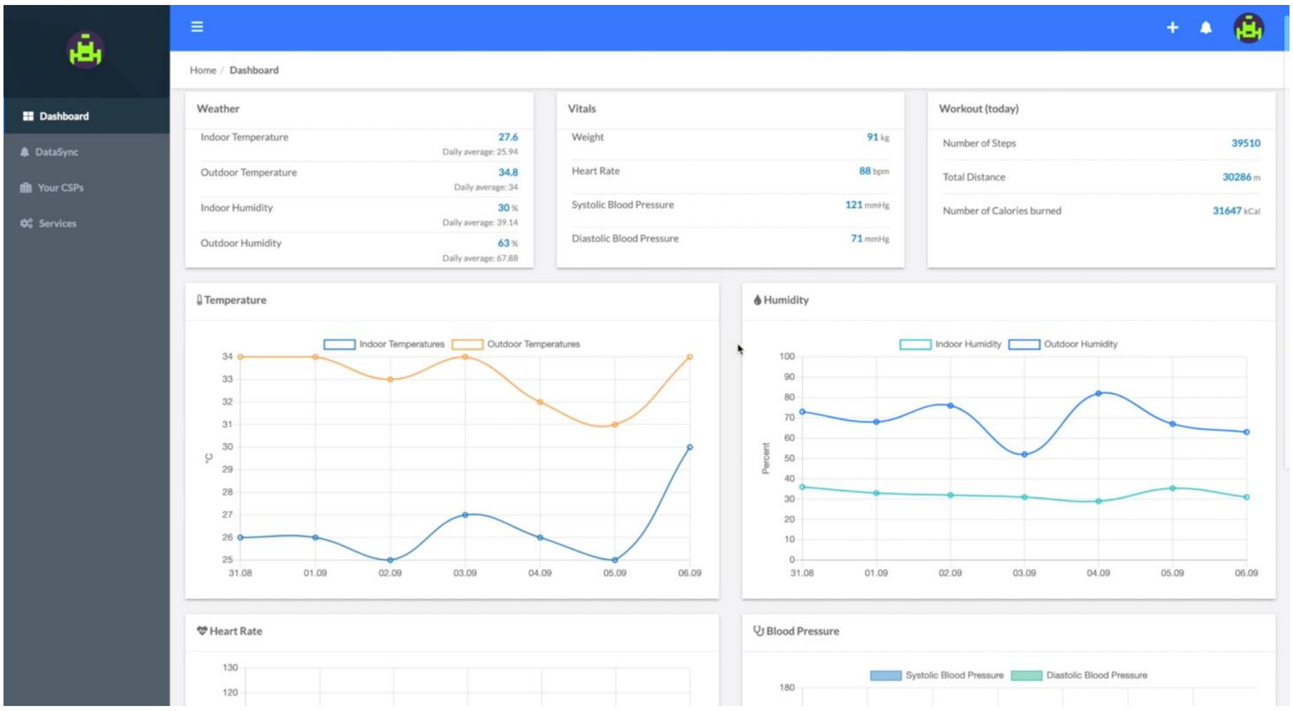

FutureTPM Issuer
1
S5PersonalTracker FrontEnd
ElasticSearch
Thin Analytics Engine
FutureTPM
2
3
S5 PersonalTracker DataBase
Data Selector & Bundler
Monstach Service
Data Sender API
Data Preprocessing and Homogenisation Interface
Personal Data Sensor (not-TPM capable)
Personal Data Sensor (not-TPM capable)
Personal Data Sensor (not-TPM capable)
S5Tracker Analytics Engine FrontEnd
Spark Based Analytics Engine
FutureTPM
4
Anonymous Persona Data Store
S5Tracker Insights Store
Asset Export API
Data Check-In
Persona Builder
S5Tracker User Data Store

Figure 18: Demonstrator #2 – Revised Architecture showing entities concerned in the demonstrator for the use cases till M36

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 S5Tracker Analytics Engine 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.



Weather

Indoor Temperature	27.6 Daily average: 25.94
Outdoor Temperature	34.8 Daily average: 34
Indoor Humidity	30 % Daily average: 39.14
Outdoor Humidity	63 % Daily average: 67.88

Vitals

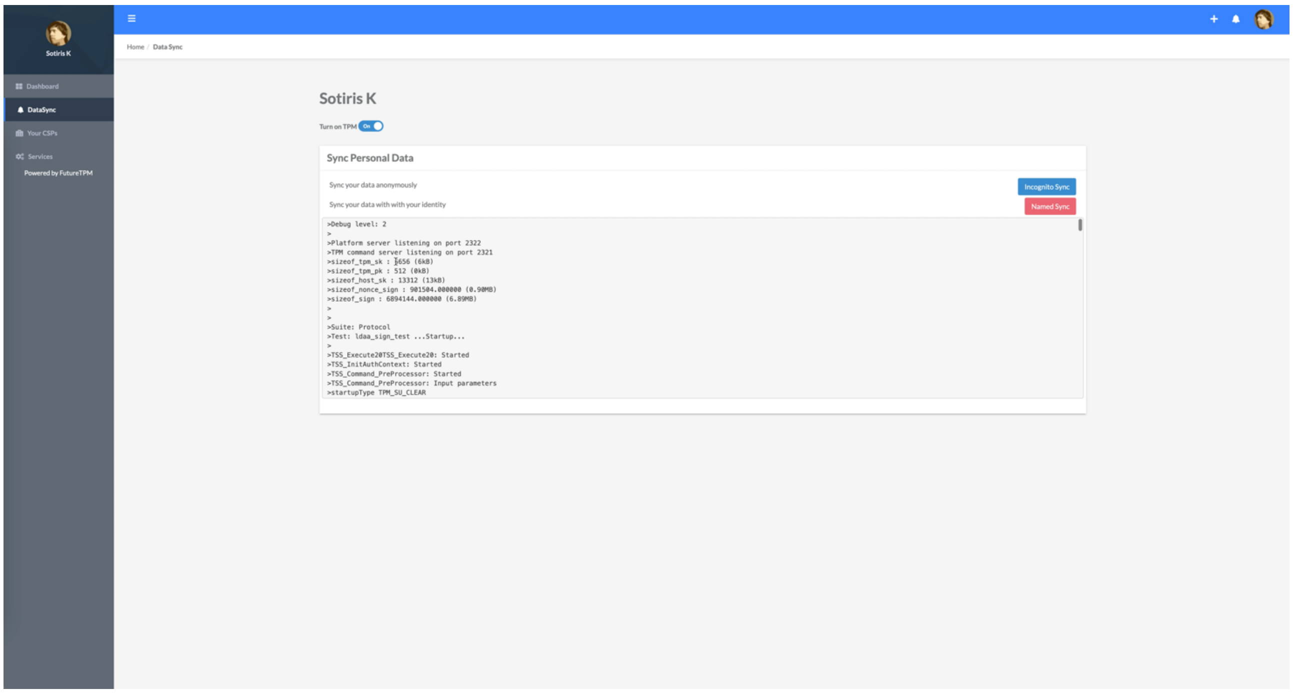
Weight	91 kg
Heart Rate	88 bpm
Systolic Blood Pressure	121 mmHg
Diastolic Blood Pressure	71 mmHg

Workout (today)

Number of Steps	39510
Total Distance	30286 m
Number of Calories burned	31647 kCal

Figure 19: Demonstrator #2 – Screenshot of the S5Personal Tracker Interface

During this second and final demonstration period, the experimentation emphasised on the DAA part between the S5PersonalTracker and the Analytics Engine infrastructure, with the focus being on the allowance of 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.



Sotiris K

Turn on TPM On

Sync Personal Data

Sync your data anonymously

Sync your data with with your identity

```
>Debug level: 2
>
>Platform server listening on port 2322
>TPM command server listening on port 2321
>sizeof_tpm_sk : 6656 (6kB)
>sizeof_tpm_pk : 512 (0kB)
>sizeof_host_sk : 13312 (13kB)
>sizeof_nonce_sign : 901504.000000 (0.90MB)
>sizeof_sign : 6894144.000000 (6.89MB)
>
>
>Suite: Protocol
>Test: ldaa_sign_test ...Startup...
>
>TSS_Execute20TSS_Execute20: Started
>TSS_InitAuthContext: Started
>TSS_Command_PreProcessor: Started
>TSS_Command_PreProcessor: Input parameters
>startupType TPM_SU_CLEAR
```

Figure 20: Demonstrator #2 – Screenshot of Sharing Selection and Execution at the S5Peronal Tracker Side

These actions have been performed using both the initial and also a complete revised version of the LDAA method of that used in the first experimentation period, to test for performance gains and the validate the overall solution as a next-generation QR infrastructure, as described below.

3.1.1 Demonstrator Needs and Challenges

The Activity Tracking demonstrator aims to incorporate TPMs into the overall ecosystem of its operations, and therefore it was essential to test that the offered trust and security guarantees could serve the purpose of providing to end-users the level of trust and anonymization they would need and of course to the platform to be able to know that the data is coming from genuine and authenticated devices which are part of the ATracker infrastructure.

In this context, during the first period, a version of the QR LDAA protocol was tested for adding the necessary features in the communication happening between the S5PersonalTrackers and the S5 Analytics Engine, to safeguard that data uploaded to the platform is genuine and comes from the authenticated endpoints.

Coming out of the first experimentation phase, two major challenges were faced. The first 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. Severe delays were experienced in the execution of the Sign() 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.

It was also necessary to implement a mechanism that truncated the payload into smaller packages, which were in total faster to sign and verify, and overcome this obstacle, and test whether this was also acceptable from a business point of view.

Nevertheless, although these challenges were dealt with at the first phase with the tweaks mentioned above, it was decided to focus during the second experimentation phase into optimising the code and the integration with the ATracker interface, to allow for the signing and transfer of larger packet sizes, as well as to test a revised version of the LDAA methods which would be developed during the project.

3.2 Implementation Path Report for the 2nd Experimentation Period

During the 2nd phase of the demonstrator within the FutureTPM project, the user stories realised had to do with optimising the overall infrastructure and ground-up rewriting certain function that led to a new deployment of the demonstrator with the aim to integrate the revised version the LDAA protocol and for allowing the signature of larger packet sizes.

As such, a heavily revised version of the ATracker infrastructure was deployed and two versions of LDAA were integrated, in order to evaluate 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).

Firstly, this allowed to test that the new data management component of the infrastructure in cooperation with the method was able to satisfy the maximum packet size envisioned for the operation of the infrastructure which is of the size of 25MB of data. This size actually refers to syncing a bulk amount of sensor data collected over a period of 1 month (the maximum timespan that the system allows for retrieving data from a S5Personaltracker) between the S5PersonalTracker and the S5 Analytics Engine for a period of 1 month.

Secondly and more important, the objective was to test a revised version of the LDAA protocol to identify if the delays witness during the first initial experimentation round could be minimized, making the overall infrastructure more robust, utilizing the same packet sizes (5kB and 25MB) and the same round of tries in each scenario. In this context, the whole process that deals with LDAA has been reconstructed for the demonstrator and therefore a new version of the demonstrator was deployed to allow the transfer of data between the collaborating entities and the according user stories have been successfully implemented.

Since the first version of the LDAA scheme was impractical in terms of execution times and memory consumption, the revised version was proposed much later as a result of an ongoing research (see [18]). Thus, the implementation of the revised scheme was a late addition to the INESC-ID tasks, which revealed some issues concerning the verification of signatures generated by the protocol. As a result, only the Join() and Commit() commands of the revised LDAA protocol was integrated into the context of the ATracker. Nonetheless, the scheme was implemented and benchmarked by INESC-ID as a standalone application. The results are summarized in subsection 3.2.2.

3.2.1 Activity Tracker Demonstrator – Experimentation with LDAA-v1 and LDAA-v2

3.2.1.1 Emulated System Description

For reasons of reproducibility, all the tests that have been performed in this second experimentation phase utilised the same hardware infrastructure as well as the same virtualized environment.

In more detail, the following hardware and OS configuration was used

- CPU: Intel i7-6700 CPU @4.00 GHz
- Memory: 16 GB of DDR4
- Host OS; Ubuntu 18.04 in the host
- VM OS: Fedora 30
- Hypervisor: KVM

To expose the virtual TPM in the VM, the packets 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 Remote Attestation Server and the server endpoint of the TLS connection have been installed in the host.

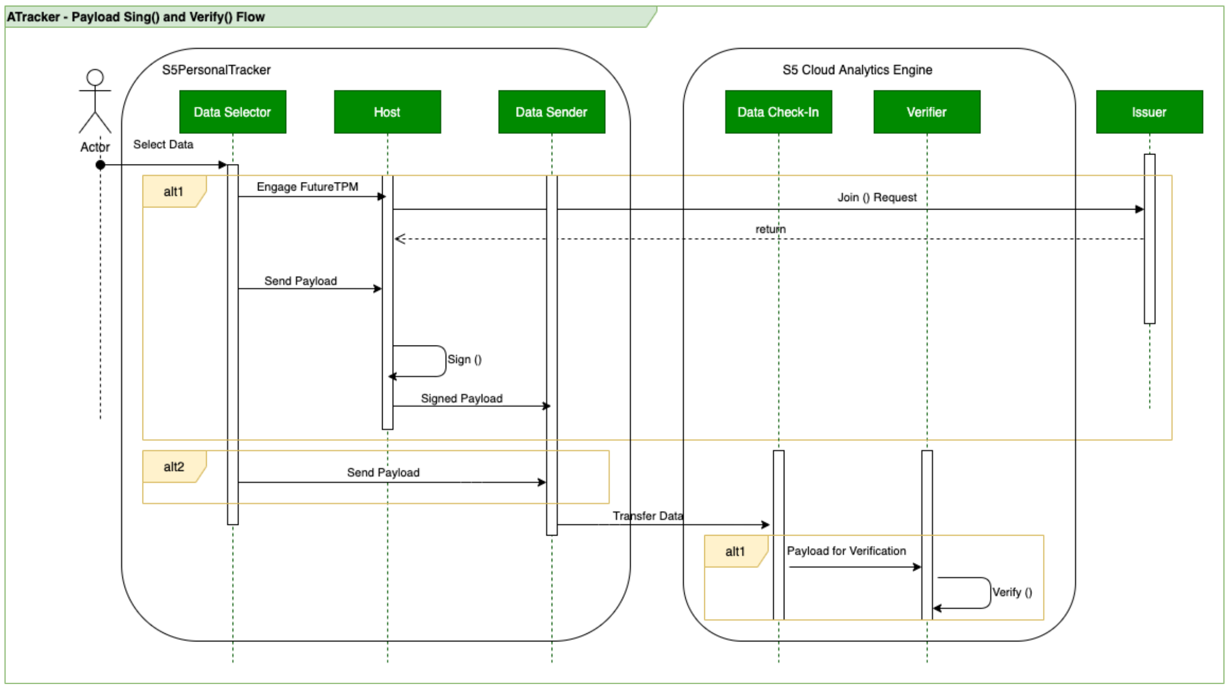
The tests results have been obtained by running 100 times the binaries that implement the four main functionality of the demonstrator (AK creation, TLS key creation, TLS connection, and TPM quote), by collecting the results and by calculating both the non-weighted and the weighted (LWMA) average, using 2 different packet sizes, namely 5kB and 25MB.

3.2.1.2 User Stories Realisation

During this experimentation round, as the focus was on re-evaluating the LDAA method with the new functionalities, the following user stories were executed.

Those are provided in the following tables.

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, load them and then select the payload to **Sign()**. The signed packets (as coming out of the S5PersonalTracker database) were sent to the S5Tracker Analytics Engine, which performed the **Verify()** command to check the signature and

Description
either store the payload in the bucket of the designated user, or drop it.  ATracker - Payload Sing() and Verify() Flow S5PersonalTracker S5 Cloud Analytics Engine Data Selector Host Data Sender Data Check-In Verifier Issuer Actor Select Data alt1 Engage FutureTPM Join () Request return Send Payload Sign () Signed Payload alt2 Send Payload Transfer Data alt1 Payload for Verification Verify () **Figure 21:** S5.IU.1 sequence diagram
Issues Encountered: None
Status: Completed
Degree of Realisation: Full
Comments (if any): User story successfully completed using the new LDAA method. In contrast to initial experimentation with the LDAA algorithm (hereafter called LDAA v1), where timeouts were encountered resulting in messages not able to be signed, this version has not shown signs of this problem, while the revised data management framework used allowed also the LDAA v1 method to be executed successfully as well. As such, the whole payload was able to be loaded and signed (the reference was a payload of 25MB in size, which is more than the actual payload that is transferred daily from a single S5PersonalTracker application to the cloud-based infrastructure).

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 base name 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. The sequence diagram is same as in the S5.IU.1 user story.
Issues Encountered: The Verify() and the Sign() command were not integrated to the S5 Activity

Description
Tracker infrastructure, while they were implemented and tested in a standalone environment.
Status: Completed (Party in the Integrated Demonstrator (till the Join() phase), Fully in the standalone environment
Degree of Realisation: Full
Comments (if any): In contrast to LDAA v1, where timeouts were encountered resulting in messages not able to be signed, this version has not shown signs of this problem. As such, the whole payload was able to be loaded and verified (the reference was a payload of 25MB in size, which is more than the actual payload that is transferred daily from a single S5PersonalTracker application to the cloud-based infrastructure).

3.2.1.3 KPIs Measured

During the second phase of the operation of the demonstrator, a direct comparison with the KPIs of the first period was south after, which has to do with the establishment of the LDAA scheme between the S5PersonalTracker interface and the Analytics Engine. For the needs of this demonstrator a simulated environment with synthetic data (identical of the first demonstration period) has been deployed and these data were sent 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 first version of the FutureTPM LDAA implementation, the experiment has been conducted by employing its "weak" state, as this has allowed to retrieve the fastest possible responses from the TPM and in a timely manner, as higher degrees of security are heavy-load operations which take quite longer execution times in a mainstream computing environment, thus making the overall service unresponsive in terms of business operations.

The revised LDAA version was only implemented for the highest security parameters suggested in [18]. At first, the initial prototype was intended to validate the reductions in memory sizes, thus the choice for the highest bound.

As during the first phase larger than expected performance drop has been noticed when compared with the TPM2.0 DAA protocol, it was decided to repeat the same measurement but this time taking into consideration two different payload sizes (5kB and 25MB) and for each payload the method was executed 100 times in order to measure the average timings of the different commands. The rationale behind this approach comes from the fact that in the Activity Tracking demonstrator, there is no need to transfer data in real time (for many reasons and one of those being also battery life in the S5PersonalTracker devices), and we would like to test whether, from a time delay perspective, it would make more sense to bundle more payload together and send it over, rather than sending smaller payload chunks more frequently.

3.2.1.3.1 Quantitative Metrics

In terms of the quantitative evaluation of the project, the acceptance criteria set initially in D6.1 for the scenarios of the second phase of the demonstrator (M36) have been met in their majority.

The next tables summarise the timings of the SW implementation of TPM commands for this demonstrator at the current version released in M35 of the project.

The first table starts with presenting the timings of the sequence of commands (grouped by the major 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, for signing and verifying a packet of 25MB of data. Having as a reference point these timings with a current TPM2.0 implementation, the same amount of payload has been used to perform the equivalent

LDAA 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.

Table 8: Comparison of DAA/LDAA Timings at Application level

	TPM2.0 DAA Timings	FutureTPM LDAA-v1	FutureTPM LDAA-v2
Join () and Commit ()	1,190 sec	1,936 sec	10,754 sec
Sign ()	1,116 sec	58,986 sec	1,585 sec [1]
Verify ()	0,382 sec	2,784 sec	n/a [2]

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.

Seeing the table above and the figures that follow, it is noted that LDAA-v1 runs faster in the Join() phase, but significantly lags when it comes to the Sign() phase. The reason for this is that the LDAA-v1 results were obtained by using the -LDAA1 flag, aka “weak” parameter of the software QR-TPM implementation which selects the weakest security parameters (q = 3329 (12 bits); cyclotomic polynomial of 256; k = 3; etc). This was done like this in order to forego security in favour of performance due to the inefficiency of the implemented LDAA-v1 algorithm which resulted in the system halting when trying to use stronger parameters, or in the best-case scenario having the performance being significantly reduced.

In contrast, the measured performance of LDAA-v2 in the Join() phase was lasting longer, yet within acceptable limits, considering also the significant performance gains expected (see [1]) during the Sign() phase.

In more detail, as the figures showcase, using the LDAA-v1 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 than the current TPM2.0 implementation. The other noticeable delays concerned the Verify() protocol, however as the time required for this operation is lower than 3 seconds, they are acceptable from the business point of view for the current demonstrator.

[1] Actual figures were not measured in the integrated demonstrator testbed as the Sing() command was not ready to be integrated. The numbers provided here are an extrapolation of the findings of Section 4.2.2 of the standalone environment where 150x performance gains during the Sign() operation are noticed. These numbers are placed here to conduct a “qualitative” comparison” and are calculated by multiplying the LDAA-v1 figures with 150, always referring to a scenario with -weak parameters. Using the LDAA-v2 with the strongest security parameters results in performance gains of 33% at TSS level (38seconds compared to 58 seconds)

[2] The Verify() command were not part of the integration in the demonstrator’s testbed.

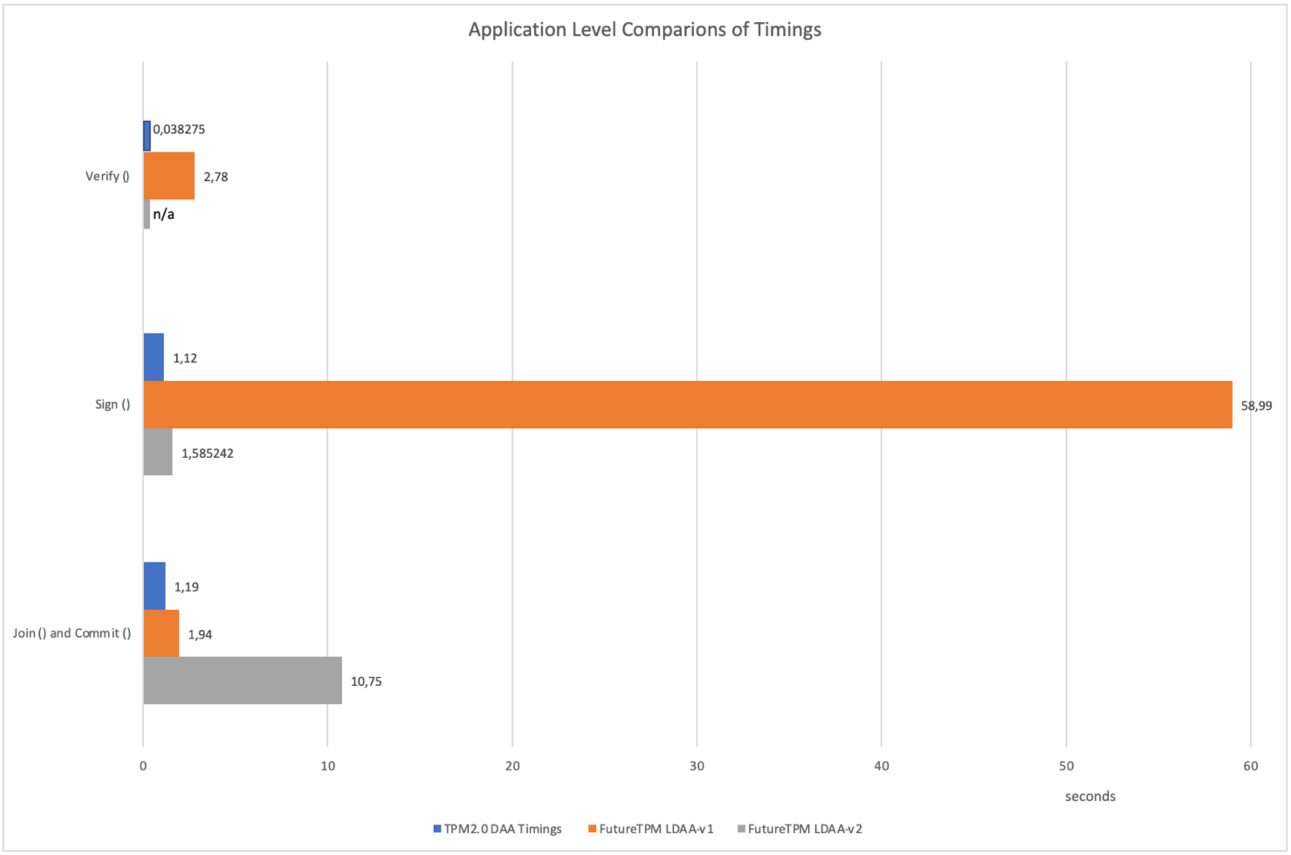


Figure 22: Comparison Graphs for different timings at application level

These delays indicated the need to work on a revised version of the LDAA protocol which was meant to achieve better performance, and that was achieved with the provision of the LDAA-v2 protocol. The core reasons for this decision was that the delay imposed during the Sign() process could lead to timeouts in the S5PersonalTracker devices, and thus deem the overall usage of TPM not feasible. After revising the TPM and delivering LDAA v2, the expected timings were significantly improved. However, as there was an error in the verification method, and it was not possible to extract any accurate result for the performance during the Sign() and Verify() steps, these stages have not been tested as part of the integration, as it would not make sense to verify a signature that would eventually fail. Nevertheless, the performance improvements of these steps have been tested in a standalone environment and are presented below, as part of subsection 3.2.2, while extrapolated figures for the Sign() method are provided as part of the demonstrator's measurements, by applying the in the lab measured gain of 150x times on the LDAA-v1 timings.

Again looking at the numbers in the following tables and figures which compare LDAA-v1 to LDAA-v2, it is obvious that the LDAA-v2 reached its goal of delivering a much faster implementation, as apart from taking a considerable amount of time during the Join() phase due to the higher security guarantees selected, results in the Sign() phase are quite close to those of the TPM.20 implementation

Apart from the measurements at the application level, measurements at the TSS were also performed, as those allowed to have a better and more objective representation of the protocol's performance.

Below we provide the average timing figures of these operations at TSS level for packets of 5kiloytes and of 25Megabytes, which were executed 100 times each.

Table 9: Demonstrator #2 –Timings at TSS Level using the FutureTPM QR Implementation (SW) of LDAA-v1

QR FutureTPM Timings LDAA – 5kB file size vs 25MB file		
	LDAA v1 – 5kB	LDAA v1 – 25MB
Initialise and Join ()	**741,246 ms**	**736,321 ms**
init_issuer_ldaa	5,425 ms	5,442 ms
init_host	0,005 ms	0,003 ms
startup	134,624 ms	133,338 ms
create_primary	134,130 ms	133,312 ms
create	151,538 ms	152,154 ms
load	179,377 ms	176,436 ms
ldaa_join	136,144 ms	135,633 ms
Sign ()	**58.262,955 ms**	**57.786,741 ms**
ldaa_sign	133,751 ms	132,366 ms
join	148,566 ms	151,570 ms
ldaa_commit_token_link	142,748 ms	142,615 ms
LDAA Sign Commit (multiples)	27706,912 ms	27231,861 ms
host_sign	772,777 ms	739,210 ms
host_generate_challenge	262,261 ms	767,719 ms
ldaa_sign_proof	623,742 ms	672,170 ms
sign_merge	765,283 ms	717,864 ms
Verify ()	**1.322,778 ms**	**1.584,747 ms**
Total	**60.326,979 ms**	**60.107,809 ms**

As depicted in the above placed table and in the following figure, there seems to not be a considerable difference between the total time it takes to process smaller or larger files, in contrast with the first experimentation period where it was impossible to sign larger files. As it is logical, the 25MB files takes a bit longer, in the Sign() and in the Verify() phase, however these differences are not noticeable at the user level.

Furthermore, as identified in the previous deliverable, 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 quite large, it would take longer to transfer it to the TPM than to regenerate it, so for the experiments to become a reality it was decided to regenerate the matrix using a pre-determined seed, which slows down the processing immensely as every time a call to a sign command is made, this matrix needs to be regenerated. The reason behind not using a cache is because the TPM doesn't possess any cache and in the current software implementation there was a need to simulate the conditions as close as to those of a physical device.

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.

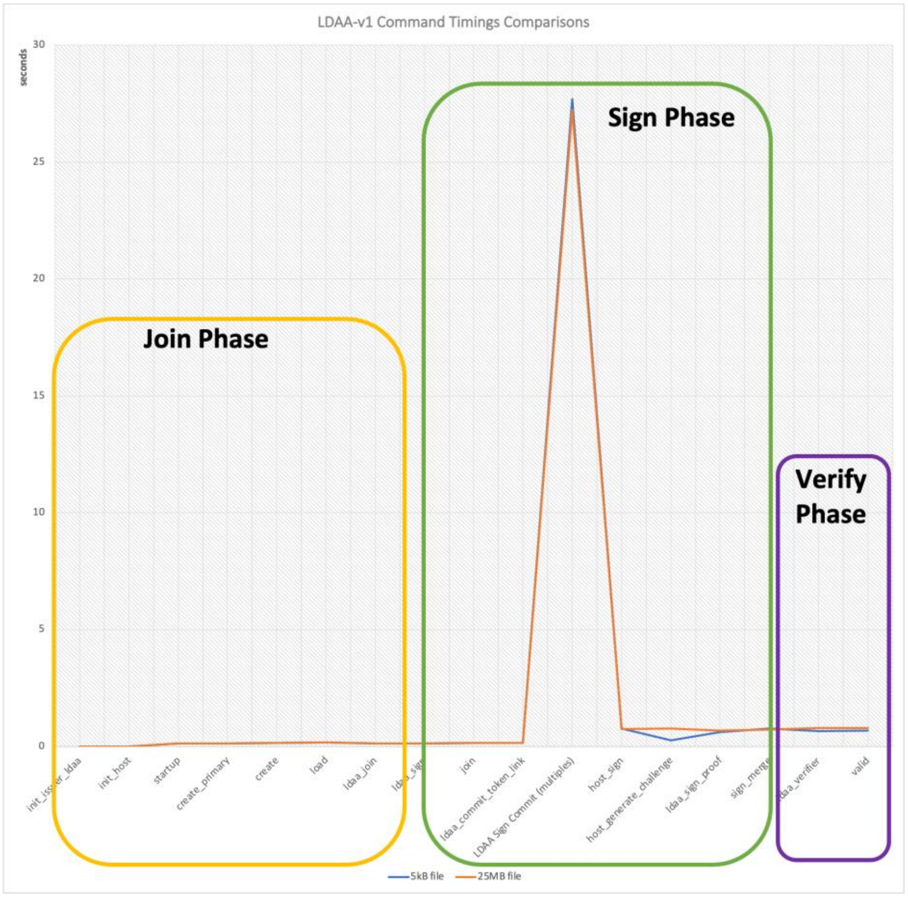


Figure 23: Comparison Graphs for different timings for 5kB and 25MB files

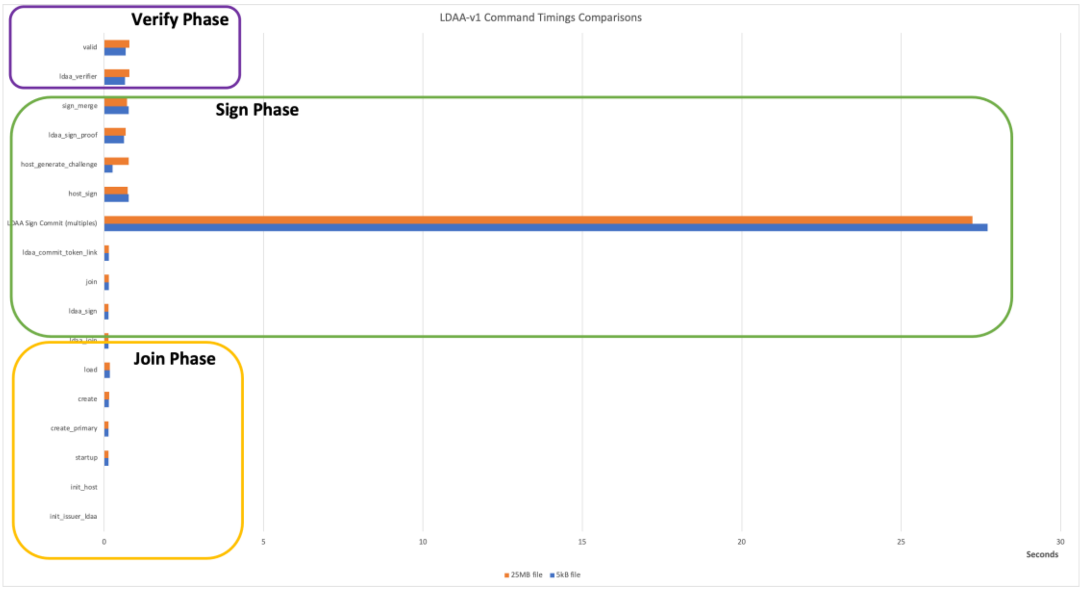


Figure 24: Comparison Graph 2 for different timings for 5kb and 25Mb files

The following table presents the timings of the experimentation with the LDAA v2 protocol at the TSS level, considering only the Initialise() and Join() commands.

Table 10: Demonstrator #2 – Timings at TSS Level using the FutureTPM QR Implementation (SW) of LDAA-v2

QR FutureTPM Timings LDAA v2 – 5kB file size vs 25MB file		
	LDAA v2 – 5kB	LDAA v2 – 25MB
Initialise and Join ()	**9.135,296 ms**	**9.554,546 ms**
init_issuer_ldaa	8425,015 ms	8831,208 ms
init_host	0,001 ms	0,002 ms
startup	145,337	151,624 ms
create_primary	142,781 ms	147,222 ms
create	140,297 ms	142,205 ms
load	142,208 ms	140,948 ms
ldaa_join	139,654 ms	141,334 ms

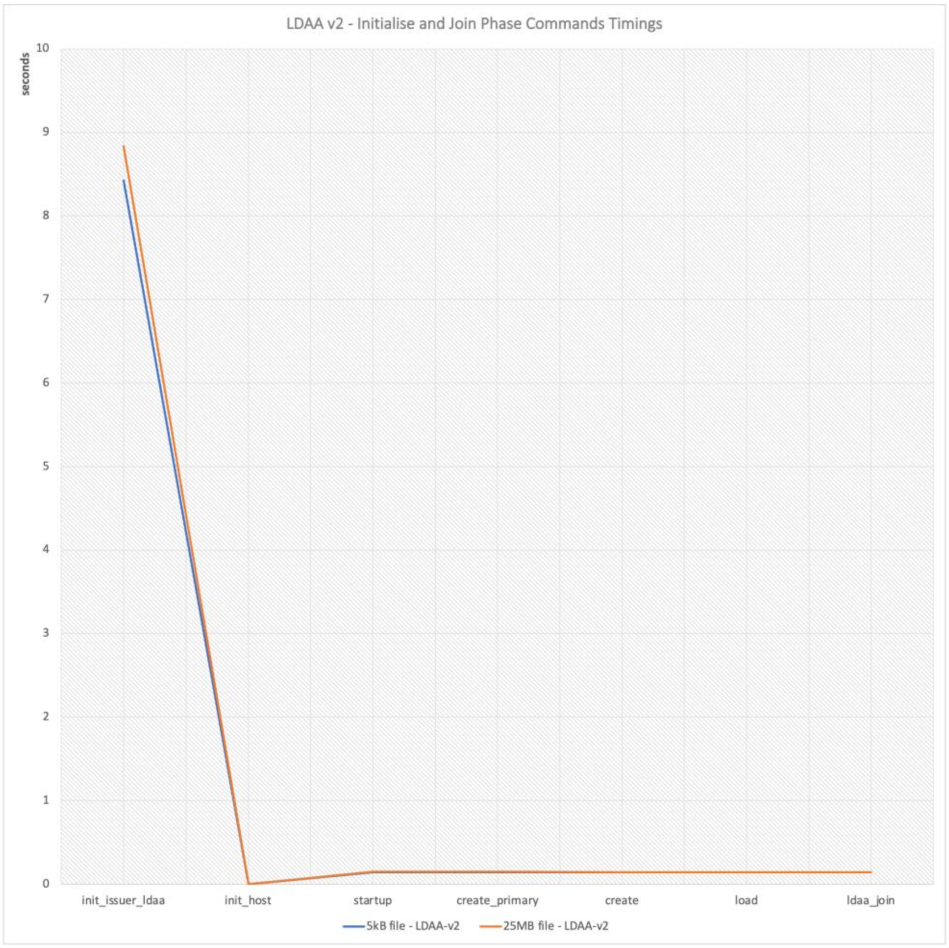


Figure 25: Comparison Graphs for different timings for 5kB and 25MB files

As the above-mentioned figure displays, the new LDAA implementation (LDAA-v2) aimed at improving the overall timings and therefore to increase the overall performance of the LDAA methods. Based on the timings recorded the new LDAA protocol (LDAA v2) it seems to be x11 times slower during the Join phase, and as the standalone results have shown (see section 3.2.2), this phase runs slower than this of the LDAA-v1. However, the main interest was in the Sign phase where the major delays have been recorded using LDAA-v1, and even worse with weak security parameters, therefore it was essential to have a better performance at that stage.

The next table provides a comparison of the timings of LDAA-v1 and the LDAA-v2 protocols in the Join and Sign phases, by using the actual measurements of the testbed during the Join phase of both LDAAv-1 and LDAAv2 as well as of the Sign phase of LDAA-v1, and extrapolated numbers for the LDAA-v2 regarding the Sign phase, by applying a multiplicator of 1/150 to LDAA-v1 timings, as it was found that during that phase there is a x150 faster execution.

Table 11: Demonstrator #2 – Timings at TSS Level using the FutureTPM QR Implementation (SW) of LDAA-v1 versus LDAA-v2

QR FutureTPM Timings – TSS Level Comparison LDAA-v1 vs LDAA-v2 (Average Numbers 5kB file)				
	LDAA-v1 (A)	**LDAA-v2 (B)**	**% Difference between B and A**	**Time Difference (B-A)**
Initialise and Join ()	**741,246 ms**	**9.135,296 ms**	**1132,42%**	**8.394,050 ms**
init_issuer_ldaa	5,426 ms	8.425,015 ms	155.184,65%	8.419,590 ms
init_host	0,005 ms	0,002 ms	-64,00%	-0,003 ms
startup	134,624 ms	145,337 ms	7,96%	10,713 ms
create_primary	134,131 ms	142,782 ms	6,45%	8,651 ms
create	151,539 ms	140,297 ms	-7,42%	-11,241 ms
load	179,377 ms	142,208 ms	-20,72%	-37,169 ms
ldaa_join	136,144 ms	139,655 ms	2,58%	3,511 ms
Sign ()[3]	**58.262,955 ms**	**388,420 ms**	**-99,99%**	**-57.874,535 ms**
ldaa_sign	133,752 ms	-	-	-
join	148,566 ms	-	-	-
LDAA Sign Commit (multiples)	27.706,913 ms	-	-	-
host_sign	772,777 ms	-	-	-
host_generate_challenge	262,261 ms	-	-	-
ldaa_sign_proof	623,743 ms	-	-	-
sign_merge	765,283 ms	-	-	-
Total Time	**59.004,201 ms**	**9.525,598 ms**	**-83,86%**	**-49.478,603 ms**

[3] Results for the LDAA-v2 Sign() command are extrapolated based on the findings of section 3.2.2

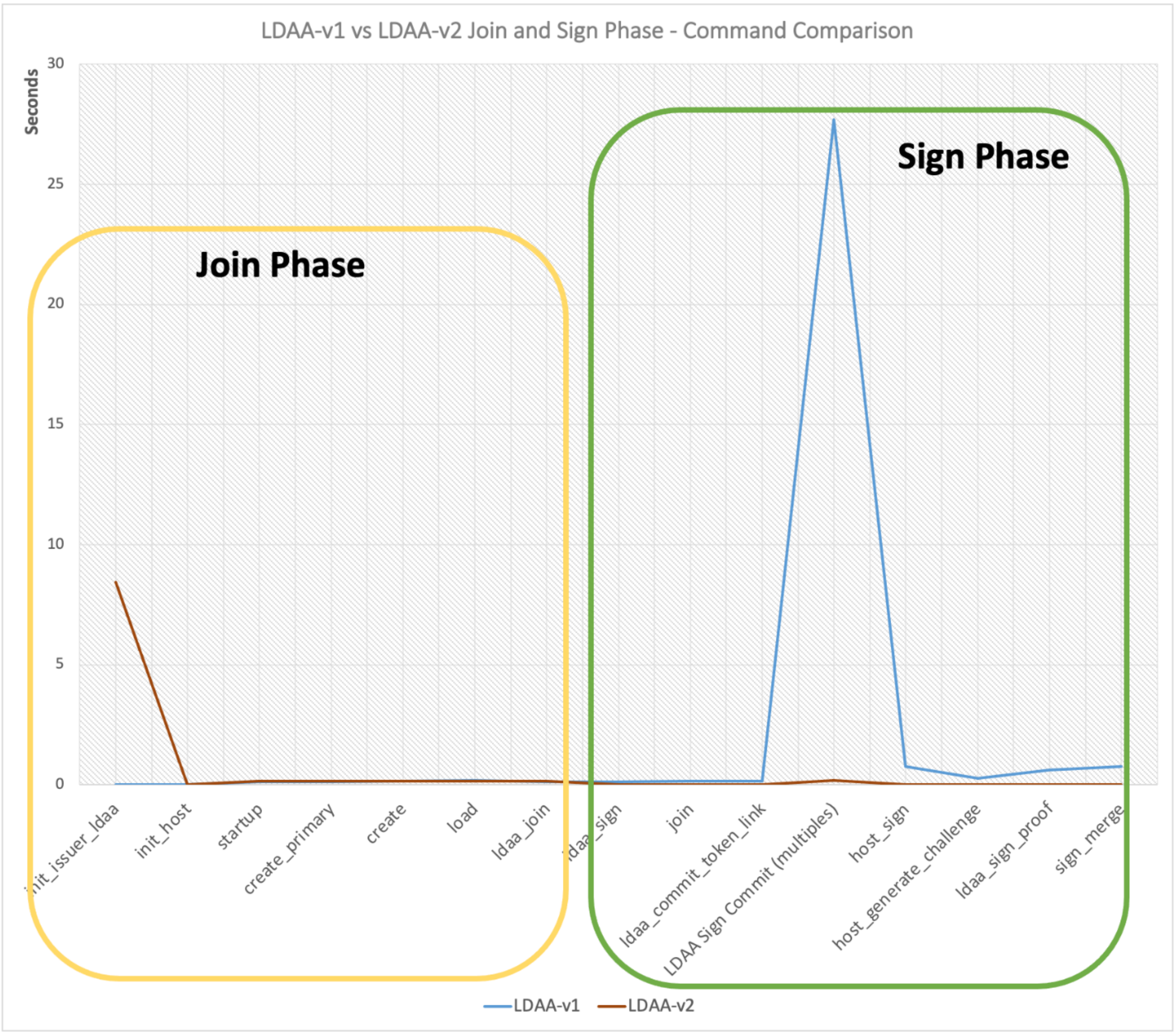


Figure 26: Comparison Graph for different timings for 5kb LDAA-v1 vs LDAA-v2

In the figure below we present the comparison of both versions of the protocols during these two phases in a 100% stacked bar diagram

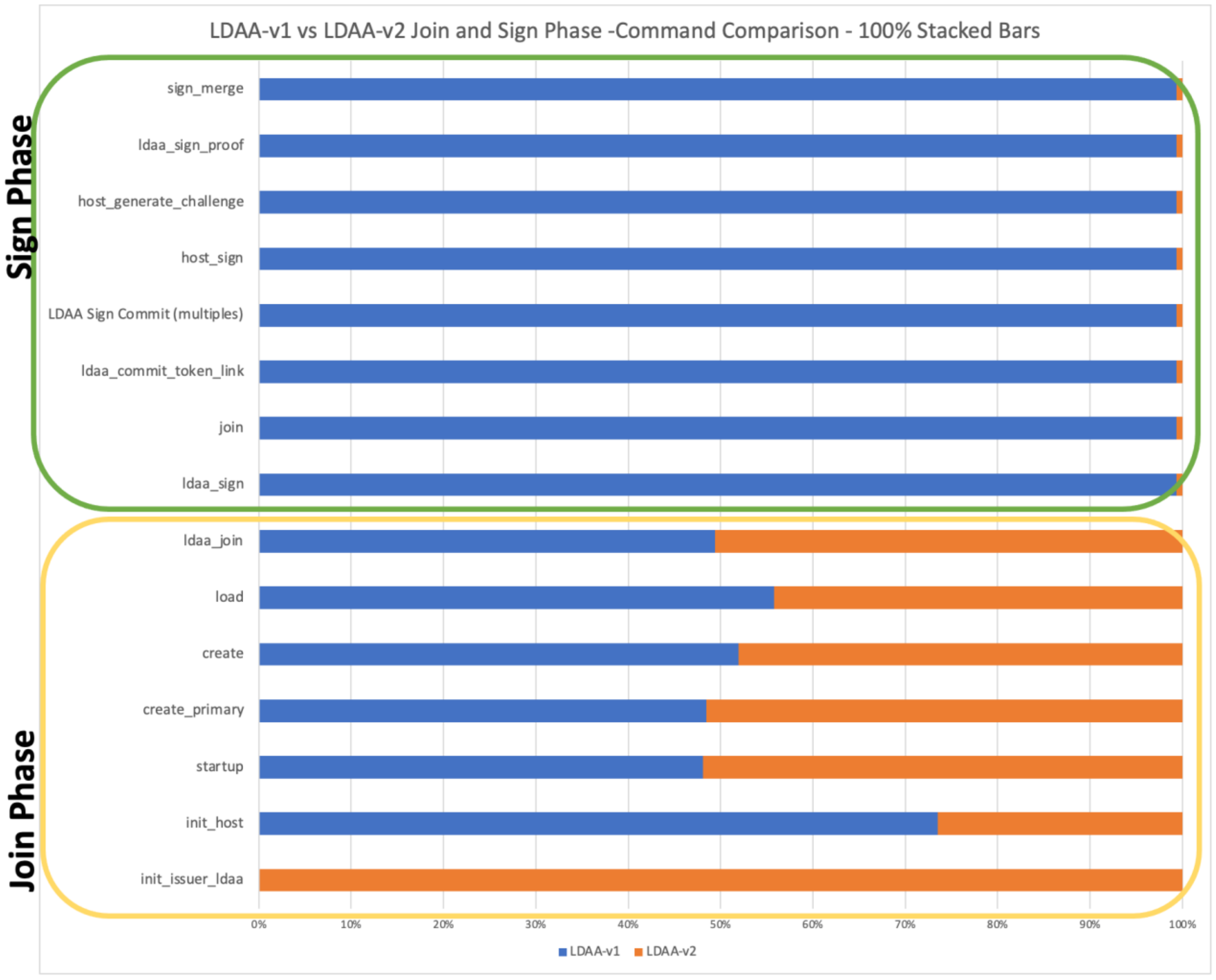


Figure 27: Comparison Graphs for different timings for 5kb and 25kb files for the Initialise and Join phase

As the above-mentioned figure shows, the new LDAA implementation during the second period aimed at improving these timings and increasing the overall performance of the LDAA methods.

Based on the timings recorded, and as seen in the next figure, the significant reduction in the Sign phase **allowed in the business context of the Activity Tracker demonstrator to lower the overall execution time from 59 seconds to 9,18 seconds**, resulting a the **LDAA-v2 protocol being 83,36%** faster (so **6x times faster**) at the side of the client, when packets need to be signed using the same security parameters.

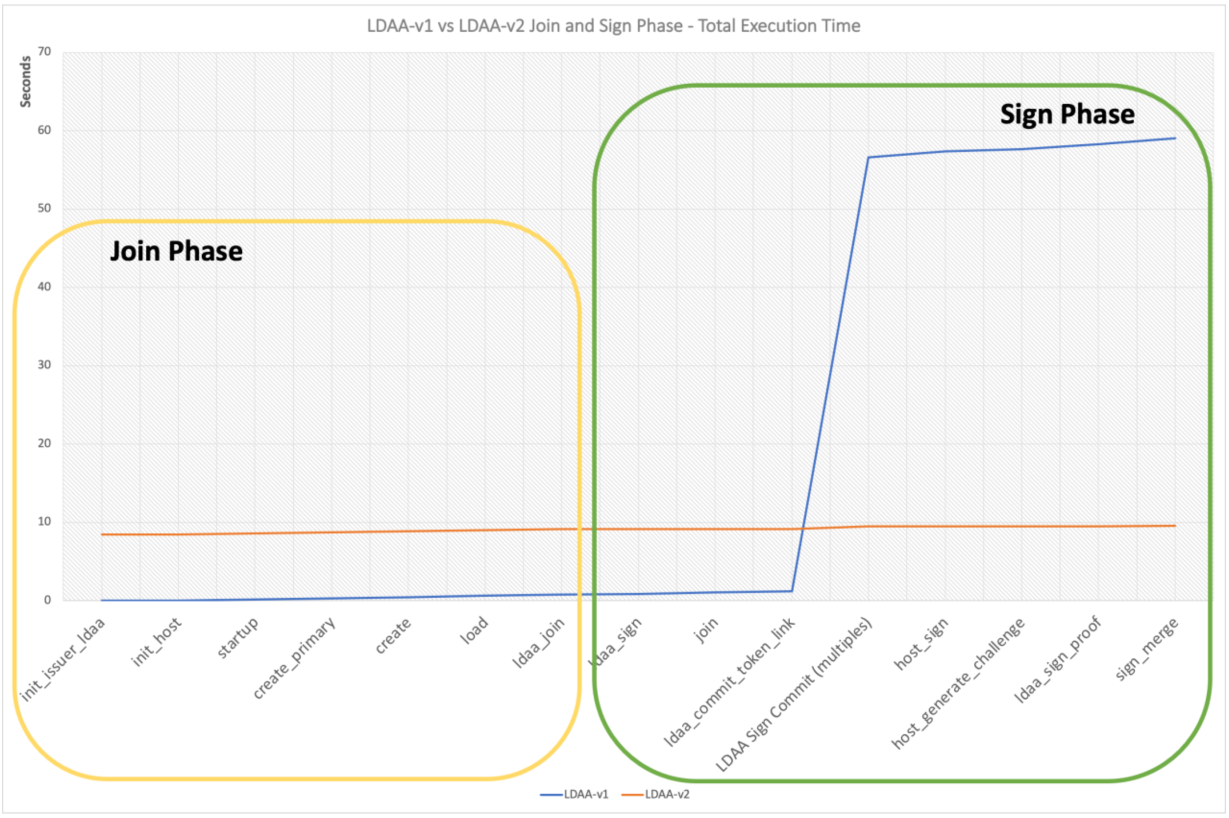


Figure 28: Total Execution Time for the Join and Sign phase at the TSS level

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

Table 12: Demonstrator #2 – Quantitative Metrics by M36

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% With FutureTPM (LDDA v1): 100% With FutureTPM (LDDA v2): 100%[4]	Target Achieved. Packets that have not be signed, are automatically dropped
2	Performance evaluation of process of sending for analyses an average set of 5kB of daily collected	-35%	-45%	M	With TPM2.0: 1,5 seconds With FutureTPM (LDDA v1): 61,40 seconds With FutureTPM	Target not achieved, however using the LDAA-v2 the timings can be accepted from a business point of view, when

[4] Based on the evaluation condiucted in the lab environment.

Id	Metric	Target Value	Acceptance criteria	(M)andatory / (G)ood to Have / (O)ptional	Measured by M24	Comments
	personal data at application leve				(LDDA v2): 10,07 seconds	transport Is performed on a schedule manner
3	Performance evaluation of the infrastructure during the Join() phase at application level	800 ms	2.000 ms	G	With TPM2.0: 1,190250 seconds With FutureTPM (LDAA v1): 1,94 seconds With FutureTPM (LDAA v2): 10,33 seconds	Target not achieved but within the acceptable space
4	Improved perception of Individual Users' trust to S5PersonalTracker as a data hub[5]	100%	60%	G	With TPM2.0: 100% With FutureTPM LDAA v1: 90% With FutureTPM LDAA v2: 95%	Target not achieved but highly acceptable 1 out of the 20 users gave anegative evaluation due to the delay experienced, which impacted negatively his perception of trust.

3.2.1.3.2 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.

Table 13: Demonstrator #2 – Qualitative Metrics by M36

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 With FutureTPM LDAA v1: Yes With FutureTPM LDAA v2: Yes	DAA support has been successfully implemented

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

3.2.2 LDAA-v2 Experimentation as a Standalone Application

The LDAA-V2 protocol was quantitively evaluated as a standalone application considering the highest security settings suggested in [18] (i.e., q = 1180591620683051565059 (70 bits); cyclotomic polynomial of 4096; k = 60).

All the measured times result from taking the median over one hundred runs on an Intel Xeon Gold 6140 running at fixed 2.3 GHz. The LDAA-v1 considers medium security parameters (i.e., q = 8380417 (23 bits) and cyclotomic polynomial of 512).

3.2.2.1 KPIs Measured

As can be seen from the following figure the LDAA-v2 improves the execution time for the Sign command significantly.

However, key creation through the Join() command is not as fast as before, which can be attributed to the inefficient ring of the LDAA-v2. Since the modulus q has to be congruent to 3 mod 8, in the new scheme, the Number Theoretic Transform (NTT) is not easily applied. In consequence, we chose the Karatsuba Algorithm for polynomial multiplication, which is significantly slower than applying the NTT.

It should also be noted that the degree of the polynomial ring is greater in the LDAA-v2, which plays a significant role as well. In summary, despite the challenges in the implementation, the LDAA-v2 is around 150 times faster for the Sign(), while being approximately 10 times slower in the join phase.

Table 14: Execution timings for LDAA-v2 in comparison with comparison with LDAA-v1.

QR FutureTPM Timings LDAA			
TPM Timings – LDAA v1		TPM Timings – LDAA v2	
Create()	**Timings (miliseconds)**	**Create()**	**Timings (miliseconds)**
	374		335
Join()	**Timings (miliseconds)**	**Join()**	**Timings (miliseconds)**
	179		1.756
Sign()	**Timings (miliseconds)**	**Sign()**	**Timings (miliseconds)**
	$7{,}2 \times 10^6$		38.336

In terms of keys and signature sizes the results presented in Figure 15 show the LDAA-v2 with bigger public key and private keys, while signature sizes are reduced by 480 times. It should be noted that this considers an upper bound for the public and private keys, which could be compressed further in future works.

Table 15: Public Key, Private Key and Sigature Sizes for LDAA-v2 in comparison with comparison with LDAA-v1

QR FutureTPM Public Key, Private Key and Sigature Sizes LDAA			
TPM Memory – LDAA v1		TPM Memory – LDAA v2	
Public Key	Sizes (kB)	Join()	Sizes (kB)
	25		32,8
Private Key	Sizes (kB)	Sign	Sizes (kB)
	24		65,7
Signature	Sizes (kB)	Sign	Sizes (kB)
	624.000		1.300

The required memory on the TPM for executing the LDAA-v1 and LDAA-v2 is reported in Figure 16. In terms of persistent memory, a reduction around 22 times is observed. For versatile memory the reduction is greater, with almost 400 times less memory begin needed by the TPM.

The reduction in TCP IO buffers follows the reduction in signature sizes. Thus, in the LDAA-v2 the buffers can be 80 times smaller than before.

Table 16: Maximum memory requirements for LDAA-v2 in comparison with comparison with LDAA-v1

QR FutureTPM Memory Consumption LDAA			
TPM Memory – LDAA v1		TPM Memory – LDAA v2	
Persistent Memory	Memory in kB (max)	Join()	Memory in kB (max)
	35.000		1.600
Versatile Memory	Memory in kB (max)	Sign	Memory in kB (max)
	512.000		1.300
TCP I/O buffers	Memory in kB (max)	Sign	Memory in kB (max)
	128.000		1.600

In summary, the early results obtained for the LDAA-v2 as a standalone software already shows a step towards more efficient and practical TPM with support for LDAA schemes.

It should be noted, however, that development still in an early stage where several improvements are possible. Future works should focus mainly on fixing the problems related to the Verify() command, as well as improving polynomial multiplication and reducing keys sizes.

3.3 Use Case Evaluation

During the FutureTPM project and the phase of the demonstrator, the LDAA protocol based on a software QR-TPM infrastructure has been successfully implemented in the S5 Activity Tracking demonstrator. As anticipated, due to the increased complexity and resource heavy operations (in terms of QR-computations), performance issues (in terms of delays) have been noticed, which were expected and are inherited by the nature and the overall architecture of the TPM and of course by the resources needed to work with QR algorithms and the associated security schemes.

In any case, and when looking at the records presented above, from a business perspective the integration of the QR-TPM methods in the infrastructure is in a position to provide <u>acceptable results in an operational environment, even if the measured performance is not meeting the ideal targets set.</u> The main reason for this, is the pivotal move in the design of the overall infrastructure which since the last year is not focused on the delivery of real-time data, but on the provisioning of data in bulk and scheduled loads.

As identified when experimenting with the LDAA-v1, the application of the LDAA protocol in its first version, was creating a local burden and delays in the different peers, thus affecting the overall system of the S5 Activity Tracker, and eventually having a negative impact on its performance which had to be mitigated by having all data transfers happening based on an overnight scheduled programme. However, it needs to be noted that the way to work with LDAA-v1 dictated the use of weaker security guarantees to have the packets signed within acceptable, yet quite long-time limits. As such, the omitted need of having real-time data flows allowed to accept longer delays, however, the benefits of integrating such a solution in the overall infrastructure was not obvious and was greatly impacting the business value of the application due to the delays brought forward by the LDAA-v1 algorithms.

Based on the above facts, it was deemed necessary to experiment with a new revised and improved LDAA version (that is LDAA-v2) which promised to provide faster deliveries. The other big advantage of this would be also the highly elevated security guarantees over the ones offered by LDAA-v1. As the experiments have shown, <u>LDAA-v2 comes quite close to the targets set for the Activity Tracker demonstrator if the same security parameters are used as those in LDAA-v1</u>, and is in fact 150x faster that those of LDAA-v1. Still delays are there, however the time to wait is drastically lower of that of LDAA-v1, and despite the fact that the infrastructure still worked with transferring packets based on a schedule, it is possible to invoke the LDAA-v2 also on demand, as the delay is at the timeframe of 10-15seconds. It is also added, that as the experimentation in the stand-alone version has shown, the use of the strongest security parameters, at TSS level delivers results also faster that those of the -weak parameter of LDAA-v1, with an estimated reduction of around 33%.

Nevertheless, in both cases (and especially for the LDAA-v1) it needs to be pointed out that in case there exist requirements for real-time streaming data, these algorithms prove quite hard to use as they impose a delay which in many business cases is not tolerable. However, when it comes to requirements that are “close-to-real-time”, then different mechanisms can be employed, such as caching and creating continuously running application daemons, etc. to lower as much as possible the delay that comes in place mostly by the Sign() command.

In any case, it is stressed out that the demonstrator has been grounded around emulating the TPM and thus every time a command is issued the operating system 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, which is a process that adds extra delay in the overall process. This means that when the same algorithms are ready to be used in a hardware TPM, delays will be quite smaller and possibly not important.

Chapter 4 Demonstrator #3 – *Device Management*

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 centrally managed network devices.

In the context of this 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.

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 to 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

During the 2nd period, the architecture of the demonstrator has remained unchanged and is illustrated in Figure 29.

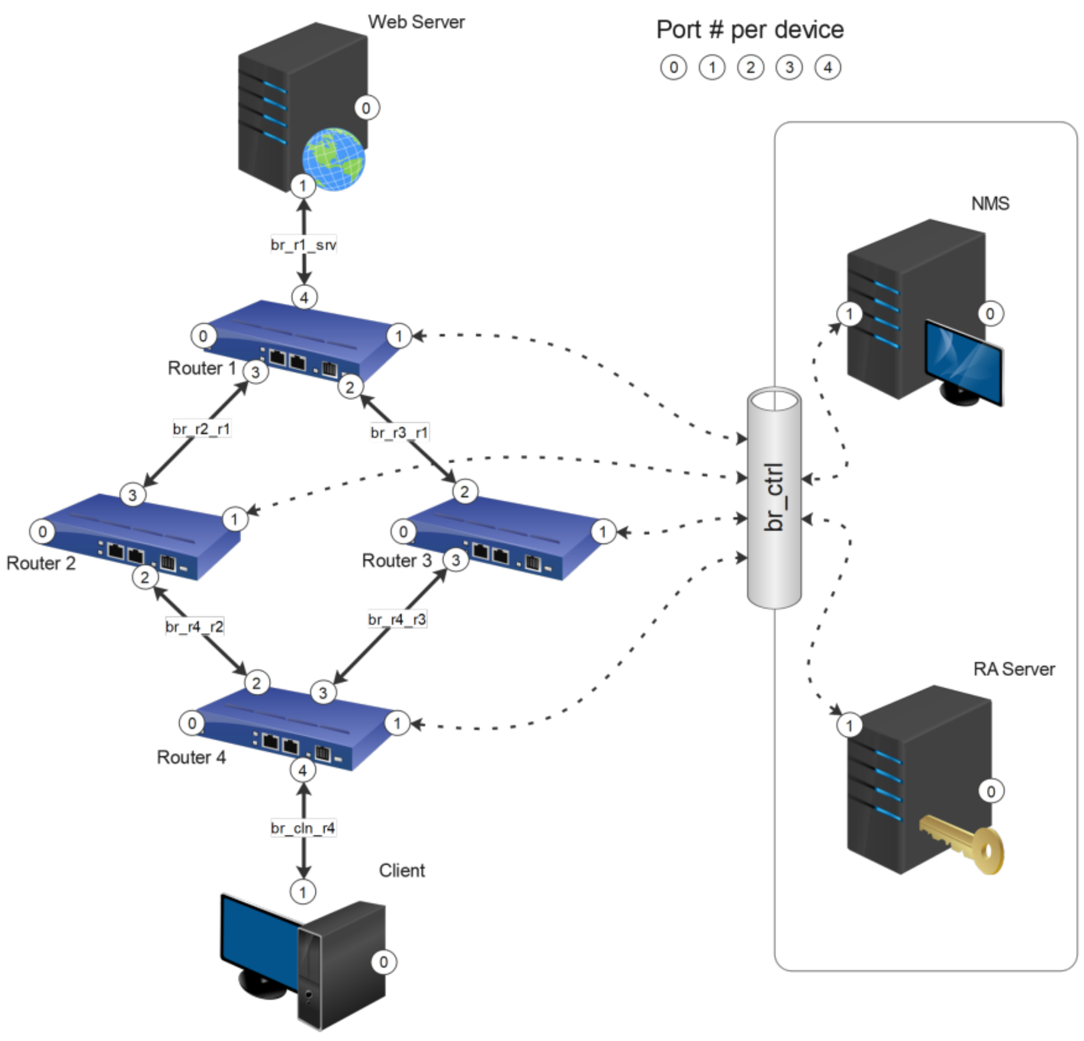


Figure 29: Demonstrator #3 – overall architecture and main entities

The Demonstrator is implemented as a virtualized network environment that consists of several key entities:

- **Routers** - route the traffic between Web Server and Client.
- **Network Management System (NMS)** - a server that manages routers over TLS channels.
- **Remote Attestation (RA) Server** - a server that is responsible for attesting the routers.
- **Web Server and Client** - machines that communicate to each other via the network of 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 interfaces 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

In the 2nd testing period, the demonstrator has been implemented and evaluated in an upgraded virtualized environment compared to the first cycle where the traditional SW-based TPM was leveraged. The intuition is to proceed to a detailed experimentation of the implemented QR v-TPM in such a resource-constrained decentralized environment. **Recall that v-TPMs perform cryptographic co-processor capabilities in software.** In the context of FutureTPM, additional cryptographic primitives have been implemented as part of the QR SW-based TPM – and its trusted software stack – that were instantiated to be able to run in virtualized execution environments. These are BIKE, SPHINCS+ and RAINBOW.

When added to a virtual machine, a v-TPM enables the guest operating system to create and store keys that are private. These keys are not exposed to the guest operating system itself. Therefore, the virtual machine attack surface is reduced. Usually, compromising the guest operating system compromises its secrets, but enabling a v-TPM greatly reduces this risk. These keys can be used only by the guest operating system for encryption or signing. In the context of the "Device Management" use case, the goal is to enable the NMS to remotely attest and validate the identity and correct configuration state of a router's firmware and operating system. The hardware platform contains an Intel i7-9700T CPU (2.00 GHz – 4.30 GHz) and 32 GB of RAM. The operating systems used are Ubuntu 20.04 in the host and openEuler 20.09 in the virtual machines. The hypervisor used is KVM, together with the QEMU emulator.

The v-TPM depends on the correct integration of the underlying software-based trusted platform module in a virtual machine. In this context, the QR SW-based TPM from INESC-ID, already used in the demonstrator of the 1st period, has been upgraded for better integration in virtualized environments according to the architecture defined by Stefan Berger. In particular, the automatic provisioning of a software TPM for a virtual machine, an existing feature of libvirt, has been enabled together with a custom script to perform QR-TPM-specific configuration tasks. The necessary changes across the software stack both in the host and in the virtual machines to support the QR-TPM have been already described in D6.3, section 4.3. Lastly, compared to the 1st period, the RA Server and the NMS have been moved to dedicate virtual machines.

Furthermore, to better evaluate the impact of v-TPMs on the performance of cloud-based applications (going beyond the target device management ecosystem), we have also considered the benchmarking of some core v-TPM services (i.e., key generation, signing, verification, etc.) as standalone processes. More specifically, in Section 4.3.2, we are documenting the performance and execution timings of the core set of TPM commands that have also been considered in the other reference scenarios. This evaluation was conducted by integrating SPHINCS+, Rainbow and BIKE on top of Dilithium, NTTRU and Kyber. **The motivation is to be able to compare not only the security offered by a v-TPM (against an actual hardware TPM chip with stronger security considerations) but also identify the performance overhead posed with relation to the level of security provided.**

4.3 Implementation Path Report for the 2nd Experimentation Period

While in the 1st testing period, the focus has been on the underlying trusted computing technologies combined under Comprehensive Integrity Verification (CIV), of which the main parts are a kernel-based functionality and associated user-space software, in the 2nd period HWDU has focused on the actual demonstrator platform and its components, which have been enhanced.

The following master images were pre-configured and used in the deployment:

- **NMS** - An image based on the openEuler 20.09 distribution, which has the core NMS code based on Django framework.
- **RA Server** - An image based on the openEuler 20.09 distribution, which has the core RA Server code based on Flask framework.
- **Router** - An image based on the openEuler 20.09 distribution. This image includes the FRRouting routing suite to make the machine act as a router in the network.
- **PC** - An image based on the Ubuntu Server 20.04 distribution, which was used to create Client and Web Server machines. This image does not include any additional programs and code, as only basic ping functionality is required.

In the NMS, the RA Server and the Routers an attest-tools library is used. It is a C library for abstracting the TPM hardware functionality to functions specific for simple remote attestation protocols. Each of the entities above interface with each other through standard Representational State Transfer (REST) Application Programming Interfaces (APIs).

The software stack of the complete environment is illustrated in Figure 30.

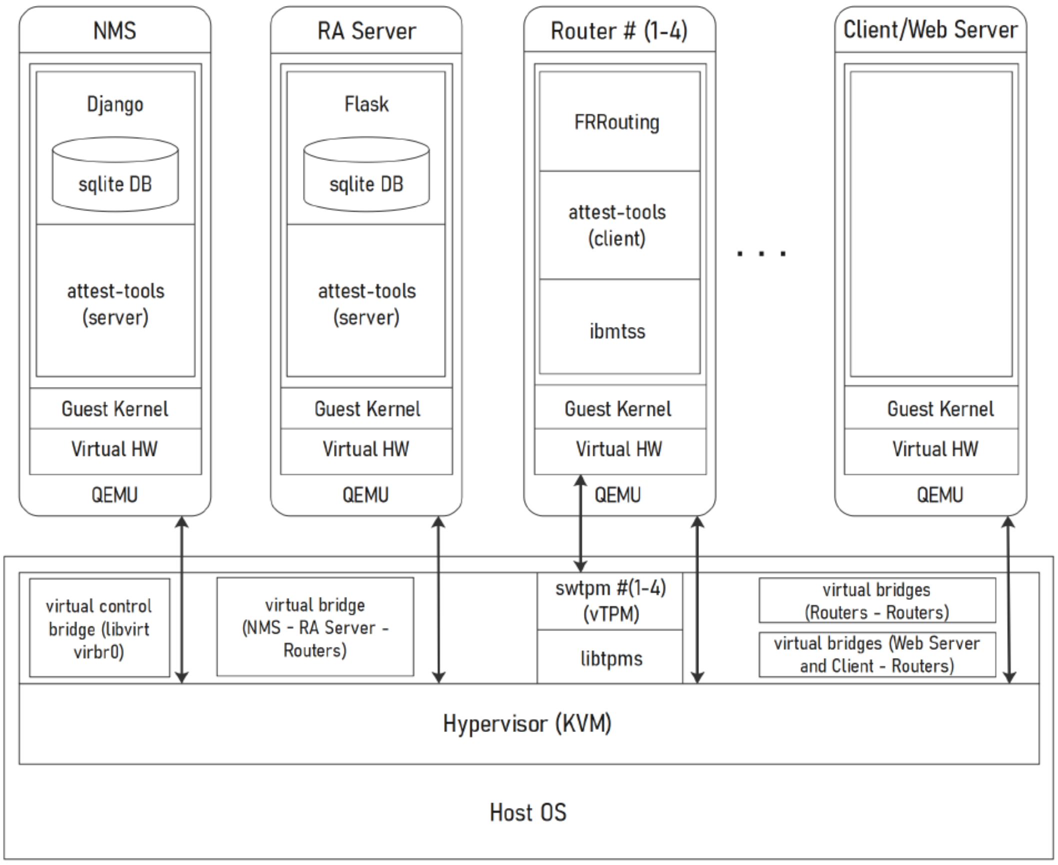


Figure 30: Demonstrator #3 – Software stack diagram

During the 2nd testing period, also some changes at a lower level of the software stack were made. The IMA Digest Lists extension has been reworked to become ready for production and to be accepted in the Linux kernel. In particular, work has been done to remove the unnecessary functionality (the removed ones will be reintroduced later, after the core part is accepted by kernel developers). Also work has been done to make the code easier to review, which is a prerequisite for contributing to the kernel.

In addition, more extensions to user space software were developed to support for digest lists during the entire software lifecycle. For the rpm package manager, two extensions were developed. The first is responsible to call an external program (from digest-list-tools) during the package building process to generate a digest list of files included in that package. The second is responsible to take the digest list from the package, when the package is installed in a running system, and to upload it to the kernel so that the kernel allows the programs inside the package to be executed. These extensions are used by OS build services (the Huawei's and SUSE's ones were tested) with very small configuration changes.

This work has been integrated in the Huawei OSes, EulerOS the commercial version and openEuler 20.09 LTS, the open-source version widely used in China. Since all packages include a digest list, users can easily use the Simple RA solution described above, and enable secure boot for applications, which protects both content and file metadata. The complete list of changes for openEuler can be found at the URL:

https://gitee.com/openeuler/security-facility/blob/master/ima/src/README.md

The full CIV solution that was previously developed during the 1st period has been updated to leverage the reworked IMA Digest Lists extension. Apart from small adaption work due to the fact that the code changed, no significant modifications were made on the remaining parts of full CIV, such as Infoflow LSM.

Lastly, the reworked code was proposed to kernel developers by sending the patches to the kernel mailing lists and it was partially accepted for kernel versions v5.5-v5.10.

4.3.1 User Stories Realisation

Compared to the 1st period, we have finalized user stories HWDU.NA.2, HWDU.NA.3, HWDU.NA.4 and we have completely implemented user story HWDU.EU.1, as described in D6.1.

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: ➢ *A routing policy depending, among others, on the trust state of routers is defined in the NMS.*
Workflow: ➢ In the NMS a metric is assigned to 4 router states: ATTEST_GOOD, ATTEST_UNKNOWN, ATTEST_BAD, OFFLINE ➢ The defined metric gets assigned to the routers when they change states, and sent to other routers
Issues encountered: -
Status: Completed

Description
User Story Title: <u>*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.*</u>
User Story Confirmations: ➢ *Routing tables on adjacent routers are modified when the trust state of a given neighbouring router changes*
Workflow: ➢ During the network lifetime the attestation state of some router changes   ➢ NMS sends a metric update of the affected router to all the other managed routers   ➢ The routers check if they are neighbours with the modified router, and adjust their internal routing metrics according to the new state of the modified router
Issues encountered: -
Status: Completed

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

Description

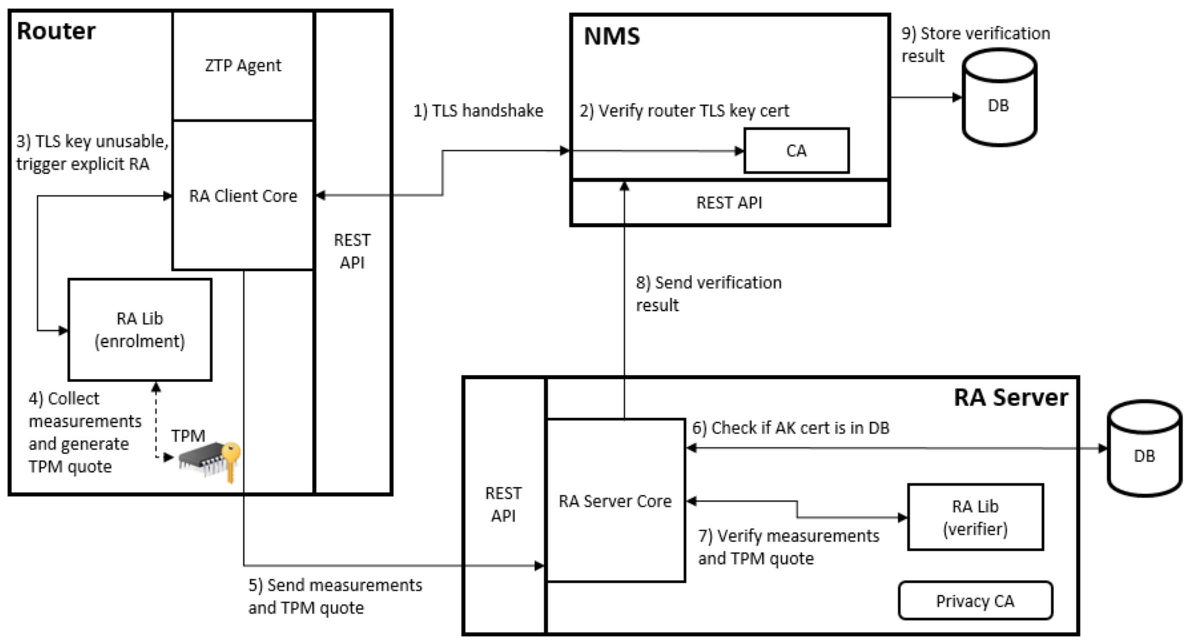

Router
ZTP Agent
3) TLS key unusable, trigger explicit RA
RA Client Core
REST API
RA Lib (enrolment)
4) Collect measurements and generate TPM quote
TPM
1) TLS handshake
NMS
2) Verify router TLS key cert
CA
REST API
9) Store verification result
DB
8) Send verification result
RA Server
6) Check if AK cert is in DB
DB
REST API
RA Server Core
7) Verify measurements and TPM quote
RA Lib (verifier)
Privacy CA
5) Send measurements and TPM quote

Figure 31: Router runtime verification

Components involved:

- **RA Server**: Remote Attestation Server that exposes a REST API to routers for device enrolment and explicit RA.
- **RA Lib (verifier):** Library running on RA Server to verify CSRs (for implicit RA) and quotes (for explicit RA).
- **RA Client Core**: Core logic running on each router
- **RA Lib (enrolment)**: **):** Library running on RA Client to generate TPM keys, quotes and CSRs
- **NMS**: Network Management System.

Workflow:

1. Establish TLS connection
   - The NMS initiates a TLS handshake with the router.
   - The router replies to the NMS and sends the certificate associated to the generated TLS key.
2. Verify router TLS key cert
   - The NMS verifies the router TLS key certificate against its CA.

Description
3. TLS key unusable, trigger explicit RA ➢ If implicit RA fails (TPM key unusable in the router due to configuration change), explicit RA is initialized by the RA Client Core **4.** Collect measurements and generate TPM quote. ➢ RA lib (enrollment) 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. ➢ NMS modifies the routing policy of the network according to the change of router's attestation state.
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: Complete
Degree of realisation: Full

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:

Workflow (AK Certificate)

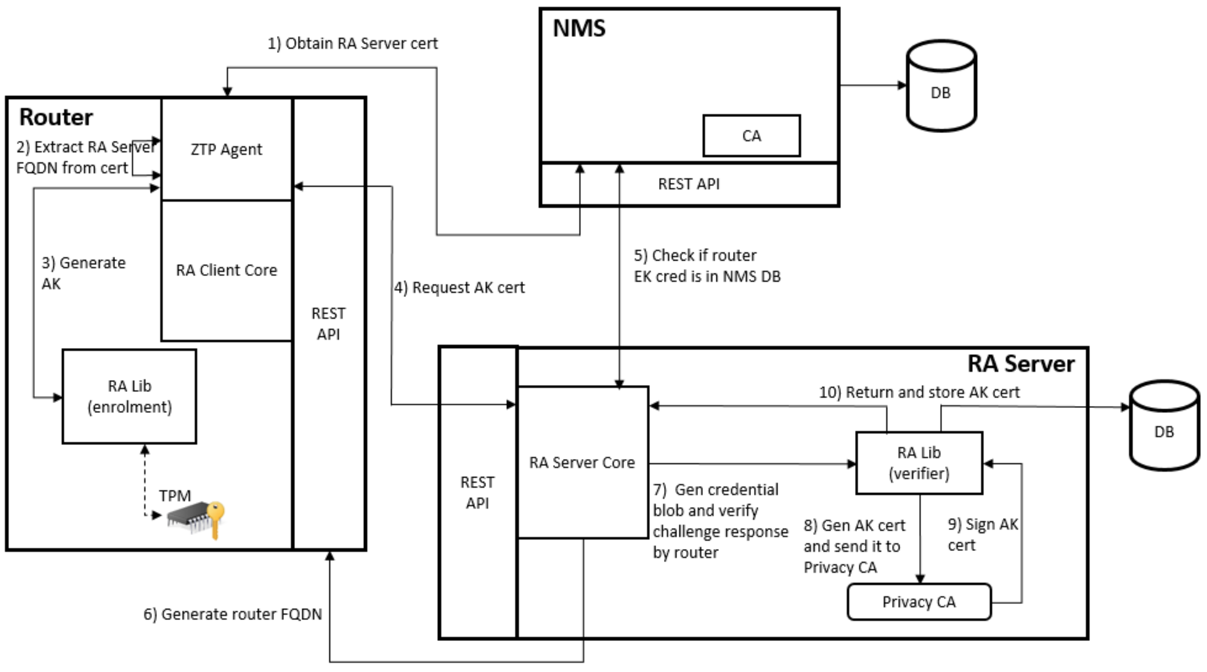


Figure 32: Router AK certificate generation

Description
1. Obtain RA Server cert ➢ ZTP Agent requests 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 for contacting it. **3.** Generate AK ➢ RA Client generates an AK that will be used to certify the TLS key and sign TPM quotes. **4.** Request AK cert ➢ ZTP agent 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 is enrolled (present in the database); this prevents unwanted routers from getting an AK certificate. **6.** Generate a FQDN for the router ➢ If the router has not been activated before, RA Server generates a new FQDN for the router based on the template defined by the network administrator (i.e. *router.huawei.X*), and sends it back to the router, so it applies a new FQDN **7.** 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. **8.** 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 **9.** Sign AK cert ➢ Privacy CA signs the AK certificate; RA Server sends it to the router. **10.** Return and store AK cert ➢ RA Server stores the signed AK certificate in its DB and returns the certificate to the router

Description

Workflow (TLS key certificate):

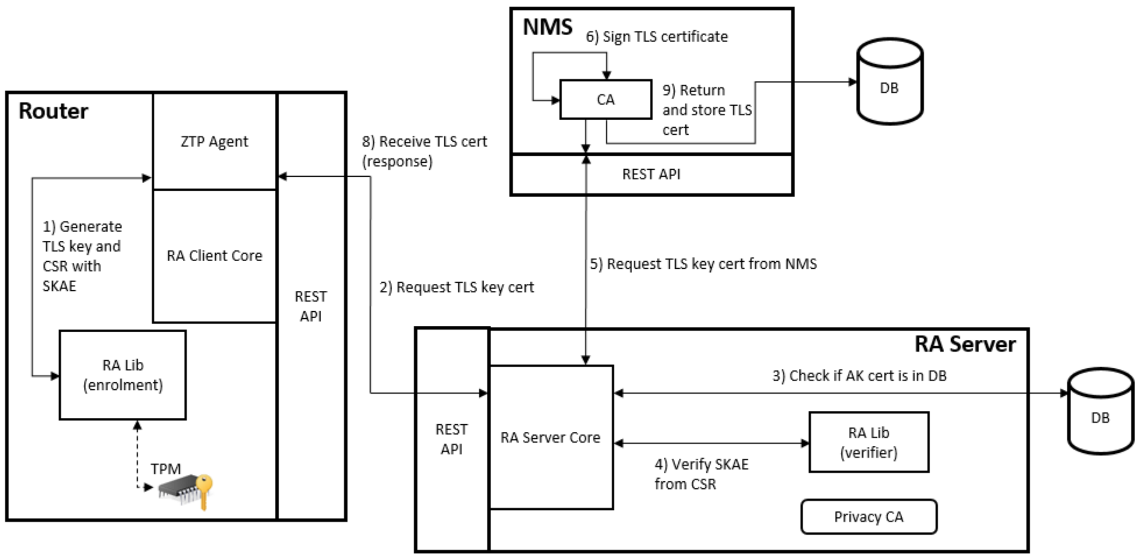


Figure 33: Router TLS cert generation

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. Request TLS key

   ➢ ZTP agent 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

Description
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. Request TLS key cert from NMS ➢ RA Server sends the CSR with the verified SKAE to the NMS, so that the NMS CA can sign it. 6. Sign TLS key cert ➢ The NMS CA signs the TLS key certificate 7. 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. 8. 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, NMS tries to establish a TLS connection with the router based on the new certificate to verify 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

Description
User Story Title: HWDU.EU.1 – As an End User, I want to access a web application hosted in a server that is remotely connected via the network of routers managed by the NMS
User Story Confirmations: ➢ The End User's browser successfully connects to the server and displays the application Alternative User Story Confirmation (server unreachable, potentially due to routing policy disallowing untrusted routers): ➢ The End User's browser cannot connect to the web server

Description
Workflow: 1. A request is sent from web client to web server through the network of managed routers when all the routers are in good state. ➢ The route of the request is recorded 2. One of the routers In the network changes its state (potentially due to untrusted configuration) 3. A request from client to server is sent again ➢ The route of the new request can be seen to avoid the affected router as long as it has alternative paths
Issues encountered: -
Status: Complete
Degree of realisation: Full

4.3.1.1 KPIs Measured

4.3.1.1.1 Quantitative Metrics

The table below shows the differences in performance when the demonstrator uses TPM 2.0 and the FutureTPM QR-TPM. Entries in bold report the total time needed to execute a demonstrator functionality. The time was taken from the host 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 commands 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 extracted by monitoring TSS Execute() function of the underlying trusted software stack.

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 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). Furthermore, NVRAM operations are slower due to the different amount of data to load (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 crypto (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.

Table 17: Demonstrator #3 – Comparison of Timings between TPM2.0 (SW) and FutureTPM (SW)

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
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**

TPM 2.0 Command	TPM 2.0 Timings (TSS)	FutureTPM Command	FutureTPM Timings (TSS)
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**
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

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.

During the 2nd period, we completed the missing functionality in the demonstrator to be able to measure KPI 4. During measurement we realized that the chosen criteria was not appropriate, as the amount of traffic diverted depends on the duration of the experiment. With a longer experiment we were able to obtain a higher percentage of diverted traffic than with a shorter one. Therefore, we decided to first measure the time between the detection of a compromised router by the NMS and the transmission of the first packet through the alternative path after the routing tables of the non-compromised routers have been updated. Then, we defined a real world scenario, a Zoom call, and we derived what the amount of traffic diverted would be depending on the network statistics specific of that scenario.

We performed the experiment in the network depicted in the following Figure:

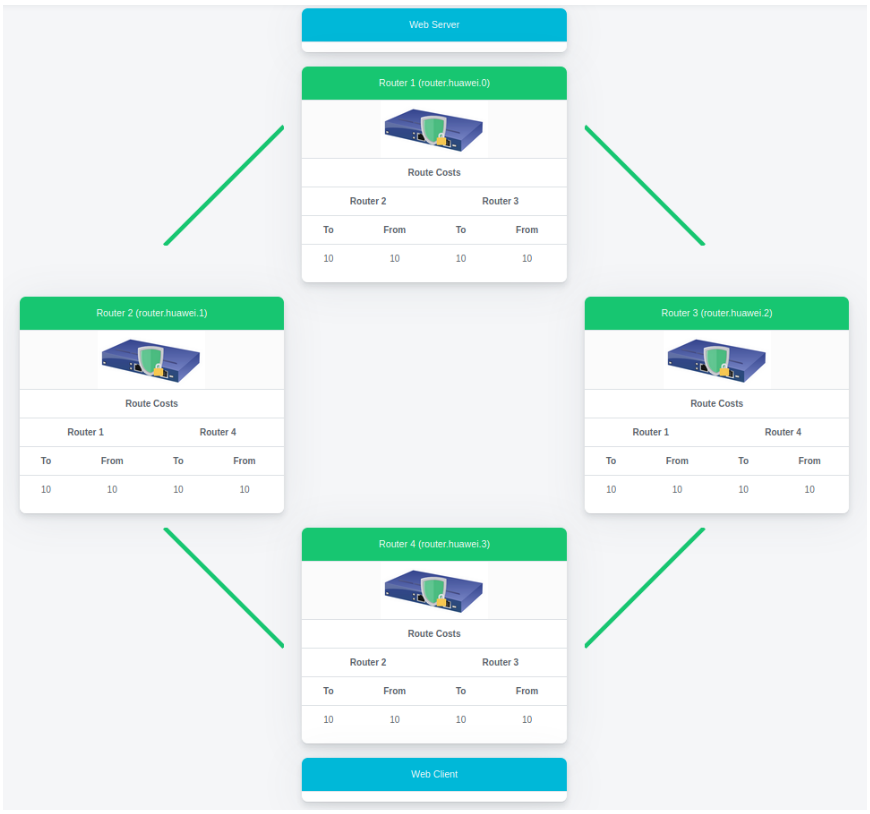
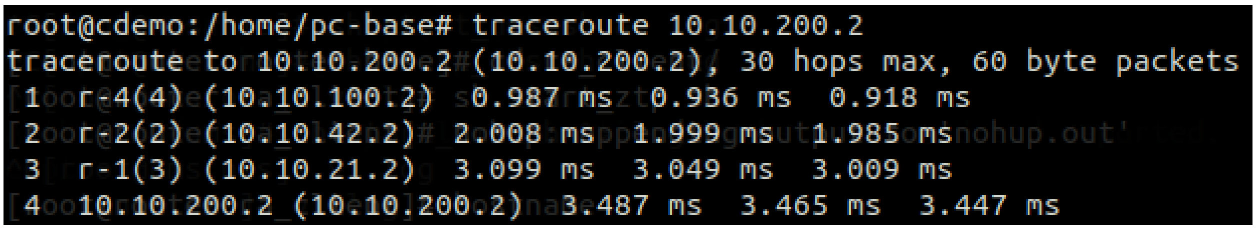


Figure 34: Network graph with all routers healthy

Initially, all routers are healthy and the metric for each route is 10. For the experiment, we executed the ping command from the client (10.10.100.1) to the server (10.10.200.2), with a rate of 100 packets per second. With this configuration, the traffic goes through Routers 4, 2 and Router 1:



```
root@cdemo:/home/pc-base# traceroute 10.10.200.2
traceroute to 10.10.200.2 (10.10.200.2), 30 hops max, 60 byte packets
 1  r-4(4) (10.10.100.2)  0.987 ms  0.936 ms  0.918 ms
 2  r-2(2) (10.10.42.2)  2.008 ms  1.999 ms  1.985 ms
 3  r-1(3) (10.10.21.2)  3.099 ms  3.049 ms  3.009 ms
 4  10.10.200.2 (10.10.200.2)  3.487 ms  3.465 ms  3.447 ms
```

Figure 35: traceroute output with all routers healthy

We started a network capture with tcpdump on Router 1 to get the MAC address of the packets going from the client to the server. After synchronizing the time on all systems, we performed an attack on Router 2 that made the TLS key unusable and we took the time when Router 2 was found compromised from the NMS log (Router 2 wasn't able to complete the TLS handshake). At this point, the NMS updates the routing tables of non-compromised routers, as depicted in Figure 36.

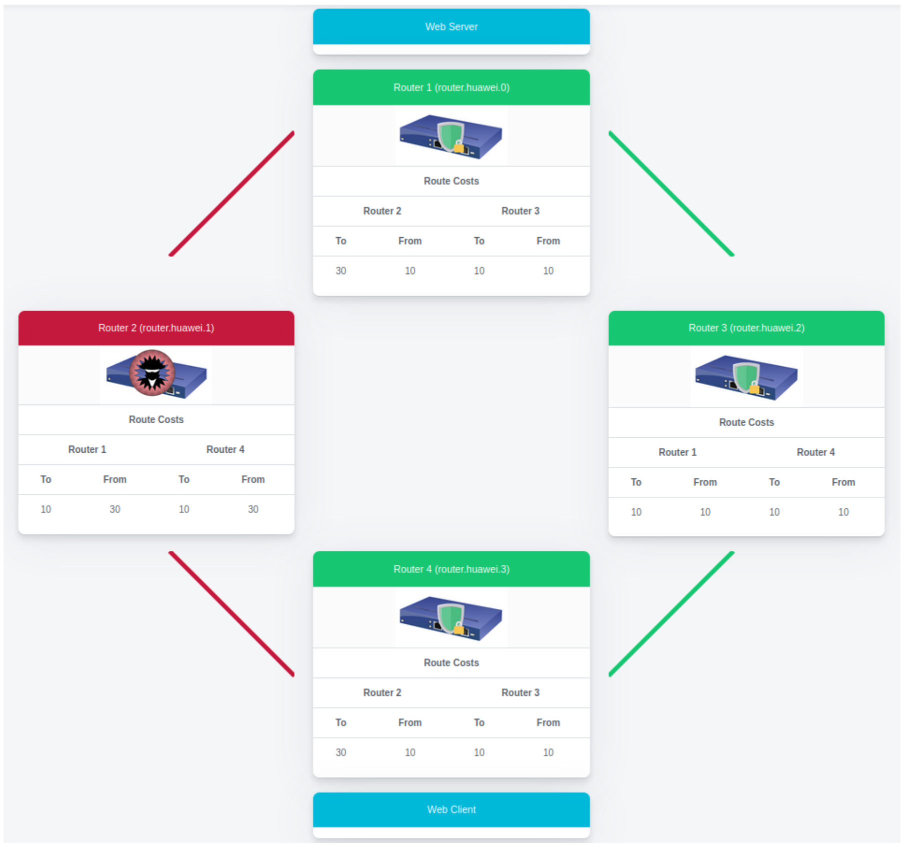
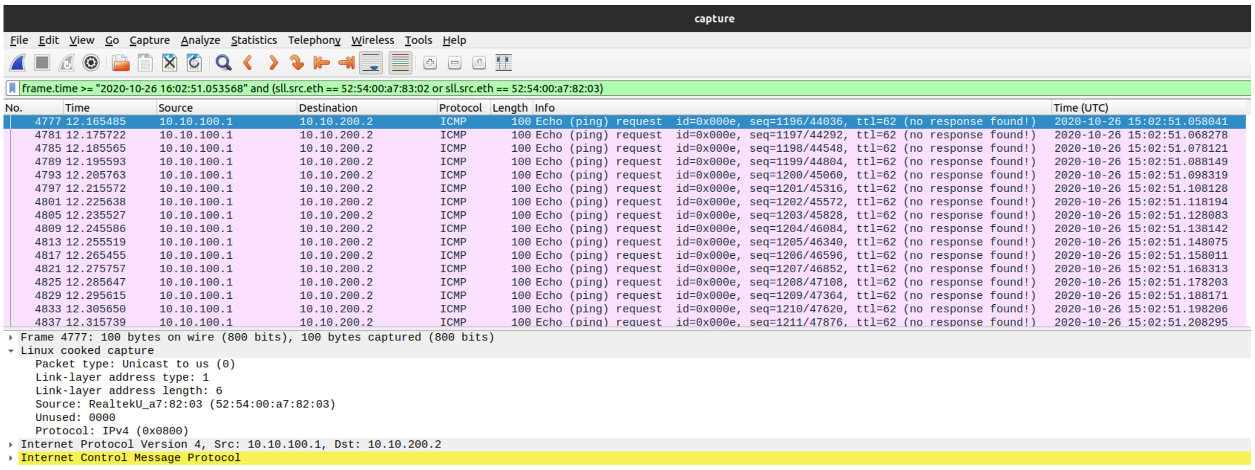


Figure 36: Network graph with one router compromised

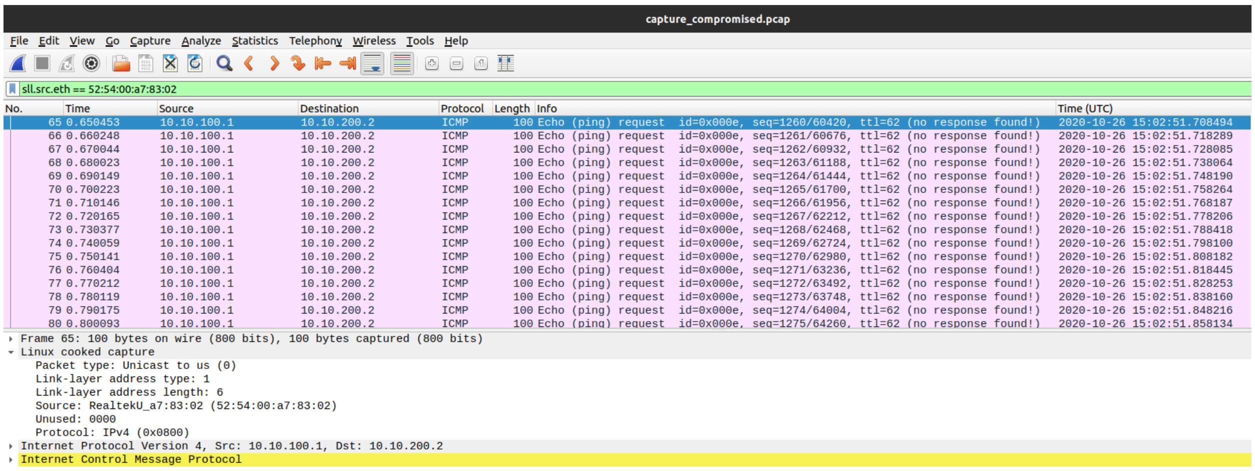
We then passed the network capture from Router 1 to Wireshark and selected the packets of interest, as depicted in Figure 37:


capture
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

```
frame.time >= "2020-10-26 16:02:51.053568" and (sll.src.eth == 52:54:00:a7:83:02 or sll.src.eth == 52:54:00:a7:82:03)
No.   Time        Source        Destination   Protocol Length Info                                                                    Time (UTC)
 4777 12.165485   10.10.100.1   10.10.200.2   ICMP     100 Echo (ping) request  id=0x000e, seq=1196/44036, ttl=62 (no response found!)  2020-10-26 15:02:51.058041
 4781 12.175722   10.10.100.1   10.10.200.2   ICMP     100 Echo (ping) request  id=0x000e, seq=1197/44292, ttl=62 (no response found!)  2020-10-26 15:02:51.068278
 4785 12.185565   10.10.100.1   10.10.200.2   ICMP     100 Echo (ping) request  id=0x000e, seq=1198/44548, ttl=62 (no response found!)  2020-10-26 15:02:51.078121
 4789 12.195593   10.10.100.1   10.10.200.2   ICMP     100 Echo (ping) request  id=0x000e, seq=1199/44804, ttl=62 (no response found!)  2020-10-26 15:02:51.088149
 4793 12.205763   10.10.100.1   10.10.200.2   ICMP     100 Echo (ping) request  id=0x000e, seq=1200/45060, ttl=62 (no response found!)  2020-10-26 15:02:51.098319
 4797 12.215572   10.10.100.1   10.10.200.2   ICMP     100 Echo (ping) request  id=0x000e, seq=1201/45316, ttl=62 (no response found!)  2020-10-26 15:02:51.108128
 4801 12.225638   10.10.100.1   10.10.200.2   ICMP     100 Echo (ping) request  id=0x000e, seq=1202/45572, ttl=62 (no response found!)  2020-10-26 15:02:51.118194
 4805 12.235527   10.10.100.1   10.10.200.2   ICMP     100 Echo (ping) request  id=0x000e, seq=1203/45828, ttl=62 (no response found!)  2020-10-26 15:02:51.128083
 4809 12.245586   10.10.100.1   10.10.200.2   ICMP     100 Echo (ping) request  id=0x000e, seq=1204/46084, ttl=62 (no response found!)  2020-10-26 15:02:51.138142
 4813 12.255519   10.10.100.1   10.10.200.2   ICMP     100 Echo (ping) request  id=0x000e, seq=1205/46340, ttl=62 (no response found!)  2020-10-26 15:02:51.148075
 4817 12.265455   10.10.100.1   10.10.200.2   ICMP     100 Echo (ping) request  id=0x000e, seq=1206/46596, ttl=62 (no response found!)  2020-10-26 15:02:51.158011
 4821 12.275757   10.10.100.1   10.10.200.2   ICMP     100 Echo (ping) request  id=0x000e, seq=1207/46852, ttl=62 (no response found!)  2020-10-26 15:02:51.168313
 4825 12.285647   10.10.100.1   10.10.200.2   ICMP     100 Echo (ping) request  id=0x000e, seq=1208/47108, ttl=62 (no response found!)  2020-10-26 15:02:51.178203
 4829 12.295615   10.10.100.1   10.10.200.2   ICMP     100 Echo (ping) request  id=0x000e, seq=1209/47364, ttl=62 (no response found!)  2020-10-26 15:02:51.188171
 4833 12.305650   10.10.100.1   10.10.200.2   ICMP     100 Echo (ping) request  id=0x000e, seq=1210/47620, ttl=62 (no response found!)  2020-10-26 15:02:51.198206
 4837 12.315739   10.10.100.1   10.10.200.2   ICMP     100 Echo (ping) request  id=0x000e, seq=1211/47876, ttl=62 (no response found!)  2020-10-26 15:02:51.208295
› Frame 4777: 100 bytes on wire (800 bits), 100 bytes captured (800 bits)
⌄ Linux cooked capture
    Packet type: Unicast to us (0)
    Link-layer address type: 1
    Link-layer address length: 6
    Source: RealtekU_a7:82:03 (52:54:00:a7:82:03)
    Unused: 0000
    Protocol: IPv4 (0x0800)
› Internet Protocol Version 4, Src: 10.10.100.1, Dst: 10.10.200.2
› Internet Control Message Protocol
```

Figure 37: Network capture from the time the NMS detected the attack on Router 2

After saving the selected packets in a different network capture, we obtained the time of the first diverted packet:



capture_compromised.pcap

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

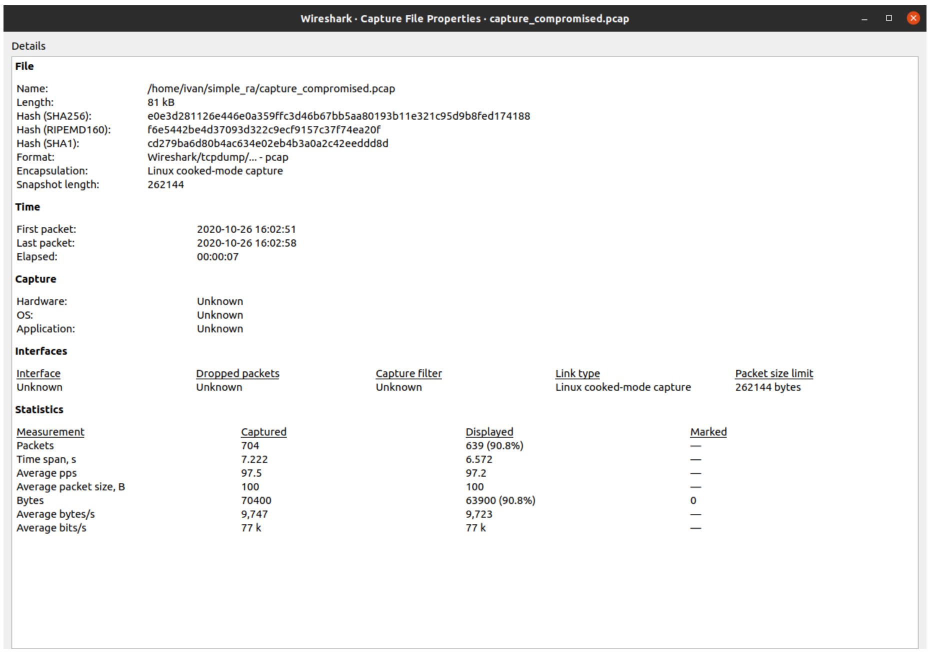
```
sll.src.eth == 52:54:00:a7:83:02
```

```
No.  Time      Source       Destination  Protocol Length Info                                                                      Time (UTC)
  65 0.650453  10.10.100.1  10.10.200.2  ICMP     100 Echo (ping) request  id=0x000e, seq=1260/60420, ttl=62 (no response found!)  2020-10-26 15:02:51.708494
  66 0.660248  10.10.100.1  10.10.200.2  ICMP     100 Echo (ping) request  id=0x000e, seq=1261/60676, ttl=62 (no response found!)  2020-10-26 15:02:51.718289
  67 0.670044  10.10.100.1  10.10.200.2  ICMP     100 Echo (ping) request  id=0x000e, seq=1262/60932, ttl=62 (no response found!)  2020-10-26 15:02:51.728085
  68 0.680023  10.10.100.1  10.10.200.2  ICMP     100 Echo (ping) request  id=0x000e, seq=1263/61188, ttl=62 (no response found!)  2020-10-26 15:02:51.738064
  69 0.690149  10.10.100.1  10.10.200.2  ICMP     100 Echo (ping) request  id=0x000e, seq=1264/61444, ttl=62 (no response found!)  2020-10-26 15:02:51.748190
  70 0.700223  10.10.100.1  10.10.200.2  ICMP     100 Echo (ping) request  id=0x000e, seq=1265/61700, ttl=62 (no response found!)  2020-10-26 15:02:51.758264
  71 0.710146  10.10.100.1  10.10.200.2  ICMP     100 Echo (ping) request  id=0x000e, seq=1266/61956, ttl=62 (no response found!)  2020-10-26 15:02:51.768187
  72 0.720165  10.10.100.1  10.10.200.2  ICMP     100 Echo (ping) request  id=0x000e, seq=1267/62212, ttl=62 (no response found!)  2020-10-26 15:02:51.778206
  73 0.730377  10.10.100.1  10.10.200.2  ICMP     100 Echo (ping) request  id=0x000e, seq=1268/62468, ttl=62 (no response found!)  2020-10-26 15:02:51.788418
  74 0.740059  10.10.100.1  10.10.200.2  ICMP     100 Echo (ping) request  id=0x000e, seq=1269/62724, ttl=62 (no response found!)  2020-10-26 15:02:51.798100
  75 0.750141  10.10.100.1  10.10.200.2  ICMP     100 Echo (ping) request  id=0x000e, seq=1270/62980, ttl=62 (no response found!)  2020-10-26 15:02:51.808182
  76 0.760404  10.10.100.1  10.10.200.2  ICMP     100 Echo (ping) request  id=0x000e, seq=1271/63236, ttl=62 (no response found!)  2020-10-26 15:02:51.818445
  77 0.770212  10.10.100.1  10.10.200.2  ICMP     100 Echo (ping) request  id=0x000e, seq=1272/63492, ttl=62 (no response found!)  2020-10-26 15:02:51.828253
  78 0.780119  10.10.100.1  10.10.200.2  ICMP     100 Echo (ping) request  id=0x000e, seq=1273/63748, ttl=62 (no response found!)  2020-10-26 15:02:51.838160
  79 0.790175  10.10.100.1  10.10.200.2  ICMP     100 Echo (ping) request  id=0x000e, seq=1274/64004, ttl=62 (no response found!)  2020-10-26 15:02:51.848216
  80 0.800093  10.10.100.1  10.10.200.2  ICMP     100 Echo (ping) request  id=0x000e, seq=1275/64260, ttl=62 (no response found!)  2020-10-26 15:02:51.858134
```

```
› Frame 65: 100 bytes on wire (800 bits), 100 bytes captured (800 bits)
⌄ Linux cooked capture
    Packet type: Unicast to us (0)
    Link-layer address type: 1
    Link-layer address length: 6
    Source: RealtekU_a7:83:02 (52:54:00:a7:83:02)
    Unused: 0000
    Protocol: IPv4 (0x0800)
› Internet Protocol Version 4, Src: 10.10.100.1, Dst: 10.10.200.2
› Internet Control Message Protocol
```

Figure 38: Network capture from the time packets are diverted to Router 3

The first packet was diverted after 0.65 seconds since Router 2 was found compromised by the NMS.



Wireshark · Capture File Properties · capture_compromised.pcap

Details

File

Name:	/home/ivan/simple_ra/capture_compromised.pcap
Length:	81 kB
Hash (SHA256):	e0e3d281126e446e0a359ffc3d46b67bb5aa80193b11e321c95d9b8fed174188
Hash (RIPEMD160):	f6e5442be4d37093d322c9ecf9157c37f74ea20f
Hash (SHA1):	cd279ba6d80b4ac634e02eb4b3a0a2c42eeddd8d
Format:	Wireshark/tcpdump/... - pcap
Encapsulation:	Linux cooked-mode capture
Snapshot length:	262144

Time

First packet:	2020-10-26 16:02:51
Last packet:	2020-10-26 16:02:58
Elapsed:	00:00:07

Capture

Hardware:	Unknown
OS:	Unknown
Application:	Unknown

Interfaces

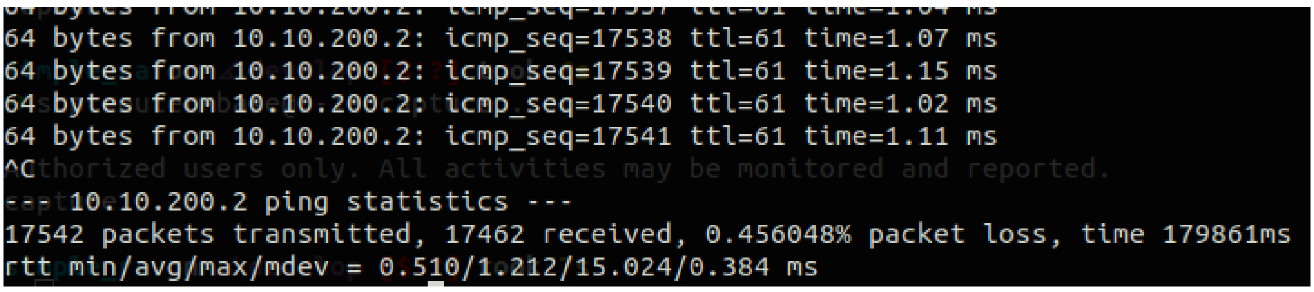
Interface	Dropped packets	Capture filter	Link type	Packet size limit
Unknown	Unknown	Unknown	Linux cooked-mode capture	262144 bytes

Statistics

Measurement	Captured	Displayed	Marked
Packets	704	639 (90.8%)	—
Time span, s	7.222	6.572	—
Average pps	97.5	97.2	—
Average packet size, B	100	100	—
Bytes	70400	63900 (90.8%)	0
Average bytes/s	9,747	9,723	—
Average bits/s	77 k	77 k	—

Figure 39: Wireshark statistics

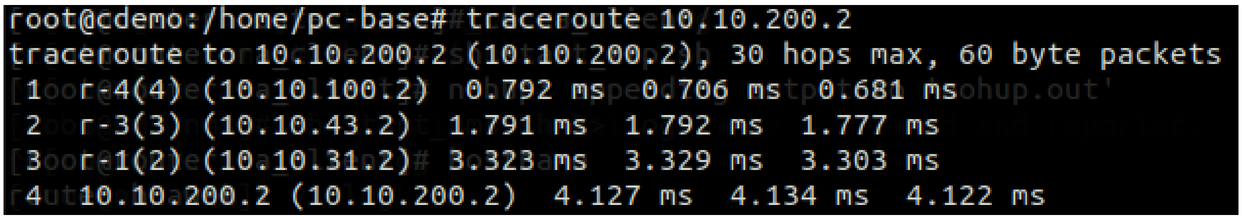
From the statistics, we can see that the percentage of packets diverted is 90.8% for this particular experiment.



```
64 bytes from 10.10.200.2: icmp_seq=17538 ttl=61 time=1.07 ms
64 bytes from 10.10.200.2: icmp_seq=17539 ttl=61 time=1.15 ms
64 bytes from 10.10.200.2: icmp_seq=17540 ttl=61 time=1.02 ms
64 bytes from 10.10.200.2: icmp_seq=17541 ttl=61 time=1.11 ms
^C
--- 10.10.200.2 ping statistics ---
17542 packets transmitted, 17462 received, 0.456048% packet loss, time 179861ms
rtt min/avg/max/mdev = 0.510/1.212/15.024/0.384 ms
```

Figure 40: ping output from the client

We also observed that a small percentage of packets was lost (0.46%), due to the reconfiguration of the routers. With traceroute, we got the confirmation that traffic was diverted to Router 3:



```
root@cdemo:/home/pc-base# traceroute 10.10.200.2
traceroute to 10.10.200.2 (10.10.200.2), 30 hops max, 60 byte packets
 1  r-4(4) (10.10.100.2)  0.792 ms  0.706 ms  0.681 ms
 2  r-3(3) (10.10.43.2)  1.791 ms  1.792 ms  1.777 ms
 3  r-1(2) (10.10.31.2)  3.323 ms  3.329 ms  3.303 ms
 4  10.10.200.2 (10.10.200.2)  4.127 ms  4.134 ms  4.122 ms
```

Figure 41: traceroute output after the attack is detected on Router 2

This figure shows the network utilization of the VMs and the purpose of the communication.

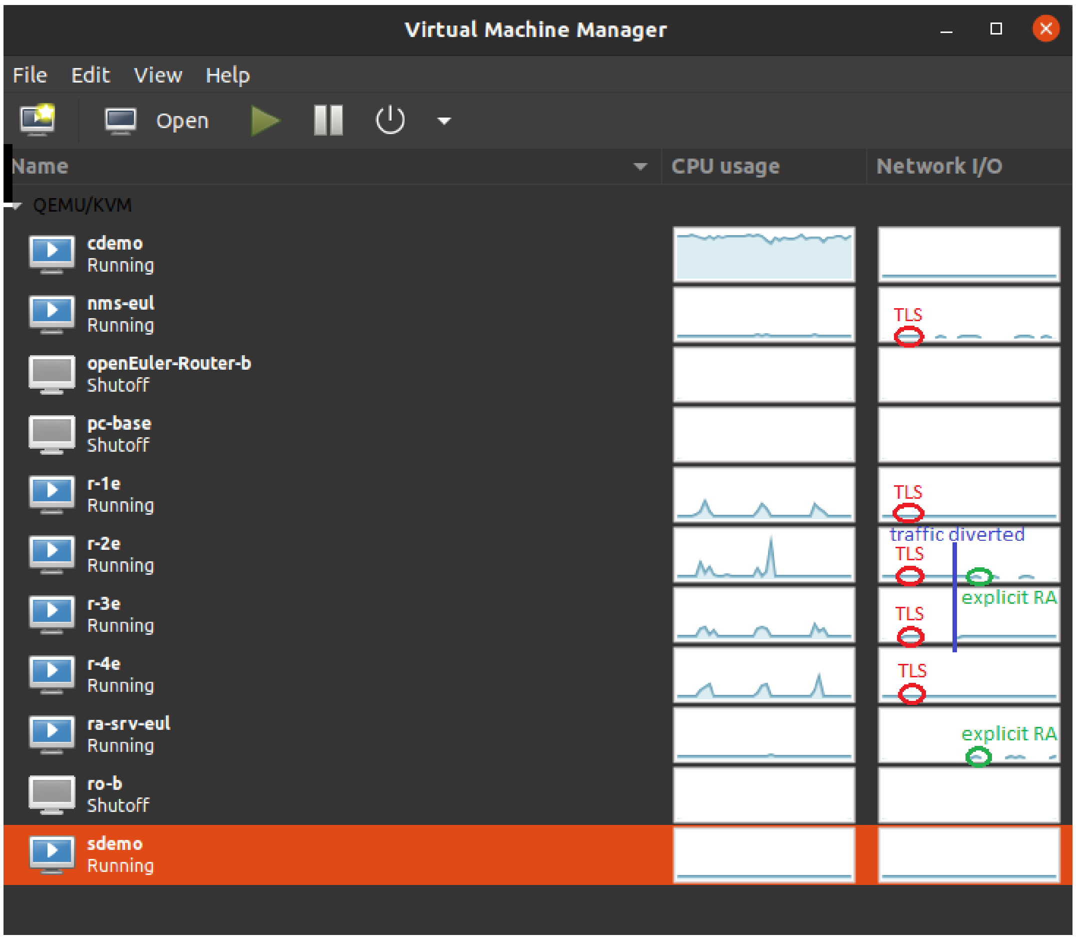


Figure 42: Virtual Machine Manager output before and after the attack

Lastly, for a Zoom call, we obtained the following statistics. According to https://skillscouter.com/video-conferencing-statistics/, the average duration of a call is between 31 and 60 minutes. Considering a call of 31 minutes, 0.65 seconds for reconfiguration and 0.82 seconds of traffic interruption, the amount of diverted traffic would be 99.92% ((1860 - 1.47) / 1860 * 100).

Table 18: Demonstrator #3 – Quantitative Metrics by M36

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% With FutureTPM: 100%	
2	Amount of routers hiding their integrity status	0%	0%	M	With TPM2.0: 0% 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.
3	Amount of detected integrity attacks on routers	80% (with integrity models)	60% (standard IMA)	M	With TPM2.0: 80% With FutureTPM: 80%	Besides attacks detected by standard IMA, we additionally cover attacks on: - mutable files; - non-regular files (e.g. IPC, socket etc.). Not covered: - control flow attacks; - file path protection.
4	Amount of traffic diverted to alternative paths when a router is compromised	75%	55%	G	With TPM2.0: 90.8% With FutureTPM: 90.8%	With TPM 2.0, the percentage could be higher due to faster reconfiguration with shorter TLS keys.
5	Amount of files whose integrity can be verified	100% (with integrity models)	99% (standard IMA)	G M	With TPM2.0: 100% With FutureTPM: 100%	All files can be verified.

4.3.1.1.2 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.

Table 19: Demonstrator #3 – Qualitative Metrics by M36

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: Supported With FutureTPM: Supported	
2	Trusted channels between NMS and each router in the network	Supported	M	With TPM2.0: Supported With FutureTPM: Supported	
3	Device authentication key for trusted channel protected by TPM	Supported	M	With TPM2.0: Supported With FutureTPM: Supported	
4	Integrity protection of router configuration data using a TPM key	Supported	M	With TPM2.0: Supported With FutureTPM: Supported	

4.3.2 QR Virtual Trusted Platform Module Experimentation

As aforementioned, to better evaluate the impact of v-TPMs on the performance of cloud-based applications, we have also considered the benchmarking of core v-TPM services as standalone processes. The goal is to be able to document the performance overhead posed with relation to the level of security provided; by experimenting with different security settings (as was the case for the L-DAA algorithm in the Activity Tracking use case).

There are four aspects of the v-TPMs in virtualized environments that define their level of security regarding to a physical TPM: **protection of the vTPM secrets, link between the vTPMs and the virtual guests, extension of the chain-of-trust from the host machine to the virtual guests and key hierarchies and management**. In what follows, we analyze the implementation of the FutureTPM QR v-TPM environment against the commands that are linked to these four service aspects. The evaluation was conducted by integrating SPHINCS+, Rainbow and BIKE in addition to Dilithium, NTTRU and Kyber.

We have to note, however, that this extra evaluation was also considered due to some challenges that were faced during the integration of a **full virtual TPM approach** in the context of the device management reference scenario. Towards this direction, special focus needs to be given to the direct mapping of all the physical TPM registers, NVRAM and keys to the v-TPM. Due to lack of existing drivers in the literature that can support such an extensive virtual TPM passthrough – a v-TPM service should be designed as a backend driver for the physical TPM to communicate with an emulated TPM TIS frontend -, **FutureTPM opted to provide a v-TPM environment designed as a software TPM backend implementation linked with an external library libtpms** (Section 4.2). As described in the previous chapters, this library provides TPM emulation. On the guest side there is an emulated TPM TIS frontend (see Figure 43) and a modified open source BIOS.

Within the QR v-TPM we have tested and measured the performance of the following crypto primitives: Dilithium, Kyber, Rainbow, SPHINCS+, and BIKE. In the following tables, we compare the virtual and software QR TPM variants in order to measure **how running the TPM in a virtualized environment would affect its usability** and what are the challenges that need to be taken into consideration when **running a TPM on top of a hypervisor**.

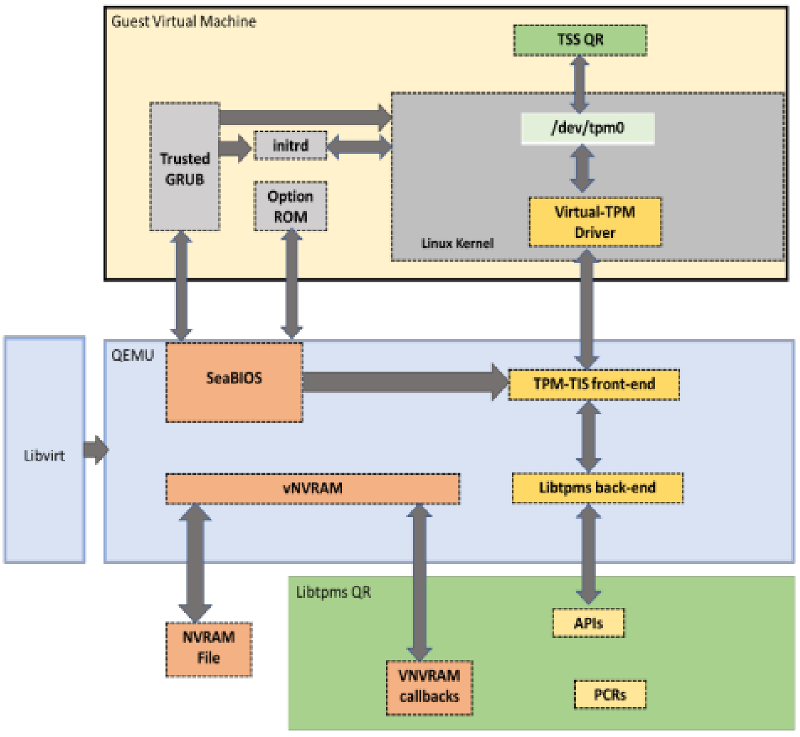


Figure 43: V-TPM Architecture

Firstly, we have measured the timings for the TPM commands mapped to the core key management services; a functionality, which as was described in the previous sections, served as the trust anchor for many of the demonstrator components. The results are shown in Table 20.

Table 20: Results of V-TPM Tests for Dilithium and Kyber

TPM2 Command	Command used	Timing (ms)
TPM2_create	./create -hp 80000000 -si -dilithium mode=2 -kt f -kt p -opr dil_priv.bin -opu dil_pub.bin -pwdp sto	278
TPM2_create	./create -hp 80000000 -kyber k=4 -den -kt f -kt p -opr kyber_priv.bin -opu kyber_pub.bin -pwdp sto -pwdk kyber	270
TPM2_LoadExternal	./loadexternal -hi p -ipu kyber_pub.bin	273
TPM2_FlushContext	./flushcontext -ha 80000001	296
TPM2_Load	./load -hp 80000000 -ipr dil_priv.bin -ipu dil_pub.bin -pwdp sto	636
TPM2_Sign	./sign -hk 80000001 -dilithium -if enc.bin -os sig.bin -pwdk dilithium	282
TPM2_LoadExternal	./loadexternal -hi p -ipu dil_pub.bin	284
(Separate functionality)	./kyberencrypt -hk 80000001 -id test.txt -oe enc.bin	276

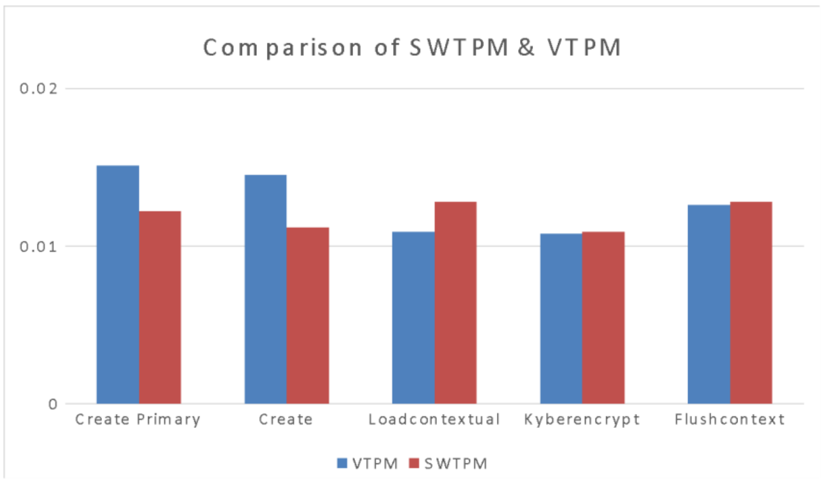


Figure 44: Timing Comparison of SW-TPM and V-TPM for Dilithium and Kyber

Secondly, we have also measured the timings for the SPHINCS+, Rainbow and BIKE commands tested for the v-TPM[6]. The results are shown in the following tables.

Table 21: SPHINCS+ Command Timings

Command	Timing (ms)
Generate Key	11
Generate Signature Key	66
Verify Signature	76

Table 22: Rainbow Command Timings

Command	Timing (ms)
Generate Key	589
Generate Signature Key	26
Verify Signature	24

Table 23: BIKE Command Timings

Command	Timing (ms)
Generate Key	6
Encapsulate	5
Decapsulate	6

Figure 45 below, reports the performance timing of all the tested V-TPM QR commands for comparison. With each of the commands executed, RAINBOW's key generation proved to be significantly slower, and this is mainly due to the keys used: *in fact, the size of the key generated was 148.5 KB whereas the key size of SPHINCS+ was 41 KB*. The timings for signing and verification of the keys within the V-TPM are more or less consistent among the tested algorithms.

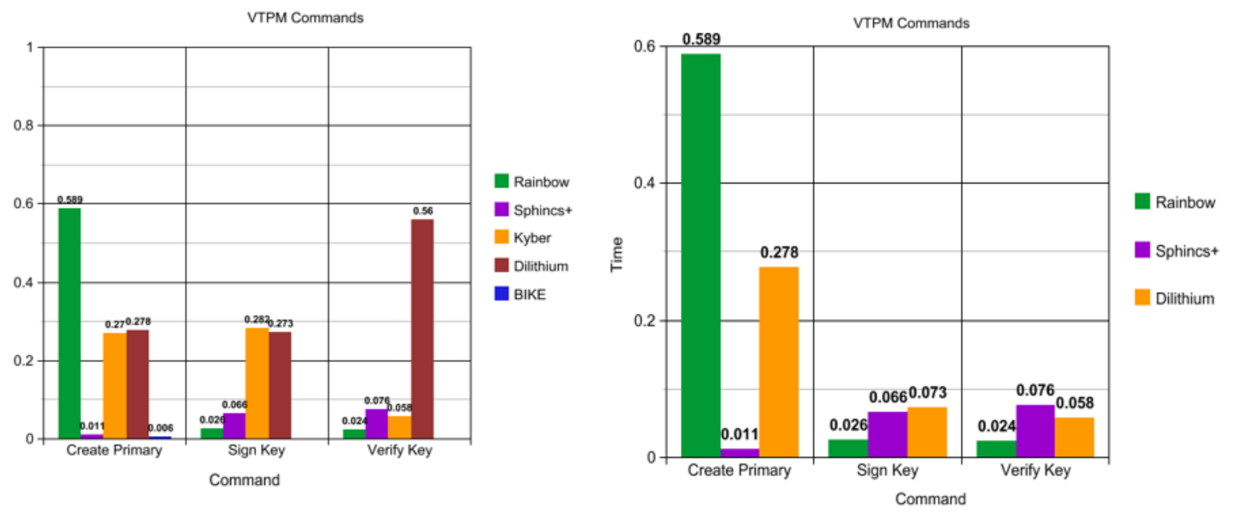
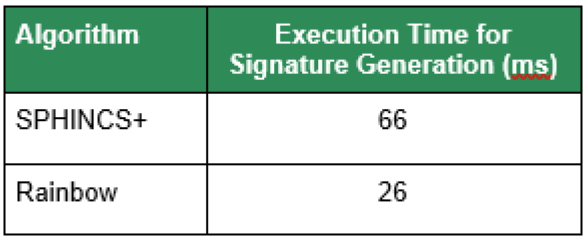


Figure 45: Timing Comparison of all V-TPM Commands for Rainbow, SPHINCS+ and Dilithium

[6] See the document "Technical Guide to V-TPM" for more details on the integration of these algorithms.

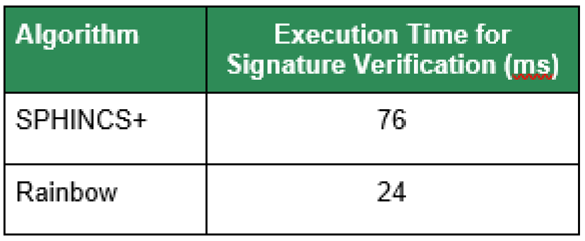
The following tables show the timings gathered for the execution of the signing and verification processes on the v-TPM: *the commands timings are listed on the right-hand side and the algorithm on the left-hand side*. Overall, given the security consideration, it can be stated that the performance overhead currently posed by the QR v-TPM is equivalent to the one of the software trusted platform module when neglecting any underlying network related issues (i.e., latency. Bandwidth, etc.). **Therefore, such a complete virtualized v-TPM of QUEMU sets the basis for a new future usage of this technology.**

Table 24: Execution Time for Signature Generation



Algorithm	Execution Time for Signature Generation (ms)
SPHINCS+	66
Rainbow	26

Table 25: Execution Time for Signature Verification



Algorithm	Execution Time for Signature Verification (ms)
SPHINCS+	76
Rainbow	24

4.4 Use Case Evaluation

At the end of the 2nd period, HWDU has achieved the objectives that have been defined for the device management demonstrator. All user stories have been successfully implemented and unit tested. The performance measurements obtained throughout the entire testing and evaluation life cycle of the project verify that the integration of advanced QR TPMs do not dramatically impact the routing capabilities of a device management system and enabled us to achieve the target KPIs.

Over the 1st period, the effort focused on implementing the underlying CIV technology, as well as the enrolment and remote attestation parts. 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. During the 2nd period, the demonstrator has been completed, adding support for the regular device management logic according to the remaining user stories.

The software TPM implementation has been integrated as a virtual TPM in this demonstrator, with all related infrastructure such as libvirt, qemu, kernel drivers. Such an implementation provided the following characteristics:

- **Transparent v-TPM offered to guest host:** The service of vTPM is based on full TPM emulation. Since the TPM TIS interface is emulated, no modifications have to be performed to the guest operating system.
- **No physical TPM required:** Since the vTPMs are fully emulated and not bound to a pTPM, this solution does not require the presence of a pTPM in the system.

The demonstrator is modular and abstracts the underlying TC technology from the actual demonstrator functionality, for simple product integration and administration. A graphical user interface (GUI) has been implemented to help visualize the results and map the technology on a real product scenario.

In conclusion, the device management demonstrator has proven the viability of a Quantum-Resistant TPM, of the operating system-based integrity technologies, as well as of the new use cases introduced in this project, for introducing trusted-computing-based management to telecom network infrastructure scenarios.

Chapter 5 Conclusions

The current deliverable, as identified in the introduction aims to cover main activities of the second evaluation cycle which dealt with the **evaluation, validation and refinement** of the algorithms and the platform offered by the FutureTPM project, as seen through the viewpoints of 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 such, D6.5 reflects on the "pilot' implementation and integration of the FutureTPM framework in those settings, testing the **assumptions of the project, 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**.

Towards this direction, the work performed for each one of the aforementioned demonstrators till M35 of the project was presented here.

- In the context of the <u>Secure Mobile Wallet and Payment</u> use case, during this period the HW QR-TPM was integrated, which has been implemented on an FPGA board and provides the NewHope and BLISS QR schemes. **The experimentation results proved that the performance of the HW TPM implementation - of FutureTPM - meets the vast majority of the performance KPIs and that the time discrepancies among the measurements are justified by the technical limitations of the FPGA-based implementation of the TPM and the intrinsic characteristics of the QR algorithms, while the overall, the performance of HW QR FutureTPM meets the goals of the demonstrator.**
- In the context of the <u>Personal Activity and Health Kit Data Tracking</u> use case, **an entirely new version of the LDAA protocol (LDAA-v2) based on a software QR-TPM infrastructure has been successfully implemented** in the S5 Activity Tracking demonstrator to overcome the obstacles faced by the integration of LDAA-v1 in the first evaluation cycle. Based on the experimentation outputs, the LDAA-v2 protocol achieved to come pretty close to the targets set for the Activity Tracker demonstrator and from a business perspective, it succeeded in delivering acceptable results in an operational environment, even if the measured performance was not fully meeting the ideal targets set. As such, this demonstrator showed that **QR-TPMs could offer a substantial improvement to the degree of offered privacy and security in data handling applications** when it comes to systems whose requirements are "close-to-real-time",
- In the context of the <u>Device Management</u> use case, during this period support for the regular device management logic has been added and the **software TPM implementation has been integrated as a virtual TPM** in this demonstrator, with all related infrastructure while a graphical user interface (GUI) has been implemented to help visualize the results and map the technology on a real product scenario. Based on the overall findings, the device management demonstrator has proven the viability of QR-TPM for introducing trusted-computing-based management to telecom network infrastructure scenarios.

This summary of all key performance indicators from the QR algorithms developed and tested, as well as the new remote attestation enablers, set the scene for the critical appraisal of all the project's artefacts towards securing both extremes of a network, namely the edge and the cloud. For this detailed analysis of the potential impact generated by the introduction of FutureTPM and discussions about the different outcomes, the reader is referred to deliverable D6.6 of the project, titled "<u>Validation Results, Performance Evaluation and Adoption Guidelines</u>".

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
DAA	Direct Anonymous Attestation
DH	Diffie-Hellman
eBPF	enhanced Berkeley Packet Filter
ECC	Elliptic Curve Cryptography
FIDO	Fast ID Online
FPGA	Field Programmable Gate Arrays
GDB	GNU Debugger
KPI	Key Performance Indicators
KVM	Kernel-based Virtual Machine
LDAA	Lattice based Direct Anonymous Attestation
LWMA	Linearly Weighted Moving Average
MFA	Multifactor Authentication
NFC	Near Field Communication
NMS	Network Management System
PC	Personal Computer
PCR	Platform Configuration Register
PDP	Policy Decision Point
PE	Policy Enforcement
PEP	Policy Enforcement Point

Abbreviation	Translation
QEMU	Quick Emulator
QR	Quantum Resistant
RA	Risk Assessment
SAK	Service Attestation Key
SKAE	Subject Key Attestation Evidence
TCB	Trusted Computing Base
TPM	Trusted Platform Module
TSS	TPM Software Stack
U2F	Universal 2nd Factor
URL	Uniform Resource Locator
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.

15 Rainbow. Available at: https://csrc.nist.gov/CSRC/media/Presentations/Rainbow/images-media/Rainbow-April2018.pdf. 2018

16 SPHINCS: practical stateless hash-based signatures. Website: https://sphincs.cr.yp.to/. 2017

17 R. del Pino, V. Lyubashevsky and G. Seiler, “Lattice-Based Group Signatures and Zero-Knowledge Proofs of Automorphism Stability”, ACM SIGSAC Conference, 2018.

18 L. Chen, N. El Kassem, A. Lehmann and V. Lyubashevsky, “A Framework for Efficient Lattice-Based DAA”, Proceedings of the 1st ACM Workshop on Workshop on Cyber-Security Arms Race – CYSARM'19, 2019.

19 ProVerif: Cryptographic protocol verifier in the formal model - [online] https://prosecco.gforge.inria.fr/personal/bblanche/proverif/

20 Tamarin Prover - a security protocol verification tool that supports both falsification and unbounded verification in the symbolic model. [online] http://tamarin-prover.github.io

21 GALATICS Repository. Website: https://github.com/espitau/GALACTICS

22 T. F. Consortium, “D6.3 – Demonstrators Implementation Report – First Release,” 2020.

23 T.F. Consortium, “D5.3 – Final Version of QR TPM Implementation”, 2020.

24 T.F. Consortium, “D4.5 – Runtime Risk Assessment, Resilience and Mitigation Planning”, 2021.

25 T. F. Consortium, “D4.4 – FutureTPM Risk Assessment Framework”, 2021.