L-DAA: Lattice-Based Direct Anonymous Attestation

Nada EL Kassem[1], Liqun Chen[1], Rachid El Bansarkhani[2], Ali El Kaafarani[3], Jan Camenisch[4], and Patrick Hough[3]

[1] University of Surrey, UK
{ n.elkassem@surrey.ac.uk, liqun.chen@surrey.ac.uk}
[2] TU Darmstadt, Germany
elbansarkhani@cdc.informatik.tu-darmstadt.de
[3] University of Oxford, UK
{elkaafarani@maths.ox.ac.uk, patrick.hough@maths.ox.ac.uk}
[4] IBM Reaserch-Zurich, Switzerland
jca@zurich.ibm.com

Abstract. Direct Anonymous Attestation (DAA) is an anonymous digital signature that aims to provide both signer authentication and privacy. DAA was designed for the attestation service of the Trusted Platform Module (TPM). In this application, a DAA signer role is divided into two parts: the principal signer which is a TPM, and an assistant signer which is a standard computing platform in which the TPM is embedded, called the Host. A design feature of a DAA solution is to make the TPM workload as low as possible. This paper presents a lattice-based DAA (L-DAA) scheme to meet this requirement. Security of this scheme is proved in the Universally Composable (UC) security model under the hard assumptions of the Ring Inhomogeneous Short Integer Solution (Ring-ISIS) and Ring Learning With Errors (Ring-LWE) problems. Our L-DAA scheme includes two building blocks, one is a modification of the Boyen lattice based signature scheme and another is a modification of the Baum et al. lattice based commitment scheme. These two building blocks may be of independent interest.

Keywords: Lattice based cryptography, Direct Anonymous Attestation, Universally Composable security model.

1 Introduction

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. A signer consists of the (Host, TPM) pair, and can prove their membership by providing a DAA signature to a verifier. The verifier validates the existance of the membership credential from the given signature without knowing the credential, so, the verifier learns nothing about the identity of the signer. Compared with another type of membership based anonymous digital signtures, group signatures, DAA does not support the property of traceability, i.e., a group manager can identify the signer from a given group signature. 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 abusing anonymity, DAA provides two alternative properties as the replacement of the traceability. One is the rogue signer detection, i.e., with a signer's private signing key anyone can check whether or not a given DAA signature was created under this key. 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.

Related Work As stated by the developer of the TPM specifications, the trusted computing group [24], more than a billion devices include the TPM technology; virtually all enterprise PCs, many servers and embedded systems include the TPM. Every TPM supports DAA. The existing DAA schemes used in the TPMs are based on either the factorisation problem in the RSA setting or the discrete logarithm problem in the Elliptic-Curve (EC) setting. The concept and first DAA scheme was proposed in 2004 by Brickell, Camenisch, and Chen [4]. This scheme is called RSA-DAA and supported by the TPM version 1.2. Later, Brickell, Chen and Li proposed the first EC-DAA scheme based on symmetric pairings [5, 6]. There are many EC-DAA schemes, which improve the performance of this scheme. Two EC-DAA schemes, based on asymmetric (Type 3) pairings, are supported by the TPM version 2.0 [7, 12, 13].

Since the factorisation problem and discrete logarithm problem are known to be vulnerable to quantum computer attacks [22], all the existing DAA schemes will not be secure in the post-quantum computer age. In this paper, we design a lattice based DAA (L-DAA) scheme suitable for inclusion in a future TPM.

This L-DAA scheme is developed from the scheme recently proposed by El Bansarkhani and El Kaafarani [1], Section 7 gives a brief overview of this scheme. Our L-DAA scheme is designed to reduce the demands on the TPM, both in terms of storage costs and computational resources. Table 1 provides a comparison between the two schemes, the detailed comparison between these two schemes will be given after they are implemented. This will be addressed in a future work. The security of both of these L-DAA schemes is based on the hardness of the Ring Inhomogeneous Short Integer Solution (Ring-ISIS) and Ring Learning With Errors (Ring-LWE) problems. As there is no known quantum algorithm that solves either of these lattice based problems, this provides a promising DAA scheme for the post-quantum age.

Contribution The proposed L-DAA scheme makes use of two building blocks. The first is a modification of the Boyen lattice based signature [3], which combines a TPM signing key with a DAA credential in an efficient way. The second is a modification of the Baum et al lattice based commitment scheme [2], which allows a TPM and its Host to jointly create a commitment, where each of them commits and proves his own secret knowledge and the combination of these two proved commitments is a DAA siganture. These two building blocks may have independent interests.

The rest of this paper is structured as follows. In the next section, we introduce the relevant lattice-based problems on which we build the security of our L-DAA scheme. In Section 3, we introduce the two building blocks that will be used to create our L-DAA scheme. In Section 5, we describe our L-DAA scheme, which will be followed by a sketched security proof in Section 6. Finally, we discuss the performance of the L-DAA scheme in Section 7 and conclude the paper in Section 8. In Appendices A and B, we present the security analysis of our modification of the Boyen signature scheme and Baum et al commitment scheme. Appendix C presents a discussion on several functionalities necessary for our L-DAA security proof, a detailed security proof of the proposed L-DAA scheme is presented in Appendix D.

2 Preliminaries

Notations The following notation will be used throughout the paper. $[d]$ is the set $\{1, \ldots, d\}$ for a positive integer d. $x \hookleftarrow S$ means that x is a uniformly random sample drawn from S. $\mathbb{Z}_q$ represents the quotient ring $\mathbb{Z}/q\mathbb{Z}$. $\mathbf{a}, \mathbf{b}, \ldots$ in bold letters represent vectors with entries in $\mathbb{Z}_q$. We can also represent a polynomial whose coefficients are in $\mathbb{Z}_q$ as a vector whose entries are the coefficients of that polynomial. $\|\mathbf{a}\|_\infty$ denotes the infinity norm of polynomial $\mathbf{a}$, $\|\mathbf{a}\|_\infty = \max_i |a^i|$ where a^i are the coefficients of the polynomial. $\hat{A} = (\mathbf{a}_1, \ldots, \mathbf{a}_m)$ is a vector of polynomials where m is some positive integer and $\mathbf{a}_1, \ldots, \mathbf{a}_m$ are polynomials. $\|\hat{A}\|_\infty$ is the infinity norm of the vector of polynomials $\hat{A}$ defined by $\|\hat{A}\|_\infty = \max_i \|\mathbf{a}_i\|_\infty$. B_{3n} denotes the set of vectors $\mathbf{u} \in \{-1, 0, 1\}^{3n}$ having excatly 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 convenience, throughout this paper, we will describe the commitment algorithm using the same notation, as it is used for a signature based proof SPK:

$$\theta_{t/h} = \mathsf{COM}\Big\{\mathsf{public}, \mathsf{witness} : \mathsf{construction}\Big\}.$$

Parameters Throughout this paper we will use the polynomial rings $\mathcal{R}_q = \mathbb{Z}_q[x]/\langle \mathbf{f}(x) \rangle$, such that $\mathbf{f}(x)$ is a monic irreducible polynomial over $\mathcal{R}$, we consider only cyclotomic polynomials $\mathbf{f}(x) = x^n + 1$, with n being a power of 2 (this restriction on n may not be required for the Ring-LWE problem as it was shown in [20]). Let $q \geq 2$ represents an integer modulus such that $q = poly(n)$. For correctness, we require the main hardness parameter n, to be large enough (e.g., $n \geq 100$) and $q > \beta$ as both being at least a small polynomial in n. We also let $m = O(\log q)$. A concrete choice of parameters from [8] can be as follows: $n = 256,\ q = 8380417,\ m = 14$, and $\beta = 275$.

Definition 1 (Lattices [14]). *Let $\mathbf{b}_1, \mathbf{b}_2, \cdots, \mathbf{b}_n$ be linearly independent vectors over $\mathbb{R}^m$, the lattice spanned by these vectors is given by*

$$L = \Big\{ \sum_{i=1}^{n} z_i \mathbf{b}_i : z_i \in \mathbb{Z} \Big\}$$

The vectors $\mathbf{b}_1, \mathbf{b}_2, \cdots, \mathbf{b}_n$ are called a basis of the lattice. Let $B = [\mathbf{b}_1 | \mathbf{b}_2 | \cdots | \mathbf{b}_n] \in \mathbb{R}^{m \times n}$ having the basis vectors as columns. The lattice generated by B is denoted by $L(B)$, and 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 (Discrete Gaussian Distributions [21]). *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 $exp(-\pi(\|\mathbf{x}\|/s)^2)$.*

Definition 3 (Shortest Vector Problem (SVP) [19]). *Given an arbitrary basis B of some lattice $L = L(B)$, find a shortest nonzero lattice vector, i.e., a $\boldsymbol{v} \in L$ for which $\|\boldsymbol{v}\| = \lambda_1(L)$ (where $\lambda_1(L)$ is the length of a shortest nonzero lattice vector).*

Definition 4 (The Ring Short Integer Solution Problem (Ring-SIS$_{n,m,q,\beta}$) [19]). *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 $\hat{Z} \in \mathcal{R}_q^m$ of norm $\|\hat{Z}\|_\infty \leq \beta$ such that:*

$$f_{\hat{A}}(\hat{Z}) = \sum_{i \in [m]} \mathbf{a}_i \cdot \mathbf{z}_i = \boldsymbol{0} \in \mathcal{R}_q$$

The Ring Inhomogeneous Short Integer Solution (Ring-ISIS$_{n,m,q,\beta}$) problem asks to find $\hat{Z} = (\mathbf{z}_1, \mathbf{z}_2, \cdots, \mathbf{z}_m) \in \mathcal{R}_q^m$ of norm $\|\hat{Z}\|_\infty \leq \beta$, and such that:

$$f_{\hat{A}}(\hat{Z}) = \sum_{i \in [m]} \mathbf{a}_i \cdot \mathbf{z}_i = \boldsymbol{y} \in \mathcal{R}_q$$

for some uniform random polynomial $\boldsymbol{y}$.

Definition 5 (The Ring Learning With Error Problem (Ring-LWE) [21]). *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, 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 $\mathbf{s} \hookleftarrow \mathcal{R}_q$.*

3 Building Blocks

In this section, we presents a modified Boyen signature scheme and a modified Baum et al commitment scheme, which will be used as two building blocks of our L-DAA scheme presented in Section 5. These two modifications have their independent interests.

3.1 A Modification of the Boyen Signature Scheme

We first recall the Boyen's signature scheme [3], which is 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 following algorithms:

– KeyGen(1^λ):
  1. Generates a vector of polynomials $\hat{A} \in \mathcal{R}_q^m$ together with a trapdoor $\hat{T}$.
  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 a vector of polynomials $\hat{Z} = (\mathbf{z}_1, \cdots, \mathbf{z}_{2m}) \hookleftarrow \mathcal{D}_{L^\perp_{\mathbf{u}(\hat{A}_{\mathsf{id}}),s}}$, such that $\hat{A}_{\mathsf{id}} \cdot \hat{Z} \equiv \mathbf{u} \mod q$ and $\|\hat{Z}\|_\infty \leq \beta$.
  3. Outputs the signature $\hat{Z} = (\mathbf{z}_1, \cdots, \mathbf{z}_{2m})$.

– Verify$(pk,\ \mathsf{id},\ \hat{Z})$: 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.

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 [3] for the security proof. The proof was improved later in [16] by using a new trapdoor and ring analogue.

In order to create a DAA credential in our L-DAA scheme presented in Section 5, we modified the ring variant of Boyen's signature scheme [17] as follows:

– KeyGen(1^λ) samples one more uniform random vector of polynomials $\hat{A}_t \in \mathcal{R}_q^{m'}$, where $m' \leq m$, 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 $\hat{Z}_t = (\mathbf{z}_1, \cdots, \mathbf{z}_{m'}) \hookleftarrow \mathcal{D}^{m'}_{\mathbb{Z}^n,s}$ 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 a vector of polynomials $\hat{Z}_h = (\mathbf{z}_{m'+1}, \cdots, \mathbf{z}_{m'+2m}) \hookleftarrow \mathcal{D}_{L^\perp_{\mathbf{u}_h}(\hat{A}_{\mathsf{id}}),s}$, 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 $\hat{Z} = [\hat{Z}_t|\hat{Z}_h] = (\mathbf{z}_1, \cdots, \mathbf{z}_{m'+2m})$.

– 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.

In the L-DAA scheme, for the simplicity of the persentation of the scheme, we let $m' = m$. The security of this modified Boyen signature scheme is based on the original Boyen signature scheme which is unforgeable under the hard assumptions of the SIS problem [3]. The unforgeability of the modified Boyen signature can be reduced to the existential unforgeability of the original Boyen signature scheme. A detailed analysis will be given in Appendix A.

3.2 A Modification of the Baum et al Commitment Scheme

We first briefly describe the Baum et al. commitment scheme presented in [2] that includes the following algorithms:

– 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 $\mathcal{R}_q$, $\mathcal{R}_q = \mathbb{Z}_q[x]/\langle \mathbf{f}(x)\rangle$ where $\mathbf{f}(x)$ is a monic and irreducible polynomial over $\mathbb{Q}$, 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

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

The security of this commitment scheme is based on the hardness of the Ring-ISIS problem and we refer to [2] for the security proof. In order to create a DAA signature, which is jointly signed by a TPM and its Host, we modify the above Baum et al. commitment scheme by allowing two parties to commit a set of secret values jointly. This modification is based on the additional homomorphic property of this scheme.

Let $\hat{S}_t \in \mathcal{R}_q^{l_t}$, and $\hat{S}_h \in \mathcal{R}_q^{l_h}$ be TPM and the host's corresponding independent inputs respectively (to be concatenated), $\mathbf{s}_t$ and $\mathbf{s}_h$ in $\mathcal{R}_q$ be the TPM and the host's corresponding inputs to be added.

The commitment algorithm performed by the TPM and Host works as follows:

To commit 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 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 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 $\|\hat{R}\|_\infty < \alpha_t + \alpha_h$.

In summary, we modify the original Baum et al scheme by splitting the prover into two entities; in the L-DAA scheme, these two entities are the TPM and the Host. The original Baum et al. scheme was proved to hold the properties of statistically hidding 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 will be given in Appendix B of this paper.

4 Security Model of DAA

In this paper, we follow the security model for DAA given by Camenish et al. in [9]. The security definition is given in the Universal Composability (UC) model with respect to an ideal functionality F^l_{daa}. In UC, an environment ε should not be able to distinguish with a non negligible probability between two worlds:

1. The real world, where each part in the DAA protocol Π executes its assigned part of the protocol. The network is controlled by an adversary $\mathcal{A}$ that communicates with ε.
2. The ideal world, in which all parties forward their inputs to a trusted third party, called the ideal functionality F^l_{daa}, which internally performs all the required tasks and creates the party's outputs.

A protocol Π is said to securely realize F^l_{daa} if for every adversary $\mathcal{A}$ performing an attack in the real world, there is an ideal world adversary $\mathcal{S}$ that performs the same attack in the ideal world. More precisely, given a protocol Π, an ideal functionality F^l_{daa} and an environment ε, we say that Π securely realises F^l_{daa} if the real world in which Π is used is as secure as the ideal world in which F^l_{daa} is used. In other worlds, for any adversary $\mathcal{A}$ in the real world, there exists a simulator $\mathcal{S}$ in the ideal world such that $(\varepsilon, F^l_{daa}, \mathcal{S})$ is indistinguishable from $(\varepsilon, \Pi, \mathcal{A})$.
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 can't 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 existing 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 the host are supposed to be honest. In [10] 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}

We now formally define the ideal functionality F^l_{daa} 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 below. In the UC model as in [9], 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 define the JOIN and SIGN sub-sessions by jsid and ssid. F^l_{daa} is paramitrized 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 algorithms that will be used inside the functionality as follows:

- Kgen(1^λ): A probabilistic algorithm that takes a security parameter λ and generates keys gsk for honest TPMs.
- sig(gsk, μ, bsn): A probabilistic algorithm used for honest TPMs. On input of a key gsk, a message μ and a basename bsn, it out puts a signature σ.
- ver(σ, μ, 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 out puts $f = 1$ if the signature is valid, $f = 0$ otherwise.
- link(σ_1, μ_1, σ_2, μ_2, 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, 0 otherwise.

– 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.

We now define useful functions to check whether or not a TPM key is consistent with the internal records of F^{l}_{daa}. We distinguish between the two cases whether a TPM is honest or corrupt as follows:

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.

We now explain the interfaces of F^{l}_{daa} and identify the checks, labed in roman numerals, that are done by the ideal functionality.

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.

JOIN

1. JOIN REQUEST: On input (JOIN, $\mathsf{sid},\ \mathsf{jsid},\ \mathsf{tpm_i}$) 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, $\mathsf{sid},\ \mathsf{jsid},\ \mathsf{tpm_i},\ \mathsf{host_j}$) to $\mathcal{S}$.
2. JOIN REQUEST DELIVERY: Proceed upon recieving 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, $\mathsf{sid},\ \mathsf{jsid},\ \mathsf{tpm_i}$) 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, $\mathsf{sid},\ \mathsf{jsid}$) to $\mathcal{S}$.
4. KEY GENERATION: On input (JOINCOMPLETE, $\mathsf{sid},\ \mathsf{jsid},\ gsk$) 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 $\mathsf{host_j}$.

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 (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 (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 $\mathsf{host_j}$.

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 $\langle\star,\ \mu,\ \mathsf{tpm_i},\ \mathsf{bsn}\rangle$ 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 $\langle\sigma,\ \mu,\ \mathsf{bsn},\ RL,\ b\rangle$ to VerResults, and output (VERIFIED, sid, b) to V.

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.

5 The L-DAA Scheme

We now present our L-DAA scheme. Security of this scheme is based on the Ring-ISIS problem and Ring-LWE problem. The DAA credential is a modification of the Boyen signature [3, 15], that was described in Subsection 3.1 and its security is proved in Appendix A. Before proceeding with the L-DAA scheme, we define some standard functionalities that are used in the TPM technology, as specified in [9], a detailed describtions of these funtionalities is presented in Appendix C of this paper.

- F_{ca} is a common certificate authority functionality that is available to all parties.
- $F^{\mathcal{D}}_{crs}$ 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^{l}_{smt} 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 as follows:

SETUP: F_{crs} creates the system parameters: $\mathsf{sp} = (\lambda, q, n, m, \mathcal{R}_q, c, \beta, \beta', \ell)$, where λ and c 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 in the Boyen signature.

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. Issuer's public key is $\mathsf{pp} = (\mathsf{sp}, \hat{A}_t, \hat{A}_I, \hat{A}_0, \hat{A}_1, ..., \hat{A}_\ell,\ \mathbf{u},\ \mathcal{H}, \mathcal{H}_0)$, where $\hat{A}_t, \hat{A}_I, \hat{A}_i(i = 0, 1, ..., \ell) \in \mathcal{R}_q^m$, $\mathbf{u} \in \mathcal{R}_q$, $\mathcal{H} : \{0,1\}^* \to \mathcal{R}_q$, and $\mathcal{H}_0 : \{0,1\}^* \to \{1,2,3\}^c$. Issuer's private key is $\hat{T}_I$, which is the trapdoor of $\hat{A}_I$ and $\|\hat{T}_I\|_\infty \leq \beta$.

The issuer initializes the list of joining members Memebers $\leftarrow \emptyset$. The issuer proves that his secret key is well formed in π_I, and registers the key $(\hat{T}_I, \pi_I)$ with F_{ca} and outputs (SETUPDONE, sid).

JOIN: The Join process is a protocol running between the Issuer I and a platform, consisting of a TPM $\mathsf{tpm_i}$ and a Host $\mathsf{host_j}$ (with an identifier id). More than one Join session 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 make 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 phases, Join request and Join proceed, as follows:

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 $(\mathsf{sid},\ \mathsf{jsid},\ \rho, \mathsf{bsn}_I)$ 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: $\hat{X}_t = (\mathbf{x}_1, \cdots, \mathbf{x}_m) \hookleftarrow \mathcal{R}_q^m$ with the condition $\|\hat{X}_t\|_\infty \leq \beta$, and stores its key as $(\mathsf{sid},\ \mathsf{host_j},\ \hat{X}_t, \mathsf{id})$.
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:
$$\begin{gathered}\pi_{\mathbf{u}_t} = \mathsf{SPK}\Big\{\mathsf{public} := \{\mathsf{sp},\ \hat{A}_t,\ \mathbf{u}_t,\ \mathsf{bsn}_I,\ \mathsf{nym}_I\},\\ \mathsf{witness} := \{\hat{X}_t = (\mathbf{x}_1, \cdots, \mathbf{x}_m),\ \mathbf{e}_I\} :\\ \mathbf{u}_t = \hat{A}_t \cdot \hat{X}_t \mod q\ \wedge\ \|\hat{X}_t\|_\infty \leq \beta\ \wedge\ \mathsf{nym}_I = \mathcal{H}(\mathsf{bsn}_I) \cdot \mathbf{x}_1 + \mathbf{e}_I\\ \mod q\ \wedge\ \|\mathbf{e}_I\|_\infty \leq \beta'\Big\}(\rho).\end{gathered}$$
4. It sends $(\mathsf{nym}_I,\ \mathsf{id},\ \mathbf{u_t},\ \pi_{\mathbf{u_t}})$ to the issuer I via the host by means of F_{auth^*}, i.e., it gives F_{auth^*} an input (SEND, $(\mathsf{nym}_I,\ \pi_{\mathbf{u}_t}), (\mathsf{sid},\ \mathsf{tpm_i}, I)$, jsid, $\mathsf{host_j}$).

The host, upon receiving (APPEND, $(\mathsf{nym}_I,\ \pi_{\mathbf{u}_t}), (\mathsf{sid},\ \mathsf{tpm_i}, I)$) from F_{auth^*}, forwards it to I by sending (APPEND, $(\mathsf{nym}_I,\ \pi_{\mathbf{u}_t}), (\mathsf{sid},\ \mathsf{tpm_i},\ I)$) to F_{auth^*} and keeps the state $(\mathsf{jsid},\ \mathbf{u_t},\ \mathsf{id})$. I upon receiving (SENT, $(\mathsf{nym}_I,\ \pi_{\mathbf{u}_t}), (\mathsf{sid},\ \mathsf{tpm_i}, I)$, jsid, $\mathsf{host_j}$) from F_{auth^*}, it verifies the proof $\pi_{\mathbf{u}_t}$ to make sure that $\mathsf{tpm_i} \notin$ Members. I stores $(\mathsf{jsid},\ \mathsf{nym}_I,\ \pi_{\mathbf{u}_t},\ \mathsf{id}, \mathsf{tpm_i},\ \mathsf{host_j})$, and generates the message (JOINPROCEED, sid, jsid, id, $\pi_{\mathbf{u}_t}$).

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 $\hat{X}_h = (\mathbf{x}_{m+1}, ...,$ $\mathbf{x}_{3m})$ 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 recieves 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 check is correct, then $\mathsf{host_j}$ stores (sid, $\mathsf{tpm_i}$, id, $\hat{X}_h$, $\mathbf{u}_t$) and outputs (JOINED, sid, jsid).

SIGN: After obtainng the credential from the Join process, $\mathsf{tpm_i}$ and $\mathsf{host_j}$ can sign a message μ with respect to a basename bsn. We use a unique sub-session identifier ssid to allow multiple Sign sessions. Each session has two phases, Sign request and Sign proceed.
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).

Sign Proceed: When $\mathsf{tmp_i}$ gets permission to proceed for ssid, the TPM proceeds as follows:

1. It retrieves the records (sid, id, $\mathsf{host_j}$, $\pi_{\mathbf{u}_t}$) and (sid, ssid, bsn, μ).
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 3.2:

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

If $\mathsf{bsn}=\perp$, the $\mathsf{tpm_i}$ samples a random value $\mathsf{bsn} \leftarrow \{0,1\}^\lambda$, and then follows the previous case.

3. $\mathsf{tpm_i}$ sends (sid, ssid, θ_t, μ) to $\mathsf{host_j}$.
4. When $\mathsf{host_j}$ recieves the message (sid, ssid, θ_t, μ), it checks that the proof θ_t is valid, and subsequently generates a commitment again as described in Subsection 3.2:
$$\theta_h = \mathsf{COM}\Big\{\mathsf{public} := \{\mathsf{sp},\ \hat{A}_h,\ \mathbf{u}_h,\ \mu,\ \theta_t\}, \\ \mathsf{witness} := \{\hat{X}_h = (\mathbf{x}_{m+1}, \cdots, \mathbf{x}_{3m}),\ \mathsf{id}\} : \\ \{\hat{A}_h \cdot \hat{X}_h = \mathbf{u}_h \ \wedge\ \|\hat{X}_h\|_\infty \leq \beta\}\Big\}.$$
The combination of these two commitments θ_t and θ_h as described in Subsection 3.2 follows the additional 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):
$$\pi = \mathsf{SPK}\Big\{\mathsf{public} := \{\mathsf{pp},\ \mathsf{nym},\ \mathsf{bsn}\}, \\ \mathsf{witness} := \{\hat{X} = (\mathbf{x}_1, \cdots, \mathbf{x}_{3m}),\ \mathsf{id},\ \mathbf{e}\} : \\ [\hat{A}_t|\hat{A}_h] \cdot \hat{X} = \mathbf{u} \ \wedge\ \|\hat{X}\|_\infty \leq \beta \ \wedge\ \mathsf{nym} = \mathcal{H}(\mathsf{bsn}) \cdot \mathbf{x}_1 + \mathbf{e} \\ \bmod q \ \wedge\ \|\mathbf{e}\|_\infty \leq \beta'\Big\}(\mu).$$
The details of the θ_t, θ_h and π computation will be given below.
6. $\mathsf{host_j}$ outputs the L-DAA signature $\sigma = (\mathsf{nym}, \mathsf{bsn}, \pi)$.

VERIFY: The verify algorithm allows anyone to check whether a signature σ on a message μ with respect to a basename bsn is valid. Let RL denotes a revocation list with all the rogue TPM's secret keys $\hat{X}_t$. 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 $\mathbf{u}$, then verifies 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.

LINK: The link algorithm 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. 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 are invalid, the verifier outputs $\perp$.
3. Otherwise if $\|\mathsf{nym} - \mathsf{nym}'\|_\infty < 2\beta'$, the verifier outputs 1 (linked); otherwise 0 (not linked).

The details of θ_t, θ_h and π: Now we explain the details on how to compute θ_t, θ_h and π.

Let $k = \lfloor \log \beta \rfloor + 1$ and let $\{\beta_1, ..., \beta_k\} \in \{0,1\}^k$ be the binary representation of β. 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 any linear transformation into matrix vector product. We construct the matrices $\bar{A}_i = \mathsf{rot}(\mathbf{a}_i)$ 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.
Let's consider the following extensions:

– $\mathsf{id} = \{\mathsf{id}_1, ..., \mathsf{id}_\ell\} \in \{0,1\}^\ell$ 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.
– $\bar{A}_i^* = [\bar{A}_i | \mathbf{0} \in \mathbb{Z}^{n\times 2n}]$ for $i = 1$ *to* $i = (3+\ell)m$.

Apply the techniques of decomposition and extension described in [18] on each of the vectors of $\hat{X}$ and the vector $\mathbf{e}$, to get the vectors:

$$\{\mathbf{e}_j\}_{j=1}^k, \{\mathbf{x}_1^j\}_{j=1}^k, \{\mathbf{x}_2^j\}_{j=1}^k, \dots, \{\mathbf{x}_{3m}^j\}_{j=1}^k \in B_{3n}$$

Let $\bar{A}^*_{i+(j+2)m} = \mathbf{0}$ for $j > \ell$, and let $\mathbf{x}_i = \mathbf{0} \in \mathbb{Z}^n$ for $3m < i \le (3+2\ell)m$. Now we have,

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

Before proceeding with the proof, the prover:

1. Samples the following masking vectors: $\{\mathbf{r}_{\mathbf{e}}^j \hookleftarrow \mathbb{Z}_q^{3n}\}_{j=1}^k$, $\{\mathbf{r}_i^j \hookleftarrow \mathbb{Z}_q^{3n}\}_{j=1}^k$ for $i \in [3m]$ and $j \in [k]$, and $\mathbf{r}_{\mathsf{id}^*} \hookleftarrow \mathbb{Z}_q^{2\ell}$.
2. Defines the following terms: $D = [\mathsf{rot}(\mathcal{H}(\mathsf{bsn})) | \mathbf{0}] \in \mathbb{Z}_q^{n\times 3n}$, $\mathbf{v}_i^j = \mathbf{x}_i^j + \mathbf{r}_i^j$, $\mathbf{v}_{\mathbf{e}}^j = \mathbf{e}^j + \mathbf{r}_{\mathbf{e}}^j$, and $\mathbf{v}_{\mathsf{id}^*} = \mathsf{id}^* + \mathbf{r}_{\mathsf{id}^*}$.
3. Samples the permutations as follows: $\tau \hookleftarrow \mathsf{S}_{2\ell}$ for id^*, $\{\delta_j \hookleftarrow \mathsf{S}_{3n}\}_{j=1}^k$ for $\hat{X}_{h_1}$, $\{\psi_j \hookleftarrow \mathsf{S}_{3n}\}_{j=1}^k$ for $\hat{X}_{h_2}$, where $\hat{X}_h = [\hat{X}_{h_1} \in \mathcal{R}_q^m | \hat{X}_{h_2} \in \mathcal{R}_q^m]$, $\{\phi_j \hookleftarrow \mathsf{S}_{3n}\}_{j=1}^k$ for $\hat{X}_t$, and $\{\varphi_j \hookleftarrow \mathsf{S}_{3n}\}_{j=1}^k$ for $\mathbf{e}$.

Now, we are ready to explain the result. The commitment algorithm COM used below is as explained in Subsection 3.2.

θ_t: For the TPM's commitment θ_t, the commitment is $CMT_t = (\mathbf{C}_{t1}, \mathbf{C}_{t2}, \mathbf{C}_{t3})$ where:

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

θ_h: For the host commitment θ_h, the commitment is $CMT_h = (\mathbf{C}_{h1}, \mathbf{C}_{h2}, \mathbf{C}_{h3})$ where:

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

π: $\mathsf{tpm_i}$ hands out the commitments of the total c rounds to $\mathsf{host_j}$, $\mathsf{host_j}$ then adds it's own commitments homomorphically to the TPM's commitments. The homomorphic addition of the commitments generates the resulting commitments $CMT = (\mathbf{C}_1, \mathbf{C}_2, \mathbf{C}_3)$ where:

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

The following step is the Fiat-Shamir transformation, which has been used in the existing DAA schemes. The only difference is that the hash-function output is used as a random distribution of $\{1, 2, 3\}^c$.

Challenge: $\mathsf{host_j}$ generates the challenges using the Fiat-Shamir's hash-function transformation, which is based on a random oracle, which should only include CMT:

$$\{\mathbf{CH}_j\}_{j=1}^c = \mathcal{H}_0(\mu, \{CMT_j\}_{j=1}^c, \mathsf{pp}) = \{1, 2, 3\}^c.$$

Response: For each challenge, $\mathsf{tpm_i}$ sends it's own response to $\mathsf{host_j}$, then $\mathsf{host_j}$ provides it's own response and combines the two responses together. Finally $\mathsf{host_j}$ sends the proof to the verifier. The resulting responses are treated as follows:

– **CH = 1** : reveal $\mathbf{C}_2$ and $\mathbf{C}_3$, i.e., output all the permuted $\tau(\mathsf{id}^*)$, $\tau(\mathbf{r}_{id^*})$, $\{\phi_j(\mathbf{x}_i^j)\}_{j=1}^k$, $\{\delta_j(\mathbf{x}_i^j)\}_{j=1}^k$, $\{\psi_j(\mathbf{x}_i^j)\}_{j=1}^k$, $\{\varphi_j(\mathbf{e}^j)\}_{j=1}^k$, $\{\varphi_j(\mathbf{r}_e^j)\}_{j=1}^k$, $\{\phi_j(\mathbf{r}_i^j)\}_{j=1}^k$, $\{\delta_j(\mathbf{r}_i^j)\}_{j=1}^k$, $\{\psi_j(\mathbf{r}_i^j)\}_{j=1}^k$.
– **CH = 2** : reveal $\mathbf{C}_1$ and $\mathbf{C}_3$, i.e., output all the permutations $\tau, \{\phi_j\}_{j=1}^k$, $\{\delta_j\}_{j=1}^k$, $\{\psi_j\}_{j=1}^k, \{\varphi_j\}_{j=1}^k$. and all the $\mathbf{r}$ values.
– **CH = 3** : reveal $\mathbf{C}_1$ and $\mathbf{C}_2$, i.e., output all the permutations $\tau, \{\phi_j\}_{j=1}^k$, $\{\delta_j\}_{j=1}^k$, $\{\psi_j\}_{j=1}^k, \{\varphi_j\}_{j=1}^k$. and all the $\mathbf{v}$ values.

Verification: Depending on the prover's inputs, the verifier can always check 2 out of 3 commitments. Note that the responses to all 3 commitments allows one to deduce the witness.

6 Security Proof

(*sketch*) In this section, we provide a sketch of the security proof and will give the detailed proof in Appendix D. During the proof, we present a sequence of games based on Camenish et al. in [9] (also recalled in Section 4), and show that there exists no environment ε that can distinguish the real world protocol Π with an adversary $\mathcal{A}$, from the ideal world F_{daa}^l with a simulator $\mathcal{S}$. 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. We will explain the sequence of games as follows:
Game 1: This is the real world protocol.
Game 2: An entity C is introduced, C receives all inputs from the honest parties and simulates the real world protocol for them. This is equivilent to Game 1.
Game 3: We now split C into two parts, F and $\mathcal{S}$, where 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 outputs to F. F then forwards the outputs to ε. This game is simply Game 2 but with different structure, so Game 3=Game 2.
Game 4: F now behaves differently in the setup interface, it stores the algorithms for the issuer I, F also does checks and ensures that the stucture 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. Clearly ε will notice no change, so Game 3=Game 4.
Game 5: F now performs the verification and linking checks instead of forwarding them to $\mathcal{S}$. There are no protocol messages and the outputs are excatly as in the real world protocol. However, the only difference is that the verification algorithm that F uses doesn't contain a revocation check, so F can perform this check separately so the outcomes are equal, Game 4=Game 5.
Game 6: The join interface of F is now changed, F stores in it's records the

members that joined. If I is honest, F stores the secret key gsk, extracted from $\mathcal{S}$, for corrupt TPM's. $\mathcal{S}$ 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 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 fully corrupted platforms don't matter. In the only case, when the TPM is already registered in Members, F may abort the protocol, but I has already tested this case before continuing with the query JOINPROCEED, hence F will not abort. Thus in all cases, F and $\mathcal{S}$ can interact to simulate the real world protocol, and Game 6=Game 5.
Game 7: In this game, F 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, subsequent inputs are then forwarded to S. For $i < k$, F handles all the signing queries with tpm_i using algorithms. For $i > k$, F forwards all signing queries with tpm_i to $\mathcal{S}$ who creates signatures as before. Now from the definition of Game 7.k.k, 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 TPM's, and Game 7 is indistinguishable from Game 7.k.k'.
We want to prove now that Game 7.k.k'+1 is indistinguishable from Game 7.k.k'. Suppose that there exists an environment that can distinguish a signature of an honest party using $gsk = \hat{X}_t$ from a signature using a different $gsk' = \hat{X'}_t$, 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 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 distingisher between Game 7.k.k' and Game 7.k.$k'+1$ can solve the Decision Ring-LWE Problem.
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 $l(\mu'$, $\mathsf{bsn}')$=$l(\mu$, $\mathsf{bsn})$, an environment ε observes no difference, and thus Game

8=Game 7.
Game 9: If I is honest, then F now only allows the platform that joined to sign. An honest host will always check whether it joined with a TPM in the real world protocol, so 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. Therefore Game 9=Game 8.
Game 10: When storing a new $gsk = \hat{X}_t$, F checks CheckGskCorrupt(gsk)=1 or CheckGskHonest(gsk)=1. We want to show that these checks will always pass. In fact, 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 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. Hence Game 10=Game 9.
Game 11: In this game F checks that honestly generated signatures are always valid. This is true as sig algorithm always produces signatures passing through verification checks, also those signatures satisfy $\mathsf{identify}(gsk,\ \sigma,\ \mu,\ \mathsf{bsn}) = 1$ which is checked via nym. F also makes sure, using it's internal records 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, and hence Game 11=Game 10.
Game 12: Add Check-IX to ensure that there are no multiple gsk values matching to one signature. However, since there exists only one pair $(\mathbf{x}_1, \mathbf{e_I})$ such that $\|\mathbf{x}_1\|_\infty < \beta$ and $\|\mathbf{e}_I\|_\infty < \beta'$, satisfying $\mathsf{nym_I} = \mathcal{H}(\mathsf{bsn}) \cdot \mathbf{x}_1 + \mathbf{e_I}$, thus two different $gsk's$ can't share the same $\mathbf{x}_1$, thus any valid signature should be identified to one gsk. Thus Game 12=Game 11.
Game 13: To prevent accepting signatures that were issued by use of join credentials not issued by honest issuer, F adds a further check Check-X. This is due to the unforgeability of Boyen signatures that is based on the hardness of the Ring-ISIS Search Problem, so we get Game 13=Game 12.
Game 14: Check-XI is added to F, this would prevent anyone forging signatures using 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 extracts $\mathbf{x}_1$, and thus breaks the Ring-LWE problem. So Game 14=Game 13.
Game 15: Check-XII is added to F, this ensures that honest TPMs are not being revoked. If an 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 equivilant to solving the search Ring-LWE problem.
Game 16: All the remaing 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 to one

signature and not the other, Game 16 is indistinguishable from Game 15, and F now includes all the functionalities of F^l_{daa}. This concludes the proof. $\square$

7 Performance

Overview of El Bansarkhani and El Kaafarani DAA Scheme [1] This DAA scheme 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 $\mathbf{b} \in \mathcal{R}_q$. The TPM generates a small secret $\hat{Z}_1 \in \mathcal{R}_q^{2m+1}$ such that $[\mathbf{b}|\hat{A}_{\mathsf{id}}][\hat{Z}_1] = \tilde{\mathbf{u}} \mod q$. 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 $\hat{A}_{\mathsf{id}}\hat{Z}_2 = \mathbf{u} - \tilde{\mathbf{u}} \mod q$. The TPM and the host together combine their secret data to obtain a valid credential satisfying $\mathbf{u} = [\mathbf{b}|\hat{A}_{\mathsf{id}}][\hat{Z}_1 + (\mathbf{0}|\hat{Z}_2)]$. To create a signature, the TPM samples a small random vector $\hat{T} \in \mathcal{R}_q^{2m}$, such that $\hat{T}\hat{A}_{\mathsf{id}} \mod q$ is uniform, and shares it with the host in order to randomize the signature. The TPM and the host generate π_2 and π_3 separately, where π_2 proves $\mathbf{u}' = [\mathbf{b}|\hat{A}_{\mathsf{id}}][\hat{Z}_1 + (\mathbf{0}|\hat{T})]$ and π_3 proves $\mathbf{u} - \mathbf{u}' = \hat{A}_{\mathsf{id}}(\hat{Z}_2 - \hat{T})$. Finally, the host outputs the signature $\sigma = (\pi_2,\ \pi_3,\ \mathbf{u}',\ \mu)$.

Size Comparison In our L-DAA scheme, the TPM's secret key size is reduced to $m' \leq m$ polynomials in $\mathcal{R}_q$, instead of $2m+1$ polynomials in [1], while keeping the same credential size. Such a change has a significant contribution in reducing the TPM's computation costs in the join and sign interfaces, as well as reducing the signature size. For instance, the host outputs the L-DAA signature after c rounds of the proof π, the size of the response for each round is bounded by $\mathcal{O}(n)km(2\ell+2)$ elements in $\mathbb{Z}_q$ for the host, and $\mathcal{O}(n)k(m'+1)$ for the TPM. In [1], the size of the response for each round is bounded by $\mathcal{O}(n)km(2\ell+2)$ for the host, and $\mathcal{O}(n)km(2\ell+2)$ for the TPM. Thus in our L-DAA scheme, the signature's size has been significantly reduced especially for large ℓ. The verification key set in [1] 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 with $\ell+2$ vectors of polynomials in $\mathcal{R}_q^m$, a vector of polynomials $\hat{A}_t \in \mathcal{R}_q^{m'}$ and a polynomial $\mathbf{u}$. Note that as we consider m' to be relatively small, then adding $\hat{A}_t$ may only have a slight impact on increasing the size of the verification key set. Table 1 compares the space efficiency between the proposed L-DAA scheme and the scheme presented in [1].

Computation Costs Table 2 compares the computation costs between the proposed L-DAA scheme and the scheme presented in [1] in the join and sign interfaces.

To calculate nym_I, nym, $\mathbf{u}_\mathsf{t}$ and generate one round of $\pi_{\mathbf{u}_t}$ and θ_t in the join and sign interfaces of the L-DAA scheme, the TPM has to perform at most $m'+1$ polynomial multiplications. In [1], the TPM performs at most $2m+2$ polynomial

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

Table 1: Space efficiency in $\mathbb{Z}_q$, with $m' \leq m$

	Join		Sign		Verify	
	This scheme	Scheme in [1]	This scheme	Scheme in [1]	This scheme	Scheme in [1]
TPM	$m'+1$	$2m+2$	$m'+1$	$2m+2$	-	-
Host	$2m$	$2m$	$2m$	$2m$	-	-
Issuer	$m'+1$	$2m+2$	-	-	-	-
Verifier	-	-	-	-	$2m+m'$	$4m+2$

Table 2: Computation Cost, total number of polynomial multiplications in $\mathcal{R}_q$

multiplications for calculating $\mathsf{nym_I}$, $\tilde{\mathbf{u}}$ and generating each round of π_1 and π_2 in the join and sign interfaces respectively. The computation costs for the host in the join interface is $2m$ polynomial multiplications for checking the equality $\mathbf{u}_h = \hat{A}_h \cdot \hat{X}_h$ for both schemes. The Issuer verifies the reponses for each round of $\pi_{\mathbf{u}_t}$, π_1 in both schemes in the join interface. Thus the issuer's computation cost for each round is thus bounded by $m'+1$ for the L-DAA scheme and $2m+2$ in [1]. In both L-DAA and [1], the host performs $2m$ polynomial multiplications for generating one round of θ_h and π_3 in the sign interfaces.

8 Conclusion and Future Work

In this paper, we have presented a lattice based DAA (L-DAA) scheme. Our construction relies on the lattice problems which provides promising security against the known quantum computer attacks. In the future work, the proposed L-DAA scheme will be implemented in the full range of TPM environments, i.e., hardware, software and virtualization environments. The scheme may be further optimised based on the performance evaluation. The final solution of this research will be a L-DAA scheme suitable for inclusion in the future quantum-resistant TPM.

References

1. Rachid El Bansarkhani and Ali El Kaafarani. Direct anonymous attestation from lattices. Cryptology ePrint Archive, Report 2017/1022, 2017. http://eprint.iacr.org/2017/1022.
2. Carsten Baum, Ivan Damgård, Sabine Oechsner, and Chris Peikert. Efficient commitments and zero-knowledge protocols from ring-sis with applications to lattice-based threshold cryptosystems. *IACR Cryptology ePrint Archive*, 2016:997, 2016.

3. Xavier Boyen. Lattice mixing and vanishing trapdoors: A framework for fully secure short signatures and more. In *International Workshop on Public Key Cryptography*, pages 499–517. Springer, 2010.
4. Ernie Brickell, Jan Camenisch, and Liqun Chen. Direct anonymous attestation. In *Proceedings of the 11th ACM conference on Computer and communications security*, pages 132–145. ACM, 2004.
5. Ernie Brickell, Liqun Chen, and Jiangtao Li. A new direct anonymous attestation scheme from bilinear maps. In *International Conference on Trusted Computing*, pages 166–178. Springer, 2008.
6. Ernie Brickell, Liqun Chen, and Jiangtao Li. Simplified security notions of direct anonymous attestation and a concrete scheme from pairings. *International journal of information security*, 8(5):315–330, 2009.
7. Ernie Brickell and Jiangtao Li. A pairing-based DAA scheme further reducing TPM resources. In *Proceedings of 3rd International Conference on Trust and Trustworthy Computing*, volume 6101 of *LNCS*, pages 181–195. Springer, 2010.
8. Léo Ducas, Tancrede Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. Crystals–dilithium: Digital signatures from module lattices. Technical report.
9. Jan Camenisch, Manu Drijvers, and Anja Lehmann. Universally composable direct anonymous attestation. In *Public-Key Cryptography – PKC 2016*, volume 9615 of *LNCS*, pages 234–264. Springer, 2016.
10. Jan Camenisch, Liqun Chen, Manu Drijvers, Anja Lehmann, David Novick and Rainer Urian. One TPM to Bind Them All: Fixing TPM 2.0 for Provably Secure Anonymous Attestation. In *IEEE Security & Privacy – S&P 2017*, IEEE, 2017.
11. David Cash, Dennis Hofheinz, Eike Kiltz, and Chris Peikert. Bonsai trees, or how to delegate a lattice basis. J. Cryptology, 25(4):601-639, 2012.
12. Liqun Chen and Jiangtao Li. Flexible and scalable digital signatures in tpm 2.0. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 37–48. ACM, 2013.
13. Liqun Chen, Dan Page, and Nigel P. Smart. On the design and implementation of an efficient DAA scheme. In *Proceedings of the 9th Smart Card Research and Advanced Application IFIP Conference (CARDIS), LNCS 6035*, pages 223–237. Springer, 2010.
14. Jeffrey Hoffstein, Jill Catherine Pipher, and Joseph H. Silverman. *An introduction to mathematical cryptography.* 2014.
15. Benoît Libert, San Ling, Fabrice Mouhartem, Khoa Nguyen, and Huaxiong Wang. Signature schemes with efficient protocols and dynamic group signatures from lattice assumptions. In *Advances in Cryptology–ASIACRYPT 2016: 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part II 22*, pages 373–403. Springer, 2016.
16. Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 700–718. Springer, 2012.
17. San Ling, Khoa Nguyen, and Huaxiong Wang. Group signatures from lattices: simpler, tighter, shorter, ring-based. In *IACR International Workshop on Public Key Cryptography*, pages 427–449. Springer, 2015.
18. San Ling, Khoa Nguyen, Damien Stehlé, and Huaxiong Wang. Improved zero-knowledge proofs of knowledge for the isis problem, and applications. In *Public-Key Cryptography–PKC 2013*, pages 107–124. Springer, 2013.

19. Chris Peikert et al. A decade of lattice cryptography. *Foundations and Trends® in Theoretical Computer Science*, 10(4):283–424, 2016.
20. Chris Peikert, Oded Regev, and Noah Stephens-Davidowitz. Pseudorandomness of ring-lwe for any ring and modulus. 2017.
21. Oded Regev. The learning with errors problem (invited survey). In *2010 IEEE 25th Annual Conference on Computational Complexity.*
22. Peter W Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM review*, 41(2):303–332, 1999.
23. Vadim Lyubashevsky. Lattice signatures without trapdoors. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 738–755. Springer, 2012.
24. The TCG. https://trustedcomputinggroup.org.

A Security Proof of the Modified Boyen Signture Scheme

In this section we examine a signature scheme (described in Subsection 3.1) based on the one from [3]. We claim that the same security result applies to this scheme as in the one in [3] 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 $\boldsymbol{A}_0 \in \mathbb{Z}_q^{n\times m}$ and a uniform vector $\boldsymbol{u} \in \mathbb{Z}_q^n$, and must find $\boldsymbol{e}_0 \in \mathbb{Z}^m$ with $\|\boldsymbol{e}_0\|_\infty \leq \beta$ and $\boldsymbol{A}_0\boldsymbol{e}_0 = \boldsymbol{u} \mod q$.

Setup:

1. Pick uniformly random $\boldsymbol{B}_0 \in \mathbb{Z}_q^{n\times m}$ with associated short trapdoor matrix $\boldsymbol{T}_{\boldsymbol{B}_0} \subset \wedge^{\perp}(\boldsymbol{B}_0)$.
2. Pick $l + 2$ short matrices $\boldsymbol{R}_t, \boldsymbol{R}_0, ..., \boldsymbol{R}_l \in \mathbb{Z}_q^{m\times m}$.
   -Do so by sampling the columns from $\mathcal{D}_{\mathbb{Z}^m,\eta}$.
3. Define $\boldsymbol{A}_t := \boldsymbol{A}_0\boldsymbol{R}_t$. Pick a random vector $\boldsymbol{d}_t \in \mathbb{Z}^m$ and compute $\boldsymbol{A_t}\boldsymbol{d}_t =: \boldsymbol{u}_t \mod q$.
4. Pick $l + 1$ random scalars $h_0, ..., h_l \in \mathbb{Z}_q$ and set $h_0 = 1$.
5. Output the verification key

$$VK = [\boldsymbol{A}_t, \boldsymbol{A}_0, \boldsymbol{C}_0 = (\boldsymbol{A}_0\boldsymbol{R}_0 + h_0\boldsymbol{B}_0), ..., (\boldsymbol{A}_0\boldsymbol{R}_l + h_l\boldsymbol{B}_0)].$$

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

1. Compute the matrix $\boldsymbol{R}_{\mathrm{msg}} = \sum_{i=0}^{l}(-1)^{\mathrm{msg}[i]}\boldsymbol{R}_i$.
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

$$\begin{aligned}\boldsymbol{F} &= [\boldsymbol{A}_0|\sum_{i=0}^{l}(-1)^{\mathrm{msg}[i]}\boldsymbol{C}_i] \\ &= [\boldsymbol{A}_0|\boldsymbol{A}_0\boldsymbol{R}_{\mathrm{msg}} + h_{\mathrm{msg}}\boldsymbol{B}_0],\end{aligned}$$

   sample $\boldsymbol{d}_h \in \mathbb{Z}^{2m}$ such that $\boldsymbol{F} \cdot \boldsymbol{d}_h = \boldsymbol{u}_h := (\boldsymbol{u} - \boldsymbol{u}_t) \mod q$ and $\|\boldsymbol{d}_h\|_\infty \leq \beta$. Write $\boldsymbol{d}_h = [\boldsymbol{d}_{h_0}, \boldsymbol{d}_{h_1}]$, where $\boldsymbol{d}_{h_0}, \boldsymbol{d}_{h_1} \in \mathbb{Z}^m$.
   -Do so by taking the trapdoor $\boldsymbol{T}_{\boldsymbol{B}_0}$ and delegating this to one for the matrix $\boldsymbol{F}$ via standards methods [11].

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

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

1. Compute the matrix $\boldsymbol{R}_{\text{msg}^*} = \sum_{i=0}^{l}(-1)^{\text{msg}^*[i]}\boldsymbol{R}_i$.
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

$$\begin{aligned} \boldsymbol{u} &= [\boldsymbol{A}_t | \boldsymbol{A}_0 | \boldsymbol{A}_0 \boldsymbol{R}_{\text{msg}^*} + h_{\text{msg}^*}\boldsymbol{B}_0] \cdot \boldsymbol{d} \mod q, \\ &= [\boldsymbol{A}_0\boldsymbol{R}_t | \boldsymbol{A}_0 | \boldsymbol{A}_0 \boldsymbol{R}_{\text{msg}^*}] \cdot \begin{bmatrix} \boldsymbol{d}_t^T \\ \boldsymbol{d}_{h_0}^T \\ \boldsymbol{d}_{h_1}^T \end{bmatrix} \mod q. \end{aligned}$$

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

All that remains to show is that

– $\boldsymbol{e}_0$ 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 [3]. A slight modification needs to be made to the parameter β. In particular, we have that with overwhelming probability $\|\boldsymbol{e}_0\|_\infty \leq \beta$ 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 [3].

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

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

and

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

Multiply equation 1 by $\mathbf{p}'$, and equation 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 [2]). 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. □

C Ideal Functionalities From [9]

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 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 S;
2. On input (REPLACE1,sid, ssid, μ_2', F) from 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 S.
4. On input (REPLACE2, sid, ssid, μ_2''') from S, output (SENT, sid, ssid, μ_1, μ_2''') to R

Fig. 1: The special authenticated communicatioin functionality F_{auth^*}

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'.

Fig. 2: Ideal certification authority functionality F_{ca}

C.2 Certification Authority

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.

Fig. 3: Ideal secure message transmission functionality F^l_{smt}

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.

Fig. 4: Ideal crs functionality $F_{crs}^{\mathcal{D}}$

D Detailed Security Proof of the L-DAA Scheme

- **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.

Fig. 5: Game 3 for F

- **Key Gen**

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

- **JOIN**
  1. Upon receiving (FORWARD, (JOIN, sid, jsid, tpm_i), host_j) from F, give input (JOIN, sid, jsid, tpm_i) to the host "host_j"
  2. Upon receiving intput (FORWARD, (JOINPROCEED, sid, jsid), I) from F, give "I" input (JOINPROCEED, sid, jsid).

- **SIGN**
  1. Upon recieving input (FORWARD, (SIGN, sid, ssid, tpm_i, bsn), host_j) from F,give "host_j" input (SIGN, sid, ssid, tpm_i, bsn).
  2. Upon receiving input (FORWARD, (SIGNPROCEED, sid, ssid), tpm_i) from F, give "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 recieving 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.

Fig. 6: 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.

Fig. 7: 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($gsk,\ \mu,\ \mathsf{bsn}$) as follows:
    - ∗ Define SamplePre($\hat{A}_{id},\ \hat{T}_I,\ q,\ \mathbf{u}_h,\ s$) that outputs a Boyen signature $\hat{X}_h$ [3], 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 $(\mathsf{nym},\ \mathsf{bsn}, \pi)$, and checks SPK on π with respect to bsn, nym, μ and $\mathbf{u}$. It output 1 if the proof is valid and 0 otherwise.
  - Define link($\sigma,\ \mu, \mathsf{bsn},\ \sigma',\ \mu'$): Check whether two signatures $(\sigma,\ \mu)$ and $(\sigma',\ \mu')$ 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($\sigma,\ \mu,\ \mathsf{bsn}, gsk$) as follows:It parses σ as $(\mathsf{nym},\ \mathsf{bsn}, \pi)$ 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 as follows: 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 $(\hat{A}_I,\ \pi_I)$ 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**

  Unchanged.
- **SIGN**

  Unchanged.
- **VERIFY**

  Unchanged.
- **LINK**

  Unchanged.
- **OUTPUT**

  When any simulated party "P" outputs a message μ that is not handled by $\mathcal{S}$, $\mathcal{S}$ sends (OUTPUT, P, μ) to F.

Fig. 8: Game 4 for S

- **SETUP**
  1. On input (SETUP, sid) from I, verify that 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, out put (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.

Fig. 9: Game 5 for F

- **Key Gen**

  Unchanged.
- **JOIN**

  Unchanged.
- **SIGN**

  Unchanged.
- **VERIFY**

  Nothing to simulate.
- **LINK**

  Nothing to simulate.

Fig. 10: 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 $\langle \mathsf{jsid}$, $\mathsf{tpm_i}$, $\mathsf{host_j}$, request $\rangle$.
   - Output (JOINSTART, sid, jsid, $\mathsf{tpm_i}$, $\mathsf{host_j}$) to $\mathcal{S}$.
2. JOIN REQUEST DELIVERY: Proceed upon recieving 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 $\langle \mathsf{jsid}$, $\mathsf{tpm_i}$, $\mathsf{host_j}$, 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, out put (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.

Fig. 11: 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**
  Unchanged.
- **VERIFY**
  Nothing to simulate.
- **LINK**
  Nothing to simulate.

Fig. 12: Game 6 for $\mathcal{S}$

– **SETUP**
  Unchanged.

– **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 $gsk \leftarrow \text{Kgen}(1^\lambda)$.
    - ∗ 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, out put (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.

Fig. 13: Game 7 for F

– **KeyGen**
  Unchanged.

– **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**

  Nothing to simulate.

– **LINK**

  Nothing to simulate.

Fig. 14: Game 7 for $\mathcal{S}$

- **SETUP**
  Unchanged.
- **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 $\langle \mathsf{ssid}, \mathsf{tpm_i}, \mathsf{host_j}, \mu, \mathsf{bsn}$, 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$(\sigma, \mu, \mathsf{bsn}, gsk') = 1$.
  - If $f \neq 0$, set f=ver$(\sigma, \mu, \mathsf{bsn})$.
  - Add $(\sigma, \mu, \mathsf{bsn}, RL, f)$ to VerResults, out put (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.

Fig. 15: Game 8 for F

- **KeyGen**
  Unchanged.
- **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**

  Nothing to simulate.
- **LINK**

  Nothing to simulate.

Fig. 16: Game 8 for $\mathcal{S}$

- **SETUP**
  Unchanged.
- **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 $gsk \leftarrow Kgen(1^\lambda)$.
    - $*$ 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, out put (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.

Fig. 17: Game 9 for F

- **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 ⟨jsid, $\mathsf{tpm_i}$, $\mathsf{host_j}$, request ⟩.
     - Output (JOINSTART, sid, jsid, $\mathsf{tpm_i}$, $\mathsf{host_j}$) to $\mathcal{S}$.
  2. JOIN REQUEST DELIVERY: Proceed upon recieving delivery notification from $\mathcal{S}$.
     - Update the session record to ⟨jsid, $\mathsf{tpm_i}$, $\mathsf{host_j}$, delivered⟩.
     - If I or $\mathsf{tpm_i}$ is honest and ⟨$\mathsf{tpm_i}$, $\star$, $\star$⟩ 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 ⟨jsid, sid, $\mathsf{tpm_i}$, $\mathsf{host_j}$, complete⟩.
     - Output (JOINCOMPLETE, sid, jsid) to $\mathcal{S}$.
  4. KEY GENERATION: On input (JOINCOMPLETE, sid, jsid, gsk) from $\mathcal{S}$.
     - Update the session record to ⟨jsid, $\mathsf{tpm_i}$, $\mathsf{host_j}$, complete⟩
     - 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 ⟨$\mathsf{tpm_i}$, $\mathsf{host_j}$, gsk⟩ 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 ⟨$\mathsf{tpm_i}$, $\mathsf{host_j}$, $\star$⟩ exists in Members.
    * Else, create a sign session ⟨ssid, $\mathsf{tpm_i}$, $\mathsf{host_j}$, μ, bsn, request⟩.
    * 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 ⟨ssid, $\mathsf{tpm_i}$, $\mathsf{host_j}$, μ, bsn, delivered⟩.
  - 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 ⟨ssid, $\mathsf{tpm_i}$, $\mathsf{host_j}$, μ, bsn, delivered⟩.
    * 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 bsn $\neq \perp$, then retrieve gsk from the ⟨$\mathsf{tpm_i}$, bsn, gsk⟩ $\in$ DomainKeys.
    * If bsn $= \perp$ or no gsk was found, generate a fresh key $gsk \leftarrow Kgen(1^\lambda)$.
    * Check CheckGskHonest(gsk)=1
    * Store ⟨$\mathsf{tpm_i}$, bsn, gsk⟩ in DomainKeys.
    * Generate the signature $\sigma \leftarrow sig(gsk, \mu, \mathsf{bsn})$.
    * If $\mathsf{tpm_i}$ is honest, then store ⟨σ, μ, $\mathsf{tpm_i}$, bsn⟩ in Signed and output (SIGNATURE, sid, ssid, σ) to $\mathsf{host_j}$.
- **VERIFY**

  Unchanged
- **LINK**

  Unchanged

Fig. 18: Game 10 for F

– **SETUP**
  Unchanged.

– **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
  - 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, out put (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.

Fig. 19: Game 11 for F

- **SETUP**
  Unchanged.
- **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 bsn $\neq \perp$, then retrieve gsk from the $\langle \mathsf{tpm_i}, \mathsf{bsn}, gsk\rangle \in$ DomainKeys.
    - ∗ If bsn $= \perp$ or no gsk was found, generate a fresh key $gsk \leftarrow Kgen(1^{\lambda})$.
    - ∗ Check CheckGskHonest(gsk)=1.
    - ∗ 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, out put (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.

Fig. 20: Game 12 for F

– **SETUP**
Unchanged.

– **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.
    - 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, out put (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.

Fig. 21: Game 13 for F

- **SETUP**
  Unchanged.
- **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 bsn $\neq \perp$, then retrieve gsk from the $\langle \mathsf{tpm_i}$, bsn, $gsk\rangle \in$ DomainKeys.
    - ∗ If bsn $= \perp$ or no gsk was found, generate a fresh key $gsk \leftarrow Kgen(1^\lambda)$.
    - ∗ Check CheckGskHonest(gsk)=1.
    - ∗ Store $\langle \mathsf{tpm_i}$, 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 $\langle\sigma$, μ, $\mathsf{tpm_i}$, 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 $\langle\star$, μ, $\mathsf{tpm_i}$, 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, out put (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.

Fig. 22: Game 14 for F

- **SETUP**
  Unchanged.
- **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, out put (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.

Fig. 23: Game 15 for F

- **SETUP**
  Unchanged.
- **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, out put (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.

Fig. 24: Game 16 for F