Lattice-based Direct Anonymous Attestation (LDAA)

Nada EL Kassem[a], Liqun Chen[a], Rachid El Bansarkhani[b], Ali El Kaafarani[c], Jan Camenisch[d], Patrick Hough[c], Paulo Martins[e], Leonel Sousa[e]

[a] *University of Surrey, UK*
[b] *TU Darmstadt, Germany*
[c] *University of Oxford, UK*
[d] *Dfinity, Switzerland*
[e] *INESC-ID, Instituto Superior Técnico, Universidade de Lisboa, Portugal*

Abstract

The Cloud-Edges (CE) framework, wherein small groups of Internet of Things (IoT) devices are serviced by local edge devices, enables a more scalable solution to IoT networks. The trustworthiness of the network may be ensured with Trusted Platform Modules (TPMs). This small hardware chip is capable of measuring and reporting a representation of the state of an IoT device. When connecting to a network, the IoT platform might have its state signed by the TPM in an anonymous way to prove both its genuineness and secure state through the Direct Anonymous Attestation (DAA) protocol. Currently standardised DAA schemes have their security supported on the factoring and discrete logarithm problems. Should a quantum-computer become available in the next few decades, these schemes will be broken. There is therefore a need to start developing a post-quantum DAA protocol. This paper presents a Lattice-based DAA (LDAA) scheme to meet this requirement. The security of this scheme is proved in the Universally Composable (UC) security model under the hardness assumptions of the Ring Inhomogeneous Short Integer Solution (Ring-ISIS) and Ring Learning With Errors (Ring-LWE) problems. Compared to the only other post-quantum DAA scheme available in related art, the storage requirements of the TPM are reduced twofold and the signature sizes 5 times. Moreover, experimental results show that the signing and verification operations are accelerated 1.1 and 2.0 times, respectively.

Keywords: Lattice based Cryptography, Direct Anonymous Attestation, Universally Composable Security Model

☆The research of Liqun Chen, Paulo Martins and Leonel Sousa was supported by European Union's Horizon 2020 research and innovation programme under grant agreement No. 779391 (FutureTPM).

Email addresses: n.elkassem@surrey.ac.uk (Nada EL Kassem), liqun.chen@surrey.ac.uk (Liqun Chen), elbansarkhani@cdc.informatik.tu-darmstadt.de (Rachid El Bansarkhani), elkaafarani@maths.ox.ac.uk (Ali El Kaafarani), jan@dfinity.org (Jan Camenisch), patrick.hough@maths.ox.ac.uk (Patrick Hough), paulo.sergio@netcabo.pt (Paulo Martins), las@inesc-id.pt (Leonel Sousa)

1. Introduction

Internet of Things (IoT) devices are becoming increasingly common in our everyday lives and on industrial applications. Networks of sensors and actuators are being implemented in the general infrastructure to achieve better energy efficiency; in production to obtain customised mass production; and in the automotive industry to improve safety, among others. Due to the always larger amounts of data produced by such sensors, the Cloud-assisted IoT (CoT) paradigm, wherein the sensors are connected to a central Cloud server, is suffering from scalability issues. In contrast, with the emerging Cloud-Edges (CE) technology, distributed edge devices, such as smart gateways and local PCs, offer cloud-like services to only a limited group of devices. These edge devices should ensure that IoT platforms connecting to the network are in a trustworthy state and do not constitute an hazard to the remaining nodes.

The Trusted Computing Group (TCG) is an industrial consortium that aims at the design of standards for trusted systems [1], including the Trusted Platform Module (TPM) [2] and the Trusted Network Connect (TNC) [3]. The TPM is a piece of security hardware that may be included on the motherboards of IoT devices [4]. When loading code, including bootloaders, BIOSs, etc., the binaries are measured and extended into the TPM Platform Configuration Registers (PCRs). As a result, the PCRs contain a value that is representative of the current state of the system. When combined with the TNC, which is an architecture for network access control, the functionality offered by the TPM may be exploited to ensure that all devices connected to an edge IoT network are in a secure state [5].

In particular, the Direct Anonymous Attestation (DAA) is a protocol enabling a TPM and a Host IoT device not only to authenticate themselves to a Verifying edge device and to prove that the Host is in a trustworthy state, but also to do so in a privacy-preserving manner. More concretely, the DAA provides the TPM with the ability to sign its register values in an anonymous way, whilst still convincing the Verifier that it possesses valid DAA credentials. Anonymity is particularly important in applications such as the automotive industry, wherein tracking of drivers should be prevented.

Recent estimates predict that a quantum computer capable of breaking cryptosystems based on the hardness of factoring and computing the discrete logarithm may be produced in the next few decades [6]. Since the operating period of IoT devices may be very long, there is a need to update the cryptographic primitives of current DAA schemes, which rely on classical cryptography, to be quantum-resistant. In this paper, we design a Lattice-based DAA (LDAA) scheme suitable for inclusion in future TPMs. Our LDAA scheme is based on [7], and designed to reduce the workload of the TPM as much as possible. Unlike in [7], where the Host operations do not help reduce the workload of the TPM, herein a new lattice based direct anonymous attestation scheme is proposed that reduces the demands on the TPM in terms of storage costs and computational resources. Experimental results show that the storage requirements of the TPM are reduced twofold and the size of the signatures 5 times when compared with [7]. The signing and verification operations are also accelerated 1.1 and 2.0 times. Furthermore, the security of our LDAA scheme is based on the hardness of the Ring-ISIS and Ring-LWE problems. As there is no known quantum algorithm that solves either of these problems, this provides a promising DAA scheme for the post-quantum age. We also prove the security of our L-DAA scheme in the Universal Composability (UC) model.

In Section 2, the backgrounds regarding the TPM and lattice-based cryptography are reviewed. Section 3 proposes novel post-quantum cryptographic primitives, and the new LDAA scheme is built supported on these primitives. Afterwards, in Section 4, the LDAA security model, supported in the UC framework, is defined. A sketched security proof for our L-DAA scheme, based on this security model, is presented in Section 5. Finally, we discuss the performance of the proposed scheme in comparison to related art in Section 6, evaluate it experimentally in Section 7 and conclude the paper in Section 8. Appendix A and Appendix B present the security analysis of the novel post-quantum cryptographic primitives. Appendix C discusses several functionalities necessary for our L-DAA security proof and a detailed security proof of the proposed L-DAA scheme is presented in Appendix D.

2. Background

In this section, the TPM functionality is reviewed, with a particular focus on the DAA protocol. Moreover, an overview of lattice-based cryptography is provided, namely in what regards commitments, signatures and zero-knowledge proofs-of-knowledge.

2.1. TPM and TNC

The TPM standard defines an hardware chip that serves as the root of trust of a platform [2]. It can securely store cryptographic key material and platform measurements that help ensure IoT platforms remain trustworthy. Figure 1 illustrates how the TPM builds a representation of the platform state. As software is loaded, hashes of the binaries are extended into the PCRs. The extension corresponds to the hashing of the concatenation of the previous content of the PCR with the inputted hash, and the storing of the result in the PCR. The nature of hardware-based cryptography ensures that the information stored in the TPM is better protected than software-preserved data. Use-cases benefiting from the TPM technology are manifold. One may, for instance, seal an hard drive decryption key to a specific platform state. If the platform is later infected with a rootkit, the TPM representation of the platform state will change and access to the key will not be made available. While this functionality may benefit CE computing, herein we focus on the attestation functionalities made possible with the TPM. The TNC defines an architecture for network access control where these functionalities may be exploited [3]. An edge device, in charge of

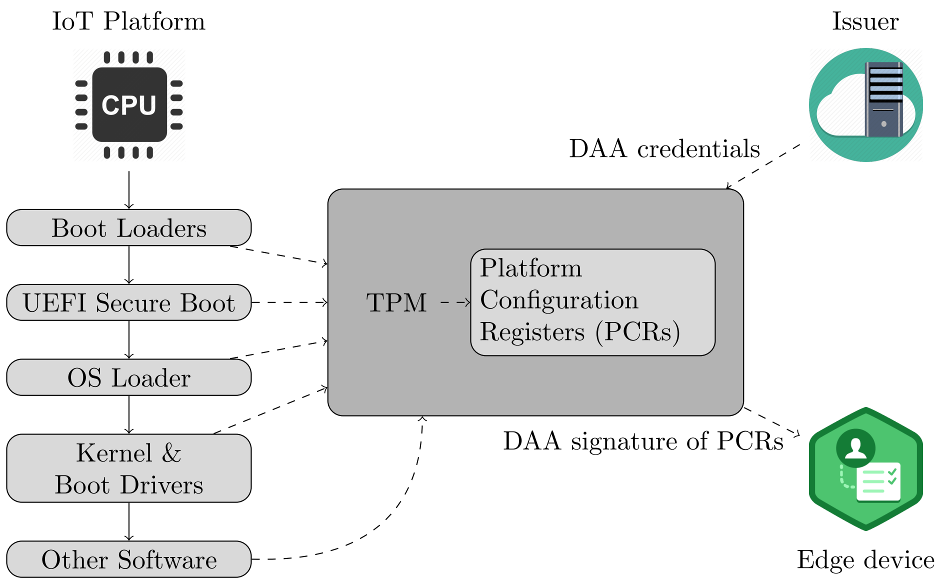

Figure 1: TPM-based attestation of an IoT system. As software is loaded, the TPM builds a representation of the platform state. When access is requested to an Edge network, a DAA signature of the PCRs is sent to the Edge device

an edge network, may require IoT devices to prove that they possess a genuine TPM platform and that they are in a trustworthy state before access to the network is granted [5]. This is achieved with the DAA protocol, as represented on the right-hand side of Figure 1.

DAA. 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 consists of the (Host, TPM) pair. Their membership to the DAA community and trustworthy state is proved by providing the verifier with a DAA signature of the TPM representation of the Host state. The DAA signature includes a zero-knowledge proof-of-knowledge, which is a cryptographic construct 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. In contrast to other privacy-preserving constructs, like group signatures, DAA does not support the property of traceability, wherein a group manager can identify the signer from a given group signature. Furthermore, 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 as the replacement of traceability. One is the rogue signer detection, i.e. with a signer's private key anyone can check whether a given DAA signature was created under this key or not. The other is the 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.

2.2. Lattice-based Cryptography

Throughout this paper polynomial rings $\mathcal{R}_q = \mathbb{Z}_q[x]/\langle x^n+1\rangle$ will be used to build cryptographic operations, where $\mathbb{Z}_q$ represents the quotient ring $\mathbb{Z}/q\mathbb{Z}$ and n is power of 2. Elements of $\mathbf{a} \in \mathcal{R}_q$ are represented as polynomials $\mathbf{a} = a_0 + a_1x + \cdots + a_{n-1}x^{n-1}$ of degree $n-1$ with integer coefficients. $\mathbf{a}$ can also be represented as a vector $(a_0, a_1, \ldots, a_{n-1}) \in \mathbb{Z}^n$, with $\|\mathbf{a}\|_\infty$ denoting the infinity norm of $\mathbf{a}$ ($\|\mathbf{a}\|_\infty = \max_{0\le j\le n} |a_j|$). Vectors of polynomials are 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 $\|\hat{A}\|_\infty = \max_i \|\mathbf{a}_i\|_\infty$. Finally, the following notations are also used. $[d]$ is the set $\{1, \ldots, d\}$ for a 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 $\hat{A}$, the inner-product product $\hat{A}\cdot\hat{Z}$, $\forall \hat{Z} \in \mathcal{R}_q^m$ generates a lattice $L(\hat{A}) = \left\{\mathbf{v} | \exists_{\hat{Z}\in\mathcal{R}_q^m} \hat{A}\cdot\hat{Z} = \mathbf{v}\right\}$ satisfying Definition 1. Moreover, for a given $\mathbf{u}$, the lattice $L_{\mathbf{u}}^{\perp}(\hat{A})$ is defined as

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

The security of the proposed DAA scheme will be based on the problems characterised in Definitions 2 and 3. Furthermore, Gaussian distributed samples over lattices will be required, as per Definition 4.

Definition 1 (Lattices [8])**.** *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.

Definition 2 (The Ring Short Integer Solution Problem (Ring-SIS$_{n,m,q,\beta}$) [9])**.** *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 $\|\hat{Z}\|_\infty \le \beta$ 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 $\|\hat{Z}\|_\infty \le \beta$, and such that: $\hat{A}\cdot\hat{Z} = \boldsymbol{y}$, for some uniform random polynomial $\boldsymbol{y}$.*

Definition 3 (The Ring Learning With Error Problem (Ring-LWE) [10])**.** *Let χ be an error distribution defined over $\mathcal{R}$, 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 problem of Ring-LWE 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 $\mathbf{s} \hookleftarrow \mathcal{R}_q$.*

Definition 4 (Discrete Gaussian Distributions [10])**.** *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 $exp(-\pi(\|\mathbf{x}\|/s)^2)$.*

2.3. Boyen's Signature Scheme

A DAA credential corresponds to a signature of the IoT platform identifier produced by the Issuer. Herein, signatures will be designed based on Boyen's scheme. This scheme operates over a ring $\mathcal{R}_q$, with $m = O(\log q)$, and can sign any message $\mathsf{id} \in \{0,1\}^{\ell}$. The scheme includes the algorithms depicted in Scheme 1. The security of the Boyen signature scheme is based on the hardness of the Ring-ISIS problem and is proved to be secure in the standard model. We refer to [11] for the security proof. The proof was improved later in [12] by using a new trapdoor and ring analogue. A modification to Boyen's signature scheme will be proposed in Section 3 to support the issuing of DAA credentials. This modification will be proven to be secure in Appendix A.

2.4. Baum et al's Commitment Scheme

The proposed DAA technique follows a non-interactive sigma protocol construction. Sigma protocols are a basic building block for zero-knowledge proofs-of-knowledge, wherein the prover commits to a series of interrelated values, an array of challenges is produced, and the prover opens a subset of the commitments according to the challenges. Each opening reveals a property of the DAA credential held by the IoT device. Given sufficient challenges/openings, an Edge device will be convinced of the validity of those credentials. Herein, Baum et al's commitment scheme [14], shown in Scheme 2 is exploited to develop a quantum-resistant DAA. The security of this commitment scheme is based on the hardness of the Ring-ISIS problem. We refer to [14] for the security proof. In Section 3, Baum et al's commitment scheme will be modified to handle the splitting of the prover into two entities (the TPM and the Host) efficiently, and the new security proof will be introduced in Appendix B.

Scheme 1: Boyen's Signature Scheme

- KeyGen(1^λ):
  1. Generates a vector of polynomials $\hat{A} \in \mathcal{R}_q^m$ together with a trapdoor $\hat{T}$. The trapdoor enables sampling vectors of polynomials following a discrete Gaussian distribution on $L_{\mathbf{v}}^{\perp}(\hat{A}|\hat{A}_{\mathsf{id}})$ for any $\mathbf{v} \in \mathcal{R}_q$ and $\hat{A}_{\mathsf{id}} \in \mathcal{R}_q^m$ where $|$ denotes concatenation [13].
  2. Samples uniform random vectors of polynomials $\hat{A}_i \in \mathcal{R}_q^m$ for $i \in (0, [\ell])$.
  3. Selects a uniform random syndrome $\mathbf{u} \in \mathcal{R}_q$.
  4. Outputs the secret key $sk := \hat{T}$ and the public key $pk := (\hat{A}, \hat{A}_0, \hat{A}_1, \ldots, \hat{A}_\ell, \mathbf{u}, q, \beta)$.
- Sign(sk, $\mathsf{id} \in \{0,1\}^\ell$):
  1. Generates a vector of polynomials $\hat{A}_{\mathsf{id}} = [\hat{A}|\hat{A}_0 + \sum_{i=1}^{\ell} \mathsf{id}_i \cdot \hat{A}_i] \in \mathcal{R}_q^{2m}$.
  2. Using the secret key $\hat{T}$, samples $\hat{Z} = (\mathbf{z}_1, \ldots, \mathbf{z}_{2m}) \hookleftarrow \mathcal{D}_{L_{\mathbf{u}}^{\perp}(\hat{A}_{\mathsf{id}}),s}$, such that $\hat{A}_{\mathsf{id}} \cdot \hat{Z} \equiv \mathbf{u} \mod q$ and $\|\hat{Z}\|_\infty \leq \beta$.
  3. Outputs the signature $\hat{Z} = (\mathbf{z}_1, \ldots, \mathbf{z}_{2m})$.
- Verify(pk, id, $\hat{Z}$): If $\hat{A}_{\mathsf{id}} \cdot \hat{Z} \equiv \mathbf{u} \mod q$ and $\|\hat{Z}\|_\infty \leq \beta$ are satisfied, output 1, else 0.

Scheme 2: Baum et al's Commitment Scheme

- C.KeyGen(k): Given a security parameter k, generates the system parameters (q, $\mathcal{R}_q$, α, γ, $\hat{B}$), where q is a prime modulus defining $\mathcal{R}_q$, α and γ are positive numbers, and $\hat{B}$ is a uniformly random vector of polynomials in $\mathcal{R}_q^{(d+1)\times k}$, for some positive integer d.
- Commit ($\hat{S}$): To commit to a message $\hat{S} \in \mathcal{R}_q^d$, choose a uniformly random vector of invertible polynomials $\hat{R} \in \mathcal{D} \subseteq \mathcal{R}^k$ such that $\|\hat{R}\|_\infty \leq \alpha$. Compute $\mathbf{C} = \mathsf{COM}(\hat{S}, \hat{R}) := \hat{B}\hat{R} + (\mathbf{0}, \hat{S})$, and output $\mathbf{C}$.
- Open($\mathbf{C}, \hat{S}, \hat{R}, \mathbf{p}$): A valid opening of a commitment $\mathbf{C}$ is a 3-tuple: $\hat{S} \in \mathcal{R}_q$, $\hat{R} \in \mathcal{R}^k$ and an invertible polynomial $\mathbf{p} \in \mathcal{R}$ such that $\|\mathbf{p}\|_\infty \leq \gamma$. The verifier checks that

$$\hat{B}\hat{R} + (\mathbf{0}, \mathbf{p}\hat{S}) = \mathbf{p}\mathbf{C} \quad \text{with} \quad \|\hat{R}\|_\infty \leq \alpha$$

2.5. ISIS Proof

DAA signatures will correspond to a zero-knowledge proof of a Boyen's signature. The techniques developed herein to achieve that can be seen as a generalisation of the scheme proposed by Ling et al [15] to prove the knowledge of a small vector $\mathbf{x}$ with $||\mathbf{x}||_\infty \leq \beta$ such that $A\mathbf{x} = \mathbf{y} \bmod q$ for a secret $\mathbf{x} \in \mathbb{Z}_q^n$ and public $A \in \mathbb{Z}_q^{m\times n}$, $\mathbf{y} \in \mathbb{Z}_q^m$. Instead of arguing directly about $\mathbf{x}$, $\mathbf{x}$ is decomposed into $k = \lceil \log_2 \beta \rceil$ vectors of norm at most 1:

$$\mathbf{x} = \sum_{i=1}^{k} 2^{i-1}\mathbf{b_i}$$

In order to prevent the leakage of the $\mathbf{b_i}$, elements from $\{-1, 0, 1\}$ are added to the decomposed vectors, so the number of each of them is the same, producing $\mathbf{x^i} = (\mathbf{b_i}|\mathbf{t_i}) \in B_{3n}$. Finally the matrix A is also extended with $2n$ $\mathbf{0}$ columns $A' = (A|0^{2m\times n})$ such that:

$$A' \sum_{i=1}^{k} 2^{i-1}\mathbf{x^i} = \mathbf{y}$$

The prover now commits to

$$\begin{aligned} c_1 &= \mathsf{COM}\left(\pi_0, \ldots, \pi_{k-1}, A' \textstyle\sum_{i=1}^{k} 2^{i-1}\mathbf{r_i}\right) \\ c_2 &= \mathsf{COM}\left(\pi_0(\mathbf{r_0}), \ldots, \pi_{k-1}(\mathbf{r_{k-1}})\right) \\ c_3 &= \mathsf{COM}\left(\pi_0(\mathbf{r_0} + \mathbf{x^0}), \ldots, \pi_{k-1}(\mathbf{r_{k-1}} + \mathbf{x^{k-1}})\right) \end{aligned}$$

for random $\mathbf{r_0}, \ldots, \mathbf{r_{k-1}} \hookleftarrow \mathbb{Z}_q^{3n}$ and uniformly random permutations $\pi_0, \ldots, \pi_{k-1}$. Then, the verifier randomly chooses a challenge $i \hookleftarrow \{1, 2, 3\}$ and the prover reveals $c_j \forall j \neq i$. If c_2, c_3 are revealed, the prover will be convinced that the $\mathbf{x^i}$ are indeed small. In the other two cases the verifier will be able to validate either the left or the right-hand side of

$$A' \sum_{i=1}^{k} 2^{i-1}\mathbf{r_i} = A' \sum_{i=1}^{k} 2^{i-1}(\mathbf{r_i} + \mathbf{x^i}) - \mathbf{y},$$

giving the verifier confidence that the prover knows a preimage of $\mathbf{y}$. Since revealing all commitments would also reveal the $\mathbf{x}$, the above described process has to be repeated several times.

3. Proposed Quantum-Resistant DAA Techniques

As noticed in Section 2.1, DAA credentials correspond to a signature of the IoT identifier produced by the Issuer. In practice, DAA credentials are not held by a single party, but part of them are stored in the TPM, and another part in the IoT Host. Boyen's signature scheme is herein modified as described in Scheme 3 to achieve this key splitting. The Issuer's public-key now includes one

Scheme 3: Modified Boyen's Signature Scheme

- KeyGen(1^λ): samples one more uniform random vector of polynomials $\hat{A}_t \in \mathcal{R}_q^m$ than KeyGen in Scheme 1 and 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, $\mathsf{id} \in \{0,1\}^\ell$):
  1. Samples a vector of polynomials $\hat{Z}_t = (\mathbf{z}_1, \ldots, \mathbf{z}_m) \hookleftarrow \mathcal{D}^m_{\mathbb{Z}^n,s}$ 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 $\hat{A}_{\mathsf{id}} = [\hat{A}|\hat{A}_0 + \sum_{i=1}^{\ell} \mathsf{id}_i \cdot \hat{A}_i] \in \mathcal{R}_q^{2m}$, as in the Boyen scheme.
  3. Using the secret key $\hat{T}$, samples $\hat{Z}_h = (\mathbf{z}_{m+1} \ldots, \mathbf{z}_{3m}) \hookleftarrow \mathcal{D}_{L^{\perp}_{\mathbf{u}_h}(\hat{A}_{\mathsf{id}}),s}$, with $\|\hat{Z}_h\|_\infty \leq \beta$ and such that $\hat{A}_{\mathsf{id}} \cdot \hat{Z}_h \equiv \mathbf{u}_h = (\mathbf{u} - \mathbf{u}_t) \mod q$.
  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 $[\hat{A}_t|\hat{A}_{\mathsf{id}}] \cdot \hat{Z} \equiv \mathbf{u} \mod q$ and $\|\hat{Z}\|_\infty \leq \beta$ are satisfied, output 1, else 0.

more random vector of polynomials $\hat{A}_t \in \mathcal{R}_q$. Each signature is comprised of two vectors of polynomials $\hat{Z}_t$ and $\hat{Z}_h$ of small norm such that $[\hat{A}_t|\hat{A}_{\mathsf{id}}] \cdot [\hat{Z}_t|\hat{Z}_h] = \mathbf{u}$. $\hat{Z}_t$ is held by the TPM and $\hat{Z}_h$ by the Host. The security of this modified Boyen signature scheme is based on the original Boyen signature scheme which is unforgeable under the hardness assumption of the SIS problem [11]. The unforgeability of the modified Boyen signature can be reduced to the existential unforgeability of the original Boyen signature scheme. A detailed analysis of the security of the modified scheme can be found in Appendix A.

In order to create a DAA signature, which is jointly signed by a TPM and its Host, we modify Scheme 2 to allow for two parties to commit a set of secret values jointly. This modification is reflected in Scheme 4 and is based on the additive homomorphism of the scheme. Let $\hat{S}_t \in \mathcal{R}_q^{l_t}$ and $\hat{S}_h \in \mathcal{R}_q^{l_h}$, for some integers l_t and l_h, respectively be the TPM and the Host's inputs to be concatenated, and $\mathbf{s}_t$ and $\mathbf{s}_h$ in $\mathcal{R}_q$ be the TPM and the host's corresponding inputs to be added. With Scheme 4, the TPM and the Host are able to jointly commit to 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 Baum et al. scheme was proved to hold the properties of statistically hiding and computationally binding and the proof is based on an instantiation of the Ring-SIS problem. The security of this modified commitment scheme is based on the original scheme. We argue that splitting the prover role into two entities does not affect these two properties. A detailed security analysis of our modification is given in Appendix B.

Scheme 4: 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 the 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 $\|\hat{R}\|_\infty < \alpha_t+\alpha_h$.

3.1. Proposed LDAA Scheme

A DAA scheme supported on the above-described quantum-resistant cryptographic constructs is now proposed. The security of this scheme is based on the Ring-ISIS and Ring-LWE problems. Before proceeding with the description of the LDAA scheme, we define some standard functionalities that are used in the TPM technology, as specified in [16]. A detailed characterisation of these functionalities is presented in Appendix C of this paper.

- F_{ca} is a common certificate authority functionality that is available to all parties.
- $F_{crs}^{\mathcal{D}}$ 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.

The L-DAA scheme includes the *Setup*, *Join*, *Sign*, *Verify*, and *Link* processes. The *Setup* step is described in Scheme 5 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 5: LDAA Setup

- F_{crs} creates the system parameters: $\mathsf{sp} = (\lambda, q, n, m, \mathcal{R}_q, c, \beta, \beta', \ell, \eta)$, 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 $\mathsf{sid} = (\mathsf{I},\ \mathsf{sid}')$ for some sid', then creates its key pair. The Issuer's public key is $\mathsf{pp} = (\mathsf{sp}, \hat{A}_t, \hat{A}_I, \hat{A}_0, \hat{A}_1, ..., \hat{A}_\ell,$ $\mathbf{u},\ \mathcal{H}, \mathcal{H}_0, \mathcal{H}_1)$, 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 $\|\hat{T}_I\|_\infty \le \omega$, for some small real number ω. The Issuer initialises the list of joining members ($\mathsf{Members} \leftarrow \emptyset$) 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).

The Join process is a protocol running between the Issuer I and a platform, consisting of a TPM tpm_i and a Host host_j (with an identifier id). More than one Join session be may run in parallel. A unique sub-session identifier jsid is used and this value is given to all parties. The issuer I checks that the TPM-Host is qualified to execute the trusted computing attestation service, then issues a credential enabling the platform to create attestations. Via the unique session identifier jsid, the issuer can differentiate between various Join sessions that are executed simultaneously. A Join session works in two distinct phases, Join request and Join proceed. During the Join request, described in Scheme 6, the TPM generates its private-key $\hat{X}_t$ and the corresponding public-key $\mathbf{u}_t$, along with a a linking token 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 TPM-Host has not been provisioned before. The linking token enables the Issuer to ensure that no two TPMs hold the same private key in the Join proceed step. This prevents a signature from a TPM from tracing back to the signature of another TPM. Moreover, during Join proceed (see Scheme 7), 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.

After obtaining the credential from the Join process, tpm_i and host_j can sign a message μ with respect to a basename bsn. We use a unique sub-session identifier ssid to allow for multiple Sign sessions. Each session has two phases, Sign request and Sign proceed. While the first, described in Scheme 8, is mostly responsible for ensuring the TPM and the Host have compatible secret-key shares; in the second, described in Scheme 9, a zero-knowledge proof-of-knowledge of small $\hat{X}_t$ and $\hat{X}_h$ such that $[\hat{A}_t|\hat{A}_{\mathsf{id}}] \cdot [\hat{X}_t|\hat{X}_h] = \mathbf{u}$ is produced. More concretely, the

Scheme 6: 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 $\|\hat{X}_t\|_\infty \le \beta$, 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:

$$\pi_{\mathbf{u}_t} = \mathsf{SPK}\Big\{\mathsf{public} := \{\mathsf{sp},\ \hat{A}_t,\ \mathbf{u}_t,\ \mathsf{bsn}_I,\ \mathsf{nym}_I\},$$
$$\mathsf{witness} := \{\hat{X}_t = (\mathbf{x}_1, \ldots, \mathbf{x}_m),\ \mathbf{e}_I\}:$$
$$\mathbf{u}_t = \hat{A}_t \cdot \hat{X}_t \mod q \ \wedge\ \|\hat{X}_t\|_\infty \le \beta \ \wedge\ \mathsf{nym}_I = \mathcal{H}(\mathsf{bsn}_I) \cdot \mathbf{x}_1 + \mathbf{e}_I$$
$$\mod q \ \wedge\ \|\mathbf{e}_I\|_\infty \le \beta' \Big\}(\rho).$$

  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, (nym_I, $\pi_{\mathbf{u}_t}$), (sid, $\mathsf{tpm_i}$, I), jsid, $\mathsf{host_j}$).

- The host, upon receiving (APPEND, (nym_I, $\pi_{\mathbf{u}_t}$), (sid, $\mathsf{tpm_i}$, I)) from F_{auth^*}, forwards it to I by sending (APPEND, (nym_I, $\pi_{\mathbf{u}_t}$), (sid, $\mathsf{tpm_i}$, I)) to F_{auth^*} and keeps the state (jsid, $\mathbf{u_t}$, id).

- The Issuer, upon receiving (SENT, (nym_I, $\pi_{\mathbf{u}_t}$), (sid, $\mathsf{tpm_i}$, I), 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, $\pi_{\mathbf{u}_t}$).

Scheme 7: 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).

Scheme 8: 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).

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 $\hat{A}_h \cdot \hat{X}_h = \mathbf{u}_h$. Through the additive homomorphism of Scheme 4, the addition of these commitments results on commitments to strings showing that either $\hat{X}_t|\hat{X}_h$ is small or that $[\hat{A}_t|\hat{A}_{\mathsf{id}}] \cdot [\hat{X}_t|\hat{X}_h] = \mathbf{u}$. The opening of 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. The prover, by checking that the expected properties are verified for each opening, is convinced that it is communicating with a genuine TPM/Host pair. This proof is described in a detailed manner in Section 3.2. In addition, a nym tag is produced that is associated with the basename bsn.

The verify algorithm, described in Scheme 10, allows anyone to check whether a signature σ on a message μ with respect to a basename bsn is valid. Moreover, the link algorithm, depicted in Scheme 11 allows anyone to check whether two signatures (σ, μ) and (σ', μ') that were generated for the same basename bsn stem from the same TPM. This done by checking whether the difference between the two nym tags has a small norm.

3.2. The proofs θ_t, θ_h and π

In this section, the computation of the θ_t, θ_h and π is described in detail. The techniques employed herein can be seen as generalisations of those described in Section 2.5. While in the proof described in Section 2.5 the matrix A was public, here the matrix $[\hat{A}_t|\hat{A}_h]$ depends on the secret identifier id. We employ the techniques described in [17] of rewriting $[\hat{A}_t|\hat{A}_h] \cdot [\hat{X}_t|\hat{X}_h]$ as $[\hat{A}_t|\hat{A}_I|\hat{A}_0|\hat{A}_1|\ldots|\hat{A}_l] \cdot [\hat{X}_t|\hat{X}_{h_1}|\hat{X}_{h_2}|\mathsf{id}_1\hat{X}_{h_2}|\ldots|\mathsf{id}_l\hat{X}_{h_2}]$, 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 $[\hat{A}_t|\hat{A}_I|\hat{A}_0|\hat{A}_1|\ldots|\hat{A}_l]$, and extending and randomising id such that [15] 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

Scheme 9: 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 as described in Subsection 2.4:
  $$\theta_t = \mathsf{COM}\Big\{\mathsf{public} := \{\mathsf{sp},\ \hat{A}_t,\ \mathsf{nym},\ \mathsf{bsn},\ \mathcal{H},\ \mathbf{u}_t\},\\ \mathsf{witness} := \{\hat{X}_t = (\mathbf{x}_1, \ldots, \mathbf{x}_m), \mathbf{e}\} :\\ \{\hat{A}_t \cdot \hat{X}_t = \mathbf{u}_t\ \wedge\ \|\hat{X}_t\|_\infty \leq \beta\}\ \wedge\ \mathsf{nym} =\\ \mathcal{H}(\mathsf{bsn}) \cdot \mathbf{x}_1 + \mathbf{e}\ \wedge\ \|\mathbf{e}\|_\infty \leq \beta'\Big\}.$$
  If $\mathsf{bsn}{=}\perp$, 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 again as described in Subsection 2.4:
  $$\theta_h = \mathsf{COM}\Big\{\mathsf{public} := \{\mathsf{sp},\ \hat{A}_h,\ \mathbf{u}_h,\ \mu,\ \theta_t\},\\ \mathsf{witness} := \{\hat{X}_h = (\mathbf{x}_{m+1}, \ldots, \mathbf{x}_{3m}),\ \mathsf{id}\} :\\ \{\hat{A}_h \cdot \hat{X}_h = \mathbf{u}_h\ \wedge\ \|\hat{X}_h\|_\infty \leq \beta\}\Big\}.$$
  The combination of these two commitments θ_t and θ_h as described in Subsection 2.4 follows the additive homomorphic property of the commitment scheme.
  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 10: LDAA Verify

- Let RL denote a revocation list with all the rogue TPM's secret keys. 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:
  $$[\hat{A}_t|\hat{A}_h] \cdot \hat{X} = \mathbf{u} \wedge \|\hat{X}\|_\infty \leq \beta \wedge \\ \mathsf{nym} = \mathcal{H}(\mathsf{bsn}) \cdot \mathbf{x}_1 + \mathbf{e} \mod q \wedge \|\mathbf{e}\|_\infty \leq \beta'.$$
  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:
  $$\forall \mathbf{x}_1 \in RL, \|\mathcal{H}(\mathsf{bsn}) \cdot \mathbf{x}_1 - \mathsf{nym}\|_\infty \leq \beta'.$$
  3. If all checks passed, the verifier outputs (VERIFIED, ssid, 1), and (VERIFIED, ssid, 0) otherwise.

Scheme 11: LDAA Link

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

of elements in $\mathcal{R}_q$ into matrix-vector products. More concretely, we construct the matrices $\bar{A}_i = \mathsf{rot}(\mathbf{a}_i)$, as defined in [17], for $i = (1, 2, ..., (\ell + 3)m)$, for all polynomials $\mathbf{a}_i$ in $\hat{A}_t$, $\hat{A}_I$, $\hat{A}_0$, ..., $\hat{A}_\ell$, respectively, and the vectors $\bar{\mathbf{x}}_i$ whose entries are the coefficients of $\mathbf{x}_i$, for $i = (1, 2, \ldots, 3m)$, for all polynomials $\mathbf{x_i}$ in $\hat{X}_t$ and $\hat{X}_h$, respectively, such that 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 $(3 + l)m < i \leq (3 + 2l)m$.
- $\bar{\mathbf{x}}_{(2+i)m+j} = \mathsf{id}^*_i \cdot \bar{\mathbf{x}}_{2m+j}$ for $1 \leq i \leq 2\ell$ and $1 \leq j \leq m$.

Using techniques similar to those described in Section 2.5, the vectors $\bar{\mathbf{x}}_i$ and $\mathbf{e}$ are decomposed and extended into vectors of norm at most 1 such that $\bar{\mathbf{x}}_i = \sum_{d=1}^{k} 2^{d-1} \bar{\mathbf{x}}^d_i[1 : n]$ and $\mathbf{e} = \sum_{j=1}^{k} \mathbf{e}^j[1 : n] 2^{j-1}$, where $\bar{\mathbf{x}}^d_i[1 : n]$ and $\mathbf{e}^j[1 : n]$ correspond to the first n entries of $\bar{\mathbf{x}}^d_i$ and $\mathbf{e}^j$, respectively, and

$$\{\mathbf{e}^j\}_{j=1}^k, \{\bar{\mathbf{x}}^j_1\}_{j=1}^k, \{\bar{\mathbf{x}}^j_2\}_{j=1}^k, \ldots, \{\bar{\mathbf{x}}^j_{(3+2l)m}\}_{j=1}^k \in B_{3n},$$

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:

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

The commitment algorithm COM used to compute θ_t and θ_h is as explained in Scheme 4. To produce θ_t, the TPM samples the vectors $\{\mathbf{r}^j_{\mathbf{e}} \hookleftarrow \mathbb{Z}^{3n}_q\}_{j=1}^k$ and $\{\mathbf{r}^j_i \hookleftarrow \mathbb{Z}^{3n}_q\}_{j=1}^k$ 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: $D = [\mathsf{rot}(\mathcal{H}(\mathsf{bsn})) | \mathbf{0}] \in \mathbb{Z}^{n \times 3n}_q$, $\mathbf{v}^j_i = \mathbf{x}^j_i + \mathbf{r}^j_i$ and $\mathbf{v}^j_{\mathbf{e}} = \mathbf{e}^j + \mathbf{r}^j_{\mathbf{e}}$. Now, $\theta_t = (\mathbf{C}_{t1}, \mathbf{C}_{t2}, \mathbf{C}_{t3})$ is computed with:

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

- $\mathbf{C}_{t2} = \mathsf{COM}(\{\phi_j(\mathbf{r}_1^j), \ldots, \phi_j(\mathbf{r}_m^j)\}_{j=1}^k, \{\varphi_j(\mathbf{r}_{\mathbf{e}}^j)\}_{j=1}^k)$.

- $\mathbf{C}_{t3} = \mathsf{COM}(\{\phi_j(\mathbf{v}_1^j), \ldots, \phi_j(\mathbf{v}_m^j)\}_{j=1}^k, \{\varphi_j(\mathbf{v}_{\mathbf{e}}^j)\}_{j=1}^k)$.

In a similar fashion, the Host samples the 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 $\hat{X}_{h_2}$. It 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 $\theta_t = (\mathbf{C}_{h1}, \mathbf{C}_{h2}, \mathbf{C}_{h3})$ is computed:

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

- $\mathbf{C}_{h2} = \mathsf{COM}(\{\delta_j(\mathbf{r}_{m+1}^j), \cdots,\ \delta_j(\mathbf{r}_{2m}^j),\ \psi_j(\mathbf{r}_{2m+1}^j), \ldots,\ \psi_j(\mathbf{r}_{3m}^j), \psi_j(\mathbf{r}_{(\tau(1)+2)m+1}^j),$ $\ldots, \psi_j(\mathbf{r}_{(\tau(1)+3)m}^j), \ldots, \psi_j(\mathbf{r}_{(\tau(2\ell)+2)m+1}^j), \ldots, \psi_j(\mathbf{r}_{(\tau(2\ell)+3)m}^j)\}_{j=1}^k, \tau(\mathbf{r}_{\mathsf{id}^*}))$.

- $\mathbf{C}_{h3} = \mathsf{COM}(\{\delta_j(\mathbf{v}_{m+1}^j), \ldots, \delta_j(\mathbf{v}_{2m}^j), \psi_j(\mathbf{v}_{2m+1}^j), \ldots, \psi_j(\mathbf{v}_{3m}^j), \psi_j(\mathbf{v}_{(\tau(1)+2)m+1}^j),$ $\ldots, \psi_j(\mathbf{v}_{(\tau(1)+3)m}^j), \ldots, \psi_j(\mathbf{v}_{(\tau(2\ell)+2)m+1}^j), \ldots, \psi_j(\mathbf{v}_{(\tau(2\ell)+3)m}^j)\}_{j=1}^k, \tau(\mathbf{v}_{\mathsf{id}^*}))$.

The proof π is computed using a strategy similar to Section 2.5, but where the commitments are produced using the homomorphic properties of Scheme 4. Since multiple iterations of the process described in Section 2.5 are necessary to achieve high soundness, $\mathsf{tpm_i}$ 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 [18], instead of directly storing the $\mathbf{C}_1$, $\mathbf{C}_2$ and $\mathbf{C}_3$ values in π we opt to store an hash of them instead. A significant reduction in the proof size is achieved. Challenges are generated following a 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:

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

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

4. Security Model of DAA

In this paper, we adapt the security model for DAA given by Camenish et al. in [16]. The security definition is given in the Universal Composability (UC) model, represented in Figure 2, with respect to an ideal functionality F_{daa}^l. 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_{daa}^l, which internally performs all the required tasks and creates the parties' outputs.

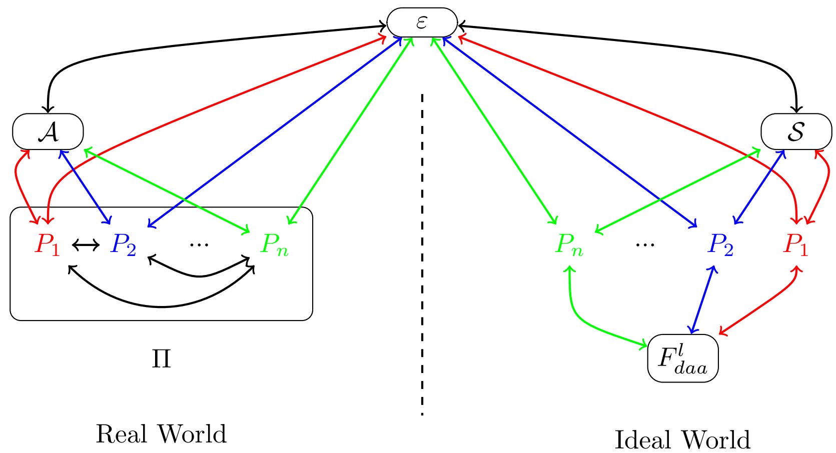

Figure 2: Universal composability security model: the real and the ideal world executions are indistinguishable to the environment ε.

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. In general, the security properties that a DAA scheme should enjoy are the following:

- *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.

- *Non-frameability*: This requires that the entire platform ($\mathsf{tpm_i} + \mathsf{host_j}$) 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 was built on the honesty of the entire platform, i.e., both the TPM and

Scheme 12: Ideal Setup

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

- Verify that $(I,\ \mathsf{sid}') = \mathsf{sid}$ 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.

the host are supposed to be honest. In [19] it is considered that the TPM may be corrupt and privacy must hold whenever the host is honest, regardless of the corruption state of the TPM. In order to achieve the best performance, we do not consider this case in this work and leave it for a future work.

4.1. The Ideal Functionality F^l_{daa}

The ideal functionality F^l_{daa} is now formally defined under the assumption of static corruption, i.e., the adversary decides beforehand which parties are corrupt and informs F^l_{daa} about them. F^l_{daa} has five interfaces (Setup, Join, Sign, Verify, Link) described in Schemes 12, 13, 14, 15 and 16. Scheme 12 provides a functionality akin to Scheme 5; 13 to 6 and 7; 14 to 8 and 9; 15 to 10; and 16 to 11. 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. $\mathsf{sid} = (\mathsf{sid}', \mathsf{I})$. We also uniquely identify the Join and Sign sub-sessions with jsid and ssid. F^l_{daa} is parameterized by a leakage function $l : \{0,1\}^* \to \{0,1\}^*$, which models the information leakage that occurs in the communication between a host $\mathsf{host_j}$ and a TPM $\mathsf{tpm_i}$. We also define the following algorithms that are used in Schemes 12, 13, 14, 15 and 16:

- Kgen(1^λ): A probabilistic algorithm that takes a security parameter λ and generates keys gsk for honest TPMs.
- sig($gsk,\ \mu,\ \mathsf{bsn}$): 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($\sigma,\ \mu,\ \mathsf{bsn}$): 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($\sigma_1,\ \mu_1,\ \sigma_2,\ \mu_2,\ \mathsf{bsn}$): 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.

Scheme 13: 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^l_{daa} proceeds as follows:
   - Create a join session $\langle \mathsf{jsid},\ \mathsf{tpm_i},\ \mathsf{host_j}, \text{request}\rangle$.
   - 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 $\langle \mathsf{jsid},\ \mathsf{tpm_i},\ \mathsf{host_j}, \text{delivery}\rangle$.
   - 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^l_{daa} updates the session record to $\langle \mathsf{jsid},\ \mathsf{sid},\ \mathsf{tpm_i},\ \mathsf{host_j}, \text{complete}\rangle$.
   - 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 14: 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^l_{daa} 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 $\langle \mathsf{ssid},\ \mathsf{tpm_i},\ \mathsf{host_j},\ \mu,\ \mathsf{bsn}, \text{delivered} \rangle$.
   - 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 $\mathsf{bsn} \neq \perp$, then retrieve gsk from the $\langle \mathsf{tpm_i},\ \mathsf{bsn},\ gsk \rangle \in$ DomainKeys.
   - If $\mathsf{bsn} = \perp$ 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 15: 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 $\mathsf{tpm_i}$ was found, but no entry $\langle\star$, μ, $\mathsf{tpm_i}$, $\mathsf{bsn}\rangle$ 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 $\mathsf{tpm_i}$ was found [Check XII].

- If $b \neq 0$, set $b \leftarrow$ver(σ, μ, bsn) [Check XIII].

- Add $\langle\sigma$, μ, bsn, RL, $b\rangle$ to VerResults, and output (VERIFIED, sid, b) to V.

Scheme 16: Ideal Link

On input (LINK, sid, σ_1, μ_1, σ_2, μ_2, bsn), with $\mathsf{bsn} \neq \perp$, 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 (σ_1, μ_1, bsn) or (σ_2, μ_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(σ_1, μ_1, bsn, gsk_i) and b'_i= identify(σ_2, μ_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(σ_1, μ_1, σ_2, μ_2, bsn), then output (LINK, sid, f) to V.

- identify($gsk, \sigma,\ \mu,\ \mathsf{bsn}$): A deterministic algorithm that will be used to ensure consistency with the ideal functionality F^{l}_{daa}'s internal records. It outputs 1 if a key gsk was used to produce a signature σ, 0 otherwise.

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

1. CkeckGskHonest(gsk): If the $\mathsf{tpm_i}$ 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. CkeckGskCorrupt(gsk): If the $\mathsf{tpm_i}$ is corrupt and $\nexists gsk' \neq gsk$ and ($\mu,\ \sigma,\ \mathsf{bsn}$) 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.

5. Security Proof

In this section, we provide a sketch of the security proof. The detailed proof can be found in Appendix D. A sequence of games based on the model of Camenish et al. in [16] is presented, and it is shown that there exists no environment ε that can distinguish the real world protocol Π with an adversary $\mathcal{A}$, from the ideal world F^{l}_{daa} with a simulator $\mathcal{S}$. Starting with the real world protocol game, we change the protocol game by game in a computationally indistinguishable way, finally ending with the ideal world protocol. The sequence of games is as follows:

$\boxed{\text{Game 1}}$ ▷ The Real World Protocol.

↓

$\boxed{\text{Game 2}}$ ▷ 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.

↓

$\boxed{\text{Game 3}}$ ▷ 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.

↓

$\boxed{\text{Game 4}}$ ▷ 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.*

↓

$\boxed{\text{Game 5}}$ ▷ 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 $l(\mu', \mathsf{bsn}')$=$l(\mu, \mathsf{bsn})$.*

$\boxed{\text{Game 9}}$

$\triangleright$ 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.

$\boxed{\text{Game 10}}$

$\triangleright$ 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.

$\boxed{\text{Game 11}}$

$\triangleright$ 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.*

$\boxed{\text{Game 12}}$

$\triangleright$ 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 $\mathsf{nym_I} = \mathcal{H}(\mathsf{bsn}) \cdot \mathbf{x}_1 + \mathbf{e}_\mathsf{I}$, two different gsks can't share the same $\mathbf{x}_1$.

$\boxed{\text{Game 13}}$

$\triangleright$ 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.

$\boxed{\text{Game 14}}$

$\triangleright$ 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.

$\boxed{\text{Game 15}}$ $\triangleright$ 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.

$\boxed{\text{Game 16}}$ $\triangleright$ F now includes all the functionalities of F^l_{daa}. $\square$

All the remaining checks of the ideal functionality F^l_{daa} 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.

6. Related Work

The DAA schemes used in the current TPM standards are based on either the factorisation problem in the RSA setting or the discrete logarithm problem in the Elliptic-Curve (EC) setting. The DAA protocol was firstly conceptualised in 2004 by Brickell, Camenisch, and Chen [20]. [20] proposed a scheme based on the difficulty of RSA that was standardised in TPM 1.2. Later, Brickell, Chen and Li proposed the first EC-DAA scheme based on symmetric pairings [21, 22]. Two EC-DAA schemes, based on asymmetric (Type 3) pairings, have later been proposed to improve the performance of [21, 22], and are supported by TPM 2.0 [23, 24, 25]. Since the factorisation and discrete logarithm problems are known to be vulnerable to quantum computer attacks [26], all the standardised DAA schemes will not be secure in the post-quantum computer age. Recently, El Bansarkhani and El Kaafarani [7] proposed the first post-quantum direct anonymous attestation scheme from lattice assumptions. However, the scheme requires massive storage and computation resources. Section 6.1 gives a brief overview of this scheme. Since [7] is the only post-quantum DAA scheme available in related art, the following sections focus on the comparison between the scheme herein proposed and [7].

6.1. El Bansarkhani and El Kaafarani DAA Scheme [7]

The DAA scheme proposed in [7] works as follows. The Issuer's public key consists of $\ell + 2$ vectors in $\mathcal{R}_q^m$, namely $\hat{A}_I$, $\hat{A}_i$ for $i = 0, 1, \cdots \ell$, and 2 polynomials $\mathbf{u}$ and $\mathbf{b} \in \mathcal{R}_q$. During the join step, the TPM generates a small secret $\hat{Z}_1 \in \mathcal{R}_q^{2m+1}$ such that $[\mathbf{b}|\hat{A}_{\mathsf{id}}] \cdot [\hat{Z}_1] = \tilde{\mathbf{u}} \mod q$. The TPM sends $\tilde{\mathbf{u}}$ together with a proof of knowledge π_1 to the issuer, who registers both $\tilde{\mathbf{u}}$ and the corresponding TPM, and samples (using his secret key) a small credential $\hat{Z}_2$ such that $\hat{A}_{\mathsf{id}} \cdot \hat{Z}_2 = \mathbf{u} - \tilde{\mathbf{u}} \mod q$. $\hat{Z}_2$ corresponds to the host secret-key

Schemes	This paper	Scheme in [7]
TPM's Secret key	mn	$(2m+1)n$
Credential	$2mn$	$2mn$
Issuer's Secret Key	m^2n	m^2n
Signature	$c\mathcal{O}(n)[km\,((2\ell+3)+1)]$	$2ckm\mathcal{O}(n)(2\ell+2)$
Verification key	$((\ell+3)m+1)\,n$	$((\ell+2)m+2)\,n$

Table 2: Keys and signature sizes in terms of the number of elements in $\mathbb{Z}_q$. Our main contribution is reducing the TPM's secret key size (less than half the size in [7]), as well as the signature size.

share. The TPM and the host secret-key shares together satisfy $\mathbf{u} = [\mathbf{b}|\hat{A}_{\mathsf{id}}] \cdot [\hat{Z}_1 + (\mathbf{0}|\hat{Z}_2)]$. To create a signature, the TPM samples a small random vector $\hat{T} \in \mathcal{R}_q^{2m}$, such that $\hat{T} \cdot \hat{A}_{\mathsf{id}} \mod q$ is uniform, and shares it with the host. $\hat{T}$ is used to randomise signatures. Then, the TPM and the host independently compute π_2 and π_3, where π_2 proves that $\mathbf{u}' = [\mathbf{b}|\hat{A}_{\mathsf{id}}] \cdot [\hat{Z}_1 + (\mathbf{0}|\hat{T})]$ and π_3 proves that $\mathbf{u} - \mathbf{u}' = \hat{A}_{\mathsf{id}} \cdot (\hat{Z}_2 - \hat{T})$. Notice that, in [7], commitments are not hashed before being stored in π_2 or π_3. Finally, the host outputs the signature $\sigma = (\pi_2,\ \pi_3,\ \mathbf{u}',\ \mu)$.

6.1.1. Size Comparison

In our LDAA scheme, the TPM's secret key size is reduced to m polynomials in $\mathcal{R}_q$, instead of $2m+1$ polynomials in [7]. Such a change significantly reduces the TPM's computation costs in the join and sign interfaces, as well as the TPM's key and the signature sizes. For instance, in the proposed scheme, the LDAA signature includes c responses to the Fiat-Shamir challenges, where each response is comprised of $\mathcal{O}(n)km(2\ell+2)$ elements in $\mathbb{Z}_q$ provided by the host and $\mathcal{O}(n)k(m'+1)$ provided by the TPM. In [7], the size of the response for each round is bounded by $\mathcal{O}(n)km(2\ell+2)$ elements in $\mathbb{Z}_q$ for both the host and the TPM. Thus in our L-DAA scheme, the signature's size has been significantly reduced especially for large ℓ. Moreover, the hashing of the commitments in π further reduces the sizes of the signatures. The verification key set in [7] consists of the $\ell+2$ vectors of polynomials $\hat{A}_I,\ \hat{A}_i$ for $i = 0, 1, \cdots \ell$ and two polynomials $\mathbf{u}$ and $\mathbf{b}$. In our L-DAA scheme, we add $\hat{A}_t$ to the verification key set resulting in $\ell+2$ vectors of polynomials in $\mathcal{R}_q^m$, a vector of polynomials $\hat{A}_t \in \mathcal{R}_q^m$ and a polynomial $\mathbf{u}$. Table 2 compares the space efficiency between the proposed LDAA scheme and the scheme presented in [7].

6.1.2. Computation Costs

To generate the values to be committed for one round of $\pi_{\mathbf{u}_t}$ and θ_t in the join and sign interfaces of our LDAA scheme, the TPM has to perform at most $m+1$ polynomial multiplications. In [7], the TPM performs at most $2m+2$ polynomial multiplications for generating the values to be committed for each round of π_1 and π_2 in the join and sign interfaces, respectively. The computational cost for the host is of $2m$ polynomial multiplications for checking the equality $\mathbf{u}_h = \hat{A}_h \cdot \hat{X}_h$ in the join interface, and $2m$ polynomial multiplications

	Join		Sign		Verify	
	Ours	In [7]	Ours	In [7]	Ours	In [7]
TPM	$m+1$	$2m+2$	$m+1$	$2m+2$	-	-
Host	$2m$	$2m$	$2m$	$2m$	-	-
Issuer	$m+1$	$2m+2$	-	-	-	-
Verifier	-	-	-	-	$3m+1$	$4m+2$

Table 3: This table compares the computation costs for the generation of the values to be committed in both schemes, represented by the total number of polynomial multiplications in $\mathcal{R}_q$ for each round of the signatures. The table shows that the computation costs in our LDAA scheme are significantly reduced for the TPM during joining and signing, and for the issuer and the verifier.

for generating the values to be committed for each round of θ_h and π_3 in the sign interfaces for both schemes. The Issuer verifies the responses for each round of $\pi_{\mathbf{u}_t}$ and π_1 in both schemes in the join interface. Thus, the issuer's computation cost for each round is bounded by $m+1$ polynomial multiplications for our L-DAA scheme and $2m+2$ in [7]. The verifier validates both the TPM and the host's responses. Therefore, the verifier's computation cost in our L-DAA scheme is $1+3m$ polynomial multiplications. In [7], the verifier's computation cost is $4m+2$. All in all, the computational complexity for both the joining and the signing is approximately halved for the TPM, and verification is accelerated $4/3$ times, when the proposed scheme is compared to that of [7].

7. Experimental Results

In order to evaluate the proposed LDAA scheme, both the proposed scheme and [7] were described in C. Basic operations, such as the Number Theoretic Transform (NTT)-based multiplication over $\mathcal{R}_q$ and the Issuer's Gaussian sampling, were implemented once and shared between the two schemes. The implementations made use of the following cryptographic parameters: $n = 256$, $q = 8380417$, $l = 32$, $m = 24$ and $\beta = 256$. Moreover, commitments based on Baum et al's proposal were used for both cases. Since all entities, namely the TPM, the Issuer, the Host and the Verifier, were executed in the same platform, we have opted to run the experiments on an Intel i9 7900X CPU with 64GB running at 3.3 GHz operated by CentOS 7.5. The code was compiled with gcc 4.8.5 with the `-Ofast` and `-march=native` flags. The experimental results herein presented focus on the signing and on the verification operations since they are the most often executed, and mostly target at comparing the proposed scheme with [7].

Fig. 3 shows that in practice the size of the TPM private-key share is halved when the proposed scheme is compared with [7]. This is of particular importance for TPM platforms, where memory resources are constrained. In Fig. 4, signature sizes are shown for the proposed scheme, for the proposed scheme when commitments are not hashed, and for [7]. As predicted in Section 6.1.1,

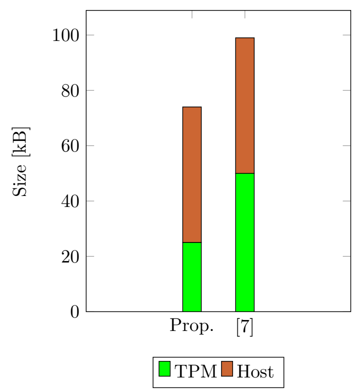

Figure 3: Private-key material for the proposed scheme and [7]

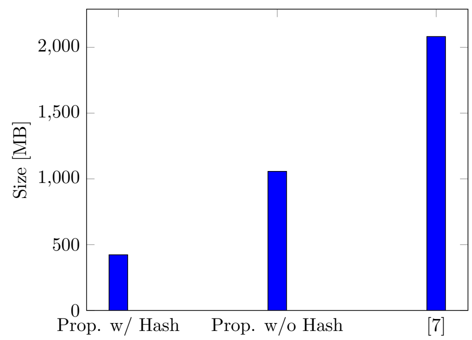

Figure 4: Signature size for the proposed scheme with hashed commitments (Prop. w/ Hash), with regular commitments (Prop. w/o Hash) and [7]

the signature size of the proposed scheme is halved, when hashing is not considered, in comparison to the scheme in [7]. This improvement is achieved through the exploitation of the modified Baum et al's commitment scheme, enabling the construction of a single proof of knowledge that reflects the secret-key shared between the TPM and Host, instead of the two required by [7]. Moreover, by hashing commitments, the size of the proofs are themselves reduced, resulting in signatures that are 5 times smaller than [7].

While in Section 6.1.2 it was predicted that the TPM computation during signing would asymptotically be twice as fast with the proposed scheme than with [7], Fig. 5 shows that, in practice, other operations, such as the computation of the commitments, reduce the speed-up to 1.13. Nevertheless, the verification operation is significantly enhanced when comparing the proposed scheme with [7], as shown in Fig. 6. Since a single proof needs to be verified instead of two, speed-ups of 2.04 are achieved. Notice that in Figures 5 and 6 the execution times include the hashing of the commitments.

8. Conclusion and Future Work

The growing IoT infrastructure requires scalable and reliable networks. While solutions based on the CE paradigm solve part of the problem, by having Edge devices servicing small networks of IoT devices, current solutions ensuring the trustworthiness of the network based on TPMs might not be secure in the long-term. DAA protocols enable IoT systems to prove their genuineness and trustworthy state to an Edge node in an anonymous way. However, currently standardised DAA schemes are susceptible to post-quantum attacks. While [7] has recently proposed a lattice-based DAA protocol achieving quantum-resistance, it is computationally cumbersome, since both the TPM and the Host need to individually generate large proofs-of-knowledge about their secret-key shares. In contrast, a novel commitment scheme is herein proposed allowing for the

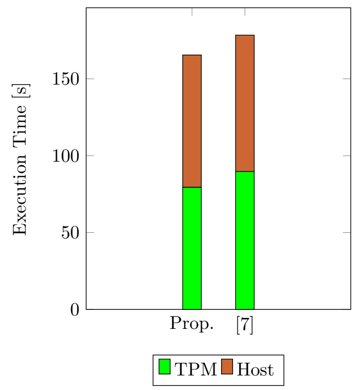

Figure 5: Signing performance for the proposed scheme and [7]

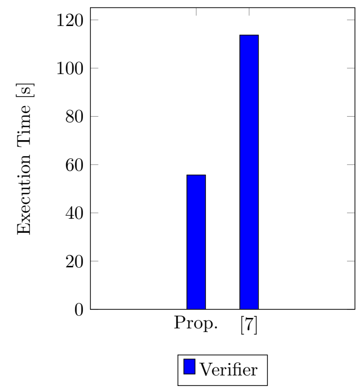

Figure 6: Verification performance for the proposed scheme and [7]

construction of commitments to values shared between the TPM and the Host. Building on this technique, we are then able to propose a lattice-based DAA scheme wherein the TPM and the Host interact to produce a single proof-of-knowledge about their shared secret-key. The proposed protocol is proved to be secure in the UC security model. Experimental results show that the resulting scheme reduces the storage requirements of the TPM twofold and the signature size 5 times. Moreover, the signing and verification operations are accelerated 1.1 and 2.0 times, respectively. Future work will focus on further optimising the proposed scheme, to make it even more suitable to hardware implementations and IoT applications.

References

[1] S. A. Rotondo, Trusted Computing Group, Springer US, Boston, MA, 2011, pp. 1331–1331. doi:10.1007/978-1-4419-5906-5_498. URL https://doi.org/10.1007/978-1-4419-5906-5_498

[2] W. Arthur, D. Challener, A Practical Guide to TPM 2.0: Using the Trusted Platform Module in the New Age of Security, 1st Edition, Apress, Berkely, CA, USA, 2015.

[3] L. Lorenzin, A. Shah, Trusted network communications, http://trustedcomputinggroup.org/work-groups/trusted-network-communications/.

[4] Tpm from pcs to the iot, https://trustedcomputinggroup.org/tpm-pcs-iot/, accessed: 2018-11-28 (mar 2017).

[5] Trusted network connect (tnc) howto, https://wiki.strongswan.org/projects/1/wiki/trustednetworkconnect (2018).

[6] Commercial National Security Algorithm Suite and Quantum Computing FAQ, Tech. rep., National Security Agency/Central Security Service (jan 2016).
URL https://cryptome.org/2016/01/CNSA-Suite-and-Quantum-Computing-FAQ.pdf

[7] R. E. Bansarkhani, A. E. Kaafarani, Direct anonymous attestation from lattices, Cryptology ePrint Archive, Report 2017/1022, https://eprint.iacr.org/2017/1022 (2017).

[8] J. Hoffstein, J. Pipher, J. Silverman, An Introduction to Mathematical Cryptography, 1st Edition, Springer Publishing Company, Incorporated, Springer-Verlag New York, 2008.

[9] C. Peikert, A decade of lattice cryptography, Foundations and Trends® in Theoretical Computer Science 10 (4) (2016) 283–424. doi:10.1561/0400000074.
URL http://dx.doi.org/10.1561/0400000074

[10] O. Regev, The learning with errors problem (invited survey). doi:10.1109/CCC.2010.26.

[11] X. Boyen, Lattice mixing and vanishing trapdoors: A framework for fully secure short signatures and more, in: P. Q. Nguyen, D. Pointcheval (Eds.), Public Key Cryptography – PKC 2010, Springer Berlin Heidelberg, Berlin, Heidelberg, 2010, pp. 499–517.

[12] D. Micciancio, C. Peikert, Trapdoors for lattices: Simpler, tighter, faster, smaller, in: D. Pointcheval, T. Johansson (Eds.), Advances in Cryptology – EUROCRYPT 2012, Springer Berlin Heidelberg, Berlin, Heidelberg, 2012, pp. 700–718.

[13] R. W. F. Lai, H. K. F. Cheung, S. S. M. Chow, Trapdoors for ideal lattices with applications, in: D. Lin, M. Yung, J. Zhou (Eds.), Information Security and Cryptology, Springer International Publishing, Cham, 2015, pp. 239–256.

[14] C. Baum, I. Damgård, V. Lyubashevsky, S. Oechsner, C. Peikert, Efficient commitments and zero-knowledge protocols from ring-sis with applications to lattice-based threshold cryptosystems, Cryptology ePrint Archive, Report 2016/997, https://eprint.iacr.org/2016/997 (2016).

[15] S. Ling, K. Nguyen, D. Stehlé, H. Wang, Improved zero-knowledge proofs of knowledge for the isis problem, and applications, in: K. Kurosawa, G. Hanaoka (Eds.), Public-Key Cryptography – PKC 2013, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 107–124.

[16] J. Camenisch, M. Drijvers, A. Lehmann, Universally composable direct anonymous attestation, in: C.-M. Cheng, K.-M. Chung, G. Persiano, B.-Y. Yang (Eds.), Public-Key Cryptography – PKC 2016, Springer Berlin Heidelberg, Berlin, Heidelberg, 2016, pp. 234–264.

[17] S. Ling, K. Nguyen, H. Wang, Group signatures from lattices: Simpler, tighter, shorter, ring-based, in: J. Katz (Ed.), Public-Key Cryptography – PKC 2015, Springer Berlin Heidelberg, Berlin, Heidelberg, 2015, pp. 427–449.

[18] G. Poupard, J. Stern, Short proofs of knowledge for factoring, in: H. Imai, Y. Zheng (Eds.), Public Key Cryptography, Springer Berlin Heidelberg, Berlin, Heidelberg, 2000, pp. 147–166.

[19] J. Camenisch, L. Chen, M. Drijvers, A. Lehmann, D. Novick, R. 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), 2017, pp. 901–920. doi:10.1109/SP.2017.22.

[20] E. Brickell, J. Camenisch, L. Chen, Direct anonymous attestation, in: Proceedings of the 11th ACM Conference on Computer and Communications Security, CCS '04, ACM, New York, NY, USA, 2004, pp. 132–145. doi:10.1145/1030083.1030103.
URL http://doi.acm.org/10.1145/1030083.1030103

[21] E. Brickell, L. Chen, J. Li, A new direct anonymous attestation scheme from bilinear maps, in: Proceedings of the 1st International Conference on Trusted Computing and Trust in Information Technologies: Trusted Computing - Challenges and Applications, Trust '08, Springer-Verlag, Berlin, Heidelberg, 2008, pp. 166–178. doi:10.1007/978-3-540-68979-9_13.
URL http://dx.doi.org/10.1007/978-3-540-68979-9_13

[22] E. Brickell, L. Chen, J. Li, Simplified security notions of direct anonymous attestation and a concrete scheme from pairings, International Journal of Information Security 8 (5) (2009) 315–330. doi:10.1007/s10207-009-0076-3.
URL https://doi.org/10.1007/s10207-009-0076-3

[23] E. Brickell, J. Li, A pairing-based daa scheme further reducing tpm resources, in: Proceedings of the 3rd International Conference on Trust and Trustworthy Computing, TRUST'10, Springer-Verlag, Berlin, Heidelberg, 2010, pp. 181–195.
URL http://dl.acm.org/citation.cfm?id=1875652.1875665

[24] L. Chen, J. Li, Flexible and scalable digital signatures in tpm 2.0, in: Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security, CCS '13, ACM, New York, NY, USA, 2013, pp. 37–48. doi:10.1145/2508859.2516729.
URL http://doi.acm.org/10.1145/2508859.2516729

[25] L. Chen, D. Page, N. P. Smart, On the design and implementation of an efficient DAA scheme, in: Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2010, pp. 223–237. doi:10.1007/978-3-642-12510-2_

16.
URL https://doi.org/10.1007%2F978-3-642-12510-2_16

[26] P. W. Shor, Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer, SIAM J. on Computing (1997) 1484–1509.

[27] D. Cash, D. Hofheinz, E. Kiltz, C. Peikert, Bonsai trees, or how to delegate a lattice basis, J. Cryptol. 25 (4) (2012) 601–639. doi:10.1007/s00145-011-9105-2.
URL http://dx.doi.org/10.1007/s00145-011-9105-2

Appendix A. Security Proof of the Modified Boyen Signture Scheme

In this section we examine a signature scheme (described in Scheme 3) based on the one from [11]. We claim that the same security result applies to this scheme as in the one in [11] except that here the security of the scheme reduces to the hardness of solving the *inhomogeneous*-SIS problem.

Theorem 1. *For a prime modulus $q = q(\lambda)$, if there is a probabilistic algorithm $\mathcal{A}$ that outputs an existential signature forgery, with probability ϵ, in time τ, and making $Q \leq q/2$ adaptive chosen-message queries, then there is a probabilistic algorithm $\mathcal{B}$ that solves the (q, n, m, β)-ISIS problem in time $\tau' \approx \tau$ and with probability $\epsilon' \geq \epsilon/(3q)$, for some polynomial function $\beta = \mathrm{poly}(\lambda)$.*

Proof. We begin by assuming that there is such a forger $\mathcal{A}$. Using the power of $\mathcal{A}$, we construct a solver $\mathcal{B}$ that simulates the attack environment for $\mathcal{A}$ and uses the forgery produced by $\mathcal{A}$ to create an ISIS solution. $\mathcal{B}$ does the following.

Invocation: $\mathcal{B}$ receives the random (q, n, m, β)-ISIS problem instance in the form of a uniformly random matrix $A_0 \in \mathbb{Z}_q^{n\times m}$ and a uniform vector $\mathbf{u} \in \mathbb{Z}_q^n$, and must find $\mathbf{e}_0 \in \mathbb{Z}^m$ with $\|\mathbf{e}_0\|_\infty \leq \beta$ and $A_0\mathbf{e}_0 = \mathbf{u} \mod q$.

Setup:

1. Pick uniformly random $B_0 \in \mathbb{Z}_q^{n\times m}$ with associated short trapdoor matrix $T_{B_0} \subset \wedge^\perp(B_0)$.
2. Pick $l + 2$ short matrices $R_t, R_0, ..., R_l \in \mathbb{Z}_q^{m\times m}$.
   -Do so by sampling the columns from $\mathcal{D}_{\mathbb{Z}^m,\eta}$.
3. Define $A_t := A_0R_t$. Pick a random vector $\mathbf{d}_t \in \mathbb{Z}^m$ and compute $A_t\mathbf{d}_t =: \mathbf{u}_t \mod q$.
4. Pick $l + 1$ random scalars $h_0, ..., h_l \in \mathbb{Z}_q$ and set $h_0 = 1$.
5. Output the verification key

$$VK = [A_t, A_0, C_0 = (A_0R_0 + h_0B_0), ..., (A_0R_l + h_lB_0)].$$

Queries: Now $\mathcal{A}$ requests signature queries on any message msg which $\mathcal{B}$ answers as follows.

1. Compute the matrix $R_{\mathrm{msg}} = \sum_{i=0}^{l}(-1)^{\mathrm{msg}[i]}R_i$.
2. Compute the scalar $h_{\mathrm{msg}} = \sum_{i=0}^{l}(-1)^{\mathrm{msg}[i]}h_i$. If $h_{\mathrm{msg}} = 0$, abort the simulation.
3. Setting

$$\begin{aligned} F &= [A_0|\sum_{i=0}^{l}(-1)^{\mathrm{msg}[i]}C_i] \\ &= [A_0|A_0R_{\mathrm{msg}} + h_{\mathrm{msg}}B_0], \end{aligned}$$

   sample $\mathbf{d}_h \in \mathbb{Z}^{2m}$ such that $F \cdot \mathbf{d}_h = \mathbf{u}_h := (\mathbf{u} - \mathbf{u}_t) \mod q$ and $\|\mathbf{d}_h\|_\infty \leq \beta$. Write $\mathbf{d}_h = [\mathbf{d}_{h_0}, \mathbf{d}_{h_1}]$, where $\mathbf{d}_{h_0}, \mathbf{d}_{h_1} \in \mathbb{Z}^m$.
   -Do so by taking the trapdoor T_{B_0} and delegating this to one for the matrix F via standards methods [27].

4. Output the signature $\mathbf{d} = \begin{bmatrix} \mathbf{d}_t^T \\ \mathbf{d}_{h_0}^T \\ \mathbf{d}_{h_1}^T \end{bmatrix} \in \mathbb{Z}^{3m}$.

Forgery: After providing $\mathcal{A}$ with signatures on the queried messages, $\mathcal{A}$ produces a forged signature $\mathbf{d}^*$ on a new (unqueried) message msg*. $\mathcal{B}$ then does the following.

1. Compute the matrix $R_{\text{msg}^*} = \sum_{i=0}^{l} (-1)^{\text{msg}^*[i]} R_i$.
2. Compute the scalar $h_{\text{msg}^*} = \sum_{i=0}^{l} (-1)^{\text{msg}^*[i]} h_i$. If $h_{\text{msg}^*} \neq 0$, abort the simulation.
3. Assuming $h_{\text{msg}^*} = 0$, we have that

$$\begin{aligned} \mathbf{u} &= [A_t | A_0 | A_0 R_{\text{msg}^*} + h_{\text{msg}^*} B_0] \cdot \mathbf{d} \mod q, \\ &= [A_0 R_t | A_0 | A_0 R_{\text{msg}^*}] \cdot \begin{bmatrix} \mathbf{d}_t^T \\ \mathbf{d}_{h_0}^T \\ \mathbf{d}_{h_1}^T \end{bmatrix} \mod q. \end{aligned}$$

Setting $\mathbf{e}_0 = R_t \cdot \mathbf{d}_t + \mathbf{d}_{h_0} + R_{\text{msg}^*} \cdot \mathbf{d}_{h_1}$ we have that $A_0 \mathbf{e}_0 = \mathbf{u} \mod q$. We claim that at this point $\mathcal{B}$ has found a (q, n, m, β)-ISIS solution. □

All that remains to show is that

- $\mathbf{e}_0$ is small and non-zero with good probability and therefore a valid ISIS solution for the stated approximation.

- The completion probability of this procedure (without aborts) is substantial against an arbitrary attack method for $\mathcal{A}$.

The first of these points is covered by the discussion of Lemma 26 in [11]. A slight modification needs to be made to the parameter β. In particular, we have that with overwhelming probability $\|\mathbf{e}_0\|_\infty \leq \beta$ for $\beta = \text{poly}(l, n, m) = \text{poly}(\lambda)$ provided we set,

$$\beta = (1 + (1 + \sqrt{l+1})\sqrt{m}\eta)\sqrt{3m}\sigma.$$

Note the extra '+1' in the innermost brackets and the factor of 3 as opposed to 2 in Boyen's original scheme. These changes have no overall impact on the size of the (I)SIS parameter which is still $\beta = O(\lambda^{3.5})$.

The completion probability result can be exactly lifted from Lemma 27 of [11].

Appendix B. Security Proof of the Modified Baum Commitment Scheme

We will now prove the security requirements of our modified commitement scheme based on the hardness of the Ring SIS problem. First we prove that breaking the binding property implies solving a Ring SIS problem over $\mathcal{R}_q$.

Lemma 1. *(Binding Property): Starting from two correct distinct openings* $(\hat{S}, \mathbf{p}, \hat{R})$ *and* $(\hat{S}', \mathbf{p}', \hat{R}')$ *for the same commitement* $\mathbf{C}$*, one can efficiently compute a small solution, with norm bounded by some real number* $h = f(\alpha, \gamma)$*, to the Ring SIS instance defined by the top row of* $\hat{B}$*.*

Proof. : Let $(\hat{S}, \mathbf{p}, \hat{R})$ and $(\hat{S}', \mathbf{p}', \hat{R}')$ be two different openings for the same commitement $\mathbf{C}$, then

$$\mathbf{p}\mathbf{C} = \hat{B}\hat{R} + (\mathbf{0}, \mathbf{p}\hat{S}) \tag{B.1}$$

and

$$\mathbf{p}'\mathbf{C} = \hat{B}\hat{R}' + (\mathbf{0}, \mathbf{p}'\hat{S}') \tag{B.2}$$

Multiply equation B.1 by $\mathbf{p}'$, and equation B.2 by $\mathbf{p}$, then subtract we get:

$$\hat{B}(\mathbf{p}'\hat{R} - \mathbf{p}\hat{R}') = (\mathbf{0}, \mathbf{p}'\mathbf{p}(\hat{S} - \hat{S}'))$$

Since $\hat{S} - \hat{S}' \neq 0$ and both $\mathbf{p}$ and $\mathbf{p}'$ are invertible, then we have $\mathbf{p}'\mathbf{p}(\hat{S} - \hat{S}') \neq 0$, therefore $\mathbf{p}'\hat{R} - \mathbf{p}\hat{R}' \neq 0$. Hence a solution $\mathbf{p}'\hat{R} - \mathbf{p}\hat{R}'$ such that $\|\mathbf{p}'\hat{R} - \mathbf{p}\hat{R}'\|_\infty < h$, to the Ring SIS instance defined by the first row of $\hat{B}$. □ □

Lemma 2. *(Hiding Property): Assume that the mini-entropy of the vectors* $\hat{R}_t$ *and* $\hat{R}_h$ *sampled from* $\mathcal{D}$ *is at least* $(l_t + l_h + 2)\log(|\mathcal{R}_q|) + \lambda$*, where* λ *is a security parameter, and the function* $f_{\hat{B}}(\hat{R}) = \hat{A}\hat{R}$ *for some* $\hat{A} \in \mathcal{R}_q^k$*, is universal (as defined in [14]). Then the scheme is statistically hiding.*

Proof. : Although the commitment gives the adversary $\log(|\mathcal{R}_q|)$ bits of information on $\hat{R}$, precisely the dot product of $\hat{R}$ with the first row $\hat{B}_1$ in $\hat{B}$, we still have $(l_t + l_h + 1)\log(|\mathcal{R}_q|) + \lambda$ bits of randomness left in $\hat{R}$. Let $\hat{B} = [\hat{B}_1 \in \mathcal{R}_q^{1\times k} | \hat{B}_r \in \mathcal{R}_q^{(l_t + l_h + 1)\times k}]^T$, then by the left over hash lemma, it follows that $h_{\hat{B}_r}(\hat{R})$ is statistically close to random, even given $h_{\hat{B}_1}(\hat{R})$. Thus, the scheme is statistically hiding. □ □

Appendix C. Ideal Functionalities From [16]

Appendix C.1. Semi-Authenticated Channels via F_{auth*}

This functionality must captures the fact that a sender S sends a message containing both authenticated and unauthenticated parts to a receiver R, while giving the host F the power to block the message, replace it and block the communication. F_{auth^*} capture these requirements.

1. On input (SEND, sid, ssid, μ_1, μ_2, F) from S, check that $\mathsf{sid} = (S, R, \mathsf{sid}')$ for some R and output (REPLACE1, sid, ssid, μ_1, μ_2, F) to $\mathcal{S}$;
2. On input (REPLACE1,sid, ssid, μ_2', F) from $\mathcal{S}$, output (APPEND, sid, ssid, μ_1, μ_2') to F.
3. On input (APPEND, sid, ssid, μ_2'') from F, output (REPLACE2, sid, ssid, μ_1, μ_2'') to $\mathcal{S}$.
4. On input (REPLACE2, sid, ssid, μ_2''') from $\mathcal{S}$, output (SENT, sid, ssid, μ_1, μ_2''') to R

Figure C.7: The special authenticated communicatioin functionality F_{auth^*}

Appendix C.2. Certification Authority

1. Upon receiving the first message (Register, sid, v) from a party P, send (Rigester, sid, v) to the adversary;
2. Upon receiving ok from the adversary, if $\mathsf{sid} = P$ and this is the first request from P, then record the pair (P, v).
3. Upon receiving a message (Retrieve, sid) from a party P', send (Retrieve, sid, P') to the adversary, and wait for an ok response from the adversary.
4. If there is a recorded pair (sid, v), output (Retrieve, sid, v) to P'.
5. Else, output (Retrieve, sid, $\perp$) to P'.

Figure C.8: Ideal certification authority functionality F_{ca}

Appendix C.3. Secure Message Transmission

This functionality is parametrized by a leakage function $l : \{0,1\}^* \to \{0,1\}^*$. For the security proof, it is required that the leakage function l satisfies the following property:

$$l(b) = l(b') \implies l(a, b) = l(a, b')$$

This is a natural requirement, as most secure channels will at most leak the lenght of the plaintext, for which this property holds.

1. Upon receiving input (Send, S, R, sid, μ) from S, send (Sent, S, R, sid, $l(\mu)$) to the adversary;
2. Generate a private delayed output (Sent, S, sid, μ) to R and halt.
3. Upon receiving (Corrupt, sid, P) from the adversay, where $P \in \{S,\ R\}$, disclose μ to the adversary.
4. If the adversary provides a value μ', and $P = S$, and no output has been given to R, then output (Sent, S, sid, μ') to R and halt.

Figure C.9: Ideal secure message transmission functionality F^{l}_{smt}

Appendix C.4. Common Reference String

This functionality is parametrized by a distribution $\mathcal{D}$, from which crs is sampled.

1. Upon receiving input (CRS, sid) from a party P,verify that $\mathsf{sid} = (\mathcal{P},\ \mathsf{sid}')$ where $\mathcal{P}$ is the set of identities, and $P \in \mathcal{P}$, else ignore the input.
2. If there is no r recorded, then choose and record $r \leftarrow \mathcal{D}$.
3. Finally, send a public delayed output (CRS, sid, r) to P.

Figure C.10: Ideal crs functionality $F^{\mathcal{D}}_{crs}$

- **SETUP**

  On input (SETUP, sid) from I, output (FORWARD, (SETUP, sid, I) to $\mathcal{S}$.

- **JOIN**

  1. On input (JOIN, sid, jsid, $\mathsf{tpm_i}$) from the host $\mathsf{host_j}$, output (FORWARD, (JOIN, sid, jsid, $\mathsf{tpm_i}$), $\mathsf{host_j}$) to $\mathcal{S}$.
  2. On input (JOINPROCEED, sid, jsid) from I, output (FORWARD, (JOINPROCEED, sid, jsid), I) to $\mathcal{S}$.

- **SIGN**

  1. On input (SIGN, sid, ssid, $\mathsf{tpm_i}$, bsn) from the host $\mathsf{host_j}$, output (FORWARD, (SIGN, sid, ssid, $\mathsf{tpm_i}$, bsn), $\mathsf{host_j}$) to $\mathcal{S}$.
  2. On input (SIGNPROCEED, sid, ssid) from $\mathsf{tpm_i}$, output (FORWARD, (SIGNPROCEED, sid, ssid), $\mathsf{tpm_i}$) to $\mathcal{S}$.

- **VERIFY**

  On input (VERIFY, sid, μ, bsn, σ, RL) from V, output (FORWARD, (VERIFY, sid, μ, bsn, σ, RL), V) to $\mathcal{S}$.

- **LINK**

  On the input (LINK, sid, σ_1, μ_1, σ_2, μ_2, bsn) from V, output (FORWARD, (LINK, sid, σ_1, μ_1, σ_2, μ_2, bsn), V) to $\mathcal{S}$.

- **OUTPUT**

  On input (OUTPUT, P, μ) from $\mathcal{S}$, output μ to P.

Figure D.11: Game 3 for F

- **KeyGen**

  Upon receiving input (FORWARD, (SETUP, sid, I)from F, give "I" (SETUP, sid) .

- **JOIN**

1. Upon receiving (FORWARD, (JOIN, sid, jsid, $\mathsf{tpm_i}$), $\mathsf{host_j}$) from F, give input (JOIN, sid, jsid, $\mathsf{tpm_i}$) to the host "$\mathsf{host_j}$"
2. Upon receiving intput (FORWARD, (JOINPROCEED, sid, jsid), I) from F, give "I" input (JOINPROCEED, sid, jsid).

- **SIGN**

1. Upon receiving input (FORWARD, (SIGN, sid, ssid, $\mathsf{tpm_i}$, bsn), $\mathsf{host_j}$) from F, give "$\mathsf{host_j}$" input (SIGN, sid, ssid, $\mathsf{tpm_i}$, bsn).
2. Upon receiving input (FORWARD, (SIGNPROCEED, sid, ssid), $\mathsf{tpm_i}$) from F, give "$\mathsf{tpm_i}$" input (SIGNPROCEED, sid, ssid).

- **VERIFY**

  Upon receiving input (FORWARD, (VERIFY, sid, μ, bsn, σ, RL), V) from F, give "V" input (VERIFY, sid, μ, bsn, σ, RL).

- **LINK**

  Upon receiving input (FORWARD, (LINK, sid, σ_1, μ_1, σ_2, μ_2, bsn), V) from F, give "V" input (LINK, sid, σ_1, μ_1, σ_2, μ_2, bsn).

- **OUTPUT**

  When any simulated party "P" outputs a message μ, $\mathcal{S}$ sends (OUTPUT, P, μ)to F.

Figure D.12: Game 3 for $\mathcal{S}$

- **SETUP**

1. On input (SETUP, sid) from I, verify that $\mathsf{sid} = (I, \mathsf{sid}')$ and output (SETUP, sid) to $\mathcal{S}$.
2. On input (ALGORITHMS, sid, sign, ver, link, identify, Kgen) from $\mathcal{S}$, check that ver, link, and identify are deterministic. Store (sid, sign, ver, link, identify, Kgen) and output (SETUPDOE, sid) to I.

- **JOIN**

1. On input (JOIN, sid, jsid, $\mathsf{tpm_i}$) from the host $\mathsf{host_j}$, output (FORWARD, (JOIN, sid, jsid, $\mathsf{tpm_i}$), $\mathsf{host_j}$) to $\mathcal{S}$.
2. On input (JOINPROCEED, sid, jsid) from I, output (FORWARD, (JOINPROCEED, sid, jsid), I) to $\mathcal{S}$.

- **SIGN**

  1. On input (SIGN, sid, ssid, $\mathsf{tpm_i}$, bsn) from the host $\mathsf{host_j}$, output (FORWARD, (SIGN, sid, ssid, $\mathsf{tpm_i}$, bsn), $\mathsf{host_j}$) to $\mathcal{S}$.
  2. On input (SIGNPROCEED, sid, ssid) from $\mathsf{tpm_i}$, output (FORWARD, (SIGNPROCEED, sid, ssid), $\mathsf{tpm_i}$) to $\mathcal{S}$.

- **VERIFY**

  On input (VERIFY, sid, μ, bsn, σ, RL) from V, output (FORWARD, (VERIFY, sid, μ, bsn, σ, RL), V) to $\mathcal{S}$.

- **LINK**

  On the input (LINK, sid, σ_1, μ_1, σ_2, μ_2, bsn) from V, output (FORWARD, (LINK, sid, σ_1, μ_1, σ_2, μ_2, bsn), V) to $\mathcal{S}$.

- **OUTPUT**

  On input (OUTPUT, P, μ) from $\mathcal{S}$, output μ to P.

Figure D.13: Game 4 for F

- **KeyGen**: Honest I: On input (SETUP, sid) from F
  - Check $\mathsf{sid} = (I,\ \mathsf{sid}')$, output $\perp$ to I if the check fails.
  - Give "I" input (SETUP, sid).
  - Upon receiving output (SETUPDONE, sid) from "I", $\mathcal{S}$ takes its private key $\hat{T}_I$.
  - Define sig(gsk, μ, bsn) as follows:
    - Define SamplePre($\hat{A}_{id}$, $\hat{T}_I$, q, $\mathbf{u}_h$, s) that outputs a Boyen signature $\hat{X}_h$ [11], where $\mathbf{u}_h = \mathbf{u} - \mathbf{u}_t$ with $\mathbf{u}_t = \hat{A}_t \cdot gsk$, $\hat{X}_h$ will be our L-DAA credential.
    - $\mathsf{nym} = \mathcal{H}(\mathsf{bsn}) \cdot \mathbf{x}_1 + \mathbf{e} \mod q$ with $\|\mathbf{e}\|_\infty < \beta'$.
    - $\pi = \mathsf{SPK}\Big\{\mathsf{public} := \{\mathsf{pp},\ \mathsf{nym},\ \mathsf{bsn}\},$ $\mathsf{witness} := \{\hat{X} = (\mathbf{x}_1, \cdots, \mathbf{x}_{3m}),\ \mathsf{id},\ \mathbf{e}\}$: $[\hat{A}_t|\hat{A}_h]\cdot\hat{X} = \mathbf{u} \wedge \|\hat{X}\|_\infty \le \beta \wedge \mathsf{nym} = \mathcal{H}(\mathsf{bsn})\cdot\mathbf{x}_1+\mathbf{e} \mod q \wedge \|\mathbf{e}\|_\infty \le \beta'\Big\}(\mu)$. Output the L- DAA signature $\sigma = (\mathsf{nym}, \mathsf{bsn}, \pi)$.
  - Define ver($\sigma, \mu, \mathsf{bsn}$) as follows: It parses σ as (nym, bsn, π), and checks SPK on π w.r.t bsn, nym, μ and $\mathbf{u}$. It output 1 if the proof is valid and 0 otherwise.

– Define link(σ, μ, bsn, σ', μ'): Check whether two signatures (σ, μ) and (σ', μ') that were generated for the same basename bsn stems from the same TPM. Upon input (LINK, sid, σ, μ, σ', μ', bsn) the verifier follow the following steps:

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

– Define identify(σ, μ, bsn, gsk) as follows: It parses σ as (nym, bsn, π) and checks that $gsk = (\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_m) \in \mathcal{R}_q^m$ and $\|gsk\|_\infty < \beta$, ver($\sigma, \mu, \mathsf{bsn}$)=1 and

$$\|\mathsf{nym} - \mathbf{x}_1 \cdot \mathsf{bsn}\|_\infty < \beta'$$

If so output 1, otherwise output 0.

– Define Kgen, take $gsk \in \mathcal{R}_q^m$ with $\|gsk\|_\infty < \beta$ and output gsk.

– $\mathcal{S}$ sends (KEYS, sid, sig, ver, link, identify, Kgen) to F.

Corrupt I: $\mathcal{S}$ notices this setup as it notices I registering a public key with F_{ca} with $\mathsf{sid} = (I, \mathsf{sid}')$.

– If the registered key is in the form ($\hat{A}_I$, π_I) and π_I is valid, then $\mathcal{S}$ extracts $\hat{T}_I$ from π_I.

– $\mathcal{S}$ defines the algorithms sig, ver, link, and identify as before, but now depending on the extracted key. $\mathcal{S}$ sends (SETUP, sid) to F on behalf of I. On input (KEYGEN, sid) from F, $\mathcal{S}$ sends (KEYS, sid, sig, ver, link, identify, Kgen) to F.

– On input (SETUPDONE, sid) from F. $\mathcal{S}$ continues simulating "I".

- **JOIN**, **SIGN**, **VERIFY**, **LINK**: Unchanged.

Figure D.14: Game 4 for S

- **SETUP**

1. On input (SETUP, sid) from I, verify that $\mathsf{sid} = (I, \mathsf{sid}')$ and output (SETUP, sid) to $\mathcal{S}$.
2. On input (ALGORITHMS, sid, sign, ver, link, identify, Kgen) from $\mathcal{S}$, check that ver, link, and identify are deterministic. Store (sid, sign, ver, link, identify, Kgen) and output (SETUPDOE, sid) to I.

- **JOIN**

1. On input (JOIN, sid, jsid, $\mathsf{tpm_i}$) from the host $\mathsf{host_j}$, output (FORWARD, (JOIN, sid, jsid, $\mathsf{tpm_i}$), $\mathsf{host_j}$) to $\mathcal{S}$.
2. On input (JOINPROCEED, sid, jsid) from I, output (FORWARD, (JOINPROCEED, sid, jsid), I) to $\mathcal{S}$.

- **SIGN**

1. On input (SIGN, sid, ssid, $\mathsf{tpm_i}$, bsn) from the host $\mathsf{host_j}$, output (FORWARD, (SIGN, sid, ssid, $\mathsf{tpm_i}$, bsn), $\mathsf{host_j}$) to $\mathcal{S}$.
2. On input (SIGNPROCEED, sid, ssid) from $\mathsf{tpm_i}$, output (FORWARD, (SIGNPROCEED, sid, ssid), $\mathsf{tpm_i}$) to $\mathcal{S}$.

- **VERIFY**

  On input (VERIFY, sid, μ, bsn, σ, RL) from V

  - Set $f = 0$ if there is a $gsk' \in RL$ such that identify(σ, μ, bsn, gsk') = 1.
  - If $f \neq 0$, set f=ver(σ, μ, bsn).
  - Add (σ, μ, bsn, RL, f) to VerResults, output (VERIFIED, sid, f) to V.

- **LINK**

  On the input (LINK, sid, σ_1, μ_1, σ_2, μ_2, bsn) from V

  - Output $\perp$ if at least one of the signatures (σ_1, μ_1, bsn) or (σ_2, μ_2, bsn) is not valid.
  - Set f=link(σ_1, μ_1, σ_2, μ_2, bsn), and output (LINK, sid, f) to V.

- **OUTPUT**

  On input (OUTPUT, P, μ) from $\mathcal{S}$, output μ to P.

Figure D.15: Game 5 for F

- **KeyGen**, **JOIN**, **SIGN** : Unchanged.
- **VERIFY**, **LINK**: Nothing to simulate.

Figure D.16: Game 5 for $\mathcal{S}$

- **SETUP**

  1. On input (SETUP, sid) from I, verify that $\mathsf{sid} = (I, \mathsf{sid}')$ and output (SETUP, sid) to $\mathcal{S}$.
  2. On input (ALGORITHMS, sid, sign, ver, link, identify, Kgen) from $\mathcal{S}$, check that ver, link, and identify are deterministic. Store (sid, sign, ver, link, identify, Kgen) and output (SETUPDOE, sid) to I.

- **JOIN**

  1. JOINREQUEST: On input (JOIN, sid, jsid, $\mathsf{tpm_i}$) from the host $\mathsf{host_j}$ to join the TPM $\mathsf{tpm_i}$
     – Create a join session $\langle \mathsf{jsid}, \mathsf{tpm_i}, \mathsf{host_j}, \text{request} \rangle$.
     – 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 $\langle \mathsf{jsid}, \mathsf{tpm_i}, \mathsf{host_j}, \text{delivered} \rangle$.
     – If I or $\mathsf{tpm_i}$ is honest and $\langle \mathsf{tpm_i}, \star, \star \rangle$ is already in Members, output $\perp$.
     – Output (JOINPROCEED, sid, jsid, $\mathsf{tpm_i}$) to I.
  3. JOIN PROCEED: Upon receiving (JOINPROCEED, sid, jsid, $\mathsf{tpm_i}$) from I
     – Update the session record to $\langle \mathsf{jsid}, \mathsf{sid}, \mathsf{tpm_i}, \mathsf{host_j}, \text{complete} \rangle$.
     – Output (JOINCOMPLETE, sid, jsid) to $\mathcal{S}$.
  4. KEY GENERATION: On input (JOINCOMPLETE, sid, jsid, gsk) from $\mathcal{S}$.
     – Update the session record to $\langle \mathsf{jsid}, \mathsf{tpm_i}, \mathsf{host_j}, \text{complete} \rangle$
     – If both $\mathsf{tpm_i}$ and $\mathsf{host_j}$ are honest, set $gsk = \perp$.
     – Insert $\langle \mathsf{tpm_i}, \mathsf{host_j}, gsk \rangle$ into Members, and output (JOINED, sid, jsid) to $\mathsf{host_j}$.

- **SIGN**

  1. On input (SIGN, sid, ssid, $\mathsf{tpm_i}$, bsn) from the host $\mathsf{host_j}$, output (FORWARD, (SIGN, sid, ssid, $\mathsf{tpm_i}$, bsn), $\mathsf{host_j}$) to $\mathcal{S}$.
  2. On input (SIGNPROCEED, sid, ssid) from $\mathsf{tpm_i}$, output (FORWARD, (SIGNPROCEED, sid, ssid), $\mathsf{tpm_i}$) to $\mathcal{S}$.

- **VERIFY**

  On input (VERIFY, sid, μ, bsn, σ, RL) from V

  – Set $f = 0$ if there is a $gsk' \in RL$ such that identify(σ, μ, bsn, gsk') $= 1$.

– If $f \neq 0$, set f=ver(σ, μ, bsn).
– Add (σ, μ, bsn, RL, f) to VerResults, output (VERIFIED, sid, f) to V.

- **LINK**

  On the input (LINK, sid, σ_1, μ_1, σ_2, μ_2, bsn) from V

  – Output $\perp$ if at least one of the signatures (σ_1, μ_1, bsn) or (σ_2, μ_2, bsn) is not valid.
  – Set f=link(σ_1, μ_1, σ_2, μ_2, bsn), and output (LINK, sid, f) to V.

- **OUTPUT**

  On input (OUTPUT, P, μ) from $\mathcal{S}$, output μ to P.

Figure D.17: Game 6 for F

- **KeyGen**: Unchanged.

  JOIN: **Honest host,** I

  – When $\mathcal{S}$ receives (JOINSTART, sid, jsid, $\mathsf{tpm_i}$, $\mathsf{host_j}$) from F
  – It simulates the real world protocol by giving "$\mathsf{host_j}$" input (JOIN, sid, jsid, $\mathsf{tpm_i}$) and waits for output (JOINPROCEED, sid, jsid, $\mathsf{tpm_i}$) from "I".
  – If $\mathsf{tpm_i}$ is corrupt, $\mathcal{S}$ extracts gsk from the proof $\pi_{\mathbf{u_t}}$ and stores it. If $\mathsf{tpm_i}$ is honest, $\mathcal{S}$ already knows gsk as it is simulating $\mathsf{tpm_i}$.
  – $\mathcal{S}$ sends (JOINSTART, sid, jsid) to F.
  – Upon receiving input (JOINCOMPLETE, sid, jsid) from F, $\mathcal{S}$ gives "I" input (JOINPROCEED, sid, jsid) and waits for output (JOINED, sid, jsid) from "$\mathsf{host_j}$".
  – Output (JOINCOMPLETE, sid, jsid gsk) to F.

  Honest host, Corrupt I:

  – On input (JOINSTART, sid, jsid, $\mathsf{tpm_i}$, $\mathsf{host_j}$) from F, $\mathcal{S}$ gives "$\mathsf{host_j}$" input (JOIN, sid, jsid, $\mathsf{tpm_i}$) and waits for output (JOINED, sid, jsid, $\mathsf{tpm_i}$) from "$\mathsf{host_j}$".
  – $\mathcal{S}$ sends (JOINSTART, sid, jsid) to F.
  – Upon receiving input (JOINPROCEED, sid, jsid) from F, $\mathcal{S}$ sends (JOINPROCEED, sid, jsid) to F on behalf of I.
  – Upon receiving input (JOINCOMPLETE, sid, jsid) from F, $\mathcal{S}$ sends (JOINCOMPLETE, sid, jsid, $\perp$) to F.

Honest TPM , I, Corrupt host:

- $\mathcal{S}$ notices this join as "$\mathsf{tpm_i}$" receives a nonce ρ from $\mathsf{host_j}$.
- $\mathcal{S}$ makes a join query on behalf of $\mathsf{host_j}$ by sending (JOIN, sid, jsid, $\mathsf{tpm_i}$) to F.
- Upon input (JOINSTART, sid, jsid, $\mathsf{tpm_i}$, $\mathsf{host_j}$) from F, $\mathcal{S}$ continues the simulation of "$\mathsf{tpm_i}$" until "I" outputs (JOINPROCEED, sid, jsid, $\mathsf{tpm_i}$).
- $\mathcal{S}$ sends (JOINSTART, sid, jsid) to F.
- Upon input (JOINCOMPLETE, sid, jsid) from F, $\mathcal{S}$ sends (JOINCOMPLETE, sid, jsid, gsk) to F, where gsk is taken from simulating "$\mathsf{tpm_i}$".
- Upon receiving (JOINED, sid, jsid) from F as $\mathsf{host_j}$ is corrupt, $\mathcal{S}$ gives "I" input (JOINPROCEED, sid, jsid).

Honest I, Corrupt TPM , host:

- $\mathcal{S}$ notices this join as "I" receives (SENT, sid', $(\mathbf{u}_t, \pi_t)$, $\mathsf{host_j}$) from F_{auth^*}.
- Parse sid' as ($\mathsf{tpm_i}$, sid, I), $\mathcal{S}$ then extracts gsk from the proof $\pi_{\mathbf{u_t}}$.
- $\mathcal{S}$ doesn't know the identity of the host that started this join, so $\mathcal{S}$ chooses some corrupt $\mathsf{host_j}$ and proceeds as if this host initiated this join, although this may not be the correct host. This makes no difference as when creating signatures we only look for corrupt host or TPM, so fully corrupted platform are not considered in generating signatures.
- $\mathcal{S}$ makes a join query with $\mathsf{tpm_i}$ on behalf of $\mathsf{host_j}$ by sending (JOIN, sid, jsid, $\mathsf{tpm_i}$) to F.
- Upon receiving input (JOINSTART, sid, jsid, $\mathsf{tpm_i}$, $\mathsf{host_j}$) from F, $\mathcal{S}$ continues simulating "I" until it outputs (JOINPROCEED, sid, jsid, $\mathsf{tpm_i}$).
- $\mathcal{S}$ sends (JOINSTART, sid, jsid) to F.
- Upon receiving (JOINCOMPLETE, sid, jsid) from F, $\mathcal{S}$ sends (JOINCOMPLETE, sid, jsid, gsk) to F.
- Upon receiving (JOINED, sid, jsid) from F as $\mathsf{host_j}$ is corrupt, $\mathcal{S}$ gives "I" input (JOINPROCEED, sid, jsid).

Honest TPM, Corrupt host, I:

- $\mathcal{S}$ notices this join as $\mathsf{tpm_i}$ receives a nonce ρ from $\mathsf{host_j}$.
- $\mathcal{S}$ simply simulates $\mathsf{tpm_i}$ honestly, no need to include F as $\mathsf{tpm_i}$ doesn't receive inputs or send outputs in the join interface.

• **SIGN**, **VERIFY**, **LINK**: Unchanged.

Figure D.18: Game 6 for $\mathcal{S}$

- **SETUP**, **JOIN**: Unchanged.
- **SIGN**
  - SIGN REQUEST: On input (SIGN, sid, ssid, $\mathsf{tpm_i}$, μ, bsn) from the host $\mathsf{host_j}$,
    - Create a sign session $\langle \mathsf{ssid}, \mathsf{tpm_i}, \mathsf{host_j}, \mu, \mathsf{bsn}, \mathsf{request}\rangle$.
    - Output (SIGNSTART, sid, ssid, $\mathsf{tpm_i}$, $\mathsf{host_j}$) to $\mathcal{S}$.
  - SIGN REUEST DELIVERY: On input (SIGNSTART, sid, ssid) from $\mathcal{S}$, update the session to $\langle \mathsf{ssid}, \mathsf{tpm_i}, \mathsf{host_j}, \mu, \mathsf{bsn}, \mathsf{delivered}\rangle$.
  - Output (SIGNPROCEED, sid, ssid, μ, bsn) to $\mathsf{tpm_i}$.
  - SIGN PROCEED: On input (SIGNPROCEED, sid, ssid) from $\mathsf{tpm_i}$
    - Update the records $\langle \mathsf{ssid}, \mathsf{tpm_i}, \mathsf{host_j}, \mu, \mathsf{bsn}, \text{delivered}\rangle$.
    - Output (SIGNCOMPETE, sid, ssid) to $\mathcal{S}$.
  - SIGNATURE GENERATION: On the input (SIGNCOMPETE, 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 $\mathsf{bsn} \neq \perp$, then retrieve gsk from the $\langle \mathsf{tpm_i}, \mathsf{bsn}, gsk\rangle \in$ DomainKeys.
    - If $\mathsf{bsn} = \perp$ or no gsk was found, generate a fresh key $gsk \leftarrow \mathrm{Kgen}(1^\lambda)$.
    - Store $\langle \mathsf{tpm_i}, \mathsf{bsn}, gsk\rangle$ in DomainKeys.
    - Generate the signature $\sigma \leftarrow sig(gsk, \mu, \mathsf{bsn})$.
    - 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 $\mathsf{host_j}$.
- **VERIFY**

  On input (VERIFY, sid, μ, bsn, σ, RL) from V
  - Set $f = 0$ if there is a $gsk' \in RL$ such that identify$(\sigma, \mu, \mathsf{bsn}, gsk') = 1$.
  - If $f \neq 0$, set f=ver$(\sigma, \mu, \mathsf{bsn})$.
  - Add $(\sigma, \mu, \mathsf{bsn}, RL, f)$ to VerResults, output (VERIFIED, sid, f) to V.
- **LINK**

  On the input (LINK, sid, σ_1, μ_1, σ_2, μ_2, bsn) from V
  - Output $\perp$ if at least one of the signatures $(\sigma_1, \mu_1, \mathsf{bsn})$ or $(\sigma_2, \mu_2, \mathsf{bsn})$ is not valid.
  - Set f=link$(\sigma_1, \mu_1, \sigma_2, \mu_2, \mathsf{bsn})$, and output (LINK, sid, f) to V.
- **OUTPUT**

  On input (OUTPUT, P, μ) from $\mathcal{S}$, output μ to P.

Figure D.19: Game 7 for F

- **KeyGen**, **JOIN**: Unchanged.
- **SIGN** :**Honest TPM, host:**

  Upon receiving (SIGNSTART, sid, ssid, $\mathsf{tpm_i}$, $\mathsf{host_j}$, bsn, μ) from F.

  - $\mathcal{S}$ starts the simulation by giving "$\mathsf{host_j}$" input (SIGN, sid, ssid, $\mathsf{tpm_i}$, μ, bsn).
  - When "$\mathsf{tpm_i}$" outputs (SIGNPROCEED, sid, ssid, μ, bsn), $\mathcal{S}$ sends (SIGNSTART, sid, ssid) to F.
  - Upon receiving (SIGNCOMPLETE, sid, ssid) from F, output (SIGNPROCEED, sid, ssid) to "$\mathsf{tpm_i}$".
  - When "$\mathsf{host_j}$" outputs (SIGNATURE, sid, ssid, σ), send (SIGNCOMPLETE, sid, ssid, $\perp$) to F.

  Honest host, Corrupt TPM: Upon receiving (SIGNSTART, sid, ssid, $\mathsf{tpm_i}$, $\mathsf{host_j}$, bsn, μ) from F.

  - Send (SIGNSTART, sid, ssid) to F.
  - Upon receiving (SIGNPROCEED, sid, ssid, μ, bsn) from F on behalf of $\mathsf{tpm_i}$, as $\mathsf{tpm_i}$ is corrupt, $\mathcal{S}$ gives "$\mathsf{host_j}$" input (SIGN, sid, ssid, $\mathsf{tpm_i}$, μ, bsn).
  - When "$\mathsf{host_j}$" outputs (SIGNATURE, sid, ssid, σ), $\mathcal{S}$ sends (SIGNPROCEED, sid, ssid) to F on behlaf of $\mathsf{tpm_i}$.
  - Upon receiving (SIGNCOMPLETE, sid, ssid) from F, send (SIGNCOMPLETE, sid, ssid, σ) to F.

  Honest TPM, Corrupt host:

  - $\mathcal{S}$ notices this sign as "$\mathsf{tpm_i}$" receives a message μ and bsn from $\mathsf{host_j}$.
  - $\mathcal{S}$ sends (SIGN, sid, ssid, $\mathsf{tpm_i}$, μ, bsn) to F on behalf of $\mathsf{host_j}$.
  - Upon receiving (SIGNSTART, sid, ssid, μ, bsn, $\mathsf{tpm_i}$, $\mathsf{host_j}$) from F, continue simulating "$\mathsf{tpm_i}$", until "$\mathsf{tpm_i}$" outputs (SIGNPROCEED, sid, ssid, μ, bsn).
  - Send (SIGNSTART, sid, ssid) to F.
  - Upon receiving (SIGNCOMPLETE, sid, ssid) from F, send (SIGNCOMPLETE, sid, ssid, $\perp$) to F.
  - When F outputs (SIGNATURE, sid, ssid, σ) on behalf of $\mathsf{host_j}$, $\mathcal{S}$ sends (SIGNPROCEED, sid, ssid) to "$\mathsf{tpm_i}$".
  - send (SIGNCOMPLETE, sid, ssid, σ) to "$\mathsf{tpm_i}$".
- **VERIFY**, **LINK**: Nothing to simulate.

Figure D.20: Game 7 for $\mathcal{S}$

- **SETUP**, **JOIN**: Unchanged.
- **SIGN**
  - SIGN REQUEST: On input (SIGN, sid, ssid, $\mathsf{tpm_i}$, μ, bsn) from the host $\mathsf{host_j}$,
    - 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 $\mathcal{S}$.
  - SIGN REUEST DELIVERY: On input (SIGNSTART, sid, ssid) from $\mathcal{S}$, update the session to $\langle \mathsf{ssid},\ \mathsf{tpm_i},\ \mathsf{host_j},\ \mu,\ \mathsf{bsn},\ \mathsf{delivered}\rangle$.
  - Output (SIGNPROCEED, sid, ssid, μ, bsn) to $\mathsf{tpm_i}$.
  - SIGN PROCEED: On input (SIGN PROCEED, sid, ssid) from $\mathsf{tpm_i}$
    - Update the records $\langle \mathsf{ssid},\ \mathsf{tpm_i},\ \mathsf{host_j},\ \mu,\ \mathsf{bsn}, \text{delivered}\rangle$.
    - Output (SIGNCOMPETE, sid, ssid) to $\mathcal{S}$.
  - SIGNATURE GENERATION: On the input (SIGNCOMPETE, 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 $\mathsf{bsn} \neq \perp$, then retrieve gsk from the $\langle \mathsf{tpm_i},\ \mathsf{bsn},\ gsk\rangle \in$ DomainKeys.
    - If $\mathsf{bsn} = \perp$ or no gsk was found, generate a fresh key $gsk \leftarrow Kgen(1^{\lambda})$.
    - Store $\langle \mathsf{tpm_i},\ \mathsf{bsn},\ gsk\rangle$ in DomainKeys.
    - Generate the signature $\sigma \leftarrow sig(gsk,\ \mu,\ \mathsf{bsn})$.
    - 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 $\mathsf{host_j}$.
- **VERIFY**

  On input (VERIFY, sid, μ, bsn, σ, RL) from V
  - Set $f = 0$ if there is a $gsk' \in RL$ such that identify(σ, μ, bsn, gsk') $= 1$.
  - If $f \neq 0$, set f=ver(σ, μ, bsn).
  - Add (σ, μ, bsn, RL, f) to VerResults, output (VERIFIED, sid, f) to V.
- **LINK** On the input (LINK, sid, σ_1, μ_1, σ_2, μ_2, bsn) from V
  - Output $\perp$ if at least one of the signatures (σ_1, μ_1, bsn) or (σ_2, μ_2, bsn) is not valid.
  - Set f=link(σ_1, μ_1, σ_2, μ_2, bsn), and output (LINK, sid, f) to V.
- **OUTPUT**

  On input (OUTPUT, P, μ) from $\mathcal{S}$, output μ to P.

Figure D.21: Game 8 for F

- **KeyGen**, **JOIN**: Unchanged.
- **SIGN**

Honest TPM, host:

Upon receiving (SIGNSTART, sid, ssid, $\mathsf{tpm_i}$, $\mathsf{host_j}$, l) from F.

- $\mathcal{S}$ takes a dummy pair (μ', bsn') such that $l(\mu', \mathsf{bsn}') = l$.
- $\mathcal{S}$ starts the simulation by giving "$\mathsf{host_j}$" input (SIGN, sid, ssid, $\mathsf{tpm_i}$, μ', bsn').
- When "$\mathsf{tpm_i}$" outputs (SIGNPROCEED, sid, ssid, μ', bsn'), $\mathcal{S}$ sends (SIGNSTART, sid, ssid) to F.
- Upon receiving (SIGNCOMPLETE, sid, ssid) from F, output (SIGNPROCEED, sid, ssid) to "$\mathsf{tpm_i}$".
- When "$\mathsf{host_j}$" outputs (SIGNATURE, sid, ssid, σ), send (SIGNCOMPLETE, sid, ssid, $\perp$) to F.

Honest host, Corrupt TPM:
Upon receiving (SIGNSTART, sid, ssid, $\mathsf{tpm_i}$, $\mathsf{host_j}$, l) from F.

- Send (SIGNSTART, sid, ssid) to F.
- Upon receiving (SIGNPROCEED, sid, ssid, μ, bsn) from F on behalf of $\mathsf{tpm_i}$, as $\mathsf{tpm_i}$ is corrupt, $\mathcal{S}$ gives "$\mathsf{host_j}$" input (SIGN, sid, ssid, $\mathsf{tpm_i}$, μ, bsn).
- When "$\mathsf{host_j}$" outputs (SIGNATURE, sid, ssid, σ), $\mathcal{S}$ sends (SIGNPROCEED, sid, ssid, μ, bsn) to F on behlaf of $\mathsf{tpm_i}$.
- Upon receiving (SIGNCOMPLETE, sid, ssid) from F, send (SIGNCOMPLETE, sid, ssid, σ) to F.

Honest TPM, Corrupt host:

- $\mathcal{S}$ notices this sign as "$\mathsf{tpm_i}$" receives a message μ and bsn from $\mathsf{host_j}$.
- $\mathcal{S}$ sends (SIGN, sid, ssid, $\mathsf{tpm_i}$, μ, bsn) to F on behalf of $\mathsf{host_j}$.
- Upon receiving (SIGNSTART, sid, ssid, $\mathsf{tpm_i}$, $\mathsf{host_j}, l$) from F, continue simulating "$\mathsf{tpm_i}$", until "$\mathsf{tpm_i}$" outputs (SIGNPROCEED, sid, ssid, μ, bsn).
- Send (SIGNSTART, sid, ssid) to F.
- Upon receiving (SIGNCOMPLETE, sid, ssid) from F, send (SIGNCOMPLETE, sid, ssid, $\perp$) to F.

- When F outputs (SIGNATURE, sid, ssid, σ) on behalf of $\mathsf{host_j}$, $\mathcal{S}$ sends (SIGNPROCEED, sid, ssid) to "$\mathsf{tpm_i}$".
- send (SIGNCOMPLETE, sid, ssid, σ) to "$\mathsf{tpm_i}$".

• **VERIFY**, **LINK**: Nothing to simulate.

Figure D.22: Game 8 for $\mathcal{S}$

• **SETUP**, **JOIN**: Unchanged.

• **SIGN**

- SIGN REQUEST: On input (SIGN, sid, ssid, $\mathsf{tpm_i}$, μ, bsn) from the host $\mathsf{host_j}$,
  - 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 $\langle$ssid, $\mathsf{tpm_i}$, $\mathsf{host_j}$, μ, bsn, request$\rangle$.
  - Output (SIGNSTART, sid, ssid, $\mathsf{tpm_i}$, $\mathsf{host_j}$, $l(\mu, \mathsf{bsn})$) to $\mathcal{S}$.
- SIGN REUEST DELIVERY: On input (SIGNSTART, sid, ssid) from $\mathcal{S}$, update the session to $\langle$ssid, $\mathsf{tpm_i}$, $\mathsf{host_j}$, μ, bsn, delivered$\rangle$.
- Output (SIGNPROCEED, sid, ssid, μ, bsn) to $\mathsf{tpm_i}$.
- SIGN PROCEED: On input (SIGN PROCEED, sid, ssid) from $\mathsf{tpm_i}$
  - Update the records $\langle$ssid, $\mathsf{tpm_i}$, $\mathsf{host_j}$, μ, bsn, delivered$\rangle$.
  - Output (SIGNCOMPETE, sid, ssid) to $\mathcal{S}$.
- SIGNATURE GENERATION: On the input (SIGNCOMPETE, 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 $\mathsf{bsn} \neq \perp$, then retrieve gsk from the $\langle \mathsf{tpm_i}, \mathsf{bsn}, gsk\rangle \in$ DomainKeys.
  - If $\mathsf{bsn} = \perp$ or no gsk was found, generate a fresh key $gsk \leftarrow Kgen(1^\lambda)$.
  - Store $\langle \mathsf{tpm_i}, \mathsf{bsn}, gsk\rangle$ in DomainKeys.
  - Generate the signature $\sigma \leftarrow sig(gsk, \mu, \mathsf{bsn})$.
  - 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 $\mathsf{host_j}$.

• **VERIFY**

On input (VERIFY, sid, μ, bsn, σ, RL) from V

- Set $f = 0$ if there is a $gsk' \in RL$ such that identify(σ, μ, bsn, gsk') $= 1$.
- If $f \neq 0$, set f=ver(σ, μ, bsn).

– Add (σ, μ, bsn, RL, f) to VerResults, output (VERIFIED, sid, f) to V.

- **LINK**

  On the input (LINK, sid, σ_1, μ_1, σ_2, μ_2, bsn) from V

  – Output $\perp$ if at least one of the signatures (σ_1, μ_1, bsn) or (σ_2, μ_2, bsn) is not valid.

  – Set f=link(σ_1, μ_1, σ_2, μ_2, bsn), and output (LINK, sid, f) to V.

Figure D.23: Game 9 for F

- **SETUP**: Unchanged.
- **JOIN**: Unchanged.
- **SIGN**: Unchanged.
- **VERIFY**: Unchanged.
- **LINK**: Unchanged.

Figure D.24: Games 9-16 for $\mathcal{S}$

- **SETUP**: Unchanged.

- **JOIN**

  1. JOINREQUEST: On input (JOIN, sid, jsid, $\mathsf{tpm_i}$) from the host $\mathsf{host_j}$ to join the TPM $\mathsf{tpm_i}$
     – Create a join session $\langle \mathsf{jsid}$, $\mathsf{tpm_i}$, $\mathsf{host_j}$, 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 $\langle \mathsf{jsid}$, $\mathsf{tpm_i}$, $\mathsf{host_j}$, delivered$\rangle$.
     – If I or $\mathsf{tpm_i}$ is honest and $\langle \mathsf{tpm_i}, \star, \star \rangle$ is already in Members, output $\perp$.
     – Output (JOINPROCEED, sid, jsid, $\mathsf{tpm_i}$) to I.
  3. JOIN PROCEED: Upon receiving (JOINPROCEED, sid, jsid, $\mathsf{tpm_i}$) from I
     – Update the session record to $\langle \mathsf{jsid}$, 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 $\mathcal{S}$.
   - Update the session record to $\langle \mathsf{jsid}, \mathsf{tpm_i}, \mathsf{host_j}, \text{complete}\rangle$
   - 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.
     * If $\mathsf{tpm_i}$ is corrupt, then CheckGskCorrupt(gsk)=1.
     * Insert $\langle \mathsf{tpm_i}, \mathsf{host_j}, gsk\rangle$ into Members, and output (JOINED, sid, jsid) to $\mathsf{host_j}$.

- **SIGN**
  - SIGN REQUEST: On input (SIGN, sid, ssid, $\mathsf{tpm_i}$, μ, bsn) from the host $\mathsf{host_j}$,
    * 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 $\mathcal{S}$.
  - SIGN REUEST DELIVERY: On input (SIGNSTART, sid, ssid) from $\mathcal{S}$, update the session to $\langle \mathsf{ssid}, \mathsf{tpm_i}, \mathsf{host_j}, \mu, \mathsf{bsn}, \mathsf{delivered}\rangle$.
  - Output (SIGNPROCEED, sid, ssid, μ, bsn) to $\mathsf{tpm_i}$.
  - SIGN PROCEED: On input (SIGN PROCEED, sid, ssid) from $\mathsf{tpm_i}$
    * Update the records $\langle \mathsf{ssid}, \mathsf{tpm_i}, \mathsf{host_j}, \mu, \mathsf{bsn}, \text{delivered}\rangle$.
    * Output (SIGNCOMPETE, sid, ssid) to $\mathcal{S}$.
  - SIGNATURE GENERATION: On the input (SIGNCOMPETE, 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 $\mathsf{bsn} \neq \perp$, then retrieve gsk from the $\langle \mathsf{tpm_i}, \mathsf{bsn}, gsk\rangle \in$ DomainKeys.
    * If $\mathsf{bsn} = \perp$ or no gsk was found, generate a fresh key $gsk \leftarrow Kgen(1^\lambda)$.
    * Check CheckGskHonest(gsk)=1
    * Store $\langle \mathsf{tpm_i}, \mathsf{bsn}, gsk\rangle$ in DomainKeys.
    * Generate the signature $\sigma \leftarrow sig(gsk, \mu, \mathsf{bsn})$.
    * 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 $\mathsf{host_j}$.

- **VERIFY**, **LINK**: Unchanged

Figure D.25: Game 10 for F

- **SETUP**, **JOIN**: Unchanged.
- **SIGN**
  - SIGN REQUEST: On input (SIGN, sid, ssid, $\mathsf{tpm_i}$, μ, bsn) from the host $\mathsf{host_j}$,
    - 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 $\mathcal{S}$.
  - SIGN REUEST DELIVERY: On input (SIGNSTART, sid, ssid) from $\mathcal{S}$, update the session to $\langle \mathsf{ssid},\ \mathsf{tpm_i},\ \mathsf{host_j},\ \mu,\ \mathsf{bsn},\ \mathsf{delivered} \rangle$.
  - Output (SIGNPROCEED, sid, ssid, μ, bsn) to $\mathsf{tpm_i}$.
  - SIGN PROCEED: On input (SIGN PROCEED, sid, ssid) from $\mathsf{tpm_i}$
    - Update the records $\langle \mathsf{ssid},\ \mathsf{tpm_i},\ \mathsf{host_j},\ \mu,\ \mathsf{bsn}, \text{delivered} \rangle$.
    - Output (SIGNCOMPETE, sid, ssid) to $\mathcal{S}$.
  - SIGNATURE GENERATION: On the input (SIGNCOMPETE, 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 $\mathsf{bsn} \neq \perp$, then retrieve gsk from the $\langle \mathsf{tpm_i},\ \mathsf{bsn},\ gsk \rangle \in$ DomainKeys.
    - If $\mathsf{bsn} = \perp$ or no gsk was found, generate a fresh key $gsk \leftarrow Kgen(1^\lambda)$.
    - Check CheckGskHonest(gsk)=1.
    - 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 identify(σ, μ, bsn, gsk)=1.
    - Check the is no TPM other than $\mathsf{tpm_i}$ with key gsk' registered in Members or DomainKeys such that identify(σ, μ, bsn, gsk')=1.
    - 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 $\mathsf{host_j}$.
- **VERIFY**: On input (VERIFY, sid, μ, bsn, σ, RL) from V
  - Set $f = 0$ if there is a $gsk' \in RL$ such that identify(σ, μ, bsn, gsk') $= 1$.
  - If $f \neq 0$, set f=ver(σ, μ, bsn).
  - Add (σ, μ, bsn, RL, f) to VerResults, output (VERIFIED, sid, f) to V.
- **LINK**: On the input (LINK, sid, σ_1, μ_1, σ_2, μ_2, bsn) from V
  - Output $\perp$ if at least one of the signatures (σ_1, μ_1, bsn) or (σ_2, μ_2, bsn) is not valid.
  - Set f=link(σ_1, μ_1, σ_2, μ_2, bsn), and output (LINK, sid, f) to V.

Figure D.26: Game 11 for F

- **SETUP**, **JOIN**: Unchanged.
- **SIGN**
  - SIGN REQUEST: On input (SIGN, sid, ssid, $\mathsf{tpm_i}$, μ, bsn) from the host $\mathsf{host_j}$,
    - 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 $\mathcal{S}$.
  - SIGN REUEST DELIVERY: On input (SIGNSTART, sid, ssid) from $\mathcal{S}$, update the session to $\langle \mathsf{ssid}, \mathsf{tpm_i}, \mathsf{host_j}, \mu, \mathsf{bsn}, \mathsf{delivered} \rangle$.
  - Output (SIGNPROCEED, sid, ssid, μ, bsn) to $\mathsf{tpm_i}$.
  - SIGN PROCEED: On input (SIGN PROCEED, sid, ssid) from $\mathsf{tpm_i}$
    - Update the records $\langle \mathsf{ssid}, \mathsf{tpm_i}, \mathsf{host_j}, \mu, \mathsf{bsn}$, delivered$\rangle$.
    - Output (SIGNCOMPETE, sid, ssid) to $\mathcal{S}$.
  - SIGNATURE GENERATION: On the input (SIGNCOMPETE, 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 $\mathsf{bsn} \neq \perp$, then retrieve gsk from the $\langle \mathsf{tpm_i}, \mathsf{bsn}, gsk \rangle \in$ DomainKeys.
    - If $\mathsf{bsn} = \perp$ or no gsk was found, generate a fresh key $gsk \leftarrow Kgen(1^\lambda)$.
    - Check CheckGskHonest(gsk)=1.
    - 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 identify(σ, μ, bsn, gsk)=1.
    - Check the is no TPM other than $\mathsf{tpm_i}$ with key gsk' registered in Members or DomainKeys such that identify(σ, μ, bsn, gsk')=1.
    - 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 $\mathsf{host_j}$.
- **VERIFY**: On input (VERIFY, sid, μ, bsn, σ, RL) from V
  - Extract all pairs (gsk_i, $\mathsf{tpm_i}$) from the DomainKeys and Members, for which identify(σ, μ, bsn, gsk)=1.
  - Set $f = 0$ if any of the following holds:
    - More than one key gsk_i was found.
    - There is a key $gsk' \in RL$, such that identify(σ, μ, bsn, gsk')=1.
  - If $f \neq 0$, set f=ver(σ, μ, bsn).
  - Add (σ, μ, bsn, RL, f) to VerResults, output (VERIFIED, sid, f) to V.

- **LINK**: On the input (LINK, sid, σ_1, μ_1, σ_2, μ_2, bsn) from V
  - Output $\perp$ if at least one of the signatures (σ_1, μ_1, bsn) or (σ_2, μ_2, bsn) is not valid.
  - Set f=link(σ_1, μ_1, σ_2, μ_2, bsn), and output (LINK, sid, f) to V.

Figure D.27: Game 12 for F

- **SETUP**, **JOIN**: Unchanged.
- **SIGN**
  - SIGN REQUEST: On input (SIGN, sid, ssid, $\mathsf{tpm_i}$, μ, bsn) from the host $\mathsf{host_j}$,
    - 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 $\mathcal{S}$.
  - SIGN REUEST DELIVERY: On input (SIGNSTART, sid, ssid) from $\mathcal{S}$, update the session to $\langle \mathsf{ssid}, \mathsf{tpm_i}, \mathsf{host_j}, \mu, \mathsf{bsn}, \mathsf{delivered} \rangle$.
  - Output (SIGNPROCEED, sid, ssid, μ, bsn) to $\mathsf{tpm_i}$.
  - SIGN PROCEED: On input (SIGN PROCEED, sid, ssid) from $\mathsf{tpm_i}$
    - Update the records $\langle \mathsf{ssid}, \mathsf{tpm_i}, \mathsf{host_j}, \mu, \mathsf{bsn}, \text{delivered} \rangle$.
    - Output (SIGNCOMPETE, sid, ssid) to $\mathcal{S}$.
  - SIGNATURE GENERATION: On the input (SIGNCOMPETE, 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 $\mathsf{bsn} \neq \perp$, then retrieve gsk from the $\langle \mathsf{tpm_i}, \mathsf{bsn}, gsk \rangle \in$ DomainKeys.
    - If $\mathsf{bsn} = \perp$ or no gsk was found, generate a fresh key $gsk \leftarrow Kgen(1^\lambda)$.
    - Check CheckGskHonest(gsk)=1.
    - 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 identify(σ, μ, bsn, gsk)=1.
    - Check the is no TPM other than $\mathsf{tpm_i}$ with key gsk' registered in Members or DomainKeys such that identify(σ, μ, bsn, gsk')=1.
    - 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 $\mathsf{host_j}$.
- **VERIFY**On input (VERIFY, sid, μ, bsn, σ, RL) from V

- Extract all pairs (gsk_i, $\mathsf{tpm_i}$) from the DomainKeys and Members, for which identify(σ, μ, bsn, gsk)=1.
- Set $f = 0$ if any of the following holds:
  * More than one key gsk_i was found.
  * I is honest and no pair ($gsk_i, \mathsf{tpm_i}$) was found.
  * There is a key $gsk' \in RL$, such that identify(σ, μ, bsn, gsk')=1.
- If $f \neq 0$, set f=ver(σ, μ, bsn).
- Add (σ, μ, bsn, RL, f) to VerResults, output (VERIFIED, sid, f) to V.

• **LINK**: On the input (LINK, sid, σ_1, μ_1, σ_2, μ_2, bsn) from V

- Output $\perp$ if at least one of the signatures (σ_1, μ_1, bsn) or (σ_2, μ_2, bsn) is not valid.
- Set f=link(σ_1, μ_1, σ_2, μ_2, bsn), and output (LINK, sid, f) to V.

Figure D.28: Game 13 for F

• **SETUP**, **JOIN**: Unchanged.

• **SIGN**

- SIGN REQUEST: On input (SIGN, sid, ssid, $\mathsf{tpm_i}$, μ, bsn) from the host $\mathsf{host_j}$,
  * 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 $\mathcal{S}$.
- SIGN REUEST DELIVERY: On input (SIGNSTART, sid, ssid) from $\mathcal{S}$, update the session to $\langle \mathsf{ssid}, \mathsf{tpm_i}, \mathsf{host_j}, \mu, \mathsf{bsn}, \mathsf{delivered} \rangle$.
- Output (SIGNPROCEED, sid, ssid, μ, bsn) to $\mathsf{tpm_i}$.
- SIGN PROCEED: On input (SIGN PROCEED, sid, ssid) from $\mathsf{tpm_i}$
  * Update the records $\langle \mathsf{ssid}, \mathsf{tpm_i}, \mathsf{host_j}, \mu, \mathsf{bsn}, \text{delivered} \rangle$.
  * Output (SIGNCOMPETE, sid, ssid) to $\mathcal{S}$.
- SIGNATURE GENERATION: On the input (SIGNCOMPETE, 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 $\mathsf{bsn} \neq \perp$, then retrieve gsk from the $\langle \mathsf{tpm_i}, \mathsf{bsn}, gsk \rangle \in$ DomainKeys.
  * If $\mathsf{bsn} = \perp$ or no gsk was found, generate a fresh key $gsk \leftarrow Kgen(1^\lambda)$.
  * Check CheckGskHonest(gsk)=1.

- 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 identify(σ, μ, bsn, gsk)=1.
- Check the is no TPM other than $\mathsf{tpm_i}$ with key gsk' registered in Members or DomainKeys such that identify(σ, μ, bsn, gsk')=1.
- 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 $\mathsf{host_j}$.

- **VERIFY**: On input (VERIFY, sid, μ, bsn, σ, RL) from V
  - Extract all pairs (gsk_i, $\mathsf{tpm_i}$) from the DomainKeys and Members, for which identify(σ, μ, bsn, gsk)=1.
  - Set $f = 0$ if any of the following holds:
    - More than one key gsk_i was found.
    - I is honest and no pair ($gsk_i, \mathsf{tpm_i}$) was found.
    - An honest $\mathsf{tpm_i}$ was found, but no entry $\langle \star,\ \mu,\ \mathsf{tpm_i},\ \mathsf{bsn} \rangle$ was found in Signed.
    - There is a key $gsk' \in RL$, such that identify(σ, μ, bsn, gsk')=1.
  - If $f \neq 0$, set f=ver(σ, μ, bsn).
  - Add (σ, μ, bsn, RL, f) to VerResults, output (VERIFIED, sid, f) to V.

- **LINK**: On the input (LINK, sid, σ_1, μ_1, σ_2, μ_2, bsn) from V
  - Output $\perp$ if at least one of the signatures (σ_1, μ_1, bsn) or (σ_2, μ_2, bsn) is not valid.
  - Set f=link(σ_1, μ_1, σ_2, μ_2, bsn), and output (LINK, sid, f) to V.

Figure D.29: Game 14 for F

- **SETUP**, **JOIN**: Unchanged.
- **SIGN**
  - SIGN REQUEST: On input (SIGN, sid, ssid, $\mathsf{tpm_i}$, μ, bsn) from the host $\mathsf{host_j}$,
    - 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 $\mathcal{S}$.
  - SIGN REUEST DELIVERY: On input (SIGNSTART, sid, ssid) from $\mathcal{S}$, update the session to $\langle \mathsf{ssid},\ \mathsf{tpm_i},\ \mathsf{host_j},\ \mu,\ \mathsf{bsn},\ \mathsf{delivered} \rangle$.

- Output (SIGNPROCEED, sid, ssid, μ, bsn) to $\mathsf{tpm_i}$.
- SIGN PROCEED: On input (SIGN PROCEED, sid, ssid) from $\mathsf{tpm_i}$
  * Update the records $\langle \mathsf{ssid}, \ \mathsf{tpm_i}, \ \mathsf{host_j}, \ \mu, \ \mathsf{bsn}, \text{delivered}\rangle$.
  * Output (SIGNCOMPETE, sid, ssid) to $\mathcal{S}$.
- SIGNATURE GENERATION: On the input (SIGNCOMPETE, 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 $\mathsf{bsn} \neq \perp$, then retrieve gsk from the $\langle \mathsf{tpm_i}, \ \mathsf{bsn}, \ gsk\rangle \in$ DomainKeys.
  * If $\mathsf{bsn} = \perp$ or no gsk was found, generate a fresh key $gsk \leftarrow Kgen(1^\lambda)$.
  * Check CheckGskHonest(gsk)=1.
  * 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 identify(σ, μ, bsn, gsk)=1.
  * Check the is no TPM other than $\mathsf{tpm_i}$ with key gsk' registered in Members or DomainKeys such that identify(σ, μ, bsn, gsk')=1.
  * 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 $\mathsf{host_j}$.

- **VERIFY**: On input (VERIFY, sid, μ, bsn, σ, RL) from V
  - Extract all pairs (gsk_i, $\mathsf{tpm_i}$) from the DomainKeys and Members, for which identify(σ, μ, bsn, gsk)=1.
  - Set $f = 0$ if any of the following holds:
    * More than one key gsk_i was found.
    * I is honest and no pair $(gsk_i, \mathsf{tpm_i})$ was found.
    * An honest $\mathsf{tpm_i}$ was found, but no entry $\langle \star, \ \mu, \ \mathsf{tpm_i}, \ \mathsf{bsn}\rangle$ was found in Signed.
    * There is a key $gsk' \in RL$, such that identify(σ, μ, bsn, gsk')=1, and no pair $(\mathsf{tpm_i}, \mathsf{gsk_i})$ for honest $\mathsf{tpm_i}$ was found.
  - If $f \neq 0$, set f=ver(σ, μ, bsn).
  - Add (σ, μ, bsn, RL, f) to VerResults, output (VERIFIED, sid, f) to V.

- **LINK**: On the input (LINK, sid, σ_1, μ_1, σ_2, μ_2, bsn) from V
  - Output $\perp$ if at least one of the signatures (σ_1, μ_1, bsn) or (σ_2, μ_2, bsn) is not valid.
  - Set f=link(σ_1, μ_1, σ_2, μ_2, bsn), and output (LINK, sid, f) to V.

Figure D.30: Game 15 for F

- **SETUP**, **JOIN**: Unchanged.
- **SIGN**
  - SIGN REQUEST: On input (SIGN, sid, ssid, $\mathsf{tpm_i}$, μ, bsn) from the host $\mathsf{host_j}$,
    - 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 $\mathcal{S}$.
  - SIGN REUEST DELIVERY: On input (SIGNSTART, sid, ssid) from $\mathcal{S}$, update the session to $\langle \mathsf{ssid}, \mathsf{tpm_i}, \mathsf{host_j}, \mu, \mathsf{bsn}, \mathsf{delivered} \rangle$.
  - Output (SIGNPROCEED, sid, ssid, μ, bsn) to $\mathsf{tpm_i}$.
  - SIGN PROCEED: On input (SIGN PROCEED, sid, ssid) from $\mathsf{tpm_i}$
    - Update the records $\langle \mathsf{ssid}, \mathsf{tpm_i}, \mathsf{host_j}, \mu, \mathsf{bsn}, \text{delivered} \rangle$.
    - Output (SIGNCOMPETE, sid, ssid) to $\mathcal{S}$.
  - SIGNATURE GENERATION: On the input (SIGNCOMPETE, 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 $\mathsf{bsn} \neq \perp$, then retrieve gsk from the $\langle \mathsf{tpm_i}, \mathsf{bsn}, gsk \rangle \in$ DomainKeys.
    - If $\mathsf{bsn} = \perp$ or no gsk was found, generate a fresh key $gsk \leftarrow Kgen(1^\lambda)$.
    - Check CheckGskHonest(gsk)=1.
    - 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 identify(σ, μ, bsn, gsk)=1.
    - Check the is no TPM other than $\mathsf{tpm_i}$ with key gsk' registered in Members or DomainKeys such that identify(σ, μ, bsn, gsk')=1.
    - 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 $\mathsf{host_j}$.
- **VERIFY**: On input (VERIFY, sid, μ, bsn, σ, RL) from V
  - Extract all pairs (gsk_i, $\mathsf{tpm_i}$) from the DomainKeys and Members, for which identify(σ, μ, bsn, gsk)=1.
  - Set $f = 0$ if any of the following holds:
    - More than one key gsk_i was found.
    - I is honest and no pair ($gsk_i, \mathsf{tpm_i}$) was found.
    - An honest $\mathsf{tpm_i}$ was found, but no entry $\langle \star, \mu, \mathsf{tpm_i}, \mathsf{bsn} \rangle$ was found in Signed.

    - There is a key $gsk' \in RL$, such that identify(σ, μ, bsn, gsk')=1, and no pair ($\mathsf{tpm_i}, \mathsf{gsk_i}$) for honest $\mathsf{tpm_i}$ was found.
  - If $f \neq 0$, set f=ver(σ, μ, bsn).
  - Add (σ, μ, bsn, RL, f) to VerResults, output (VERIFIED, sid, f) to V.

- **LINK**: On the input (LINK, sid, σ_1, μ_1, σ_2, μ_2, bsn) from V
  - Output $\perp$ if at least one of the signatures (σ_1, μ_1, bsn) or (σ_2, μ_2, bsn) is not valid.
  - For each gsk_i in Members and DomainKeys, compute $b_i \leftarrow$ identify(σ_1, μ_1, bsn, gsk_i) and b'_i= identify(σ_2, μ_2, bsn, gsk_i) then set:
    - $f \leftarrow 0$ if $b_i \neq b'_i$ for some i.
    - $f \leftarrow 1$ if $b_i = b'_i = 1$ for some i.
  - If f is not defined, set f=link(σ_1, μ_1, σ_2, μ_2, bsn), and output (LINK, sid, f) to V.

Figure D.31: Game 16 for F