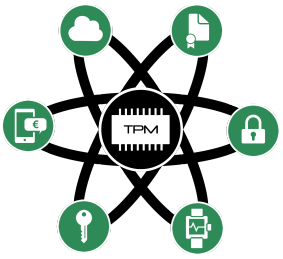

FutureTPM

D3.2

First Report on the Security of the TPM

Project number:	779391
Project acronym:	**FutureTPM**
Project title:	Future Proofing the Connected World: A Quantum-Resistant Trusted Platform Module
Project Start Date:	1st January, 2018
Duration:	36 months
Programme:	H2020-DS-LEIT-2017
Deliverable Type:	Report
Reference Number:	DS-LEIT-779391 / D3.2 / v1.1
Workpackage:	WP 3
Due Date:	1st March, 2019
Actual Submission Date:	13th June, 2019
Responsible Organisation:	SUR
Editor:	François Dupressoir
Dissemination Level:	PU
Revision:	v1.1
Abstract:	In this report, we discuss issues related to modelling and reasoning about trust, usage and authorization policies, and the TPM's cryptographic primitives, protocols, and realization of access control.
Keywords:	TPM, trust, authorization

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

Editor

François Dupressoir (SURREY)

Contributors (ordered according to beneficiary numbers)

Nada El Kassem, Liqun Chen (SURREY)
Sofianna Menesidou (UBITECH)
José Moreira (UB)
Georgios Fotiadis (UL)
Paulo Martin, Leonel Sousa (INESC-ID)
Nikos Koutoumpouxos (UPRC)
Thanassis Giannetsos (DTU)

Disclaimer

Executive Summary

This deliverable reports on current progress towards developing models, proof techniques and proofs of security for the TPM as a whole and its applications, as embodied in the FutureTPM use cases. We tackle issues of trust, authorization and usage policies, and of cryptographic realizations.

We define a framework for trust modelling in complex applications whose security and privacy requirements are achieved—at least partially—using TPMs. Our approach combines predicate- and diagram-based trust modelling techniques to capture trust that emanates from components and their interactions.

We also give an overview of a policy and usage modelling framework that could serve as a bridge between the trust model, the analysis of applications security at a high-level, and the analysis of low-level details of the TPM specification.

We report some progress towards the development of *provably secure* DAA schemes based on quantum hardness assumptions. However, we do not claim the proposed scheme is appropriate for inclusion in the FutureTPM, simply using it as evidence of (partial) realizability, and as a record of relevant proof techniques.

Finally, we use recent analyses of the interactions between the TPM's session and authorization mechanisms, backed by older analyses of the TPM's mechanisms for trust—including the DAA protocol—and of some privacy-oriented use cases to argue that the TPM as currently specified provides insufficient functionality to both bootstrap trust in a trustless software environment *and* support privacy.

We view these issues with the current specification as opportunities to develop sound and solid theoretical foundations for Trusted Computing, that could support the principled design of the next specification, and identify possible directions for these models to develop, in line with the choices made for trust and usage modelling.

This points to the need for the TPM to be modelled and analysed as a whole, rather than as the sum of independent parts, in order to properly capture possible interactions between different uses of the same cryptographic material.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In D3.1 First Report on Security Models for the TPM, we identify a number of research challenges. In this Deliverable, we refine our research methodology on each of them and describe first results towards security modelling and analysis for the TPM, both as a distinctive whole, and as a part of larger secure systems.

We tackle three broad challenges in modelling and analysis:

Trust Modelling Our use cases, and applications of the TPM and Trusted Computing more generally, are supported by complex networks of trust, where resources are shared between mutually distrustful processes and systems. This complex network of trust is an integral part of the security model for the TPM, and our security results and proofs should be fine-grained enough to support—at least—reasoning about the security of our Use Case applications. The right trust model and modelling language will not only allow us to express fine-grained trust assumptions about the TPMs, Hosts and third parties in the system, but will also allow us to *verify* that trust, through dynamic monitoring or enforcement;

Policy Modelling Interactions with the TPM are subject to complex user-defined policies. Some usage patterns will be secure, when others will trivially allow an adversary with any software access to the platform to extract application secrets from its TPM, or break its integrity. Defining security for the TPM as a distinctive whole will therefore require us to first identify usage scenarios we wish to be able to prove security in, where security includes the correct implementation of the TPM's policy enforcement. At the same time, even knowing that the TPM correctly enforces its access control, reasoning about the security of high-level applications will require the ability to check that a particular policy—or complex set of policies set by multiple users—do indeed guarantee that secrets remain secret, and that high integrity data does not get modified without detection.

Cryptography Finally, with the TPM's interfaces better defined, it is necessary to define security models for the whole of the TPM, whose complex functionalities are made even more complex by their mutual interactions. In addition, there are some interesting challenges in modelling and proving security for individual functionalities in isolation—especially in a quantum-resistant setting.

1.1 Methodology

These aspects have so far been investigated by mainly independent teams, supporting fine-grained interactions with other Work Packages in the project as needed. Since refined models

of trust and usage policies—and languages to express them—are necessary to construct the functionality interface for the TPM, we focus the modelling and proof effort for the TPM itself on developing models and proofs for its more complex cryptographic components, and its Direct Anonymous Attestation Mechanism in particular.

Throughout, contributions are mainly guided—initially—by use cases of import to the project. Other interesting applications, which leverage particular aspects of the feature being modelled or analysed are sometimes used to ensure all details are captured, and to identify places where current trusted computing practice can be improved, instead of simply being systematized.

1.2 Structure of the Report

Chapter 2 reviews trust modelling languages and describes a candidate language to serve as the interface between the monitoring and enforcement activities of WP4, and the security analysis of WP3. In Chapter 3, we then review policy modelling and start shaping the interactions between the security models and proofs fo the TPM, and the security models and proofs for systems that make use of it. Chapter 4 reviews the development of security proofs for core cryptographic constructions central to the TPM, and considers aspects related to the authorization and trust mechanisms provided by the TPM as currently specified, and their apparent lack of robustness against the compromise of a single TPM. Finally, Chapter 5 summarizes our findings, identifies remaining gaps, and scopes further research to be conducted.

Chapter 2

Trust Models

In order to provide the FutureTPM project's envisioned services with the appropriate levels of *security*, *privacy* and *assurance*, we need to define trust models that are able to capture the complex relationships between all involved entities and components. This model must not only capture the FutureTPM and the applications (and use cases) that rely on it, but also the environment in which they operate, which may involve an arbitrary number of untrusted third-party entities and assets, as described in D1.1 [24]. We leverage several modelling languages and techniques in combination, to capture assumptions and models of TPM and host operations, and their interaction with the surrounding entities. This combination allows us to express fine-grained trust assumptions in a "top-down" manner—starting from the description of the application's trust domain, and iteratively refining it to model internal interactions between the entities involved, and the specific operations performed by each device). Such fine-grained trust assumptions—once expressed—can be used to precisely delimit the contextual interactions under which the TPM's security guarantees are proved to hold (as part of WP3), and can be monitored, or even partially enforced, through the Control-Flow Attestation Toolkit to be developed as part of WP4.
In particular, the trust models we define here can be refined into attestation policies to be enforced by the Risk Assessment framework, ensuring that all devices perform all expected operations at all levels, as has been described in D4.1 [25]. More detail of this refinement process, including strategies and policies for the enforcement and monitoring of trust in a TPM-backed system, will be given in D4.2.
We first review existing techniques and languages for trust modelling. We then describe our framework for modelling trust as part of the FutureTPM project. Finally, we instantiate that framework to all three FutureTPM use cases.

2.1 Trust Modelling Languages

Trust modelling languages are used to formally define the level of trust of each entity in the operational environment. By this definition, these languages can focus on different aspects of the overall system execution in an attempt to better express the properties of interest to be achieved. Based on the requirements of the use cases envisioned in the context of FutureTPM, several languages have been identified that could be used to formally define the desired notion of trust in TPM equipped devices.

- **Predicate Based Language [51]:** This language is based on a predicative system of mathematical logic. Firstly, an informal problem statement is created that will define the exact requirements (i.e., trust assumptions) needed to patch any known problems so that a device

can be effectively identified as trusted. Then a set of specific predicates is constructed with each one of them mapping to a previously defined requirement. With this in hand, a set of axioms is finally derived from the informal problem statement, that practically glues together the predicates in a way that best represents the "*chain of trust*" that needs to ensured. This way, the problem of identifying the trustworthiness of an entity is modelled and quantified based only on the properties that are of interest in the specific use cases (that, in turn, leverage specific TPM functionalities); thus, allowing for a more fine-grained expression of the trust policies to be deployed.

- **Diagram Based [18, 45]:** Diagram-based models are used to represent trust relationships between interacting entities. Usually their purpose is to investigate how trust is propagated throughout a network or a hierarchy of entities by assessing the trust levels of each component and by identifying the types of strong trust relations (federations) that need to be established among the different entities in the system. This enables the representation of a "Web of Trust", by leveraging tree-like structures, that needs to be established and continuously monitored and can serve as the basis for concluding on stronger arguments about the system's design-level trustworthiness.

- **Algebra Based [3, 36]:** Algebra-based languages are often used in conjunction with the previously described diagram-based models. Once again they focus on trust relationships, that need to be established between interacting entities (rather than the trust modelling of the execution of a deployed platform), following a more formal modelling approach with varying levels of detail. These languages aside from static trust relationship estimates also include protocols to dynamically update the quantitative values used for expressing the federated trust, among entities, based on experience factors.

2.2 A Trust Modelling Framework for FutureTPM

In the context of FutureTPM, the goal is to observe, model and monitor not only the trust level of each TPM-equipped platform but also the strong trust relations that must be established among interacting entities. This requires the consideration of different aspects in each case; for instance, trusting a TPM first requires trusting that it operates correctly, and in particular that sequences of TPM commands are executed correctly, while ensuring that the interactions between attested entities is secure is required in ensure to maintain the trust between them. Thus, the best approach is to use a combination of appropriate modelling languages; namely, the predicate and diagram-based languages. The motivation behind this design choice is that predicates can better serve the modelling of the trusted execution of TPM-equipped devices (at device level), based on both behavioural properties and low-level concrete properties about the entities' configuration and execution, while diagram-based models can enable the expression of the trust federations (network level) that need to be established and maintained to support trust in distributed applications. Overall, the goal is to describe the chain of trusted interactions that need to take place between entities assuming the correct state of each (based on the necessary and sufficient identified predicates and axioms).

The models we describe here will only provide a high-level representation of the networks of trust that need to be established as part of the security models for our applications. For instance, our models (see Section 2.3) contain abstract states, such as "Device Integrity", that express high-level assumption on the attestation state of a device that must be fulfilled before making use of specific TPM functionalities. In order to construct complete application-specific models, such

assumptions will need to be refined, using additional predicates that can specify, for instance, the types of properties that should be attested. We will investigate these low-level predicates, and express them in detail in the context of WP4 (D4.2) as part of the formal modeling of the Control-Flow Attestation Toolkit and its internal attestation policies [25].

The remainder of this chapter follows this top-down approach. First, we identify the common security and safety requirements we aim to achieve with limited trust. Then, we define a semi-formal and high-level trust model in which to describe security and safety of devices and networks, using predicates and state diagrams. Finally, we give concrete representation, in this model, of the FutureTPM use cases. These are expressed as a combination of predicates, axioms and state diagrams.

2.2.1 Trust Assumptions and Security & Safety Requirements

A number of requirements need to be met in order to establish and maintain strong guarantees of trust in a platform, both in the context of the envisioned use cases and for trusted computing-based applications in general. For instance, a common requirement is that each device in the system must be equipped with hardware support for remote attestation. This requirement serves to establish a hardware-based root-of-trust, which cannot be compromised withut physical access to the device, and on which the attestation process will both measure the device's configuration and communicate those results securely and privately to other devices in the system. Furthermore, we require that the device is resistant to non-invasive attacks (such as side-channel attacks, or non-invasive fault injection) while the system should be able to identify nodes that have been offline for a long time and could be victims of more invasive exploitation attempts [62] (such as micro-probing or reverse engineering).

Besides the traditional data confidentiality, integrity and availability, trusted systems must fulfill the following security and trust requirements:

Memory-Safety. Memory safety is a crucial and desirable property for any device loaded with various software components. Its absence may lead to software bugs but most importantly exploitable vulnerabilities that will reduce the trust level of the device running the problematic software. In a nutshell, all accesses performed by loaded processes/services in the underlying memory map of the host device need to be "*correct*" in the sense that they respect the: (i) logical separation of program and data memory spaces, (ii) array boundaries of any data structures (thus, not allowing software-based attacks exploiting possible buffer overflows), and (iii) don't access the memory region of another running process that they should not have access to. Memory-safety vulnerabilities can be detected in design-time with static code analysis techniques [10, 15] and during run-time [57] with the well known tool Valgrind [50] that is designed to identify memory leaks of an executable binary. For instance, memory safety will prevent information from leaking in a security sensitive application that uses a TPM.

Type-Safety. Type-safety is closely related to memory safety as it also specifies a functionality that restricts how memory addresses are accessed in order to protect against common vulnerabilities that try to exploit shared data spaces (i.e., stack, heap, etc.). Type-safety is usually checked during design-time with most programming languages providing some degree of correctness (by default) paired with static code analysis tools that might catch some exceptions not covered by the language compiler (i.e., "fuzzing" tools or concolic execution engines). However, type-safety can also be checked during run-time with the possibility of identifying issues that the static method did not identify [14]

Control-Flow Safety. Besides the aforementioned static methodologies that check the code and the binaries for possible vulnerabilities and bugs, the executable binaries should be dynamically checked for their proper functionality and execution during run-time. This is done through control-flow attestation: All control transfers are envisioned by the allowed program. This translates to no arbitrary jumps in the code, no calls to random library routines, etc. This information is depicted by the allowed control-flow graphs (CFGs) that are calculated prior to the deployment of a service and are used as a baseline of the normal (trusted) sequence of execution states against which run-time control-flow footprints will be assessed [2, 40]. The basic attribute that is required here is that the binary monitoring entity (tracer), the attesting entity (prover) and the checking entity (verifier) are all operating in a trusted mode and are in a state to correctly identify bad behaviour from the monitored binary.

Operational-Correctness. This concept is an intermediate abstraction of control-flow safety. Besides integrating the control-flow mechanism that was described before, it also checks for the static state of the system and relies on the fact that a crucial part of the underlying kernel is in a trusted state. The operational-correctness aims to provide a more holistic view of the system by combining dynamic and static data collected by the FutureTPM Control-Flow Attestation Toolkit [25] in order to produce guarantees on the operational trust state of the system.

Cryptography. Having strong cryptographic primitives is a fundamental requirement of any security oriented system. What is needed towards this direction is a good source of entropy that will be utilized in a secure pseudo-random number generator (PRNG) so that the keys generated by the system are secure. To make good use of this source of entropy, we also must ensure that the cryptographic primitives deployed in the TPM and related systems are fit for purpose. Although in most cases, the security of cryptographic primitives is a matter of design, the system's cryptographically secure pseudo-random generator, which is used in particular to generate keys, is often left to implementers, with potentially disastrous consequences on the security of the whole system [49]. In the context of the FutureTPM, we assume security against QS1 adversaries [21]. We note that analysing the cryptographic primitives and their usage in the TPM and in the FutureTPM use cases is in scope of the FutureTPM project, and discuss some aspects in Chapter 4.

Physical Security. TPMs are discrete hardware chips that interconnect with the Low Pin Count (LPC) bus of the system through. The LPC interface can be subject to attacks from eavesdropping to injection. Many recent and old attacks [47, 48, 67, 4] have shown that through the LPC interface, an attacker can spoof PCR values and steal sensitive data (like the BitLocker disk encryption key), bypassing critical TPM trust guarantees. That is why the physical security of both the device as a whole and the actual pins that connect the TPM on the device motherboard should be carefully designed if the TPM is to be trusted. Another option that has been investigated by [7, 38], is to constantly require each device to provide a "heartbeat" attestation in a specific frequency. Because hardware attacks are time consuming and require that the device is taken offline for a considerable amount of time, the TPM will not be able to provide the attestation messages timely. In this case, the device should be considered as untrusted or partially trusted depending on the policies and the trust requirements that are in place.

Definition 1. System Correctness & Trustworthiness*: A system S is at a trusted state if and only if all of the above requirements are successfully met.*

For the purpose of the FutureTPM project, some of the requirements are left out of scope, and will be considered as explicit assumptions.

2.2.2 Formal Trust Model

In this section we will elaborate a formal trust model which directly maps to the requirements and assumptions described previously. Based on the use of the aforementioned predicate-based language, this model is split into three components: (i) the predicates which are essentially the "words" of the language, (ii) the axioms which define how these predicates fit together to produce meaningful trust statements, and (iii) the assumptions which are the predicates that are out of the scope of the FutureTPM solution.

The predicates are meant to be the dictionary of the trust model that enlists each statement as a word and pairs it with its meaning. They are split into six categories. The first one (Table 2.1) aims to give a set of predicates that represent the final (and trusted) stages of the overall system and network (trusted domain). These will be satisfied only when the system and the TPM-equipped devices are trusted; if one precondition of these predicates fails then the system will be in an untrusted state and no guarantees on the security posture can be verified. The second and third set of predicates (Tables 2.2 and 2.3, respectively) represent intermediate states of the system and the TPM-equipped devices. These intermediate states are meant to group together the requirements in separate categories based on the previously described security and safety requirements. For instance, the $CryptoSafe_{TPM}(T)$ represents requirements for the cryptographic security of the TPM while the $MS_{System}(S)$ represents requirements related to the memory safety of the system. Likewise, predicates in the fourth and the fifth categories (Tables 2.5 and 2.4, respectively) represent the actual requirements that were fit into the aforementioned categories. For example the $DynamicAnalysis_{MS}(A, S)$ predicate translates to the fact that application A was tested dynamically for memory issues which in turn fits under the $MS_{System}(S)$ intermediate state predicate. Finally, we use a separate category (Table 2.6) to capture such low-level predicates on the configuration and execution properties of the system such as firmware running, the version of its configuration file or presence of specific hardware properties, ports and network interfaces, etc. It is separated from the other groups as it aims to cover local and remote identification of a TPM with the target of modelling the trust of specific TPM functionalities considered in the envisioned use cases (i.e., Direct Anonymous Attestation in the context of Personal Activity and Health Kit Data Tracking).

The axioms (Table 2.7) define exactly how these requirements and their intermediate stages fit together in an overall model that leads up to the trusted state of a device or network. These axioms glue together the low-level predicates, the intermediate predicates and the high-level predicates according to the architecture described above. Each low-level predicate is defined to partially satisfy the intermediate category it belongs to and each intermediate TPM or System predicate builds up to the final $Trusted_{TPM}(T)$ and $Trusted_{System}(S)$ high-level predicates. For example, the $Trusted_{Tracer}(S)$ that defines the tracer running on system S is trusted, is required by the intermediate predicate $CFS_{System}(S)$ that represents the control-flow safety of the system and the $CFS_{System}(S)$ which in turn is needed by the $Trusted_{System}(S)$ predicate. This relationship between the axioms can be seen in Figure 2.1 where each axiom (labelled by the letter inside the brackets found in Table 2.7) points to its direct dependencies creating an axiom dependency tree.

Finally, Table 2.8 describes the assumptions we do not intend on discharging, as they are our of the scope of the FutureTPM use cases, where the focus is on specific TPM functionalities as

identified in D4.1 [25]. More specifically, these are:

1. The physical security of the hardware;
2. The source of entropy of the system; and
3. The proper identification of the installed TPM.

We leave those as assumptions in this chapter, but note that some may be addressed during the course of the project. In particular, WP5 may consider some aspects of physical security (item 1), including hardware and software side-channels, and our analysis of TPM cryptography will consider how trust can be bootstrapped to avoid impersonation of new TPMs by malicious entities with access to other TPMs (item 3). We review some recent attacks related to this issue, and discuss potential solutions to consider during the security analysis of the TPM's mechanism in Section 4.2.

Finally, we have to highlight that the security of the BIOS/Kernel/OS of the system is considered as a prerequisite. If the kernel is approached as a monolithic system, then it should be assumed that it is trusted in its whole since if even a single component diverges then the entire kernel is deemed untrusted ($Trusted_{Soft}(S)$). On the other hand, the kernel in the emerging edge- and cloud-computing applications where everything is considered as a service, can also be seen as a set of micro-services where only a specific set of them should be considered trusted in order for the entire system to be at a correct state. This reduction of the trusted code base of the kernel can introduce a chance for the tracing capabilities of the FutureTPM Control-Flow Attestation Toolkit to monitor exactly those functionalities. Thus, the security of the BIOS/Kernel/OS of the system is caught within the scope of the FutureTPM project depending on how this entity is considered.

Besides these considerations, we will also assume that the TPM itself satisfies memory-safety, type-safety and control-flow safety. In other words: hardware TPMs are correct, secure and tamper-resistant. With these given assumptions, the FutureTPM solution should be able to *assess*, *monitor* and *verify* the trust level of a network of devices based on the following model.

Table 2.1: High-level predicates for trusted TPMs, systems and domains

High Level Predicates	
$Trusted_{Domain}(D)$	The domain D composed of a set of TPM-enabled devices is trusted
$Trusted_{System}(S)$	The system S is trusted
$Trusted_{TPM}(T)$	TPM T is trusted

Table 2.2: Intermediate predicates representing abstract states necessary for TPM trust.

Intermediate TPM State Predicates	
$PhySecure_{TPM}(T)$	TPM T is **physically secure**
$MS_{TPM}(T)$	TPM T has **memory safety**

Intermediate TPM State Predicates (continued)	
$TS_{TPM}(T)$	TPM T has **type safety**
$CFS_{TPM}(T)$	TPM T has **control flow safety**
$CryptoSafe_{TPM}(T)$	TPM T uses **secure cryptographic primitives**
$Verified_{TPM}(T)$	TPM T is verified and it is resilient against the Cuckoo attack.

Table 2.3: Intermediate predicates representing abstract states necessary for trust.

Intermediate System State Predicates	
$OP_{Correctness}(s)$	The system S has **operational correctness** based on specific properties and functions to be attested through the control-flow property-based attestation toolkit of WP4
$PhySecure_{System}(s)$	System S is **physically secure**
$MS_{System}(S)$	System S has **memory safety**
$TS_{System}(S)$	System S has **type safety**
$CFS_{System}(S)$	System S has **control flow safety**
$Database_{Correctness}(S)$	The database measurements of system S are correct and trusted by the communicating parties.

Table 2.4: Predicates for TPM-specific trust requirements.

TPM Predicates	
$Trusted_{Comm}(T)$	The communication of TPM T with the TSS is trusted
$TPM2_{Compliant}(T)$	TPM T is compliant with the latest TPM2.0 specification
$PhySecure_{LPC}(T)$	The pins (LPC) connecting TPM T with the system are physically secure
$Secure_{PRNG}(T)$	The PRNG that TPM T uses is secure
$PQCrypto_{Asymmetric}(T)$	The asymmetric cryptographic primitives that TPM T uses are post-quantum secure
$PQCrypto_{Symmetric}(T)$	The symmetric cryptographic primitives that TPM T uses are post-quantum secure

Table 2.5: Predicates for the OS and the TPM-backed applications.

System Predicates	
Trusted$_{Soft}(S)$	The (monolithic) BIOS/Kernel/OS that runs on a system S are trusted
PartTrusted$_{Soft}(S)$	Specific parts of the BIOS/Kernel/OS (ie microservices) that runs on a system S are trusted
Trusted$_{CFPA}(S)$	The attestation toolkit checking the control flow integrity of System S is trusted
Integrity$_{Code}(A, S)$	In a system comprising of multiple applications, application A running on system S and using a TPM, has been checked for its authenticity & integrity.
CompilerOps$_{TS}(A, S)$	All type safety related compiler options for application A running on system S are enabled
StaticAnalysis$_{MS\&TS}(A, S)$	In a system comprising of multiple applications, application A running on system S and using a TPM has been put through static code analysis for memory safety and type safety issues
DynamicAnalysis$_{MS}(A, S)$	In a system comprising of multiple applications, application A running on system S and using TPM has been put through dynamic analysis for memory safety issues
StaticAnalysis$_{State}(S)$	The state of system S is checked against a set of policies for allowed components to be installed in the device.
PhySecure$_{Circuit}(S)$	The circuits of the system S are adequately physically secure

Table 2.6: Predicates for recognition and identification of local and remote TPMs.

Local/Remote TPM Predicates	
Installed$_{TPM}(S, T)$	TPM T is installed on system S
Provisioned$_{TPM}(S, T)$	TPM T has been initialized and setup for system S (TPM Provisioning includes: turning the TPM on, making sure it has an endorsement key, making sure it has an endorsement credential, taking ownership of the TPM, creating any initial keys and certifying these keys.)
Identified$_{Local}(T)$	The local TPM T is securely identified and recognized as a legit TPM by the end user. The TPM must be checked by the end user that it is indeed installed on the system in order to prevent the Cuckoo attack [51].

Local/Remote TPM Predicates (continued)	
TPMCredential$_{Cert}(T, I)$	A TPM T is certified by issuer I during the join phase of the DAA protocol.

Table 2.7: Axioms for constructing trusted systems and domains. The axioms directly map to the informal model of the previous section.

Axioms
(Ax$_{01}$) $R[t \leftarrow T], R[Sr_i \leftarrow Sr], R[s \leftarrow D] : Trusted_{System}(s) \wedge Trusted_{TPM}(t, Sr_i) \Leftrightarrow Trusted_{Domain}(D)$, where $\forall(s_i, t_j) \in D$ For every TPM t, from a set of secure TPMs T, the services Sr_i it provides, installed in all systems, and for every system s (hosting a TPM) that belong to domain D: The domain D (comprising of all s_i hosting TPMs t_j) is considered trusted if and only if every system s_i and TPM t_j are both trusted. This does not include the communication channel between the devices.
(Ax$_{02}$) $PhySecure_{System}(S) \wedge MS_{System}(S) \wedge TS_{System}(S) \Leftrightarrow Trusted_{System}(S)$ A System S is trusted if and only if it has physical security, memory safety, type safety, control-flow safety and operational correctness.
(Ax$_{03}$) $Database_{Correctness}(S) \wedge MS_{System}(S) \wedge TPMCredential_{Cert}(T, I) \wedge R[Sr_i \leftarrow Sr] : PhySecure_{TPM}(T) \wedge CryptoSafe_{TPM}(T) \wedge Verified_{TPM}(T) \Leftrightarrow Trusted_{TPM}(T, Sr_i)$ A TPM T and the services Sr_i (encryption, signing, key management etc.) it provides, are trusted if and only if T has physical security, secure cryptography and it is properly identified. These services include the functionality of the FutureTPM reference scenarios.
(Ax$_{04}$) $Op_{Correctness}(S) \wedge Integrity_{Code}(A, S) \wedge StaticAnalysis_{MS\&TS}(A, S) \wedge DynamicAnalysis_{MS}(A, S) \Leftrightarrow MS_{System}(S)$ A system S has memory safety if and only if its applications (A_i) have operational correctness, code integrity and they have been undergone static and dynamic analysis. Each application (A_i) is quantified and represented through its internal control-flow path (CFP_i). The collection of ($CFP_i \forall A_i$) constitute the overall control-flow graph of system S (CFG_S). Check Ax$_{06}$.
(Ax$_{05}$) $CompilerOps_{TS}(A, S) \wedge Integrity_{Code}(A, S) \wedge StaticAnalysis_{MS\&TS}(A, S) \Leftrightarrow TS_{System}(S)$ A system S has type safety if and only if proper compiler options are set, the code is checked for its integrity and it has also been undergone static analysis. Check Ax$_{06}$.
(Ax$_{06}$) $(R[cfp_i \leftarrow CFP], CFG : cfp_i \in CFG) \wedge CFS_{System}(S) \wedge StaticAnalysis_{State}(S) \wedge PartTrusted_{Soft}(S) \Leftrightarrow Op_{Correctness}(S)$ A system S has operational correctness if and only if each monitored control-flow path CFP$_i$, of a process i, is attested against the set of expected and measured control-flow paths in the overall graph. The system must also have control-flow safety, a partially trusted kernel (for verifying the correctness of the local control-flow attestation service) and it should be checked for its currents state.

Axioms (continued)
Ax$_{07}$ $Trusted_{CFPA}(S) \Leftrightarrow CFS_{System}(S)$
A system S has control flow safety if and only if the attestation toolkit is in a trusted state
Ax$_{08}$ $PhySecure_{Circuit} \Rightarrow PhySecure_{System}(S)$
A system S is physically secure if its circuits are properly protected against tampering.
Ax$_{09}$ $PhySecure_{LPC}(T) \Rightarrow PhySecure_{TPM}(T)$
A TPM T is physically secure if its interface is properly protected.
Ax$_{10}$ $Secure_{PRNG}(T) \wedge PQCrypto_{Asymmetric}(T) \wedge PQCrypto_{Symmetric}(T) \Leftrightarrow CryptoSafe_{TPM}(T)$
A TPM T is cryptographically secure if and only if it has a secure PRNG and it uses safe PQ symmetric and asymmetric primitives with the required configurations.
Ax$_{11}$ $\forall S, T \in LocalTPM : Installed_{TPM}(S,T) \wedge Provisioned_{TPM}(S,T) \wedge Identified_{Local}(T) \Rightarrow Verified_{TPM}(T)$
A TPM T is verified if and only if it is installed on system S, provisioned and identified by the end user.

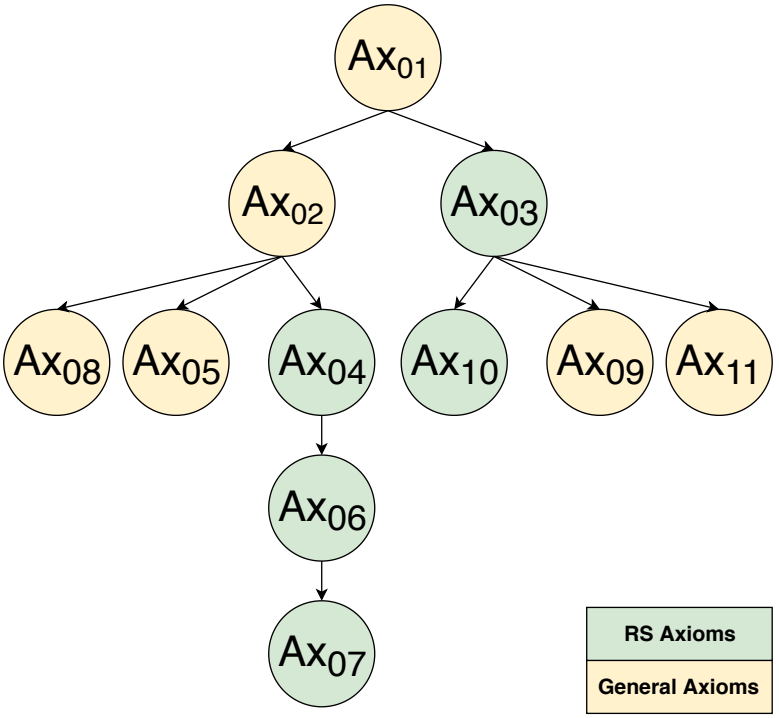

Figure 2.1: Relationships between the axioms of Table 2.7. The axioms are divided in general axioms (yellow) and FutureTPM reference scenario axioms (green).

As is depicted in Figure 2.1, the axioms are categorized in "*general*" and "*reference scenario*" (RS) axioms. This categorization provides further clarity in the defined trust models, and helps us better describe the subsets of axioms that best express the requirements of the specific (TPM) functionalities that each reference scenario requires. The arrows represent the trust chain that

needs to be established in order to achieve the required trust assumptions; this trust chain comprises the general axiom representing the final (and trusted) stage of the overall system and network (trusted domain), as the root node, and the intermediate predicates/axioms (as tree and leaf nodes) required to be fulfilled for this general axiom to be true. As is also evident from Table 2.7, in order to have the necessary guarantees that an axiom is true in a system, a number of intermediate axioms need to also be valid. For instance, in order for a domain, (D), to be trusted (Ax_{01}) then all systems comprising this domain, $(\forall s_i \in D)$, must be trusted (Ax_{04}) which in turns requires that each system, s_i, has memory safety (Ax_{04}), type safety (Ax_{05}), operational correctness (Ax_{06}) and contro-flow safety (Ax_{07}). Based on this approach, the following trust chains need to be established for each reference scenario:

1. **Reference Scenario 1 — Secure Mobile Wallet and Payments:** This scenario requires device integrity which is covered by $Op_{Correctness}$ — Ax_{06}, execution integrity which is covered by $MS_{System}(S)$ — Ax_{04} and symmetric cryptographic security which is covered by $PQCrypto_{Symmetric}$ — Ax_{10}.

2. **Reference Scenario 2 — Personal Activity and Health Kit Data Tracking:** This scenario requires device integrity which is covered by $Op_{Correctness}$ — Ax_{06}, execution integrity which is covered by $MS_{System}(S)$ — Ax_{04}, cryptographic security for both symmetric and asymmetric cryptography which is covered by $CryptoSafe_{TPM}(T)$ — Ax_{10} and $Database_{Correctness}(S)$ which is covered by $Trusted_{TPM}(T, Sr_i)$ — Ax_{03}.

3. **Reference Scenario 3 — Device Management:** This scenario requires device integrity which is covered by $Op_{Correctness}$ — Ax_{06}, execution integrity which is covered by $MS_{System}(S)$ — Ax_{04} and cryptographic security for symmetric cryptography which is covered by $PQCrypto_{Symmetric}$ — Ax_{10}.

Table 2.8: Trust assumptions in FutureTPM use cases.

Trust Assumptions on Trusted Computing	
$PhySecure_{LPC}(A)$	$Trusted_{Comm}(A)$
$PhySecure_{Circuit}(A)$	$Trusted_{Soft}(A)$
$Secure_{PRNG}(A)$	$Installed_{TPM}(S, A)$
$Provisioned_{TPM}(S, A)$	

Based on this trust modeling, we then proceed to also extract the state diagrams that best capture the trust relationships that need to be established between all interacting entities in each reference scenario. Recall that the intuition behind using both types of trust modeling languages (predicate- and diagram-based) is to be able to have a better representation of the trust assumptions in a "top-down" manner - starting from the description of the application's trust domain, and iteratively refining it to model internal interactions between the entities involved, and the specific operations performed by each device. For the latter, the use of predicates (and the extraction of the subsequent axioms) is better suited for capturing all the low-level system and behavioral properties that need to be guaranteed, in order for a system (hosting a TPM) to be trusted, whereas for the former the use of state-based diagrams is more appropriate for expressing the (trust)

relationships that need to be established between this hierarchy of systems/entities.

Thus, an inherent assumption is that both both categories (general & reference scenario axioms) are needed in order for a TPM and in turn a domain of systems to be trusted; each state of the following diagrams also assumes the trust level of each system which is expressed through the aforementioned predicates and axioms. This information is derived by and provides a meeting point with the next chapter where the functionalities of each reference scenario is analysed in state diagrams. (Figures 2.2, 2.3, 2.4).

2.3 State Diagrams

We can now discuss instantiations of the modelling language discussed so far to the three FutureTPM reference scenarios. The discussions below focus mainly on modelling the trust relations between the different entities in each of the system, with the specifics of device-level low-level predicates to be defined in D4.2. Thus, the state diagrams shown below, model the reference scenarios' environment based on the core functionalities and the TPM commands (per use case) identified in D4.1 [25].

As depicted in the following diagrams, we capture trust relations and the mechanisms through which they are measured a sequence of states that best reflect the correct sequential execution of core TPM functionalities of interest. Depending on the scenario, this may be sealing, unsealing, trusted communication with third-parties, DAA, ... These states are used to represent the incremental construction of complex trust relations between multiple TPM-equipped platforms and third-party servers, thus enabling the establishment of "*chains of trust*" among all conceptual components and entities. In order to support the secure provision of the service under study, all architectural entities must meet the requirements of their current states if overall security is to be guaranteed; if any of the desired state is not achieved then overall security cannot be assured.

Furthermore, the "*Execution Integrity*" state, identified with a black colour, reflects the continuous run-time monitoring and verification of the integrity of a software-based service running in the deployed platforms. It essentially captures the requirement for TPM-based services to securely attest to their integrity. This is achieved through the advanced *remote attestation* functionalities offered by the Control-Flow Property-based Attestation Toolkit (CFPA) described in D4.1 [25] and to be further refined in D4.2. This state, intuitively, serves to guarantee the integrity of the TSS execution during run-time, especially against attacks that attempt to maliciously tamper with the program's control-flow [63, 31]. These types of attacks often try to exploit memory- [32] and data-related vulnerabilities [20] to alter the execution path of the underlying system processes; either by injecting new malicious code or by dynamically generating malicious programs based on already existing benign code snippets [55]. Such Remote Code Execution vulnerbilities are mong the most devastating, with pticularly significant consequences in the world of Trusted Computing, since they can bypass static attestation techniques (whose application is depicted through the "*Device Integrity*") by allowing the adversary full control of a platform *after* measurement of an uncompromised configuration. The mechanisms that will be leveraged by the CFPA toolkit and the trust models that will further refine the sufficient and necessary behavioural and execution properties that need to be attested per use case, will be detailed in D4.2

2.3.1 Reference Scenario 1 — Secure Mobile Wallet and Payments

Figure 2.2 below presents the two state diagrams of the main functionalities (i.e. sealing, unsealing) of the Secure Mobile Wallet and Payments reference scenario.

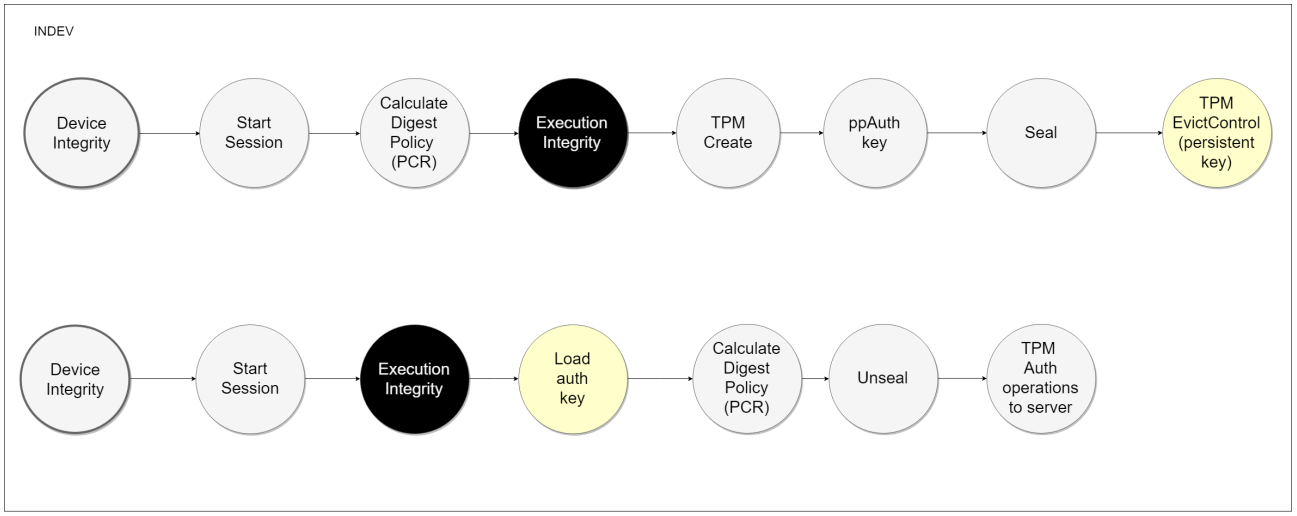

Figure 2.2: Secure Mobile Wallet and Payments State Diagrams

The Sealing Operation (Figure 2.2 (a)):

Device Integrity: Integrity calculations of the applications installed on the TPM-equipped deployed platform. Applications need to be attested against a whitelist of allowed and trusted application instances; any deviation will result in the TPM not allowing the further execution of the system operations.

Start Session: Start a trial session in order to calculate the Digest Policy (based on the correct state of the installed applications) and then create the necessary authorization key for connecting to the Free POS server.

Calculate Digest Policy (PCR): Checks the policy digest with the Device Integrity values of the application whitelist and the password to be provided by the user.

Execution Integrity: Integrity of the execution of the TPM command flow by monitoring the Trusted Software Stack (TSS); i.e., invocations made by what applications, command configurations, parameters passed, etc. It reflects the continuous monitoring and attestation of the low-level system and behavioural properties to be performed by the Control-Flow Attestation toolkit (more information to be included in D4.2).

TPM Create: Creates a password protected key (ppAuth key) for the authorized communication of this TPM-equipeed device with the Free POS server.

ppAuth key: The password protected authorization key.

Seal: Seal the ppAuth key, in order to be used only when the password-based authorization policy is met. The key is sealed based on a) the authorization policy, that express the device integrity requirements through the whitest of allowed applications and b) the password to be provided by the user.

TPM EvictControl (persistent key): Make the key persistent by storing it in the volatile memory of the TPM and not allowing its usage outside of the TPM environment. This secure storage is associated with the TPM and, when authorized by the user, the TPM allows access to the stored secrets. The TPM does this via the Protected Storage Hierarchy (PSH), a tree of keys containing: (i) a root key, (ii) sets of user keys and identity keys as leaf keys, and (iii) a set of encrypting keys (in the middle) between the root and leaf keys. However, we have to highlight that this trust assumption may be weakened in the case where secure migration of the keys is required. This will allow the transferring of the "key wallets" from one TPM of a user to another using means like TPM key migration in order to allow users to maintain multiple "key wallets". This trusted migration service allows users to access the Free POS server from any device they are using being confident that they "key wallet" has not been tampered with or changed in anyway.

The Unsealing Operation (Figure 2.2 (b)):

Device Integrity: Integrity calculations of the applications installed on the TPM-equipped deployed platform. Applications need to be attested against a whitelist of allowed and trusted application instances; any deviation will result to the TPM not allowing the further execution of the system operations.

Start Session: Start a trial session in order to calculate the Digest Policy (based on the correct state of the installed applications).

Execution Integrity: Integrity of the execution of the TPM command flow by monitoring the Trusted Software Stack (TSS); i.e., invocations made by what applications, command configurations, parameters passed, etc. It reflects the continuous monitoring and attestation of the low-level system and behavioural properties to be performed by the Control-Flow Attestation toolkit (more information to be included in D4.2).

Load ppAuth key: Loads the password protected authorization key. This state will be used if we decide to remove the TPM EvictControl state form the previous state diagram.

Calculate Digest Policy (PCR): Checks the policy digest with the Device Integrity values of the application whitelist and the password.

Unseal: Unseals the ppAuth key.

TPM Auth operations to server: Authentication operations to the server based on the ppAuth key.

2.3.2 Reference Scenario 2 — Personal Activity & Health Kit Data Tracking

As has been described in D1.2 [23] and D4.1 [25], this reference scenario attempts to implement: (i) data transfer preserving **anonymity** and **privacy**, using the **Direct Anonymous Attestation (DAA)** functionality, (ii) attestation of the correct state of the S5 Tracker Analytics Engine that will store the user (anonymously) provided data in a verified and attested database, and (iii) the establishment of a secure and anonymous communication channel between the user devices and the S5 Tracker. Thus, the derived trust model captures the necessary requirements for the correct execution of these services:

R1. Correctness: Valid DAA signatures are verifiable, and linkable, where needed. This also requires the correct execution of the protocol even in the presence of an adversary having compromised part of the host $PLATFORM$. For instance, only valid and trustworthy TPMs can join the system by ensuring that the endorsed TPM keys have not been previously compromised;

R2. User-controlled anonymity: Identity of the user cannot be revealed from the DAA signature. This means that an adversary who does not know the $PLATFORM's$ private key cannot link a signed message to the TPM of this platform;

R3. User-controlled linkability: Users control whether signatures can be linked. A user has control over its DAA credential and can decide whether or not to "blind" it through the use of a single or different basenames bsn;

R4. Non-frameability: Adversaries cannot produce signatures originating from a valid trusted component. Essentially, this represents that no combination of dishonest $ISSUERS$ and $PLATFORMS$ can create a valid signed message m unless this signature was produced by an honest $PLATFORM\ D_i$.

Figure 2.3 depicts the state diagrams for the aforementioned functionalities: The first one denotes the correct execution of the core DAA phases [21], namely the SETUP and JOIN phases (upper branch), for certifying the TPM used by a host platform from the $ISSUER$, and the SIGN (or VERIFY) phases (lower branch) for signing/verifying message digests, m_i, originating from the users. The second diagram denotes the establishment of a secure communication channel between a user device and the S5 Tracker Analytics Engine capturing such a TLS key establishment (e.g., based on the execution of a Diffie-Hellman protocol) that can be anonymously signed by the host $PLATFORM$ using its DAA key. Finally, the third one denotes the attestation and verification of the correct state of the S5 Tracker and its database that will store the provided data (through quoted PCR values generated by the TPM component of the S5 Tracker). Overall, the light green colour represents a state of the host $PLATFORM$ (and not the TPM itself), the light blue colour represents a state of the S5 Tracker and the light grey colour represents a state of the TPM.

State Diagram of the DAA Protocol Execution (Figure 2.3 (a)):

Device Integrity: Integrity calculations of the applications installed on the TPM-equipped $PLATFORM$. Applications need to be attested against a whitelist of allowed and trusted application instances. The S5 Tracker will be responsible for managing such whitelists in order to determine whether the DAA key generated by the TPM of a $PLATFORM$ is associated to a good software configuration. This property will be achieved by the TPM that can securely store the current system state in its Platform Configuration Registers (PCRs) and it will allow certain crypto operations to be performed with the DAA key only if the current state is the same as when the key was created.

Start Session: $PLATFORM$ applications to start a session with the TPM in order to create the DAA key and activate its credentials (SETUP and JOIN phases) and then execute a range of signing/verification operations: **SETUP**: The system parameters must be chosen and the $ISSUER$ needs to generate its keys. These parameters and the $ISSUER'S$ public keys are then published and available to anyone who wants to verify the validity of a signature. **JOIN:** a user $PLATFORM$ using a TPM obtains an Attestation Key Credential (from the $ISSUER$) for the DAA key create by the TPM.

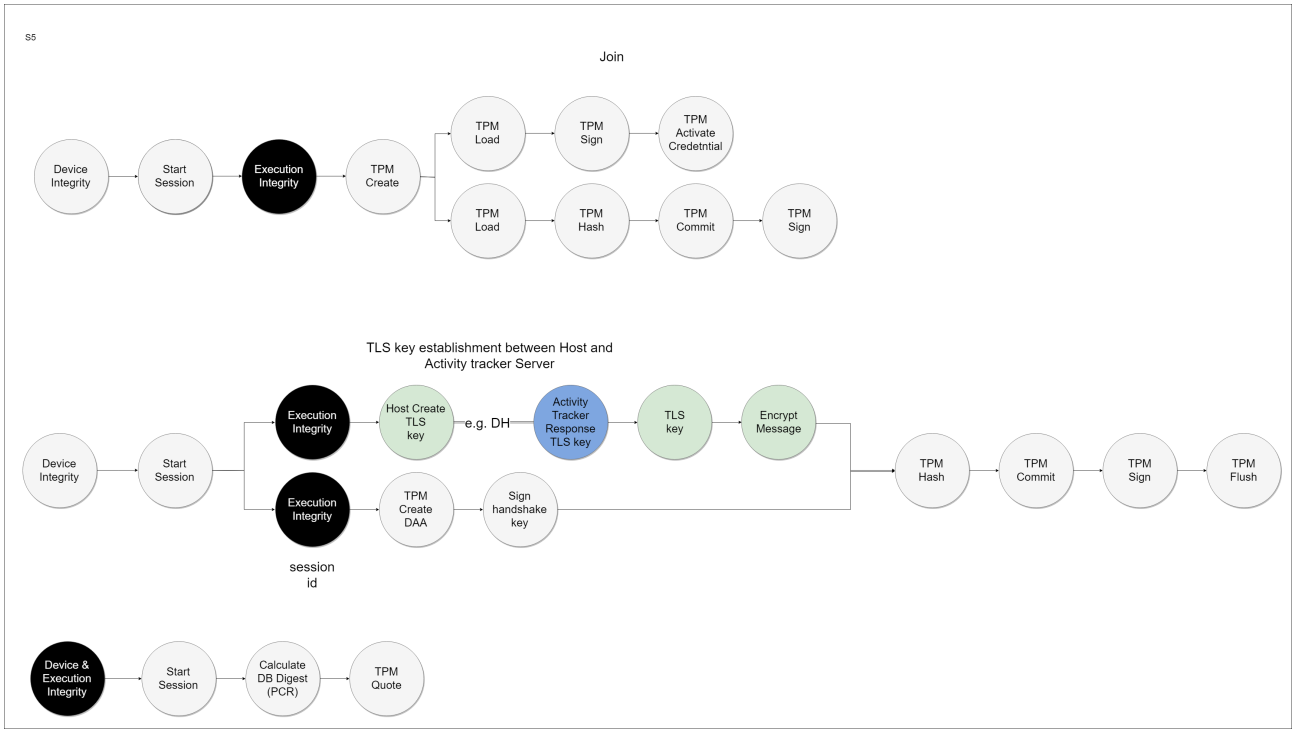

Figure 2.3: Activity Tracking State Diagrams

Execution Integrity: Integrity of the execution of the TPM command flow by monitoring the Trusted Software Stack (TSS); i.e., invocations made by what applications, command configurations, parameters passed, etc. It reflects the continuous monitoring and attestation of the low-level system and behavioral properties to be performed by the Control-Flow Attestation toolkit (more information to be included in D4.2). The control-flow path (CFP) associated with this specific session can be "binded" to the TPM session ID. This will provide the necessary guarantees that the tracing of this CFP is trusted as it has been signed by the TPM itself, thus, weakening the requirements on the trustworthiness of the BIOS/Kernel (in the case of micro-service oriented architecture - see Section 2.2.2).

TPM Create: Creates a restricted key blob, in order to create the DAA key.

Activate Attestation Key Credential in JOIN Phase:

TPM Load: The TPM loads the created $ECC - DAA$ key. This key must be fixed to this TPM, fixed to this $TPM's$ key hierarchy and restricted to sign only message digests been created by itself (once data have been forward by the use $PLATFORM$ hosting the TPM).

TPM Activate Credential: Enables the association of a credential with an object (provided by the $ISSUER$) in a way that ensures that the TPM has validates the provided system parameters. In a nutshell, this TPM call is used to convince the $ISSUER$ that the $ECC - DAA$ key that it has received, has been generated by a TPM whose endorsement key has already been checked.

Activity Tracker Response TLS key: Establishing a secure communication Channel from the S5 Activity Tracker Server to the $PLATFORM$.

TPM Sign: In the context of the JOIN protocol and in order to successfully complete the TPM activate credential command, it is necessary to perform one TPM sign based on the created DAA key.

The SIGN Operation:

TPM Load: The TPM loads the required keys for the signing operation

TPM Hash: Compute hash digests for the data bunches produced by the host $PLATFORM$ after being forwarded to the internal TPM. This operation provides the necessary guarantees that the message digest to be later signed have been created by the TPM itself. The results of the hash will be used in the signing operation that uses the restricted DAA key and the ticket returned by this command can indicate that the hash is safe to sign.

TPM Commit: After the attestation key certificate has been randomised, this command is used for preparing the parameters for the subsequent signing operation. It basically provides the required anonymity level by using either different or the same basename for the signing.

TPM Sign: After the execution of all previous states, the $DAAkey$ can now be used for any signing operation.

State Diagram of the TLS Key Establishment (Figure 2.3 (b)):

Device Integrity: Integrity calculations of the applications installed on the TPM-equipped $PLATFORM$. Applications need to be attested against a whitelist of allowed and trusted application instances. The S5 Tracker will be responsible for managing such whitelists in order to determine whether the DAA key generated by the TPM of a $PLATFORM$ is associated to a good software configuration. This property will be achieved by the TPM that can securely store the current system state in its Platform Configuration Registers (PCRs) and it will allow certain crypto operations to be performed with the DAA key only if the current state is the same as when the key was created.

Start Session: $PLATFORM$ applications to start a session with the TPM in order to establish a secure and anonymous communication channel with the S5 Tracker (through the TLS establishment functionality)

TLS Handshake Operation:

Execution Integrity: Integrity of the execution of the TPM command flow by monitoring the Trusted Software Stack (TSS); i.e., invocations made by what applications, command configurations, parameters passed, etc. It reflects the continuous monitoring and attestation of the low-level system and behavioral properties to be performed by the Control-Flow Attestation toolkit (more information to be included in D4.2). The control-flow path (CFP) associated with this specific session can be “binded” to the TPM session ID. This will provide the necessary guarantees that the tracing of this CFP is trusted as it has been signed by the TPM itself, thus, weakening the requirements on the trustworthiness of the BIOS/Kernel (in the case of micro-service oriented architecture - see Section 2.2.2).

Host Create TLS key: In the case that we also want confidentiality when it comes to establishing a secure communication Channel from the $PLATFORM$ to the S5 Activity Tracker Server, we need to be able to establish a symmetric encryption key (e.g. Diffie-Hellman).

Activity Tracker Response TLS key: Establishing a secure communication Channel from the S5 Activity Tracker Server to the $PLATFORM$.

TLS key: The established session key.

Encrypt Message: Symmetric encryption with the session key established.

Execution Integrity: Integrity of the TPM command flow (see previews one).

Sign handshake key: Signing of the established session key using the $DAAkey$ created by the TPM hosted by the $PLATFORM$.

TPM Hash: Compute a hash digest with different message lengths. The results of the hash will be used in a signing operation that uses a restricted signing key and the ticket returned by this command can indicate that the hash is safe to sign.

TPM Commit: Performs the first part of an ECC signing operation.

TPM Sign: Signing with the DAA key.

TPM Flush: Unload unnecessary loaded keys.

State Diagram of the S5 Tracker Analytic Engine Attestation & Verification (Figure 2.3 (c)):

Device & Execution Integrity: Integrity of the TPM command flow.

Start Session: Start a trial session in order to calculate the Digest Policy (based on the correct state of the installed applications) and then create the necessary TPM keys for signing the quoted PCR values representing the correct state of the S5 Tracker's Database.

Calculate DB Digest (PCR): Checks the PCR values representing the correct state of the tracker entity before making any transactions to the Database. Each time the Activity tracker receives new data from the users and before making the actual updates to the underlying DB, we check the PCR values so that we make sure that the server is at an allowed state.

TPM Quote: This is the quote that is provided to the host devices for attesting the integrity of the Activity Tracker database. The quote is basically containing the PCR values (i.e. the database digest).

2.3.3 Reference Scenario 3 — Device Management

The goal of this use case is to protect the communication keys that that are used between network devices and the Network Management System (NMS), and to prevent that user data is processed by compromised devices. Thus, the derived trust model captures the necessary requirements for the correct execution of mainly the TPM sealing and unsealing functionalities. Figure 2.4 captures the trusted relations (federations) that need to be established among the Network Management System (NMS), the trusted device itself ($PLATFORM$) and the TPM.

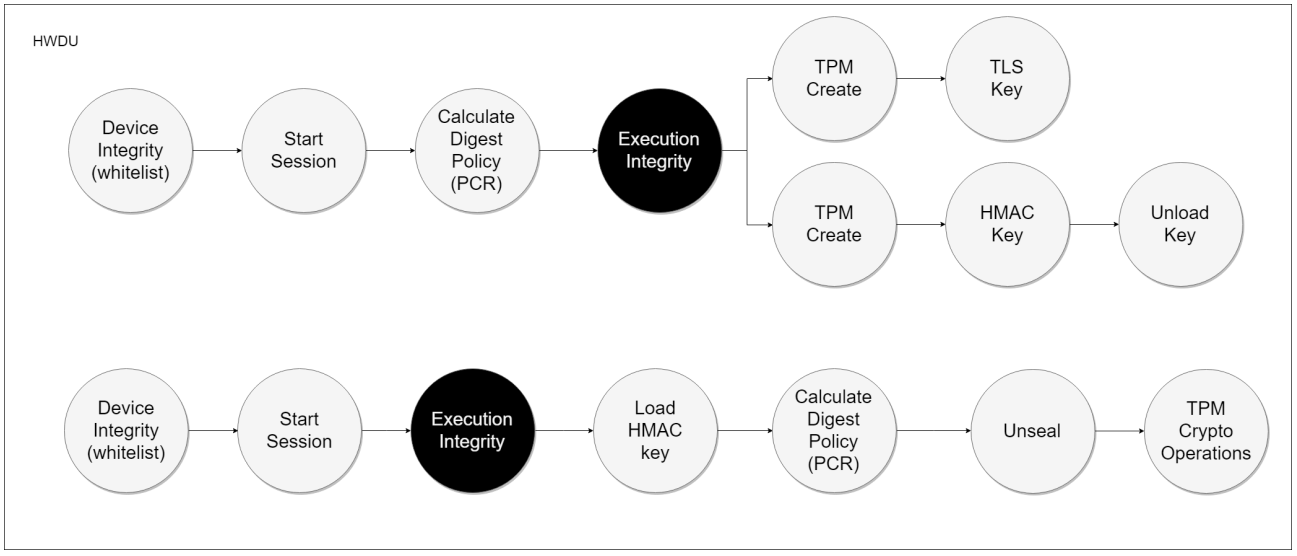

Figure 2.4: Device Management State Diagrams

The Sealing Operation (Figure 2.4 (a)):

Device Integrity (whitelist): Integrity calculations of the applications installed on the TPM-equipped $PLATFORM$. Applications need to be attested against a whitelist of allowed and trusted application instances. The NMS will be responsible for managing such whitelists in order to determine whether the keys generated by the TPM of a $PLATFORM$ are associated to a good software configuration. This property will be achieved by the TPM that can securely store the current system state in its Platform Configuration Registers (PCRs) and it will allow certain crypto operations to be performed with the TPM keys only if the current state is the same as when the key was created.

Start Session: Start a trial session in order to calculate the Digest Policy (based on the correct state of the installed applications) and then create the necessary TPM keys for connecting to the NMS.

Calculate Digest Policy (PCR): Checks the policy digest with the Device Integrity values of the application whitelist.

Execution Integrity: Integrity of the execution of the TPM command flow by monitoring the Trusted Software Stack (TSS); i.e., invocations made by what applications, command configurations, parameters passed, etc. It reflects the continuous monitoring and attestation of the low-level system and behavioural properties to be performed by the Control-Flow Attestation toolkit (more information to be included in D4.2). The main goal of this state is to protect against ROP attacks [55] that try to exploit the loaded software and TSS (and subsequently the created TLS key) without altering its state; thus, it will not be captures by the Device Integrity state.

TPM Create: Generate a new TLS key to establish a trusted channel.

TLS Key: TLS key is created and never leaves the TPM.

TPM Create: Generate a new random HMAC key.

HMAC Key: HMAC key is created.

Unload Key: HMAC key is not stored in the $PLATFORM$.

The Unsealing Operation (Figure 2.4 (b)):

Device Integrity (whitelist): Integrity calculations of the applications installed on the TPM-equipped $PLATFORM$. Applications need to be attested against a whitelist of allowed and trusted application instances. The NMS will be responsible for managing such whitelists in order to determine whether the keys generated by the TPM of a $PLATFORM$ are associated to a good software configuration. This property will be achieved by the TPM that can securely store the current system state in its Platform Configuration Registers (PCRs) and it will allow certain crypto operations to be performed with the TPM keys only if the current state is the same as when the key was created.

Start Session: Start an authorization (policy) session.

Execution Integrity: Integrity of the execution of the TPM command flow by monitoring the Trusted Software Stack (TSS); i.e., invocations made by what applications, command configurations, parameters passed, etc. It reflects the continuous monitoring and attestation of the low-level system and behavioural properties to be performed by the Control-Flow Attestation toolkit (more information to be included in D4.2). The main goal of this state is to protect against ROP attacks [55] that try to exploit the loaded software and TSS (and subsequently the created TLS key) without altering its state; thus, it will not be captures by the Device Integrity state.

Load HMAC Key: Load the HMAC key, since it is not loaded automatically

Calculate Digest Policy (PCR): Update the policy digest with the correct PCR values. The PCR values are related with the whitelist of the installed applications.

Unseal: TPM evaluates the session's policy digest and compares it against the key's expected authPolicy digest. If they match, the sealed data is returned to the user, otherwise, it fails with an error.

TPM Cryptographic Operations: Perform a cryptographic operation during the establishment of the trusted channel.

Chapter 3

Policy Modelling

In Chapter 2, we discuss strategies to describe trust in a TPM-backed system and its hierarchy of components. In particular, we describe a predicate-based language to describe trust, and identify high-level mechanisms—to be refined in WP4 and D4.2 FutureTPM Risk Assessment Framework – First Release—for the enforcement and monitoring of trust in TPM-backed platforms.

In this Chapter, we discuss strategies to describe and analyse the security of *high-level applications* that make use of the TPM. In this first model, we assume that all components involved in the application are trusted, but take care to choose a modelling approach that interacts well with the trust modelling techniques of Chapter 2, in order to later capture the low-level trust monitoring and enforcement mechanisms to be developed as part of WP4. We briefly outline our modelling approach, which uses the ability to perform TPM operations as the building blocks for describing trust domains at application level. Finally, we discuss some interesting applications, beyond those considered as part of the FutureTPM project, for which we believe the current TPM and policies are insufficient, and argue that a more principled design of trust computing components is needed.

3.1 Overview of the Modelling Approach

We aim to define a modelling language a framework that enables us to separate concerns into different layers without inducing too much cost at abstraction interfaces. In theory, this should enable us to provide application designers and developers who wish to make use of the TPM with the ability to quickly check that they use the TPM appropriately for their security functional requirements, while also allowing lower-level library developers—including the partners involved in WP4—to precisely reason about the guarantees their libraries provide in lower-level adversary models. Figure 3.1 presents the abstraction layers we are currently considering.

Application Security The highest level of abstraction considers a TPM-backed application's high-level security goals of confidentiality, integrity and availability. Although we do not consider these concerns in scope of the FutureTPM project, defining a policy framework that supports reasoning about application-level security may lead to more impactful results in the long term, ensuring that the TPM can be reasonably used to build "secure-by-design" systems. We also note that the high-level security requirements of the FutureTPM use cases were indeed taken into account, as part of D1.1 [24] and D4.1 [25] to tease out some of the TPM security requirements.

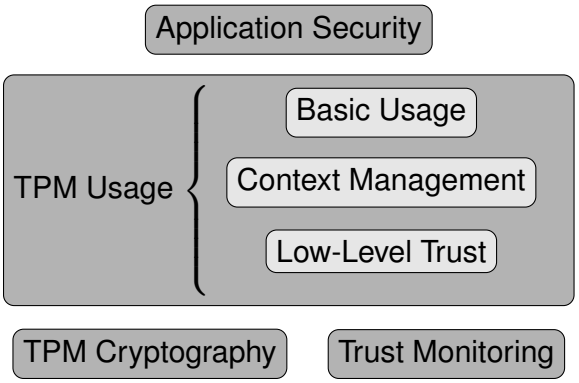

Figure 3.1: Policy Abstractions

TPM Usage The second most abstract layer considers usage policies on the TPM and its objects and capabilities. This layer is the main focus of the discussions in this Deliverable, and will serve as a three-way bridge between the Application Security layer, the Trust Monitoring layer, and the TPM Cryptography layer. Although we divide the TPM Usage layer in three—separating Context Management and Low-Level Trust from Basic Usage—this separation is purely made for the purpose of simplifying early discussions.

TPM Cryptography and Trust Monitoring Finally, the bottom most layers consider the TPM Cryptography—or how well-defined usage policies are in fact enforced by the TPM (or a network of TPMs); and the Trust Monitoring mechanisms to be developed in WP4. These two layers form the foundation of security in the FutureTPM: the former reduces security to hardness assumptions, in a well-defined trust model, while the latter monitors or prevents breaches of trust.

We now give an overview of the TPM Usage layer, focusing for now on its Basic Usage—which assumes that application components are always trustworthy (from the point of view of the application), and that the application performs its own context management as needed; and on the TPM Usage layer's interfaces with the TPM Cryptography and Trust Monitoring layers.

We note that, even though we consider here that all components of an application are always trusted by the application itself, we do consider scenarios where one or several TPMs interact with both a (single) trusted application and an adversary that has the same access to the TPM. In particular, we wish to be able to deem insecure an application that uses the TPM to manage a cryptographic key (or, more generally, any secret) as a protected blob, but interacts with the TPM without making use of encrypted sessions (thereby leaking the blob through the TPM's I/O bus). On the other hand, we wish—in this first instance—to be able to deem secure the same application which uses encrypted sessions correctly to ensure that the key could only be extracted through a deeper compromise (for example, the application itself, or the capture of a password through a keylogger installed on the platform).

In order to achieve this, while still articulating smoothly with the underlying layers, we consider a framework for access control that revolves around *trust domains* (which can be seen as distinct applications and interactions between them, but also as more atomic components such as particular TPMs, or platforms). In order to connect trust domains to the underlying cryptography, but also to enable the modelling of fine-grained trust for distributed applications, we use a party's ability to perform a particular sequence of TPM operations as the definition of a trust domain.

This aligns particularly well with several trusted computing concepts that are core to the TPM trust model, for example:

platform trust, which aims at proving that the platform (the TPM and its host, including its low-level software) are in a known configuration, is demonstrated by measuring platform components into the TPM's PCRs, and quoting them;

ownership, which represents authorization from the platform's owner, can be demonstrated through the ability to load the Storage Primary Seed (which is protected by owner authorization) into the TPM;

enhanced authorization policies, which allow combinations of authorization factors to be used as authorization to operate on protected objects, are constructed component by component through TPM calls; and

audit session-based authorization, which aims at measuring a subset of interactions with the TPM to be ued as authorization, locally or remotely, is immediately and flexibly captured as the fact that a sequence of TPM commands can be called within a session.

Importantly, this allows us to capture without too much difficulty the large flexibility afforded by the TPM's authorization mechanisms. This is useful at all 3 interfaces of the TPM Usage layer:

- This ensures that unseen application scenarios can be analysed without modifying the policy framework;
- This provides the most fine-grained way of describing the current TPM state for use in the TPM Cryptography layer, where security can only be guaranteed when the state follows certain invariants (for example, that keys used for session encryption are TPM-resident, or that particular keys were never unsealed outside a session); and
- This coincides neatly with the mechanisms being developed in WP4 for Trust Monitoring, which rely on measuring interactions with the TPM. Captured interaction traces can then be analysed for membership to the set of interaction traces under which security has been proved. At the extreme, and provided sufficient performance can be obtained, it may even be possible to prevent interactions with the TPM that would lead the system to a state under which security can no longer be guaranteed.

A complete formalization of this framework is progressing along with the modelling of the TPM's cryptographic systems, which we discuss briefly in Chapter 4.

We note, however, that our focus is no longer on developing models and proofs of security for the TPM as currently specified, although that remains an objective of the project. However, developing strong foundations to support the sound and principled development of new specifications for trusted computing is more promising in the long term than analysing the existing–organically grown—designs.

Indeed, we discuss in Appendix A two use cases that, similarly to extensions of the FutureTPM's Activity Tracking use case identified in D1.1 [24] (also see Figure 2.2, rely on cloud-side TPMs to protect the privacy of user data.

3.2 Identified Limitations in Current HSA Technology

As discussed in Appendix A and argued, among others, by van Dijk and Juels [64], access to an HSA is necessary for the full implementation of privacy guarantees in untrusted clouds. Developing such solutions remains an active area of research [26, 11, 46], especially in settings where data is shared among users whose relationships should remain private.

HSAs generally fall within one of two categories: fixed-API HSAs, and programmable HSAs. TPMs fall into the former category: a tamper-proof device with protected storage where the only possible interactions with the "outer world" is through a set of pre-specified commands. Intel SGX, on the other hand, can be seen as a programmable HSA, as it allows (almost) arbitrary programs to establish secure enclaves on general-purpose CPUs.

This flexibility, however, comes at a price: the resource-sharing entailed by having enclaves and general-purpose code running on the same hardware leads to rather vague security claims, and devastating attacks through side-channels [13], some of which can be triggered from very high-level applications. Discrete programmable HSAs (or Hardware Security Modules) exist, but are often too costly to be deployed in all but the highest-assurance scenarios.

On the other hand, while fixed-API HSAs are not immune to attacks, attacks against them are naturally less impactful. Indeed, attacks against TPM specifications [19] or specific TPM implementations [35] usually assume specific and hard-to-obtain adversary capabilities, such as physical access to power management, which makes them more difficult to exploit. However, while the rigidity of fixed-API offers significantly more security than its programmable counterpart, it also poses severe constraints on its practicality for implementing arbitrary protocols, as we have discussed in the sections above.

We summarize here the main limitations and challenges for current fixed-API HSAs (such as TPM 2.0) that prevent them from more fully supporting privacy-friendly implementations of communication protocols.

Lack of support for Trusted Execution. Technologies like Intel Trusted eXecution Technology (TXT) exist. In conjunction with a TPM, these enable the execution of arbitrary trusted code outside the enclave. Unfortunately this technology does not provide the full set of security properties we require here [68].

Limited internal resources. TPM devices have very limited internal memory. This can be overcome by allowing the TPM to store information in the cloud memory using its protected storage hierarchy. However, any algorithm or functionality that relies on this assumption must necessarily ensure an oblivious access of such memory. This also requires to minimize the amount of information that the TPM has to process and communicate. Some approaches to solve this issue might require, for example, several calls to a single stateful method. For example, in order to obtain a random, uniform permutation, a naive approach needs N/M calls, where N is the number of items to be permuted, and M is the number of items the TPM can store in memory.

Limited throughput. As resource-constrained devices, TPMs are slow, especially when dealing with public-key operations. This requires efforts to minimize the usage of cryptography within the TPM, and relying on collaboration from the host (here the cloud) cloud to execute certain operations that do not compromise the privacy requirements (for example, by using homomorphic properties of some supported schemes). Cryptography operations should be delegated to the cloud where possible, minimizing (especially) the usage of public key operations inside the TPM.

Accountability of external operations. While the TPM provides an audit session mechanism which allows a platform to attest to a sequence of commands being executed in order (with their inputs and outputs) it is still unclear if this can be extended to capture also sequences of commands that implement only part of the functionality, supported by computations performed on the host.

These lacks in the TPM's functionalities are significant barriers to its adoption in privacy applications. We will thus consider minimal extensions to the TPM commands, in light of the use cases discussed in Appendix A, but also those put forward as part of WP6, some of which include extensions for which the features discussed above may be needed. We note, in particular, that minimal trusted execution support could be used, in conjunction with audit sessions—seen as providing an immutable append-only record of interactions between TPM and host, to construct functionalities similar to those provided by state-channel networks [28].

Chapter 4

Cryptography for the TPM

In this Chapter, we review recent contributions by consortium members and collaborators in the domain of provable security for TPM mechanisms. In developing these contributions, we follow both bottom-up and top-down approaches.

- Bottom-up contributions—here a proof of security, and the underlying proof techniques, for a lattice-based DAA protocol—contribute to a better understanding of the building blocks that are put together in the TPM, and of their security properties.
- Top-down contributions—here a discussion on the limitations of extant mechanisms for bootstrapping trust in an unknown TPM—can be used to guide an exploration of the ways in which the TPM's low-level cryptographic mechanisms are combined to implement high-level security requirements such as secrecy, integrity or—in this case—authentication.

Combining both approaches therefore allows us to gain a better understanding of the way in which the TPM functionalities are constructed, at the same time as we gain a sufficient understanding of the security guarantees their construction is meant to guarantee. However, the top-down investigation is very closely linked to a high-level understanding of the TPM, both as a self-contained whole and as a component of complex secure systems.

4.1 A Provably-Secure Lattice Based Direct Anonymous Attestation Scheme

D2.1 [21] describes choices for QR primitives, and discusses potential design choices for lattice-based DAA, based on informal security and performance arguments. In this section, we describe a security proof for one of these proposals. It makes use of an honest-TPM variant of the UC model of Camenisch, Drijvers and Lehmann [17] in a QR0 setting. (See D2.1 [21] for a description of Quantum-Resistance levels.)

Figure 4.1 gives an overview of DAA parties and usage. In general, a DAA scheme consists of an issuer, a set of signers and a set of verifiers. The issuer creates a DAA membership credential for each signer. In practice, a DAA credential corresponds to a signature of the signer's identifier produced by the issuer. A DAA signer, or platform, is composed of a Host (usually a general purpose computing platform), and a TPM. The primary role of the DAA scheme is to prove, or attest to, the fact that a platform belongs to the DAA community, composed of platforms for which the issuer has issued a DAA credential, without revealing which platform it is. In addition, DAA

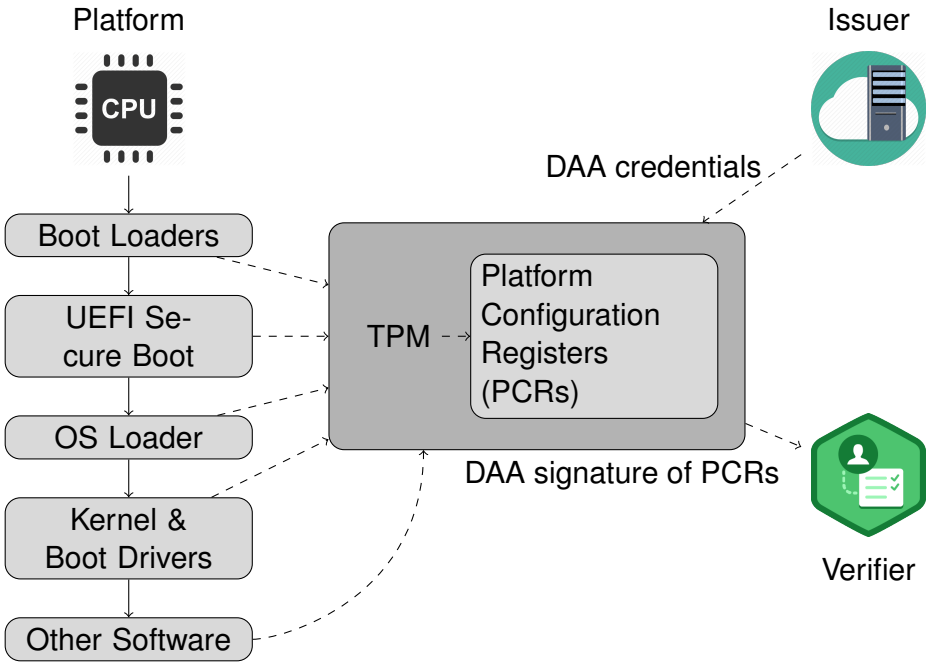

Figure 4.1: Generic TPM-based anonymous attestation. As software is loaded, the TPM builds a representation of the platform state. When access is requested to a resource, a DAA signature attesting to platform and TPM state is sent to the resource

can be used to attest to the state of the platform, including the version of software running on the Host, and internal TPM state.

The DAA signature includes a zero-knowledge proof-of-knowledge, which is used to convince the verifier that the signer possesses a valid membership credential, but without the verifier learning anything else about the identity of the signer. This mechanism provides both *unforgeability* and *anonymity*.

In contrast to other privacy-preserving constructs, like group signatures [27, 42, 44], a DAA scheme is not *traceable*. In fact, even when the DAA issuer also plays the role of a verifier, the issuer does not obtain more information from a given signature than any arbitrary verifier. However, to prevent a malicious signer from abusing anonymity, DAA provides two alternative properties to traceability: *rogue signer detection* and *user-controlled linkability*.

Firstly, it is possible to detect rogue signers: given a signer's private key, it is possible for anyone to check whether a particular DAA signature was created by this signer. Therefore, if a private key is somehow extracted from a TPM and leaked, it can be efficiently revoked and all past signatures invalidated. In addition, the scheme supports user-controlled linkability: two DAA signatures created by the same signer may or may not be linked from a verifier's point of view. The linkability of DAA signatures is controlled by an input parameter called the *basename*. If a signer uses the same basename in two signatures, they are linked; otherwise they are not.

4.1.1 Results

El Kassem et al. [39] propose a new scheme for DAA, that relies on lattice operations. They prove their scheme secure in the Universally Composable security model developed by Camenisch, Drijvers and Lehmann [17], reducing its security to standard lattice problems. We do not detail the

construction and proof in detail here, but include overviews in Appendix B, referring the interested reader to El Kassem et al. [39] for full detail.

4.2 Towards Mechanisms for Bootstrapping Trust with Privacy

Past and recent applications of formal analysis techniques to some of the TPM's mechanisms, including DAA [8, 66] and encrypted sessions [61], have revealed that *using* the TPM securely requires special care, beyond what would be expected from a security-oriented device.

We will now review these issues specifically, demonstrating that they are in fact symptoms of a singular and deep design problem. We then discuss potential solutions for these symptoms, including in light of Parno's early wrk on the topic of bootstrapping trust [51], and identify ways in which these consierations can be included in the modelling and analysis work.

4.2.1 DAA Authentication Attacks

As discussed in Section 4.1, the TPM's Direct Anonymous Attestation protocol includes a *Join* phase, during which a platform obtains credentials from an issuer after demonstrating that it belongs to that issuer's network. As captured in the security model, and discussed in the relevant standards, the security of the protocol assumes that the channel between the platform's TPM and the issuer is authenticated (that is, the issuer can trust that it is interacting with the specific TPM it believes it is interacting with).

Backes et al. [8] use ProVerif to highlight the need for strong authentication on this channel for the TPM1.2 version of DAA, and demonstrate an attack through which a rogue TPM can convince an issuer to issue credentials for a third party without that third party's participation. Although such credentials cannot, then, be used to produce valid signatures, a strict issuer may then refuse to issue valid credentials to that third party TPM's platform, which would constitute a Denial-of-Service. Although newer standards propose additional mechanisms to establish this authenticated channel [1], Whitefield et al. [66] demonstrate, using Tamarin, that these mechanisms are insufficient when issuers can be involved in concurrent *Join* phases.

Both Backes et al. [8] and Whitefield et al. [66] propose and formally verify protocol fixes, that simply involve binding *Join* sessions to TPM secrets the issuer can verify, the TPMs' *endorsement keys*, which are pre-loaded into all TPMs, and for which manufacturers provide certificates. If the fix is simple, the existence of the problem highlights a deeper issue of bootstrapping trust in a privacy-preserving way: the goal of the DAA protocol is to enable platforms to produce signatures that only reveal the involvement of a recognised TPM, without revealing any more about that TPM's identity. Any use of a TPM secret in the protocol, although it contributes to the protocol's security goals, may put at risk its privacy goal, and needs to be carefully reviewed in light of other uses of that TPM secret.

4.2.2 A Hierarchy Authentication Attack

The weakness of authentication in DAA's *Join* discussed above is only known to lead to a Denial-of-Service. However, Shao, Qin and Feng [61] identify a weakness in the TPM's session mechanism, which can lead to the *authValue* of protected objects becoming known to an adversary

even when they are meant to be protected by an encrypted session. This allows the adversary to load and use these objects as if authorized to do so, regardless of intended security. We now describe the weakness and the countermeasure proposed by Shao, Qin and Feng [61]. We note before giving details that the weakness is in the trust model, rather than the cryptography, and that the countermeasure is based on constraining slightly how the TPM is used, rather than on deep changes to the specification.

Assume a freshly cleared (or acquired) TPM, on which the Primary Seed's *authValue* and *authPolicy* are clear, allowing anyone with access to the TPM to load it into a TPM handle and make use of it. Shao, Qin and Feng [61] consider a scenario where a caller attempts to create a protected object, whose use would be protected by a password (using the HMAC authorization policy), and show that it is impossible to do so without an adversary learning the *authValue* for the freshly created object. Indeed, the TPM provides both symmetric and asymmetric options to protect the secrecy of command parameters and outputs such as the new *authValue*, but is not equipped with any mechanism (beyond that discussed in the countermeasure, which we argue here is inadequate) for the caller to verify that interactions are indeed occuring with the target TPM. In the case of symmetric mechanisms, a freshly cleared TPM is not equipped with any secrets that can be shared with the caller. Creating such a secret on the TPM and outputting it to the caller would reveal it to an adversary with simple eavesdropping access to the TPM bus. Conversely, having the caller create a secret and load it into the TPM would be impossible without the adversary learning it through the command parameters. In the case of asymmetric mechanisms, any adversary could intercept communication betwen the TPM and caller, and present their own public key instead of the TPM's, allowing them to decrypt sessions.

As in the case of DAA, the problem here is in establishing an authenticated channel between the TPM and the caller. This is sufficient to bootstrap trust and support the desired use case, and the futher construction of complex and secure object hierarchies. Shao, Qin and Feng [61] suggest using the endorsement key, and its certificate to establish this trusted channel using the existing session mechanism.

4.2.3 Bootstrapping Trust

Note that both issues discussed above are in fact instances of a deeper problem: that of bootstrapping trust in an unknown TPM, which was already discussed by Parno [51]. Indeed, in the case of the DAA attacks, the Issuer does not know *a priori* which TPM she is interacting with, unless this interaction is carried out over an authenticated channel which binds to an identity known to the Issuer. Similarly, encrypted sessions can be circumvented when the session is established between an application and a TPM it has never before interacted with, and therefore does not share any knowledge with.

In both cases, using the endorsement key in naive ways simply ensures that a TPM's endorsement key was involved in the protocol. (And in fact, DAA does already guarantee this without requiring an authenticated channel.) This is sufficient if one assumes that no TPM endorsement key can ever be compromised. However, in both cases, a single TPM becoming compromised leads to all trust being lost until it is detected and added to revocation lists.

In the case of DAA, the solutions proposed by Backes et al. [8] and Whitefield et al. [66], which consist in binding the TPM's specific endorsement key to the session, is sufficient to convince the issuer that:

1. the other party is a member of its group (based on a certificate over an endorsement key); and

2. the other party is indeed in possession of the private key corresponding to the certified public key.

This is a relatively weak form of authentication, which does not require a specific TPM to identify itself and be authenticated.

In the case of the hierarchy authentication attacks by Shao, Qin and Feng [61], such a weak form of authentication is insufficient: the caller needs to be convinced it is interacting with a specific TPM, namely the one installed in its platform. For this purpose, proving simple knowledge of a specific private key whose public counterpart has been certified is insufficient when other certified keys are compromised: an adversary could simply present a different certificate to the caller and retrieve all session data, including the *authValue* for the new object.

Bootstrapping Trust in a Specific Local TPM

A naive solution to this problem would be to include, in the manufacturer certificates, information identifying the specific TPM. This could be verified—perhaps by physical inspection—by the caller or its user, even if the rest of the platform is compromised. This approach is not suitable for two main reasons:

1. The relation between a platform and its endorsement key is not meant to be public; and
2. In practice, endorsement certificates are given only to issuers, instead of being publicly broadcast.

These concerns also extend to existing security proofs. Camenisch, Drijvers and Lehmann [17] model the TCG scenario where only the issuer is given the TPM public keys, and properties proved in their model (or its variations) may no longer hold when public endorsement keys and their relations to specific TPMs are broadcast. The relation between a platform and an endorsement key is *not* communicated to the issuer in the model. Modifying the TPM in such a way as to require certificates, including links between endorsement keys and physical TPMs, to be publicly broadcast will therefore require new models to be devised, and new proofs developed.

In addition, further concerns arise from the repeated use of the endorsement key—a long-lived asymmetric encryption key—in normal TPM usage. Indeed, the TPM's endorsement key is a simple asymmetric decryption key, and using it as discussed in both scenarios above would put significant pressure on its security, by essentially providing a relatively weak adversary with a decryption oracle despite the fact that the endorsement key *cannot* be renewed. Further, repeated use of the endorsement key raises privacy concerns, as noted in the specification: "[t]he uniqueness of an [Endorsement Key] and its cryptographic verifiability raises the issue of whether direct use of that identity could result in aggregation of activity logs. [...] TCG encourages [...] restrictions on the use of the [Endorsement Key]." [34, Part 1-9.4.3.4].

In practice, mechanisms do exist that bind a TPM's endorsement key to the TPM's host (for example, Platform Certificates [34, Part 1-9.5.3.1], whose implementation is not yet defined for TPM 2.0 [33]). These may be used, when available, to bootstrap trust for local interactions. Models and proofs for security and privacy currently do not take the existence of such certificates into account, and it is unclear what security requirements should be imposed on them. Our modelling will aim to clarify this, and to incorporate discussions of some of the solutions proposed by Parno [51], allowing us to also consider their privacy implications.

In addition, it is still important to ensure that the endorsement key itself does not have to be used repeatedly everytime the TPM is turned on. We will define, analyse and evaluate a "best practice"

approach to using bootstrapped sessions to construct chains of trust that provide more resilience with only limited loss of performance.

Chapter 5

Conclusion

In this report, we define a framework for trust modelling in complex applications whose security and privacy requirements are achieved—at least partially—using TPMs.

We also give an overview of a compatible policy and usage modelling framework that could serve as a bridge between the trust model, the analysis of applications security at a high-level, and the analysis of low-level details of the TPM specification. This includes the specification of TPM cryptography, but also the TPM's cryptographic realization of authorization and trust mechanisms.

About the former, we report some progress towards the development of *provably secure* DAA schemes based on quantum hardness assumptions. We use recent analyses of the interactions between the TPM's session and authorization mechanisms, backed by older analyses of the TPM's mechanisms for trust—including the DAA protocol—and of some privacy-oriented use cases to argue that the TPM as currently specified provides insufficient functionality to both bootstrap trust in a trustless software environment *and* support privacy.

We view these issues with the current specification as opportunities to develop sound and solid theoretical foundations for Trusted Computing, that could support the principled design of the next specification, and identify possible directions for these models to develop, in line with the choices made for trust and usage modelling. All still points to the need for the TPM to be modelled and analysed as a whole, rather than as the sum of independent parts, in order to properly capture possible interactions between different uses of the same cryptographic material, the prime example of which is the use of Endorsement Keys for demonstrating group membership anonymously in DAA while also having to rely on them for individual authentication of a specific TPM in other settings.

Chapter 6

List of Abbreviations

Abbreviation	Translation
CFPA	Control-Flow Properties Attestation
CMS	Conference Management System
DAA	Direct Anonymous Attestation
HMAC	Hash based Message Authentication Code
HSA	Hardware Security Anchor
ISIS	Inhomogeneous Short Integer Solution
LPC	Low Pin Count
LWE	Learning With Error
NMS	Network Management System
PCR	Platform Configuration Register
PQ	Post Quantum
PRNG	Pseudo-Random Number Generator
QR	Quantum Resistant
SIS	Short Integer Solution
TPM	Trusted Platform Module
TSS	TPM Software Stack
UC	Universal Composability
WP	Work Package

References

[1] ISO/IEC 20008-2:2013. Information technology – Security techniques – Anonymous digital signatures – Part 2: Mechanisms using a group public key. Standard ISO/IEC 20008-2:2013, International Organization for Standardization, 2013. Last revised 2019.

[2] Tigist Abera, N Asokan, Lucas Davi, Jan-Erik Ekberg, Thomas Nyman, Andrew Paverd, Ahmad-Reza Sadeghi, and Gene Tsudik. C-FLAT: control-flow attestation for embedded systems software. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 743–754. ACM, 2016.

[3] Alessandro Aldini. A formal framework for modeling trust and reputation in collective adaptive systems. *arXiv preprint arXiv:1607.02232*, 2016.

[4] Denis Andzakovic. Extracting bitlocker keys from a TPM. https://pulsesecurity.co.nz/articles/TPM-sniffing, 2019.

[5] Myrto Arapinis, Sergiu Bursuc, and Mark Ryan. Privacy supporting cloud computing: Confichair, a case study. In *Principles of Security and Trust (POST*, volume 7215 of *LNCS*, pages 89–108, Tallinn, Estonia, March 2012. Springer.

[6] Myrto Arapinis, Sergiu Bursuc, and Mark Ryan. Privacy-supporting cloud computing by in-browser key translation. *Journal of Computer Security*, 21(6):847–880, December 2013.

[7] N Asokan, Ferdinand Brasser, Ahmad Ibrahim, Ahmad-Reza Sadeghi, Matthias Schunter, Gene Tsudik, and Christian Wachsmann. Seda: Scalable embedded device attestation. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 964–975. ACM, 2015.

[8] Michael Backes, Matteo Maffei, and Dominique Unruh. Zero-knowledge in the applied pi-calculus and automated verification of the direct anonymous attestation protocol. In *2008 IEEE Symposium on Security and Privacy (S&P 2008), 18-21 May 2008, Oakland, California, USA*, pages 202–215, 2008.

[9] Carsten Baum, Ivan Damgård, Vadim Lyubashevsky, Sabine Oechsner, and Chris Peikert. Efficient commitments and zero-knowledge protocols from ring-sis with applications to lattice-based threshold cryptosystems. Cryptology ePrint Archive, Report 2016/997, 2016. https://eprint.iacr.org/2016/997.

[10] Josh Berdine, Byron Cook, and Samin Ishtiaq. SLAyer: Memory safety for systems-level code. In *International Conference on Computer Aided Verification*, pages 178–183. Springer, 2011.

[11] Andrea Bittau, Bernhard Seefeld, Úlfar Erlingsson, Petros Maniatis, Ilya Mironov, Ananth Raghunathan, David Lie, Mitch Rudominer, Ushasree Kode, and Julien Tinnes. Prochlo: Strong privacy for analytics in the crowd. In *Symposium on Operating Systems Principles (SOSP)*, pages 441–459, Shanghai, China, October 2017. ACM.

[12] Xavier Boyen. Lattice mixing and vanishing trapdoors: A framework for fully secure short signatures and more. In *Public Key Cryptography – PKC 2010*, pages 499–517, 2010.

[13] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F. Wenisch, Yuval Yarom, and Raoul Strackx. Foreshadow: Extracting the keys to the Intel SGX kingdom with transient out-of-order execution. In *USENIX Conference on Security Symposium*, pages 991–1008, Baltimore, MD, August 2018. USENIX.

[14] Michael Burrows, Stephen N Freund, and Janet L Wiener. Run-time type checking for binary programs. In *International Conference on Compiler Construction*, pages 90–105. Springer, 2003.

[15] Cristiano Calcagno and Dino Distefano. Infer: An automatic program verifier for memory safety of C programs. In *NASA Formal Methods Symposium*, pages 459–465. Springer, 2011.

[16] Jan Camenisch, Liqun Chen, Manu Drijvers, Anja Lehmann, David Novick, and Rainer Urian. One TPM to bind them all: Fixing TPM 2.0 for provably secure anonymous attestation. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 901–920, 2017.

[17] Jan Camenisch, Manu Drijvers, and Anja Lehmann. Universally composable direct anonymous attestation. In *Public-Key Cryptography – PKC 2016*, pages 234–264, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[18] M. Carbone, M. Nielsen, and V. Sassone. A formal model for trust in dynamic networks. In *First International Conference onSoftware Engineering and Formal Methods, 2003.Proceedings.*, pages 54–61, Sep. 2003.

[19] Liqun Chen and Mark Ryan. Attack, solution and verification for shared authorisation data in TCG TPM. In *International Workshop on Formal Aspects in Security and Trust*, volume 5983 of *LNCS*, pages 201–216, Eindhoven, The Netherlands, November 2009. Springer.

[20] Shuo Chen, Jun Xu, Emre C. Sezer, Prachi Gauriar, and Ravishankar K. Iyer. Non-control-data attacks are realistic threats. In *Proceedings of the 14th Conference on USENIX Security Symposium - Volume 14*, SSYM'05, pages 12–12, 2005.

[21] The FutureTPM Consortium. First report on new qr cryptographic primitives. Deliverable D4.1, September 2018.

[22] The FutureTPM Consortium. First report on security models for the TPM. Deliverable D3.1, September 2018.

[23] The FutureTPM Consortium. FutureTPM reference architecture. Deliverable D1.2, October 2018.

[24] The FutureTPM Consortium. FutureTPM use case and system requirements. Deliverable D1.1, June 2018.

[25] The FutureTPM Consortium. Threat modelling & risk assessment methodology. Deliverable D4.1, February 2019.

[26] Hung Dang, Tien Tuan Anh Dinh, Ee-Chien Chang, and Beng Chin Ooi. Privacy-preserving computation with trusted computing via scramble-then-compute. In *Privacy Enhancing Technologies Symposium (PETS*, pages 21–38, Minneapolis, MN, July 2017.

[27] Rafaël del Pino, Vadim Lyubashevsky, and Gregor Seiler. Lattice-based group signatures and zero-knowledge proofs of automorphism stability. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 574–591, 2018.

[28] Stefan Dziembowski, Sebastian Faust, and Kristina Hostáková. General state channel networks. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 949–966, 2018.

[29] Anna Lisa Ferrara, Georg Fuchsbauer, Bin Liu, and Bogdan Warinschi. Policy privacy in cryptographic access control. In *IEEE 28th Computer Security Foundations Symposium, CSF 2015, Verona, Italy, 13-17 July, 2015*, pages 46–60, 2015.

[30] Anna Lisa Ferrara, Georg Fuchsbauer, and Bogdan Warinschi. Cryptographically enforced RBAC. In *2013 IEEE 26th Computer Security Foundations Symposium, New Orleans, LA, USA, June 26-28, 2013*, pages 115–129, 2013.

[31] Thanassis Giannetsos and Tassos Dimitriou. Spy-sense: Spyware tool for executing stealthy exploits against sensor networks. In *Proceedings of the 2Nd ACM Workshop on Hot Topics on Wireless Network Security and Privacy*, HotWiSec '13, pages 7–12, 2013.

[32] Thanassis Giannetsos, Tassos Dimitriou, Ioannis Krontiris, and Neeli R. Prasad. Arbitrary code injection through self-propagating worms in von neumann architecture devices. *Comput. J.*, 53(10):1576–1593, December 2010.

[33] Trusted Computing Group. TCG EK credential profile for TPM family 2.0; level 0, November 2014. Revision 14.

[34] Trusted Computing Group. Trusted platform module library, September 2016. Revision 1.38, as amended by Errata Version 1.4.

[35] Seunghun Han, Wook Shin, Jun-Hyeok Park, and HyoungChun Kim. A bad dream: Subverting trusted platform module while you are sleeping. In *USENIX Conference on Security Symposium*, pages 1229–1246, Baltimore, MD, August 2018. USENIX.

[36] Munirul M Haque and Sheikh I Ahamed. An omnipresent formal trust model (FTM) for pervasive computing environment. In *31st Annual International Computer Software and Applications Conference (COMPSAC 2007)*, volume 1, pages 49–56. IEEE, 2007.

[37] Jeffrey Hoffstein, Jill Pipher, and J.H. Silverman. *An Introduction to Mathematical Cryptography*. Springer Publishing Company, Incorporated, Springer-Verlag New York, 1 edition, 2008.

[38] Ahmad Ibrahim, Ahmad-Reza Sadeghi, Gene Tsudik, and Shaza Zeitouni. Darpa: Device attestation resilient to physical attacks. In *Proceedings of the 9th ACM Conference on Security & Privacy in Wireless and Mobile Networks*, pages 171–182. ACM, 2016.

[39] Nada EL Kassem, Liqun Chen, Rachid El Bansarkhani, Ali El Kaafarani, Jan Camenisch, Patrick Hough, Paulo Martins, and Leonel Sousa. More efficient, provably-secure direct anonymous attestation from lattices. *Future Generation Computer Systems*, 99:425–458, 2019.

[40] Nikos Koutroumpouchos, Christoforos Ntantogian, Sofia-Anna Menesidou, Kaitai Liang, Panagiotis Gouvas, Christos Xenakis, and Thanassis Giannetsos. Secure edge computing with lightweight control-flow property-based attestation. In *International Conference Network Softwarization (NetSoft)*, 2019.

[41] Russell W. F. Lai, Henry K. F. Cheung, and Sherman S. M. Chow. Trapdoors for ideal lattices with applications. In *Information Security and Cryptology*, pages 239–256, 2015.

[42] Benoît Libert, San Ling, Fabrice Mouhartem, Khoa Nguyen, and Huaxiong Wang. Signature schemes with efficient protocols and dynamic group signatures from lattice assumptions. In *Advances in Cryptology – ASIACRYPT 2016*, pages 373–403, 2016.

[43] San Ling, Khoa Nguyen, Damien Stehlé, and Huaxiong Wang. Improved zero-knowledge proofs of knowledge for the ISIS problem, and applications. In *Public-Key Cryptography – PKC 2013*, pages 107–124, 2013.

[44] San Ling, Khoa Nguyen, and Huaxiong Wang. Group signatures from lattices: Simpler, tighter, shorter, ring-based. In *Public-Key Cryptography – PKC 2015*, pages 427–449, 2015.

[45] Michael E Locasto, Steven J Greenwald, and Sergey Bratus. Trust distribution diagrams: Theory and applications. In *Proceedings of the 4th Layered Assurance Workshop (LAW 2010), Austin, TX*, 2010.

[46] Sinisa Matetic, Moritz Schneider, Andrew Miller, Ari Juels, and Srdjan Capkun. DelegaTEE: brokered delegation using trusted execution environments. In *USENIX Conference on Security Symposium*, pages 1387–1403, Baltimore, MD, August 2018. USENIX.

[47] Lawson Nate. TPM hardware attacks. https://rdist.root.org/2007/07/16/tpm-hardware-attacks/, 2007.

[48] Lawson Nate. TPM hardware attacks (part 2). https://rdist.root.org/2007/07/17/tpm-hardware-attacks-part-2/, 2007.

[49] Matus Nemec, Marek Sys, Petr Svenda, Dusan Klinec, and Vashek Matyas. The return of coppersmith's attack: Practical factorization of widely used rsa moduli. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1631–1648. ACM, 2017.

[50] Nicholas Nethercote and Julian Seward. Valgrind: a framework for heavyweight dynamic binary instrumentation. In *ACM Sigplan notices*, volume 42, pages 89–100. ACM, 2007.

[51] Bryan Parno. Bootstrapping trust in a "trusted" platform. In *Proceedings of the 3rd Conference on Hot Topics in Security*, HOTSEC'08, pages 9:1–9:6, Berkeley, CA, USA, 2008. USENIX Association.

[52] Chris Peikert. A decade of lattice cryptography. *Foundations and Trends® in Theoretical Computer Science*, 10(4):283–424, 2016.

[53] Guillaume Poupard and Jacques Stern. Short proofs of knowledge for factoring. In *Public Key Cryptography*, pages 147–166, 2000.

[54] Oded Regev. The learning with errors problem (invited survey).

[55] Ryan Roemer, Erik Buchanan, Hovav Shacham, and Stefan Savage. Return-oriented programming: Systems, languages, and applications. *ACM Trans. Inf. Syst. Secur.*, 15(1):2:1–2:34, March 2012.

[56] Sasha Romanosky, Rahul Telang, and Alessandro Acquisti. Do data breach disclosure laws reduce identity theft? *Journal of policy analysis and management*, 30(2):256–286, March 2011.

[57] Grigore Roşu, Wolfram Schulte, and Traian Florin Şerbănuţă. Runtime verification of c memory safety. In *International Workshop on Runtime Verification*, pages 132–151. Springer, 2009.

[58] Mark D. Ryan. Cloud computing privacy concerns on our doorstep. *Communications of the ACM*, 54(1):36–38, January 2011.

[59] Lalitha Sankar, S. Raj Rajagopalan, and H. Vincent Poor. Utility-privacy tradeoffs in databases: An information-theoretic approach. *IEEE Transactions on Information Theory*, 8(6):838–852, March 2013.

[60] Ravi Sen and Sharad Borle. Estimating the contextual risk of data breach: An empirical approach. *Journal of Management Information Systems*, 32(2):314–341, August 2015.

[61] Jianxiong Shao, Yu Qin, and Dengguo Feng. Formal analysis of HMAC authorisation in the TPM 2.0 specification. *IET Information Security*, 12(2):133–140, 2018.

[62] Sergei Skorobogatov. *Physical Attacks and Tamper Resistance*, pages 143–173. Springer New York, 2012.

[63] Giannetsos Thanassis, Dimitriou Tassos, and Prasad Neeli R. Weaponizing wireless networks: An attack tool for launching attacks against sensor networks. In *Black Hat Europe 2010*, Barcelona, Spain, April 12-15, 2010.

[64] Marten van Dijk and Ari Juels. On the impossibility of cryptography alone for privacy-preserving cloud computing. In *USENIX conference on Hot topics in security (HotSec)*, pages 1–8, Washington, DC, August 2010. USENIX.

[65] Cong Wang, Qian Wang, Kui Ren, and Wenjing Lou. Privacy-preserving public auditing for data storage security in cloud computing. In *IEEE INFOCOM*, pages 1–9, San Diego, CA, March 2010. IEEE.

[66] Jorden Whitefield, Liqun Chen, Ralf Sasse, Steve Schneider, Helen Treharne, and Steve Wesemeyer. A symbolic analysis of ECC-based direct anonymous attestation. In *2019 IEEE European Symposium on Security and Privacy*, 2019. To appear.

[67] Johannes Winter. Trusted computing and local hardware attacks. Master's thesis, Graz University of Technology, 5 2014.

[68] Rafal Wojtczuk and Joanna Rutkowska. Attacking intel trusted execution technology, 2009. Accessed 31 May, 2019.

Appendix A

Hardware Security Anchors in Malicious Cloud Scenarios

Cloud services support the outsourcing of data processing on remote servers. This processing can take a variety of forms, from storing private files for archival to forwarding messages to communicating parties, to executing complex data mining algorithms on the outsourced data. In those scenarios, the cloud provider clearly needs access to the user's data in some form. This fact raises privacy concerns.

Indeed, users' data can be leaked by a malicious insider, an outsider breach, or the cloud provider colluding with a third party. Even though there are now legal and regulatory means to mitigate these threats, these do not *prevent* the breach, and are thus insufficient to guarantee the required levels of privacy and confidentiality for private datasets [56, 60]. We thus consider the problem of ensuring that the user can trust that the cloud provider processes the data only in the way it indicated [58] without considering issues of incentives. In some scenarios, the user may even wish to not trust the cloud provider at all, but still requires the ability to outsource some functionality over their data. Solutions at both ends of this spectrum have been proposed: when the cloud provider is completely trusted, the user can still consider privacy/utility trade-offs [59] to limit exposure; when the cloud provider is completely untrusted, it is possible to achieve almost arbitrary functionality without loss of privacy through the use of advanced cryptographic mechanisms (such as, for example, fully homomorphic encryption or multi-party computation protocols) [65]. In this latter setting, van Dijk and Juels [64], among others point out the difficulties of enforcing privacy using cryptography along, and suggest the use of trusted hardware to make progress towards a feasible solution. We investigate the question of whether current trusted computing solutions (generically referred to as *Hardware Security Anchors* (HSAs)) are ready to support privacy in a generic cud-based multi-user scenario where the cloud provider is untrusted. We will assume that the cloud is an honest-but-curious adversary (that is, it follows the protocol but tries to learn as much information as they can).

We generically model this scenario as a "social network", and consider privacy concerns related to both the confidentiality of users' *data* (during transmission and processing, and at rest), and the visibility of *metadata* (for example, connections between users). Privacy properties where the adversary can access both data and activity (or metadata) are best captured as *unlinkability* properties. Consider, for example, the case of a conference management system. The sensitive data, in this case, includes the papers—submitted in confidence—and the reviews. The metadata would include, among other sensitive details, the connections between authors and reviewers, which would ideally only be known to the chair of the programme committee. Yet, we still wish the cloud provider to support the system's functionality (for example, by routing information to the

correct users) without learning any of the sensitive information.

In the remainder of this Chapter, we discuss possible directions for unlinkable conference management systems A.1 and a more generic social network A.2. We then discuss how current HSAs seem to fall short of the functionality required to fully support privacy in these scenarios (and, by extension, also in the FutureTPM's third use case).

A.1 Use Case: Private Conference Management System

We first consider the ConfiChair protocol [5], a privacy-preserving cloud-based conference management system that uses in-browser key translation techniques to protect the confidentiality of data. Concretely, the confidentiality properties attained by the protocol ensure that the cloud does not learn:

- the content of the submitted papers,
- the content of the submitted reviews, or
- the scores attributed to submitted papers.

Further, it also provides a strong unlinkability property, ensuring that the cloud does not learn the author-reviewer relation. The cloud server only manages sensitive data as encrypted blobs, and the chair's browser executes a critical obfuscation part on the data in order to achieve unlinkability. In terms of functionality, the protocol still enables the cloud provider to route information to the necessary chairs, reviewers and authors, to enforce access control, and optionally to perform statistics collection.

Overview of ConfiChair. There are some particularities of the ConfiChair protocol that play a key role in achieving its unlinkability property:

1. The protocol manages a relatively small social network;
2. The protocol is divided into different phases: the papers can be uploaded to the cloud (but not retrieved) during the submission phase. The privacy-preserving actions are triggered after this phase.
3. Users (reviewers) are allowed to download entire databases of encrypted papers or metadata.
4. Users have different roles. There is a "special user" (the chair) which plays a paramount role in the privacy-preserving feature.
5. The chair is in charge, not only of assigning reviewers to papers, but also of re-encrypting cryptographic material so that the cloud can perform the routing tasks correctly. The re-encryption is required to prevent the cloud from learning the links.

An important observation is that, while ConfiChair protects the author-reviewer link, it does not protect the "friends of friends" link. That is, the cloud knows what reviewers are reviewing the same paper, but not which one. We discuss this stronger unlinkability property in a more general scenario in Section A.2.

Re-encryption for privacy. We consider four different roles: the chair, the cloud provider, and two different types of regular users, namely authors and reviewers. The protocol critically relies on strict interactions between the chair and the cloud provider. Recall that the cloud is responsible for the correct and secure routing of information between the other three participants. In order for the ConfiChair protocol to achieve this goal, while ensuring the integrity and confidentiality of the transmitted data, the chair is burdened with more computational load, especially in the review phase.

More concretely, the chair is responsible for decrypting data with one key and encrypting it with another one while mixing and re-randomising the order of the encrypted items. During the review phase, the chair downloads from the cloud a database $\mathtt{DB}_{\mathtt{keys}}$. Each entry in this database is a pair of a symmetric key and a unique identifier for the paper, both encrypted under a conference public key $\mathtt{pub}(\mathtt{Conf})$. The chair, who uniquely possesses the corresponding conference private key $\mathtt{priv}(\mathtt{Conf})$ decrypts each entry, generates a new unique identifier (at random) which is used to assign papers to reviewers, and re-encrypts the entry under a symmetric key K_{conf} shared by all reviewers. The resulting ciphertexts are then mixed, forming a new database $\mathtt{DB}^{r}_{\mathtt{keys}}$, which is returned to the cloud. The cloud is responsible for access control related to conflicts of interest. Thus, the conference chair has to perform the following operations: a public key decryption, a symmetric encryption and a shuffle. These operations are performed on the chair's browser, adding a significant computational burden on the chair side. In addition, any failure of the browser, or the loss of connection during the process, would require the phase to be re-executed from the beginning. We refer the reader to the published version of ConfiChair [5] or its extended version [6] for a further discussion on the protocol.

HSAs for Privacy. Consider the critical privacy-preserving operation discussed above, whose interface can be abstracted in terms of inputs and outputs as:

$$\begin{aligned}&\text{Input: } (x_1, x_2 \ldots, x_r)\\ &\text{Output: } (f(x_{\sigma(1)}), f(x_{\sigma(2)}), \ldots, f(x_{\sigma(r)})),\end{aligned}$$

where $\sigma : \{1, 2 \ldots, r\} \to \{1, 2 \ldots, r\}$ is a (secret) permutation, and f is a function parametrized by the two keys ($\mathtt{priv}(\mathtt{Conf})$ and K_{conf}) above. The function f is in charge of decomposing its inputs, executing some checks, rearranging items and performing a decryption and an encryption (under some provided keys), obtaining a functionality to that of a reencryption mixnet, but with concrete details incompatible with common realizations. Further, reencryption mixnets only distribute trust, and require incentives to be aligned to deter collusion, which we aimed to avoid.

As such, we wish to consider the use of a minimal HSA—possible an extended-functionality version of a TPM—to be placed on the conference server (which runs the cloud provider's operations), to avoid having to involve the chair in the distribution of papers to reviewers. (Note that the allocation would still be performed by the chair.) It is difficult to expect such a bespoke functionality to be made available in a general-purpose fixed-API device such as the TPM. Further, we wish to avoid assuming fully-programmable HSAs: lightweight version such as Intel SGX provide only nebulous security guarantees, whereas dedicated fully-programmable Hardware Security Modules are often very costly. We also note that the use case's requirements—nd in particular that the permutation over the whole database of submissions be performed obliviously—places strong requirements on the HSA, which are unlikely to be met by any TPM.

We therefore separate the desired functionality into two smaller, but more generic functionalities, that would suffice to support this use case in practice:

- $\texttt{HSA.Apply}(\pi, x)$: This command applies a transformation described by π to each entry of a vector input $x = (x_1, x_2, \ldots, x_r)$. In the ConfiChair scenario, the input is the database $\texttt{DB}_{\texttt{keys}}$ and for each of the entries x_i, the function $\texttt{HSA.Apply}(\pi, x)$ derives the key associated to each author, verifies the validity of each key, generates a new random identifier that will be used to assign each paper to a reviewer and returns the encryption of these values. In other words, this function can be described as:

  $$(y_1, y_2, \ldots, y_r) \leftarrow \texttt{HSA.Apply}(\pi, (x_1, x_2, \ldots, x_r))$$

  The benefit of adding this function in the HSA API is that it can rely on the HSA's inherent trust to manage and protect encryption and decryption keys, and demonstrate to a remote caller that operations were performed correctly—perhaps through the use of standard and existing auditing mechanisms. Contrasting with general purpose trusted execution environments such as Intel's SGX, this functionality could use language-based mechanisms to limit the set of functionalities (such as π) that could be applied, preventing looping behaviours, for example. On the other hand, some features are required for our use case, and the language must allow:

  - Access to loaded HSA objects following their authorization policy;
  - Execution of elementary HSA operations such as encryption and decryption using authorized objects;
  - Simple value checks (such as equality checks);
  - The generation of random values; and
  - Conditional branching.

  Additionally, we also require that the output of `HSA.Apply` can also be protected as an HSA-protected object, ensuring it cannot be read or manipulated outside of the HSA it was created on.

- $\texttt{HSA.Shuffle}(y)$: As the name suggests, this function applies a mixing on an input vector $y = (y_1, y_2, \ldots, y_r)$. In other words:

  $$(y_{\sigma(1)}, y_{\sigma(2)}, \ldots, y_{\sigma(r)}) \leftarrow \texttt{HSA.Shuffle}(y_1, y_2, \ldots, y_r),$$

  where $\sigma : \{1, 2, \ldots, r\} \rightarrow \{1, 2, \ldots, r\}$ is a random permutation for the r entries y_i. Of course, this permutation command only makes sense if the input y is encrypted and only accessible to the HSA, or if the y_i are encrypted using a malleable scheme that allows re-randomization. We note that such a feature would also be useful in more general applications, where shuffling requires either partial trust (for example, through distribution and threshold operations in electronic voting) or complex and slow cryptography (as in online card games).

An interesting similarity arises here, which we do not reflect on further in this deliverable, with the standard data analysis map/reduce technique, whereby an operation is first *mapped* over each element of a large dataset, and the results are then aggregated (or *reduced*) to produce the final result. In our case, the functionality π we pass to the `HSA.Apply` functionality could indeed be described as an element-by-element mapping (although there is a need to guarantee the absence of collision in the random identifiers), and `HSA.Shuffle`, along with the decryption of the final result, could be seen as a reduce step. Identifying ways in which the above commands could be amended to better support this standard cloud-based workflow is left as future work.

A.2 Use Case: a Generic Social Network

The ConfiChair protocol is particularly amenable to the kind of discussion carried out above, since it already identifies a privileged user that is trusted for privacy and functionality, and whose functionality can be offloaded to a sufficiently capable HSA. In general, cloud applications—and social networks more particularly—do not have such privileged roles, and relying on particular users' ability to perform complex tasks for functionality or security would be entirely unrealistic.

Motivated by the conference management system setting, we now wish to generalize protocols like ConfiChair to more generic social network scenarios, and incorporate the use of a HSA to avoid the computational burden imposed on the users by the use of various cryptographic mechanisms. The approach taken to support ConfiChair will not be applicable to general social network settings. Beyond the already discussed absence of a centrally trusted party, which can perform critical operations and distribute shared secrets, in general social networks, social networks are not clearly divided in phases, all users can write content and associate unique access policies to them, and all users can read the content they have been granted access to. Further, the "friend of a friend" problem represents a more serious privacy concern than it did in the case of conference management. Finally, friendship relations tend to change regularly, requiring our models to capture the dynamic aspects of friendships, but also to capture privacy in this dynamic setting. In such a setting, solutions for cryptographically-enforced access control [30] are not suitable even when they guarantee policy privacy [29]. In particular, policy changes tend to be very costly in such settings, and they do not guarantee unlinkability of requests.

Problem Statement. We thus consider the problem of providing standard social network functionalities (including access policies set per-content, rather than per-user) whilst providing strong privacy and unlinkability properties. In particular, although cloud C_1 can learn the identities of writers and readers as they make access requests, we require that neither C_1 nor C_2 is able to learn the contents, policies, or any relevant metadata that can link users. Further, we will look for solutions in which neither C_1 nor C_2 can distinguish between the type of requests (read, write, or change policy). We distribute the network's operations across two non-colluding, honest-but-curious cloud providers C_1 and C_2. C_1 will be given access to an HSA, and will be managing write, read, and policy change operations from users, and for routing data, while C_2 is used for storage.

Analysing the privacy requirements. Considering only privacy, and since C_1 has immediate access to all information about operations required to route messages, it is important to hide from C_1 the links between those requests and the encrypted blobs it routes between users and C_2, and must also hide relations between incoming and outgoing blobs of encrypted data. Encrypted blobs in this scenario may be quite large (for example, if they are files in a cloud data store with fine-grained sharing), and it may not be feasible to have the HSA perform decryptions and encryptions over the data itself. However, the HSA could be used to distribute per-object secrets to authorized users, essentially serving as a trusted monitor using simple cryptographic techniques to protect itself from its immediate environment (the cloud C_1).

However, the naive solution here, which consists in simply passing around encrypted blobs of data and encrypted keys would leak significant amounts of information about links and policies: C_1, in particular, could observe the same encrypted blobs being passed between users. We must, therefore, consider other solutions, which would for example prevent C_1 from learning information

by having the HSA request more blobs than necessary from C_2, or would involve activity cooperation between the HSA and C_2. All those solutions have in common the fact that existing TPM features will in fact be insufficient, since the HSA must either be able to make decisions (about which data to request from C_2), or monitor the behaviour of an untrusted party.

Appendix B

Provably-Secure Lattice-Based Direct Anonymous Attestation[1]

We now give details of the mathematics, and of the proposed lattice-based DAA construction, and an overview of the model and proof from El Kassem et al. [39].

B.1 Lattice-Based Cryptography: Some Notations and Assumptions

Throughout the rest of this section, we use polynomial rings $\mathcal{R}_q = \mathbb{Z}_q[x]/\langle x^n + 1\rangle$ to build cryptographic operations, where $\mathbb{Z}_q$ represents the quotient ring $\mathbb{Z}/q\mathbb{Z}$ and n is a power of 2.

Elements $\mathbf{a} \in \mathcal{R}_q$ are represented as polynomials $\mathbf{a} = a_0 + a_1 x + \cdots + a_{n-1}x^{n-1}$ of degree $n-1$ with integer coefficients. Operations over $\mathcal{R}_q$ handle these elements with the polynomials reduced modulo $x^n + 1$ and the coefficients reduced modulo q. An element $\mathbf{a} \in \mathcal{R}_q$ can thus be represented as a vector $(a_0, a_1, \ldots, a_{n-1}) \in \mathbb{Z}_q^n$. We use $\|\mathbf{a}\|_\infty$ to denote the infinity norm of $\mathbf{a} \in \mathcal{R}_q$, defined as $\|\mathbf{a}\|_\infty = (\|\mathbf{a}\|_\infty = \max_{0\leq j\leq n} |a_j|)$.

We also manipulate vectors of polynomials in dimension m, represented as $\hat{A} = (\mathbf{a}_1, \ldots, \mathbf{a}_m)$ where m is some positive integer. $\|\hat{A}\|_\infty$ is the infinity norm of the vector of polynomials $\hat{A}$ defined by $\|\hat{A}\|_\infty = \max_i \|\mathbf{a}_i\|_\infty$.

We write $[d]$ for the set $\{1, \ldots, d\}$ given any positive integer d. B_{3n} denotes the set of vectors $\mathbf{u} \in \{-1, 0, 1\}^{3n}$ having exactly n coordinates equal to -1, n coordinates equal to 0 and n coordinates equal to 1. β denotes a positive real norm bound and λ represents a security parameter.

For a fixed $\hat{A}$, we can use the inner-product $\hat{A} \cdot \hat{Z}$ with $\hat{Z}$ varying $\mathcal{R}_q^m$ to generate a lattice

$$L(\hat{A}) = \left\{\mathbf{v} \mid \exists_{\hat{Z}\in\mathcal{R}_q^m} \hat{A} \cdot \hat{Z} = \mathbf{v}\right\}$$

It is easy to see that the lattice $L(\hat{A})$ such defines satisfies Definition 2. Moreover, for a given $\mathbf{u}$, we define $L_{\mathbf{u}}^{\perp}(\hat{A})$ as

$$L_{\mathbf{u}}^{\perp}(\hat{A}) = \left\{\hat{Z} \in \mathcal{R}_q^m | \hat{A} \cdot \hat{Z} = \mathbf{u}\right\}$$

[1] This Appendix is based on a journal article authored by Nada El Kassem, Liqun Chen, Rachid El Bansarkhani (TU Darmstadt), Ali El Kaafarani (University of Oxford), Jan Camenisch (Dfinity), Patrick Hough (University of Oxford), Paulo Martins, and Leonel Sousa. [39]

Definition 2 (Lattices [37]). *Let $\mathbf{b}_1, \mathbf{b}_2, \cdots, \mathbf{b}_n$ be linearly independent vectors over $\mathbb{R}^m$. Let $B = [\mathbf{b}_1|\mathbf{b}_2|\cdots|\mathbf{b}_n] \in \mathbb{R}^{m\times n}$ having these vectors as columns. The lattice spanned by B is given by*

$$L(B) = \left\{ \sum_{i=1}^{n} z_i \mathbf{b}_i : z_i \in \mathbb{Z} \right\}$$

The vectors $\mathbf{b}_1, \mathbf{b}_2, \cdots, \mathbf{b}_n$ are called a basis of the lattice. The rank n of the lattice is defined to be the number of vectors in B. If $n = m$ then the lattice L is said to be a full-rank lattice.

The security of the proposed DAA scheme is based on the Ring-ISIS and Ring-LWE problems, characterised in Definitions 3 and 4. Both of these problems are usually assumed to be hard problems to solve, even when leveraging a general purpose quantum computer.

Definition 3 (The Ring Short Integer Solution Problem (Ring-SIS$_{n,m,q,\beta}$) [52]). *Given m uniformly random elements $\mathbf{a}_i \in \mathcal{R}_q$ defining a vector $\hat{A} = (\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_m)$, find a nonzero vector of polynomials $\hat{Z} = (\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_m) \in \mathcal{R}_q^m$ with $\|\hat{Z}\|_\infty \leq \beta$ such that: $\hat{A} \cdot \hat{Z} = \sum_{i\in[m]} \mathbf{a}_i \cdot \mathbf{z}_i = \mathbf{0}$.*
The Ring Inhomogeneous Short Integer Solution (Ring-ISIS$_{n,m,q,\beta}$) problem asks to find $\hat{Z}$ with $\|\hat{Z}\|_\infty \leq \beta$, and such that: $\hat{A} \cdot \hat{Z} = \mathbf{y}$, for some uniform random polynomial $\mathbf{y}$.

Definition 4 (The Ring Learning With Error Problem (Ring-LWE) [54]). *Let χ be an error distribution defined over $\mathcal{R}_q$, we define the following:*

***Ring-LWE distribution**: Choose a uniformly random ring element $\mathbf{s} \hookleftarrow \mathcal{R}_q$ called the secret, and a distribution χ. The ring-LWE distribution $A_{s,\chi}$ over $\mathcal{R}_q \times \mathcal{R}_q$ is sampled by choosing $\mathbf{a} \in \mathcal{R}_q$ uniformly at random, choosing randomly the noise $\mathbf{e} \hookleftarrow \chi$ and outputting $(\mathbf{a}, \mathbf{b}) = (\mathbf{a},\ \mathbf{s}\cdot\mathbf{a} + \mathbf{e} \mod q) \in \mathcal{R}_q \times \mathcal{R}_q$.*

***Ring-LWE Problems**: Let $\mathbf{u}$ be uniformly sampled from $\mathcal{R}_q$*

1. *The decision Ring-LWE problem asks to distinguish between $(\mathbf{a}, \mathbf{b}) \leftarrow A_{s,\chi}$ and $(\mathbf{a}, \mathbf{u})$ for a uniformly sampled secret $\mathbf{s} \hookleftarrow \mathcal{R}_q$.*
2. *The search Ring-LWE problem asks to return the secret vector $\mathbf{s} \in \mathcal{R}_q$ given a Ring-LWE sample $(\mathbf{a}, \mathbf{b}) \leftarrow A_{s,\chi}$ for a uniformly sampled secret $\mathbf{s} \hookleftarrow \mathcal{R}_q$.*

We will also sample values from Gaussian distributions over lattices, which we define below (Definition 5).

Definition 5 (Discrete Gaussian Distributions [54]). *The discrete Gaussian distribution on a non empty set L with parameter s, denoted by $\mathcal{D}_{L,s}$, is the distribution that assigns to each $\mathbf{x} \in L$ a probability proportional to $exp(-\pi(\|\mathbf{x}\|/s)^2)$.*

We can now describe the proposed scheme, starting with the signature and commitment schemes that form its core.

B.2 Building Blocks

The scheme modifies and combines a signature scheme by Boyen [12] and a commitment scheme by Baum et al [9]. We refer the reader to the full article [39] for detailed discussions of the original schemes.

Scheme B.1: Modified Version of Boyen's Signature Scheme

- KeyGen(1^λ):
  1. Generates a vector $\hat{A}_t \in \mathcal{R}_q^m$.
  2. Generates a vector $\hat{A} \in \mathcal{R}_q^m$ together with a trapdoor $\hat{T}$. The trapdoor enables sampling vectors of polynomials following a discrete Gaussian distribution on $L_{\mathbf{v}}^{\perp}(\hat{A}|\hat{B})$ for any $\mathbf{v} \in \mathcal{R}_q$ and $\hat{B} \in \mathcal{R}_q^m$ where $|$ denotes concatenation [41].
  3. Samples uniform random vectors of polynomials $\hat{A}_i \in \mathcal{R}_q^m$ for $i \in (0, [\ell])$.
  4. Selects a uniform random syndrome $\mathbf{u} \in \mathcal{R}_q$.
  5. Outputs the secret key $sk := \hat{T}$ and the public key $pk := (\hat{A}_t, \hat{A}, \hat{A}_0, \hat{A}_1, \ldots, \hat{A}_\ell, \mathbf{u}, q, \beta)$.
- Sign(sk, id $\in \{0,1\}^\ell$):
  1. Samples a vector of polynomials $\hat{Z}_t = (\mathbf{z}_1, \ldots, \mathbf{z}_m) \hookleftarrow \mathcal{D}_{\mathbb{Z}^n,s}^m$ such that $\|\hat{Z}_t\|_\infty \leq \beta$, and computes $\hat{A}_t \cdot \hat{Z}_t \equiv \mathbf{u}_t \mod q$.
  2. Generates a vector of polynomials $\hat{A}_{\text{id}} = [\hat{A}|\hat{A}_0 + \sum_{i=1}^{\ell} \text{id}_i \cdot \hat{A}_i] \in \mathcal{R}_q^{2m}$, as in the Boyen scheme.
  3. Using the secret key $\hat{T}$, samples $\hat{Z}_h = (\mathbf{z}_{m+1} \ldots, \mathbf{z}_{3m}) \hookleftarrow \mathcal{D}_{L_{\mathbf{u}_h}^{\perp}(\hat{A}_{\text{id}}),s}$, with $\|\hat{Z}_h\|_\infty \leq \beta$ and such that $\hat{A}_{\text{id}} \cdot \hat{Z}_h \equiv \mathbf{u}_h = (\mathbf{u} - \mathbf{u}_t) \mod q$.
  4. Outputs the signature $\hat{Z} = [\hat{Z}_t|\hat{Z}_h] = (\mathbf{z}_1, \ldots, \mathbf{z}_{3m})$.
- Verify(pk, id, $\hat{Z}$): If $[\hat{A}_t|\hat{A}_{\text{id}}] \cdot \hat{Z} \equiv \mathbf{u} \mod q$ and $\|\hat{Z}\|_\infty \leq \beta$ are satisfied, output 1, else 0.

As noted in Section 4.1, DAA credentials correspond to a signature of the platform identifier produced by the Issuer. In practice, DAA credentials are not held by a single party, but are split across the TPM and Host that constitutes the platform. We modify the signature scheme by Boyen [12], as described in Scheme B.1, to allow the key to be split. As a consequence of this modification, the Issuer's public-key must now include one more random vector of polynomials $\hat{A}_t \in \mathcal{R}_q$, and each signature is composed of two vectors of polynomials $\hat{Z}_t$ and $\hat{Z}_h$ of small norm such that $[\hat{A}_t|\hat{A}_{\text{id}}] \cdot [\hat{Z}_t|\hat{Z}_h] = \mathbf{u}$. The key share $\hat{Z}_t$ is held by the TPM whereas key share $\hat{Z}_h$ is held by the Host.

The security of this modified Boyen signature scheme is based on the original Boyen signature scheme which is unforgeable if the SIS problem is unfeasibly hard [12]. The unforgeability of the modified Boyen signature can be reduced to the existential unforgeability of the original Boyen signature scheme. We do not detail this analysis here.

In order to create a DAA signature, which is jointly signed by a TPM and its Host, we modify a commitment scheme by Baum et al. [9] to allow for two parties to commit to a set of secret values jointly. This modification is reflected in Scheme B.2 and leverages the scheme's additive homomorphism. Let $\hat{S}_t \in \mathcal{R}_q^{l_t}$ and $\hat{S}_h \in \mathcal{R}_q^{l_h}$, where l_t and l_h are some integers representing, respectively, the TPM and the Host's inputs. Let $\mathbf{s}_t$ and $\mathbf{s}_h$ in $\mathcal{R}_q$ be the TPM and the host's corresponding inputs to be added. Scheme B.2 allows the TPM and Host to jointly commit to

Scheme B.2: Modified Baum et al's Commitment Scheme

To commit to a message $\hat{S} = [(\mathbf{s}_t + \mathbf{s}_h)|\hat{S}_t|\hat{S}_h] \in \mathcal{R}_q^{l_t+l_h+1}$, the TPM and Host share a uniformly random vector of polynomials $\hat{B}$ in $\mathcal{R}_q^{(l_t+l_h+2)\times k}$.

To commit to a message $[\mathbf{s}_t|\hat{S}_t]$, the TPM:

- Chooses a uniformly random vector of invertible polynomials $\hat{R}_t \in \mathcal{D}$ such that $\|\hat{R}_t\|_\infty \leq \alpha_t$ for some small constant α_t.
- Computes $\mathbf{C}_t = \mathsf{COM}([\mathbf{s}_t|\hat{S}_t], \hat{R}_t) := \hat{B}\hat{R}_t + (\mathbf{0}|\mathbf{s}_t|\hat{S}_t|\hat{0} \in \mathcal{R}_q^{l_h})$, outputs $\mathbf{C}_t$.

To commit to a message $[\mathbf{s}_h|\hat{S}_h]$ the host:

- Chooses a uniformly random vector of invertible polynomials $\hat{R}_h \in \mathcal{D}$ such that $\|\hat{R}_h\|_\infty \leq \alpha_h$ for some small constant α_h.
- Computes $\mathbf{C}_h = \mathsf{COM}([\mathbf{s}_h|\hat{S}_h], \hat{R}_h) := \hat{B}\hat{R}_h + (\mathbf{0}|\mathbf{s}_h|\hat{0} \in \mathcal{R}_q^{l_t}|\hat{S}_h)$, outputs $\mathbf{C}_h$.

Now we have $\mathbf{C} = \mathbf{C}_t + \mathbf{C}_h = \hat{B}(\hat{R}_t + \hat{R}_h) + (\mathbf{0}|\mathbf{s}_t + \mathbf{s}_h|\hat{S}_t|\hat{S}_h) = \mathsf{COM}([\mathbf{s}_t + \mathbf{s}_h|\hat{S}_t|\hat{S}_h],\ \hat{R}_t + \hat{R}_h) = \mathsf{COM}(\hat{S}, \hat{R})$, where $\hat{R} = \hat{R}_t + \hat{R}_h$ and $\|\hat{R}\|_\infty < \alpha_t + \alpha_h$.

the vector $\left(\mathbf{s}_t + \mathbf{s}_h|\hat{S}_t|\hat{S}_h\right)$ without one learning about the input values of the other. The original scheme by Baum et al. was proved to be statistically hiding and computationally binding assuming an instantiation of the Ring-SIS problem. The security of our modified commitment scheme is based on that of the original scheme. In particular, we argue that splitting the prover role into two entities does not affect these two properties, but do not detail the security proof here.

B.3 The Proposed LDAA Scheme

We now describe our proposed DAA scheme, which builds upon the lattice-based constructions described above. The scheme's algorithms are described as reactive systems, which react to specific inputs from their environment, rather than more classic oracle-based escriptions in order to align with the UC-based security model we choose to use. The security of this scheme is based on the Ring-ISIS and Ring-LWE problems from Definitions 3 and 4.

Beyond the primitives already constructed bove, the scheme also assumes some standard functionalities, widely used in TPM applications. These additional assumptions follow Camenisch, Drijvers and Lehmann [17], and are not discussed further here. However, we note that precisely identifying them and their expected security is critical in developing sound foundations for the analysis of the security of TPM as a whole. Indeed, most of these functionalities will be *implemented* by other components or high-level functionalities provided by the TPM, perhaps jointly with its Host.

We assume the following functionalities:

- F_{ca} is a common certificate authority functionality that is available to all parties.

Scheme B.3: LDAA Setup

- F_{crs} creates the system parameters: sp $= (\lambda, q, n, m, \mathcal{R}_q, c, \beta, \beta', \ell, \eta)$, where λ, c and η are positive integer security parameters, β and β' are positive real numbers such that $\beta, \beta' < q$, and ℓ is the length of a message to be signed with Boyen's signature scheme.
- Upon input (SETUP, sid), where sid is a unique session identifier, the Issuer first checks that sid $= (\mathsf{I},\ \mathsf{sid}')$ for some sid′, then creates its key pair. The Issuer's public key is pp $= (\mathsf{sp}, \hat{A}_t, \hat{A}_I, \hat{A}_0, \hat{A}_1, ..., \hat{A}_\ell,\ \mathbf{u},\ \mathcal{H}, \mathcal{H}_0, \mathcal{H}_1)$, where $\hat{A}_t, \hat{A}_I, \hat{A}_i (i = 0, 1, ..., \ell) \in \mathcal{R}_q^m$, $\mathbf{u} \in \mathcal{R}_q$, $\mathcal{H} : \{0,1\}^* \to \mathcal{R}_q$, $\mathcal{H}_0 : \{0,1\}^* \to \{1,2,3\}^c$ and $\mathcal{H}_1 : \{0,1\}^* \to \{0,1\}^\eta$ be a collision resistant hash function. The Issuer's private key is $\hat{T}_I$, which is the trapdoor of $\hat{A}_I$ and $\|\hat{T}_I\|_\infty \leq \omega$, for some small real number ω. The Issuer initialises the list of joining members (Members $\leftarrow \emptyset$) and proves that his secret key is well formed by generating a proof of knowledge π_I, and registers the key $(\hat{T}_I, \pi_I)$ with F_{ca}. Finally, it outputs (SETUPDONE, sid).

- $F_{crs}^{\mathcal{D}}$ is a common reference string functionality that provides participants with all system parameters.
- F_{auth^*} is a special authenticated communication functionality that provides an authenticated channel between the issuer and the TPM via the host.
- F_{smt}^{l} is a secure message transmission functionality that provides an authenticated and encrypted communication between the TPM and the host.

Defining a DAA scheme requires the specification of five algorithms:

- *Setup*, which initializes and distributes secrets;
- *Join*, through which a platform requests credentials from the issuer and demonstrates its trust status;
- *Sign*,through which a platform produces a DAA signature for any chosen message;
- *Verify*, through which a verifier checks the validity of a given signature for a given message; and
- *Link*, through which anyone can test whether two signatures were produced using the same basename.

We describe each of these algorithms in turn.

Setup.

The *Setup* step is described in Scheme B.3 and corresponds to the generation of the parameters shared by the Issuer's community, along with the Issuer's secret-key and the initialisation of its internal state.

Scheme B.4: LDAA Join Request

- On input query (JOIN, sid, jsid, $\mathsf{tpm_i}$), the host $\mathsf{host_j}$ forwards (JOIN, sid, jsid) to I, who replies by sending (sid, jsid, ρ, bsn_I) back to $\mathsf{host_j}$, where ρ is a uniform random nonce $\rho \hookleftarrow \{0,1\}^\lambda$, and bsn_I is the Issuer's base name. This message is then forwarded to $\mathsf{tpm_i}$.
- The TPM proceeds as follows:
  1. It checks that no such entry exists in its storage.
  2. It samples a private key: $\hat{X}_t = (\mathbf{x}_1, \ldots, \mathbf{x}_m) \hookleftarrow \mathcal{R}_q^m$ with the condition $\|\hat{X}_t\|_\infty \leq \beta$, and stores its key as (sid, $\mathsf{host_j}$, $\hat{X}_t$, id).
  3. It computes the corresponding public key $\mathbf{u}_t = \hat{A}_t \cdot \hat{X}_t \mod q$, a link token $\mathsf{nym}_I = \mathcal{H}(\mathsf{bsn}_I) \cdot \mathbf{x}_1 + \mathbf{e}_I \mod q$ for some error $\mathbf{e}_I \hookleftarrow \mathcal{D}_{\mathbb{Z}^n, s'}$ such that $\|\mathbf{e}_I\|_\infty < \beta'$, and generates a signature based proof:
$$\pi_{\mathbf{u}_t} = \mathsf{SPK}\Big\{\mathsf{public} := \{\mathsf{sp}, \hat{A}_t, \mathbf{u}_t, \mathsf{bsn}_I, \mathsf{nym}_I\},$$
$$\mathsf{witness} := \{\hat{X}_t = (\mathbf{x}_1, \ldots, \mathbf{x}_m), \mathbf{e}_I\} :$$
$$\mathbf{u}_t = \hat{A}_t \cdot \hat{X}_t \mod q \ \wedge\ \|\hat{X}_t\|_\infty \leq \beta \ \wedge\ \mathsf{nym}_I = \mathcal{H}(\mathsf{bsn}_I) \cdot \mathbf{x}_1 + \mathbf{e}_I$$
$$\mathrm{mod}\ q \ \wedge\ \|\mathbf{e}_I\|_\infty \leq \beta'\Big\}(\rho).$$
  4. It sends (nym_I, id, $\mathbf{u_t}$, $\pi_{\mathbf{u_t}}$) to the issuer I via the host by means of F_{auth^*}, i.e., it gives F_{auth^*} an input (SEND, $(\mathsf{nym}_I, \pi_{\mathbf{u}_t}), (\mathsf{sid}, \mathsf{tpm_i}, I), \mathsf{jsid}, \mathsf{host_j}$).
- The host, upon receiving (APPEND, $(\mathsf{nym}_I, \pi_{\mathbf{u}_t}), (\mathsf{sid}, \mathsf{tpm_i}, I)$) from F_{auth^*}, forwards it to I by sending (APPEND, $(\mathsf{nym}_I, \pi_{\mathbf{u}_t}), (\mathsf{sid}, \mathsf{tpm_i}, I)$) to F_{auth^*} and keeps the state (jsid, $\mathbf{u_t}$, id).
- The Issuer, upon receiving (SENT, $(\mathsf{nym}_I, \pi_{\mathbf{u}_t}), (\mathsf{sid}, \mathsf{tpm_i}, I), \mathsf{jsid}, \mathsf{host_j}$) from F_{auth^*}, verifies the proof $\pi_{\mathbf{u}_t}$ to make sure that $\mathsf{tpm_i} \notin$ Members. I stores (jsid, nym_I, $\pi_{\mathbf{u}_t}$, id, $\mathsf{tpm_i}$, $\mathsf{host_j}$), and generates the message (JOINPROCEED, sid, jsid, id, $\pi_{\mathbf{u}_t}$).

Join.

The Join process is a protocol running between the Issuer I and a platform, consisting of a TPM $\mathsf{tpm_i}$ and a Host $\mathsf{host_j}$ (with an identifier id). More than one Join session be may run in parallel, and a unique sub-session identifier jsid is distributed to all participating parties to allow the Issuer to differentiate between them. The goal of the potocol is to allow the issuer I to check that the platform is qualified to execute the trusted computing attestation service, and to issue qualified platforms with a credential enabling it to create attestations.

A Join session works in two distinct phases, Join request and Join proceed. During the Join request, described in Scheme B.4, the TPM generates its private-key $\hat{X}_t$ and the corresponding public-key $\mathbf{u}_t$, along with a linking token $\mathsf{nym_I}$ associated with the Issuer, and a proof that the public-key and the token are well formed. The Issuer finalises the Join request phase by checking the validity of the proof and that the platform has not been provisioned before.
During Join proceed (Scheme B.5), the Issuer samples a small $\hat{X}_h$ such that $[\hat{A}_t|\hat{A}_{\mathsf{id}}] \cdot [\hat{X}_t|\hat{X}_h] = \mathbf{u}$. $\hat{X}_h$ is then transmitted to the Host.

Scheme B.5: LDAA Join Proceed

- If the platform chooses to proceed with the Join session, the message (JOINPROCEED, sid, jsid) is sent to the issuer, who performs as follows:
  1. It checks the record (jsid, nym_I, id, $\mathsf{tpm_i}$, $\mathsf{host_j}$, $\pi_{\mathbf{u}_t}$). For all nym'_I from the previous Join records, the issuer checks whether $\|\mathsf{nym}_I - \mathsf{nym}'_I\|_\infty \leq 2\beta'$ holds; if yes, the issuer treats this session as a rerun of the Join process; otherwise the issuer adds $\mathsf{tpm_i}$ to Members and goes to Step 2. If this is a rerun, the issuer will further check if $\mathbf{u}_t = \mathbf{u}'_t$; if not the issuer will abort; otherwise the issuer will jump to Step 4 returning $\hat{X}_h = \hat{X}'_h$. Note that this double check will make sure that any two DAA keys will not include the same $\mathbf{x}_1$ value.
  2. It calculates the vector of polynomials $\hat{A}_h = [\hat{A}_I | \hat{A}_0 + \sum_{i=1}^{\ell} \mathsf{id}_i \cdot \hat{A}_i] \in \mathcal{R}_q^{2m}$.
  3. It samples, using the issuer's private key $\hat{T}_I$, a preimage $\hat{X}_h = (\mathbf{x}_{m+1}, \ldots, \mathbf{x}_{3m})$ of $\mathbf{u} - \mathbf{u}_t$ such that: $\hat{A}_h \cdot \hat{X}_h = \mathbf{u}_h = \mathbf{u} - \mathbf{u}_t \mod q$ and $\|\hat{X}_h\|_\infty \leq \beta$.
  4. It sends (sid, jsid, $\hat{X}_h$) to $\mathsf{host_j}$ via F_{auth^*}.
- When the host receives the message (sid, jsid, $\hat{X}_h$), it checks that the equations $\hat{A}_h \cdot \hat{X}_h = \mathbf{u}_h \mod q$ and $\mathbf{u} = \mathbf{u}_t + \mathbf{u}_h$ are satisfied with $\|\hat{X}_h\|_\infty \leq \beta$. If the checks are correct, then $\mathsf{host_j}$ stores (sid, $\mathsf{tpm_i}$, id, $\hat{X}_h$, $\mathbf{u}_t$) and outputs (JOINED, sid, jsid).

The linking token created during the Join request phase allows the Issuer to check that the same private key is not used for two different TPMs in the Join proceed step, preventing unwanted linking of signatures that were not created by the same platform.

Sign.

After obtaining the credential from the Join process, $\mathsf{tpm_i}$ and $\mathsf{host_j}$ can jointly sign a message μ with respect to a basename bsn. Here again, we use a unique sub-session identifier ssid to allow for multiple parallel Sign sessions to take place.

Each session has two phases, Sign request and Sign proceed. Sign request, described in Scheme B.6, is mostly responsible for ensuring that the TPM and the Host have compatible secret-key shares.
Sign proceed, described in Scheme B.7, produces a zero-knowledge proof-of-knowledge of small $\hat{X}_t$ and $\hat{X}_h$ such that $[\hat{A}_t | \hat{A}_{\mathsf{id}}] \cdot [\hat{X}_t | \hat{X}_h] = \mathbf{u}$. More concretely, the TPM and the Host respectively commit to random strings each showing that either $\hat{X}_t$ and $\hat{X}_h$ are small or that $\hat{A}_t \cdot \hat{X}_t = \mathbf{u}_t$ and $\hat{A}_h \cdot \hat{X}_h = \mathbf{u}_h$. We then leverage the additive homomorphism of Scheme B.2 to produce commitments to strings showing that either $\hat{X}_t | \hat{X}_h$ is small or that $[\hat{A}_t | \hat{A}_{\mathsf{id}}] \cdot [\hat{X}_t | \hat{X}_h] = \mathbf{u}$. However, opening all the commitments would reveal the value of $\hat{X}_t | \hat{X}_h$. Instead, this procedure is iterated multiple times, and at each iteration a subset of the commitments is revealed at random. The randomness is derived from the message to be signed, as is standard in producing signature proofs of knowledge.

The computation also generates a nym tag, associated with the basename bsn.

Scheme B.6: LDAA Sign Request

- Upon input (SIGN, sid, ssid, $\mathsf{tpm_i}$, bsn, μ), $\mathsf{host_j}$ looks up the record (sid, $\mathsf{tpm_i}$, id, $\mathbf{u}_t$, $\hat{X}_h$), and sends the message (sid, ssid, bsn, μ) to $\mathsf{tpm_i}$.
- The TPM then does the following:
  1. It asks $\mathsf{host_j}$ for a permission to proceed.
  2. It makes sure to have a Join record (sid, id, $\hat{X}_t$, $\mathsf{host_j}$).
  3. It generates a sign entry (sid, ssid, bsn, μ) in its record.
  4. Finally it outputs (SIGNPROCEED, sid, ssid, bsn, μ).

Verify.

The verify algorithm, described in Scheme B.8, allows anyone to check whether a signature σ on a message μ with respect to a basename bsn is valid.

Link.

Finally, the link algorithm, depicted in Scheme B.9 allows anyone to check whether two signatures $(\sigma,\ \mu)$ and $(\sigma',\ \mu')$ that were generated for the same basename bsn stem from the same TPM. This is done by checking whether the difference between the two nym tags has a small norm.

B.3.1 The Proofs θ_t, θ_h and π.

We now describe the commitments and proofs of knowledge θ_t, θ_h and π in more detail. We borrow the techniques from [44] and rewrite $[\hat{A}_t|\hat{A}_h] \cdot [\hat{X}_t|\hat{X}_h]$ as

$$[\hat{A}_t|\hat{A}_I|\hat{A}_0|\hat{A}_1|\dots|\hat{A}_l] \cdot [\hat{X}_t|\hat{X}_{h_1}|\hat{X}_{h_2}|\mathsf{id}_1\hat{X}_{h_2}|\dots|\mathsf{id}_\mathsf{l}\hat{X}_{h_2}],$$

where $\hat{X}_h = [\hat{X}_{h_1} \in \mathcal{R}_q^m|\hat{X}_{h_2} \in \mathcal{R}_q^m]$, for a public $[\hat{A}_t|\hat{A}_I|\hat{A}_0|\hat{A}_1|\dots|\hat{A}_l]$, and extending and randomising id such that [43] is still applicable.

Our main technical innovation is the proposal of a proof about values that are shared between the TPM and the Host. Let $k = \lceil \log_2 \beta \rceil$. Since we are operating in the ring $\mathcal{R}_q = \mathbb{Z}_q[x]/\langle x^n + 1\rangle$, with $n = O(\lambda)$, then we can transform products of elements in $\mathcal{R}_q$ into matrix-vector products. The matrices $\bar{A}_i = \mathsf{rot}(\mathbf{a}_i)$ are constructed as defined in [44], for $i = (1, 2, \dots, (\ell + 3)m)$, for all polynomials $\mathbf{a}_i$ in $\hat{A}_t$, $\hat{A}_I$, $\hat{A}_0$, ..., $\hat{A}_\ell$, respectively. More concretely, the matrices $\bar{A}_i = \mathsf{rot}(\mathbf{a}_i)$ for $i = (1, 2, \dots, m)$ are associated with the m polynomials in $\hat{A}_t$, the matrices $\bar{A}_i = \mathsf{rot}(\mathbf{a}_i)$ for $i = (m, m + 1, \dots, 2m)$ with the m polynomials in $\hat{A}_I$, etc. Similarly, the vectors $\bar{\mathbf{x}}_i$ whose entries are the coefficients of $\mathbf{x}_i$, for $i = (1, 2, \dots, 3m)$, are produced for all polynomials $\mathbf{x_i}$ in $\hat{X}_t$ and $\hat{X}_h$, respectively. As a result of these operations, the products $\bar{A}_i\bar{\mathbf{x}}_i$ and $\mathbf{a}_i\mathbf{x}_i$ are isomorphic. Furthermore, the following extensions are considered:

- $\mathsf{id} = \{\mathsf{id}_1, ..., \mathsf{id}_\ell\} \in \{0, 1\}^\ell$ is extended to $\mathsf{id}^* \in \mathsf{B}_{2\ell}$ which is the set of vectors in $\{0, 1\}^{2\ell}$ of hamming weight ℓ.
- $\bar{A}_i^* = [\bar{A}_i|\mathbf{0} \in \mathbb{Z}^{n\times 3n}]$ for $i = 1$ to $i = (3 + \ell)m$ and $\bar{A}_i^* = \mathbf{0}$ for $(3 + l)m < i \leq (3 + 2l)m$.

Scheme B.7: LDAA Sign Proceed

- When $\mathsf{tpm_i}$ gets permission to proceed for ssid, the TPM proceeds as follows:

  1. It retrieves the records $(\mathsf{sid},\ \mathsf{id},\ \mathsf{host_j},\ \pi_{\mathbf{u}_t})$ and $(\mathsf{sid},\ \mathsf{ssid},\ \mathsf{bsn},\ \mu)$.
  2. Depending on the input bsn, there are two cases: If $\mathsf{bsn} \neq \perp$, the tpm computes the tag $\mathsf{nym} = \mathcal{H}(\mathsf{bsn}) \cdot \mathbf{x}_1 + \mathbf{e} \mod q$, for an error term $\mathbf{e} \hookleftarrow \mathcal{D}_{\mathbb{Z}^n,s'}$ such that $\|\mathbf{e}\|_\infty < \beta'$ and generates a commitment using Scheme B.2:

  $$\theta_t = \mathsf{COM}\Big\{\mathsf{public} := \{\mathsf{sp},\ \hat{A}_t,\ \mathsf{nym},\ \mathsf{bsn},\ \mathcal{H},\ \mathbf{u}_t\},$$
  $$\mathsf{witness} := \{\hat{X}_t = (\mathbf{x}_1, \ldots, \mathbf{x}_m), \mathbf{e}\} :$$
  $$\{\hat{A}_t \cdot \hat{X}_t = \mathbf{u}_t \ \wedge\ \|\hat{X}_t\|_\infty \leq \beta\} \ \wedge\ \mathsf{nym} = \mathcal{H}(\mathsf{bsn}) \cdot \mathbf{x}_1 + \mathbf{e} \ \wedge\ \|\mathbf{e}\|_\infty \leq \beta'\Big\}.$$

  If bsn=$\perp$, then $\mathsf{tpm_i}$ samples a random value $\mathsf{bsn} \leftarrow \{0,1\}^\lambda$, and then follows the previous case.
  3. $\mathsf{tpm_i}$ sends $(\mathsf{sid},\ \mathsf{ssid},\ \theta_t,\ \mu)$ to $\mathsf{host_j}$.
  4. When $\mathsf{host_j}$ receives the message $(\mathsf{sid},\ \mathsf{ssid},\ \theta_t,\ \mu)$, it checks that θ_t is valid, and subsequently generates a commitment using, again, Scheme B.2:

  $$\theta_h = \mathsf{COM}\Big\{\mathsf{public} := \{\mathsf{sp},\ \hat{A}_h,\ \mathbf{u}_h,\ \mu,\ \theta_t\},$$
  $$\mathsf{witness} := \{\hat{X}_h = (\mathbf{x}_{m+1}, \ldots, \mathbf{x}_{3m}),\ \mathsf{id}\} :$$
  $$\{\hat{A}_h \cdot \hat{X}_h = \mathbf{u}_h \ \wedge\ \|\hat{X}_h\|_\infty \leq \beta\}\Big\}.$$

  The two commitments θ_t and θ_h are homomorphically combined.
  5. The TPM and Host run the standard Fiat-Shamir transformation, and the result is a signature based proof (signed on the message μ):

  $$\pi = \mathsf{SPK}\Big\{\mathsf{public} := \{\mathsf{pp},\ \mathsf{nym},\ \mathsf{bsn}\},$$
  $$\mathsf{witness} := \{\hat{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_{3m}),\ \mathsf{id},\ \mathbf{e}\} :$$
  $$[\hat{A}_t|\hat{A}_h] \cdot \hat{X} = \mathbf{u} \ \wedge\ \|\hat{X}\|_\infty \leq \beta \ \wedge\ \mathsf{nym} = \mathcal{H}(\mathsf{bsn}) \cdot \mathbf{x}_1 + \mathbf{e}$$
  $$\mod q \ \wedge\ \|\mathbf{e}\|_\infty \leq \beta'\Big\}(\mu).$$

  The details of the θ_t, θ_h and π computation will be given below.
  6. $\mathsf{host_j}$ outputs the L-DAA signature $\sigma = (\mathsf{nym}, \mathsf{bsn}, \pi)$.

Scheme B.8: LDAA Verify

- Let RL denote a revocation list containing the secret keys of all known rogue TPMs. Upon input (VERIFY, sid, bsn, σ, μ, RL), the verifier proceeds as follows:

  1. It parses σ as (nym, bsn, π), and checks SPK on π with respect to bsn, nym, μ and $\mathbf{u}$, verifying the statement:

  $$[\hat{A}_t|\hat{A}_h] \cdot \hat{X} = \mathbf{u} \wedge \|\hat{X}\|_\infty \leq \beta \wedge \\ \text{nym} = \mathcal{H}(\text{bsn}) \cdot \mathbf{x}_1 + \mathbf{e} \mod q \wedge \|\mathbf{e}\|_\infty \leq \beta'.$$

  2. It checks that the secret key $\hat{X}_t$ that was used to generate nym, doesn't belong to the revocation list RL. This is done by checking whether the following equation holds:

  $$\forall \mathbf{x}_1 \in RL, \|\mathcal{H}(\text{bsn}) \cdot \mathbf{x}_1 - \text{nym}\|_\infty \leq \beta'.$$

  3. If all checks passed, the verifier outputs (VERIFIED, ssid, 1), and (VERIFIED, ssid, 0) otherwise.

Scheme B.9: LDAA Link

- Upon input (LINK, sid, σ, μ, σ', μ', bsn) the verifier follows the following steps:

  1. Starting from $\sigma = (\text{nym}, \text{bsn}, \pi)$ and $\sigma' = (\text{nym}', \text{bsn}, \pi')$, the verifier verifies σ and σ' individually.
  2. If any of the signatures is invalid, the verifier outputs $\perp$.
  3. Otherwise if $\|\text{nym} - \text{nym}'\|_\infty < 2\beta'$, the verifier outputs 1 (linked); otherwise 0 (not linked).

- $\bar{\mathbf{x}}_{(2+i)m+j} = \mathsf{id}_i^* \cdot \bar{\mathbf{x}}_{2m+j}$ for $1 \leq i \leq 2\ell$ and $1 \leq j \leq m$.

We decompose and extend the vectors $\bar{\mathbf{x}}_i$ and $\mathbf{e}$ into vectors of norm at most 1 such that $\bar{\mathbf{x}}_i = \sum_{d=1}^{k} 2^{d-1}\bar{\mathbf{x}}_i^d[1:n]$ and $\mathbf{e} = \sum_{j=1}^{k} \mathbf{e}^j[1:n]2^{j-1}$, where $\bar{\mathbf{x}}_i^d[1:n]$ and $\mathbf{e}^j[1:n]$ correspond to the first n entries of $\bar{\mathbf{x}}_i^d$ and $\mathbf{e}^j$, respectively, and

$$\{\mathbf{e}^j\}_{j=1}^k, \{\bar{\mathbf{x}}_1^j\}_{j=1}^k, \{\bar{\mathbf{x}}_2^j\}_{j=1}^k, \ldots, \{\bar{\mathbf{x}}_{(3+2l)m}^j\}_{j=1}^k \in B_{3n},$$

i.e. they have n entries equal to -1, n entries equal to 0 and n entries equal to 1.

The extensions of the $\bar{A}_i$ and id and the decompositions of the extensions of the $\mathbf{x_i}$ satisfy:

$$\begin{aligned}
\mathbf{u} &= [\hat{A}_t|\hat{A}_h] \cdot \hat{X} \\
&= [\hat{A}_t|\hat{A}_I|\hat{A}_0 + \sum_{i=1}^{\ell} \mathsf{id}_i \cdot \hat{A}_i] \cdot \hat{X} \\
&= \sum_{i=1}^{3m} \bar{A}_i \cdot \bar{\mathbf{x}}_i + \sum_{j=1}^{\ell} \mathsf{id}_j \cdot \sum_{i=1}^{m} \bar{A}_{i+(j+2)m} \cdot \bar{\mathbf{x}}_{i+2m} \\
&= \sum_{i=1}^{3m} \bar{A}_i^* \cdot \left(\sum_{d=1}^{k} 2^{d-1}\bar{\mathbf{x}}_i^d\right) + \sum_{j=1}^{2\ell} \mathsf{id}_j^* \cdot \sum_{i=1}^{m} \bar{A}_{i+(j+2)m}^* \cdot \left(\sum_{d=1}^{k} 2^{d-1}\bar{\mathbf{x}}_{i+2m}^d\right) \\
&= \sum_{i=1}^{(3+2\ell)m} \bar{A}_i^* \left(\sum_{d=1}^{k} 2^{d-1}\bar{\mathbf{x}}_i^d\right)
\end{aligned}$$

The commitment algorithm COM used to compute θ_t and θ_h is as explained in Scheme B.2. To produce θ_t, the TPM samples two vectors $\{\mathbf{r}_{\mathbf{e}}^j \hookleftarrow \mathbb{Z}_q^{3n}\}_{j=1}^k$ and $\{\mathbf{r}_i^j \hookleftarrow \mathbb{Z}_q^{3n}\}_{j=1}^k$ for $i \in [m]$ and $j \in [k]$; and the permutations $\{\phi_j \hookleftarrow \mathsf{S}_{3n}\}_{j=1}^k$ associated with $\hat{X}_t$, and $\{\varphi_j \hookleftarrow \mathsf{S}_{3n}\}_{j=1}^k$ for $\mathbf{e}$. The following terms are also calculated: $D = [\mathsf{rot}(\mathcal{H}(\mathsf{bsn}))|\mathbf{0}] \in \mathbb{Z}_q^{n\times 3n}$, $\mathbf{v}_i^j = \mathbf{x}_i^j + \mathbf{r}_i^j$ and $\mathbf{v}_{\mathbf{e}}^j = \mathbf{e}^j + \mathbf{r}_{\mathbf{e}}^j$. $\theta_t = (\mathbf{C}_{t1}, \mathbf{C}_{t2}, \mathbf{C}_{t3})$ can then be computed as:

- $\mathbf{C}_{t1} = \mathsf{COM}(\sum_{i=1}^{m} \bar{A}_i^* \cdot (\sum_{j=1}^{k} 2^{j-1}\mathbf{r}_i^j),\ D \cdot (\sum_{j=1}^{k} 2^{j-1}\mathbf{r}_1^j) + [\mathbf{I}|\mathbf{0}] \cdot (\sum_{j=1}^{k} 2^{j-1}\mathbf{r}_{\mathbf{e}}^j), \{\phi_j\}_{j=1}^k, \{\varphi_j\}_{j=1}^k)$.
- $\mathbf{C}_{t2} = \mathsf{COM}(\{\phi_j(\mathbf{r}_1^j), \ldots, \phi_j(\mathbf{r}_m^j)\}_{j=1}^k, \{\varphi_j(\mathbf{r}_{\mathbf{e}}^j)\}_{j=1}^k)$.
- $\mathbf{C}_{t3} = \mathsf{COM}(\{\phi_j(\mathbf{v}_1^j), \ldots, \phi_j(\mathbf{v}_m^j)\}_{j=1}^k, \{\varphi_j(\mathbf{v}_{\mathbf{e}}^j)\}_{j=1}^k)$.

In a similar fashion, the Host samples two vectors $\{\mathbf{r}_i^j \hookleftarrow \mathbb{Z}_q^{3n}\}_{j=1}^k$ for $i - m \in [(2+2\ell)m]$ and $j \in [k]$, and $\mathbf{r}_{\mathsf{id}^*} \hookleftarrow \mathbb{Z}_q^{2\ell}$; and the permutations $\tau \hookleftarrow \mathsf{S}_{2\ell}$ for id^*, $\{\delta_j \hookleftarrow \mathsf{S}_{3n}\}_{j=1}^k$ for $\hat{X}_{h_1}$ and $\{\psi_j \hookleftarrow \mathsf{S}_{3n}\}_{j=1}^k$ for $\hat{X}_{h_2}$. The Host also computes $\mathbf{v}_i^j = \mathbf{x}_i^j + \mathbf{r}_i^j$ and $\mathbf{v}_{\mathsf{id}^*} = \mathsf{id}^* + \mathbf{r}_{\mathsf{id}^*}$. Then, it computes $\theta_t = (\mathbf{C}_{h1}, \mathbf{C}_{h2}, \mathbf{C}_{h3})$:

- $\mathbf{C}_{h1} = \mathsf{COM}(\sum_{i=m+1}^{(3+2\ell)m} \bar{A}_i^* \cdot (\sum_{j=1}^{k} 2^{j-1}\mathbf{r}_i^j), \tau, \{\delta_j\}_{j=1}^k, \{\psi_j\}_{j=1}^k)$.
- $\mathbf{C}_{h2} = \mathsf{COM}(\{\delta_j(\mathbf{r}_{m+1}^j), \cdots,\ \delta_j(\mathbf{r}_{2m}^j),\ \psi_j(\mathbf{r}_{2m+1}^j), \ldots,\ \psi_j(\mathbf{r}_{3m}^j), \psi_j(\mathbf{r}_{(\tau(1)+2)m+1}^j),$
  $\ldots, \psi_j(\mathbf{r}_{(\tau(1)+3)m}^j), \ldots, \psi_j(\mathbf{r}_{(\tau(2\ell)+2)m+1}^j), \ldots, \psi_j(\mathbf{r}_{(\tau(2\ell)+3)m}^j)\}_{j=1}^k, \tau(\mathbf{r}_{\mathsf{id}^*}))$.
- $\mathbf{C}_{h3} = \mathsf{COM}(\{\delta_j(\mathbf{v}_{m+1}^j), \ldots, \delta_j(\mathbf{v}_{2m}^j), \psi_j(\mathbf{v}_{2m+1}^j), \ldots, \psi_j(\mathbf{v}_{3m}^j), \psi_j(\mathbf{v}_{(\tau(1)+2)m+1}^j),$
  $\ldots, \psi_j(\mathbf{v}_{(\tau(1)+3)m}^j), \ldots, \psi_j(\mathbf{v}_{(\tau(2\ell)+2)m+1}^j), \ldots, \psi_j(\mathbf{v}_{(\tau(2\ell)+3)m}^j)\}_{j=1}^k, \tau(\mathbf{v}_{\mathsf{id}^*}))$.

The proof π is computed using a similar strategy, but where the commitments are produced using the homomorphic properties of Scheme B.2. Since soudness of the proof requires multiple iterations of the proving process, $\mathsf{tpm_i}$ hands out the commitments of the total c rounds to $\mathsf{host_j}$. $\mathsf{host_j}$ then adds its own commitments to those of the TPM, generating $CMT = (\mathbf{C}_1, \mathbf{C}_2, \mathbf{C}_3)$ such that:

- $\mathbf{C}_1 = \mathsf{COM}(\sum_{i=1}^{m} \bar{A}_i^* \cdot (\sum_{j=1}^{k} 2^{j-1}\mathbf{r}_i^j) + \sum_{i=m+1}^{(3+2\ell)m} \bar{A}_i^* \cdot (\sum_{j=1}^{k} 2^{j-1}\mathbf{r}_i^j), D \cdot (\sum_{j=1}^{k} 2^{j-1}\mathbf{r}_1^j) + [\mathbf{I}|\mathbf{0}] \cdot (\sum_{j=1}^{k} 2^{j-1}\mathbf{r}_\mathbf{e}^j), \tau, \{\phi_j\}_{j=1}^k, \{\delta_j\}_{j=1}^k, \{\psi_j\}_{j=1}^k, \{\varphi_j\}_{j=1}^k)$.
- $\mathbf{C}_2 = \mathsf{COM}(\{\phi_j(\mathbf{r}_1^j), \ldots, \phi_j(\mathbf{r}_m^j), \delta_j(\mathbf{r}_{m+1}^j), \ldots, \delta_j(\mathbf{r}_{2m}^j), \psi_j(\mathbf{r}_{2m+1}^j), \ldots, \psi_j(\mathbf{r}_{3m}^j), \psi_j(\mathbf{r}_{(\tau(1)+2)m+1}^j), \ldots, \psi_j(\mathbf{r}_{(\tau(1)+3)m}^j), \ldots \psi_j(\mathbf{r}_{(\tau(2\ell)+2)m+1}^j), \ldots, \psi_j(\mathbf{r}_{(\tau(2\ell)+3)m}^j)\}_{j=1}^k, \{\varphi_j(\mathbf{r}_\mathbf{e}^j)\}_{j=1}^k, \tau(\mathbf{r}_{\mathsf{id}^*}))$.
- $\mathbf{C}_3 = \mathsf{COM}(\{\phi_j(\mathbf{v}_1^j), \ldots, \phi_j(\mathbf{v}_m^j), \delta_j(\mathbf{v}_{m+1}^j), \ldots, \delta_j(\mathbf{v}_{2m}^j), \psi_j(\mathbf{v}_{2m+1}^j), \ldots, \psi_j(\mathbf{v}_{3m}^j), \psi_j(\mathbf{v}_{(\tau(1)+2)m+1}^j), \ldots \psi_j(\mathbf{v}_{(\tau(1)+3)m}^j), \ldots, \psi_j(\mathbf{v}_{(\tau(2\ell)+2)m+1}^j), \ldots, \psi_j(\mathbf{v}_{(\tau(2\ell)+3)m}^j)\}_{j=1}^k, \{\varphi_j(\mathbf{v}_\mathbf{e}^j)\}_{j=1}^k, \tau(\mathbf{v}_{\mathsf{id}^*}))$.

Inspired by [53], instead of directly storing the $\mathbf{C}_1$, $\mathbf{C}_2$ and $\mathbf{C}_3$ values in π we opt to store their hash, and gaining a significant reduction in the proof size. The challenges are generated following the Fiat-Shamir approach, namely by using a hash function that consumes $\mathcal{H}_1(\mathbf{C}_1)|\mathcal{H}_1(\mathbf{C}_2)|\mathcal{H}_1(\mathbf{C}_3)$ and outputs a random looking distribution of $\{1,2,3\}^c$:

$$\{\mathbf{CH}_j\}_{j=1}^c = \mathcal{H}_0(\mu, \mathcal{H}_1(\mathbf{C}_1^j)|\mathcal{H}_1(\mathbf{C}_2^j)|\mathcal{H}_1(\mathbf{C}_3^j)\}_{j=1}^c, \mathsf{pp}) \in \{1,2,3\}^c.$$

For each challenge, the $\mathsf{tpm_i}$ and the $\mathsf{host_j}$ combine the required values to produce the following responses:

- if $\mathbf{CH} = \mathbf{1}$, $\mathbf{C}_2$ and $\mathbf{C}_3$ are revealed, corresponding to all the permuted $\tau(\mathsf{id}^*)$, $\tau(\mathbf{r}_{id^*})$, $\{\phi_j(\mathbf{x}_i^j)\}_{j=1}^k$, $\{\delta_j(\mathbf{x}_i^j)\}_{j=1}^k$, $\{\psi_j(\mathbf{x}_i^j)\}_{j=1}^k$, $\{\varphi_j(\mathbf{e}^j)\}_{j=1}^k$, $\{\varphi_j(\mathbf{r}_e^j)\}_{j=1}^k$, $\{\phi_j(\mathbf{r}_i^j)\}_{j=1}^k$, $\{\delta_j(\mathbf{r}_i^j)\}_{j=1}^k$ and $\{\psi_j(\mathbf{r}_i^j)\}_{j=1}^k$.
- if $\mathbf{CH} = \mathbf{2}$, $\mathbf{C}_1$ and $\mathbf{C}_3$ are revealed, corresponding to the permutations τ, $\{\phi_j\}_{j=1}^k$, $\{\delta_j\}_{j=1}^k$, $\{\psi_j\}_{j=1}^k$, $\{\varphi_j\}_{j=1}^k$ and all the $\mathbf{v}$ values.
- if $\mathbf{CH} = \mathbf{3}$, $\mathbf{C}_1$ and $\mathbf{C}_2$ are revealed, corresponding to all the permutations τ, $\{\phi_j\}_{j=1}^k$, $\{\delta_j\}_{j=1}^k$, $\{\psi_j\}_{j=1}^k$, $\{\varphi_j\}_{j=1}^k$ and all the $\mathbf{r}$ values.

Finally $\mathsf{host_j}$ sends the proof to the verifier.

Depending on the prover's inputs, the verifier can always check 2 out of 3 commitments. When $\mathbf{CH} = \mathbf{1}$, the verifier will be convinced that the $\mathbf{e_i^j}$ and the $\bar{\mathbf{x}}_\mathbf{i}^\mathbf{j}$ were small. When $\mathbf{CH} = \mathbf{2}$ or $\mathbf{CH} = \mathbf{3}$, the verifier will be able to validate either the left or the right-hand side of the following expressions:

$$\sum_{i=1}^{(3+2\ell)m} \hat{A}_i^* \sum_{d=1}^{k} 2^{d-1}\mathbf{r}_\mathbf{i}^\mathbf{d} = \sum_{i=1}^{(3+2\ell)m} \hat{A}_i^* \sum_{d=1}^{k} 2^{d-1}\left(\bar{\mathbf{x}}_i^d + \mathbf{r}_\mathbf{i}^\mathbf{d}\right) - \mathbf{u}$$

$$D \cdot \sum_{d=1}^{k} 2^{d-1}\mathbf{r}_1^d + [\mathbf{I}|\mathbf{0}] \cdot \sum_{d=1}^{k} 2^{d-1}\mathbf{r}_\mathbf{e}^d = D \cdot \sum_{d=1}^{k} 2^{d-1}\left(\bar{\mathbf{x}}_1^d + \mathbf{r}_1^d\right) + [\mathbf{I}|\mathbf{0}] \cdot \sum_{d=1}^{k} 2^{d-1}\left(\mathbf{r}_\mathbf{e}^d + \mathbf{e}^d\right) - \mathsf{nym}$$

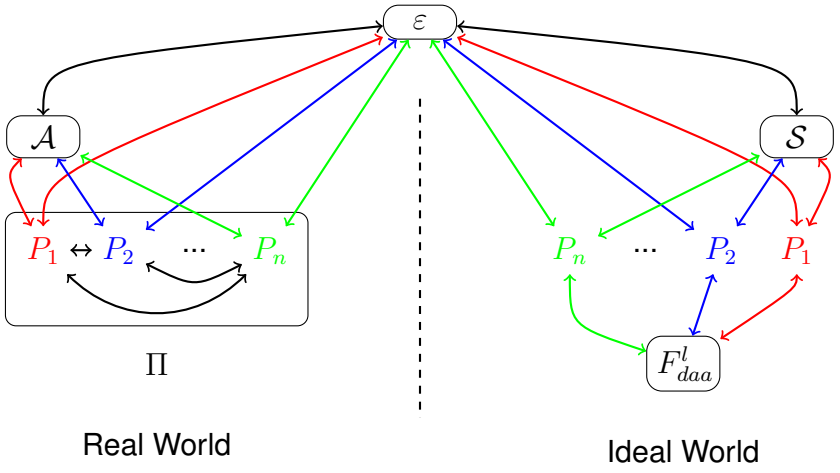

Figure B.1: Universal composability security model: the real and the ideal world executions are indistinguishable to the environment ε.

B.4 Security Model and Proof

We use the security model for DAA proposed by Camenisch, Drijvers and Lehmann [17]. The security definition is given in the Universal Composability (UC) model, represented in Figure B.1, with respect to an ideal functionality F^l_{daa}. In UC, an environment ε should not be able to distinguish with a non-negligible probability between two worlds:

1. The real world, where each party P_i in the DAA protocol executes its assigned part of the protocol Π. The network is controlled by an adversary $\mathcal{A}$ that communicates with ε.

2. The ideal world, in which all parties forward their inputs to a trusted third party, called the ideal functionality F^l_{daa}, which internally performs all the required tasks and creates the parties' outputs.

A protocol Π is said to securely realize F^l_{daa} if for every adversary $\mathcal{A}$ performing an attack in the real world, there is an ideal world adversary $\mathcal{S}$ that performs the same attack in the ideal world. More precisely, given a protocol Π, an ideal functionality F^l_{daa} and an environment ε, we say that Π securely realises F^l_{daa} if the real world in which Π is used is as secure as the ideal world in which F^l_{daa} is used. A UC security model therefore requires the specification of an ideal functionality which provides the same interface as the protocol under study—in this case, all stages of the five DAA algorithms. That ideal functionality should separtely be argued to have the desired security properties.

In general, a DAA scheme should have the following properties:

- *Unforgeability* This property requires that the issuer is honest and should hold even if the host is corrupt. If all the TPMs are honest, then no adversary can output a signature on a message M with respect to a basename (bsn). On the other hand, if not all the TPMs are honest, say n TPMs are corrupt, the adversary can at most output n unlinkable signatures with respect to the same basename.

- *Anonymity*: This property requires that the entire platform $(\mathsf{tpm_i} + \mathsf{host_j})$ is honest and should hold even if the issuer is corrupt. Starting from two valid signatures with respect to two different basenames, the adversary cannot tell whether these signatures were produced by one or two different honest platforms.

Scheme B.10: Ideal Setup

On input(SETUP, sid) from the issuer I, F_{daa}^{l} does the following:

- Verify that $(I,\ \text{sid}') = \text{sid}$ and output (SETUP, sid) to $\mathcal{S}$.
- SET Algorithms. Upon receiving the algorithms (Kgen, sig, ver, link, identify) from the simulator $\mathcal{S}$, it checks that (ver, link, identify) are deterministic [Check-I].
- Output (SETUPDONE, sid) to I.

- *Non-frameability*: This requires that the entire platform $(\text{tpm}_\text{i} + \text{host}_\text{j})$ is honest and should hold even if the issuer is corrupt. It ensures that no adversary can produce a signature that links to signatures generated by an honest platform.

As in the standardised DAA schemes supported by the TPM (either the TPM Version 1.2 or the TPM Version 2.0), in the proposed L-DAA scheme, privacy assumes that the entire platform—both TPM and Host—are honest. Camenisch et al. [16] consider that the TPM may be compromised, and that privacy must still hold whenever the host is honest, regardless of the corruption state of the TPM. Our analysis does not yet cover this scenario.

B.4.1 The Ideal Functionality F_{daa}^{l}

The ideal functionality F_{daa}^{l} is defined using the algorithms described in Schemes B.10 (*Setup*), B.11 (*Join*), B.12 (*Sign*), B.13 (*Verify*), and B.14 (*Link*).
In the UC model, several sessions of the protocol are allowed to run at the same time and each session will be given a global identifier sid that consists of an issuer I and a unique string sid′, i.e. $\text{sid} = (\text{sid}', \text{I})$. We also uniquely identify the Join and Sign sub-sessions with jsid and ssid. F_{daa}^{l} is parameterized by a leakage function $l : \{0,1\}^* \rightarrow \{0,1\}^*$, which models the information leakage that occurs in the communication between a host host_j and a TPM tpm_i.

Schemes B.10, B.11, B.12, B.13 and B.14 aso make use of some auxiliary algorithms, which we define below.

- Kgen(1^λ): A probabilistic algorithm that takes a security parameter λ and generates keys gsk for honest TPMs.
- sig$(gsk,\ \mu,\ \text{bsn})$: A probabilistic algorithm used for honest TPMs. On input of a key gsk, a message μ and a basename bsn, it outputs a signature σ.
- ver$(\sigma,\ \mu,\ \text{bsn})$: A deterministic algorithm that is used in the VERIFY interface. On input of a signature σ, a message μ and a basename bsn, it outputs $f = 1$ if the signature is valid, $f = 0$ otherwise.
- link$(\sigma_1,\ \mu_1,\ \sigma_2,\ \mu_2,\ \text{bsn})$: A deterministic algorithm that will be used in the LINK interface. It outputs 1 if both σ_1 and σ_2 were generated by the same TPM with respect to the same bsn, 0 otherwise.
- identify$(gsk, \sigma,\ \mu,\ \text{bsn})$: A deterministic algorithm that will be used to ensure consistency with the ideal functionality F_{daa}^{l}'s internal records. It outputs 1 if a key gsk was used to produce a signature σ, 0 otherwise.

Scheme B.11: Ideal Join

JOIN

1. JOIN REQUEST: On input (JOIN, sid, jsid, $\mathsf{tpm_i}$) from the host $\mathsf{host_j}$ to join the TPM $\mathsf{tpm_i}$, the ideal functionality F_{daa}^{l} proceeds as follows:
   - Create a join session $\langle$jsid, $\mathsf{tpm_i}$, $\mathsf{host_j}$, request$\rangle$.
   - Output (JOINSTART, sid, jsid, $\mathsf{tpm_i}$, $\mathsf{host_j}$) to $\mathcal{S}$.
2. JOIN REQUEST DELIVERY: Proceed upon receiving delivery notification from $\mathcal{S}$.
   - Update the session record to $\langle$jsid, $\mathsf{tpm_i}$, $\mathsf{host_j}$, delivery$\rangle$.
   - If I or $\mathsf{tpm_i}$ is honest and $\langle \mathsf{tpm_i}, \star, \star \rangle$ is already in Members, output $\perp$ [Check II].
   - Output (JOINPROCEED, sid, jsid, $\mathsf{tpm_i}$) to I.
3. JOIN PROCEED:
   - Upon receiving an approval from I, F_{daa}^{l} updates the session record to $\langle$jsid, sid, $\mathsf{tpm_i}$, $\mathsf{host_j}$, complete$\rangle$.
   - Output (JOINCOMPLETE, sid, jsid) to $\mathcal{S}$.
4. KEY GENERATION: On input (JOINCOMPLETE, sid, jsid, gsk) from $\mathcal{S}$.
   - If both $\mathsf{tpm_i}$ and $\mathsf{host_j}$ are honest, set $gsk = \perp$.
   - Else, verify that the provided gsk is eligible by performing the following checks:
     - If $\mathsf{host_j}$ is corrupt and $\mathsf{tpm_i}$ is honest, then CheckGskHonest(gsk)=1 [Check III].
     - If $\mathsf{tpm_i}$ is corrupt, then CheckGskCorrupt(gsk)=1 [Check IV].
   - Insert $\langle \mathsf{tpm_i}, \mathsf{host_j}, gsk \rangle$ into Members, and output (JOINED, sid, jsid) to $\mathsf{host_j}$.

Scheme B.12: Ideal Sign

1. SIGN REQUEST: On input (SIGN, sid, ssid, $\mathsf{tpm_i}$, μ, bsn) from the host $\mathsf{host_j}$ requesting a DAA signature by a TPM $\mathsf{tpm_i}$ on a message μ with respect to a basename bsn, the ideal functionality does the following:
   - Abort if I is honest and no entry $\langle \mathsf{tpm_i}, \mathsf{host_j}, \star \rangle$ exists in Members.
   - Else, create a sign session $\langle \mathsf{ssid}, \mathsf{tpm_i}, \mathsf{host_j}, \mu, \mathsf{bsn}, \mathsf{request} \rangle$.
   - Output (SIGNSTART, sid, ssid, $\mathsf{tpm_i}$, $\mathsf{host_j}$, $l(\mu, \mathsf{bsn})$) to $\mathcal{S}$.
2. SIGN REQUEST DELIVERY: On input (SIGNSTART, sid, ssid) from $\mathcal{S}$, update the session to $\langle \mathsf{ssid}, \mathsf{tpm_i}, \mathsf{host_j}, \mu, \mathsf{bsn}, \mathsf{delivered} \rangle$. F_{daa}^{l} output (SIGNPROCEED, sid, ssid, μ, bsn) to $\mathsf{tpm_i}$.
3. SIGN PROCEED: On input (SIGNPROCEED, sid, ssid) from $\mathsf{tpm_i}$
   - Update the records $\langle \mathsf{ssid}, \mathsf{tpm_i}, \mathsf{host_j}, \mu, \mathsf{bsn}, \mathsf{delivered} \rangle$.
   - Output (SIGNCOMPLETE, sid, ssid) to $\mathcal{S}$.
4. SIGNATURE GENERATION: On the input (SIGNCOMPLETE, sid, ssid, σ) from $\mathcal{S}$, if both $\mathsf{tpm_i}$ and $\mathsf{host_j}$ are honest then:
   - Ignore the adversary's signature σ.
   - If bsn $\neq \perp$, then retrieve gsk from the $\langle \mathsf{tpm_i}, \mathsf{bsn}, gsk \rangle \in$ DomainKeys.
   - If bsn $= \perp$ or no gsk was found, generate a fresh key $gsk \leftarrow Kgen(1^{\lambda})$.
   - Check CheckGskHonest(gsk)=1 [Check V].
   - Store $\langle \mathsf{tpm_i}, \mathsf{bsn}, gsk \rangle$ in DomainKeys.
   - Generate the signature $\sigma \leftarrow sig(gsk, \mu, \mathsf{bsn})$.
   - Check ver(σ, μ, bsn)=1 [Check VI].
   - Check identify(σ, μ, bsn, gsk)=1 [Check VII].
   - Check that there is no TPM other than $\mathsf{tpm_i}$ with key gsk' registered in Members or DomainKeys such that identify(σ, μ, bsn, gsk')=1 [Check VIII].
   - If $\mathsf{tpm_i}$ is honest, then store $\langle \sigma, \mu, \mathsf{tpm_i}, \mathsf{bsn} \rangle$ in Signed and output (SIGNATURE, sid, ssid, σ) to $\mathsf{host_j}$.

Scheme B.13: Ideal Verify

- On input (VERIFY, sid, μ, bsn, σ, RL), from a party V to check whether a given signature σ is a valid signature on a message μ with respect to a basename bsn and the revocation list RL, the ideal functionality does the following:
- Extract all pairs $(gsk_i,$ $\mathsf{tpm}_i)$ from the DomainKeys and Members, for which identify(σ, μ, bsn, gsk)=1. Set $b = 0$ if any of the following holds:
  - More than one key gsk_i was found [Check IX].
  - I is honest and no pair $(gsk_i,$ $\mathsf{tpm}_i)$ was found [Check X].
  - An honest tpm_i was found, but no entry $\langle \star,$ $\mu,$ $\mathsf{tpm}_i,$ bsn$\rangle$ was found in Signed [Check XI].
  - There is a key $gsk' \in RL$, such that identify(σ, μ, bsn, gsk')=1 and no pair $(gsk,$ $\mathsf{tpm}_i)$ for an honest tpm_i was found [Check XII].
- If $b \neq 0$, set $b \leftarrow$ver$(\sigma,$ $\mu,$ bsn$)$ [Check XIII].
- Add $\langle \sigma,$ $\mu,$ bsn, $RL,$ $b\rangle$ to VerResults, and output (VERIFIED, sid, b) to V.

Scheme B.14: Ideal Link

On input (LINK, sid, $\sigma_1,$ $\mu_1,$ $\sigma_2,$ $\mu_2,$ bsn), with bsn $\neq \perp$, from a party V to check if the two signatures stem from the same signer or not. The ideal functionality deals with the request as follows:

- If at least one of the signatures ($\sigma_1,$ $\mu_1,$ bsn) or ($\sigma_2,$ $\mu_2,$ bsn) is not valid (verified via the VERIFY interface with $RL \neq \emptyset$), output $\perp$ [Check XIV].
- For each gsk_i in Members and DomainKeys, compute $b_i \leftarrow$ identify($\sigma_1,$ $\mu_1,$ bsn, gsk_i) and b'_i= identify($\sigma_2,$ $\mu_2,$ bsn, gsk_i) then set:
  - $f \leftarrow 0$ if $b_i \neq b'_i$ for some i [Check XV].
  - $f \leftarrow 1$ if $b_i = b'_i = 1$ for some i [Check XVI].
- If f is not defined, set $f \leftarrow$link($\sigma_1,$ $\mu_1,$ $\sigma_2,$ $\mu_2,$ bsn), then output (LINK, sid, f) to V.

The following functions are also used to check whether or not a TPM key is consistent with the internal records of F_{daa}^{l}:

1. CheckGskHonest(gsk): If the tpm_i is honest, and no signatures in Signed or valid signatures in VerResults identify to be signed by gsk, then gsk is eligible and the function returns 1, otherwise it returns 0.

2. CheckGskCorrupt(gsk): If the tpm_i is corrupt and there does not exist a $gsk' \neq gsk$ and a $(\mu,\ \sigma,\ \mathsf{bsn})$ such that both keys identify to be the owners of the same signature σ, then gsk is eligible and the function returns 1, otherwise it returns 0.

B.4.2 Proof Sketch.

We can now provide a sketch of the security proof of the proposed protocol, referring the reader to the journal article [39] for details of the proof steps. Our proof shows that there can exist no environment ε that can distinguish the real world protocol Π with an adversary $\mathcal{A}$ from the ideal world F_{daa}^{l} with a simulator $\mathcal{S}$. It works by exhibiting a sequence of transformations that gradually transform the game representing the real world interactions into that representing the ideal world interactions, taking care to ensure that each transition only induces a negligible distinguishing probability. The resulting sequence of 16 games reflects transitions and transformations similar to those used by Camenisch, Drijvers and Lehmann [17], but differ in the complexity of each transition, and of the arguments supporting them, which now rely on lattice-based assumptions. The sequence of games is as follows:

Game 1 ▷ The Real World Protocol.

↓

Game 2 ▷ Equivalent to Game 1.

An entity C is introduced. C receives all inputs from the honest parties and simulates the real world protocol for them.

↓

Game 3 ▷ Similar to Game 2 but with a different structure.

C is split into two parts, F and $\mathcal{S}$. F behaves as an ideal functionality. It receives all the inputs and forwards them to $\mathcal{S}$, who simulates the real world protocol for honest parties and sends the output to F. F then forwards the outputs to ε.

↓

Game 4 ▷ ε will notice no change from Game 3.

F now stores the algorithms for the issuer I in the setup interface and ensures that the structure of sid *is correct for an honest I, and aborts if not. In case I is corrupt, $\mathcal{S}$ extracts the secret key for I and proceeds in the setup interface on behalf of I.*

↓

Game 5 ▷ Same outcomes as Game 4.

F now performs the verification and linking checks instead of forwarding them to $\mathcal{S}$. There are no protocol messages and the outputs are exactly as in the real world protocol. However, F doesn't contain a revocation check in the verification algorithm. Nonetheless, F can perform this check separately.

Game 6

▷ In all cases F and $\mathcal{S}$ can interact to simulate the real world protocol.

The join interface of F is now changed. F now stores in its records the members that have joined. If I is honest, F stores the secret key, extracted from $\mathcal{S}$, for corrupt TPMs. $\mathcal{S}$ always has enough information to simulate the real world protocol except when the issuer is the only honest party. In this case, $\mathcal{S}$ doesn't know who initiated the join, so it can't make a join query with F on the host's behalf. Thus, to deal with this case, F can safely choose any corrupt host and put it into Members. The identities of hosts are only used to create signatures for platforms with an honest TPM or honest host, so one needn't worry about fully corrupted platforms.

Game 7

▷ A distinguisher between Game 6 and 7 could solve Decision RLWE.

F now creates anonymous signatures for honest platforms by running the algorithms defined in the setup interface. Let us start by defining Game $7.k.k'$: in this game F handles the first k' signing inputs of tpm_k with algorithms and subsequent inputs are forwarded to S as before. We note that Game 7.0.0=Game 6. For increasing k', Game $7.k.k'$ will be at some stage equal to Game $7.k+1.0$, this is because there can only be a polynomial number of signing queries to be processed. Therefore, for large enough k and k', F handles all the signing queries of all TPMs, and Game 7 is indistinguishable from Game $7.k.k'$. To prove that Game $7.k.k'+1$ is indistinguishable from Game $7.k.k'$, suppose there exists an environment that can distinguish a signature of an honest party using $\hat{X}_t$ from a signature using a different $\hat{X'}_t$, then the environment can solve the Decision Ring -LWE Problem. Suppose that $\mathcal{S}$ is given tuples $\{(\mathbf{a}_i, \mathbf{b}_i)\}_{i=1}^{k'}, (\mathbf{c}, \mathbf{d})$, where $\mathbf{b}_i = \mathbf{a}_i \cdot \mathbf{x}_1 + \mathbf{e}_i$ for a uniform random $\mathbf{a}_i$ and $\mathbf{c} \in \mathcal{R}_q$, and it is challenged to decide if the pair $(\mathbf{c}, \mathbf{d})$ is chosen from a Ring LWE distribution (for some secret $\mathbf{x}_1$) or uniform random. $\mathcal{S}$ proceeds in simulating the TPM without knowing the secret $\mathbf{x}_1$. $\mathcal{S}$ can answer all the $\mathcal{H}$ queries, as $\mathcal{S}$ is controlling F_{crs}, on $\mathsf{bsn_j}$ with $\mathcal{H}(\mathsf{bsn}_j) = \mathbf{a}_j$ for $j \leq k'$. For $j = k'+1$, S sets $\mathcal{H}(\mathsf{bsn}_{k'+1}) = \mathbf{c}$, otherwise $\mathcal{H}(\mathsf{bsn}_j) = \mathbf{r}_j$ for some uniform random $\mathbf{r}_j$ and $j > k'+1$. Signing queries on behalf of tpm_i for $i < k$ are forwarded by F to $\mathcal{S}$, which calls the real world protocol. For $i > k$, gsks are freshly sampled for each $\mathsf{bsn_i}$. However, for tpm_k and $i \leq k'$, the simulator $\mathcal{S}$ sets $\mathsf{nym}_i = \mathbf{b}_i$, and for $i = k'+1$ it sets $\mathsf{nym} = \mathbf{d}$. For $i > k'+1$, $\mathcal{S}$ samples fresh $\mathbf{x}_i$ and generates $\mathsf{nym}_i = \mathcal{H}(\mathsf{bsn}_i) \cdot \mathbf{x}_i + \mathbf{e}_i$, keeping track of all the generated nym_i such that it always output the same nym_i for an associated bsn_i. For each case, tpm_k can provide a simulated proof. Any distinguisher between Game $7.k.k'$ and Game $7.k.k'+1$ can solve the Decision Ring-LWE Problem.

Game 8 ▷ ε observes no difference between Game 7 and Game 8.

F now no longer informs $\mathcal{S}$ about the message and the basename that are being signed. If the whole platform is honest, then $\mathcal{S}$ can learn nothing about the message μ and the basename bsn. Instead, $\mathcal{S}$ knows only the leakage $l(\mu, \mathsf{bsn})$. To simulate the real world, $\mathcal{S}$ chooses a pair (μ', bsn') such that $l(\mu', \mathsf{bsn}')$=$l(\mu, \mathsf{bsn})$.

Game 9 ▷ Game 8 and Game 9 are indistinguishable.

If I is honest, then F now only allows platforms that joined to sign. An honest host will always check whether it joined with a TPM in the real world protocol, so there is no difference for honest hosts. Also an honest TPM only signs when it has joined with the host before. In the case that an honest $\mathsf{tpm_i}$ *performs a join protocol with a corrupt host* $\mathsf{host_j}$ *and honest issuer, the simulator will make a join query with F, to ensure that* $\mathsf{tpm_i}$ *and* $\mathsf{host_j}$ *are in Members.*

Game 10

▷ Checks in Game 10 produce the same results as those of Game 9.

When storing a new $gsk = \hat{X}_t$, F checks CheckGskCorrupt(gsk)=1 or CheckGskHonest(gsk)=1. These checks will always pass. Valid signatures always satisfy $\mathsf{nym} = \mathcal{H}(\mathsf{bsn}) \cdot \mathbf{x}_1 + \mathbf{e}$ *where $\|\mathbf{x}_1\|_\infty < \beta$ and $\|\mathbf{e}\|_\infty < \beta'$. By the unique Short Vector Problem, there exists only one tuple $(\mathbf{x}_1, \mathbf{e})$ such that $\|\mathbf{x}_1\|_\infty < \beta$ and $\|\mathbf{e}\|_\infty < \beta'$ for small enough β and β'. Thus, CheckGskCorrupt(gsk) will always give the correct output. Also due to the large min-entropy of discrete Gaussians the probability that sampling a gsk $\hat{X}'_t = \hat{X}_t$ is negligible, thus with overwhelming probability there doesn't exist a signature already using the same $gsk = \hat{X}_t$, which implies that CheckGskHonest(gsk) will always give the correct output.*

Game 11

▷ Game 11 produces the same results as Game 10 based on RLWE.

In this game, F checks that honestly generated signatures are valid. This is true as the sig algorithm always produces signatures passing through the verification checks. Also those signatures satisfy $\mathsf{identify}(gsk,\ \sigma,\ \mu,\ \mathsf{bsn}) = 1$ *which is checked via* nym*. F also makes sure, using Members and DomainKeys, that honest users are not sharing the same secret key gsk. If there exists a key $gsk = \hat{X}_t$ in Members and DomainKeys such that $\|\mathsf{nym} - \mathcal{H}(\mathsf{bsn})\mathbf{x}_1\|_\infty < \beta'$, then this breaks the search Ring-LWE problem.*

Game 12

▷ Valid signatures are associated with a single gsk.

Check-IX is added to ensure that there are no multiple gsk values matching one signature. Since there exists only one pair $(\mathbf{x}_1, \mathbf{e_I})$ such that $\|\mathbf{x}_1\|_\infty < \beta$ and $\|\mathbf{e}_I\|_\infty < \beta'$, satisfying $\mathsf{nym_I} = \mathcal{H}(\mathsf{bsn}) \cdot \mathbf{x_1} + \mathbf{e_I}$*, two different gsks can't share the same $\mathbf{x}_1$.*

Game 13

▷ Game 13 is indistinguishable from Game 12 based on the hardness of the Ring-ISIS Search Problem.

To prevent accepting signatures that were issued by the use of join credentials not issued by an honest issuer, F adds a further check Check-X. This is due to the unforgeability of Boyen signatures.

Game 14 ▷ Game 14 is indistinguishable from Game 13 based on the hardness of Ring-LWE.

Check-XI is added to F, preventing anyone from forging signatures using an honest TPM's gsk and credential. In fact, if a valid signature is given on a message that the TPM never signed, the proof could not have been simulated. It would extract $\mathbf{x}_1$, breaking the Ring-LWE problem.

↓

Game 15 ▷ Game 15 is indistinguishable from Game 14 based on the hardness of Ring-LWE.

Check-XII is added to F, ensuring that honest TPMs are not revoked. If a honest TPM is simulated by means of the Ring-LWE problem instance, if a proper key RL is found, it must be the secret key of the target instance. This is again equivalent to solving the search Ring-LWE problem.

↓

Game 16 ▷ F now includes all the functionalities of F_{daa}^{l}. □

All the remaining checks of the ideal functionality F_{daa}^{l} that are related to link queries are now included. Using the fact that if a gsk matches one signature and not the other, Game 16 is indistinguishable from Game 15.

B.5 Conclusions and Lessons Learned

In Section 4.1 and in this Appendix, we have discussed recent progress towards a practical lattice-based DAA scheme that is provably secure. We note, however, that both performance and security must be improved. The scheme improves on past proposals through the development and use of a novel commitment scheme that allows TPM and Host to cooperatively produce a zero-knowledge proof-of-knowledge over the secret they share, but remains inefficient. We prove security of the proposed LDAA scheme in a static compromise model, and do not attempt to prove privacy when the TPM falls under adversary control. In addition, the use of the UC model imposes the use of session identifiers which cannot be used in practical TPM-based scenarios to disambiguate usage. We prove security in a QR0 setting.

Still, the results go some way towards refining the scheme and model, and improve the existing baseline for lattice-based DAA schemes on all fronts. Further, developing the proof has allowed us to gain some insight into the challenges of such proofs for lattice-based cryptography in a TPM-based setting.

The challenges of obtaining compositional security models for the TPM identified in D3.1 [22] remain. More precisely, in this particular proof, we assume secure message transmission between TPM and Host, and authenticated message transmission between TPM and Issuer. We note that the latter has currently not been implemented securely, due to issues related to bootstrapping trust in unknown TPMs.