D1.2

FutureTPM Reference Architecture

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 / D1.2 / 1.0
Work package contributing to the deliverable:	WP 1
Due date:	Sept. 2018 – M09
Actual submission date:	3rd October, 2018

Responsible organisation:	UB
Editor:	Jose Moreira (UB) Thanassis Giannetsos, Liqun Chen (SURREY)
Dissemination level:	PU
Revision:	1.0

Abstract:	Deliverable D1.2 provides the specification of the FutureTPM reference architecture, the functional components and interfaces between them. It also provides an analysis and point of reference for the FutureTPM in relation to the Reference Scenarios, including an analysis of the TPM commands to be used and updated, all relevant classical protocols and the use cases themselves.
Keywords:	Architecture Specification, Functional Components, Interfaces & APIs, Requirements Analysis, TPM Specification.

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

Editor

Jose Moreira (UB)

Thanassis Giannetsos, Liqun Chen (SURREY)

Contributors (ordered according to beneficiary numbers)

Sofianna Menesidou (UBITECH)

Roberto Jordaney, Daniele Sgandurra (RHUL)

Mark Ryan (UB)

Rainer Urian (IFAG)

Christian Hanser (IFAT)

Dimitris Panopoulos, Ioanna Michael, Alexandros Tsitsanis (SUITE5)

Paulo Martins, Leonel Sousa (INESC-ID)

Christos Xenakis, Christoforos Ntantogian, Eleni Veroni, Nikolaos Koutroumpouchos (UPRC)

Roberto Sassu, Silviu Vlasceanu (HWDU)

Fanis Sklinos, Stratos Moros (INDEV)

Executive Summary

With the definition of the technical and use case requirements in Deliverable D1.1 [1], the FutureTPM Consortium has been able to better identify the challenges that arise from making use of Trusted Computing technologies in industry products that currently do not support them, and to determine which ones are more relevant for the project. Understanding which functionalities are required is crucial for the definition of the overall FutureTPM framework architecture, including the interfaces exposed by the different components of the framework, which is the focus of this deliverable.

FutureTPM Deliverable D1.2 moves one step closer to the fulfilment of the project's vision which is the development of QR-based TPM environments for enhancing the security and privacy posture of cyber-physical systems. The starting point is an overview of the state-of-the art of the relevant key technology axes considered for the FutureTPM reference architecture: the TPM 2.0 Specification, released by the Trusted Computing Group (TCG), and also technologies related to Risk Management and Security Policy Enforcement aspects. A general understanding of the current TPM 2.0 Specification will, on one hand, help the Reference Scenarios' partners to adapt their applications to use the TPM platform, and on the other hand serve for the technical partners as a basis for adding support for the Quantum Resistant (QR) primitives selected in the context of WP2.

Based on this comprehensive analysis, in this deliverable, we define the overall architecture of the FutureTPM and the components that comprise it, in complete alignment with the derived functional and non-functional requirements. It consists of three main components: the **QR TPM**, the **Risk Assessment** and the **Security Enforcement** components.

- The QR TPM is a modified version of TPM 2.0 with support for QR algorithms. Applications will interact with it through a modified version of the TSS which will expose new API calls for stateful operations, and support longer key and output sizes respectively for symmetric and hash algorithms. Whereas the particular selection of QR algorithms to be implemented are still under discussion in WP2, there is some agreements, e.g., the NewHope and gTesla QR algorithms will be provided for the hardware-based QR TPM. Moreover, given that no TCG specification currently exists for QR algorithms, the Reference Scenarios will initially be based on the TPM 2.0 API, and will adapt their applications when the first version of the QR TPM API is released (in M18). Technical partners will ensure, for the entire development lifecycle that the platform still meets the functional requirements already introduced in Deliverable D1.1 [1].
- The Risk Assessment component quantifies, based on API calls tracked with techniques such as eBPF, the risk that an attacker with a quantum computer is able to steal information managed by use case demonstrators. It will be conceptually separated in two parts: a design-time component, and a run-time component, which will be implemented as a client application within the host device, and it will be in charge of updating the initial risk assessment provided during design-time.
- Finally, the Security Enforcement component will block unsafe usage of API depending on the provided policies, which will be elaborated to tailor the requirements of the three envisioned Reference Scenarios.

This document also defines the development methodology for ensuring that tasks are executed with the right priority. Development consists of three phases: during the Design Phase the QR algorithms are selected and a formal, holistic analysis of the QR TPM will be performed; during the Risk Assessment Phase, risks associated to the design of the platform are identified and an initial set of policies will be defined; during the Implementation Phase, the QR TPM, the Risk Assessment and the Security Policy Enforcement components will be implemented. The whole cycle of implementation activities will be iterative and based on two stages for better capturing any updates the need to be performed after the initial testing of the framework in the context of the envisioned demonstrators. Finally, the current deliverable elaborates on the approach that will be followed in order to realize the described functionalities during the implementation and integration phases of the FutureTPM framework.

Contents

List of Figures

List of Tables

Code Listings

Chapter 1 Introduction

Within the context of information security, the term “trust” has as many interpretations as people are talking about it. The Trusted Computing Group (TCG) is a not-for-profit industry consortium that aims to provide standards for Trusted Computing technologies and to promote their usage. Under the TCG view, Trusted Computing has a different meaning from what we understand as secure computing. “Trust” is formalized as “predictable behaviour,” even if this behaviour is far away from being worth of trust, in the social sense of the term. E.g., we trust that a banking server behaves securely (we trust that it does not have any security vulnerability that will pose a risk to our accounts), but we also trust that a laptop full of viruses stop working properly when we use it. Under this view, clearly, a trusted system is not the same as a secure system. However, a mechanism to determine the predictability of a component gives us a foundation to build secure systems.

Trusted Computing provides confidence in a system, especially if the system’s behaviour isn’t fully secure or might become insecure. It establishes whether a system is the intended system, i.e., whether it is doing what it is designed to do, and provides controlled access to keys and secrets that depend on the system’s current behaviour. It also allows a compromised system to be restored by installing and replacing software.

The TCG-defined methodology relies on the concept of Root of Trust. A Root of Trust is a component of a system on which all other trust is based, and which must be blindly trusted. It is worth to note that a Root of Trust is inherently unverifiable: if a system has a proposed Root of Trust and relies on another component to verify it, then this second component becomes, implicitly, the actual Root of Trust. While it is not possible to determine if a Root of Trust is behaving properly, there are defined certification procedures that allow manufacturers to provide assurance that a root has been implemented in a way that renders it trustworthily.

The TCG designates three Roots of Trust for a trusted platform [2], [3]:

- **Root of Trust for Measurement (RTM):** a component that makes the initial integrity measurements on certain system components, and stores that measurement in a secure location.
- **Root of Trust for Storage (RTS):** a component that provides confidentiality and integrity of data.
- **Root of Trust for Reporting (RTR):** a component that honestly and verifiably reports designated contents of the RTS.

A chain of trust is a sequence of measurements, initiated by the RTM, where each component measures the next component, and only transfers the control to that component if it produces an expected measurement. RTMs can be either static (SRTM) when the chain of trust is started at system boot, or dynamic (DRTM) when a chain of trust is started at some point after system boot. SRTMs are typically implemented within the Basic Input/Output System (BIOS) boot block, whereas DRTMs require a CPU operating in a special trusted mode, e.g., Intel Trusted Execution Technology (TXT) [4].

A Trusted Platform Module (TPM) is a component defined by a TCG specification. It is envisioned as a purely passive, inexpensive, tamper-resistant device, which acts as the RTS and the RTR of a trusted platform. Additionally, it also offers some limited secure storage, a random number generator (RNG), and (highly) constrained cryptographic functionalities. As a passive device, a TPM does not have visibility of the system where it is installed. However, the TPM together with the RTM (which reports its measurements to the TPM), can form the basis from which we can bootstrap trust in other parts of the system. Typical use cases of a TPM are: release an encryption key only if a chain of trust has produced an expected measurement, or attest reported integrity measurements to another system. The TPM does not (and cannot) judge if those measurements signify that the system is in good state or not. The trustworthiness of the measured values is simply judged by the requesting party.

One of the TCG's main tasks is to define and maintain the architectural specification of the TPM. It is conceptually divided into two stages: the main TPM specification, and a set of domain-specific profiles that define the particularities of selected platforms (e.g., servers or mobile devices). The current main TPM specification (TPM 2.0) is split into four parts, which detail the general architecture [3], the data types and structures [5], the command descriptions [6], and the set of supporting routines used by the commands [7].

Departing from the TCG view of Trusted Computing, the FutureTPM project aims at creating a whole Trusted Computing framework, where the design of a QR TPM –from a formal, holistic perspective– will constitute a paramount objective. The goal is to develop a TPM-based solution that can enhance the security posture of the hosting device: a TPM system that is not merely secure for today but that will also remain secure in the long term against attacks by quantum computers. The architecture obligations are motivated, but not restricted, by the requirements of the three Reference Scenarios considered for the project: secure mobile wallet and payments, personal activity and health kit tracking, and device management. Moreover, a lot of effort will be put on designing a system that, in addition to have the desired security and privacy properties, it will also allow a 'smooth transition' between non-QR and QR cryptography. This envisages scenarios where cryptosystems from both domains need to coexist, e.g., to allow legacy applications and services to operate until they are QR-ready. Consequently, it will be investigated how the QR TPM can become a drop-in replacement for TPM 2.0, while maintaining the interoperability of present-day systems.

1.1 Scope and Purpose

The main focus of this deliverable is to provide a comprehensive overview of the specification of the FutureTPM reference architecture that will serve as a basis for the whole duration of the project. This includes the components of the FutureTPM framework, their interfaces and the characteristics of the APIs that will be provided for the communication between them. FutureTPM revolves around three technological axes that are directly related to the core architectural components: the QR TPM (and its host device), the Risk Assessment component, and the Security Policy Enforcement component. An overview of these axes will be provided, as well as how are they going to be instantiated within FutureTPM.

Further, this document provides an analysis and point of reference for the FutureTPM architecture in relation to the three specific Reference Scenarios defined in the DoA, including an analysis of relevant classical protocols and the scenarios themselves, in terms of FutureTPM functionality. Hence, this document will also structure the technical requirements imposed by the three envisioned Reference Scenarios, documented in Deliverable D1.1 [1], to support the new protocols and the new cryptographic primitives required to achieve QR security.

To this end, Deliverable D1.2 will provide a thorough system-level architectural specification that will comprise a high-level overview of the architecture layers and components, as well as, the technology axes and contributions to open-source projects, that will guide the implementation of the particular layers. As Deliverable D1.2 will guide the development of the technical components comprising the platform, this deliverable also includes an initial overview of the interaction patterns and intercommunication schemes between system components, users and third-party entities. Thus, starting from the mapping of the system requirements to platform components, each component will be further decomposed into high-level functional blocks and supported primitives and interfaces. In turn, Deliverable D1.2 introduces the analysis performed to derive use-cases describing the implementation scenarios of the mechanisms that are to be developed within the scope of the project demonstrators, suitable acceptance criteria per demonstrator and their mapping to both system requirements and platform components.

Departing from the current TPM 2.0 Specification [3], the goal is that the API of FutureTPM mirrors as much as possible the API of present day TPMs. Therefore, the identification of the required extensions and modifications of the current API to handle the additional constraints imposed by the QR requirements and the Reference Scenarios will also lie within the scope of this deliverable.

We stress here that some of the specific details of the architecture are not likely to be known at the moment of submission of this deliverable (e.g., what specific algorithms will be implemented following the recommendations from WP2). Thus, they will be precisely defined in the context of the associated work packages.

1.2 Relation to other WPs and Deliverables

With the definition of clear use-cases that will be supported along with the documentation of the FutureTPM reference architecture and the further decomposition of the basic system entities and the intercommunication scheme among them, this deliverable (D1.2), will be used as an agreed upon instruction set guiding the development of the components that must be delivered by the FutureTPM project. Figure 1 depicts the direct and indirect relationship of the deliverable to the other Work Packages (WPs). The definition of the system-wide reference architecture is cornerstone, along with the requirement scheme documented in Deliverable D1.1 [1], in order to drive the technical work of WP2-WP5. What is more, with the clear definition of the project use-cases, demonstrator descriptions, evaluation acceptance criteria, and the prioritization of requirements to match the needs of the use-cases, the work in WP6 can begin as planned.

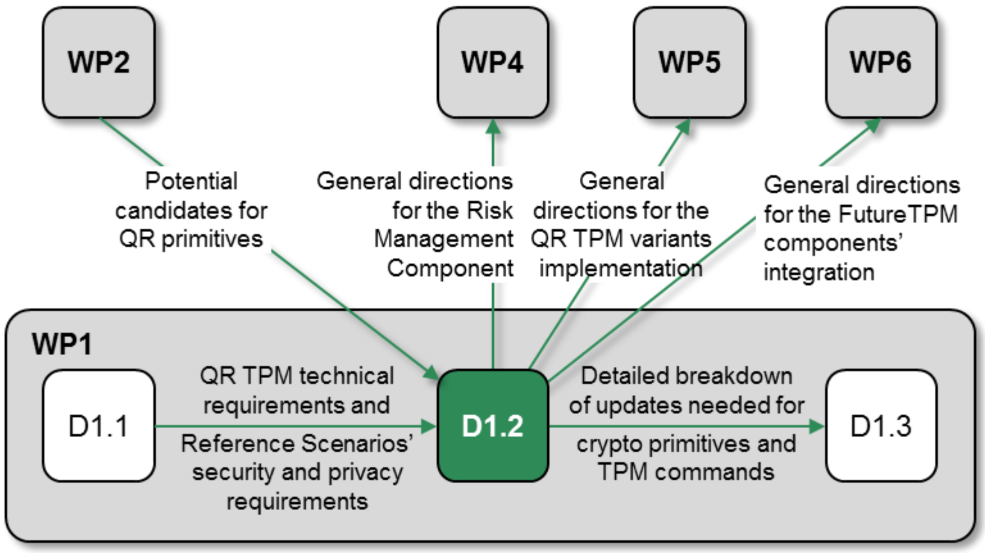

WP2
WP4
WP5
WP6
Potential candidates for QR primitives
General directions for the Risk Management Component
General directions for the QR TPM variants implementation
General directions for the FutureTPM components' integration
WP1
D1.1
QR TPM technical requirements and Reference Scenarios' security and privacy requirements
D1.2
Detailed breakdown of updates needed for crypto primitives and TPM commands
D1.3

Figure 1: Deliverable D1.2 relationships within the FutureTPM project.

Within WP1, the current document directly feeds from Deliverable D1.1 [1], which provides the definition of the technical, functional and security requirements for FutureTPM, as well as the extraction of the Reference Scenarios specific security and privacy requirements. This serves as the core information to structure and translate these requirements into concrete architectural specifications in order to meet the desired QR capabilities of the FutureTPM framework. In addition, within WP2, Deliverable D2.1 [8] provides the technical basis of the cryptography subsystem of the projected QR TPM, by proposing a set of potential QR primitives, and as such it will help establish the scope of the necessary changes in the reference architecture related to that component.

The outcome of Deliverable D1.2 is intended to support the definition of later activities in the project. More concretely, it will provide the narrative basis of the architecture requirements of the Risk Assessment framework, which will be developed in the context of WP4. There, the concrete components, and the input and output types to communicate between them will be documented. Moreover, Deliverable D1.2 provides the reference architecture that will guide the implementation of the three projected QR TPMs (software, hardware, and virtual) that will be developed within WP5, and that will constitute the basis for the integration of the platform demonstrators for the three Reference Scenarios in WP6.

1.3 Deliverable Structure

This deliverable is structured as follows. In Chapter 2, we review the key technologies that will constitute the foundations of the FutureTPM architecture. We will provide an overview of the current TPM 2.0 Specification, focusing on the details that will be relevant to the development of the QR TPM, a fundamental component of FutureTPM. Also, we review aspects of Risk Management and Security Policy Enforcement topics that will be used as a foundation to build the FutureTPM framework. In Chapter 3, we relate the Reference Scenarios' functional (technical) and non-functional (security and privacy) requirements with the QR TPM functional and security requirements. This mapping will enable us to identify the situations where a QR TPM can be a drop-in replaced of current TPM technologies, and which API calls must be modified in order to support the proposed QR primitives. Appendix A will provide most of the technical contents for this chapter. Chapter 4 is devoted to detail the overall architecture of the FutureTPM framework, its different phases, and the expected types of inputs and outputs between different components. This chapter will take as a basis the key technologies discussed in Chapter 2, and will examine and outline how these technologies need to be adapted and extended towards the QR Trusted Computing framework proposed in the project. In Chapter 5, we will describe more technical and logistic aspects related to the development phase of FutureTPM; we document the implementation guidelines that will be taken into consideration during the realization of the architecture. More specifically, the already agreed and setup development circle is analysed along with (a tentative list of) best practices that will be adopted. Finally, the conclusions will be drawn in Chapter 6.

Chapter 2 State of the Art and Key Technology Axes

This chapter is devoted to discuss the state-of-the-art of the key technology elements that will constitute the basic building blocks leveraged by the FutureTPM framework. In relation to the Background section of Deliverable D1.1 [1], this chapter provides a reference guide to the specific technologies that are embraced by the communities targeted by the FutureTPM project. We can classify these elements into three different domains:

- the **Trusted Platform Module** (TPM) and the **host device** containing the TPM;
- the **Risk Assessment** component;
- the **Security Policy Enforcement** component.

Noticeably, the core element that will serve as a basis to develop the proposed architecture is the **Trusted Platform Module** (TPM). As aforementioned, TPMs have been devised as a component of trust that enable to check the security posture of a device and provide mitigation measures against attacks such as not allowing for the device to boot up in case of a compromise. Furthermore, TPMs act as one of the main components handling the operations related to key management, such as key creation, storage, destruction and duplication. The TCG empowers the TPM to be the RTS and the RTR for a platform. The FutureTPM architecture, envisioned in the project, will take the current TPM 2.0 Specification [3] as the departing basis to define the architecture of the novel QR TPM. In what follows, Section 2.1 provides an overview of the underpinnings of the current architectural components from TPM 2.0 that are necessary to understand in order to transition to a QR version. As cryptography operations (including key management), authorizations and sessions, and platform attestation features are three essential features of a TPM, these topics and its related components will be addressed in separate sections, namely, Sections 2.2, 2.3, and 2.4.

Beyond the newly investigated QR TPM, the FutureTPM framework will provide a holistic TPM-based solution by also taking into consideration the complex threat landscape that the devices (e.g., processors, ASICS, etc.) hosting the TPM (through the TPM Software Stack (TSS)) pose to the cryptographic applications. **Risk Management** refers to the process that allows to assess, evaluate, measure and address potential security threats, in order to minimize the effect that they might pose to corporate and personal assets. Risk Assessment (RA) is a process within Risk Management that deals with the identification of threats, and determines their probability of occurrence, and their resulting impact. In the context of FutureTPM, Risk Assessment will take into consideration not only the QR TPM device itself, but also to the host device and all the remaining assets of the whole architectures envisioned by the three Reference Scenarios detailed in Deliverable D1.1 [1]. By leveraging the Risk Assessment component, a set of security policies can be defined in order to satisfy the security requirements of a given scenario, e.g., under what conditions is a certain private key allowed to sign the information of interest. Ensuring and imposing the correct usage of this set of rules is the task of the Security Policy Enforcement module.

A Risk Assessment Phase can be executed at design-time, but can also extend during run-time, to address initially unknown threats. In particular, during design-time, concrete indications regarding the security posture of a device will be provided in order to achieve the minimization of specific risks and vulnerabilities. However, considering the complexity of current cyber-physical systems where the deployed software may be updated to a more sophisticated version with new capabilities, the threat model has to take into consideration, e.g., zero day attacks that were not evaluated in the Risk Assessment Phase at design-time. Therefore, the former will essentially dictate the policies to be enforced by the **Security Policy Enforcement** module to mitigate the identified risks whereas the latter will allow the dynamic update of the already defined policies as a response to any newly identified attacks. Section 2.5 will provide an overview on current topics for Risk Assessment and its link with Policy Enforcement, in connection with TPM and the TPM Software Stack (TSS).

2.1 Current TPM 2.0 Architecture

TPM 2.0 evolved as the second major version of TPM, being built upon the previous version, TPM 1.2 [9], [10], [11]. The last revision of the TPM 2.0 Specification dates from September 2016 (and this is the one that will be used as the baseline in the context of FutureTPM). The main specification is structured into a set of four (extremely) detailed documents:

- **Part 1 – Architecture** [3]: This document describes the TPM operations in detail (e.g., how to create sessions, and all its variation types), and the rationale behind the TPM design.
- **Part 2 – Structures** [5]: This document presents a description of the data types, constants, and command returning values and error definitions.
- **Part 3 – Commands** [6]: This document presents the commands of the TPM and error conditions in detail.
- **Part 4 – Supporting routines** [7]: This document contains code for the supporting routines used in Part 3.

As introduced in Chapter 1, this common four-part library is then augmented trough platform-specific profiles, suitable for different types of devices. For example, there are specifications for PCs and server platforms [12], mobile devices [13], and information technology systems within a vehicle [14]. These profiles instantiate and define what parts of the library are mandatory, optional, or banned, what are the values and properties of the different TPM components, and detail other technical requirements for that platform. The FutureTPM QR TPM architecture will be mostly driven by the Main TPM 2.0 Specification [3], particularized through the PC Client Platform Specification [12].

A brief introduction of the TPM 2.0 architecture was already given in Deliverable D1.1 [1]. An overview of this conventional TPM architecture is shown in Figure 2. In a nutshell, its architecture consists of: (*i*) a cryptographic module (including private/public key encryption, digital signatures, hash functions and MACs), (*ii*) a random number generator, (*iii*) a protected storage region (volatile and non-volatile), (*iv*) a management component, and (*v*) an authorization component. A TPM is conceived as a System-on-a-Chip, hence all security-sensitive services are executed within a closed system. This, together with a cryptographic key storage, allows for the realization of the protected capabilities, as documented in the TCG Specification [3]. The command execution engine is a microcontroller which forms the basis of the TPM realization. This execution engine processes the incoming commands and controls the cryptographic engines accordingly. It is also intended for the realization of the cryptographic communication protocols.

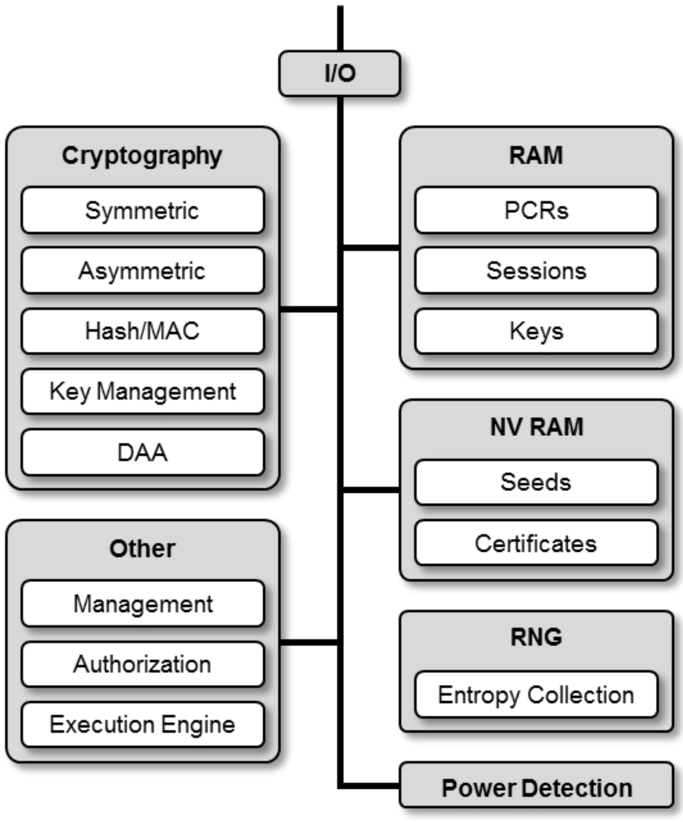


Figure 2: Overview of the TCG's TPM Architecture.

The **TPM Software Stack (TSS)** is a software specification that provides a standard API for accessing the functions of the TPM. In addition to the TCG's specifications cited above, we also refer the reader to [15], [16] and [17] for a more detailed discussion of TPM 2.0, and Trusted Computing in general.

In the following sections, we describe in more detail the main components of the TPM 2.0 architecture required to develop the FutureTPM QR TPM. The main intuition is that the architecture and API of the FutureTPM QR TPM should mirror where possible those of present day TPMs, in order to be backwards compatible and to allow for a smooth transition to the QR world.

2.1.1 TPM 2.0 Components and Interfaces

TPMs may be implemented under different forms (hardware-, software-, and virtual-TPM), each of which has its own components and interfaces [3], [18]. Nonetheless, the TCG has provided standardised mechanisms that adequately abstract specific interface implementations, so that any software using the TPM may be implemented in a portable manner [18], [19].

A **hardware-based TPM** may be comprised of the following components [20], [21]:

- a security-aware general-purpose processor;
- a highly reliable non-volatile (NV) memory;
- hardware accelerators for cryptographic operations such as private and public-key engines and hashing;
- hardware true random number generators, and
- sets of Platform Configuration Registers (PCRs).

Intra-chip communication may be achieved with protocols like: the Advanced Microcontroller Bus Architecture (AMBA) [22] that facilitates right-first-time development of embedded microcontroller products with one or more CPUs, GPUs or signal processors; the CoreConnect [23] which is a microprocessor bus-architecture for System-on-Chip (SoC) designs, planned to ease the integration and the reuse of processor, system, and peripheral cores within standard and custom SoC designs; the Open Core Protocol (OCP) [24] that defines a core-centric protocol that enables a bus-independent, configurable interface; and the Wishbone Bus [25] that is intended to let the parts of an integrated circuit communicate with each other. External communication between a TPM and a main processor may be achieved through protocols such as Low Pin Count (LPC), with a bandwidth of 20.48 Mbps, Serial Peripheral Interface (SPI) with bandwidths of about 10 Mbps, or Inter Integrated Circuit (I2C) bus with a maximum bandwidth of 3.4 Mbps [12], [26].

Software-based TPMs may either correspond to TPM emulators used for prototyping and testing or may be implemented in Trusted Execution Environments (TEEs) [27] to achieve a similar level of security as the one provided by hardware-TPMs.

In this latter case, the TPM NV memory makes use of the permanent storage that the host device (where the TPM is running) provides. Should the TEE provide a key that is only available to a certain Trusted Application (TA), the TPM might make use of it to ensure that no other application can access its storage. True random number generation (RNG) might make use of processor-specific instructions (such as Intel RdRand [28]), TEE specific functionality (such as ARM TrustZone TRNG [29]), or mechanisms provided by the Operating System (OS) (such as Linux' /dev/random [30]). The remaining TPM functionality, mostly related to cryptographic operations, may be implemented in software, make use of long-established libraries like OpenSSL [31], or use processor-specific instructions, like Intel's AES-NI [32]. Communication between the different components may be done through the passing of arguments and return codes in the case of the software components being embodied as functions, or through inter-process communication, when they are embodied as separate processes, using mechanisms like sockets, pipes and Remote Procedure Calls (RPCs). External communication is achieved through generic process communication in the case of software emulators or through platform specific mechanisms, like ARM's SMC instruction [33], when exploiting a TEE.

A **virtual TPM** implementation makes use of both hardware and software TPMs' components and interfaces. By supporting its security on a hardware TPM, it is able to provide a highly secure TPM interface to virtual machines [34].

2.1.1.1 TPM Software Stack (TSS) API

A client can abstract the previous implementations details pertaining to the external communication of a TPM by making use of the TSS. The TSS is a software specification that provides a standard API for accessing the functions of the TPM. Application developers can use this software specification to develop inter-operable client applications for more tamper-resistant computing.

In this subsection we provide an overview of the TSS API layers from the highest level of abstraction to the lowest: **Feature API (FAPI), Enhanced System API (ESAPI), System API (SAPI), TPM Command Transmission Interface (TCTI), TPM Access Broker (TAB) and Resource Manager (RM)**. The TSS specifies the software layer for application developers to use functions provided by a TPM [18], [19]. Figure 3 illustrates the TSS software stack.

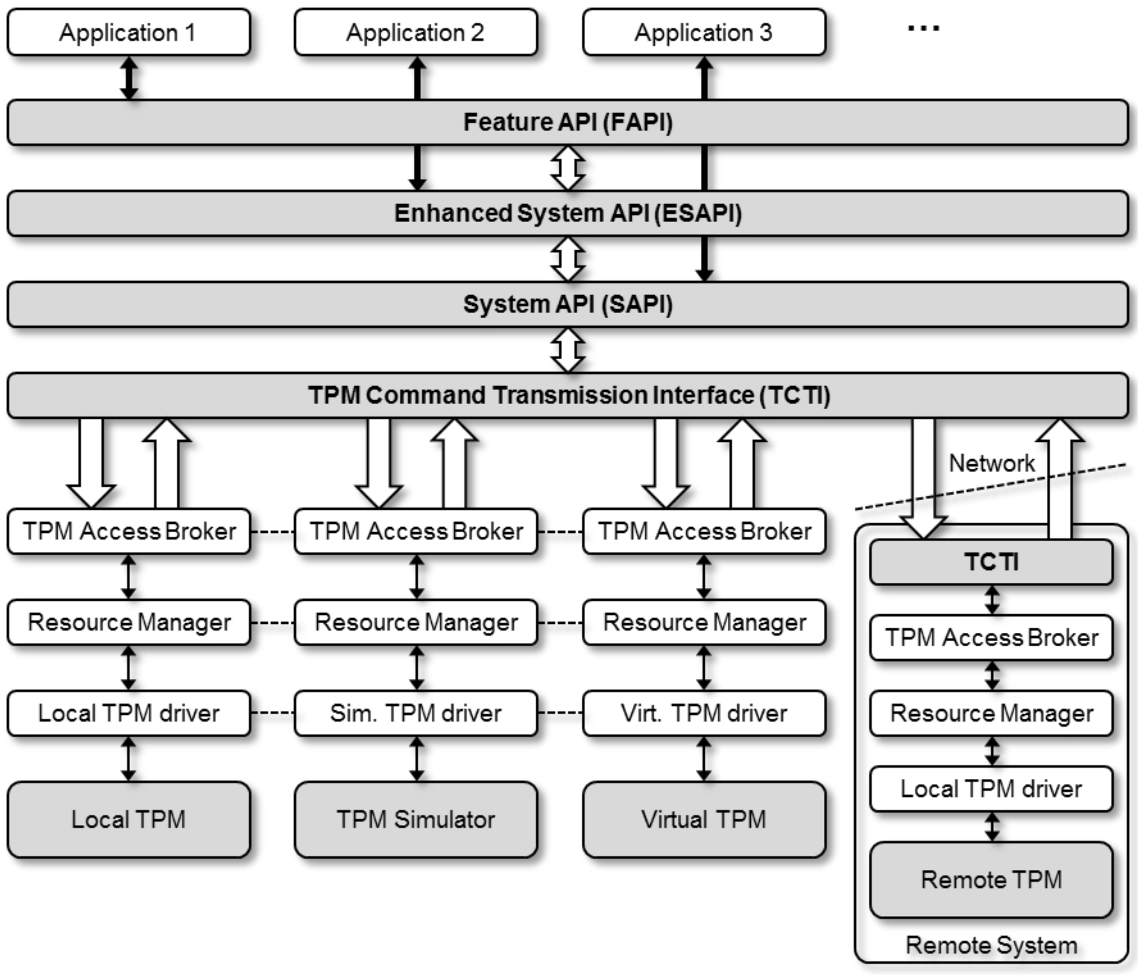


Figure 3: TSS diagram [35].

- **Feature API (FAPI):** This is meant to be a very high-level API, aimed at having commands in it that will allow 80% of the programmers who write a program using the TPM to find everything they want in the specification [36].
- **Enhanced System API (ESAPI):** The ESAPI is an interface that is intended to sit directly above the SAPI. The primary purpose of the ESAPI is to reduce the programming complexity of applications that desire to send individual “system level” TPM calls to the TPM, but that also require cryptographic operations on the data being passed to and from the TPM. In particular, applications that wish to utilize secure sessions to perform Hash-based Message Authentication Code (HMAC) operations, parameter encryption, parameter decryption, TPM command audit and TPM policy operations could benefit from using the ESAPI. Additionally, context and object management are provided by the ESAPI [37].
- **System API (SAPI):** The SAPI provides a programming interface that provides access to all the functionality of the TPM while performing the necessary marshaling (convert commands to byte streams) and unmarshaling operations (convert byte streams to commands) on the parameters of TPM commands [18].
- **TPM Command Transmission Interface (TCTI):** TCTI provides a generic interface to a wide variety of transport methods that could be used to communicate to the TPM. TCTI offers

functionality to transmit TPM command packets, receive responses, terminate a connection, cancel a command, provide handles (which can be used by the TCTI to notify a caller that a response is available for reception), and set locality. It handles the communication to and from the lower layers of the TSS, providing a common API that abstracts device drivers. Applications can be written to send binary streams of command data to the TCTI and receive binary data responses from it. This is like programming in assembly [19].

- **TPM Access Broker (TAB):** The TAB controls multi-process synchronization to the TPM. Basically, it allows multiple processes to access the TPM without stomping on each other [38].
- **Resource Manager (RM):** The RM acts in a manner similar to the virtual memory manager in an OS. Because TPMs generally have very limited on-board memory, objects, sessions, and sequences need to be swapped from the TPM to and from memory to allow TPM commands to execute [38].

As aforementioned in Chapter 1, one of the main goals of FutureTPM is to enable a 'smooth transition' between QR and non-QR cryptographic services (provided by the TPM) which entails the provision of a QR TPM capable of easily replacing the current TPM platform (plug-and-play approach). Thus, updates will be performed to only those layers that are necessary to support the newly identified QR crypto primitives. As will be documented in more detail in Chapter 3, most of the changes required will be instantiated at the SAPI/TCTI layers since these provide the core (low-level) interfaces with all the necessary TPM commands for providing the various crypto primitives; thus, in what follows we will focus on the underpinnings of these two layers. Of course, adequate measures will be taken so that any changes are successfully reflected to the upper layers, and especially the FAPI which is the one that provides the 'glue interface' with the programmers/analysts.

A **TCTI context** is a data structure containing the version of the current TCTI implementation, along with pointers to the functions offered by the TCTI. A pointer to this structure is passed to the SAPI, which can cast it to a TCTI version structure, containing the first two fields of the TCTI context structure. One of the two fields corresponds to the version offered by the TCTI. Each newer version must at least contain all fields of the previous versions in the same order. This allows the SAPI to safely cast the TCTI context to any version not greater than the value in the version field. A SAPI implementation makes use of the TCTI function pointers to offer its functionality.

The **SAPI** provides functions to initialise and finalise a SAPI command context. Furthermore, command-specific functions, Tss2_Sys_<COMMAND>_Prepare, Tss2_Sys_<COMMAND>_Complete and one-call Tss2_Sys_<COMMAND> are made available (wherein <COMMAND> denotes the name of a specific command); as well as other functions that aid with the execution of TPM commands.

The Tss2_Sys_<COMMAND>_Prepare function resets the SAPI command context and makes it ready for next flow of command functions. Functions Tss2_Sys_GetDecryptParam and Tss2_Sys_SetDecryptParam support decrypt sessions by allowing the caller to get access to the unencrypted command parameters, encrypt them, and then set the encrypted command parameters in the byte stream before sending the command. Tss2_Sys_GetCpBuffer underpins the computation of command authorisation Hash-based Message Authentication Codes (HMACs). Tss2_SetCmdAuths sets command authorisation area parameters, including nonces, session attributes, passwords and command HMACs. The pair Tss2_Sys_ExecuteAsync and Tss2_Sys_ExecuteFinish can be used to asynchronously execute a command; while Tss2_Sys_Execute executes a command synchronously. Upon receiving a response from the TPM: Tss2_Sys_GetCommandCode supports the computation of a command HMAC; Tss2_Sys_GetRspAuths gets the response authorisation data; Tss2_Sys_GetEncryptParam and Tss2_Sys_SetEncryptParam enable getting the encrypted result, decrypting it, and setting the decrypted results in the byte stream; Tss2_Sys_<COMMAND>_Complete unmarshals the decrypted results; and Tss2_Sys_GetRpBuffer returns a pointer to the unmarshalled results. In some cases, the previously described flow to issue TPM commands can be replaced with a single call to a Tss2_Sys_<COMMAND> function.

2.1.2 Commands and Data Communication Architecture

In the following, we give an overview of the command architecture for the TPM 2.0, and its associated data structures and data communication architecture.

2.1.2.1 Commands and Responses

As pointed out in Section 2.1 above, the TPM command and response structures are described in the TPM 2.0 Specification [3], [5], [6]. A TPM command is a byte stream composed of generic and command-dependent metadata as well as of command-dependent parameters. After command execution, the TPM responds with a response byte stream.

A **command** consists of

1. a *command header*, which itself is comprised of
   a. a *tag* which indicates whether the command contains sessions (i.e., an authorization area) or not (see Section 2.3 for more details on sessions),
      - TPM_ST_SESSIONS indicates the presence of an authorization area,
      - TPM_ST_NO_SESSIONS indicates that the authorization area is empty,

   b. a field *commandSize* indicating the overall size of the command byte stream (incl. the header),
   c. a field *commandCode* holding the identifier of the TPM command to be executed.
2. a *handle area* containing up to three handles (a handle uniquely identifies a resource in the TPM memory)
3. a 32-bit value *authorizationSize* holding the size of the subsequent authorization area, in particular indicating the number of sessions treated by the authorization area,
4. a so-called *authorization area*, which contains session data for up to three sessions. Each session is represented by an authorization structure, which contains authorization data, per-command session use modifiers and session state information for communication with the application. In particular, it is composed of
   a. a *session handle*, a four-byte value holding the associated session number;
   b. a *nonce size* field followed, if present, by a *nonce*, a byte array holding a nonce chosen by the caller (used for HMAC authorization to prevent replay attacks);
   c. the *session attributes* holding information on the session usage;
   d. an *authorization size* field followed, if present, by an *authorization* field containing – depending on the session type - either an HMAC or a password.
5. a command-dependent *parameter area*.

The structure of a TPM command byte stream is depicted in Figure 4 below.

Command Header	Handle Area	*authorization Size*	Authorization Area	Parameter Area
tag | commandSize | commandCode			session handle | size nonce | nonce | session attributes | size authorization | authorization	

Figure 4: Structure of a TPM command.

A **response** consists of:

1. a *response header*, which is composed of
   a. a *tag* identifying whether the response contains sessions or not (i.e. an authorization area):
      - TPM_ST_SESSIONS indicates the presence of an authorization area
      - TPM_ST_NO_SESSIONS indicates that the authorization area is empty

   b. a field *responseSize* indicating the overall size of the response byte stream (incl. the header)

   c. a field *responseCode* showing whether the TPM command execution was successful (TPM_RC_SUCCESS in this case) or an error identifier otherwise.
2. a *handle area* containing up to three handles, which are only present on success
3. a *parameter area* which contains the command-dependent response parameters
4. an authorization area that contains the response session data *area*, which contains parameters of up to three sessions. Furthermore, it contains authorization data, per-command session use modifiers and session state information for communication with the application. In particular, it is composed of
   a. a *nonce size* field followed, if present, by a *nonce*, a byte array holding a nonce chosen by the TPM (used for HMAC authorization to prevent replay attacks);
   b. the *session attributes* holding information on the session usage;
   c. an *authorization size* field followed, if present, by an *authorization* field containing – depending on the session type - either an HMAC or a password.

The structure of a TPM response byte stream is depicted in Figure 5 below.

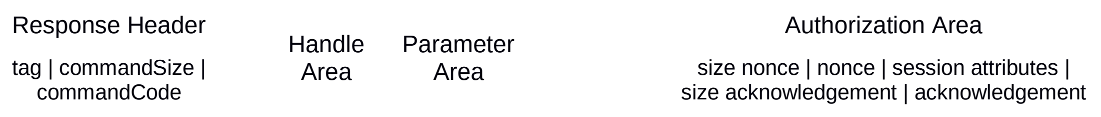

Response Header
tag | commandSize | commandCode
Handle Area
Parameter Area
Authorization Area
size nonce | nonce | session attributes | size acknowledgement | acknowledgement

Figure 5: Structure of a TPM response.

Thereby, in case of a successful command execution, the number of authorization structures in the authorization area of a response is equal to the number of authorization structures in command and zero otherwise.

There are two command types: authorized and unauthorized commands. An example for the latter is the command TPM2_Startup as it is the first command that has to be sent to a TPM. This command has neither session (indicated with tag TPM_ST_NO_SESSIONS) nor return parameters. An example for an authorized command is TPM2_Create, which is described below.

Subsequently, we will detail some often-used TPM commands in Table 1 below.

Table 1: List of frequent TPM 2.0 commands.

Command	Code	Description
TPM2_Startup	TPM_CC_Startup 0x00000144	Start up TPM.
TPM2_NV_Read	TPM_CC_NV_Read 0x0000014e	Read data from the TPM's NVM, respectively (given an NV index).
TPM2_NV_Write	TPM_CC_NV_Write 0x00000137	Write data to the TPM's NVM, respectively (given an NV index).
TPM2_Create	TPM_CC_Create 0x00000153	Generic command to generate all types of keys.
TPM2_Create_Primary	TPM_CC_CreatePrimary 0x00000131	Generic command to generate all types of primary keys, which are derived form a primary seed.

Command	Code	Description
TPM2_Load	TPM_CC_Load 0x00000157	Loads wrapped private keys.
TPM2_LoadExternal	TPM_CC_LoadExternal 0x00000167	Loads publics keys
TPM2_StartAuthSession	TPM_CC_StartAuthSession 0x00000176	Establishes parameters which will be used for the authorizations.
TPM2_RSA_Encrypt	TPM_CC_RSA_Encrypt 0x00000174	Perform RSA encryption operation.
TPM2_RSA_Decrypt	TPM_CC_RSA_Decrypt 0x00000159	Perform RSA decryption operation.
TPM2_EC_Ephemeral	TPM_CC_EC_Ephemeral 0x0000018e	Perform 1st key agreement (ECDH/ECMQV/SM2) phase by generating an ephemeral key.
TPM2_ZGen_2Phase	TPM_CC_ZGen_2Phase 0x0000018d	Perform 2nd key agreement (ECDH/ECMQV/SM2) phase.
TPM2_EncryptDecrypt	TPM_CC_EncryptDecrypt 0x00000164	Generic command to perform symmetric key en- and decryption.
TPM2_Hash	TPM_CC_Hash 0x0000017d	Compute a hash.
TPM2_HashSequenceStart	TPM_CC_HashSequenceStart 0x00000186	Start a hashing computation.
TPM2_HMAC	TPM_CC_HMAC 0x00000155	Compute an HMAC.
TPM2_HMAC_Start	TPM_CC_HMAC_Start 0x0000015b	Start an HMAC computation.
TPM2_SequenceUpdate	TPM_CC_SequenceUpdate 0x0000015c	Update the hash/HMAC computation.
TPM2_SequenceComplete	TPM_CC_SequenceComplete 0x0000013e	Finalize the hash/HMAC computation.
TPM2_Sign	TPM_CC_Sign 0x0000015d	Generic command to sign a message (the signature algorithm must be specified via the parameters).
TPM2_VerifySignature	TPM_CC_VerifySignature 0x00000177	Generic command to verify a message-signature pair.

2.1.2.2 Data Handling and Common Data Structures

In order to be able to encode and decode byte streams when communicating with the TPM, it is necessary to understand the TPM C data structures and the way data gets encoded and decoded.

2.1.2.2.1 Data Encoding and Decoding

TPMs are often connected via slow interfaces (like SPI or I²C) and internal C data structures are often larger than necessary (e.g. unions with elements of different size). Thus, the size of data sent to or received from a TPM is always optimized beforehand and, if not already the case, additionally converted to big-endian format (*canonicalization*). Response data sent from the TPM is also canonicalized.

Moreover, the conversion of data in byte stream to C structure format is called *unmarshalling*, whereas the inverse conversion from C structure format to a byte stream is called *marshalling*.

2.1.2.2.2 Common Data Structures

All TPM C data structures prefixed with TPM2B_ are byte buffers with a preceding unsigned 16-bit integer holding the buffer size. For example, the data structures for message digests and Rivest-Shamir-Adleman (RSA) public keys are shown in the code listings below.

Code Listing 1: Example of a TPM2B data structure (TPM2B_DIGEST).

```
/* Table 2:73 - Definition of TPM2B_DIGEST Structure  */
typedef union {
    struct {
       UINT16                  size;
       BYTE                    buffer[sizeof(TPMU_HA)];
    }            t;
    TPM2B        b;

} TPM2B_DIGEST;
```

Code Listing 2: Example of a TPM2B data structure (TPM2B_PUBLIC_KEY_RSA).

```
/* Table 2:165 - Definition of TPM2B_PUBLIC_KEY_RSA Structure  */
typedef union {
    struct {
       UINT16                  size;
       BYTE                    buffer[MAX_RSA_KEY_BYTES];
    }            t;
    TPM2B        b;

} TPM2B_PUBLIC_KEY_RSA;
```

New (public) key structures for QR algorithms will be drafted in a similar fashion in the FutureTPM project.

Another common data structure are unions, which are prefixed with TPM2_U.

2.1.3 Entities

A TPM entity is any item that can be directly referenced with a TPM handle [15]. TPM handles are used to uniquely identify resources that occupy TPM memory, either RAM or NV. So, an entity is a generic term that is used to define a large group of resources which are utilized by the implementation. They are divided in the following categories: **permanent entities**, **NV entities**, **objects** and **volatile entities** [3]**,** [15]. We are now going to present these categories in detail alongside with some examples.

Permanent entities are resources which are preinstalled in the TPM and cannot be created or deleted, with their handles being defined within the TPM 2.0 Specification [3]. Permanent entities include:

- **E1.1** Persistent Hierarchies
- **E1.2** Ephemeral Hierarchy
- **E1.3** Dictionary Attack Lockout Reset
- **E1.4** Platform Configuration Registers
- **E1.5** Reserved Handles
- **E1.6** Password Authorization Session
- **E1.7** Platform NV Enable

A **hierarchy** is a collection of entities that are handled as a group and have common characteristics. The TPM 2.0 Specification defines three **persistent hierarchies**, namely: platform, storage and endorsement. Each one of these hierarchies, has separate access permissions through authorization values and policies that will be presented in Section 2.1.4 below. There is also an **ephemeral hierarchy** that is used when an external entity is using the TPM for cryptographic operations. The ephemeral hierarchy has no authorization values or policies as it should be usable by anyone. All these hierarchies cannot be created or deleted, but they can be cleared of their contents with proper authorization.

The **dictionary attack lockout reset** is used by a mechanism that prevents any authentication attempts when too many failed tries have been made. This entity controls the state of the lock and, like the hierarchies, has an authorization value and a policy to be handled by the administrator of this entity. Although technically it is a hierarchy, this entity has no objects or keys and it serves as a reset switch that disables the dictionary attack lock mechanism or clears the owner persistent hierarchy. This mechanism is utilized by Dell in its laptops to prevent dictionary attacks [39], by the Microsoft Bitlocker technology [40] and by many other technology vendors.

Platform Configuration Registers, which will be discussed in detail in Section 2.1.5 below, are a core entity of the TPM. With PCRs the TPM can attest to the device's state and provide a provable list of the functions that have run in the TPM until now. PCRs can be read without any authentication, although they have authorization values and policies. The number of PCRs within the TPM depends on the implementation, (e.g. 24 PCRs in the PC Client Platform Specification [12]) but there should be at least one bank of PCRs that supports either SHA-1 or SHA-256 at boot time. The number of PCRs installed in a TPM, is defined by the manufacturer and there is no way to delete or create them. PCRs are core components of many security designs, in the work of Yang et al. [41] PCRs are proposed as a safe storage space to hold attestable information for their scheme of memory attestation for wireless sensor nodes.

Vendors can choose to install some predefined **reserved handles** in the TPM that are meant to be used in case of a critical security failure. These handles should be used to testify to the state of the software that is stored in the TPM. Although the ability to store these handles is defined in the TPM 2.0 Specification, there is no evidence of any vendor using them.

In order for callers to provide authorization, they need to authenticate themselves first. The TPM provides many ways for users to authorize commands, but the specification mandates that the **password authorization session** is a permanent entity and as such it cannot be deleted. This specific authorization session uses passwords to authorize commands, as opposed to advanced sessions that can use HMAC and advanced authentication like biometrics together with passwords. In [42] an automated proof for authorization protocols that uses TPM authorization sessions is presented. Also, in [43] a formal analysis of the TPM 2.0 enhanced authorization scheme is made.

The NV memory indices can belong to both the platform hierarchy and/or to the storage hierarchy. The indices that belong to storage, are managed by the storage hierarchy and its settings. On the other hand, platform indices are managed by a separate control: the **platform NV enable** entity. This extra entity is used in order to have a more fine-grained control over the platform hierarchy and the platform NV indices. Next, we are going to analyse the NV entities.

The only NV entities that exist in the TPM are the **E2 NV indices**. An NV Index is a storage space that is defined by a user of the TPM. These indices belong to hierarchies and have authorization mechanisms (policies and values) just like all the entities. NV indices provide a highly configurable storage that can be tailored to support almost any scenario, they can be configured to provide PCR like functionality, counter functionality or even be generic bit fields. Similar to these entities, are the **E3 objects**, which are entities that can store either keys or data. Objects should not be confused with NV indices, since they are less configurable and exist to serve more specific purposes. They are also part of hierarchies and have authorization values and policies and they provide a mechanism to control what actions are performed on the object depending on the authorization of the actor.

Finally, there are the **E4 volatile entities** that can be either persistent or nonpersistent. The entities of the latter category never persist through power cycles, some examples are: E4.1 authorization sessions and E4.2 hash/HMAC event sequences. On the other hand, persistent entities, survive the event of a TPM reset. A user can configure an object to be persistent in order to optimize loading times, but he must consider that the memory available for persistent entities is limited. Some basic examples that can be persistent entities are: E4.3 primary storage keys, E4.4 primary restricted signing keys and E4.5 endorsement keys.

2.1.3.1 Entity Names

The Name of an entity is its unique identifier. The handle associated with an object may change, but the name of an object remains constant as it is generated by the data within the entity. The name associated with an NV Index is correlated with changes to the attributes of the index. The naming concept was introduced in the TPM 2.0 Specification [3], to solve a security problem identified in the earlier TPM version. In the TPM 1.2 Specification [9], entities had only handles that were practically pointers to memory allocated and used to store this entity. With the purpose of saving memory, a key manager was used within the TPM to reallocate entities for storage space optimization. Moreover, a middleware handled these changes, and updated all the incoming commands, changing the entities referenced to point to the correct location. If someone decided to give the same password to more than one entity, then it would be possible for one of those entities to be substituted for another by an attacker, and the attacker could then authorize the wrong entity to be used in a command. Although this scenario seems unlikely, the TPM was designed to provide the highest levels of trust, and so the naming convention was introduced to provide the ability to uniquely identify each entity with the use of hashes.

2.1.4 Hierarchies

A hierarchy is a set of entities and resources contained in the TPM that are related and independently managed as a group. The TPM 1.2 Specification [9] only defined one hierarchy, namely, the Storage Hierarchy. This approach had the drawback that everything is under the control of the owner. If the TPM is not enabled, activated, and owned; there isn’t much that can be done with it, so ultimately TPM 1.2 has only one administrator. It is only the owner, who can control both the security and privacy functions.

The TPM 2.0 Specification [3] defines up to four different hierarchies, which are related to different security domains. They are divided into two classes, as discussed in Section 2.1.3 above: *persistent* hierarchies (platform, endorsement and storage) and one *ephemeral* (null) hierarchy. The employment of the different hierarchies allows several use cases, such as using the TPM as a cryptoprocessor, enabling or disabling parts of the TPM, and separating privacy-sensitive and privacy-nonsensitive operations in different control domains.

2.1.4.1 Persistent Hierarchies

The TPM 2.0 Specification defines three persistent hierarchies, namely **Platform, Storage (or Owner), and Endorsement Hierarchy**. The main features of them are:

- they can be independently enabled or disabled;
- each hierarchy has an independent authorization and a policy;

- each hierarchy has an associated persistent Primary Seed from which keys and data objects are derived (Primary Seeds are generated from the TPM internal RNG);
- each can have primary keys from which descendant keys can be created.

The authorization, policy, Primary Seed and logical switch to enable each hierarchy are referenced in the TPM 2.0 Specification as follows:

- Platform Hierarchy: *platformAuth* / *platformPolicy* / PPS / *phEnable*.
- Storage Hierarchy: *ownerAuth* / *ownerPolicy* / SPS / *shEnable*.
- Endorsement Hierarchy: *endorsementAuth* / *endorsementPolicy* / EPS / *ehEnable*.

Platform Hierarchy. This hierarchy is intended to be under control of the platform manufacturer, represented by the BIOS/ Unified Extensible Firmware Interface (UEFI), in relation to the functions that protect the integrity of the platform and firmware services. Manufacturers use the Platform Hierarchy to protect the update mechanisms of the roots of trust of the platform in order to fulfil NIST SP 800-147. When the platform boots, and upon every TPM2_Startup command execution, the platform hierarchy is enabled (i.e., *phEnable* is set to 1), *platformAuth* and *platformPolicy* are set to the empty buffer. The intent is that the platform firmware will generate a strong platform authorization value (and optionally install its policy). Unlike the other hierarchies, which may have a human enter an authorization value, the platform authorization is entered by the platform firmware. When the BIOS goes through a full initialization it has no memory of any previous authorization values. Therefore, there is no reason to have the authorization persist (and to find a secure place to store it) rather than regenerate it each time.

The Platform Hierarchy is typically used to authenticate software as part of the UEFI secure boot process, for example, by maintaining a public key in a TPM NV index within this hierarchy. During boot, the platform firmware uses this key to verify the signature to authenticate the software. The Platform Primary Seed is used to derivate any required Platform Key, which is of exclusive use by platform firmware, and should not be made available to user-installable software such as the OS or applications.

In addition to its own enabling/disabling switch *phEnable*, this hierarchy has an additional control switch, namely *phEnableNV* (which is also set to 1 upon calling TPM2_Startup). This control enables to independently turn on or off the access to the NV memory for that particular hierarchy, whereas *phEnable* is used to control the rest of the entities within the hierarchy. That is, the Platform Hierarchy can be disabled while still permitting access to its NV memory space.

Certain security-related and administrative operations are only available within the Platform Hierarchy, and are not available to an ordinary user of the TPM. Examples of commands that can only be executed in the context of this hierarchy are shown in Table 2.

Table 2: Examples of TPM 2.0 commands reserved to be used within the Platform Hierarchy.

Command	Description
TPM2_PCR_Allocate	Used to set the desired PCR allocation of PCR and algorithms.
TPM2_ChangePPS	Used to replace the current Platform Primary Seed.
TPM2_ChangeEPS	Used to replace the current Endorsement Primary Seed.
TPM2_ChangeSPS	Used to replace the current Storage Primary Seed.
TPM2_Clear	Removes all TPM context associated with a specific owner.
TPM2_SetAlgorithmSet	Allows the platform to change the set of algorithms that are used by the TPM.

Command	Description
TPM2_PP_Commands	Used to determine which commands require assertion of Physical Presence.
TPM2_PCR_SetAuthPolicy	Used to associate a policy with a PCR or group of PCRs. The policy determines the conditions under which a PCR may be extended or reset.

Storage Hierarchy. This hierarchy is intended to be used by the platform owner, whether it is the end user or the IT department in an enterprise scenario. This hierarchy is intended for non-privacy sensitive operations such as generation and storage of user application keys. As opposed to the Platform Hierarchy, the authorization and policy, *ownerAuth* / *ownerPolicy*, persist through reboots, and may be explicitly changed through designated operations. However, the intent is that they be set and rarely changed (e.g., after credential compromise).

One of the main differences compared to TPM 1.2 is that multiple key hierarchies are allowed to coexist within the Storage Hierarchy. Each one of these key hierarchies are rooted at a primary key called Storage Root Keys (SRKs), where parent keys protect subsequent child keys. In addition to the creation and managing of these storage key hierarchies, other important feature available to the owner is the management of NV memory: create, update, read, and delete objects. The TPM owner can clear all the storage hierarchies, changing the SPS and effectively erasing all storage hierarchies of keys.

In Table 3 below we show some examples of typical commands available to the platform owner through the Storage Hierarchy.

Table 3: Examples of TPM 2.0 commands available under the Storage Hierarchy.

Command	Description
TPM2_CreatePrimary	Used to create primary objects (e.g., Storage Root Keys).
TPM2_CreateLoaded	Used to create any type of object (Primary, Ordinary, or Derived). In this hierarchy, it will be used to create, e.g., child encryption keys.
TPM2_EvictControl	Allows certain Transient Objects to be made persistent or a persistent object to be evicted.
TPM2_NV_DefineSpace	Used to reserve space to hold data associated within the NV storage.
TPM2_NV_UndefineSpace	Removes data permanently from the NV storage.
TPM2_HierarchyControl	Enables and disables use of the hierarchy and its associated NV storage.
TPM2_HierarchyChangeAuth	Allows the authorization secret for the hierarchy or lockout to be changed.

Endorsement Hierarchy. The endorsement hierarchy is intended to be under the control of the privacy administrator. As above, this can be either the end user or an IT department. This is the hierarchy of choice when the user has privacy concerns. A user with high privacy concerns can disable the endorsement hierarchy while still using the storage hierarchy for TPM applications and permitting the platform software to use the TPM. As in the storage hierarchy, the authorization and policy for this hierarchy, *endorsementAuth* / *endorsementPolicy*, persist through reboots and can be changed through designated commands.

TPM and platform vendors certify that primary keys in this hierarchy, namely, Endorsement Keys (EK), are constrained to an authentic TPM attached to an authentic platform. Similarly, as in the other hierarchies, EKs are primary objects derived using a public template from the EPS, and constitute cryptographically verifiable identities for the RTR. Therefore, manufactures need not install EKs into the TPM, as they can be replicated at owner's discretion by means of the EPS.

If an EK is a signing key that directly certifies other keys, correlation between the different signatures can be traced back to that EK using the certificate chain, and in turn to the original TPM. This is the main source of privacy concerns. Therefore, EKs are typically instantiated as encryption keys, and are used to create a hierarchy of descendant keys named Attestation Identity Keys (AIKs). These AIKs can then be certified through a Privacy Certification Authority, which is trusted not to leak any correlation between EKs and AIKs. The certificate obtained, therefore, ensures that an AIK belongs to a TPM, but not to which one.

When compared to the other persistent hierarchies, the privacy administrator has a more limited domain of operations available through the Endorsement Hierarchy: essentially, only the management of EKs/AIKs, and enabling the availability of the hierarchy. Some examples of commands to carry out operations in the context of this hierarchy can be found in Table 4.

Table 4: Examples of TPM 2.0 commands available under the Endorsement Hierarchy.

Command	**Description**
TPM2_CreatePrimary	Used to create primary objects (Endorsement Keys).
TPM2_CreateLoaded	Used to create any type of object (Primary, Ordinary, or Derived). In this hierarchy, it will be used to create AIKs.
TPM2_HierarchyControl	Enables and disables use of the hierarchy and its associated NV storage.
TPM2_HierarchyChangeAuth	Allows the authorization secret for the hierarchy or lockout to be changed.

2.1.4.2 Ephemeral (Null) Hierarchy

The aim of the Null Hierarchy is to use the TPM as a cryptoprocessor, and benefit from the already implemented crypto algorithms present in it. As with the persistent hierarchies, the Null Hierarchy also allows the creation of primary keys and complete hierarchies of descendant objects. The Null Primary Seed is set to a random value on every TPM reset. Therefore, objects created in this hierarchy cannot be made persistent, as they will not be able to be reconstructed after a TPM reset. Moreover, this hierarchy cannot be disabled, and it contains an authorization value that is a zero-length password, and an empty policy.

Using a TPM as a cryptoprocessor is not one of the main use cases envisioned for these devices. As an extremely resource-constrained device, its performance executing intensive cryptographic operations might not be acceptable for some applications. It is therefore not advised that the TPM be used for bulk encryption. However, there are certain scenarios that can benefit from this use case. For example,

- Early boot resource-constrained environments, which may require the use of cryptographic operations.

- Applications where the cost in performance is affordable in comparison, e.g., to produce commercial encryption software.
- Applications that require certified implementations.
- Encryption of small, high-valuable data where the encryption keys are externally stored.

Typical operations executed within this hierarchy are common cryptographic operations such as generating random numbers, computing hash and MACs, or symmetric/asymmetric encryption and decryption. Table 5 below shows some usual commands normally used in the context of this hierarchy.

Table 5: Examples of TPM 2.0 commands available under the Null Hierarchy.

Command	Description
TPM2_Hash	Performs a hash operation on a data buffer and returns the results.
TPM2_HMAC	Performs an HMAC on the supplied data using the indicated hash algorithm.
TPM2_EncryptDecrypt2	Performs symmetric encryption or decryption.
TPM2_GetRandom	Returns the requested number of bytes from the random number generator.
TPM2_RSA_Encrypt	Performs RSA encryption.
TPM2_RSA_Decrypt	Performs RSA decryption.

2.1.5 PCRs

Platform Configuration Registers (PCRs) are one of the essential features of a TPM that allows it to act as an RTR. A PCR is a memory register that can store the entire output of a hash algorithm (e.g., 256 bits for SHA-256), and provides a method to cryptographically record a log of measurements corresponding to the software states that affect the security condition of a platform. In the context of Trusted Computing, such measurements are initiated by the RTM, and are expected to take place, at least, during the boot phase of the collection of system resources responsible of maintaining the security policy of the system.

PCRs cannot be modified arbitrarily. When an entry is appended to a log of measurements, the TPM receives a copy of such entry (or a digest of the data described by the log), and the data sent to the TPM is employed to update the value of the PCR to its next value. This operation is executed using a hash algorithm as described in Section 2.1.5.1 below, and is known as PCR extend operation. The TPM can provide an attestation of the value of a PCR (or a set of PCRs) upon request, corresponding to the cumulative value of log measurements up to that point. This is used to verify the contents of the log. This attested measurement allows an independent entity to determine whether the platform has been compromised or not.

PCRs can also be used in conjunction with the authorization mechanisms to restrict access to a TPM-protected object, e.g., a decryption key. If certain PCRs do not have the required values, then the TPM will not allow access to that object. A well-known example of this use case is Microsoft BitLocker full disk encryption [44]. BitLocker is used in conjunction with a TPM to ensure that the integrity of the trusted boot path of a platform (e.g. BIOS and boot sector) is in a trustworthy state, in order to prevent most offline physical attacks and boot sector malware. That is, full-disk decryption keys are only released if the PCRs report an expected set of measurements after platform boot.

Theoretically, a platform requires only a single PCR to record its entire history of measurements. However, this would make difficult to evaluate the platform state at different stages, and typically several PCRs are allocated to the various software layers. For example, some PCRs measure the booting environment, others record the OS environment, yet others are devoted to application measurements. Therefore, an individual that cared about which OS was loaded, but not what the OS had done since it loaded, could restrict its data to the set of PCRs that represent the booting environment, and ignore the remaining PCRs.

The TPM 2.0 Specification [3] considers PCRs as Shielded Locations. That is, a location within the TPM that contains data that is secured from access by any entity other than the TPM itself, and which may be operated on only by a Protected Capability. Within the architecture of TPM 2.0, PCRs are regarded as part of volatile memory. However, the specification allows a certain degree of flexibility concerning the persistence of these registers: it is an implementation-dependent decision if the values of PCRs are retained in TPM RAM after it is powered off. Moreover, PCRs need not be maintained in RAM. It is possible, but not advisable, that they are kept in NV memory, where consideration must be made for the possible impact on TPM performance during the critical boot cycle, which could also subject the NV memory to wear-out. It is therefore recommended to use RAM for PCRs. If a TPM uses NV memory for PCRs, then the vendor is strongly recommended to provide a cache for the most recently used PCRs. Again, PCR persistence is an optional feature, and as far as the specification is concerned, PCRs are assumed to be reset to their initial values after the power is removed.

PCRs are grouped into PCR banks. A PCR bank is simply a collection of PCRs that are extended with the same hash algorithm. PCR banks are identified by the hash algorithm used to perform the extend operation. This architecture is useful, for example, when a certain hash algorithm is required for legacy or compatibility applications, and a different hash algorithm is required to meet stronger security criteria. Not all PCR banks are required to have the same number of PCRs. However, the attributes of all PCR with the same index must be equal, except for the particular hash function employed. For example, if PCR[0] of a certain bank has an attribute that allows it to be reset by command TPM2_PCR_Reset, then this attribute applies to the PCR[0] of every bank. The specification requires that a TPM implement a PCR bank for each supported hash algorithm. However, it also allows a PCR bank be defined so that it contains no PCRs.

The PC Client Platform Specification [12] requires at least 24 PCRs and one bank, although it allows fewer than 24 PCRs per bank. This minimum is expected to be the actual number in PCs. Other applications such as vehicular TPMs may have more. Also, if only one bank of PCRs is supported, [12] requires that a conformant TPM use SHA-256 for this bank. If the platform supports more banks, it shall support both SHA-256, SHA-1, and it may support additional hash algorithms. Table 6 below shows the intended usage for PCRs in the PC Client Platform Specification.

Table 6: PC Client Platform Specification PCR usage.

PCR Index	Alias
0 – 15	SRTM
16	Debug
17	Locality 4
18	Locality 3
19	Locality 2
20	Locality 1

PCR Index	Alias
21	Dynamic OS Controlled
22	Dynamic OS Controlled
23	Application Specific

2.1.5.1 Localities

A **locality** is an assertion to the TPM to indicate the source of the command from within the platform. Locality can be thought of as a hardware-based authorization, and it was conceived as a way to allocate different security privileges to different memory address ranges. Each range corresponds to a different locality indication. In a PC platform, the locality is communicated by the platform's chip set as a signal on the Low Pin Count (LPC) bus to which the TPM is attached. A TPM has no idea, however, where the command originated and does not care: it simply trusts the locality indication supplied with the command.

Localities have a direct impact on the usage of PCRs: some PCRs can only be extended or reset by commands at a certain locality. For example, when the DRTM is executing, locality is set to 4 to each TPM command, indicating that the command was issued to the CPU by the DRTM firmware. When the DRTM passes control to user-provided code, but the CPU is still in secure mode, commands are sent with locality 3. Normal execution uses locality 0. Localities 1 and 2 are controlled by the OS, based on the memory page the code is being executed. Table 7 summarizes locality usages. Localities 1 and 2 are rarely used in today's systems, as they rely both on having high trust in the OS enforcement mechanisms, and on having a trusted TPM driver capable of operating at multiple localities.

Table 7: Types of localities.

Locality	Intended usage	Common usage
4	Trusted Hardware/DRTM	DRTM
3	Auxiliary Components/DRTM	Software launched by DRTM
2	Trusted OS	Not used
1	Trusted Applications	Not used
0	SRTM/Legacy	STRM, Default

2.1.5.2 PCR Operations

Below we summarize the set of PCR-related operations as defined in [3], focused on the PC Client Platform Specification [12].

Initialization. PCRs are reset to their default values upon reaching the following TPM operational state or invoking the following operations:

- TPM Reset (TPM2_Shutdown(CLEAR) + TPM2_Startup(CLEAR), or TPM2_Startup(CLEAR)). PCRs designated as being preserved will be restored to the state they had at the last TPM2_Shutdown(STATE).
- TPM Restart (TPM2_Shutdown(STATE) + TPM2_Startup(CLEAR)). Same behaviour as above for preserved PCRs.
- TPM2_PCR_Reset. Only when this operation is explicitly allowed by the referred PCR attributes. This operation requires authorization.

- Upon a Dynamic Root of Trust Measurement (DRTM) executed after calling TPM2_Startup. On each invocation of the DRTM sequence, the RTM must be in the same known state. For DRTM, the TPM will initialize one or more PCR to zero and then extend PCR[17] in each bank with the Hardware Core Root of Trust Measurement (H-CRTM) data accumulated in the H-CRTM Event Sequence.

Default values. Platform-specific implementations are allowed to choose from several options for defining the initial values of PCRs. The default values for all PCRs, except PCR[0] is either all bits set to 0 or all bits set to 1. For PCR[0], the default value can be all bits set to 0, all bits set to 1, the locality at which TPM2_Startup was received, or an indicator that the first measurement came from an H-CRTM.

Extend. The command TPM2_PCR_Extend is used to update the indicated PCR. Except for DRTM, authorization is required to extend a PCR. The extend operation works in the same way for all PCR, except for the particular hash algorithm associated with the PCR bank. The current contents of the PCR are concatenated with the measurement and processed in a hash algorithm, which creates an output of the same size as the PCR. The output is then stored again in the same PCR:

$$PCR.digest_{new} \coloneqq \mathrm{H}_{alg}(PCR.digest_{old} \parallel digest).$$

In more detail, the command TPM2_PCR_Extend has a handle referencing the PCR to be extended (that is, the PCR index), and a $digests$ parameter that contains one or more tagged digest values, i.e., identified by an algorithm ID. For each digest in $digests$, the PCR associated with the handle is extended into the bank identified by the tag alg as

$$PCR.digest_{new}[pcrNum][alg] \coloneqq \mathrm{H}_{alg}(PCR.digest_{old}[pcrNum][alg] \parallel digests[alg].buffer),$$

where

- H_{alg}: hash function associated to the PCR bank identified by alg.
- $PCR.digest$: digest value of the PCR.
- $pcrNum$: PCR numeric selector.
- alg: PCR algorithm selector for the digest. It will update the corresponding PCR of the bank identified by alg. If no digest value is specified for a bank, then the PCRs in that bank will not be modified.
- $digests[alg].buffer$: the data specific to bank alg to be extended.

The TPM maintains a *pcrUpdateCounter*, which is incremented each time a PCR is modified (either extended or reset). A platform-specific implementation may designate that updates of selected PCR will not cause a change to *pcrUpdateCounter*.

Recording events. Rather than extending PCRs with supplied computed hash values, the TPM supports extending PCRs with the actual value of the log entries. This can be done through two standard operations:

- A single call to TPM_PCR_Event, for events no larger than 1024 bytes. This command implicitly selects all PCR with the same index in all the banks.
- For longer messages, using the standard sequence TPM2_HashSequenceStart, TPM2_SequenceUpdate and TPM2_EventSequenceComplete. The last operation requires authorization.

In addition to the execution of the commands above, the TPM 2.0 Specification also states that a recording of an event can also be triggered from the TPM interface (and not from the command buffer) as the result of an H-CRTM. This is achieved through the sequence of indications _TPM_Hash_Start, _TPM_Hash_Data, and _TPM_Hash_End.

Reading. Retrieving values from PCRs can be achieved through the command TPM2_PCR_Read. This command receives as input a PCR selection structure named TPML_PCR_SELECTION. This structure is an array of lists, where each entry has a hash identifier indicating the PCR bank, and a bit field indicating the PCRs being selected in the bank. The response provides a PCR selection structure indicating the PCR values that are present in the return structure.

Attesting to PCR. There are cases where it is required that certain PCRs stay in a certain state. Concretely, the following commands require this functionality:

- TPM2_PolicyPCR: used to cause conditional access of a policy based on PCR.
- TPM2_Quote: to produce a report that evidence that the platform is in a certain state.

Instead of comparing the values of the PCRs with a list of values specifying the value for each one, the specification states that the comparison is made through the digest of the concatenation of values of the PCRs. The PCRs included in this digest are selected, again, via the TPMl_PCR_SELECTION structure.

Allocation. PCR allocation refers to the assignment of the desired PCR algorithms. This process is likely to be done once at most, if the default algorithm is to be changed. It is achieved through the TPM2_PCR_Allocate command, which requires Platform Authorization. The allocation will take effect upon the next TPM2_Startup(TPM_SU_CLEAR) execution, and will persist until a new allocation is executed. As above, the allocation structure to select the particular PCRs in this command is TPML_PCR_SELECTION.

2.1.5.3 PCR Attributes

Each PCR have an associated set of attributes (PCR Property Tags), which are defined in the TPM 2.0 Specification. However, which PCR indexes have which attributes is left to the platform-specific implementation. The attributes are shown in Table 8 below.

Table 8: PCR attributes.

PCR attribute	Description
TPM_PT_PCR_SAVE	Indicates that the PCR is saved and restored after executing TPM2_Shutdown.
TPM_PT_PCR_EXTEND_Lx	Indicates that the PCR may be extended from locality x (x=4,3,2,1,0) by using the TPM2_PCR_Extend or TPM2_PCR_Event commands.
TPM_PT_PCR_RESET_Lx	Indicates that the PCR may be reset by TPM2_PCR_Reset from locality x (x=4,3,2,1,0).
TPM_PT_PCR_NO_INCREMENT	Indicates that modifications to this PCR (reset or extend) will not increment the *pcrUpdateCounter*.
TPM_PT_PCR_DRTM_RESET	Indicates that the PCR is reset by a DRTM event. These PCR are reset to all-ones on TPM2_Startup and reset to all-zeros on a _TPM_Hash_End event following a _TPM_Hash_Start event.
TPM_PT_PCR_POLICY	Indicates that the access to the PCR is controlled by policy. This property is only present if the TPM supports policy control of a PCR.
TPM_PT_PCR_AUTH	Indicates that the PCR is controlled by an authorization value. This property is only present if the TPM supports authorization control of a PCR.

2.1.5.4 PCR Quotes

TPM PCR quotes are defined in the TPM 2.0 Specification – Part 2: Structures [5] as the data structure TPMS_ATTEST. The quote structure, i.e., the structure that is hashed and signed as a result of an attestation of the platform state, is composed of the fields described below. The values in parenthesis denote the data type.

- *magic* (TPM_GENERATED): The indication that this structure was created by a TPM. Prevents an attacker from signing arbitrary data with a restricted signing key and claiming later that it was a TPM quote.
- *type* (TPMI_ST_ATTEST): Type of attestation structure (a quote, in this case)
- *qualifiedSigner* (TPM2B_NAME): Qualified Name of the signing key. A key could appear strong but be protected by an ancestor with a weaker algorithm. The qualified name represents the entire ancestry of the key.
- *extraData* (TPM2B_DATA): External information supplied by the caller. This data is typically an anti-replay nonce.
- *clockInfo* (TPMS_CLOCK_INFO): The clock information is obfuscated when signing with a key outside the endorsement hierarchy.
- firmwareVersion (UINT64): TPM-vendor-specific value identifying the version number of the firmware. Included so that the verifier can decide if it thrusts a particular TPM code version.
- [type]attested (TPMU_ATTEST): The type-specific attestation information. In this case, a TPMS_QUOTE_INFO structure which contains:
    - pcrSelect (TPML_PCR_SELECTION): Information on the PCR selected for the quote.
    - pcrDigest (TPM2B_DIGEST): Digest of the selected PCR using the hash of the signing key.

2.2 Cryptography Subsystem, Keys and Key Operations

2.2.1 Cryptography Subsystem

The cryptography subsystem implements the TPM's cryptographic functions. They include hash functions, asymmetric encryption and decryption, asymmetric signing and signature verification, symmetric encryption and decryption, symmetric signing (HMAC), signature verification and key generation.

A TPM may only implement algorithms that have a TCG-assigned algorithm ID. Algorithm IDs are available in the TCG Algorithm Registry [45], and also in the TPM 2.0 Specification – Part 2: Structures [5].

The strength of at least one algorithm set supported by a TPM should be at least 112 bits. Other algorithms and algorithm sets may be supported in any combination.

TCG requires various platform-specific algorithms to be implemented in different platforms (including one hash algorithm, one symmetric encryption algorithm with approved parameters, and one asymmetric encryption/signing algorithm with approved parameters).

A TPM implementation may support additional (crypto) algorithms in addition to the platform-specific algorithms required by the TCG standard.

2.2.1.1 Hash Functions

Hash functions may be used directly by external software or as the side effect of many TPM operations. The TPM uses hashing to provide integrity checking and authentication as well as one-way functions, as needed, e.g., Key Derivation Functions (KDFs).

A TPM should implement an approved hash algorithm that has approximately the same security strength as its strongest asymmetric algorithm.

The hash functions are also used when validating certain types of authorizations or used in support of other operations in the TPM such as PCR operations.

2.2.1.2 HMAC Algorithm

The TPM implements the Hash-based Message Authentication Code (HMAC) algorithm described in ISO/IEC 9797-2.

An HMAC is a form of symmetric signature over some data. It provides assurance that protected data was not modified and that it came from an entity with access to a key value. To have usefulness in protecting data, the key value needs to be a secret or a shared secret.

A TPM module may use the HMAC function to validate an authorization. The HMAC function may be used by the Command Execution Engine in support of its operations.

2.2.1.3 Asymmetric Operation

A TPM uses asymmetric algorithms for attestation, identification, and secret sharing. An asymmetric algorithm identifier will indicate a family of algorithms and methods that are used with that algorithm.

The only supported asymmetric algorithms in TPM 2.0 are RSA and Elliptic-Curve Cryptography (ECC) using prime curves.

A TPM is required to implement at least one asymmetric algorithm.

2.2.1.4 Signature Operation

The TPM may sign using either an asymmetric or a symmetric algorithm. The method of signing depends on the type of the key. For an asymmetric algorithm, the methods of signing are dependent on the algorithm (RSA or ECC). For symmetric signatures, only the HMAC signing scheme is currently defined. If a key may be used for signing, then it will have an attribute to allow it for.

A key may be restricted to sign messages with specific contents. When a key has this restriction, the TPM will not use the key to sign message digests that the TPM did not compute.

Any attestation message produced by a TPM will have a header to identify the data as being produced within a TPM. If a restricted key is used to sign this data, then a relying party can have assurance that the message data came from a TPM.

To allow a restricted key to sign an externally generated message, the TPM can produce the message digest. When the TPM computes the digest, it will not produce a special certification (called ticket) that indicates that the digest was produced by the TPM and is safe to sign with a restricted key.

A signing scheme can be used when a key allows for it because not all schemes are valid for all keys. A TPM generates an error if the scheme is not allowed with the indicated key type.

A restricted signing key requires to have a signing scheme specified in the key definition and that is the only signing scheme that is allowed to be used with the key. Unrestricted keys may contain a signing scheme selection, or the signing scheme may be determined when the key is used.

2.2.1.5 Symmetric Encryption

The TPM uses symmetric encryption to encrypt some command parameters (e.g.: authentication information) and to encrypt objects stored outside it. Cipher Feedback mode (CFB) is the only block cipher mode required by TPM 2.0 Specification to encrypt command parameters (as well as sessions and sensitive area of a key object).

Any symmetric block cipher supported by a TPM may be used for parameter encryption. Weak keys are not permitted to be used (some algorithms have known weak keys, if such a key is generated, it must be discarded, and a new key generated by starting over with another iteration).

A TPM should support Exclusive OR (XOR) obfuscation, which is a hash-based stream cipher. XOR obfuscation may be used only for confidential parameter passing.

When paired with an asymmetric key, a symmetric key is required to have as many bits of security strength as the asymmetric key with which it is paired.

When a symmetric key is used for data encryption, the encrypted data has an HMAC. This HMAC is checked before the data is decrypted.

2.2.1.6 Random Number Generator (RNG) Module

The RNG module is the source of randomness in the TPM. The TPM uses random values for nonce, in key generation, and for randomness in signatures.

It nominally consists of an entropy source and collector, a state register, and a mixing function (typically, an approved hash function). The entropy collector collects entropy from entropy sources (e.g. noise, clock variations, air movement, event timestamps, or jitter) and removes bias. A TPM should have at least one internal source of entropy, and possibly more. It is also possible to add externally generated entropy through the TPM2_StirRandom command. The collected entropy is then used to update the state register that will provide input to the mixing function to produce the random numbers.

The mixing function may be implemented with a pseudo-random number generator (a PRNG). A PRNG may produce numbers that are apparently random from a non-random input such as a counter. An approved PRNG that combines an input with much more entropy than a counter will yields a RNG with properties no worse than the underlying PRNG and possible better.

Each RNG access produces a new value regardless of the data usage. There is no distinction between accesses for internal versus external purposes.

It is worth noticing that the entropy sources, as defined in the TPM 2.0 Specification [3], largely depend on the manufacturer of the hardware chip and the application domain it is envisaged for usage (intended market). Whereas the vendor implementations of the entropy collection process are usually non-disclosed information, it is possible to find public evaluation reports that demonstrate the quality of the RNG. See for example report [46] for Infineon SLB 9670 TPM 2.0. Also, more information can be found on datasheets provided by manufacturers, such as Atmel (Microchip) AT97SC3205T TPM 1.2, which claims to contain a hardware RNG, including a FIPS-140-2 [47] certified PRNG [48].

2.2.2 Keys and Key Operations

The correct and secure handling of keys in a cryptographic system is essential for its operation. After the description of a key internal structure (public and private areas), this section will go through the lifeline of a key from its creation (generation or derivation) to its destruction.

These operations are referred as key management operations and they include mechanisms to store a key outside the TPM. To complete the section, the authorization concept that mandate the usage of a key is introduced (alongside the session that carries it) and further discussed in Section 2.3.

2.2.2.1 Key Structure

A Key Object is composed of two areas: a *public* and a *sensitive* area. Values within parenthesis denote the data type, as defined in [5].

The *public* area contains the attributes of the key and a public identity, including:

- *type* (TPMI_ALG_PUBLIC): algorithm ID used to create the key.
- *nameAlg* (TPMI_ALG_HASH): algorithm ID used as hash algorithm to compute the name of the object, it may be TPM_ALG_NULL
- *objectAttributes* (TPMA_OBJECT): usage, authorization, duplication, creation, persistence
- *authPolity* (TPM2B_DIGEST): authorization policy
- [type]*parameters* (TPMU_PUBLIC_PARMS): parameters for the algorithm specified as *type* (e.g.: key size)
- [type]*unique* (TPMU_PUBLIC_ID): for asymmetric key it will be the public key, for symmetric it will be a value hashed of information in the *sensitive* area

The sensitive area contains data that are required to be encrypted, including:

- *sensitiveType* (TPMI_ALG_PUBLIC): type of object in the sensitive area, it must be equal to the *type* parameter in the *public* area
- *authValue* (TPM2B_AUTH): authorization value for the object, it's a bite array with length equal to the length of the digest produced by *nameAlg*.
- *seedValue* (TPM2B_DIGEST): it may represents the optional protection seed (for a parent key) or an obfuscation value
- [sentitiveType]*sensitive* (TPMU_SENSITIVE_COMPOSITE): parameters dependent on the *sensitiveType* (e.g.: private key for asymmetric key)

2.2.2.2 Key Generation

Keys can be generated in two different ways. The first way is to produce a key starting from a random number generator (see RNG, Section 2.2.1.6). The second way is to produce a key starting from a Primary Seed following a KDF.

2.2.2.3 Key Derivation Functions (KDFs)

A key can be generated by deriving it from another secret value. The TPM has 3 primary seeds, which are large random numbers stored persistently in the different TPM hierarchies. Generating a key using one of these seeds creates a hierarchy of keys. See Section 2.1.4 for further details.

The TPM uses two different KDF schemes: one scheme for ECDH (Elliptic curve Diffie-Hellman) and one for all other crypto operations. These schemes are based on hash-functions (Section 2.2.1.1). For ECDH the KDF is SP800-56A, for all the others it is SP800-108.

2.2.2.4 Key Import

Before using a key, it must be loaded in the TPM. Loading a key may require authorization. It is possible to load the *public* portion or the *public* and *sensitive* portion of a key object.

Multiple consistency checks are performed to assure that the *sensitive* area was not modified, that the *sensitive* area is bonded to the correct *public* area, and that the attributes are consistent.

There are cases when only the public portion of the key is loaded (such as duplication or signature verification). In these cases, it is required to associate the key to a hierarchy to determine which proof value need to be used. If the hierarchy is disabled, the key will not be loaded in the TPM.

External objects might be loaded but a TPM will not create or load an object that uses an algorithm that is not supported by the TPM.

2.2.2.4.1 Context Management and Loading

A key can be loaded into the TPM in a wrapped form (i.e., encrypted) with a specified parent. The TPM will unwrap it and check along the chain from the parent for authorization (this may require inconvenient password prompts or certain PCR states that might be passed).

A TPM can also context-save and context-load a key outside/inside it. The key is wrapped with a key derived from a hierarchy therefore it is attached to a hierarchy but not connected to any parent.

In the TPM 2.0 Specification, both context management and loading use symmetric encryption to export a key outside the TPM.

2.2.2.5 Key Destruction

When a key needs to be destroyed, its parent needs to be destroyed such that when the key is imported it can’t be verified. If the key to be destroyed is a *primary key* then its primary seed needs to be destroyed.

2.3 Authorisations and Sessions

Authorisations relate to mechanisms granting someone access to a TPM entity [3]. The properties of that entity, often defined at creation time, determine the kind of authorisation that is required by each role. The TPM 2.0 Specification considers 3 roles: the USER role is used for the normal uses of a key (e.g., signing with a signing key, or loading the child of a storage key); the ADMIN role controls the certification and the changing of the authorisation value of an object; and the DUP role is only used for the duplication of keys.

Authorisation may be granted by two means. The first corresponds to a proof of knowledge of an authorisation value, also known as a password. This can be achieved by sending the password in the command authorisation area, or via an HMAC whose behaviour is determined by the password. The second means is through a policy digest, which requires that specific tests or actions are performed before an action is completed.

A session is defined to be a collection of TPM state that changes after each use [3]. They provide means to communicate authorisation data, audit a sequence of commands, build a policy digest, and encrypt command parameters.

In the TPM, there is a single, always-available password session that is used to authorise a single TPM command. Because of this, a client never needs to start a session to be granted authorisation with a password. It suffices that he passes the password in clear text format to the TPM as part of a command. This type of authorisation is of limited flexibility and presents security issues when a TPM is accessed remotely.

Sessions can be created through the TPM2_StartAuthSession command. They have associated session and HMAC keys. The values of the keys are determined not only by the authorisation value of the entity that is being accessed but can also depend on salts and on the authorisation value of another entity. When a session is started, the caller might indicate a size of nonces and an initial nonce. After initialising the session, the TPM returns a nonce generated by it. Each time the session is used for authorisation, nonces are updated, and the session and HMAC keys are updated accordingly. An entity handle might be sent with the TPM2_StartAuthSession command to indicate that that entity's authorisation value should also be included in the calculation of the session and HMAC keys of the session being initiated.

Sessions might be of 3 types: HMAC, policy or trial policy.

- When an **HMAC session** is in place, a client might compute an HMAC of the digest of the command parameters. Since the HMAC key depends on the entity's authorisation value, the correct computation of the HMAC proves knowledge of the authorisation value. If the entity's properties are compatible with this type of authorisation and authorisation role, the command will execute successfully. The command response parameters may also be HMACed, guaranteeing their integrity and authenticity.
- Access to entities might also be made dependent of a **policy session**, ensuring that a sequence of conditions have been satisfied before that entity can be accessed. A policy session is a form of enhanced authorization to allow for complex type of authorizations. It may include authorization based on TPM command sequences, TPM states or information coming from external devices (e.g., fingerprint and retina scanners, smart cards etc…). The policy is encapsulated in a value that is associated with the entity.
  The value representing a policy corresponds to a digest. After initiating a policy session, the TPM is given a sequence of policy-related commands that modify the digest in the policy session. Commands that affect the policy digest include assertions (for example, TPM2_PolicySigned is a policy assertion that an authorization was signed by a specific entity; and TPM2_PolicyPCR is an assertion that a selected set of PCRs have a specific value), ANDs that require two assertions or compounded assertions to be satisfied, and ORs that require that one of two assertions or compounded assertions are satisfied. After executing all the required commands, the policy session is used to access an entity. Certain policy-related commands do deferred assertions at this point. If the deferred assertions are satisfied, and the policy session digest matches that of the entity's policy value, access is granted.
- A **trial policy session** provides a means to compute a policy value that can be associated with an entity. Like in a normal policy session, after the session is created, a number of commands are issued that update the trial policy session digest. In contrast to a normal policy session, all the assertions are assumed to be true, and the trial policy digest is updated accordingly. After the computation of the trial policy trial digest has been finalised, the policy value can be read from the TPM. Then, when creating an entity, this value can be set as the policy value associated with that entity. Trial policy sessions cannot be used to be granted access to entities.

Per-command session modifiers are available. In the case of HMAC sessions, one may encrypt the first parameter of certain commands that are sent to a TPM; or ask for a response parameter to be encrypted; or ask for commands to be audited. Similar options are available for the policy sessions, apart from the auditing. Two modes of encryption are available: CFB and XOR. The former requires both access to a block cipher and a hash function, while for the latter access to a hash function suffices. The type of encryption to be used is established at session creation time. For the CFB mode, a KDF is used to produce both the key and the Initialisation Vector (IV) from the session key and the nonces. For the XOR mode, a one-time pad is produced with a KDF using the HMAC key and the nonces as input.

A host may maintain a record of the command and response parameters that are passed between it and a TPM. As these commands are issued, a host may furthermore request the TPM to extend the command and response parameters into an audit digest, as part of an HMAC session. An auditor can later request a signed copy of the audit digest to validate the integrity of the host's log. In addition, a host may have a single exclusive audit session, which may be used to prove to an auditor that no other commands were interleaved with the logged sequence.

2.4 Attestation Protocols

FutureTPM will include the provision of secure, robust, and efficient run-time behavioural attestation and verification methods to check the internal state of an untrusted cyber-physical system towards establishing its trustworthiness and privacy. This is considered as one of the main goals towards **"security and privacy by design"** solutions, including all methods, techniques, and tools that aim at enforcing security and privacy at software and system level from the conception and guaranteeing the validity of these properties. For privacy, FutureTPM will leverage advanced crypto primitives, namely **Direct Anonymous Attestation (DAA)** [49], whereas for security and operational assurance, it will enable the provision of **Control Flow Attestation**.

In general, remote attestation is a means of integrity verification of software running on a remote device. It is a mechanism, typically realised as a challenge-response protocol, which enables a trusted party (verifier) to obtain an authentic, accurate and timely report about the software state of a potentially untrusted remote device (prover). The verifier then checks whether the reported state is trustworthy, i.e., whether only benign software is loaded on the prover.

On the privacy side, **DAA** is a platform authentication mechanism that enables the provision of privacy-preserving and accountable authentication services. DAA is based on group signatures, which give strong anonymity guarantees [50]. The key security and privacy properties documented in [51], [52], [53] are:

- **User-controlled Anonymity:** Identity of user cannot be revealed from the signature.
- **User-controlled Linkability:** User controls whether signatures can be linked.
- **Non-frameability:** Adversaries cannot produce signatures originating from a valid TPM.
- **Correctness:** Valid signatures are verifiable, and linkable, where needed.

A DAA scheme considers a set of issuers, hosts, Trusted Components (TCs), and verifiers. A host and its TCs together form a Trusted Platform. An issuer is a trusted third-party responsible for attesting and authorizing platforms to join a network. A verifier is any other system entity or trusted third-party that can verify a platforms' credentials in a privacy-preserving manner using DAA algorithms, i.e., without the need of knowing the platform's identity. The TCG has standardized the ECC-based DAA scheme in the TPM 2.0 Specification [3]. This specification has also been published as the international standard ISO/IEC 11889:2015 [54] and comprises five algorithms: SETUP, JOIN, SIGN, VERIFY and LINK.

In a nutshell, DAA is essentially a two-step process where, firstly, the registration of a TC is executed once, and during this phase the TC chooses a secret key (SETUP). This secret key is stored in secure storage so that the host cannot have access to it; see Section 2.1.4.1. Next, the TC talks to the issuer so that it can provide the necessary guarantees for its validity (JOIN). The issuer then places a signature on the public key, producing an AIK <*cre*>. The second step is to use this <*cre*> for anonymous attestations on the platform (SIGN), using Zero-Knowledge Proofs [55]. These proofs convince a verifier that a message is signed by some key that was certified by the issuer, without knowledge of the TC's DAA key or AIK <*cre*> (VERIFY). Of course, the verifier has to trust that the issuer only issues <*cres*> to valid TCs. More details on the underpinnings of each one of these phases and various proposed DAA schemes can be found in Section 7.4 of D2.1 [8].

Based on the security and privacy requirements that have been specified for the three envisioned Reference Scenarios [1], the *anonymity*, *pseudonymity* and *unobservability* properties are built into DAA's algorithms, JOIN and SIGN / VERIFY by using anonymous digital signatures. Therefore, third-parties cannot identify and link subsequent service requests originating from the same user/system. This is also true in the presence of colluding third-parties. The JOIN protocol is intentionally not privacy-preserving as the Issuer needs to be aware of the user/system to be authenticated. However, successful completion of the protocol results in the user/system solely owning a DAA credential.

Unlinkability (and/or different levels of *user linkability*) is controlled by the user through the DAA SIGN / VERIFY phases. A user/system has control over its credential, and can decide whether or not to "blind" it, thus, producing pseudonyms (and revocation) that are linkable. In addition, DAA also provides non-frameability and correctness properties which are security attributes that are vital to the envisioned scenarios and especially in the context of Reference Scenario 2 (Personal Activity Tracking). DAA ensures that only valid and trustworthy TCs are able to join a network by ensuring that the endorsed TC keys have not been previously compromised. This ensures that TCs only produce valid signatures and can only be linked when specified by a particular authorized service.

On the security side, there exist different kinds of attestation, particularly **static attestation** and **dynamic attestation** [56]. Static attestation allows the attestation of static properties and configuration of a remote platform. The most prominent example is the attestation of the integrity of binaries [57]. As the name implies, dynamic attestation deals with dynamic properties of the runtime. For instance, it is concerned about the execution and data flow of programs, and not the static integrity of binaries. Naturally, attesting dynamic properties is significantly more challenging than attestation of static (already known) properties. Hence, the majority of research has focused on static attestation including industry effort in the Trusted Computing Base introducing secure and authenticated boot loading mechanisms of operating systems. However, given the continuous attacks on dynamic properties such as zero-day exploits which corrupt program's control flows, static attestation alone cannot be considered a viable security solution in the long-run, and needs to enhanced with advanced dynamic attestation mechanisms.

There does not yet exist a comprehensive design nor an effective as well as efficient implementation to enabling dynamic attestation. The most prominent approach in this context is **Control Flow Attestation** [58]. Control Flow Attestation is one of the most important dynamic properties at the software layer since it captures diverse instantiations of software exploits that hijack a program's control flow. In FutureTPM, we will develop automated and scalable behavioural-based attestation techniques focusing on the attestation of properties of software and hardware for cyber-physical systems. For this, we plan to adopt and extend static and dynamic attestation techniques so that both static and run-time properties of a remote platform can be attested. See Section 4.5.

2.5 Risk and Vulnerability Assessment

Risk Management is a key aspect for the secure and efficient operation of (deployed) cyber-physical systems. Risk analysis methods are used to evaluate the effectiveness of mitigation actions that are associated with a given risk/incident. The FutureTPM project will provide a framework that transparently augments the security of the QR TPM developed tailored to both the security requirements of the whole QR TPM-based system (i.e., the QR TPM and the host device) but also of the targeted ICT deployments through the three envisioned use cases, providing a risk quantification methodology which is model-driven. The goal of Risk Management is to minimize the risk to corporate and personal assets due to malicious and accidental loss or exposure. Risk Management processes help assess and mitigate risk. An element of Risk Management is risk and vulnerability assessment. Asset owners seek to understand techniques employed to protect their assets and identify vulnerabilities associated with the protection mechanisms.

2.5.1 Risk Assessment for TPM and TSS

This subsection provides an overview of the literature regarding the most known Risk Assessment methods and tools used a high level reference point. For more information, Deliverable D3.1 and WP4 will provide more details in the topic. The Risk Assessment phase is completed during design time. Various techniques have been proposed in the literature for performing threat and vulnerability modelling and risk assessment. Techniques performing threat modelling and risk assessment include, for example, STRIDE, Attack Tree, Attack Libraries, COBRA, and Mehari. A more exhaustive list of methods and tools can be found in [59].

One work that focuses on threat modelling and TPM 2.0 is [60]. This work presents a threat model for TPM 2.0 constructed using Microsoft Security Development Lifecycle Threat Modelling Tool and STRIDE model. The work, even though is based on simple scenarios, highlights some potential pitfalls that should be considered when conducting further research into the applications of TPM. Until now there aren't many research works related to TPM risk assessment. For instance, risk assessment focused on specific threats in TPM and TPM commands is still missing from the literature.

Regarding the risk assessment, tracing techniques can be used. Tracing has proved itself to be a robust and efficient approach to debugging and reverse engineering complex systems. Tracing is common in user applications but is also widely used in the Linux kernel, which provides multiple tracing infrastructures [61]. A work that leverages eBPF tracing is [62], where a side-channel attack on Java heap management is presented. A basic component of risk assessment is vulnerability analysis, which is the study of identified vulnerabilities and to what extent they affect the object of the assessment. There is a wide range of generic and TPM specific vulnerabilities in the literature, more specifically, an attack called “Bad Dream” [63] uses a power management flaw in order to forge custom PCR values that are used by the TPM to testify to its integrity. This attack is part of a general category that targets the hardware of the implementation, these hardware attacks include side channel attacks, clock glitching and fault injections, with all of these affecting any kind of hardware depending on how it is implemented and what primitives are used. On the other hand, software based attacks, target the firmware and the applications that are running on the platform. In CVE-2017-15361 [64], there was identified a flaw in the RSA key generation software used by the TPM from a specific manufacturer, this led to the predictability of private keys given the corresponding public keys. As it is obvious, software attacks are highly dependent on the quality of the software implementation. Software attacks include fuzzing, denial of service, buffer overflows and spoofing.

2.5.2 Security Policy Enforcement Mechanism

Security policy enforcement is executed both during the design phase and the run-time phase of product development. The design phase enforcement will enforce policies that are expected to reach a specific security level. If some criteria fail and the desired level is not met, the run-time enforcement sets new policies that are calculated with a security level goal system, in order to improve the system’s security posture against newly identified attacks. The run-time phase can be run iteratively until all the criteria for the required security level are met. There is a large body of related work on information flow security enforcement mechanisms. There are two major approaches to information flow security enforcement: static techniques and dynamic techniques [65]. In particular, during the design phase of the policy enforcement, the main approaches to enforce a particular information flow security policy, is called noninterference. Noninterference for programs means that a variation of confidential (high) input does not cause a variation of public (low) outputs. Static analysis techniques have one major drawback: they accept the program only if all its executions ensure non-interference. A common mechanism for ensuring that software, in our case the risk assessment, behaves securely is to monitor programs at run-time and check that they dynamically adhere to constraints specified by a security policy. Policies enforced by the run-time composition, configuration, and regulation of security services. The principle of separating security policy and dynamically enforcing security on applications is not new. Several authors have proposed security policy enforcement mechanisms using code modification as a technique for enforcing security policies such as resource limits, access controls, and network information flows. However, these approaches are typically ad hoc and are implemented without a high level abstract framework for code modification [66]. Another approach is by using reflection as a mechanism for implementing code modifications within an abstract framework based on the semantics of the underlying programming language. A recent survey [67], presents novel methods that employ machine learning and artificial intelligence in the pursuit for security vulnerability mitigation.

2.5.3 Risk Assessment Components and Interfaces

According to the supervisory guidance, a model could constitute any quantitative method, methodology or rule-set. These approaches apply statistical, economic, financial, or mathematical theories, techniques or assumptions that function to transform input data into quantitative estimates. This construct has three components according to [68]:

- Inputs: These constitute either data (which could be ‘hard’ or an expert opinion based), hypotheses, assumptions or other model output.
- Processing apparatus: This is a method, technique, system or algorithm for transforming model inputs into model outputs. This may be a statistical, mathematical or judgemental.

- Reporting component: This is the system for converting model outputs into a form that is useful for making business decisions.

From the standpoint of prudent Risk Management, the main quantitative **Risk Assessment (RA)** life cycle components can be described in three phases below [68]:

- The identification of model risk sources and classification.
- The quantification of the model risk inherent to each source.
- The mitigation of model risk identified and quantified by applying the appropriate measures, which will depend on the nature of the source.

Chapter 3 Analysis of Reference Scenarios' Requirements wrt. QR TPM Interfaces

3.1 FutureTPM Functional Requirements

In this subsection, we provide the analysis of the FutureTPM functional requirements of the Reference Scenarios. An initial mapping between the three scenarios and the technical requirements is listed in Deliverable D1.1 [1], which will be used as a reference point and input to further analyse the functional requirements per use case. For this further analysis we will include only the MVP functional requirements that covered from the scenario requirements. More specifically, only the basic blocks category of the mandatory requirements are the functional ones. The rest of the categories, such as performance and cost-effectiveness, implementation and deployment, are non-functional requirements and we will not be addressed in this subsection. We will further analyse non-functional requirements in the context of WP6. Table 9 below summarises the number of functional requirements that will be demonstrated per scenario.

Table 9: Number of functional requirements per use case.

No. Functional Requirements	**Reference Scenario 1** Secure Mobile Wallet and Payments (INDEV)	**Reference Scenario 2** Personal Activity and Health Kit Tracking (S5)	**Reference Scenario 3** Device Management (HWDU)
Mandatory	2	2	3
Desirable	0	0	0
Total	**2**	**2**	**3**

In Table 10, we give a more detailed breakdown of the functional requirements per Reference Scenario. The Future QR TPM functional requirements are depicted in black colour, whereas the way each requirement is envisioned to be leveraged in the scenarios are depicted in blue colour. The functional requirement IDs can be found in Deliverable D1.1 [1], Section 5.1.

Table 10: Breakdown of QR TPM Functional Requirements per Reference Scenario.

Functional Requirement ID	**Functional Requirement description / Intended usage in the Reference Scenario**
Reference Scenario 1: Secure Mobile Wallet and Payments (INDEV)	
TR.1.1.1	It should provide non-volatile random-access memory (NVRAM) storage NVRAM will be used for OAuth Bearer & FreePOS authentication token storage
TR.1.1.4	It should support enhanced authorization (EA) EA will be used for both user and server authorization
Reference Scenario 2: Personal Activity and Health Kit Tracking (S5)	

Functional Requirement ID	Functional Requirement description / Intended usage in the Reference Scenario
TR.1.1.2	It should provide a small set of platform configuration registers (PCR) PCR can be used to reassure that S5Tracker Analytics Engine is trusted
TR.1.1.4	It should support enhanced authorization (EA) EA will be used for both Data Analyst and User authorization
Reference Scenario 3: Device Management (HWDU)	
TR.1.1.1	It should provide non-volatile random-access memory (NVRAM) storage NVRAM will be used for key storage to establish a secure channel
TR.1.1.2	It should provide a small set of platform configuration registers (PCR) PCRs will be used to accumulate measurements and determine if the software on controlled devices are enforcing the management commands
TR.1.1.4	It should support enhanced authorization (EA) EA will be used for network devices

3.2 FutureTPM Security Requirements

In this subsection, we provide the analysis of the FutureTPM security requirements of the Reference Scenarios. An initial mapping between the three scenarios and the security requirements is also listed in Deliverable D1.1 [1], which will be used as a reference and input to further analyse the security requirements per scenario. For this further analysis we will include only the MVP functional requirements that covered from the use case requirements (both mandatory and desirable). Table 11 below summarises the number of security requirements that will demonstrated per use case.

Table 11**:** Number of security requirements per use case.

No. Security Requirements	**Reference Scenario 1** Secure Mobile Wallet and Payments (INDEV)	**Reference Scenario 2** Personal Activity and Health Kit Tracking (S5)	**Reference Scenario 3** Device Management (HWDU)
Mandatory	7	12	13
Desirable	2	2	4
Total	**9**	**14**	**17**

In Table 12 below, we are giving a more detailed breakdown of the security requirements per Reference Scenario. Similarly, as in the previous section, the QR TPM security requirements are depicted in black colour, whereas the way it will be utilized in the scenarios are depicted in blue colour. The security requirement IDs can be found in Deliverable D1.1 [1], Section 5.2.

Table 12: Breakdown of QR TPM Security Requirements per Reference Scenario.

Security Requirement ID	Functional Requirement description / Intended usage in the Reference Scenario
Reference Scenario 1: Secure Mobile Wallet and Payments (INDEV)	
SR.1.1.2	Key generation and storage functionalities RNG will be used to generate encryption key and NVRAM will be used for token storage
SR.1.1.3	Hash functions HMAC will be used for database integrity
SR.1.1.4	MAC HMAC will be used for database integrity
SR.1.1.5	Symmetric encryption SE will be used for database confidentiality
SR.1.1.6	Digital signatures DS will be used for database integrity
SR.1.2.1	Support for possible QR-crypto candidates for each category (symmetric, asymmetric and DAA)
SR.1.2.2	QR Support for signing, key exchange, attestation
SR.2.2.1	Support for a broader range of access policies EA will be used to support such policies
SR.2.2.6	Secure key backup and recovery on TPM damage
Reference Scenario 2: Personal Activity and Health Kit Tracking (S5)	
SR.1.1.1	Pseudorandom number generator RNG will be used to generate encryption key
SR.1.1.2	Key generation and storage functionalities RNG will be used to generate encryption key
SR.1.1.3	Hash functions HMAC will be used for database integrity
SR.1.1.4	MAC HMAC will be used for database integrity
SR.1.1.5	Symmetric encryption Analytics Engine and users raw data are encrypted in the S5Tracker Analytics Engine

Security Requirement ID	Functional Requirement description / Intended usage in the Reference Scenario
SR.1.1.6	Digital signatures DS will be used for database integrity
SR.1.1.7	Public key encryption and key exchange Public key cryptography will be used for key exchange
SR.1.1.8	Direct Anonymous Attestation (DAA) (for SW QR TPM) DAA will be used between S5PersonalTracker and the S5Tracker Analytics Engine
SR.1.2.1	Support for possible QR-crypto candidates for each category (symmetric, asymmetric and DAA)
SR.1.2.2	QR Support for signing, key exchange, attestation
SR.1.3.1	Support software measurement (PCR extend) and measurement reporting (Quote), using QR algorithms
SR.1.3.2	Support remote attestation functionalities DAA will be used between S5PersonalTracker and the S5Tracker Analytics Engine
SR.2.2.3	Use of TPM to support cryptographic operations in blockchains and other services such as verifiable data access
SR.2.2.5	Secure logging of access to cryptographic operations in a blockchain (ability to provide accountable decryption and similar constructs)
Reference Scenario 3: Device Management (HWDU)	
SR.1.1.1	Pseudorandom number generator RNG will be used to generate encryption key
SR.1.1.2	Key generation and storage functionalities RNG will be used to generate encryption key
SR.1.1.3	Hash functions HMAC will be used for database integrity
SR.1.1.4	MAC HMAC will be used for database integrity
SR.1.1.5	Symmetric encryption SE will be used to establish trusted channels
SR.1.1.6	Digital signatures Network devices will be checked against TPM signature validity

Security Requirement ID	Functional Requirement description / Intended usage in the Reference Scenario
SR.1.1.7	Public key encryption and key exchange Public key cryptography will be used for key exchange
SR.1.2.1	Support for possible QR-crypto candidates for each category (symmetric, asymmetric and DAA)
SR.1.2.2	QR Support for signing, key exchange, attestation
SR.1.2.4	Provide a crypto library with TPM backed keys implementing TLS with QR algorithms
SR.1.3.1	Support software measurement (PCR extend) and measurement reporting (Quote), using QR algorithms
SR.1.3.2	Support remote attestation functionalities Remote attestation will support between NMS and a network device
SR.1.3.3	Support sealing and binding operations Sealing and binding will be used on routing table
SR.2.1.1	The future TPM implements a PQ-DAA schema
SR.2.1.5	Add support for QR algorithms and for TPM as key storage back-end to an IPSEC (IKE) implementation
SR.2.2.1	Support for a broader range of access policies Policies will allow the key usage depending on the integrity of the software
SR.2.2.6	Secure key backup and recovery on TPM damage

3.3 QR TPM Interfaces

Within the FutureTPM project, an important effort will be directed towards making the API for the QR TPM as similar and compatible as possible to the TPM 2.0 API, thus, achieving backwards compatibility. Therefore, the FutureTPM QR TPM software stack will be largely based on the present-day TSS introduced in Section 2.1.1.1.

As already pointed out in Deliverable D1.1 [1], with the primary goal of adding quantum resistance to existing TPM functionalities, ideally, the QR TPM would be as a drop-in replacement for present-day TPMs. In some cases, this ‘plug-and-play’ approach will be achievable, although there will inevitably be many cases where this is not possible. Interfaces between the components and external interfaces will be identified and specified. The architecture will include the essential components (various cryptographic engines, shielded memory, counters, etc.). Extensions will be needed to support the additional requirements of QR crypto; for example, hash-based signatures require machinery to ensure that signing keys are used only on one (perhaps composite) message, and machinery to manage key renewal. To support later activities, we will also outline how the reference architecture will be adapted to the three use cases.

The inclusion and support of the new QR-related features and commands can potentially trigger changes in the FAPI, ESAPI, SAPI and TCTI layers of the TSS. This will follow a bottom-up approach. As commented in Section 2.1.1.1, in order to satisfy the requirements identified in

Deliverable D1.1 [1], it is expected that most of the updates will be instantiated and triggered at the SAPI layer, which will also include the TCTI. These changes will be taken into consideration to be propagated and reflected to the upper layers (reaching the FAPI layer, which offers a more user-friendly version of the SAPI commands), but only where it is strictly necessary to do so. There are two main reasons to justify this approach:

- First and foremost, it is within the objectives of FutureTPM to achieve a drop-in functionality with the newly developed QR TPM. That is, it should be possible a seamless replacement of a TPM 2.0 device by a FutureTPM QR TPM device, so that any present-day system (featuring a TPM 2.0) continues to work normally after that replacement. Consequently, this requires a minimum propagation of changes to the upper TSS layers, which, of course, will be investigated when they are required by the QR nature of the algorithms.
- The second reason is that it can be the case that not all Reference Scenarios will use the upper layers such as FAPI or ESAPI, and will satisfy their requirements by using the SAPI layer only.

This will be investigated and documented within WP5 and WP6, in order to ensure that all the new features are reflected and addressed appropriately throughout the different API layers, where necessary.

An initial study of the FutureTPM interfaces based on the functional and security requirements is provided below. The aforementioned functional requirements (Section 3.1 and Table 13) are the already existing in TPM 2.0, meaning the interfaces are the same. However, the aforementioned security requirements (Section 3.2 and Table 14) need some additions and adjustments. As we can see there is an overlapbetween some of the requirements. For instance, there is an overlap between SR.1.1.7 Public key encryption and key exchange and SR.1.2.1 Support for possible QR-crypto candidates for each category (symmetric, asymmetric and DAA).

We have to note here that the list of TPM commands put forth in Table 13 andTable **14** is the output of an initial investigation of commands that need to be taken into consideration to satisfy the scenarios' requirements. By no means is it an indicative list of the entire set of commands that is going to be implemented, which, again, will be detailed in the context of WP5 and WP6.

Table 13: Functional interfaces per use case requirements.

Reference Scenarios	Interface	API Commands
[TR.1.1.1] It should provide non-volatile random-access memory (NVRAM) storage		
Ref. Scen. 1 (INDEV) Ref. Scen. 3 (HWDU)	SAPI *(also reflected in FAPI)*	TPM2_NV_DefineSpace TPM2_NV_UndefineSpace TPM2_NV_UndefineSpaceSpecial TPM2_NV_ReadPublic TPM2_NV_Write TPM2_NV_Increment TPM2_NV_Extend TPM2_NV_SetBits TPM2_NV_WriteLock TPM2_NV_GlobalWriteLock TPM2_NV_ChangeAuth TPM2_NV_Certify
[TR.1.1.2] It should provide a small set of platform configuration registers (PCR)		
Ref. Scen. 2 (S5) Ref. Scen. 3 (HWDU)	SAPI	TPM2_PCR_Extend TPM2_PCR_Event

Reference Scenarios	Interface	API Commands
	(also reflected in FAPI)	TPM2_PCR_Read TPM2_PCR_Allocate TPM2_PCR_SetAuthPolicy TPM2_PCR_SetAuthValue TPM2_PCR_Reset TPM2_PCR_Start TPM2_PCR_Data TPM2_PCR_End
[TR.1.1.4] It should support enhanced authorization (EA)		
Ref. Scen. 1 (INDEV) Ref. Scen. 3 (HWDU)	SAPI *(also reflected in FAPI)*	TPM2_PolicySigned TPM2_PolicySecret TPM2_PolicyTicket TPM2_PolicyOR TPM2_PolicyPCR TPM2_PolicyLocality TPM2_PolicyNV TPM2_PolicyCounterTimer TPM2_PolicyCommandCode TPM2_PolicyPhysicalPresence TPM2_PolicyCpHash TPM2_PolicyDuplicationSelect TPM2_PolicyAuthorize TPM2_PolicyAuthValue TPM2_PolicyPassword TPM2_PolicyGetDigest TPM2_PolicyNvWritten TPM2_PolicyTemplate TPM2_PolicyAuthorizeNV

Table 14: Security interfaces per use case requirement.

Ref. Scenarios	Interface	API Commands	Possible Changes
[SR.1.1.1] Pseudorandom number generator			
Ref. Scen. 2 (S5) Ref. Scen. 3 (HWDU)	SAPI *(also reflected in FAPI)*	TPM2_GetRandom TPM2_StirRandom	
[SR.1.1.2] Key generation and storage functionalities			
Ref. Scen. 1 (INDEV) Ref. Scen. 2 (S5) Ref. Scen. 3 (HWDU)	SAPI *(also reflected in FAPI)*	TPM2_ECDH_KeyGen	ECDH BROKEN EC BROKEN
[SR.1.1.3] Hash functions			
Ref. Scen. 1 (INDEV) Ref. Scen. 2 (S5) Ref. Scen. 3 (HWDU)	ESAPI	TPM2_HashSequenceStart TPM2_SequenceUpdate TPM2_SequenceComplete TPM2_EventSequenceComplete	
[SR.1.1.4] MAC			

Ref. Scenarios	Interface	API Commands	Possible Changes
Ref. Scen. 1 (INDEV) Ref. Scen. 2 (S5) Ref. Scen. 3 (HWDU)	ESAPI	TPM2_HMAC_Start TPM2_HMAC	
[SR.1.1.5] Symmetric encryption			
Ref. Scen. 1 (INDEV) Ref. Scen. 2 (S5) Ref. Scen. 3 (HWDU)	SAPI *(also reflected in FAPI)*	TPM2_EncryptDecrypt TPM2_EncryptDecrypt2 TPM2_Hash TPM2_HMAC	
[SR.1.1.6] Digital signatures			
Ref. Scen. 1 (INDEV) Ref. Scen. 2 (S5) Ref. Scen. 3 (HWDU)	SAPI *(also reflected in FAPI)*	TPM2_VerifySignature TPM2_Sign TPM2_Commit	RSA signature BROKEN EC signature BROKEN
[SR.1.1.7] Public key encryption and key exchange			
Ref. Scen. 2 (S5) Ref. Scen. 3 (HWDU)	ESAPI	TPM2_RSA_Encrypt TPM2_RSA_Decrypt TPM2_ECDH_ZGen TPM2_ECC_Parameters TPM2_EC_Ephemeral TPM2_ZGen_2Phase	RSA BROKEN EC BROKEN ECDH BROKEN

Ref. Scenarios	Interface	API Commands	Possible Changes
[SR.1.1.8] Direct Anonymous Attestation (DAA) [for SW TPM]			
Ref. Scen. 2 (S5)	ESAPI	TPM2_Certify TPM2_CertifyCreation TPM2_Quote TPM2_GetSessionAuditDigest TPM2_GetCommandAuditDigest TPM2_GetTime	EC-DAA BROKEN
[SR.1.2.1] Support for possible QR-crypto candidates for each category (symmetric, asymmetric and DAA)			
Ref. Scen. 1 (INDEV) Ref. Scen. 2 (S5) Ref. Scen. 3 (HWDU)	NEW	Overlapping with existing commands that need modification	TO ADD based on algorithms selected from WP2
[SR.1.2.2] QR Support for signing, key exchange, attestation			
Ref. Scen. 1 (INDEV) Ref. Scen. 2 (S5) Ref. Scen. 3 (HWDU)	NEW	Overlapping with existing commands that need modification	TO ADD based on algorithms selected from WP2
[SR.1.2.4] Provide a crypto library with TPM backed keys implementing TLS with QR algorithms			
Ref. Scen. 3 (HWDU)	NEW	-	TO ADD based on algorithms selected from WP2
[SR.1.3.1] Support software measurement (PCR extend) and measurement reporting (Quote), using QR algorithms			
Ref. Scen. 2 (S5) Ref. Scen. 3 (HWDU)	NEW	Overlapping with existing commands that need modification	TO ADD based on algorithms selected from WP2
[SR.1.3.2] Support remote attestation functionalities			
Ref. Scen. 2 (S5) Ref. Scen. 3 (HWDU)	ESAPI	TPM2_Certify TPM2_CertifyCreation TPM2_Quote TPM2_GetSessionAuditDigest TPM2_GetCommandAuditDigest TPM2_GetTime	EC-DAA BROKEN

Ref. Scenarios	Interface	API Commands	Possible Changes
[SR.1.3.3] Support sealing and binding operations			
Ref. Scen. 3 (HWDU)	SAPI *(also reflected in FAPI)*	Overlapping with existing commands that need modification TPM2_PolicyPCR TPM2_GetRandom TPM2_Create TPM2_Load TPM2_PolicyPCR TPM2_Unseal	TO ADD based on algorithms selected from WP2
[SR.2.1.1] The future TPM implements a PQ-DAA scheme			
Ref. Scen. 3 (HWDU)	NEW	-	TO ADD based on algorithms selected from WP2
[SR.2.1.5] Add support for QR algorithms and for TPM as key storage back-end to an IPSEC (IKE) implementation			
Ref. Scen. 3 (HWDU)	NEW	-	TO ADD based on algorithms selected from WP2
[SR.2.2.1] Support for a broader range of access policies			
Ref. Scen. 1 (INDEV) Ref. Scen. 3 (HWDU)	SAPI	Overlapping with existing commands that need modification	-
[SR.2.2.3] Use of FutureTPM to support cryptographic operations in blockchains and other services such as verifiable data access			
Ref. Scen. 2 (S5)	NEW	-	TO ADD
[SR.2.2.5] Secure logging of access to cryptographic operations in a blockchain (ability to provide accountable decryption and similar constructs)			
Ref. Scen. 2 (S5)	NEW	-	TO ADD
[SR.2.2.6] Secure key backup and recovery on TPM damage			
Ref. Scen. 1 (INDEV) Ref. Scen. 3 (HWDU)	SAPI *(also reflected in FAPI)*	TPM2_Duplicate TPM2_Rewarp TPM2_Import TPM2_LoadExternal	-

3.4 QR TPM API Updates

The aim for a direct drop-in was outlined in Deliverable D1.1 [1]. In addition, Deliverable D1.1 summarised an initial study on the features that require adjustments. This study might be updated on WP4, WP5 and WP6. These include:

- Increased key sizes, which may exceed the memory buffer available to the hardware QR TPM depending on the algorithms selected in WP2;

- Key renewal and stateful processes, which may require new API calls for hash-based signatures in comparison with the lack of stateful signatures in TPM 2.0;
- Key management may also provide issues for a plug-and-play approach;
- PQ keys and classical keys might not be able to co-exist in the same hierarchy;
- The current firmware updates of TPMs may not be secure under quantum security models (for example, if signed with RSA), but making firmware updates QR may require changes to the implementation from present-day TPM. Deliverable D3.1 will provide more information on this topic.

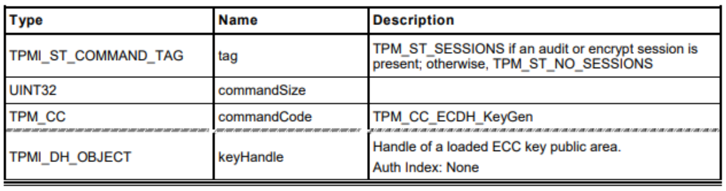
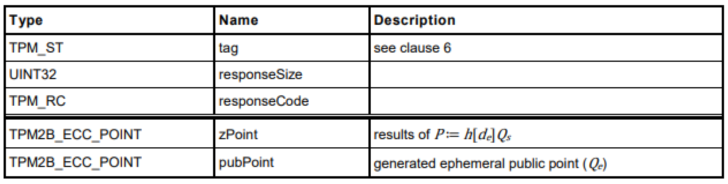
The envisioned “plug-and-play” replacement of the full list of existing TPM API calls is still a difficult task, assess at this stage. However, the API SHALL provide access to all features of all TPM 2.0 and FutureTPM QR TPM commands. In order to keep the TPM quantum resistant it is necessary to update the default sizes to the appropriate number of bits and at the same time that do not exceed hardware requirements. As already pointed out all public-key encryption and digital signature schemes as well as the DAA algorithm are not QR, since their security relies on the difficulty of factoring large composite integers and computing discrete logarithms and need replacement. Hash functions, RNGs and symmetric encryption algorithms seems to be fine but possible with an update on the default key sizes. Table 15 below summarizes the impact of quantum computing on common cryptographic algorithms [69]. In addition, for more information regarding the new FutureTPM QR algorithms we will focus on WP2.

Table 15: Impact of quantum computing on common cryptographic algorithms [69].

Cryptographic Algorithm	Type	Purpose	Impact from large-scale quantum computer
AES	Symmetric key	Encryption	Larger key sizes needed
SHA-2, SHA-3	-	Hash functions	Larger output needed
RSA	Public key	Signatures, key establishment	No longer secure
ECDSA, ECDH (Elliptic Curve Cryptography)	Public key	Signatures, key establishment	No longer secure
DSA (Finite Field Cryptography)	Public key	Signatures, key establishment	No longer secure

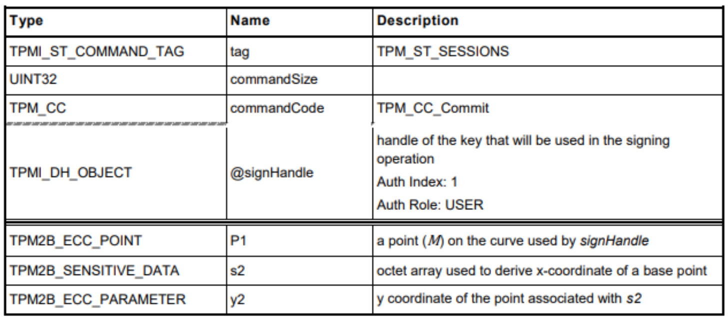
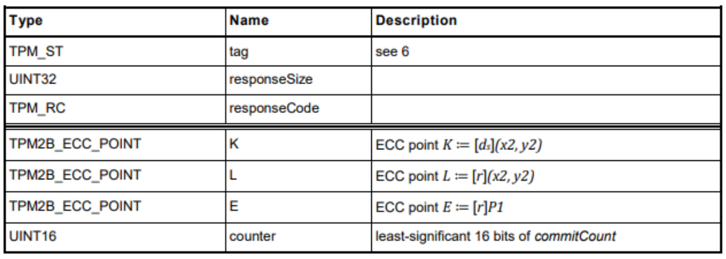
Table 16 below lists all necessary changes from the newly added and the modifications needed for the QR TPM based on the impact of quantum computing on common cryptographic algorithms. We will focus only on the **34** extracted commands related to the security use case requirements that possibly need changes/update. All the 34 commands can be found on Appendix A. While Table 16 below provide only the interfaces that might need update. For further information regarding the commands, the reader can refer to [6] and for replacement algorithms per crypto family to Deliverable D2.1 [8].

Table 16: QR TPM command updates.

Command | Description | Parameters | Response	Changes Needed
TPM2_ECDH_KeyGen	
This command uses the TPM to generate an ephemeral key pair.	
 	**YES** *ECDH is broken need to be updated with another algorithm*

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS if an audit or encrypt session is present; otherwise, TPM_ST_NO_SESSIONS
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_ECDH_KeyGen
TPMI_DH_OBJECT	keyHandle	Handle of a loaded ECC key public area. Auth Index: None

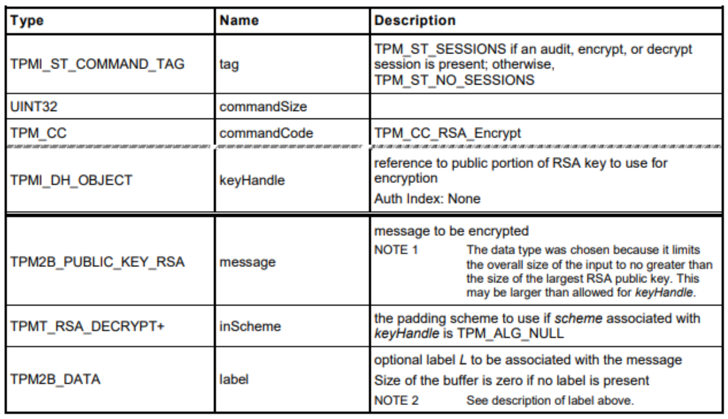
Type	Name	Description
TPM_ST	tag	see clause 6
UINT32	responseSize	
TPM_RC	responseCode	
TPM2B_ECC_POINT	zPoint	results of $P := h[d_e]Q_s$
TPM2B_ECC_POINT	pubPoint	generated ephemeral public point (Q_e)

Command | Description | Parameters | Response	Changes Needed
TPM2_Commit	
TPM2_Commit performs the first part of an ECC anonymous signing operation.	
 	**YES** *ECC is broken need to be updated with another algorithm*

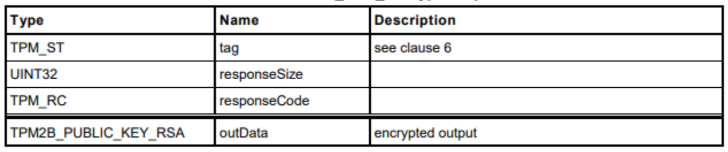
Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_Commit
TPMI_DH_OBJECT	@signHandle	handle of the key that will be used in the signing operation Auth Index: 1 Auth Role: USER
TPM2B_ECC_POINT	P1	a point (M) on the curve used by *signHandle*
TPM2B_SENSITIVE_DATA	s2	octet array used to derive x-coordinate of a base point
TPM2B_ECC_PARAMETER	y2	y coordinate of the point associated with *s2*

Type	Name	Description
TPM_ST	tag	see 6
UINT32	responseSize	
TPM_RC	responseCode	
TPM2B_ECC_POINT	K	ECC point $K := [d_s](x2, y2)$
TPM2B_ECC_POINT	L	ECC point $L := [r](x2, y2)$
TPM2B_ECC_POINT	E	ECC point $E := [r]P1$
UINT16	counter	least-significant 16 bits of *commitCount*

Command | Description | Parameters | Response	Changes Needed
TPM2_RSA_Encrypt	
This command performs RSA encryption using the indicated padding scheme according to IETF RFC 3447 (PKCS#1) [70].	
(see command and response tables below)	**YES** *RSA is broken need to be updated with another algorithm*

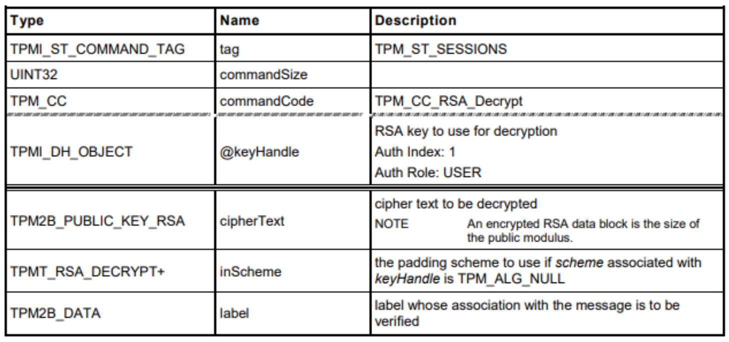


Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS if an audit, encrypt, or decrypt session is present; otherwise, TPM_ST_NO_SESSIONS
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_RSA_Encrypt
TPMI_DH_OBJECT	keyHandle	reference to public portion of RSA key to use for encryption Auth Index: None
TPM2B_PUBLIC_KEY_RSA	message	message to be encrypted NOTE 1 The data type was chosen because it limits the overall size of the input to no greater than the size of the largest RSA public key. This may be larger than allowed for *keyHandle*.
TPMT_RSA_DECRYPT+	inScheme	the padding scheme to use if *scheme* associated with *keyHandle* is TPM_ALG_NULL
TPM2B_DATA	label	optional label *L* to be associated with the message Size of the buffer is zero if no label is present NOTE 2 See description of label above.

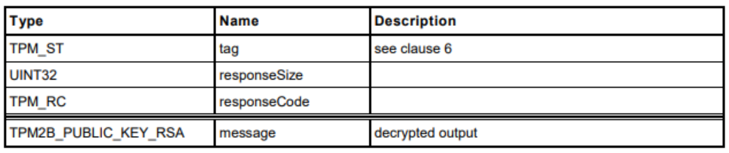


Type	Name	Description
TPM_ST	tag	see clause 6
UINT32	responseSize	
TPM_RC	responseCode	
TPM2B_PUBLIC_KEY_RSA	outData	encrypted output

Command | Description | Parameters | Response	Changes Needed
TPM2_RSA_Decrypt	
This command performs RSA decryption using the indicated padding scheme according to IETF RFC 3447 (PKCS#1) [70].	
(see command and response tables below)	**YES** *RSA is broken need to be updated with another algorithm*



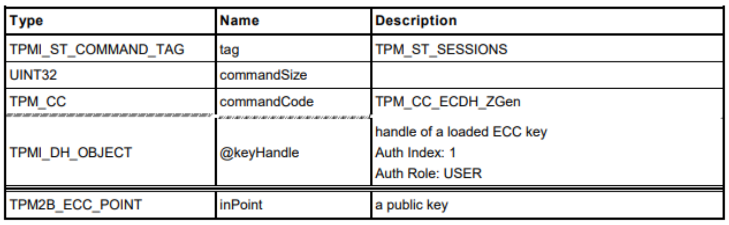
Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_RSA_Decrypt
TPMI_DH_OBJECT	@keyHandle	RSA key to use for decryption Auth Index: 1 Auth Role: USER
TPM2B_PUBLIC_KEY_RSA	cipherText	cipher text to be decrypted NOTE An encrypted RSA data block is the size of the public modulus.
TPMT_RSA_DECRYPT+	inScheme	the padding scheme to use if *scheme* associated with *keyHandle* is TPM_ALG_NULL
TPM2B_DATA	label	label whose association with the message is to be verified



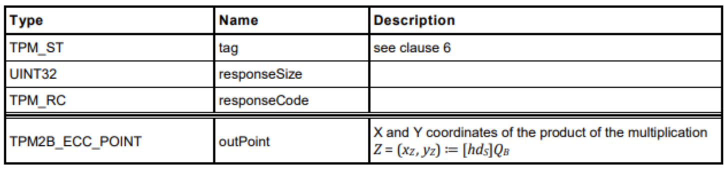
Type	Name	Description
TPM_ST	tag	see clause 6
UINT32	responseSize	
TPM_RC	responseCode	
TPM2B_PUBLIC_KEY_RSA	message	decrypted output

Command | Description | Parameters | Response	Changes Needed
TPM2_ECDH_ZGen	
This command uses the TPM to recover the Z value from a public point and a private key.	
See command and response tables below	**YES** *EDCH and ECC are broken need to be updated with other algorithms*
TPM2_ECC_Parameters	
This command returns the parameters of an ECC curve identified by its TCG-assigned curveID.	
See command and response tables below	**YES** *ECC is broken need to be updated with another algorithm*
TPM2_EC_Ephemeral	
TPM2_EC_Ephemeral creates an ephemeral key for use in a two-phase key exchange protocol.	
See command and response tables below	**YES** *ECDH is broken need to be updated with another algorithm*

TPM2_ECDH_ZGen

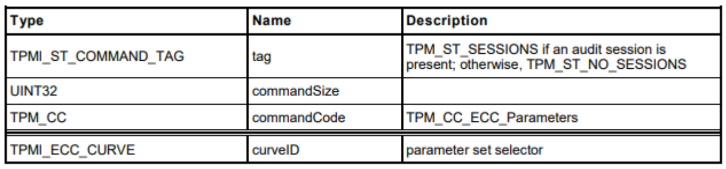


Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_ECDH_ZGen
TPMI_DH_OBJECT	@keyHandle	handle of a loaded ECC key Auth Index: 1 Auth Role: USER
TPM2B_ECC_POINT	inPoint	a public key

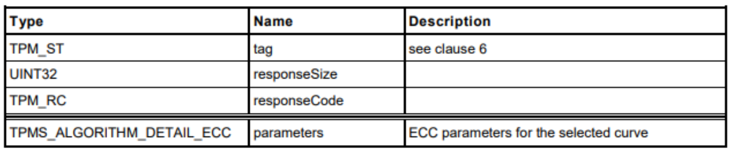


Type	Name	Description
TPM_ST	tag	see clause 6
UINT32	responseSize	
TPM_RC	responseCode	
TPM2B_ECC_POINT	outPoint	X and Y coordinates of the product of the multiplication $Z = (x_Z, y_Z) := [hd_S]Q_B$

TPM2_ECC_Parameters

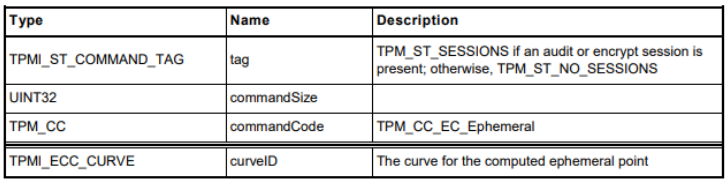


Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS if an audit session is present; otherwise, TPM_ST_NO_SESSIONS
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_ECC_Parameters
TPMI_ECC_CURVE	curveID	parameter set selector

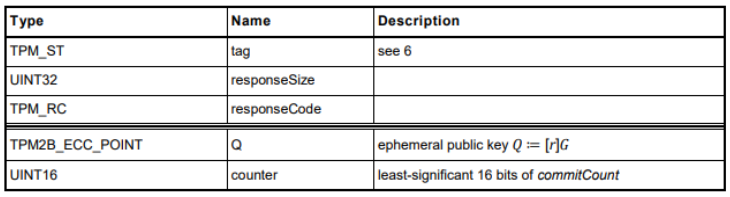


Type	Name	Description
TPM_ST	tag	see clause 6
UINT32	responseSize	
TPM_RC	responseCode	
TPMS_ALGORITHM_DETAIL_ECC	parameters	ECC parameters for the selected curve

TPM2_EC_Ephemeral



Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS if an audit or encrypt session is present; otherwise, TPM_ST_NO_SESSIONS
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_EC_Ephemeral
TPMI_ECC_CURVE	curveID	The curve for the computed ephemeral point

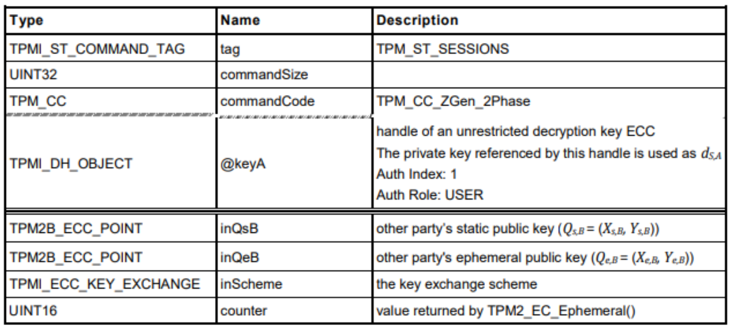


Type	Name	Description
TPM_ST	tag	see 6
UINT32	responseSize	
TPM_RC	responseCode	
TPM2B_ECC_POINT	Q	ephemeral public key $Q := [r]G$
UINT16	counter	least-significant 16 bits of *commitCount*

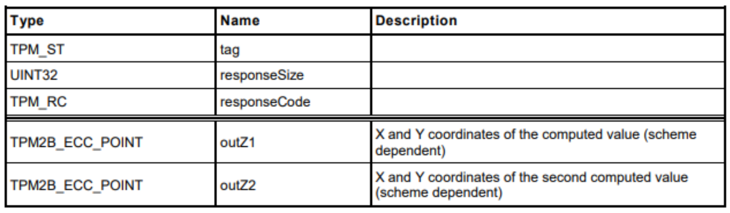
Command | Description | Parameters | Response	Changes Needed

TPM2_ZGen_2Phase

This command supports two-phase key exchange protocols. The command is used in combination with TPM2_EC_Ephemeral. TPM2_EC_Ephemeral generates an ephemeral key and returns the public point of that ephemeral key along with a numeric value that allows the TPM to regenerate the associated private key.



Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_ZGen_2Phase
TPMI_DH_OBJECT	@keyA	handle of an unrestricted decryption key ECC The private key referenced by this handle is used as $d_{S,A}$ Auth Index: 1 Auth Role: USER
TPM2B_ECC_POINT	inQsB	other party's static public key ($Q_{s,B} = (X_{s,B}, Y_{s,B})$)
TPM2B_ECC_POINT	inQeB	other party's ephemeral public key ($Q_{e,B} = (X_{e,B}, Y_{e,B})$)
TPMI_ECC_KEY_EXCHANGE	inScheme	the key exchange scheme
UINT16	counter	value returned by TPM2_EC_Ephemeral()



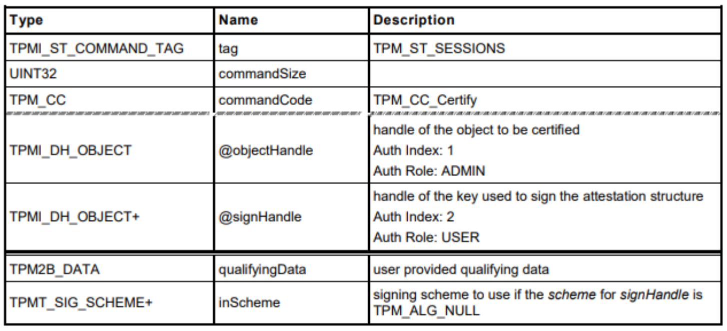
Type	Name	Description
TPM_ST	tag	
UINT32	responseSize	
TPM_RC	responseCode	
TPM2B_ECC_POINT	outZ1	X and Y coordinates of the computed value (scheme dependent)
TPM2B_ECC_POINT	outZ2	X and Y coordinates of the second computed value (scheme dependent)

Changes Needed: **YES**

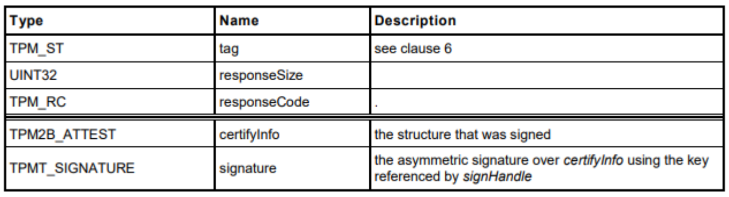
EDCH and ECC are broken need to be updated with other algorithms

TPM2_Certify

The purpose of this command is to prove that an object with a specific Name is loaded in the TPM.



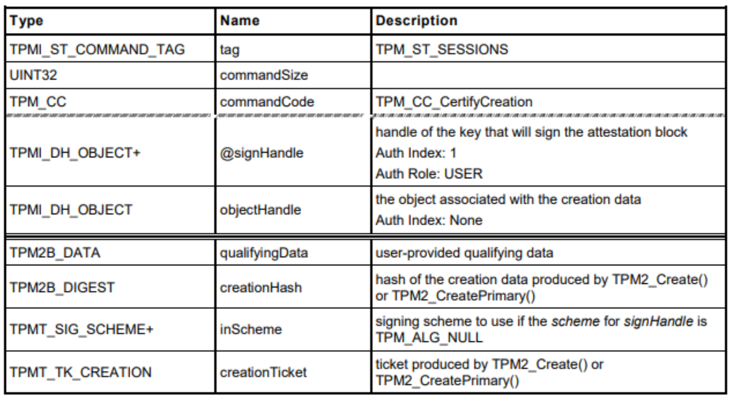
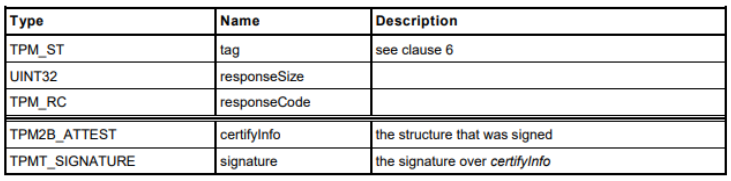
Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_Certify
TPMI_DH_OBJECT	@objectHandle	handle of the object to be certified Auth Index: 1 Auth Role: ADMIN
TPMI_DH_OBJECT+	@signHandle	handle of the key used to sign the attestation structure Auth Index: 2 Auth Role: USER
TPM2B_DATA	qualifyingData	user provided qualifying data
TPMT_SIG_SCHEME+	inScheme	signing scheme to use if the *scheme* for *signHandle* is TPM_ALG_NULL



Type	Name	Description
TPM_ST	tag	see clause 6
UINT32	responseSize	
TPM_RC	responseCode	.
TPM2B_ATTEST	certifyInfo	the structure that was signed
TPMT_SIGNATURE	signature	the asymmetric signature over *certifyInfo* using the key referenced by *signHandle*

Changes Needed: **YES**

ECC is broken need to be updated with another algorithm

Command | Description | Parameters | Response	Changes Needed
TPM2_CertifyCreation	
This command is used to prove the association between an object and its creation data.	
	**YES** *ECC is broken need to be updated with another algorithm*
	

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_CertifyCreation
TPMI_DH_OBJECT+	@signHandle	handle of the key that will sign the attestation block Auth Index: 1 Auth Role: USER
TPMI_DH_OBJECT	objectHandle	the object associated with the creation data Auth Index: None
TPM2B_DATA	qualifyingData	user-provided qualifying data
TPM2B_DIGEST	creationHash	hash of the creation data produced by TPM2_Create() or TPM2_CreatePrimary()
TPMT_SIG_SCHEME+	inScheme	signing scheme to use if the *scheme* for *signHandle* is TPM_ALG_NULL
TPMT_TK_CREATION	creationTicket	ticket produced by TPM2_Create() or TPM2_CreatePrimary()

Type	Name	Description
TPM_ST	tag	see clause 6
UINT32	responseSize	
TPM_RC	responseCode	
TPM2B_ATTEST	certifyInfo	the structure that was signed
TPMT_SIGNATURE	signature	the signature over *certifyInfo*

Chapter 4 FutureTPM Reference Architecture

FutureTPM is envisioned as a technical solution providing security, privacy and assurance services to the deployed platforms, with a special focus on the potential threats that quantum computers pose when they become a reality in the following few decades. As such, it will provide a framework supporting the design, test and validation of, not only a QR TPM, but also a set of mutually interconnected components that will help ensuring security properties. The goal is that the resulting solution offers effective means to guarantee the specific needs in terms of cybersecurity, privacy and trust, and remain secure not only today, but also in the long term against attacks carried out by adversaries possessing quantum capabilities. This chapter aims at giving more details, about the phases and the components, to serve as a guide for the development of the project.

4.1 FutureTPM Conceptual Architecture

FutureTPM is based on the technological axes described in Chapter 2 (TPM, Risk Management, and Security Policy Enforcement) to build a Trusted Computing framework, whose principal components are:

- the QR TPM, which will be the core component,
- the device hosting the QR TPM,
- the Risk Assessment component, composed of real-time and offline modules,
- the Security Policy Enforcement component.

Privacy, security and trust will be integrated from the very beginning in the development of the project. In order to combine all these technological axes together and build the proposed architecture, we propose a methodology that splits the FutureTPM development into three interlocking phases (Figure 6). Two of them are the core technical phases (**Design and Risk Management Phases**) and there is an overall Implementation Phase which is orthogonal to the whole duration of the project, consisting of the implementation and integration of the individual FutureTPM technical components.

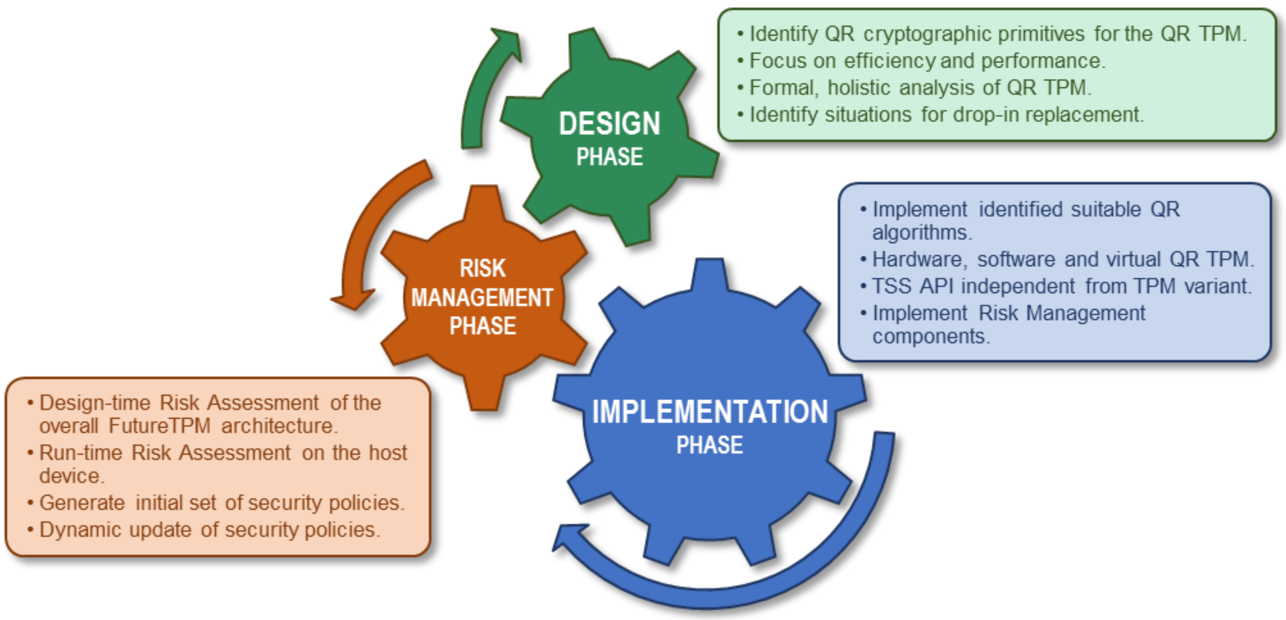

• Identify QR cryptographic primitives for the QR TPM.
• Focus on efficiency and performance.
• Formal, holistic analysis of QR TPM.
• Identify situations for drop-in replacement.
DESIGN PHASE
• Implement identified suitable QR algorithms.
• Hardware, software and virtual QR TPM.
• TSS API independent from TPM variant.
• Implement Risk Management components.
RISK MANAGEMENT PHASE
IMPLEMENTATION PHASE
• Design-time Risk Assessment of the overall FutureTPM architecture.
• Run-time Risk Assessment on the host device.
• Generate initial set of security policies.
• Dynamic update of security policies.

Figure 6: High-level FutureTPM orchestration.

Design Phase. Comprises the identification, design and development of cryptographic primitives and algorithms for use in the QR TPM, concretely, in terms of the two core TPM services: *secure storage* and *attestation*. This phase will focus on the principles of holistic security modelling, formal verification and modular design of QR algorithms to replace current non-quantum safe cryptographic primitives. Moreover, the concept of crypto agility (i.e., the ability to replace cryptographic algorithms without the need to rewrite the entire specification) will also play an important role during this phase. Crypto agility highlights the need for current algorithms to be prepared for transition to new versions, if they prove to be insecure in the future. For some protocols, QR security can be achieved by means of cryptographic agility through replacing the appropriate primitives. Unfortunately, this situation highly depends on the protocol, and no general claim can be made in this regard. In cases where algorithms that are not yet QR are used, such as DAA, adequate recommendations for outlining a migration path to QR security should be made within the scope of this task. It is also important to note that the identified algorithms must be not only secure, but also efficient, adapting to the different performance challenges of resource-constrained devices. The current TPM 2.0 Specification allows that a single TPM command can be shared among multiple algorithms. Therefore, FutureTPM will address the problem of letting multiple cryptographic algorithms be implemented efficiently with a minimum number of TPM commands. This phase will be carried out in WP2 and WP3.

Implementation Phase. Includes the implementation, integration and validation of the identified QR primitives (in the Design Phase) into the full range of possible TPM variants: hardware, software and virtual. The algorithm design and analysis must consider differences in the security threat model and tamper-resistant capability for each environment. Validation here refers to the demonstration of the security qualities of the specified QR algorithms, in the context of the specified Reference Scenarios, including formal verification, analysis, testing and performance evaluation. The FutureTPM includes the definition and implementation of all the necessary TSS functions, integrating the supported QR algorithms. This phase will be addressed in WP5, where the three QR TPMs will be developed, and used in the demonstrators of WP6.

Risk Management Phase. Comprises the development of a model to identify threats and vulnerabilities, and a risk assessment methodology for a TPM-based system. To develop a holistic QR TPM-solution, it is crucial to take into consideration the complex threat landscape posed by the ecosystem of the devices hosting the QR TPM. Failing to tackle adequately this increased surface of attack may allow an adversary with quantum capabilities mount successful attacks and recover secret information. Therefore, in addition to a risk assessment methodology during design-time, it is also required to develop a reactive run-time risk assessment and mitigation framework (for the whole TPM-based solution) to ensure security of use cases in the face of emerging threats and vulnerabilities. WP4 will be focused on this phase.

4.1.1 Communication between components

The flow of inputs and outputs among the different components can be described as follows. Based on the asset cartography, the cyber physical ecosystem envisioned on the use cases and databases of existing vulnerabilities, the offline risk assessment module is going to produce a risk graph. Using this information, the security policy enforcement module will be in charge of transforming the risk graph into a set of policies that will meet the security requirements of each Reference Scenario. These two steps will be executed during the Design Phase. The device hosting the QR TPM will be in charge of using the appropriate set of commands to comply with these policies.

Additionally, the client application that implements the real-time risk assessment component within the host device will be in charge of collecting evidence for the whole device and the TSS if a deviation from normal behaviour is detected. This will reflect the need to address attacks or vulnerabilities that were not identified during the Design Phase, e.g., zero-day vulnerabilities, but that have been detected during run-time. Control flow attestation coupled together with Berkeley Packet Filters will be leveraged to understand if the system is under attack in real-time. The output of the real-time risk assessment component will also populate the security policy enforcement module. Therefore, the framework offers an update mechanism of the policies considered during the Design Phase.

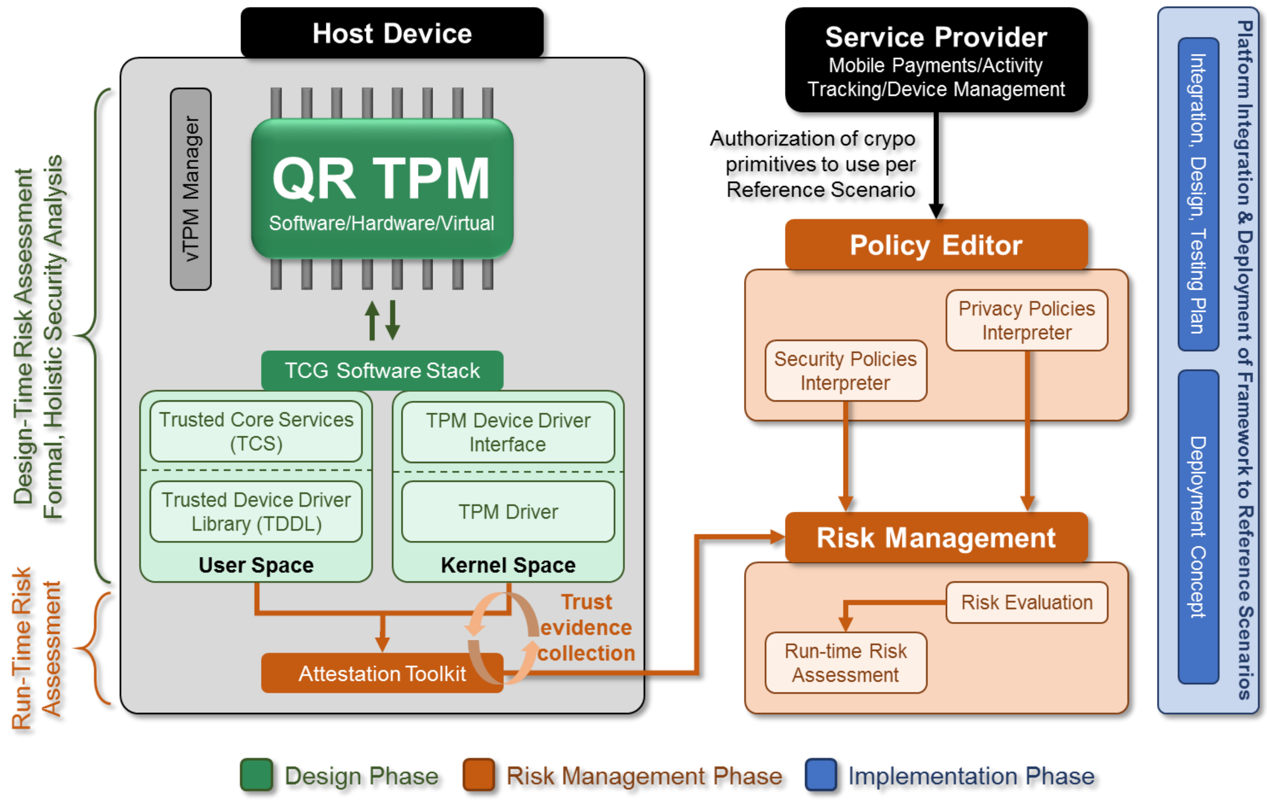

Host Device
vTPM Manager
QR TPM
Software/Hardware/Virtual
TCG Software Stack
Trusted Core Services (TCS)
Trusted Device Driver Library (TDDL)
User Space
TPM Device Driver Interface
TPM Driver
Kernel Space
Attestation Toolkit
Trust evidence collection
Design-Time Risk Assessment
Formal, Holistic Security Analysis
Run-Time Risk Assessment
Service Provider
Mobile Payments/Activity Tracking/Device Management
Authorization of crypo primitives to use per Reference Scenario
Policy Editor
Privacy Policies Interpreter
Security Policies Interpreter
Risk Management
Risk Evaluation
Run-time Risk Assessment
Platform Integration & Deployment of Framework to Reference Scenarios
Integration, Design, Testing Plan
Deployment Concept
Design Phase
Risk Management Phase
Implementation Phase

Figure 7: FutureTPM conceptual architecture.

Recall that the specific details of the implementation of the inputs and outputs for each component, and how they are going to be expressed will be made precise in the contexts of WP6. More specifically, in Deliverable D6.1 - Technical Integration Points and Testing Plan, where a detailed guideline will be provided, relating of how the different implementation components are going to be integrated and communicate with each other.

Table 17 summarizes the communication flow, the types of inputs and outputs expected, and where the concrete instantiations of the messages will be defined.

Table 17: Inputs and outputs between FutureTPM components.

Component		Expected type of inputs / outputs	From / to	Defined in
Offline Risk Assessment	Inputs	Reference Scenario asset cartography.	Security analyst	WP1, WP4
		Set of known vulnerabilities, e.g., US NIST CVE/CPE database.	Open vulnerability database	WP4
	Outputs	Risk graph	Security Policy Enforcement	WP3, WP4
		Risk assessment report	Security Policy Enforcement	WP4
Run-time Assessment (Attestation Toolkit)	Inputs	Trust evidence collection from program execution flow	Host device, TSS	WP4, WP5
		Security Policies	Security Policy Enforcement	WP3, WP4
	Outputs	Update on risk graph	Security Policy Enforcement	WP4
		Update on risk assessment report	Security Policy Enforcement	WP4
		TPM Commands that satisfy the security policies	TSS	WP5
Security Policy Enforcement	Inputs	Risk graph	Offline Risk Assessment	WP4
		Risk assessment report	Offline Risk Assessment	WP4
		Update on risk graph	Real-time Risk Assessment	WP4
		Update on risk assessment report	Real-time Risk Assessment	WP4
	Out.	Set of security policies	Real-time Risk Assessment	WP3, WP4
TSS	In.	TPM commands	Real-time Risk Assessment	WP5
QR TPM	In.	Marshalled TPM commands	TSS	WP2, WP5

Of course, we have to highlight that there will be a holistic, formal modelling of both the TSS and the QR TPM in order to identify trust assumptions, adversarial models and to understand in what cases the security properties identified hold. This will be carried in the context of WP3. This formal modelling will also provide feedback to WP4 to define the first version of the security policies and adversarial models during design-time.

4.2 QR TPM Architecture

The QR TPM architecture proposed for FutureTPM will essentially be based on the current TPM 2.0 Specification. In the following sections we describe the changes required in the architecture, which are essentially motivated by the adoption of the QR primitives.

4.2.1 QR TPM Components and Interfaces

While the elements identified in Section 2.1.1 as belonging to the TPM 2.0 architecture need to be updated to achieve QR security, the three variants of QR TPM to be implemented in FutureTPM will be based on the structure identified therein. This includes all the necessary updates to the TSS for the hardware, software and virtual QR TPM.

A hardware QR TPM may similarly be comprised of a security-aware general-purpose processor, a highly reliable NV memory, hardware accelerators for cryptographic operations, hardware true random number generators, and sets of PCRs. Nevertheless, the hardware accelerators will differ from those developed for the TPM 2.0 Specification, as cryptographic operations featured on that standard, based on RSA, EC, SHA-2, and others, are considered to be broken or offer reduced security in the presence of a quantum computer. Changes to the cryptographic operations may also be reflected on the size of the NV memory (which may need to store keys of larger size or contain state for hash-based signatures [71]) and on the size of PCRs. Standards for communication between the QR TPM modules and between the QR TPM and a main processor may be achieved via the protocols identified in Section 2.1.1. If the messages to be transmitted through these protocols are made larger as a result of changing cryptographic primitives, a larger communication overhead might be incurred on with the QR TPM.

In the context of the FutureTPM project, a Field-Programmable Gate Array (FPGA)-based coprocessor demonstrator will be targeted for the hardware QR TPM Reference Scenario (Secure Mobile Wallet and Payments). That is, the hardware QR TPM will actually be a software instance of a QR TPM running bare-metal on an FPGA. The details of this implementation will be specified within the scope of WP5.

Also, as already identified in Section 3.3 and Section 3.4, the SAPI offers functions that represent TPM commands [18], and many of the necessary changes to satisfy the requirements of FutureTPM, essentially motivated to achieve QR security in the Reference Scenarios, will be instantiated at this layer. Following a bottom-up approach, an investigation will be conducted so that these changes will be propagated and reflected appropriately in the remaining (upper) API layers. This may include, e.g., deprecating calls pertaining to RSA, EC, and any other non-QR operations, and giving access to the new QR primitives. Again, the QR TPM will be designed with a “plug-and-play” approach in mind, trying to minimize the propagation of changes to the upper layers (only where identified as necessary), so that it can smoothly replace its TPM 2.0 counterpart.

4.2.2 Commands and Data Communication Architecture

On a black-box view, QR cryptography (e.g. lattice-based cryptography) can be used as a substitute for RSA or ECC-based cryptography (the latter is being described in the Annex C of the TPM 2.0 Specificatcion – Part 1: Architecture [3]). In order extend the TPM 2.0 Specification with QR algorithms, it is necessary to introduce new data structures (e.g. for keys) and new TPM command codes. Subsequently, we will exemplify these steps for the QR key-encapsulation mechanism NewHope and the QR signature scheme qTesla.

4.2.2.1 NewHope and qTesla Data Structures

For integration of the selected QR schemes, we first define TPM 2.0-compatible C data structures for NewHope keys. NewHope and qTesla keys are treated as two separate key structures, where we give the respective public key structures below:

Code Listing 3: Proposed NewHope public key data structure.

```
typedef union {
  struct {
    UINT16 size;
    BYTE   buffer[MAX_PUBLIC_KEY_NEWHOPE_BYTES];
  } t;
  TPM2B b;

} TPM2B_PUBLIC_KEY_NEWHOPE;
```

Code Listing 4: Proposed qTesla public key data structure.

```
typedef union {
  struct {
    UINT16 size;
    BYTE   buffer[MAX_PUBLIC_KEY_QTESLA_BYTES];
  } t;
  TPM2B b;

} TPM2B_PUBLIC_KEY_QTESLA;
```

4.2.2.2 NewHope and qTesla Key Generation

For the integration of NewHope and qTesla key generation, the commands TPM2_Create and TPM2_CreatePrimary need to be extended. In doing so, the central function CryptCreateObject must be modified to dispatch to the respective key generation algorithms.

4.2.2.3 qTesla Sign and Verify Commands

The qTesla signature creation and verification will be triggered by the TPM2_Sign and TPM2_VerifySignature commands. The code must then dispatch to the corresponding qTesla function depending on the key type.

The TPM uses signature generation also for other commands (e.g. TPM2_Certify, TPM2_Quote). To integrate qTesla with these other commands the qTesla implementation must be called by the central function CryptSign. The same holds for signature verification, which is also used by other commands (e.g. PM2_PolicySigned) via the central function CryptValidateSignature):

4.2.2.4 NewHope Key En- and Decapsulation Commands

In the following, we show how to add TPM commands for NewHope CCA encryption and decryption. The NewHope key encapsulation and decapsulation will be realized by two new TPM commands and their corresponding functions.

- The command code TPM_CC_NEWHOPE_Enc (0x00000199) dispatches to the TPM2_NEWHOPE_Enc function with the following in/out parameters:
    - In: TPMI_DH_OBJECT key handle for encryption public key
    - In: TPM2B_CIPHER_NEWHOPE for the cipher object
    - Out: TPM2B_SHAREDSECRET_NEWHOPE for the shared secret
- The command code TPM_CC_NEWHOPE_Dec (0x00000198) dispatches to the TPM2_NEWHOPE_Dec function with the following in/out parameters:
    - In: TPMI_DH_OBJECT key handle for decryption private key
    - Out: TPM2B_CIPHER_NEWHOPE for the cipher object
    - Out: TPM2B_SHAREDSECRET_NEWHOPE for the shared secret

4.2.2.4.1 Integration of PQ Algorithms

The TPM uses encryption also for other commands (e.g. TPM2_Duplicate or TPM2_Rewrap), which must be integrated with the qTesla implementation (via the central function CryptSecretEncrypt).

The same holds for decryption where the qTesla implementation must be integrated for use with by commands (e.g. TPM2_Rewrap or TPM2_Import) via the central function CryptSecretDecrypt.

4.2.2.4.2 Implementation Issues

The TPM 2.0 supports the NIST SP800-56A C(2e, 2s) key agreement scheme for ECC keys. This is basically a 2-pass scheme with mutual authentication and forward secrecy. In that scheme, two ECDH exchanges will be performed:

1. ss1= static-static ECDH for mutual authentication
2. ss2= ephemeral-ephemeral ECDH for replay protection and forward secrecy

The shared secret is then a hash value of the concatenation of ss1 and ss2.

The CCA NewHope variant cannot perform a static-static key agreement with mutual authentication: It can only perform an ephemeral-static agreement with one-sided authentication, whereas the CPA variant of NewHope can provide an ephemeral-ephemeral key agreement. So, one could implement a NewHope protocol with mutual authentication and forward secrecy by the following 3-pass protocol:

1. ss1 = Static-ephemeral NewHope CCA
2. ss2 = Ephemeral-static NewHope CCA
3. ss3 = Ephemeral-ephemeral NewHope CPA

The shared secret is then a hash value of the concatenation of ss1, ss2 and ss3.

One could therefore consider to additionally include NewHope CPA encapsulation and decapsulation command. Nevertheless, an integration thereof is up for discussion and needs further evaluation. More details will be given in the respective deliverables in WP5.

4.2.3 Entities

In general, FutureTPM does not have special requirements for entities for the QR TPM. We are going to focus on the requirements specified in Deliverable D1.1 [1] and identify all the entities involved. We recall that the technical and security requirement IDs are specified in Deliverable D1.1 [1], Sections 5.1 and 5.2.

- **TR.1.1.1:** The QR TPM requires NV random access memory, so the persistent storage hierarchy is utilized here alongside with the NV indices.
- **TR.1.1.2:** There should be enough number of PCR entities to achieve the goals of FutureTPM Reference Scenarios' requirements.
- **TR.1.1.3:** No entities were identified in this requirement.
- **TR.1.1.4:** The QR TPM should support enhanced authorization. There is a related persistent entity, the password authorization session that is a part of the enhanced authorization specification. Because of this, the QR TPM will be designed designed to have greater levels of security, so that the authorization process should use more than just a password for authorization. To achieve this, the enhanced authorization can be implemented with message authentication codes and signatures as defined in the TPM 2.0 Specification [3]. In order for this method to be QR, it should be enriched with suitable algorithms identified in WP2.
- **TR.1.2.1-2:** No entities were identified in these requirements.
- **TR.1.3.1-5:** No entities were identified in these requirements.

- **TR1.3.6:** The QR TPM should have some backwards compatibility. In order to achieve that, it should support a variety of algorithms and this must reflect on the entities it implements. More specifically, the PCRs are grouped in banks according to the algorithm they use for hashing, as evident, there must be enough PCR banks in order to support the wanted backwards compatibility.
- **TR.1.4.1:** At least one major OS should be supported, like Linux. A Linux environment can be configured to work on most devices but it is mostly used in a PC. According to the PC Client Platform Specification, there are some numbers that the TPM should achieve, namely it should have:
    - The ability to hold 3 transient objects in TPM RAM.
    - The ability to hold 7 persistent objects in TPM NV memory.
    - The ability to hold 3 authorization sessions in TPM RAM.
    - It can manage 64 authorization sessions concurrently.
    - It should have at least 24 PCRs.
    - It should have at least 6962 bytes of NV memory.
- **TR.2.1.1-2:** These are two requirements which specify that it is desirable to have performance close to non-QR TPMs. As for all performance requirements, there should be enough RAM and NV memory for the algorithms defined in WP2 to have acceptable run times.
- **TR.2.2.1-4:** No entities were identified in these requirements.
- **TR.2.2.5:** In order to enhance a virtual TPM to provide better security guarantees closer to that of a physical TPM, we need better measurements. PCRs are used to measure the state of software and help the TPM to attest to its validity. A virtual QR TPM could use more PCRs in order to have a more fine-grained measuring capability.
- **TR.2.2.6:** The ability to run arbitrary code within the QR TPM requires that enough memory (RAM and NV memory) is present to support this functionality.
- **TR.2.3.1-4:** No entities were identified in these requirements.
- **SR.1.1.1-8:** No entities were identified in these requirements.
- **SR.1.2.1-4:** In order to support the new QR cryptographic primitives, we need adequate RAM and NV memory.
- **SR.1.3.1:** There should be support for measuring and reporting that uses QR algorithms. This directly impacts PCRs, as we should use QR hash functions in order to store the software state (measuring) and also use QR signing primitives to sign this measurement and report them to external auditors.
- **SR.1.3.2-3:** No entities were identified in these requirements.
- **SR.1.4.1-2:** No entities were identified in these requirements.
- **SR.1.4.3:** Credentials should be safely stored and protected from eavesdropping. The storage hierarchy should have a branch just for credentials and encrypt each one of them with a separate key in order to achieve the required level of security and protect this sensitive data.
- **SR.2.1.1-7:** These requirements build upon SR.1.2.1-4 and extend the QR requirements. In order to achieve this higher level of quantum resistance, the QR TPM is going to need the appropriate amount of internal storage and RAM.
- **SR.2.2.1-6:** No entities were identified in these requirements.

4.2.4 Hierarchies

As in the previous section, there are no specific needs for the QR TPM beyond what TPM 2.0 offers, in terms of control domain hierarchies.

Table 18 summarizes the usage of the different hierarchies in the scope of the three Reference Scenarios, along with a tentative list of relevant commands used by the different agents within their set of user stories. We exclude from the list some obvious common commands such as TPM2_Startup/TPM2_Shutdown. For a complete list and description of the commands, we refer the reader to the TPM 2.0 Library Specification - Part 3: Commands [6]. See Deliverable D1.1 [1], Chapter 4 for User Stories IDs.

We have to note that in the current stage of the project, discussions on the families of QR primitives that are going to be implemented, in the various TPM variants, is still ongoing in the context of WP2. There is still no final decision on method signatures for such primitives, or what commands need to be updated. Hence, the content from Table 18 mainly reflects the current TPM 2.0 commands pertaining to the usage in the different hierarchies. The inclusion of non-QR commands such as TPM2_RSA_Encrypt in the table reflects the potential need of a hybrid approach for using a combination of classical and QR algorithms. This comes in line with one of the main goals of the FutureTPM project, which envisions a smooth transition from current TPM 2.0 scenarios, using non-QR algorithms and protocols, to their QR version.

Table 18: Usage of hierarchies in FutureTPM.

Hierarchy	User Stories	Tentative List of Commands
Reference Scenario 1: Secure Mobile Wallet and Payments (INDEV)		
Platform	N/A	N/A
Endorsement	N/A	N/A
Storage	INDEV.AU.1 INDEV.AU.2 INDEV.AU.3 INDEV.AU.4 INDEV.AU.5	TPM2_Create,TPM2_CreatePrimary, TPM2_Load, TPM2_LoadExternal,TPM2_ReadPublic, TPM2_CreateLoaded,TPM2_RSA_Encrypt, TPM2_RSA_Decrypt, TPM2_EncryptDecrypt, TPM2_EncryptDecrypt2, TPM2_HMAC, TPM2_VerifySignature, TPM2_Sign, TPM2_NV_DefineSpace,TPM2_NV_Write, TPM2_NV_Read,TPM2_NV_ReadPublic
Null	No specific User Story	TPM2_Hash, TPM2_GetRandom
Reference Scenario 2: Personal Activity and Health Kit Data Tracking (S5)		
Platform	No specific User Story	TPM2_PCR_Allocate, TP_SetAlgorithmSet, TPM2_PCR_SetAuthPolicy.
Endorsement	S5.IU.4 S5.IU.6 S5.IU.7 S5.DA.1 S5.DA.2 S5.DA.4 S5.AE.3 S5.AE.4 S5.AE.5	TPM2_MakeCredential, TPM2_ActivateCredential, TPM2_Certify, TPM2_CertifyCreation, TPM2_Quote, TPM2_VerifySignature, TPM2_PCR_Extend, TPM2_PCR_Event, TPM2_PCR_Read, TPM2_PolicyPCR, TPM2_PolicyAuthorize, TPM2_PolicyPassword.
Storage	S5.DA.1	TPM2_Create, TPM2_CreatePrimary, TPM2_Load, TPM2_LoadExternal, TPM2_ReadPublic, TPM2_CreateLoaded, TPM2_RSA_Encrypt, TPM2_RSA_Decrypt, TPM2_EncryptDecrypt, TPM2_EncryptDecrypt2, TPM2_HMAC, TPM2_VerifySignature, TPM2_Sign, TPM2_NV_DefineSpace, TPM2_NV_Write, TPM2_NV_Read, TPM2_NV_ReadPublic.
Null	No specific User Story	TPM2_Hash, TPM2_GetRandom.

Hierarchy	User Stories	Tentative List of Commands
Reference Scenario 3: Device Management (HWDU)		
Platform	No specific User Story	TPM2_PCR_Allocate, TPM2_SetAlgorithmSet, TPM2_PCR_SetAuthPolicy.
Endorsement	HWDU.NA.1 HWDU.NA.4	TPM2_Unseal, TPM2_Certify, TPM2_CertifyCreation, TPM2_Quote, TPM2_VerifySignature, TPM2_PCR_Extend, TPM2_PCR_Event, TPM2_PCR_Read, TPM2_PolicyPCR.
Storage	HWDU.NO.1	TPM2_Create, TPM2_CreatePrimary, TPM2_Load, TPM2_LoadExternal, TPM2_ReadPublic, TPM2_CreateLoaded, TPM2_RSA_Encrypt, TPM2_RSA_Decrypt, TPM2_EncryptDecrypt, TPM2_EncryptDecrypt2, TPM2_HMAC, TPM2_VerifySignature, TPM2_Sign, TPM2_NV_DefineSpace, TPM2_NV_Write, TPM2_NV_Read, TPM2_NV_ReadPublic.
Null	No specific User Story	TPM2_Hash, TPM2_GetRandom.

As examples of command usages, we describe below how some of the commands are used in a few user stories:

- **INDEV.AU.1** ("As an Individual User I want to log in to the FreePOS Service."): This User Story is clearly linked to the Storage Hierarchy. Upon completion of the OAuth 2.0 protocol flow, the client application receives an Access Token, which is the credential that must be used to act on behalf of the user, and access the protected resource in the network, authenticating between the client and the business logic. This token can therefore be stored in the NV memory of the TPM through TPM2_NV_Write. Alternatively, it can be externally encrypted using TPM2_EncryptDecrypt, with a key generated through TPM2_Create. The resulting encryption key will be created using a parent key from the Storage Hierarchy.
- **S5.AE.5** ("As the S5Tracker Analytics Engine I want to prove that as a platform I have not been compromised regarding my initial configuration, so that I am trusted by other entities"): In this user story, the S5Tracker Analytics Engine generates a PCR quote through TPM2_Quote, after the CRTM has executed a number of measurements using TPM2_PCR_Extend. This can be the case of either a Static or Dynamic RTM.
- **HWDU.NO.1** ("The Network Operator connects the router to the network"): This User Story requires the creation of secure communication channels, which in turn requires the usage of asymmetric encryption, achieved through TPM2_RSA_Encrypt, TPM2_RSA_Decrypt. However, within the scope of FutureTPM, a QR public key cryptosystem should be considered instead of the quantum-insecure RSA.

4.2.5 PCRs

There are four mandatory technical requirements identified in Deliverable D1.1 that are directly linked to the PCRs, namely,

- **TR.1.1.2:** It should provide a small set of platform configuration registers (PCR) [2 Reference Scenarios, 13 User Stories];

- **SR.1.3.1:** Support software measurement (PCR extend) and measurement reporting (Quote), using QR algorithms) [2 Reference Scenarios, 13 User Stories];
- **SR.1.3.2:** Support remote attestation functionalities) [2 Reference Scenarios, 13 User Stories];
- **SR.1.3.3:** Support sealing and binding operations [1 Reference Scenario, 5 User Stories].

In general terms, the operational behaviour and commands of the PCRs, as described by the TPM 2.0 Specification [3] do not need any major changes to satisfy the technical requirements of the three Reference Scenarios from Deliverable D1.1. However, taking into consideration the particular instance of the PCR requirements from the PC Client Platform Specification [12], there are several considerations that have to be taken into account for the particular instance of PCRs for the QR TPM.

Algorithms. As described in Section 2.1.5 above, PCRs are grouped into banks, and all the PCRs within the same bank are updated using the same hash algorithm. Therefore, new constants will be required to reference the PCR banks extended with the new hash functions. See Section 4.3 below.

Number of PCRs. The PC Client Platform Specification [12] requires a minimum of 24 PCRs, as described in Table 6, where 16 are devoted to Static RTM measurements, and one is devoted to application measurements.

- Reference Scenario 2 (Personal Activity and Health Kit Data Tracking): The PCRs are used to verify the integrity and establish trust between the three components, namely, S5PersonalTracker, S5TrackerAnalytics and S5DataAnalysis. It is plausible that this Reference Scenario require DRTM, and therefore, an increased number of PCRs with respect [12] will probably be necessary.
- Reference Scenario 3 (Device Management): The PCRs are used to verify the integrity of the different network routers, and report their measures to the Network Management System. This scenario is likely to use both SRTM and DRTM.

As described in Section 2.1.4 above, the manufacturer through the Platform Hierarchy can allocate an arbitrary number of PCRs, if sufficient memory is available for the requested allocation. This is possible with the restriction that it is not possible to allocate more PCR in any bank than there are PCR attribute definitions. However, the PCR allocation will be retained only until the next TPM2_Startup(TPM_SU_CLEAR) is executed. Therefore, the number of PCRs that FutureTPM should support at least is the minimum required to satisfy Reference Scenarios 2 and 3. The exact number of required PCRs will be identified in the context of WP5 and WP6.

Attributes. The attributes (defined in Section 2.1.5.3 above) of each PCR index in each bank are retained across different banks. Although the TPM 2.0 Specification allows ample degree of freedom in the selection of the values of the PCR attributes, the QR TPM architecture should maintain similar, if not equal, values for them as those defined in the PC Platform Client Specification [12]. If further sets of PCR indexes are required, and unless there is a clear justification for a particular requirement, they should mimic as much as possible the values of the most similar currently defined PCR index from [12].

4.3 Cryptography Subsystem, Keys and Key Operations

The high level description of the cryptography subsystem, keys and key operations will be similar to the one described in Section 2.2 for TPM 2.0. The main difference will be the particular set of algorithms that are going to be used (which need to achieve a certain level of quantum-resistance guaranties), the increase in key/output lengths, and any change required to the API to allocate non-existent algorithms in TPM 2.0 with specific requirements. See, e.g., Section 3.4 and Section 4.2.1 above. Table 19 summarizes the difference between the set of algorithms used in the FutureTPM QR TPM versus the ones used in TPM 2.0. This table is based on the list of QR candidates proposed in Deliverable D2.1, and the current TPM 2.0 set of algorithms.

Table 19: Algorithm comparison between TPM 2.0 and FutureTPM.

Primitive type	Defined in TPM 2.0 and abandoned in FutureTPM	Defined in TPM 2.0 and maintained in FutureTPM	New in FutureTPM
Hash function	SHA-1	SHA-256, SHA-384, SHA-512, SHA3-256, SHA3-384, SHA3-512, SM3	SHAKE128, SHAKE256, BLAKE2, PHOTON
KDF		KDF2, NIST SP800-108, NIST SP800-108	
Asymmetric	RSA, ECC		NewHope, Frodo, Kyber, BIKE
Symmetric	SM4	AES, XOR, Camellia	Serpent, Twofish
Signature	HMAC (ISO/IEC 9797-2), RSASSA (IETF RFC 3447), RSASSA-PSS (IETF RFC 3447), RSASSA-PKCS1-v1_5 (IETF RFC 3447), ECDSA, ECDAA, SM2, EC Schnorr		Dilithium, Tesla, pqNTRUSign, FALCON, SPHINCS

The inclusion of these new algorithms require the definition of new constants to identify them. In the TPM 2.0 Specification, the algorithm identifier that references a given algorithm is a 32-bit unsigned integer of type TPM_ALGORITHM_ID (formerly named TPM_ALG_ID in TPM 1.2). This type is instantiated then as interface types. We will take as an example the case of hash functions. For hash functions, TPM_ALGORITHM_ID is instantiated as the interface type TPMI_ALG_HASH. The current TCG Algorithm Registry [45] defines the following constants for hash algorithms:

Table 20: Currently defined hash algorithm identifiers in TPM 2.0.

Constant	Value	Reference
TPM_ALG_SHA	0x0004	ISO/IEC 10118-3
TPM_ALG_SHA1	0x000D	ISO/IEC 10118-3 (redefinition for documentation consistency)
TPM_ALG_SHA256	0x000B	ISO/IEC 10118-3
TPM_ALG_SHA384	0x000C	ISO/IEC 10118-3
TPM_ALG_SHA512	0x000D	ISO/IEC 10118-3
TPM_ALG_SM3_256	0x0012	GM/T 0004-2012
TPM_ALG_SHA3_256	0x0027	NIST PUB FIPS 202
TPM_ALG_SHA3_384	0x0028	NIST PUB FIPS 202

Constant	Value	Reference
TPM_ALG_SHA3_512	0x0029	NIST PUB FIPS 202

Therefore, it should be taken into account the set of hash functions identified in Deliverable D2.1 as potential candidate that are not included in the current algorithm registry. Namely, constants would be required for BLAKE2, Photon, and Lesamnta-LW for the recommended bit size security, for the implementations that will include these algorithms.

- TPM_ALG_BLAKE2B_256
- TPM_ALG_BLAKE2B_384
- TPM_ALG_BLAKE2B_512
- TPM_ALG_BLAKE2S_128
- TPM_ALG_BLAKE2S_160
- TPM_ALG_BLAKE2S_224
- TPM_ALG_BLAKE2S_256
- TPM_ALG_PHOTON_80
- TPM_ALG_PHOTON_128
- TPM_ALG_PHOTON_160
- TPM_ALG_PHOTON_224
- TPM_ALG_PHOTON_256
- TPM_ALG_LESAMNTALW_256

In a similar fashion, the same approach should be taken for the rest of primitives (KDFs, asymmetric, symmetric, and signature).

As also aforementioned, we would like to stress that the decision on the specific algorithms to be implemented for each Reference Scenario is still under discussion, and it is not within the scope of this deliverable to make such a decision. The set of algorithms presented in this section is an initial, tentative selection, based on the output of Deliverable D2.1, which produced a list of potential QR algorithm candidates to be further investigated and implemented in the FutureTPM project. The first recommendation of what particular algorithms are going to be used will also come within the context of WP2 in Deliverable D2.2. This set of recommendations might be updated, if needed, according to the progress of WP2, WP5, and WP6, when it comes to the QR TPM implementation and demonstration of the Reference Scenarios. Deliverable D2.3 will provide the final list of QR algorithms, where any required change and corrective measure will be indicated, in the case where any problem had been identified in the initial set of suggested algorithms. In conjunction with WP2, a more complete description of the set of QR families to be implemented in each environment will also be addressed in the context of WP6.

For instance, for the hardware QR TPM variant, there is the following agreement regarding crypto primitives to be supported: **NewHope for asymmetric encryption, AES for symmetric encryption, qTesla for digital signature, NIST SP800-56A for KDF, and supported hash functions of TPM 2.0**.

4.4 Authorization and Sessions

The authorisations and sessions described in Section 2.3 need to be updated to achieve QR security in the FutureTPM QR TPM. In particular, hashing algorithms with larger outputs should be made available to ensure the security of the KDFs, HMACs and the extensions of policy and audit digests. Symmetric ciphers with larger blocks should be made available for encrypt/decrypt sessions when the CFB mode is used.

Furthermore, Enhanced Authorisation (EA) has been identified as a functional requirement of the project use cases. This type of authorisation is achieved through the policy sessions described in Section 2.3. Through the aforementioned updates of the FutureTPM project with respect to the description of sessions in Section 2.3, this functionality can be made QR.

4.5 Attestation Protocols

As aforementioned in Section 2.4, FutureTPM will build on-top of existing remote attestation techniques (namely, **DAA**, and a combination of **Static and Dynamic Attestation**) towards ensuring the internal state of a remote untrusted cyber-physical system in order to enhance its security and privacy posture. These methods will be used for achieving trustworthiness of the command execution flows which can be used in combination with Berkeley Packet Filters, discussed in Section 4.6, for the detection of any suspicious activities.

The reason behind employing attestation mechanisms as a means of operational assurance is twofold: First of all, one of the main challenges in managing device and network security in today's heterogeneous and scalable infrastructures is the lack of adequate containment and sufficient trust when it comes to the behaviour of a remote system that generates and processes mission-critical and/or sensitive data. An inherent property in FutureTPM is the codification of trust among computing entities that potentially are composed of heterogeneous hardware and software components, are geographically and physically widely separated, and are not centrally administered or controlled. By leveraging the artefacts of traditional security infrastructure (such as digital signatures, certificates and assurance statements) coupled with advanced crypto primitives (such as run-time property-based attestation) and building upon emerging trusted computing technologies and concepts, FutureTPM will convey trust evaluations and guarantees for each network entity.

This high level of trustworthiness which will not only include integrity of system hardware and software but also the correctness and integrity of the generated data flows will, in turn, reduce the overall attack vector and allow for the more effective operation of the FutureTPM security framework. This will allow the secure configuration, deployment and operation of distributed, scalable "Systems-of-Systems" infrastructures.

When it comes to the **DAA** schemes described in Section 2.4, they need to be updated to achieve QR security and privacy requirements, as has been already introduced in the context of Deliverable D2.1 [8]. This requires the update of all employed crypto primitives ranging from hash functions and asymmetric algorithms to signatures. In the context of FutureTPM, users will be in control of their own privacy and that of their devices. There is no dependence with the end device and/or final software application, thus, democratizing privacy and avoiding configuration problems or issues with software-level updates. Privacy enhancement will be achieved through the use of a QR-based DAA scheme as a building block in a large scalable deployment of cyber-physical systems.

For user device privacy and integrity and in order to cope with the ever-increasing attack surface targeting the dynamic execution properties of cyber-physical systems, FutureTPM shall enable the provision of ***automated and scalable behavioural-based attestation*** services, reflecting the identified (and configured during run-time) preventive, access control, information flow and functional safety policies that will be enforced by the Security Policy Enforcement component (Section 4.6.2). To this end, the properties (of interest) to be attested by the deployed cyber-physical systems will vary depending on the type of functionality that each such device offers. A generic assurance technique needs to be applied capable of coping with all these different specifications. FutureTPM aims to overcome this challenge by having a general behavioural-based attestation mechanism developed based on the novel concept of control-flow attestation.

Control-flow attestation is one of the most important dynamic properties at the software layer since it captures diverse instantiations of software exploits that hijack a program's control flow. Such attacks tamper with state information in the program's data memory area, e.g., the stack and the heap. Software bugs allow an attacker to arbitrarily alter state information and hijack the program flow of applications to induce malicious operations. While traditional attacks require the attacker to inject malicious code [72], state-of-the-art attacks such as return-oriented programming leverage code that is already present in the vulnerable application thereby bypassing modern mitigation strategies [73], [74]. In other words, the attacker resembles malicious codes through a combination of already existing benign code pieces. In contrast to traditional PC platforms and mobile phones, software exploits against Internet of Things (IoT) devices (as the ones envisioned in some of the Reference Scenarios) can have severe safety consequences. Consider a modern network which features a vast amount of heterogeneous hardware and software components with hundreds millions lines of code. A common theme of such composable infrastructures is that all of them are pushing the envelope with respect to how many application instances can be packed efficiently onto a certain physical infrastructure footprint. This co-existence of multiple micro-services, multiple applications, or even multiple tenants, enables a variety of Advanced Persistent Threats (APTs) to be exploited by adversaries.

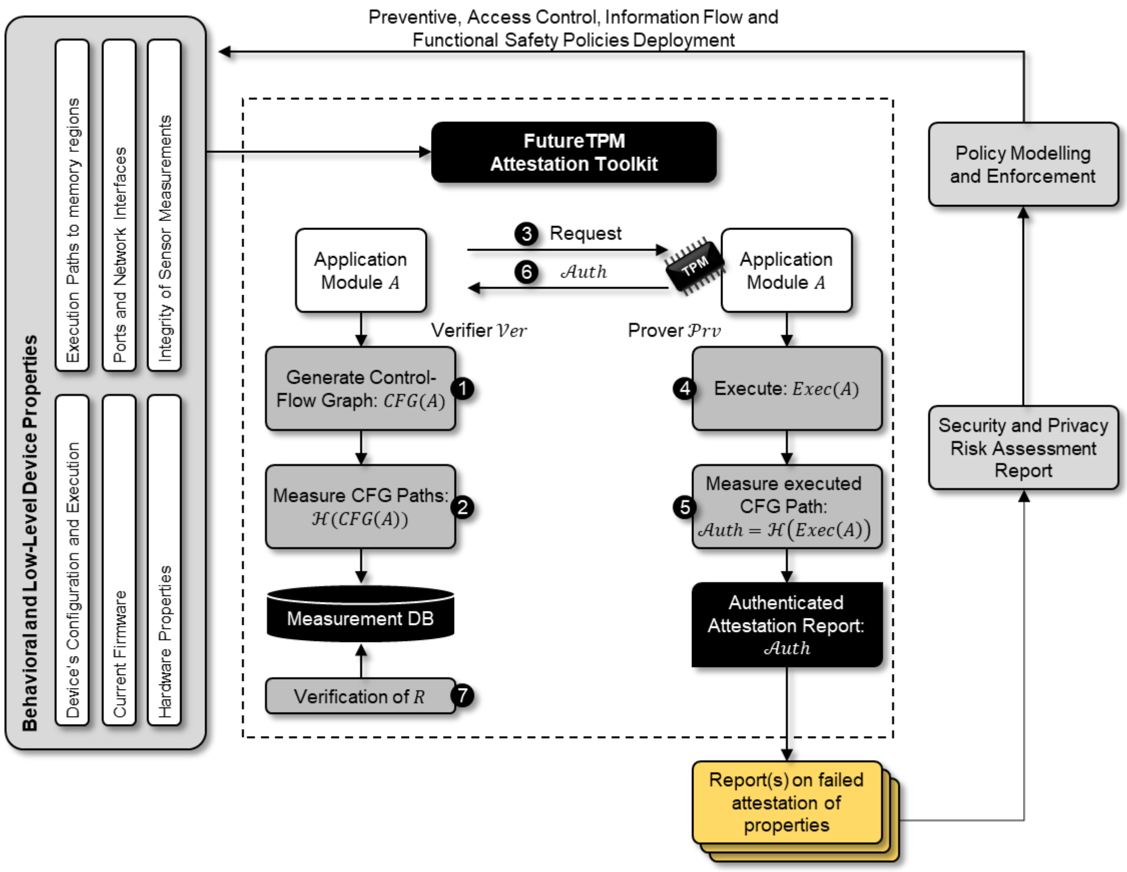

Preventive, Access Control, Information Flow and Functional Safety Policies Deployment
Behavioral and Low-Level Device Properties
Execution Paths to memory regions
Ports and Network Interfaces
Integrity of Sensor Measurements
Device's Configuration and Execution
Current Firmware
Hardware Properties
FutureTPM Attestation Toolkit
Application Module A
3 Request
6 $\mathcal{A}uth$
TPM
Application Module A
Verifier $\mathcal{V}er$
Prover $\mathcal{P}rv$
Generate Control-Flow Graph: $CFG(A)$ 1
4 Execute: $Exec(A)$
Measure CFG Paths: $\mathcal{H}(CFG(A))$ 2
5 Measure executed CFG Path: $\mathcal{A}uth = \mathcal{H}(Exec(A))$
Measurement DB
Authenticated Attestation Report: $\mathcal{A}uth$
Verification of R 7
Report(s) on failed attestation of properties
Policy Modelling and Enforcement
Security and Privacy Risk Assessment Report

Figure 8: FutureTPM behavioural-based attestation of a device's control flow.

The general approach of control-flow attestation we envision in FutureTPM is depicted in Figure 8. It shows a cyber-physical component (hosting a TPM), acting as the verifier, that first receives the necessary security policies containing the specifics of the properties to be attested. Based on the interpretation of these policies, it then computes all legitimate control-flow paths of an application, and store its measurements in a database (step 1 and 2). To trigger the run-time attestation, as dictated by an already defined security policy, the verifier sends a request to the device which acts as the prover (step 3). The prover device executes the software that the verifier desires to attest (step 4) and a trusted component measures the taken control-flow paths (step 5). For instance, this can be achieved through a hash function. Finally, the attestation result is send back to the verifier for validation (step 6 and 7). In the case of a failed attestation about a system's integrity, the information might not be sufficient to understand the device's behaviour. Thus, in this case, a more in-depth investigation of the system's behaviour is needed to detect any cheating attempts or if any type of (non-previously identified) malware is resident to the system. The goal of this functionality is to then feed this detailed analysis to the Risk Assessment component (Section 4.6.1) for dynamically defining new attestation policies against this newly identified attack vector.

4.6 Security Enforcement

FutureTPM will build on-top of existing models and will propose a holistic approach for **Risk Management** in TPM-based solutions. To this end, a complete assessment methodology complemented by a supportive tool in order to support the discrete steps of the methodology will be developed. A high-level overview of the Risk Assessment framework is depicted in Figure 9. The core of the framework is the QR TPM platform capable of creating and updating (in real-time) the risk graph based on the envisioned application and the types of security properties we want to be achieved. These properties can be described as security policies that have to be enforced to the devices hosting the QR TPM in order to secure the overall platform against the identified risks, or any new vulnerabilities that will be identified during run-time.

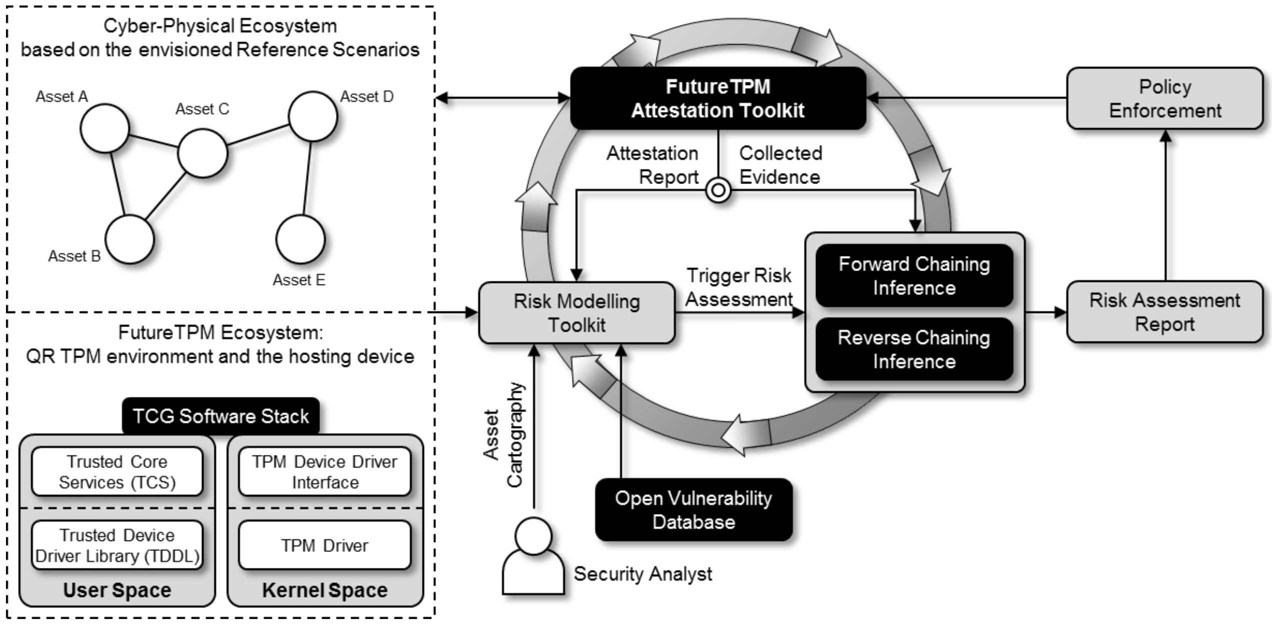


Figure 9: Risk Assessment Framework.

Furthermore, FutureTPM will perform a thorough vulnerability analysis of all identified threats and risks that can affect the final product. More specifically, there will be a security analysis of the various TPM environments both from a software and hardware point of view to ensure that the implementation does not undermine the overall security goals of the FutureTPM platform. Incorporation of widely known vulnerabilities and threat repositories (e.g., US NIST CPE/CVE) will provide a wide set of attack scenarios against which the resilience of the overall FutureTPM framework will be evaluated. In conjunction with the aforementioned process, there will also be a run-time security analysis of the algorithms to be identified in Deliverable D2.3 in order to evaluate their security and privacy properties and whether they are properly configured to achieve the required level of assurance of FutureTPM.

4.6.1 Risk Assessment for TPM and TSS

The overall architecture of the **Risk Assessment** platform will take into account all the necessary components and interfaces and how they can interact with the QR TPM and more specifically with the TSS. In particular, we will focus on the following functionalities provided by the QR TPM and operated through the TSS: a) the Components and Interfaces, b) the Commands and Data Communication Architecture, c) the Entities, d) the Hierarchies and e) the PCRs. To achieve all the activities of the Risk Assessment, the **Berkeley Packet Filter (BPF)** will be employed during design-time coupled together with the developed Control Flow Attestation mechanisms. BPF was designed in 1992 to act as a socket filter and allows working with raw link-layer packets. It consists of a virtual machine executing the bytecode, and of a specific language, somewhat close to assembly. It is possible to write programs in a subset of C and to compile them into BPF language. This language has been designed to provide easy manipulation of captured packets. It also results from strong concerns regarding **safety** and **security**: a BPF program always ends successfully, there can be no infinite loop. In order to attain this goal, several mechanisms have led the conception of BPF. Some examples include:

- Backward jumps are not allowed (no loops).
- Program length is limited to 4096 instructions.
- Instruction set has some built-in safety (no exposed stack pointer, instead load instruction has mem modifier).
- Provides dynamic packet boundary checks

Such measures enable user-defined BPF programs to be run by the in-kernel BPF machine. Running inside the kernel, in many situations, allows for performance gains since undesired packets can be dropped before even being copied to user space. In addition, BPF programs can be compiled right before execution (JIT, Just-In-Time compiling), hence benefiting from CPU optimizations. Thus, BPF is a very efficient tool to deal with packets, with performance roughly equivalent to native x86 code, but it has a limited set of instructions. Several extensions have been added over the years; but they are not to be confused with the extended version eBPF, which has been evolving since 2013.

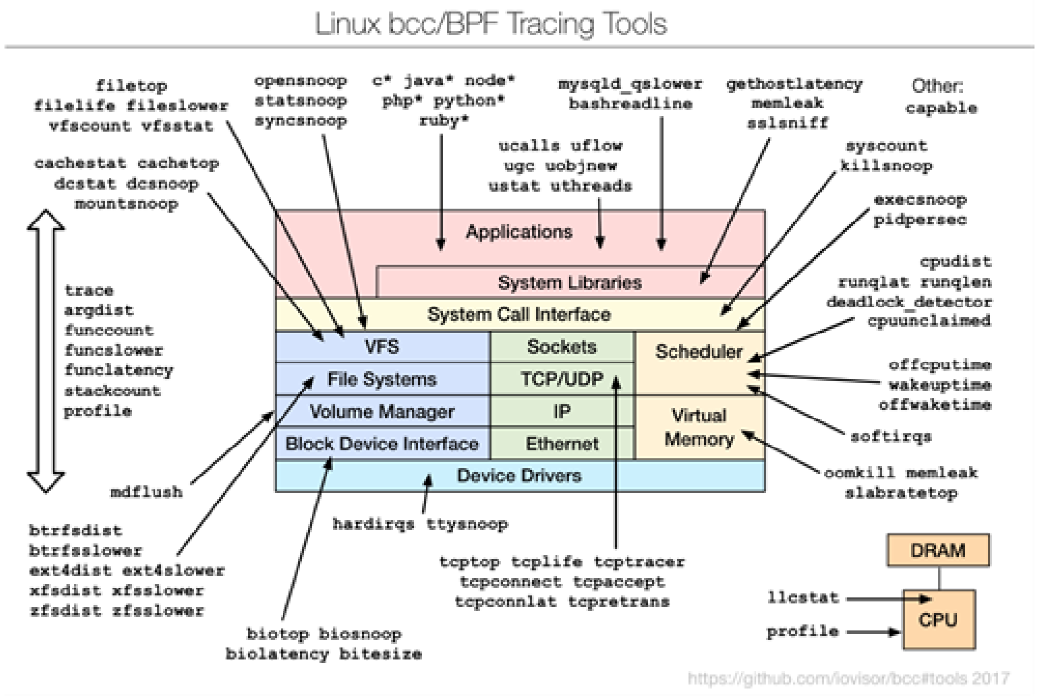


Figure 10: Tracing options [75].

eBPF extends BPF possibilities but preserves its safety measures, making it safe to run in the kernel of production systems. It redefines and extends the set of instructions, relying on common subset from several assembly languages. It can map some memory space so that it can be shared between user and kernel space. It also makes it possible to call certain kernel functions from programs. The JIT compiling is preserved. For instance, on x86 systems, the code is turned by the user space compiler into some “simplified x86 assembly”, which is in turn verified in the kernel, and then each “simplified” instruction is translated into real x86 by the JIT compiler. The process is efficient, as:

- all registers map one-to-one,
- most of instructions map one-to-one,
- BPF call instruction maps to x86 call.

These powerful features make eBPF suitable not only for packet filtering, but also for general networking, event tracing, kernel optimizations, and also for the Risk Assessment of the TSS. The goal is to use suitably tailored eBPFs to capture the execution of the command flows in the device (hosting the TPM) so that we can check and attest the integrity of the execution behaviour based on already defined policies. Figure 10 below depicts all the tracing feature supported from the eBPF.

4.6.2 Security Policy Enforcement Mechanism

FutureTPM will combine the output of Risk Assessment with re-active policy enforcement; building upon Artificial Intelligence through the usage of the Drools Expert system, where the model instances will be transformed in executable rules. This expert system will propose several mitigation controls that map to the real properties that have to be attested with the QR TPM during run-time. These properties may be a subset of the configuration and execution properties that are already defined or can be other newly identified, high-level properties that can further enable semantic remote attestation, i.e., attestation of dynamic, arbitrary and system properties as well as behaviour of executable code in an attempt to mitigate the newly discovered run-time vulnerabilities. The minimization of different risks will end-up in optimal enforcement of controls. Part of the enforcement is the attempt to attest new properties in the QR TPM deployment and, thus, may require the dynamic update of any identified security policies.

Such a policy enforcement will also allow for Policy-Based Access Control (PBAC) which is an access control model based on policy-based security management, which controls the access to resources by defining the rules and policy. There are many tools and frameworks for PBAC. Several PBAC frameworks are based on the IETF Framework for Policy-based Admission Control [76], which consists of the following main components, which can be co-located or distributed:

- **Policy Decision Point** (PDP) – The policy server responsible for handling events and making decisions based on events.
- **Policy Enforcement Point** (PEP) – This enforces the policy based on rules received from the PDP.
- **Policy Repository** – For storage or retrieval of policy information.
- **User interface** – For specifying, administering and editing policy.

FutureTPM shall enable the composition of large scale “System-of-System” to be controlled via layered and cross domain authorization decisions based upon attestation. Such decisions shall be made at each layer (and across layers) to determine whether subsystems/systems conform to policies based upon the properties to which they can attest.

The Security Policy Enforcement platform will contain a Policy Admission Point, the logical component responsible for creating policies and policy sets and makes them available to the PDP. The policies created within the PAP are to be determined by the outputs of the Risk Management Phase. While policy creation will be managed by the PAP, a Policy information point (PIP) shall act as the source of the attribute values referred to within policies. In the context of FutureTPM, attributes shall be the properties attested to by a component and the verification result of the attestation. The PIP shall be responsible for requesting and receiving such information from the attestation services. The outputs of the RA framework shall aid in determining the specific attributes managed by the PIP. The PDP is the logical component that shall evaluate the attested to properties/attributes against applicable policies and makes the final authorization decision. While the successful verification of the attestation provides evidence that the information supplied is correct, the PDP decides whether the collated information supplied sufficiently demonstrates conformance to policy. Where the PDP determines that policy has not adhered to, it may also be necessary to feed such information back into the RA framework so as to initiate the process of evidence collection and runtime verification for performing a more in-depth vulnerability analysis of the failed-to-attest system.

4.6.3 Risk Assessment Components and Interfaces

The main Risk Assessment components are the **Risk Modelling Toolkit** and the **Risk Quantification Engine**. The following subsections describe these main components as well as the intercommunication between them.

4.6.3.1 Risk Modelling Toolkit

A security analyst will use this Risk Modelling Toolkit to model several processes that are performed through the synergy of the various services within a TPM environment. The Risk Modelling Toolkit will allow the creation of asset cartographies i.e. the formal representation of the assets and their relationship. In parallel, the toolkit will make use of open databases in order to associate the modelled assets with existing vulnerabilities (e.g., Common Vulnerabilities & Exposure – CVE - database). Finally, the properties that can be potentially attested per each asset will be also provided as an input to the Risk Modelling Toolkit.

4.6.3.2 Risk Quantification Engine

After the model creation, the security analyst may trigger the Risk Quantification Engine: This engine will be multi-threaded (by-design) since each separate risk quantification request requires different set of calculations. The Drools [77] efficient expert system will be used as a cornerstone component for the sake of the engine implementation. This technical choice implies that several rules have to be created automatically regarding the calculation of vulnerabilities, generation of attack trees, propagation of an exploitation (cascading effect), etc.

Chapter 5 Implementation Aspects of Reference Architecture

In this chapter, we try to elaborate on the approach that will be followed in order to realize the functionalities described in this document and to implement the components that constitute the FutureTPM framework and have been introduced in Chapter 4.

It has to be stated that the FutureTPM framework will be realized with two major releases based on implementation cycles of all internal mechanisms, software components and toolkits. The whole cycle of activities will be iterative: The first implementation cycle is going to be completed by the end of M18 with the delivery of the first release of the FutureTPM components and mechanisms. This release will be tested in technical and functional terms and will result in the provision of the first major version of the overall FutureTPM framework in M21. The results of the evaluation, testing and design choices made in the first version of the prototype will be fed as input to the second implementation cycle for further refinement and improvements that will lead to the second release of all internal components on M27. The final FutureTPM platform will be delivered at the end of M33 with slight improvements derived and imposed from the envisioned demonstrators.

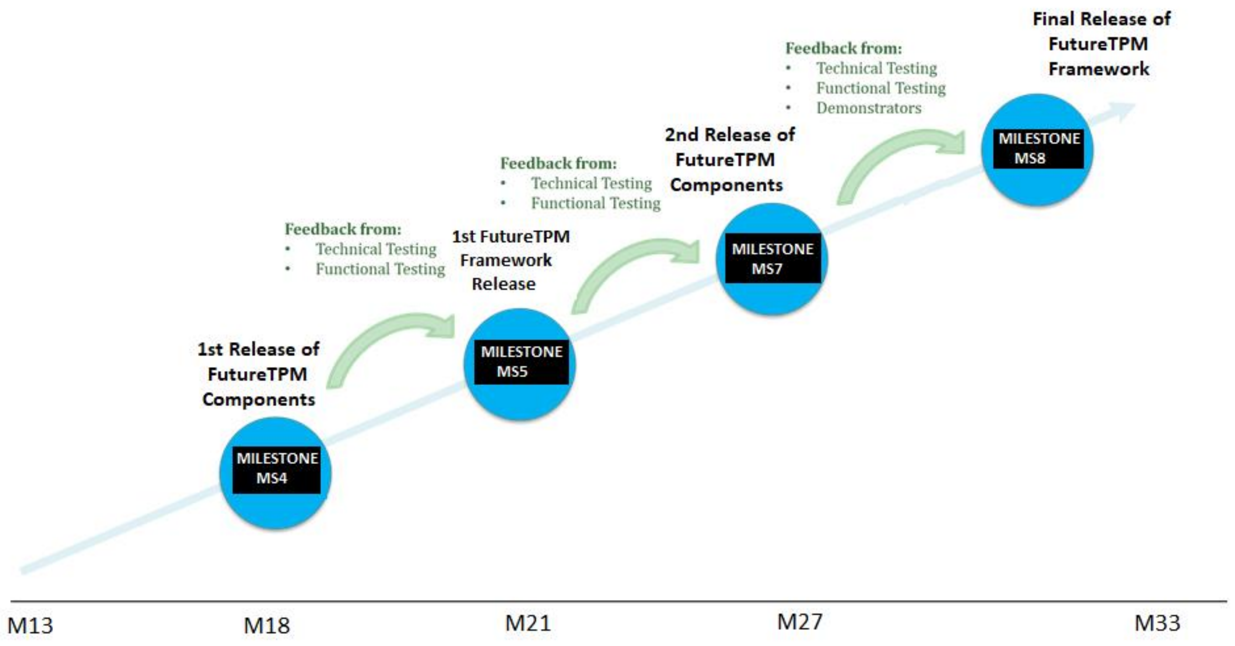


Figure 11: Major Releases of FutureTPM Integrated Framework.

This plan, as depicted in Figure 11, reflects only the major releases of the framework that are imposed with specific deadlines and milestones. The actual development of FutureTPM components' will be a continuous process imposing the continuous integration and testing of the developed mechanisms and toolkits in order to assure high quality during the entire lifetime of the project.

This process that will be followed by the consortium can be represented as a virtual circle that contains the following functional components: (*i*) source-code-versioning and management, (*ii*) continuous integration, (*iii*) quality assurance of generated code, (*iv*) persistent storage of generated builts (a.k.a. artefacts) and (*v*) issue/bug tracking. The decision for this workflow has been decided in a provisional way, at the early stages of the project, and there may be changes in the implementation aspects of the components during the project lifecycle and the tools that will be employed to support each step of this process. An initial selection of such tools, as depicted in Figure 12, is as follows: (*a*) Git for source code versioning, (*b*) Jenkins for continuous integration, (*c*) Sonar for code quality assurance, (*d*) nexus for artefact-management and (*e*) GitHub for issue/bug tracking.

In the following sections, we will briefly provide more information about these the selected tools and how these help the consortium to have a continuous pipeline for developing, integrating and testing of the FutureTPM Framework.

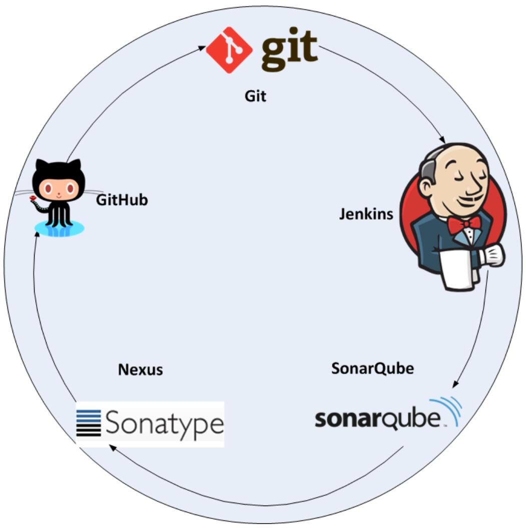


Figure 12: Development Lifecycle.

5.1 Version Control System

A Version Control System (VCS) is a repository of files, often the files for the source code of computer programs, with monitored access that tracks every change done in the filesystem, along with related metadata like date or person that changed each file. Each file that is tracked can be reverted to previous versions, while the exact changes in the file are usually available. Version control systems are essential for any form of distributed, collaborative development, as they provide the ability to collaborate on the same files, the ability to track each change that was made with great detail, and the ability to reverse changes when necessary.

In FutureTPM, the consortium has selected Git as the primary VCS system, due to its speed, distributed nature, branching capabilities, small size of the repository and the popularity of the online Git repository host and management platform of GitHub. A Git repository will created for the whole cycle of implementation and integration activities in the context of the project (WP2-WP6). Access to this repository will be limited to the consortium developers, but in later stages the consortium can decide to make the whole platform or some of the components public.

5.2 Continuous Integration

Continuous Integration (CI) is a software development practice where the members of a team frequently integrate their work – usually each contributor integrates his software code at least daily, leading to multiple integrations per day. Each integration cycle is verified by an automated build that includes testing in to detect integration errors as quickly as possible. This approach has the great benefit of reduced risk in the integration and therefore is a highly suggested practice on all distributed teams.

The selection of the consortium for CI is Jenkins [78], an open source tool written in Java, which runs in a servlet container, such as Apache Tomcat or the GlassFish application server. It supports Version Control tools like CVS, Subversion and Git and it can execute both Apache Ant and Apache Maven based projects, or even arbitrary shell scripts and Windows batch commands.

5.3 Quality Assurance

In a project like FutureTPM it is important to measure the quality of the developed software and the progresses in the development, as it is a software developed by distributed teams that create different components. Even though quality can be a subjective attribute, software structural quality characteristics will be clearly defined by the Consortium following the practices for IT Software Quality identified in the literature by CISQ, an independent organization founded by the Software Engineering Institute at Carnegie Mellon University. CISQ has defined 5 major characteristics of a piece of software that should be taken under consideration for the quality of a software; *Reliability*, *Efficiency*, *Security*, *Maintainability*, *Size*. These characteristics, among others, are very important and we will use SonarQube [79] in order to perform analysis of code quality and monitor the available metrics. SonarQube is an open source software quality platform that uses various static code analysis tools in order to extract software metrics, which then can be used to improve software quality. Basic metrics include duplicated code, coding standards compliance, unit tests coverage, code coverage, code complexity, identification of potential bugs by severity, percentage of comments. SonarQube is easily integrated Jenkins continuous integration pipeline.

5.4 Release Planning

The next step in the development lifecycle is the release planning and the management of the produced and required artefacts. An artifact repository is a collection of binary software artifacts and metadata stored in a defined directory structure and can be used by clients such Maven, Mercury, or Gradle to retrieve binaries during a build process. The introduction of an artefact repository it is crucial for distributed teams following the CI pipeline as it allows each new successful build to store the produced software components and make them available for the deployment of further development of the integrated framework. The release management in the FutureTPM project will be accomplished with the help of Nexus Repository Manager [80], but it is also tightly connected with the selected branching model.

In FutureTPM, we will use Git that allows to work with branches easily and in a structured way, so that different branches will help us to ensure the quality of the source code created and to decrease the number of failures. As usual in Git there will be a master branch, and this will be parted into a development branch, a release branch (that will be used for the major releases) and a possibly existing Hotfix-branch. Furthermore, separate branches can be created per implemented feature. Upon the completion of each feature the feature branch is merged in the development branch. Each commit that is performed in the development branch goes through the CI pipeline and creates updated versions of the binaries that are hosted in the Nexus. Official releases will also go through the CI and will be also hosted in Nexus release repository.

5.5 Issue Tracking

The last step of the development lifecycle is the issue/bug tracking, that requires a dedicated issue and bug tracking system. An issue tracker should be reachable for every developing partner needs to be included to collect development time issues like problem reports, feature requests, and work assignments. In the frame of Unicorn, for issues concerning coding, features and distribution, the GitHub issue tracker is chosen. The reporting is typically done by creating a new issue via the front end of the issue/bug tracker. The newly created issue is picked by the responsible FutureTPM developer.

Chapter 6 Summary and Conclusion

This final section will act as a synopsis of this deliverable and will summarize its findings. The scope of this deliverable was to provide the FutureTPM Reference Architecture. FutureTPM Reference Architecture aims to satisfy the functional and non-functional requirements that have been formulated during the requirements analysis and documented in FutureTPM D1.1 [1]. More specifically, Deliverable D1.1 highlighted specific functional and non-functional requirements and identified the FutureTPM actors that were required towards the formulation the FutureTPM framework.

By defining the FutureTPM Reference Architecture, we achieved the following: a) to define the architectural components that cover the functional aspects of the requirements, b) to map the identified roles to the aforementioned components, and c) to elaborate on each component by providing a usage walkthrough. At this point it should be clarified that the architecture is considered as ‘reference’ since it can be subjected to multiple ‘instantiations’. Furthermore, specific components can be implemented in a completely different way. In the frame of the project’s Implementation Phase a specific ‘instantiation’ of the components will be performed which will be tailored to the need of the use-cases. Furthermore, when defining the reference architecture, it is not within the scope of this document to recommend the specific set of QR algorithms to be used in each Reference Scenario (which is being investigated in the context WP2) neither define an exhaustive list of the QR TPM API requirements that will dictate all the subsequent changes needed during the implementation of the FutureTPM QR TPM variants (which will be addressed within WP5 and WP6).

The FutureTPM architecture will revolve around a fundamental element, namely, the QR TPM. As conceived by the TCG, TPMs are a fundamental piece in the context of Trusted Computing, and they provide a basis where secure systems can be built on. This document reviews the details of the current TPM 2.0 Specification that will be required to address the transition to a QR version. This includes aspects related to the inner components and interfaces, cryptography-related components, the communication subsystem, and how the different objects are structured within TPM 2.0. Other aspects that are reviewed to address the development of the FutureTPM framework are aspects related to Risk Management (both in design-time and in run-time) and Security Policy Enforcement.

As aforementioned, this document also relates the functional requirements already identified by the three Reference Scenarios in Deliverable D1.1 [1] to the FutureTPM functional and security requirements. This includes an analysis of the QR TPM interfaces and API updates (compared to the TPM 2.0 Specification) that will be driven by the requirements envisioned by the scenarios and the need of QR primitives. There are 34 TPM commands related to the QR security of the scenarios that have been identified as commands (possibly) requiring API updates.

Next, the document provides a description of the main components of the FutureTPM framework, describing the different mechanisms, communication interfaces and expected types of inputs and outputs between these components. A description of the required QR TPM updates in order to support the provision of various sets of QR primitives is also provided. As reflected in Chapters 3 and 4, many of the functionalities and components currently provided in the TPM 2.0 Specification will remain unchanged as they can already support the transition to a QR-based version (recall that one of the main goals of FutureTPM is to enable a ‘smooth’ transition between non-QR and QR cryptography), incorporating only the necessary changes as dictated by the identified QR requirements: larger sets of memory for supporting bigger keys, API changes for dedicated QR primitives, etc. The Risk Assessment and Security Policy Enforcement modules will be based on Berkeley Packet Filters, and the Drools expert system, respectively.

Finally, this document concludes with a proposal of the implementation plan and aspects for the overall FutureTPM framework implementation and integration. Based on this plan, FutureTPM framework will be realized with two major releases based on implementation cycles, each one leading to an improved version of the previous until we reach project’s Month 33, where the final release of FutureTPM will be launched. To assure the best quality throughout the project, a continuous integration and continuous delivery pipeline was decided that includes a) source-code-versioning and management, b) continuous integration, c) quality assurance of generated code, d) persistent storage of generated builds (a.k.a. artefacts) and e) issue/bug tracking. The tools that have been initially selected to build this pipeline are as follows: a) Git for source code versioning, b) Jenkins for continuous integration, c) Sonar for code quality assurance, d) nexus for artefact-management and e) GitHub for issue/bug tracking.

Chapter 7 List of Abbreviations

Abbreviation	Translation
AIK	Attestation Identity Key
BIOS	Basic Input/Output System
CFA	Control Flow Attestation
CFB	Cipher Feedback
DAA	Direct Anonymous Attestation
DRTM	Dynamic Root of Trust for Measurement
EK	Endorsement Key
ECC	Elliptic-Curve Cryptography
EPS	Endorsement Primary Seed
ECDH	Elliptic-curve Diffie-Hellman
ESAPI	Enhanced System API
FAPI	Feature API
FPGA	Field-Programmable Gate Array
HMAC	Hash-based Message Authentication Code
IV	Initialisation Vector
I2C	Inter Integrated Circuit
KDF	Key Derivation Function
LPC	Low Pin Count
NV	Non-Volatile
OS	Operating System
OCP	Open Core Protocol
PBAC	Policy-based Access Control
PCR	Platform Configuration Register
PPS	Platform Primary Seed
PRNG	Pseudo-Random Number Generator

QR	Quantum-Resistant
RA	Risk Assessment
RM	Resource Manager
RNG	Random Number Generator
RPC	Remote Procedure Call
RSA	Rivest-Shamir-Adleman
RTM	Root of Trust for Measurement
RTR	Root of Trust for Reporting
RTS	Root of Trust for Storage
SoC	System on Chip
SPI	Serial Peripheral Interface
SPS	Storage Primary Seed
SRK	Storage Root Key
SAPI	System API
TA	Trusted Application
TAB	TPM Access Broker
TC	Trusted Component
TCTI	TPM Command Transmission Interface
SRTM	Static Root of Trust for Measurement
TCG	Trusted Computing Group
TEE	Trusted Execution Environment
TPM	Trusted Platform Module
TSS	TPM Software Stack
UEFI	Unified Extensible Firmware Interface
XOR	Exclusive OR

Chapter 8 Bibliography

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

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

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

[4] Intel Corporation, *Intel® Trusted Execution Technology: White Paper,* 2012.

[5] Trusted Computing Group (TCG), *Trusted Platform Module Library - Part 2: Structures (Family 2.0, Revision 01.38),* 2016.

[6] Trusted Computing Group (TCG), *Trusted Platform Module Library - Part 3: Commands (Family 2.0, Revision 01.38),* 2016.

[7] Trusted Computing Group (TCG), *Trusted Platform Module Library - Part 4: Supporting Routines (Family 2.0, Revision 01.38),* 2016.

[8] The FutureTPM Consortium, “D2.1 - First Report on New QR Cryptographic Primitives,” 2018.

[9] Trusted Computing Group (TCG), *TPM Main - Part 1 Design Principles (Version 1.2, Revision 116),* 2011.

[10] Trusted Computing Group (TCG), *TPM Main - Part 2 TPM Structures (Version 1.2, Revision 116),* 2011.

[11] Trusted Computing Group (TCG), *TPM Main - Part 3 Commands (Version 1.2, Revision 116),* 2011.

[12] Trusted Computing Group (TCG), *TCG PC Client Platform - TPM Profile (PTP) Specification,* 2017.

[13] Trusted Computing Group (TCG), *TPM 2.0 Mobile Reference Architecture,* 2014.

[14] Trusted Computing Group (TCG), *TCG TPM 2.0 Automotive Thin Profile,* 2018.

[15] W. Arthur, D. Challener and K. Goldman, A Practical Guide to TPM 2.0 - Using the Trusted Platform Module in the New Age of Security, Apress Media, 2015.

[16] G. Proudler, C. Liqun and C. Dalton, Trusted Computing Platforms - TPM 2.0 in Context, Springer, 2014.

[17] A. Segall, Trusted Platform Modules - Why, when and how to use them, The Institution of Engineering and Technology, 2017.

[18] Trusted Computing Group (TCG), *TCG TSS 2.0 System Level API (SAPI) Specification,* 2018.

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

[20] STMicroelectronics, *ST33TPM12I2C: Trusted Platform Module with I2C interface based on 32-bit ARM® SecurCore® SC300™ CPU,* 2016.

[21] Infineon, *Infineon Chip Card & Security ICs Portfolio,* Munich, Germany, 2017.

[22] Arm Limited, "AMBA 5," 2013. [Online]. Available: https://developer.arm.com/products/architecture/system-architectures/amba/amba-5.

[23] International Business Machines (IBM), "IBM Announces Open On-Chip Bus Architecture," International Business Machines, June 1999. [Online]. Available: https://www-03.ibm.com/press/us/en/pressrelease/2140.wss.

[24] W. D. Schwaderer, Introduction to Open Core Protocol, New York: Springer-Verlag, 2012.

[25] OpenCores, "Wishbone B4: WISHBONE System-on-Chip (SoC) Interconnection Architecture for Portable IP Cores," OpenCores, 2010.

[26] Trusted Computing Group (TCG), *TCG TPM I2C Interface Specification,* 2016.

[27] M. Sabt, M. Achemlal and A. Bouabdallah, "Trusted Execution Environment: What It is, and What It is Not," in *2015 IEEE Trustcom/BigDataSE/ISPA*, Helsinki, Finland, 2015.

[28] Intel Corporation, "Intel Digital Random Number Generator (DRNG): Software Implementation Guide," 2012.

[29] Arm Limited, "ARM TrustZone True Random Number Generator," Cambridge, England, 2017.

[30] L. Torvald, *Linux Kernel drivers/char/random.c comment documentation @ 1da177e4,* 2014.

[31] E. Rescorla, "An Introduction to OpenSSL Programming," Linux Journal, 2001.

[32] Intel Corporation, "Securing the Enterprise with Intel® AES-NI," 2010.

[33] J. Guilbon, "Introduction to Trusted Execution Environment: ARM's TrustZone," 19 June 2018. [Online]. Available: https://blog.quarkslab.com/introduction-to-trusted-execution-environment-arms-trustzone.html. [Accessed 2018].

[34] S. Berger, R. Caceres, K. Goldman, R. Perez, R. Sailer and L. van Doorn, "vTPM: Virtualizing the Trusted Platform Module," in *Security '06: 15th USENIX Security Symposium*, Vancouver, Canada, 2006.

[35] Trusted Computing Group (TCG), *TCG TSS 2.0 Overview and Common Structures Specification,* 2018.

[36] Trusted Computing Group (TCG), *TCG Software Stack Feature API,* 2014.

[37] Trusted Computing Group (TCG), *TCG TSS 2.0 Enhanced System API (ESAPI) Specification,* 2018.

[38] Trusted Computing Group (TCG), *TCG TSS 2.0 TAB and Resource Manager Specification,* 2018.

[39] “TPM Failure Tries, Recovery Time and Lockout Recovery,” Dell, 01 April 2018. [Online]. Available: https://www.dell.com/support/article/gr/el/grbsdt1/sln304487/tpm-failure-tries-recovery-time-and-lockout-recovery. [Accessed 18 September 2018].

[40] “Manage TPM lockout,” Microsoft, 01 May 2017. [Online]. Available: https://docs.microsoft.com/en-us/windows/security/information-protection/tpm/manage-tpm-lockout. [Accessed 18 September 2018].

[41] Y. Po-Hung and Y. Sung-Ming, “Memory attestation of wireless sensor nodes,” *IET Information Security,* vol. 11, no. 6, pp. 338-344, 2016.

[42] W. Wang and D. F. Yu, “Automated Proof for Authorization Protocols of TPM 2.0 in Computational Model,” in *Information Security Practice and Experience*, Springer International Publishing, 2014, pp. 144-158.

[43] J. Shao, Y. Qin, D. Feng and W. Wang, “Formal Analysis of Enhanced Authorization in the TPM 2.0,” in *ACM*, New York, NY, USA, 2015.

[44] Microsoft Corporation, “BitLocker overview,” [Online]. Available: https://docs.microsoft.com/en-us/windows/security/information-protection/bitlocker/bitlocker-overview.

[45] Trusted Computing Group (TCG), *TCG Algorithm Registry (Family 2.0, Revision 01.27),* 2018.

[46] Federal Office for Information Security, “Certification Report BSI-DSZ-CC-1058-2018 for Trusted Plattform Module SLB9670_2.0,” Bonn, 2018.

[47] National Institute of Standards and Technology (NIST), *FIPS PUB 140-2 - Security Requirements for Cryptographic Modules,* 2001.

[48] Atmel Corporation, *AT97SC3205T I2C Interface - Summary Datasheet,* 2014.

[49] E. F. Brickell, J. Camenisch and L. Chen, “Direct anonymous attestation,” in *ACM Conference on Computer and Communications Security (CCS)*, 2004.

[50] J. Whitefield, L. Chen, T. Giannetsos, S. Schneider and H. Treharne, “Privacy-Enhanced Capabilities for VANETs using Direct Anonymous Attestation,” in *IEEE Vehicular Networking Conference (VNC)*, 2017.

[51] E. Brickell, L. Chen and J. Li, “Simplified security notions of direct anonymous attestation and a concrete scheme from pairings,” *International Journal of Information Security,* 2009.

[52] J. Camenisch, L. Chen, M. Drijvers, A. Lehmann, D. Novick and R. Urian, “One TPM to Bind Them All: Fixing TPM 2.0 for Provably Secure Anonymous Attestation,” in *IEEE Symposium on Security and Privacy (S&P)*, 2017.

[53] J. Camenisch, M. Drijvers and A. Lehmann, ““Anonymous Attestation with Subverted TPMs,” in *Advances in Cryptology - CRYPTO*, 2017.

[54] ISO/IEC, “ISO/IEC 11889:2015, Parts 1-4.,” Retrieved Nov.1, 2017 from http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=66510, 2015.

[55] S. Goldwasser, S. Micali and C. Rackoff, “The knowledge complexity of interactive proof systems,” in *SIAM Journal on computing*, 1989.

[56] K. E. Defrawy, G. Holland and G. Tsudik, “Remote Attestation of Heterogeneous Cyber-Physical Systems: The Automotive Use Case,” in *ESCAR*, 2015.

[57] R. Sailer, X. Zhang, T. Jaeger and L. v. Doorn, “Design and implementation of a TCG-based Integrity Measurement Architecture,” in *13th USENIX Symposium*, 2004.

[58] N. Asokan, F. Brasser, A. Ibrahim, A. Sadeghi, M. Schunter, G. Tsudik and C. Waschmann, “SEDA: Scalable Embedded Device Attestation,” in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, 2015.

[59] European Union Agency for Network and Information Security (ENISA), “Inventory of Risk Management / Risk Assessment Tools,” [Online]. Available: https://www.enisa.europa.eu/topics/threat-risk-management/risk-management/current-risk/risk-management-inventory/rm-ra-tools.

[60] J. Y. Yap and A. Tomlinson, “Threat Model of a Scenario Based on Trusted Platform Module 2.0 Specification,” in *Workshop on Web Applications and Secure Hardware (WASH)*, London, United Kingdom, 2013.

[61] M. Gebai and M. R. Dagenais, “Survey and Analysis of Kernel and Userspace Tracers on Linux: Design, Implementation, and Overhead,” *ACM Computing Surveys (CSUR),* vol. 51, no. 26, 2018.

[62] X. Wu, K. Suo, Y. Zhao and J. Rao, “A Side-channel Attack on HotSpot Heap Management,” in *USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 18)*, Boston, MA, 2018.

[63] S. Han, W. Shin, J.-H. Park and H. Kim, “A Bad Dream: Subverting Trusted Platform Module While You Are Sleeping,” in *USENIX Security Symposium*, Baltimore, MD, 2018.

[64] NIST National Vulnerability Database (NVD), “CVE-2017-15361,” 10 October 2017. [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2017-15361.

[65] H. Shahriar and M. Zulkernine, “Mitigating Program Security Vulnerabilities: Approaches and Challenges,” *ACM Comput. Surv.,* vol. 44, no. 3, pp. 11:1-11:46, 2012.

[66] I. Welch and R. J. Stroud, “Using Reflection as a Mechanism for Enforcing Security Policies in Mobile Code,” in *European Symposium on Research in Computer Security (ESORICS)*, Toulouse, France, 2000.

[67] S. M. Ghaffarian and H. R. Shahriari, “Software Vulnerability Analysis and Discovery Using Machine-Learning and Data-Mining Techniques: A Survey,” *ACM Comput. Surv.,* vol. 50, no. 4, pp. 56:1-56:36, 2017.

[68] M. Jacobs Jr., “The quantification and aggregation of model risk: perspectives on potential approaches,” *International Journal of Financial Engineering and Risk Management ,* vol. 2, no. 2, pp. 124-154, 2015.

[69] National Institute of Standards and Technology (NIST), “NISTIR 8105 - Report on Post-Quantum Cryptography,” 2016 .

[70] J. Jonsson and B. Kaliski, *RFC 3447 - Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1,* Internet Engineering Task Force (IETF), 2003.

[71] S.-L. Gazdag, D. Butin and J. Buchmann, “Let Live and Let Die: Handling the State of Hash-based Signatures,” in *Workshop on Cybersecurity in a Post-Quantum World*, Gaithersburg, Maryland, 2015.

[72] AlephOne, “Smashing the Stack for Fun and Profit,” *Phrack Magazine,* vol. 49, no. 14, 1996.

[73] H. Shacham, “The Geometry of Innocent Flesh on the Bone: Return-into-libc Without Function Calls (on the x86),” in *ACM Conference on Computer and Communications Security*, 2007.

[74] S. Checkoway, L. Davi, A. Dmitrienko, A.-R. Sadeghi, H. Shacham and M. Winandy, “Return-oriented Programming Without Returns,” in *ACM Conference on Computer and Communications Security*, 2010.

[75] IOVisor, “BCC - Tools for BPF-based Linux IO analysis, networking, monitoring, and more,” 2018. [Online]. Available: https://github.com/iovisor/bcc#tools. [Accessed 27 September 2018].

[76] R. Yavatkar, D. Pendarakis and R. Guerin, “RFC 2753 - A Framework for Policy-based Admission Control,” 2000.

[77] The JBoss Drools team, “Drools Expert User Guide v5.4.0.Final,” 13 May 2012. [Online]. Available: https://docs.jboss.org/drools/release/5.4.0.Final/drools-expert-docs/pdf/drools-expert-docs.pdf.

[78] “Jenkins,” [Online]. Available: https://jenkins.io/.

[79] “SonarQube,” [Online]. Available: https://www.sonarqube.org/.

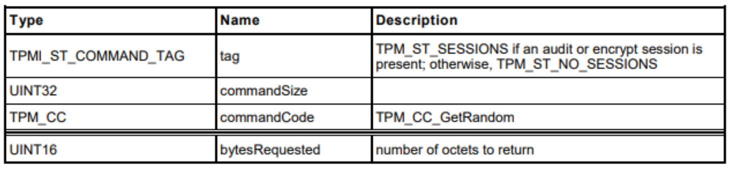
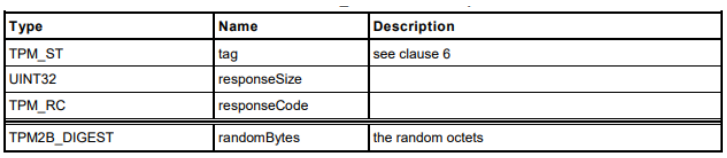
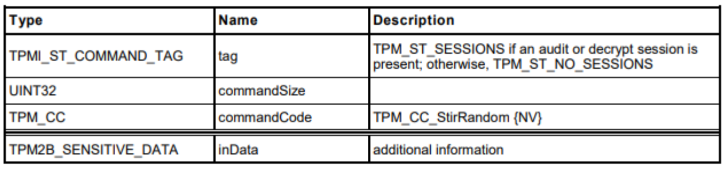
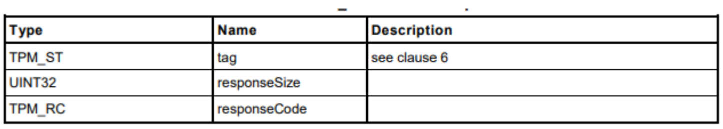
[80] “Sonartype Nexus,” [Online]. Available: https://www.sonatype.com/nexus-repository-sonatype.

[81] P. McDaniel and A. Prakash, “Security Policy Enforcement in the Antigone System,” 2005.

[82] Trusted Computing Group (TCG), *Trusted Platform Module 2.0: A Brief Introduction,* 2018.

Appendix A

Table 21: FutureTPM QR TPM-related command updates.

Command | Description | Parameters | Response	Changes Needed
TPM2_GetRandom	
This command returns the next bytesRequested octets from the random number generator (RNG).	
 	**NO** *If bytesRequested is more than will fit into a TPM2B_DIGEST on the TPM, no error is returned but the TPM will only return as much data as will fit into a TPM2B_DIGEST buffer for the TPM.* *The maximum amount of data returned by this command is TPM implementation-dependent*
TPM2_StirRandom	
This command is used to add "additional information" to the RNG state.	
 	**NO** *The inData parameter may not be larger than 128 octets.*

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS if an audit or encrypt session is present; otherwise, TPM_ST_NO_SESSIONS
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_GetRandom
UINT16	bytesRequested	number of octets to return

Type	Name	Description
TPM_ST	tag	see clause 6
UINT32	responseSize	
TPM_RC	responseCode	
TPM2B_DIGEST	randomBytes	the random octets

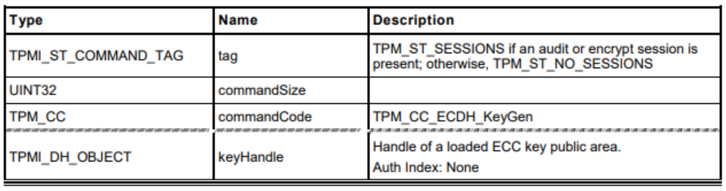
Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS if an audit or decrypt session is present; otherwise, TPM_ST_NO_SESSIONS
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_StirRandom {NV}
TPM2B_SENSITIVE_DATA	inData	additional information

Type	Name	Description
TPM_ST	tag	see clause 6
UINT32	responseSize	
TPM_RC	responseCode	

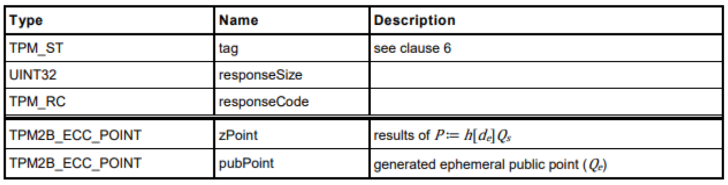
Command | Description | Parameters | Response	Changes Needed

TPM2_ECDH_KeyGen

This command uses the TPM to generate an ephemeral key pair.



Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS if an audit or encrypt session is present; otherwise, TPM_ST_NO_SESSIONS
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_ECDH_KeyGen
TPMI_DH_OBJECT	keyHandle	Handle of a loaded ECC key public area. Auth Index: None



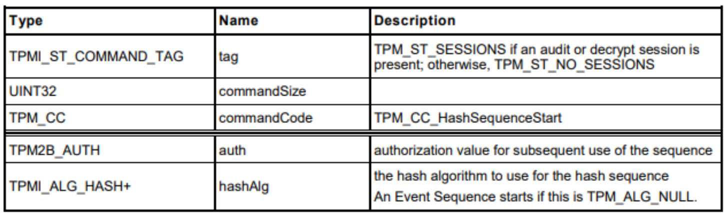
Type	Name	Description
TPM_ST	tag	see clause 6
UINT32	responseSize	
TPM_RC	responseCode	
TPM2B_ECC_POINT	zPoint	results of $P := h[d_e]Q_s$
TPM2B_ECC_POINT	pubPoint	generated ephemeral public point (Q_e)

Changes Needed: **YES**

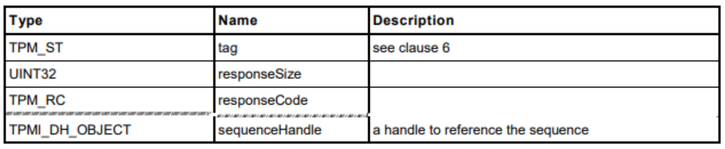
DH is broken need to be updated with another algorithm

TPM2_HashSequenceStart

This command starts a hash or an Event Sequence.



Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS if an audit or decrypt session is present; otherwise, TPM_ST_NO_SESSIONS
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_HashSequenceStart
TPM2B_AUTH	auth	authorization value for subsequent use of the sequence
TPMI_ALG_HASH+	hashAlg	the hash algorithm to use for the hash sequence An Event Sequence starts if this is TPM_ALG_NULL.



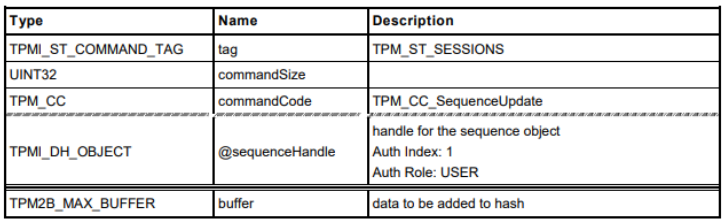
Type	Name	Description
TPM_ST	tag	see clause 6
UINT32	responseSize	
TPM_RC	responseCode	
TPMI_DH_OBJECT	sequenceHandle	a handle to reference the sequence

Changes Needed: **NO**

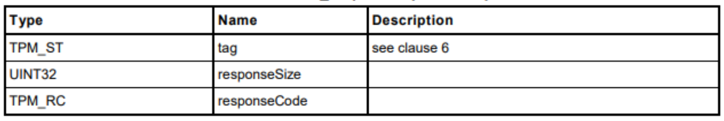
Command | Description | Parameters | Response	Changes Needed
TPM2_SequenceUpdate	
This command is used to add data to a hash or HMAC sequence. The amount of data in buffer may be any size up to the limits of the TPM.	
(command and response tables below)	**NO**
TPM2_SequenceComplete	
This command adds the last part of data, if any, to a hash/HMAC sequence and returns the result.	
(command and response tables below)	**NO**

TPM2_SequenceUpdate

This command is used to add data to a hash or HMAC sequence. The amount of data in buffer may be any size up to the limits of the TPM.



Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_SequenceUpdate
TPMI_DH_OBJECT	@sequenceHandle	handle for the sequence object Auth Index: 1 Auth Role: USER
TPM2B_MAX_BUFFER	buffer	data to be added to hash

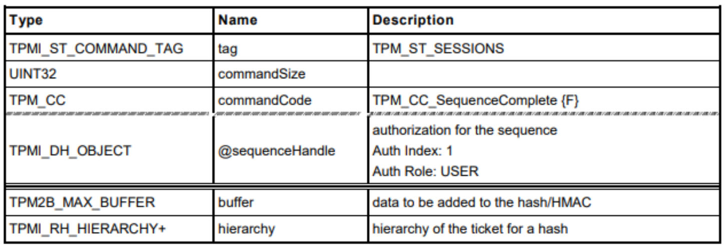


Type	Name	Description
TPM_ST	tag	see clause 6
UINT32	responseSize	
TPM_RC	responseCode	

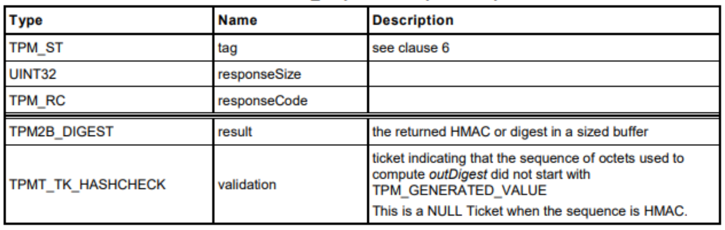
NO

TPM2_SequenceComplete

This command adds the last part of data, if any, to a hash/HMAC sequence and returns the result.



Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_SequenceComplete {F}
TPMI_DH_OBJECT	@sequenceHandle	authorization for the sequence Auth Index: 1 Auth Role: USER
TPM2B_MAX_BUFFER	buffer	data to be added to the hash/HMAC
TPMI_RH_HIERARCHY+	hierarchy	hierarchy of the ticket for a hash



Type	Name	Description
TPM_ST	tag	see clause 6
UINT32	responseSize	
TPM_RC	responseCode	
TPM2B_DIGEST	result	the returned HMAC or digest in a sized buffer
TPMT_TK_HASHCHECK	validation	ticket indicating that the sequence of octets used to compute *outDigest* did not start with TPM_GENERATED_VALUE This is a NULL Ticket when the sequence is HMAC.

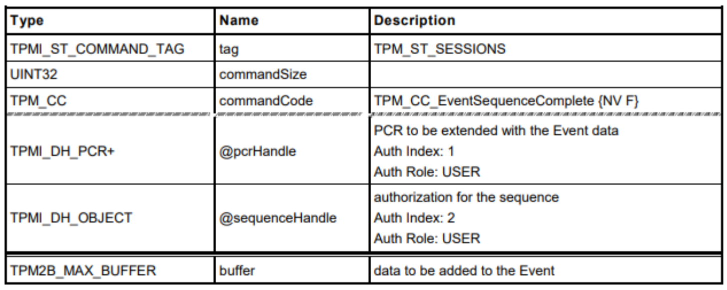
NO

Command | Description | Parameters | Response	Changes Needed

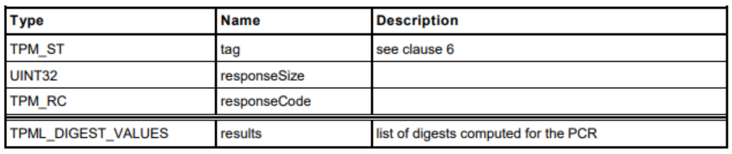
TPM2_EventSequenceComplete

This command adds the last part of data, if any, to an Event Sequence and returns the result in a digest list.

Changes Needed: **NO**



Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_EventSequenceComplete {NV F}
TPMI_DH_PCR+	@pcrHandle	PCR to be extended with the Event data Auth Index: 1 Auth Role: USER
TPMI_DH_OBJECT	@sequenceHandle	authorization for the sequence Auth Index: 2 Auth Role: USER
TPM2B_MAX_BUFFER	buffer	data to be added to the Event

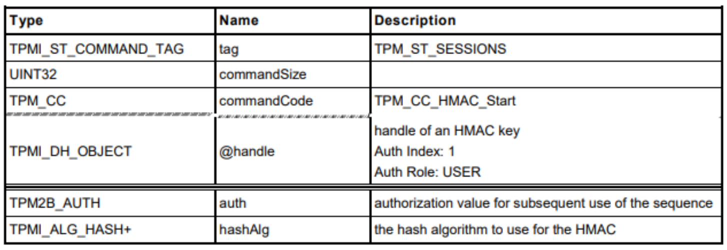


Type	Name	Description
TPM_ST	tag	see clause 6
UINT32	responseSize	
TPM_RC	responseCode	
TPML_DIGEST_VALUES	results	list of digests computed for the PCR

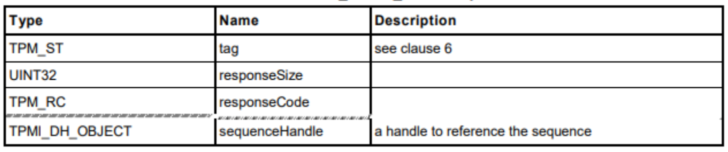
TPM2_HMAC_Start

This command starts an HMAC sequence.

Changes Needed: **NO**



Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_HMAC_Start
TPMI_DH_OBJECT	@handle	handle of an HMAC key Auth Index: 1 Auth Role: USER
TPM2B_AUTH	auth	authorization value for subsequent use of the sequence
TPMI_ALG_HASH+	hashAlg	the hash algorithm to use for the HMAC

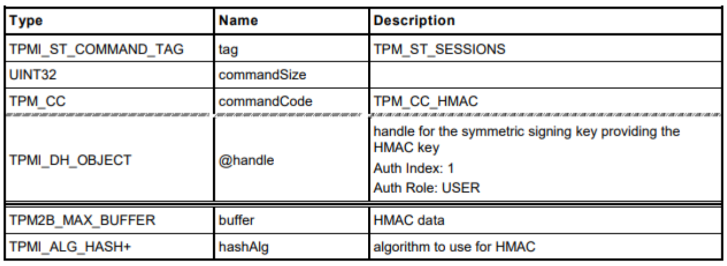


Type	Name	Description
TPM_ST	tag	see clause 6
UINT32	responseSize	
TPM_RC	responseCode	
TPMI_DH_OBJECT	sequenceHandle	a handle to reference the sequence

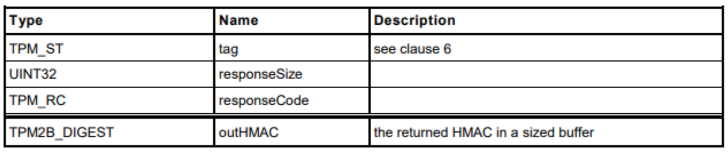
Command | Description | Parameters | Response	Changes Needed

TPM2_HMAC

This command performs an HMAC on the supplied data using the indicated hash algorithm.



Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_HMAC
TPMI_DH_OBJECT	@handle	handle for the symmetric signing key providing the HMAC key Auth Index: 1 Auth Role: USER
TPM2B_MAX_BUFFER	buffer	HMAC data
TPMI_ALG_HASH+	hashAlg	algorithm to use for HMAC

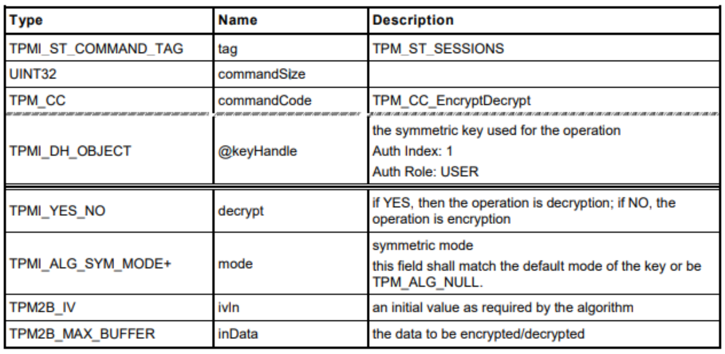


Type	Name	Description
TPM_ST	tag	see clause 6
UINT32	responseSize	
TPM_RC	responseCode	
TPM2B_DIGEST	outHMAC	the returned HMAC in a sized buffer

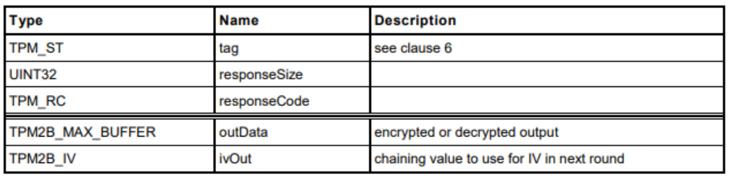
Changes Needed: **NO**

TPM2_EncryptDecrypt

This command performs symmetric encryption or decryption using the symmetric key referenced by keyHandle and the selected mode.



Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_EncryptDecrypt
TPMI_DH_OBJECT	@keyHandle	the symmetric key used for the operation Auth Index: 1 Auth Role: USER
TPMI_YES_NO	decrypt	if YES, then the operation is decryption; if NO, the operation is encryption
TPMI_ALG_SYM_MODE+	mode	symmetric mode this field shall match the default mode of the key or be TPM_ALG_NULL.
TPM2B_IV	ivIn	an initial value as required by the algorithm
TPM2B_MAX_BUFFER	inData	the data to be encrypted/decrypted



Type	Name	Description
TPM_ST	tag	see clause 6
UINT32	responseSize	
TPM_RC	responseCode	
TPM2B_MAX_BUFFER	outData	encrypted or decrypted output
TPM2B_IV	ivOut	chaining value to use for IV in next round

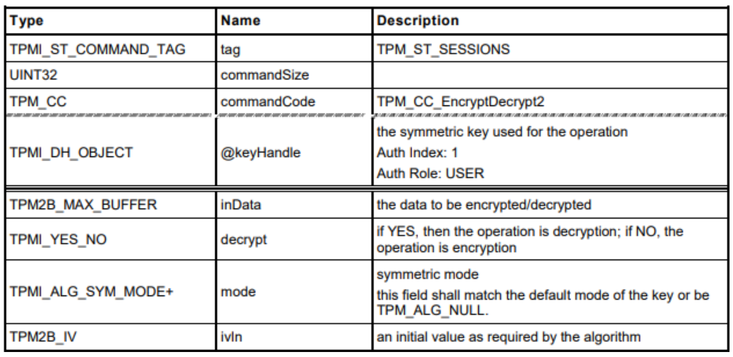
Changes Needed: **NO**

keyHandle input should be of an appropriate length

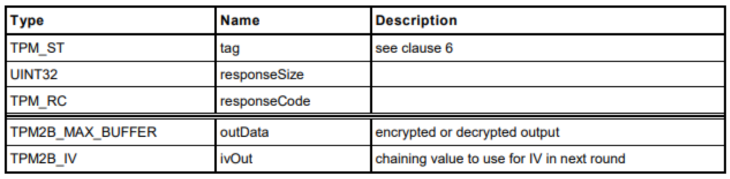
Command | Description | Parameters | Response	Changes Needed

TPM2_EncryptDecrypt2

This command is identical to TPM2_EncryptDecrypt, except that the inData parameter is the first parameter. This permits inData to be parameter encrypted.



Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_EncryptDecrypt2
TPMI_DH_OBJECT	@keyHandle	the symmetric key used for the operation Auth Index: 1 Auth Role: USER
TPM2B_MAX_BUFFER	inData	the data to be encrypted/decrypted
TPMI_YES_NO	decrypt	if YES, then the operation is decryption; if NO, the operation is encryption
TPMI_ALG_SYM_MODE+	mode	symmetric mode this field shall match the default mode of the key or be TPM_ALG_NULL.
TPM2B_IV	ivIn	an initial value as required by the algorithm



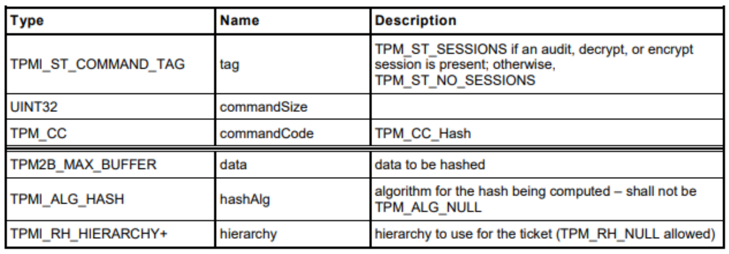
Type	Name	Description
TPM_ST	tag	see clause 6
UINT32	responseSize	
TPM_RC	responseCode	
TPM2B_MAX_BUFFER	outData	encrypted or decrypted output
TPM2B_IV	ivOut	chaining value to use for IV in next round

Changes Needed: **NO**

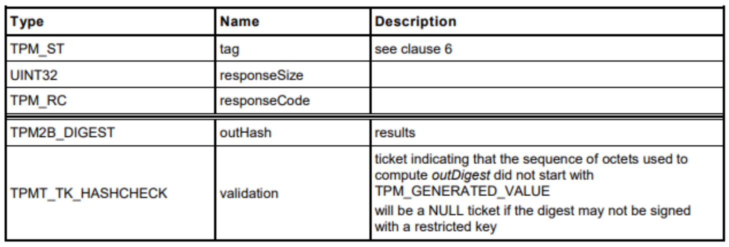
keyHandle input should be of an appropriate length

TPM2_Hash

This command performs a hash operation on a data buffer and returns the results.



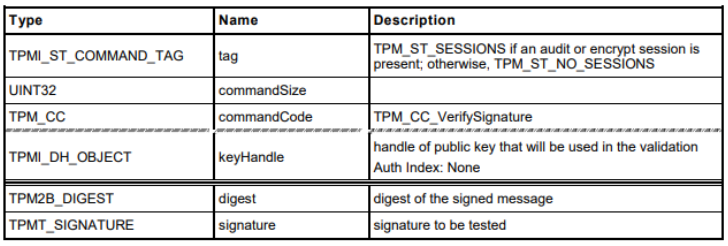
Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS if an audit, decrypt, or encrypt session is present; otherwise, TPM_ST_NO_SESSIONS
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_Hash
TPM2B_MAX_BUFFER	data	data to be hashed
TPMI_ALG_HASH	hashAlg	algorithm for the hash being computed – shall not be TPM_ALG_NULL
TPMI_RH_HIERARCHY+	hierarchy	hierarchy to use for the ticket (TPM_RH_NULL allowed)



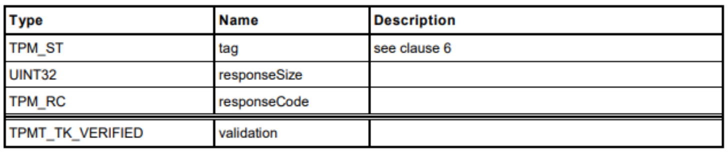
Type	Name	Description
TPM_ST	tag	see clause 6
UINT32	responseSize	
TPM_RC	responseCode	
TPM2B_DIGEST	outHash	results
TPMT_TK_HASHCHECK	validation	ticket indicating that the sequence of octets used to compute *outDigest* did not start with TPM_GENERATED_VALUE will be a NULL ticket if the digest may not be signed with a restricted key

Changes Needed: **NO**

Command | Description | Parameters | Response	Changes Needed
TPM2_VerifySignature	
This command uses loaded keys to validate a signature on a message with the message digest passed to the TPM.	
(command and response tables below)	**NO**

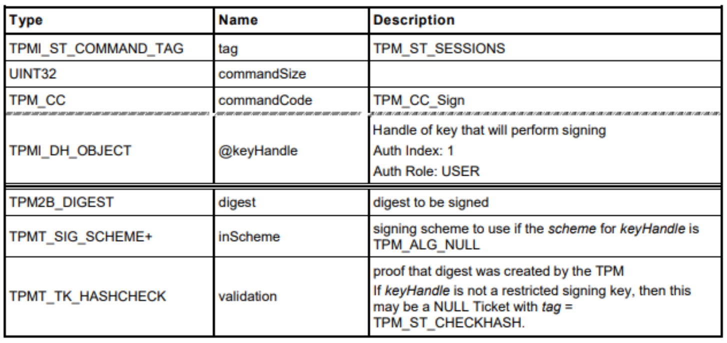


Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS if an audit or encrypt session is present; otherwise, TPM_ST_NO_SESSIONS
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_VerifySignature
TPMI_DH_OBJECT	keyHandle	handle of public key that will be used in the validation Auth Index: None
TPM2B_DIGEST	digest	digest of the signed message
TPMT_SIGNATURE	signature	signature to be tested

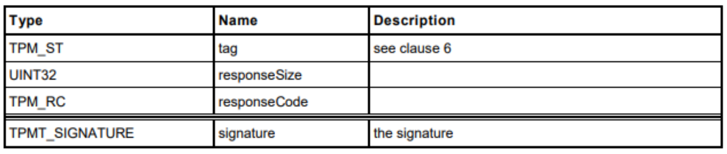


Type	Name	Description
TPM_ST	tag	see clause 6
UINT32	responseSize	
TPM_RC	responseCode	
TPMT_TK_VERIFIED	validation	

Command | Description | Parameters | Response	Changes Needed
TPM2_Sign	
This command causes the TPM to sign an externally provided hash with the specified symmetric or asymmetric signing key.	
(command and response tables below)	**NO**

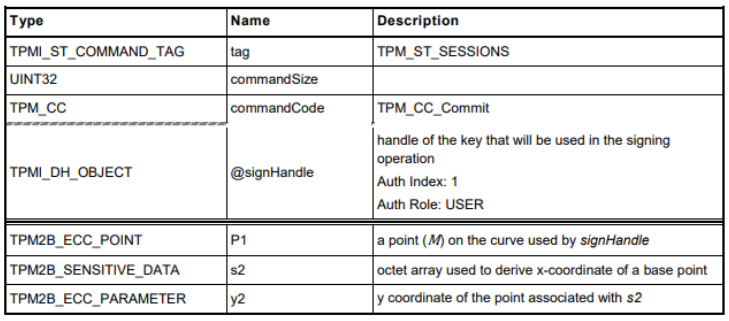


Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_Sign
TPMI_DH_OBJECT	@keyHandle	Handle of key that will perform signing Auth Index: 1 Auth Role: USER
TPM2B_DIGEST	digest	digest to be signed
TPMT_SIG_SCHEME+	inScheme	signing scheme to use if the *scheme* for *keyHandle* is TPM_ALG_NULL
TPMT_TK_HASHCHECK	validation	proof that digest was created by the TPM If *keyHandle* is not a restricted signing key, then this may be a NULL Ticket with *tag* = TPM_ST_CHECKHASH.

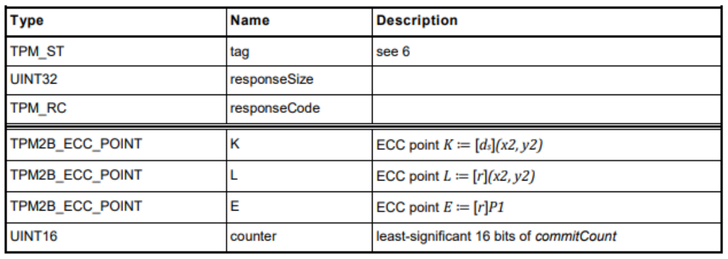


Type	Name	Description
TPM_ST	tag	see clause 6
UINT32	responseSize	
TPM_RC	responseCode	
TPMT_SIGNATURE	signature	the signature

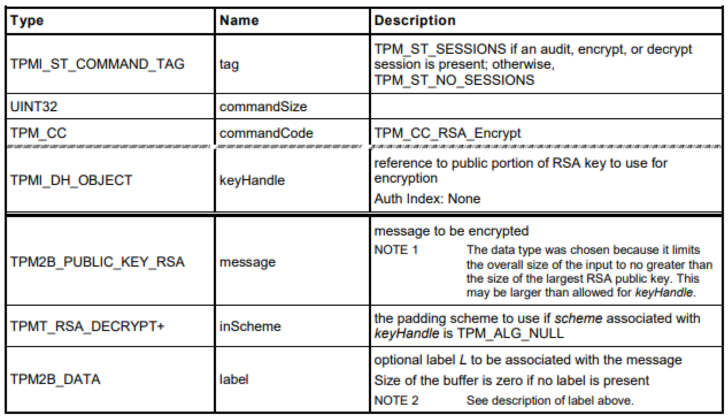
Command | Description | Parameters | Response	Changes Needed
TPM2_Commit	
TPM2_Commit performs the first part of an ECC anonymous signing operation.	
(command and response tables below)	**YES** *ECC is broken need to be updated with another algorithm*
TPM2_RSA_Encrypt	
This command performs RSA encryption using the indicated padding scheme according to IETF RFC 3447 (PKCS#1) [70].	
(command and response tables below)	**YES** *RSA is broken need to be updated with another algorithm*



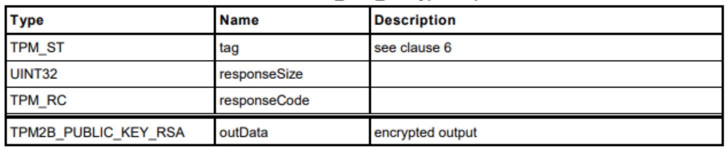
Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_Commit
TPMI_DH_OBJECT	@signHandle	handle of the key that will be used in the signing operation Auth Index: 1 Auth Role: USER
TPM2B_ECC_POINT	P1	a point (*M*) on the curve used by *signHandle*
TPM2B_SENSITIVE_DATA	s2	octet array used to derive x-coordinate of a base point
TPM2B_ECC_PARAMETER	y2	y coordinate of the point associated with *s2*



Type	Name	Description
TPM_ST	tag	see 6
UINT32	responseSize	
TPM_RC	responseCode	
TPM2B_ECC_POINT	K	ECC point $K \coloneqq [d_s](x2, y2)$
TPM2B_ECC_POINT	L	ECC point $L \coloneqq [r](x2, y2)$
TPM2B_ECC_POINT	E	ECC point $E \coloneqq [r]P1$
UINT16	counter	least-significant 16 bits of *commitCount*

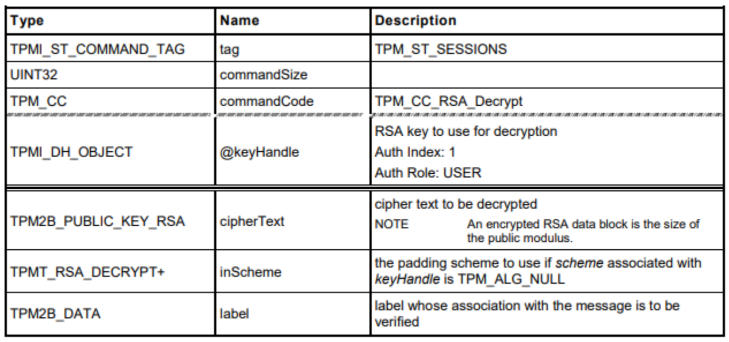


Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS if an audit, encrypt, or decrypt session is present; otherwise, TPM_ST_NO_SESSIONS
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_RSA_Encrypt
TPMI_DH_OBJECT	keyHandle	reference to public portion of RSA key to use for encryption Auth Index: None
TPM2B_PUBLIC_KEY_RSA	message	message to be encrypted NOTE 1 The data type was chosen because it limits the overall size of the input to no greater than the size of the largest RSA public key. This may be larger than allowed for *keyHandle*.
TPMT_RSA_DECRYPT+	inScheme	the padding scheme to use if *scheme* associated with *keyHandle* is TPM_ALG_NULL
TPM2B_DATA	label	optional label *L* to be associated with the message Size of the buffer is zero if no label is present NOTE 2 See description of label above.

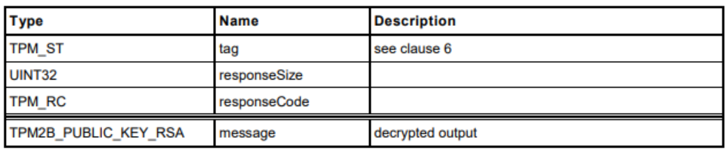


Type	Name	Description
TPM_ST	tag	see clause 6
UINT32	responseSize	
TPM_RC	responseCode	
TPM2B_PUBLIC_KEY_RSA	outData	encrypted output

Command | Description | Parameters | Response	Changes Needed
TPM2_RSA_Decrypt	
This command performs RSA decryption using the indicated padding scheme according to IETF RFC 3447 (PKCS#1) [70].	
(command and response tables below)	**YES** *RSA is broken need to be updated with another algorithm*

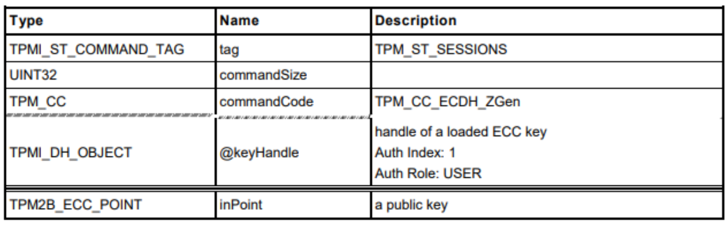


Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_RSA_Decrypt
TPMI_DH_OBJECT	@keyHandle	RSA key to use for decryption Auth Index: 1 Auth Role: USER
TPM2B_PUBLIC_KEY_RSA	cipherText	cipher text to be decrypted NOTE An encrypted RSA data block is the size of the public modulus.
TPMT_RSA_DECRYPT+	inScheme	the padding scheme to use if *scheme* associated with *keyHandle* is TPM_ALG_NULL
TPM2B_DATA	label	label whose association with the message is to be verified

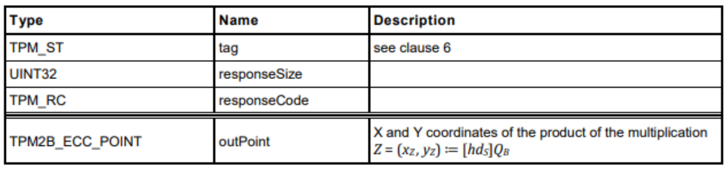


Type	Name	Description
TPM_ST	tag	see clause 6
UINT32	responseSize	
TPM_RC	responseCode	
TPM2B_PUBLIC_KEY_RSA	message	decrypted output

Command | Description | Parameters | Response	Changes Needed
TPM2_ECDH_ZGen	
This command uses the TPM to recover the Z value from a public point and a private key.	
(command and response tables below)	**YES** *EDCH and ECC are broken need to be updated with other algorithms*



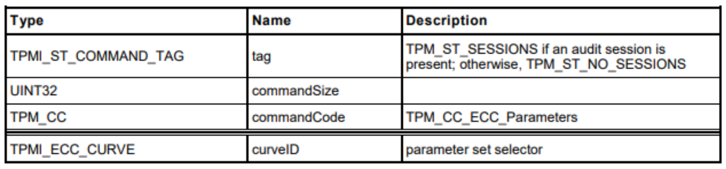
Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_ECDH_ZGen
TPMI_DH_OBJECT	@keyHandle	handle of a loaded ECC key Auth Index: 1 Auth Role: USER
TPM2B_ECC_POINT	inPoint	a public key



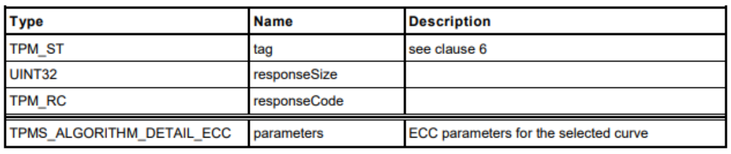
Type	Name	Description
TPM_ST	tag	see clause 6
UINT32	responseSize	
TPM_RC	responseCode	
TPM2B_ECC_POINT	outPoint	X and Y coordinates of the product of the multiplication $Z = (x_Z, y_Z) := [hd_S]Q_B$

Command | Description | Parameters | Response	Changes Needed
TPM2_ECC_Parameters	
This command returns the parameters of an ECC curve identified by its TCG-assigned curveID.	
(see command and response tables below)	**YES** *ECC is broken need to be updated with another algorithm*
TPM2_EC_Ephemeral	
TPM2_EC_Ephemeral creates an ephemeral key for use in a two-phase key exchange protocol.	
(see command and response tables below)	**YES** *ECDH is broken need to be updated with another algorithm*

TPM2_ECC_Parameters



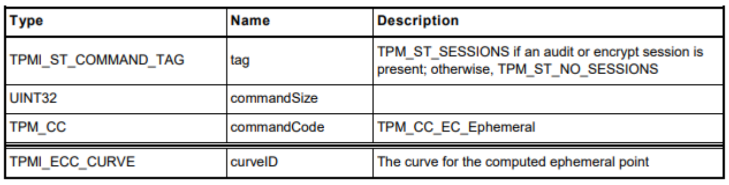
Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS if an audit session is present; otherwise, TPM_ST_NO_SESSIONS
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_ECC_Parameters
TPMI_ECC_CURVE	curveID	parameter set selector



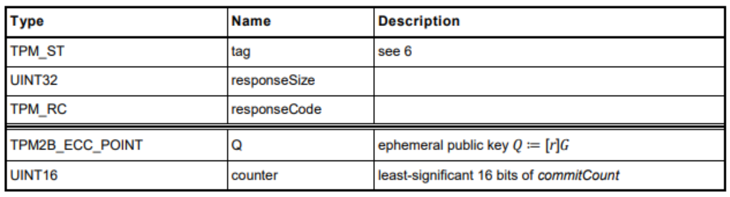
Type	Name	Description
TPM_ST	tag	see clause 6
UINT32	responseSize	
TPM_RC	responseCode	
TPMS_ALGORITHM_DETAIL_ECC	parameters	ECC parameters for the selected curve

Changes Needed: **YES** — *ECC is broken need to be updated with another algorithm*

TPM2_EC_Ephemeral



Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS if an audit or encrypt session is present; otherwise, TPM_ST_NO_SESSIONS
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_EC_Ephemeral
TPMI_ECC_CURVE	curveID	The curve for the computed ephemeral point



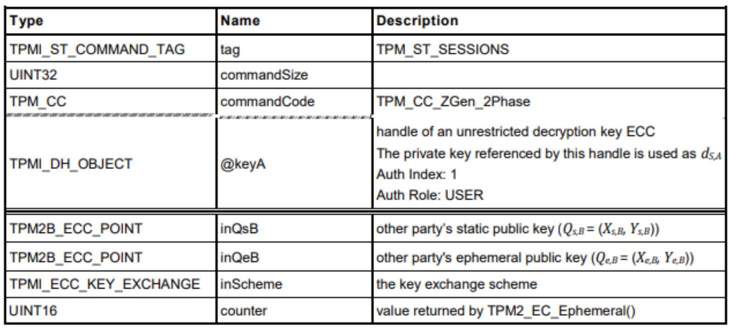
Type	Name	Description
TPM_ST	tag	see 6
UINT32	responseSize	
TPM_RC	responseCode	
TPM2B_ECC_POINT	Q	ephemeral public key $Q ≔ [r]G$
UINT16	counter	least-significant 16 bits of *commitCount*

Changes Needed: **YES** — *ECDH is broken need to be updated with another algorithm*

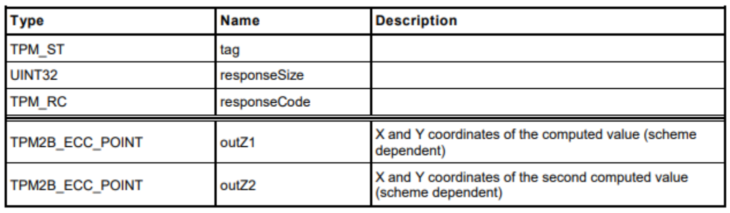
Command | Description | Parameters | Response	Changes Needed

TPM2_ZGen_2Phase

This command supports two-phase key exchange protocols. The command is used in combination with TPM2_EC_Ephemeral. TPM2_EC_Ephemeral generates an ephemeral key and returns the public point of that ephemeral key along with a numeric value that allows the TPM to regenerate the associated private key.



Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_ZGen_2Phase
TPMI_DH_OBJECT	@keyA	handle of an unrestricted decryption key ECC The private key referenced by this handle is used as $d_{S,A}$ Auth Index: 1 Auth Role: USER
TPM2B_ECC_POINT	inQsB	other party's static public key ($Q_{s,B} = (X_{s,B}, Y_{s,B})$)
TPM2B_ECC_POINT	inQeB	other party's ephemeral public key ($Q_{e,B} = (X_{e,B}, Y_{e,B})$)
TPMI_ECC_KEY_EXCHANGE	inScheme	the key exchange scheme
UINT16	counter	value returned by TPM2_EC_Ephemeral()



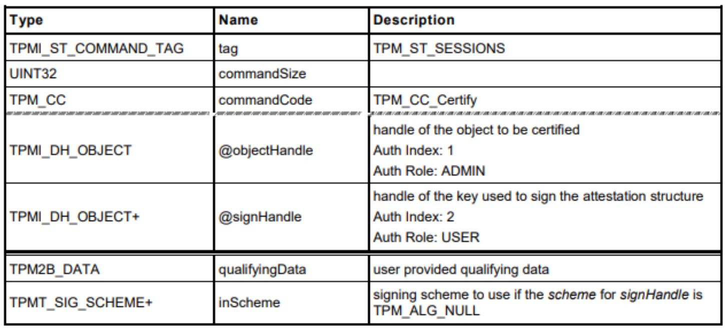
Type	Name	Description
TPM_ST	tag	
UINT32	responseSize	
TPM_RC	responseCode	
TPM2B_ECC_POINT	outZ1	X and Y coordinates of the computed value (scheme dependent)
TPM2B_ECC_POINT	outZ2	X and Y coordinates of the second computed value (scheme dependent)

Changes Needed: **YES**

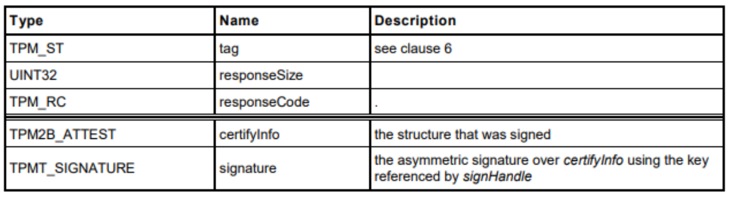
EDCH and ECC are broken need to be updated with other algorithms

TPM2_Certify

The purpose of this command is to prove that an object with a specific Name is loaded in the TPM.



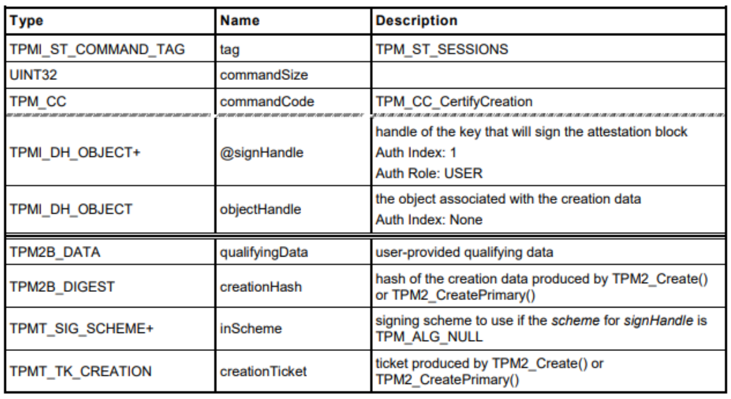
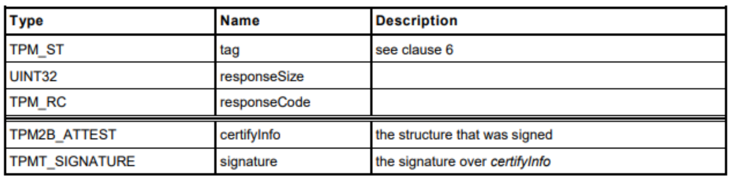
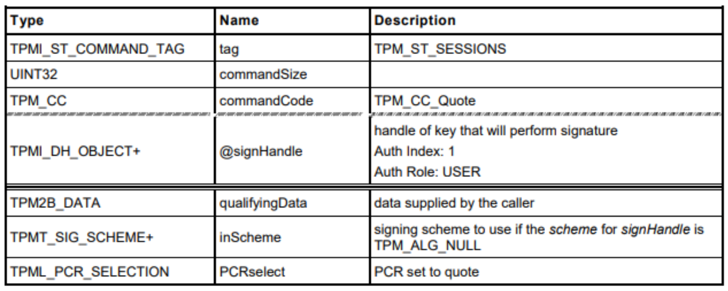
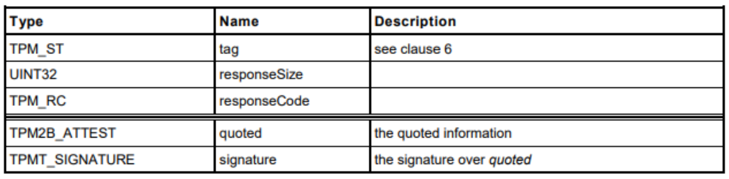
Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_Certify
TPMI_DH_OBJECT	@objectHandle	handle of the object to be certified Auth Index: 1 Auth Role: ADMIN
TPMI_DH_OBJECT+	@signHandle	handle of the key used to sign the attestation structure Auth Index: 2 Auth Role: USER
TPM2B_DATA	qualifyingData	user provided qualifying data
TPMT_SIG_SCHEME+	inScheme	signing scheme to use if the *scheme* for *signHandle* is TPM_ALG_NULL



Type	Name	Description
TPM_ST	tag	see clause 6
UINT32	responseSize	
TPM_RC	responseCode	.
TPM2B_ATTEST	certifyInfo	the structure that was signed
TPMT_SIGNATURE	signature	the asymmetric signature over *certifyInfo* using the key referenced by *signHandle*

Changes Needed: **YES**

ECC is broken need to be updated with another algorithm

Command | Description | Parameters | Response	Changes Needed
TPM2_CertifyCreation	
This command is used to prove the association between an object and its creation data.	
 	**YES** *ECC is broken need to be updated with another algorithm*
TPM2_Quote	
This command is used to quote PCR values.	
 	**NO**

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_CertifyCreation
TPMI_DH_OBJECT+	@signHandle	handle of the key that will sign the attestation block Auth Index: 1 Auth Role: USER
TPMI_DH_OBJECT	objectHandle	the object associated with the creation data Auth Index: None
TPM2B_DATA	qualifyingData	user-provided qualifying data
TPM2B_DIGEST	creationHash	hash of the creation data produced by TPM2_Create() or TPM2_CreatePrimary()
TPMT_SIG_SCHEME+	inScheme	signing scheme to use if the *scheme* for *signHandle* is TPM_ALG_NULL
TPMT_TK_CREATION	creationTicket	ticket produced by TPM2_Create() or TPM2_CreatePrimary()

Type	Name	Description
TPM_ST	tag	see clause 6
UINT32	responseSize	
TPM_RC	responseCode	
TPM2B_ATTEST	certifyInfo	the structure that was signed
TPMT_SIGNATURE	signature	the signature over *certifyInfo*

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_Quote
TPMI_DH_OBJECT+	@signHandle	handle of key that will perform signature Auth Index: 1 Auth Role: USER
TPM2B_DATA	qualifyingData	data supplied by the caller
TPMT_SIG_SCHEME+	inScheme	signing scheme to use if the *scheme* for *signHandle* is TPM_ALG_NULL
TPML_PCR_SELECTION	PCRselect	PCR set to quote

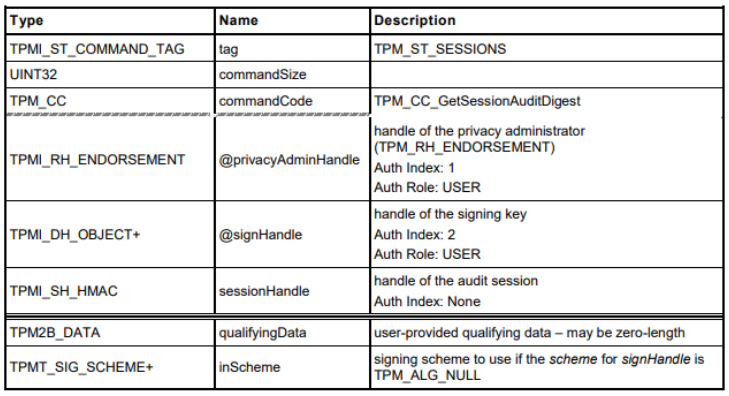
Type	Name	Description
TPM_ST	tag	see clause 6
UINT32	responseSize	
TPM_RC	responseCode	
TPM2B_ATTEST	quoted	the quoted information
TPMT_SIGNATURE	signature	the signature over *quoted*

Command | Description | Parameters | Response	Changes Needed

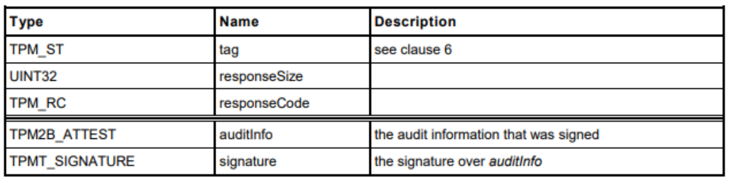
TPM2_GetSessionAuditDigest

This command returns a digital signature of the audit session digest.

Changes Needed: NO



Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_GetSessionAuditDigest
TPMI_RH_ENDORSEMENT	@privacyAdminHandle	handle of the privacy administrator (TPM_RH_ENDORSEMENT) Auth Index: 1 Auth Role: USER
TPMI_DH_OBJECT+	@signHandle	handle of the signing key Auth Index: 2 Auth Role: USER
TPMI_SH_HMAC	sessionHandle	handle of the audit session Auth Index: None
TPM2B_DATA	qualifyingData	user-provided qualifying data – may be zero-length
TPMT_SIG_SCHEME+	inScheme	signing scheme to use if the *scheme* for *signHandle* is TPM_ALG_NULL

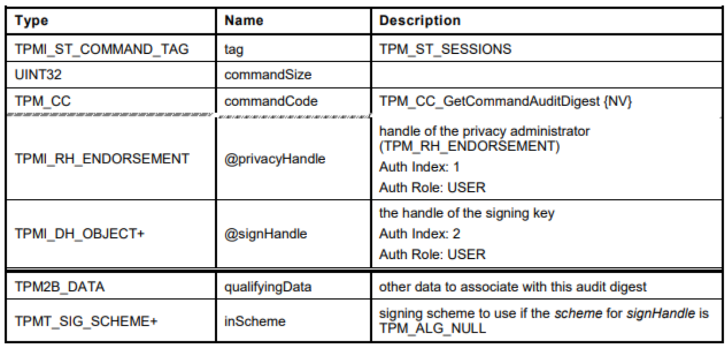


Type	Name	Description
TPM_ST	tag	see clause 6
UINT32	responseSize	
TPM_RC	responseCode	
TPM2B_ATTEST	auditInfo	the audit information that was signed
TPMT_SIGNATURE	signature	the signature over *auditInfo*

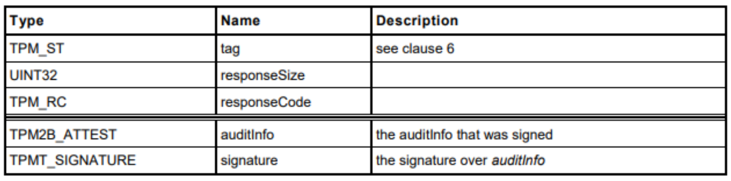
TPM2_GetCommandAuditDigest

This command returns the current value of the command audit digest, a digest of the commands being audited, and the audit hash algorithm.

Changes Needed: NO



Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_GetCommandAuditDigest {NV}
TPMI_RH_ENDORSEMENT	@privacyHandle	handle of the privacy administrator (TPM_RH_ENDORSEMENT) Auth Index: 1 Auth Role: USER
TPMI_DH_OBJECT+	@signHandle	the handle of the signing key Auth Index: 2 Auth Role: USER
TPM2B_DATA	qualifyingData	other data to associate with this audit digest
TPMT_SIG_SCHEME+	inScheme	signing scheme to use if the *scheme* for *signHandle* is TPM_ALG_NULL



Type	Name	Description
TPM_ST	tag	see clause 6
UINT32	responseSize	
TPM_RC	responseCode	
TPM2B_ATTEST	auditInfo	the auditInfo that was signed
TPMT_SIGNATURE	signature	the signature over *auditInfo*

Command | Description | Parameters | Response	Changes Needed
TPM2_GetTime	
This command returns the current values of Time and Clock	
(command table below)	**NO**
(response table below)	
TPM2_Create	
This command is used to create an object that can be loaded into a TPM using TPM2_Load.	
(command table below)	**NO**
(response table below)	

TPM2_GetTime

This command returns the current values of Time and Clock

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_GetTime
TPMI_RH_ENDORSEMENT	@privacyAdminHandle	handle of the privacy administrator (TPM_RH_ENDORSEMENT) Auth Index: 1 Auth Role: USER
TPMI_DH_OBJECT+	@signHandle	the *keyHandle* identifier of a loaded key that can perform digital signatures Auth Index: 2 Auth Role: USER
TPM2B_DATA	qualifyingData	data to tick stamp
TPMT_SIG_SCHEME+	inScheme	signing scheme to use if the *scheme* for *signHandle* is TPM_ALG_NULL

Type	Name	Description
TPM_ST	tag	see clause 6
UINT32	responseSize	
TPM_RC	responseCode	.
TPM2B_ATTEST	timeInfo	standard TPM-generated attestation block
TPMT_SIGNATURE	signature	the signature over *timeInfo*

Changes Needed: **NO**

TPM2_Create

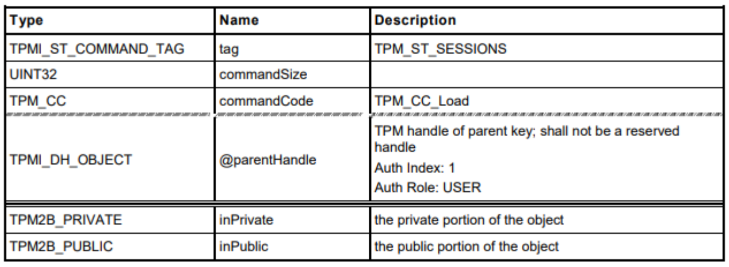
This command is used to create an object that can be loaded into a TPM using TPM2_Load.

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_Create
TPMI_DH_OBJECT	@parentHandle	handle of parent for new object Auth Index: 1 Auth Role: USER
TPM2B_SENSITIVE_CREATE	inSensitive	the sensitive data
TPM2B_PUBLIC	inPublic	the public template
TPM2B_DATA	outsideInfo	data that will be included in the creation data for this object to provide permanent, verifiable linkage between this object and some object owner data
TPML_PCR_SELECTION	creationPCR	PCR that will be used in creation data

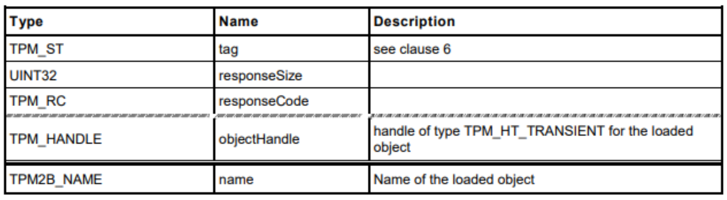
Type	Name	Description
TPM_ST	tag	see clause 6
UINT32	responseSize	
TPM_RC	responseCode	
TPM2B_PRIVATE	outPrivate	the private portion of the object
TPM2B_PUBLIC	outPublic	the public portion of the created object
TPM2B_CREATION_DATA	creationData	contains a TPMS_CREATION_DATA
TPM2B_DIGEST	creationHash	digest of *creationData* using *nameAlg* of *outPublic*
TPMT_TK_CREATION	creationTicket	ticket used by TPM2_CertifyCreation() to validate that the creation data was produced by the TPM

Changes Needed: **NO**

Command | Description | Parameters | Response	Changes Needed
TPM2_Load	
This command is used to load objects into the TPM.	
(see tables below)	**NO**

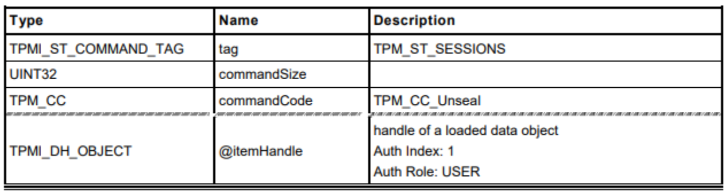


Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_Load
TPMI_DH_OBJECT	@parentHandle	TPM handle of parent key; shall not be a reserved handle Auth Index: 1 Auth Role: USER
TPM2B_PRIVATE	inPrivate	the private portion of the object
TPM2B_PUBLIC	inPublic	the public portion of the object

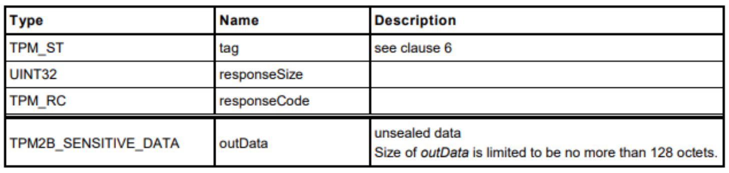


Type	Name	Description
TPM_ST	tag	see clause 6
UINT32	responseSize	
TPM_RC	responseCode	
TPM_HANDLE	objectHandle	handle of type TPM_HT_TRANSIENT for the loaded object
TPM2B_NAME	name	Name of the loaded object

Command | Description | Parameters | Response	Changes Needed
TPM2_Unseal	
This command returns the data in a loaded Sealed Data Object	
(see tables below)	**NO**

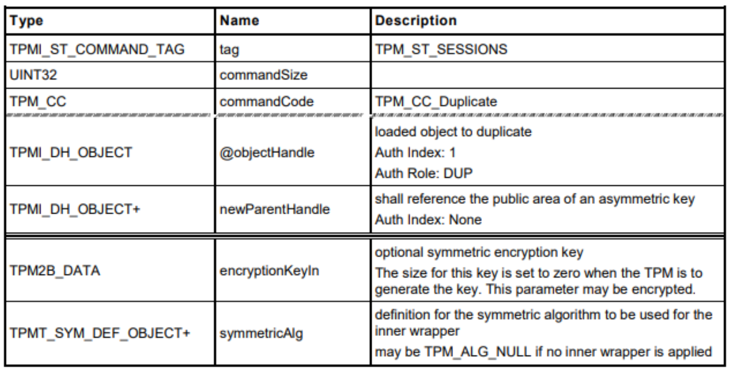


Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_Unseal
TPMI_DH_OBJECT	@itemHandle	handle of a loaded data object Auth Index: 1 Auth Role: USER

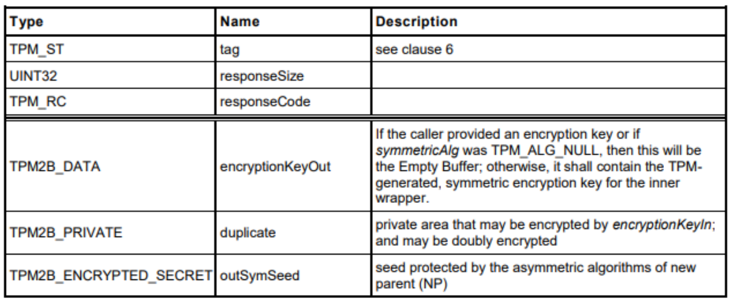


Type	Name	Description
TPM_ST	tag	see clause 6
UINT32	responseSize	
TPM_RC	responseCode	
TPM2B_SENSITIVE_DATA	outData	unsealed data Size of *outData* is limited to be no more than 128 octets.

Command | Description | Parameters | Response	Changes Needed
TPM2_Duplicate	
This command duplicates a loaded object so that it may be used in a different hierarchy.	
See command and response tables below	**NO**

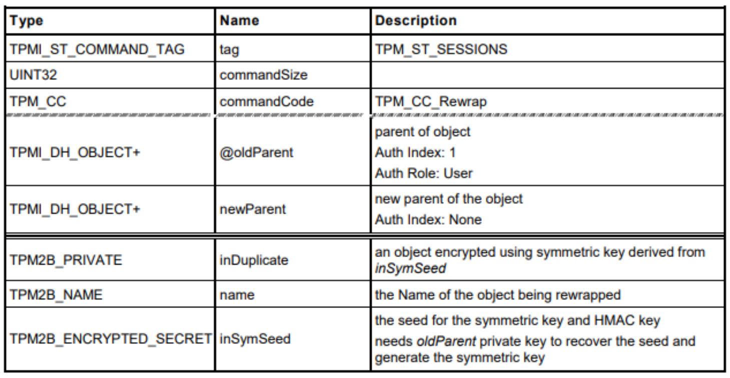


Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_Duplicate
TPMI_DH_OBJECT	@objectHandle	loaded object to duplicate Auth Index: 1 Auth Role: DUP
TPMI_DH_OBJECT+	newParentHandle	shall reference the public area of an asymmetric key Auth Index: None
TPM2B_DATA	encryptionKeyIn	optional symmetric encryption key The size for this key is set to zero when the TPM is to generate the key. This parameter may be encrypted.
TPMT_SYM_DEF_OBJECT+	symmetricAlg	definition for the symmetric algorithm to be used for the inner wrapper may be TPM_ALG_NULL if no inner wrapper is applied

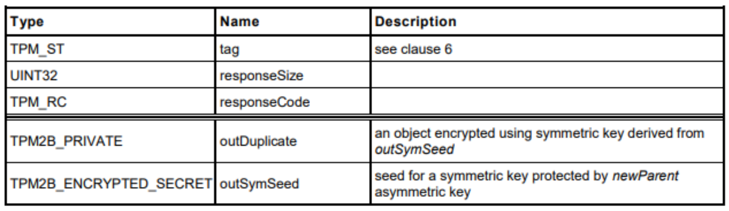


Type	Name	Description
TPM_ST	tag	see clause 6
UINT32	responseSize	
TPM_RC	responseCode	
TPM2B_DATA	encryptionKeyOut	If the caller provided an encryption key or if *symmetricAlg* was TPM_ALG_NULL, then this will be the Empty Buffer; otherwise, it shall contain the TPM-generated, symmetric encryption key for the inner wrapper.
TPM2B_PRIVATE	duplicate	private area that may be encrypted by *encryptionKeyIn*; and may be doubly encrypted
TPM2B_ENCRYPTED_SECRET	outSymSeed	seed protected by the asymmetric algorithms of new parent (NP)

Command | Description | Parameters | Response	Changes Needed
TPM2_Rewarp	
This command allows the TPM to serve in the role as a Duplication Authority	
See command and response tables below	**NO**



Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_Rewrap
TPMI_DH_OBJECT+	@oldParent	parent of object Auth Index: 1 Auth Role: User
TPMI_DH_OBJECT+	newParent	new parent of the object Auth Index: None
TPM2B_PRIVATE	inDuplicate	an object encrypted using symmetric key derived from *inSymSeed*
TPM2B_NAME	name	the Name of the object being rewrapped
TPM2B_ENCRYPTED_SECRET	inSymSeed	the seed for the symmetric key and HMAC key needs *oldParent* private key to recover the seed and generate the symmetric key



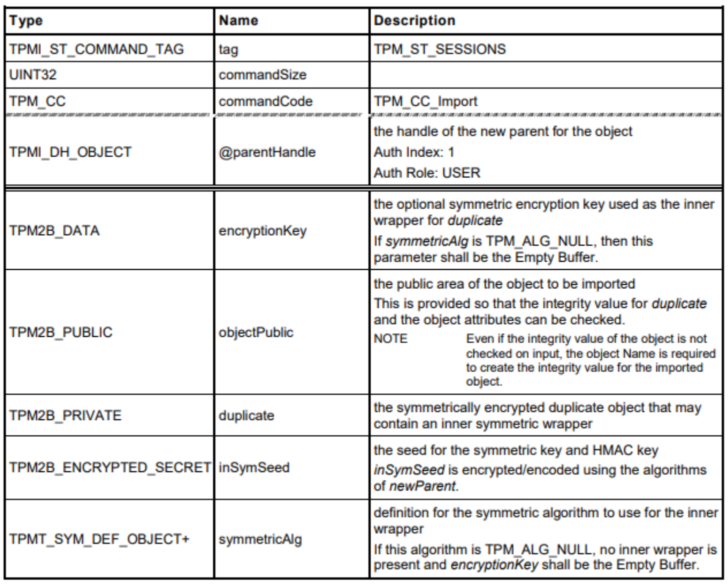
Type	Name	Description
TPM_ST	tag	see clause 6
UINT32	responseSize	
TPM_RC	responseCode	
TPM2B_PRIVATE	outDuplicate	an object encrypted using symmetric key derived from *outSymSeed*
TPM2B_ENCRYPTED_SECRET	outSymSeed	seed for a symmetric key protected by *newParent* asymmetric key

TPM2_Import

Command | Description | Parameters | Response

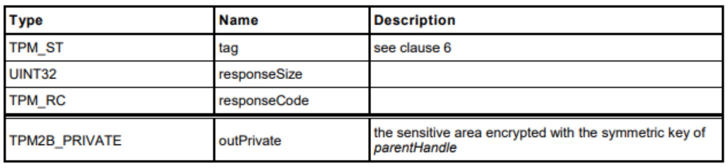
Changes Needed

This command allows an object to be encrypted using the symmetric encryption values of a Storage Key. After encryption, the object may be loaded and used in the new hierarchy.



Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_Import
TPMI_DH_OBJECT	@parentHandle	the handle of the new parent for the object Auth Index: 1 Auth Role: USER
TPM2B_DATA	encryptionKey	the optional symmetric encryption key used as the inner wrapper for *duplicate* If *symmetricAlg* is TPM_ALG_NULL, then this parameter shall be the Empty Buffer.
TPM2B_PUBLIC	objectPublic	the public area of the object to be imported This is provided so that the integrity value for *duplicate* and the object attributes can be checked. NOTE Even if the integrity value of the object is not checked on input, the object Name is required to create the integrity value for the imported object.
TPM2B_PRIVATE	duplicate	the symmetrically encrypted duplicate object that may contain an inner symmetric wrapper
TPM2B_ENCRYPTED_SECRET	inSymSeed	the seed for the symmetric key and HMAC key *inSymSeed* is encrypted/encoded using the algorithms of *newParent*.
TPMT_SYM_DEF_OBJECT+	symmetricAlg	definition for the symmetric algorithm to use for the inner wrapper If this algorithm is TPM_ALG_NULL, no inner wrapper is present and *encryptionKey* shall be the Empty Buffer.

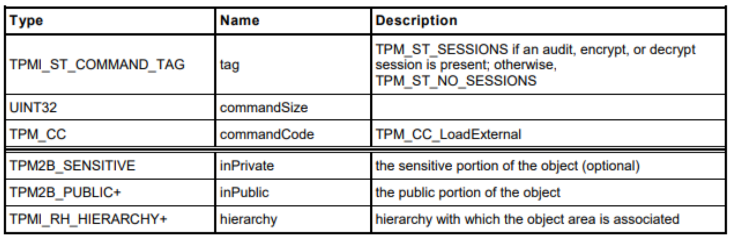
NO



Type	Name	Description
TPM_ST	tag	see clause 6
UINT32	responseSize	
TPM_RC	responseCode	
TPM2B_PRIVATE	outPrivate	the sensitive area encrypted with the symmetric key of *parentHandle*

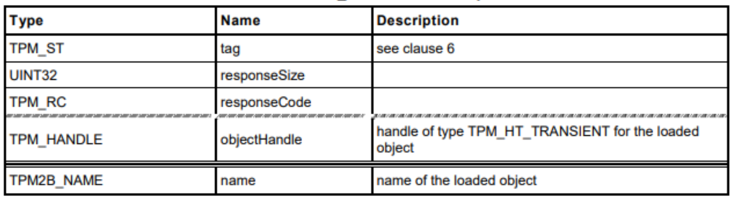
TPM2_LoadExternal

This command is used to load an object that is not a Protected Object into the TPM.



Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS if an audit, encrypt, or decrypt session is present; otherwise, TPM_ST_NO_SESSIONS
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_LoadExternal
TPM2B_SENSITIVE	inPrivate	the sensitive portion of the object (optional)
TPM2B_PUBLIC+	inPublic	the public portion of the object
TPMI_RH_HIERARCHY+	hierarchy	hierarchy with which the object area is associated

NO



Type	Name	Description
TPM_ST	tag	see clause 6
UINT32	responseSize	
TPM_RC	responseCode	
TPM_HANDLE	objectHandle	handle of type TPM_HT_TRANSIENT for the loaded object
TPM2B_NAME	name	name of the loaded object