A Lattice-based Enhanced Privacy ID

Nada EL Kassem[1], Luís Fiolhais[2], Paulo Martins[3], Liqun Chen[1], and Leonel Sousa[4]

[1] University of Surrey, UK
n.elkassem@surrey.ac.uk, liqun.chen@surrey.ac.uk
[2] INESC-ID, Portugal
luis.azenhas.fiolhais@tecnico.ulisboa.pt
[3] Universidade Técnica de Lisboa, Portugal
paulo.sergio@ist.utl.pt
[4] Universidade de Lisboa, Portugal
las@inesc-id.pt

Abstract. The Enhanced Privacy ID (EPID) scheme is currently used for hardware enclave attestation by an increasingly large number of platforms that implement Intel Software Guard Extensions (SGX). However, the scheme currently deployed by Intel is supported on Elliptic Curve Cryptography (ECC), and will become insecure should a large quantum computer become available. As part of National Institute of Standards and Technology (NIST)'s effort for the standardisation of post-quantum cryptography, there has been a great boost in research on lattice-based cryptography. As this type of cryptography is more widely used, one expects that hardware platforms start integrating specific instructions that accelerate its execution. In this article, a new EPID scheme is proposed, supported on lattice primitives, that may benefit not only from future research developments in post-quantum cryptography, but also from instructions that may extend Intel's Instruction Set Architecture (ISA) in the future. This paper presents a new security model for EPID in the Universal Composability (UC) framework. The proposed Lattice-based EPID (LEPID) scheme is proved secure under the new model. Experimentally compared with a closely related Lattice-based Direct Anonymous Attestation (DAA) (LDAA) scheme from related art, it is shown that the private-key size is reduced 1.5 times, and that signature and verification times are sped up up to 1.4 and 1.1 times, respectively, for the considered parameters, when LEPID is compared with LDAA. Moreover, the signature size compares favourably to LDAA for small and medium-sized communities.

1 Introduction

The Enhanced Privacy ID (EPID) scheme is a fundamental part of the security model underpinning Software Guard Extensions (SGX)'s functioning [1]. It gives the ability to attest that a hardware enclave was successfully established on an Intel platform.

EPID can be seen as Direct Anonymous Attestation (DAA) with different linkability requirements [2]. The DAA scheme was built having the Trusted Platform Module (TPM) standard in mind. In this context, the TPM holds a representation of the host machine state, and wishes to provide a verifier with a signature of the state representation, without revealing their identity. During an offline phase, an issuer provisions the TPM and the host with membership credentials. Based on this cryptographic material, the TPM and the host jointly prove that they belong to the DAA community in zero-knowledge, while producing the above-mentioned signature. Unlike other privacy-preserving systems, like group signatures, DAA does not support the property of traceability, wherein a group manager can identify the signer from a given signature.

Alternatively, the DAA provides two approaches to prevent a malicious signer from abusing their anonymity. Firstly, when a private-key is leaked, anyone can check whether a specific DAA signature was created under this key or not. Secondly, two DAA signatures created by the same signer may or may not be linked from a verifier's point of view. The linkability is controlled by a parameter called basename. When the same basename is used by the same signer for two signatures, they are linked; otherwise they are not. However, there are situations where this model does not suffice to prevent malicious actions. For instance, should an attacker corrupt a TPM and obtain the private-key without ever publishing it, there is no way to revoke it. While this latter problem can be mitigated by having TPMs use the same basename whenever they access a certain service, this option removes the anonymity for all uses with the same basename.

EPID is a more general scheme than DAA and thus does not split signers into TPMs and hosts, but also targets the creation of anonymous signatures. An EPID scheme consists of an issuer, signers, verifiers and a revocation manager. Like with DAA, one can check whether a certain signature was generated by a leaked private-key. Nonetheless, the ability to link signatures with the same basename is removed. Instead, whenever a signer is corrupted, they may be revoked by including one of their signatures as part of a revocation list. As a result, EPID is capable of revoking corrupted signers from the system, even when their private-key is kept hidden, whilst providing maximum privacy for the platforms. Enhanced Privacy ID signatures can also be constructed on the top of group signatures that allow members of a group to anonymously sign messages on behalf of the group, with the added property that a group manager can revoke the credentials of a misbehaving or compromised group member. This construction was adopted by a recently proposed post-quantum EPID scheme from symmetric primitives [3], an overview of the scheme will be given in Section 7.

Lattices have proven to be a flexible tool in constructing cryptographic schemes, with applications ranging from digital signatures to public-key encryption and zero-knowledge proofs, while offering post-quantum security [4,5,6]. One expects that as this type of cryptography matures, an increasing number of platforms exploiting EPID ship with accelerators for lattice-based constructs [7]. Herein, by building from a recently proposed DAA scheme [8], the range of cryptographic constructs supported by lattice-based cryptography is extended to EPID. The LEPID signature size compares favourably to Lattice-based DAA (LDAA) for small and medium-sized communities.

Organisation: The next section introduces the lattice-based hard problems and the two building blocks that support the proposed LEPID scheme, namely the LDAA scheme from [8] and the Zero Knowledge Proof of Knowledge (ZKPoK) of Ring-Learning With Errors (Ring-LWE) secrets from [6]. Section 3 presents a new security model for EPID in the UC framework. The novel LEPID scheme is proposed in Section 4 and proven secure in Section 5. The performance of the LEPID scheme is discussed in Section 6. Section 7 briefly discusses the related work. Finally, Section 8 concludes the paper.

2 Preliminaries

Throughout this paper we will use the polynomial rings $\mathcal{R}_q = \mathbb{Z}_q[X]/\langle X^n + 1\rangle$, where $\mathbb{Z}_q$ is the quotient ring $\mathbb{Z}/q\mathbb{Z}$ and n a power of 2. We use names in bold, like $\boldsymbol{a}$, both to denote elements of $\mathcal{R}_q$ and their coefficient embeddings in $\mathbb{Z}_q^n$. $\|\boldsymbol{a}\|_\infty$ represents the infinity norm of a polynomial $\boldsymbol{a}$, $\|\boldsymbol{a}\|_\infty = \max_i |a^i|$, and $\|\boldsymbol{a}\| = \sqrt{\sum_{i=1}^{n}(a^i)^2}$ where the a^i are the coefficients of $\boldsymbol{a}$. $\hat{A} = (\boldsymbol{a}_1, \ldots, \boldsymbol{a}_m)$ denotes a vector where m is a positive integer and $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_m$ are polynomials. $\|\hat{A}\|_\infty$ denotes the infinity norm of $\hat{A}$, defined as $\|\hat{A}\|_\infty = \max_i \|\mathbf{a}_i\|_\infty$. B_{3d} represents the set of vectors $\boldsymbol{u} \in \{-1, 0, 1\}^{3d}$ having exactly d coordinates equal to -1, d coordinates equal to 0, and d coordinates equal to 1. We represent a challenge set by $\mathcal{C} = \left\{X^{c_v}, |c_v \in \{0, 1, \ldots 2n-1\}\right\}$, where $\bar{\mathcal{C}}$ denotes the set of differences $\mathcal{C} - \mathcal{C}$ except 0. $\mathcal{D}_s^h$ represents the discrete Gaussian distribution of standard deviation s, s.t. $\Pr_{\boldsymbol{x}\leftarrow\mathcal{D}_s^h}[\|\boldsymbol{x}\| > \sqrt{2h}s] \le 2^{-h/4}$. We define the following rejection sampling algorithm from [9] to avoid the dependency of $\boldsymbol{z}$ on the secret $\boldsymbol{b}$, $\texttt{rej}(\boldsymbol{z}, \boldsymbol{b}, \xi)$: Let $u \leftarrow [0, 1)$; if $u > 1/3 \exp\left(\frac{-2\langle \boldsymbol{z}, \boldsymbol{b}\rangle + \|\boldsymbol{b}\|^2}{2\xi^2}\right)$ return 0, else return 1, with ξ representing a standard deviation of some distribution.

Definition 1 (The Ring Short Integer Solution Problem (Ring-SIS$_{n,m,q,\beta}$) [10]). *Given m uniformly random elements $\boldsymbol{a}_i \in \mathcal{R}_q$ defining a vector $\hat{A} = (\boldsymbol{a}_1, \boldsymbol{a}_2, \ldots, \boldsymbol{a}_m)$, find a nonzero vector of polynomials $\hat{Z} \in \mathcal{R}_q^m$ of norm $\|\hat{Z}\|_\infty \le \beta$ such that: $f_{\hat{A}}(\hat{Z}) = \sum_{i\in[m]} \boldsymbol{a}_i \boldsymbol{z}_i = \boldsymbol{0} \in \mathcal{R}_q$. The Ring Inhomogeneous Short Integer Solution (Ring-ISIS$_{n,m,q,\beta}$) problem asks to find $\hat{Z}$ of norm $\|\hat{Z}\|_\infty \le \beta$, and such that:$f_{\hat{A}}(\hat{Z}) = \boldsymbol{y} \in \mathcal{R}_q$ for some uniform random polynomial $\boldsymbol{y}$.*

Definition 2 (The Ring Learning With Error Problem (Ring-LWE) [11]). *Let χ be an error distribution defined over $\mathcal{R}$ and $\boldsymbol{s} \leftarrow \mathcal{R}_q$ a uniformly random ring element, the Ring-LWE distribution $A_{s,\chi}$ over $\mathcal{R}_q \times \mathcal{R}_q$ is sampled by choosing $\boldsymbol{a} \in \mathcal{R}_q$ uniformly at random, randomly choosing the noise $\boldsymbol{e} \leftarrow \chi$ and outputting $(\boldsymbol{a}, \boldsymbol{b}) = (\boldsymbol{a},\ \boldsymbol{s}a + \boldsymbol{e} \mod q) \in \mathcal{R}_q \times \mathcal{R}_q$. Let $\boldsymbol{u}$ be uniformly sampled from $\mathcal{R}_q$. The decision problem of Ring-LWE asks to distinguish between $(\boldsymbol{a}, \boldsymbol{b}) \leftarrow A_{s,\chi}$ and $(\boldsymbol{a}, \boldsymbol{u})$ for a uniformly sampled secret $\boldsymbol{s} \leftarrow \mathcal{R}_q$. The search*

Ring-LWE problem asks to return the secret vector $\boldsymbol{s} \in \mathcal{R}_q$ given a Ring-LWE sample $(\boldsymbol{a}, \boldsymbol{b}) \leftarrow A_{s,\chi}$.

2.1 Lattice-based Direct Anonymous Attestation

The DAA scheme proposed in [8] can be split at a high level into three parts. In a first part, a TPM-host pair with identifier $\mathsf{id} = (\mathsf{id}_1, ..., \mathsf{id}_\ell) \in \{0,1\}^\ell$ joins a DAA community. This consists of the TPM sampling small $\hat{X}_t = (\boldsymbol{x}_1, \dots, \boldsymbol{x}_m) \in \mathcal{R}_q^m$, and sending $\boldsymbol{u}_t = \hat{A}_t \hat{X}_t$ to the issuer, where $\hat{A}_t \in \mathcal{R}_q^m$ is part of the issuer's public-key. A signature proof of knowledge based on [12], showing that $\boldsymbol{u}$ is well formed is also sent, along with a link token that prevents two TPMs from having the same secret-key. Using its private-key, the issuer then samples small $\hat{X}_h = (x_{m+1}, \dots, x_{3m}) \in \mathcal{R}_q^{2m}$ such that $\hat{A}_h \hat{X}_h = \boldsymbol{u} - \boldsymbol{u}_t$, where $\hat{A}_h = [\hat{A}_\mathcal{I} | \hat{A}_0 + \sum_{i=1}^{l} \mathrm{id}_i \hat{A}_i] \in \mathcal{R}_q^{2m}$, and $\boldsymbol{u} \in \mathcal{R}_q$, $\hat{A}_\mathcal{I} \in \mathcal{R}_q^m$ and $\hat{A}_i \in \mathcal{R}_q^m$ $\forall i \in \{0, \dots, l\}$ are part of the issuer's public-key. The vector $\hat{X}_h$ is sent back to the host. After this process, the TPM and host own small key-shares satisfying

$$[X_t | X_h][\hat{A}_t | \hat{A}_h] = \boldsymbol{u}, \tag{1}$$

In a second part, the TPM and the host jointly generate a signature with respect to a message μ. The signature corresponds to a tuple (nym, bsn, π), where nym is a link token, bsn is the basename, and π is a signature-based proof:

$$\pi = \mathsf{SPK}\Big\{\mathsf{public} := \{\mathsf{pp}, \mathsf{nym}, bsn\}, \mathsf{witness} := \{\hat{X} = (\boldsymbol{x}_1, \dots, \boldsymbol{x}_{3m}), \mathrm{id},\ \boldsymbol{e}\} :$$
$$\boldsymbol{u} = \hat{X}[\hat{A}_t | \hat{A}_h] \bmod q \wedge \|\hat{X}\|_\infty \le \beta \wedge \mathsf{nym} = \mathcal{H}(bsn)\boldsymbol{x}_1 + \boldsymbol{e} \bmod q \wedge \|\boldsymbol{e}\|_\infty \le \beta \Big\}(\mu)$$

demonstrating not only (1) but also that $\mathsf{nym} = \mathcal{H}(bsn)\boldsymbol{x}_1 + \boldsymbol{e} \mod q$, where $\mathcal{H}$ is a random oracle mapping bsn to a polynomial and $\boldsymbol{e}$ is small.

A final part deals with signature verification. First, π is verified. Then, the verifier iterates over the list of revoked private-keys, consisting of the elements $\boldsymbol{x}_1^{(i)}$ of the $\hat{X}_t^{(i)}$ in (1) of the corrupt signers. In the case that $\|\mathsf{nym} - \mathcal{H}(bsn)\boldsymbol{x}_1^{(i)}\|_\infty$ is small, the signature has been generated by the i-th revoked user and is rejected. Similarly, two signatures (nym, bsn, π) and $(\mathsf{nym}', bsn, \pi')$ having the same basename are linked when $\|\mathsf{nym} - \mathsf{nym}'\|_\infty$ is small.

2.2 Zero Knowledge Proof of the Ring-LWE Secrets

The technique presented in [6] will herein be used to modify the LDAA and support the more effective revocation method of EPID. This techniques allows one to efficiently prove in zero-knowledge possession of $\boldsymbol{s}$ and $\boldsymbol{e}$, with $2\boldsymbol{s}$ and $2\boldsymbol{e}$ being short, such that $2\boldsymbol{y} = 2\boldsymbol{a}\boldsymbol{s} + 2\boldsymbol{e}$, for public $\boldsymbol{a}$ and $\boldsymbol{y}$. Random $\boldsymbol{r}_s, \boldsymbol{r}_e \leftarrow \mathcal{D}_s$ are initially produced, and $\boldsymbol{t} = \boldsymbol{a}\boldsymbol{r}_s + \boldsymbol{r}_e$ is computed. A challenge $c = H(\boldsymbol{t}) \in \{0, 1, \dots, 2n-1\}$ is generated and $\boldsymbol{s}_s = \boldsymbol{r}_s + X^c\boldsymbol{s}$, $\boldsymbol{s}_e = \boldsymbol{r}_e + X^c\boldsymbol{e}$ are outputted in response with probability $\mathrm{P}(\boldsymbol{s}_s, \boldsymbol{s}_e)$, where P is chosen in a way that prevents $\boldsymbol{s}_s$ and $\boldsymbol{s}_e$ from depending on the prover's secret inputs.

3 UC based Security Model for EPID

The security model for the DAA given by Camenisch et al. in [13] has been modified by replacing linkability with a revocation interface, adding the signature revocation check from [14], and introducing other modifications that results in a new EPID security model in the Universal Composability (UC) framework. Our new security definition is given in the UC model with respect to an ideal functionality $\mathcal{F}^{l}_{\mathsf{EPID}}$. In UC, an environment $\mathcal{E}$ should not be able to distinguish with a non-negligible probability between two worlds: the real world, where each party in the EPID protocol Π executes its assigned part of the protocol and the network is controlled by an adversary $\mathcal{A}$ that communicates with $\mathcal{E}$; and the ideal world, in which all parties forward their inputs to $\mathcal{F}^{l}_{\mathsf{EPID}}$, which internally performs all the required tasks and creates the party's outputs. A protocol Π is said to securely realise $\mathcal{F}^{l}_{\mathsf{EPID}}$ 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.

An EPID scheme should satisfy: *i*) unforgeability, i.e. for honest issuer and signers, no adversary can output a valid signature on a message μ without knowing the signer's secret key; *ii*) correctness, i.e. honestly generated signatures are always valid; and *iii*) anonymity, i.e. even for a corrupt issuer, no adversary can tell whether two honestly generated signatures were produced by the same signer. The UC framework allows us to focus on the analysis of a single protocol instance with a globally unique session identifier *sid*. $\mathcal{F}^{l}_{\mathsf{EPID}}$ uses session identifiers of the form $sid = (\mathcal{I}, sid')$ for some issuer $\mathcal{I}$ and a unique string sid'. In the procedures, functions $\mathsf{CheckTtdHonest}$ and $\mathsf{CheckTtdCorrupt}$ are used that return '1' when a key belongs to a honest signer that has produced no signature, and when a key belongs to a corrupt user such that there is no signature simultaneously linking back to the inputted key and another one, respectively; and return '0' otherwise. We label the checks that are done by the ideal functionality in roman numerals.

$\mathcal{F}^{l}_{\mathsf{EPID}}$ **Setup**: On input (SETUP, sid) from the issuer $\mathcal{I}$, $\mathcal{F}^{l}_{\mathsf{EPID}}$ verifies that $(\mathcal{I}, sid') = sid$ and outputs (SETUP, sid) to $\mathcal{S}$. $\mathcal{F}^{l}_{\mathsf{EPID}}$ receives from the simulator $\mathcal{S}$ the algorithms Kgen, sig, ver, identify and revoke. These algorithms are responsible for generating keys for honest signers, creating signatures for honest signers, verifying the validity of signatures, checking whether a signature was generated by a given key, and updating the revocation lists respectively. $\mathcal{F}^{l}_{\mathsf{EPID}}$ stores the algorithms, checks that the algorithms ver, identify and revoke are deterministic [Check-I], and outputs $(\text{SETUPDONE}, sid)$ to $\mathcal{I}$.
$\mathcal{F}^{l}_{\mathsf{EPID}}$ **Join**:

1. JOIN REQUEST: On input $(\text{JOIN}, sid, jsid)$ from a signer $\mathcal{M}_i$, create a join session $\langle jsid, \mathcal{M}_i, \text{request}\rangle$. Output $(\text{JOINSTART}, sid, jsid, \mathcal{M}_i)$ to $\mathcal{S}$.
2. JOIN REQUEST DELIVERY: Proceed upon receiving delivery notification from $\mathcal{S}$ by updating the session record to $\langle jsid, \mathcal{M}_i, \text{delivery}\rangle$.
   - If $\mathcal{I}$ or $\mathcal{M}_i$ is honest and $\langle \mathcal{M}_i, \star, \star\rangle$ is already in Member List `ML`, output $\perp$ [Check II].

- Output (JOINPROCEED, $sid, jsid, \mathcal{M}_i$) to $\mathcal{I}$.

3. JOIN PROCEED: Upon receiving an approval from $\mathcal{I}$, $\mathcal{F}^l_{\mathsf{EPID}}$ updates the session record to $\langle jsid, sid, \mathcal{M}_i, \text{complete}\rangle$. Then it outputs (JOINCOMPLETE, $sid, jsid$) to $\mathcal{S}$.
4. KEY GENERATION: On input (JOINCOMPLETE, $sid, jsid, tsk$) from $\mathcal{S}$.
   - If the signer is honest, set $tsk = \perp$, else verify that the provided tsk is eligible by performing the following two checks that are described above:
     - $\mathsf{CheckTtdHonest}(tsk)$=1 [Check III].
     - $\mathsf{CheckTtdCorrupt}(tsk)$=1 [Check IV].
   - Insert $\langle \mathcal{M}_i, tsk\rangle$ into Member List ML, and output JOINED.

$\mathcal{F}^l_{\mathsf{EPID}}$ **Sign**:

1. SIGN REQUEST: On input (SIGN, $sid, ssid, \mathcal{M}_i, \mu, \boldsymbol{p}$) from the signer on a message μ with respect to $\boldsymbol{p}$, the ideal functionality aborts if $\mathcal{I}$ is honest and no entry $\langle \mathcal{M}_i, \star\rangle$ exists in ML, else creates a sign session $\langle ssid, \mathcal{M}_i, \mu, \boldsymbol{p}$, request$\rangle$ and outputs (SIGNSTART, $sid, ssid, \mathcal{M}_i, l(\mu, \boldsymbol{p})$) to $\mathcal{S}$.
2. SIGN REQUEST DELIVERY: On input (SIGNSTART, $sid, ssid$) from $\mathcal{S}$, update the session to $\langle ssid, \mathcal{M}_i, \mu, \boldsymbol{p}$, delivered$\rangle$, and output (SIGNPROCEED, sid, $ssid$, μ, $\boldsymbol{p}$) to $\mathcal{M}_i$.
3. SIGN PROCEED: On input (SIGNPROCEED, $sid, ssid$) from $\mathcal{M}_i$, $\mathcal{F}^l_{\mathsf{EPID}}$ updates the records $\langle ssid, \mathcal{M}_i, \mu, \boldsymbol{p}$, delivered$\rangle$, and outputs (SIGNCOMPLETE, $sid, ssid$, KRL, SRL) to $\mathcal{S}$, where KRL and SRL represent the key and the signature revocation lists respectively.
4. SIGNATURE GENERATION: On input (SIGNCOMPLETE, $sid, ssid, \sigma$, KRL, SRL) from $\mathcal{S}$, if $\mathcal{M}_i$ is honest then $\mathcal{F}^l_{\mathsf{EPID}}$ will:
   - Ignore an adversary's signature σ, and generate the signature for a fresh or established tsk.
   - Check $\mathsf{CheckTtdHonest}(tsk)$=1 [Check V], and store $\langle \mathcal{M}_i, tsk\rangle$ in DomainKeys.
   - Generate the signature $\sigma \leftarrow \text{sig}(tsk, \mu, \boldsymbol{p})$.
   - Check ver($\sigma, \mu, \boldsymbol{p}$, KRL, SRL)=1 [Check VI], and check identify($\sigma, \mu, \boldsymbol{p}$, tsk) $=$ 1 [Check VII].
   - Check that there is no signer other than $\mathcal{M}_i$ with key tsk' registered in Members or DomainKeys such that identify($\sigma, \mu, \boldsymbol{p}$, tsk')=1 [Check VIII].
   - For all $(\sigma^*, \mu^*, \boldsymbol{p}^*) \in$ SRL, find all $(tsk^*, \mathcal{M}^*)$ from Members and DomainKeys such that identify$(\sigma^*, \mu^*, \boldsymbol{p}^*, \star, tsk^*) = 1$
     - Check that no two distinct keys tsk^* trace back to σ^*.
     - Check that no pair $(tsk^*, \mathcal{M}_i)$ was found.
   - If $\mathcal{M}_i$ is honest, then store $\langle \sigma, \mu, \mathcal{M}_i, \boldsymbol{p}\rangle$ in Signed and output (SIGNATURE, sid, $ssid$, σ, KRL, SRL).

$\mathcal{F}^l_{\mathsf{EPID}}$ **Verify**: On input (VERIFY, $sid, \mu, \boldsymbol{p}, \sigma$, KRL, SRL), from a party $\mathcal{V}$ to check whether σ is a valid signature on a message μ with respect to $\boldsymbol{p}$, KRL and SRL, the ideal functionality does the following:

- Extract all pairs $(tsk_i, \mathcal{M}_i)$ from the DomainKeys and ML, for which identify(σ, $\mu, \boldsymbol{p}$, tsk_i)=1. Set $b = 0$ if any of the following holds:

  - More than one key tsk_i was found [Check IX].
  - $\mathcal{I}$ is honest and no pair $(tsk_i, \mathcal{M}_i)$ was found [Check X].
  - An honest $\mathcal{M}_i$ was found, but no entry $\langle \star, \mu, \mathcal{M}_i, \boldsymbol{p} \rangle$ was found in Signed [Check XI].
  - There is a key $tsk^* \in \texttt{KRL}$, such that identify$(\sigma, \mu, \boldsymbol{p}, tsk^*)$=1 and no pair $(tsk, \mathcal{M}_i)$ for an honest $\mathcal{M}_i$ was found [Check XII].
  - For some matching tsk_i and $(\sigma^*, \mu^*, \boldsymbol{p}^*) \in \texttt{SRL}$, identify$(\sigma^*, \mu^*, \boldsymbol{p}^*, tsk_i) = 1$
- If $b \neq 0$, set $b \leftarrow$ver$(\sigma, \mu, \boldsymbol{p}, \texttt{SRL}, \texttt{KRL})$. [Check XIII]
- Add $\langle \sigma, \mu, \boldsymbol{p}, \texttt{KRL}, \texttt{SRL}, b \rangle$ to VerResults, and output (VERIFIED, sid, b) to $\mathcal{V}$.

$\mathcal{F}^l_{\mathsf{EPID}}$ **Revoke**: On input $(tsk^*, \texttt{KRL})$, the ideal functionality replaces $\texttt{KRL}$ with $\texttt{KRL} \cup tsk^*$. On input $(\sigma^*, \mu^*, \texttt{SRL})$, the ideal functionality replaces $\texttt{SRL}$ with $\texttt{SRL} \cup \sigma^*$ after verifying σ^*.

4 The Proposed LEPID Scheme

The DAA scheme proposed in [8] is herein modified so as to support the security model described in Section 3. We give a general overview of the proposed Lattice-based EPID (LEPID) scheme in Subsection 4.1 before proceeding with the details in Subsection 4.2.

4.1 High Level Description of the LEPID Scheme

The first part of the DAA protocol described in Subsection 2.1 is herein mirrored, with the exception that the TPM and the host are fused into a single signer. In particular, the issuer makes one further polynomial $\boldsymbol{b}$ available in Procedure 1. When requesting to join a DAA community in Procedure 2, the signer with identifier $\mathsf{id} = (\mathsf{id}_1, ..., \mathsf{id}_\ell) \in \{0,1\}^\ell$ samples a small $\hat{X}_t = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{m+1}) \in \mathcal{R}_q^{m+1}$ and sends $\boldsymbol{u}_t = [b|\hat{A}_{\mathcal{I}}]\hat{X}_t \bmod q$ to the issuer, along with a link token $\mathsf{nym}_{\mathcal{I}} = \mathcal{H}(bsn_{\mathcal{I}})\boldsymbol{x}_1 + \boldsymbol{e}_{\mathcal{I}}$ and a zero-knowledge proof $\pi_{\boldsymbol{u}_t}$ from [8] showing that $\boldsymbol{u}_t$ is well formed. Upon receiving this message, the issuer uses $\mathrm{nym}_{\mathcal{I}}$ to check that no other signer has the same $\boldsymbol{x}_1$, verifies $\pi_{\boldsymbol{u}_t}$ and samples small $\hat{X}_h = [\hat{X}_{h_1}|\hat{X}_{h_2}] = (\boldsymbol{y}_2, \ldots, \boldsymbol{y}_{2m+1}) \in \mathcal{R}_q^m \times \mathcal{R}_q^m$ such that $\hat{A}_h\hat{X}_h = \boldsymbol{u} - \boldsymbol{u}_t \bmod q$, with $\hat{A}_h = [\hat{A}_{\mathcal{I}}|\hat{A}_0 + \sum_{i=1}^{l} \mathrm{id}_i\hat{A}_i] \in \mathcal{R}_q^{2m}$. $\hat{X}_h$ is sent back to the signer, that updates their key as $\hat{X} = (\boldsymbol{x}_1, \forall_{i=(2,\ldots,m+1)}\boldsymbol{x}_i := \boldsymbol{x}_i + \boldsymbol{y}_i, \forall_{i=(m+2,\ldots,2m+1)}\boldsymbol{x}_i := \boldsymbol{y}_i)$ in Procedure 3.

Signatures are generated in Procedures 4 and 5 as in Subsection 2.1 for the DAA, but the basename is always chosen at random, generating link tokens $\mathsf{nym} = \boldsymbol{p}\boldsymbol{x}_1 + \boldsymbol{e} \bmod q$ for a uniformly random $\boldsymbol{p}$, and the proof-of-knowledge π is as described in Appendix A.1. In particular, this allows one to maintain linkability in the case of leaked private-keys, whilst maintaining full anonymity. In addition, when signing a message, the signer is presented with a list of signatures from revoked users and proves in zero-knowledge that their underlying $\boldsymbol{x}_1$ was not used to produce any of those signatures. We achieve this by firstly randomising the $(\mathsf{nym}_i^* = \boldsymbol{p}_i^* \boldsymbol{f}_i + \boldsymbol{l}_i, \boldsymbol{p}_i^*)$ pairs from the list of revoked signatures, where $\boldsymbol{f}_i$ corresponds to the $\boldsymbol{x}_1$ polynomial of the i-th revoked user and $\boldsymbol{l}_i$ has small norm, as

$$\boldsymbol{d}_i = \mathsf{nym}_i^* \boldsymbol{q}_i + \boldsymbol{l}_i^{'''} \tag{2}$$

$$\boldsymbol{o}_i = \boldsymbol{p}_i^* \boldsymbol{q}_i + \boldsymbol{l}_i^{'} \tag{3}$$

for small $\boldsymbol{q}_i$, $\boldsymbol{l}_i^{'''}$ and $\boldsymbol{l}_i^{'}$ sampled from a Gaussian distribution. Note that $\boldsymbol{d}_i = \boldsymbol{o}_i \cdot \boldsymbol{f}_i + \boldsymbol{e}_i$ for a small $\boldsymbol{e}_i$. The signature includes not only $\boldsymbol{d}_i$ and $\boldsymbol{o}_i$, but also $\boldsymbol{k}_i = \boldsymbol{o}_i \boldsymbol{x}_1 + \boldsymbol{l}_i^{''}$ along with a zero-knowledge proof of the construction of $\boldsymbol{d}_i$, $\boldsymbol{o}_i$ and $\boldsymbol{k}_i$. This zero-knowledge proof is an adaptation of the one described in Subsection 2.2, the details of which can be found in Appendix A.2.

Signature verification in Procedure 6 is similar to that of the DAA, with the difference that now the proof of the shape of $\boldsymbol{d}_i$, $\boldsymbol{o}_i$ and $\boldsymbol{k}_i$ is verified, and the norm of $\boldsymbol{d}_i - \boldsymbol{k}_i$ is assessed to ascertain whether the $\boldsymbol{x}_1$ used to produce the signature under verification is the same as the one used to produce the i-th revoked signature. Finally, the community revocation manager may revoke users by updating the list of revoked private-keys (`KRL`) or the list of signatures of revoked users (`SRL`) using Procedure 7.

4.2 Detailed Description of the LEPID Scheme

We now present our LEPID scheme in detail. We start by recalling some standard functionalities that are used in the UC model of the DAA [13]:

- $\mathcal{F}_{\mathsf{CA}}$ is a common certificate authority functionality that is available to all parties.
- $\mathcal{F}_{\mathsf{CRS}}$ is a common reference string functionality that provides participants with all system parameters.
- $\mathcal{F}^*_{\mathsf{auth}}$ is a special authenticated communication functionality that provides an authenticated channel between the issuer and the signer.

The LEPID scheme includes the Setup, Join, Sign, Verify and Revoke procedures that are as follows.

Procedure 1 (Setup). $\mathcal{F}_{\mathsf{CRS}}$ creates the system parameters: $\mathsf{sp} = (\lambda, t, q, n, m, \mathcal{R}_q, \beta, \ell, r, s, \xi)$, where λ, t are positive integer security parameters, β is a positive real number such that $\beta < q$, ℓ is the length of the users' identifiers, and r, s and ξ represent standard deviations of Gaussian distributions.

Upon input $(\text{SETUP}, \mathit{sid})$, where sid is a unique session identifier, the issuer first checks that $\mathit{sid} = (\mathcal{I}, \mathit{sid}')$ for some sid', then creates its key pair. The Issuer's public key is $\mathsf{pp} = (\mathsf{sp}, \boldsymbol{b}, \hat{A}_{\mathcal{I}}, \hat{A}_0, \hat{A}_1, \ldots, \hat{A}_\ell, \boldsymbol{u}, \mathcal{H}_0, \mathcal{H}, H)$, where $\hat{A}_{\mathcal{I}}, \hat{A}_i (i = 0, 1, \ldots, \ell) \in \mathcal{R}_q^m$, $\boldsymbol{b}, \boldsymbol{u} \in \mathcal{R}_q$, $\mathcal{H}_0 : \{0,1\}^* \to \{1,2,3\}^t$, $\mathcal{H} : \{0,1\}^* \to \mathcal{R}_q$, and $H : \{0,1\}^* \to \{0, 1, 2, \ldots, 2n-1\}$. The Issuer's private key is $\hat{T}_{\mathcal{I}}$, which is the trapdoor of $\hat{A}_{\mathcal{I}}$ with $\|\hat{T}_{\mathcal{I}}\|_\infty \leq \beta$.

The issuer initialises the Member List $\texttt{ML} \leftarrow \emptyset$. The issuer proves that his secret key is well formed in $\pi_{\mathcal{I}}$, and registers the key $(\hat{T}_{\mathcal{I}}, \pi_{\mathcal{I}})$ with $\mathcal{F}_{\mathsf{CA}}$ and outputs $(\text{SETUPDONE}, sid)$.

Procedure 2 (Join Request). On input query $(\text{JOIN}, sid, jsid, \mathcal{M})$, the signer $\mathcal{M}$ forwards $(\text{JOIN}, sid, jsid)$ to $\mathcal{I}$, who replies by sending $(sid, jsid, \rho, bsn_{\mathcal{I}})$ back to the signer, where ρ is a uniform random nonce $\rho \leftarrow \{0,1\}^{\lambda}$, and $bsn_{\mathcal{I}}$ is the issuer's base name. The signer $\mathcal{M}$ proceeds as follows:

1. It checks that no such entry exists in its storage.
2. It samples a private key: $\boldsymbol{x}_1 \leftarrow \mathcal{D}_s$ and $(\boldsymbol{x}_2, \ldots, \boldsymbol{x}_{m+1}) \leftarrow \mathcal{D}_r^m$. Let $\hat{X}_t = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{m+1})$ correspond to $\mathcal{M}$'s secret key with the condition $\|(\boldsymbol{x}_2, \ldots, \boldsymbol{x}_{m+1})\|_\infty \leq \beta/2$ and $\|\boldsymbol{x}_1\|_\infty \leq \beta$. $\mathcal{M}$ stores its key as $(sid, \hat{X}_t)$, and computes the corresponding public key $\boldsymbol{u}_t = [\boldsymbol{b}|\hat{A}_{\mathcal{I}}]\hat{X}_t \mod q$, a link token $\mathsf{nym}_{\mathcal{I}} = \mathcal{H}(bsn_{\mathcal{I}})\boldsymbol{x}_1 + \boldsymbol{e}_{\mathcal{I}} \mod q$ for some error $\boldsymbol{e}_{\mathcal{I}} \leftarrow \mathcal{D}_s$ such that $\|\boldsymbol{e}_{\mathcal{I}}\|_\infty \leq \beta$, and generates a signature based proof:

$$\pi_{\boldsymbol{u}_t} = \mathsf{SPK}\Big\{\mathsf{public} := \{\mathsf{pp}, \boldsymbol{u}_t, bsn_{\mathcal{I}}, \mathsf{nym}_{\mathcal{I}}\}, \mathsf{witness} := \{\hat{X}_t = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{m+1}), \boldsymbol{e}_{\mathcal{I}}\},\ \boldsymbol{u}_t = [\boldsymbol{b}|\hat{A}_{\mathcal{I}}]\hat{X}_t \mod q \wedge \|\hat{X}_t/\boldsymbol{x}_1\|_\infty \leq \beta/2\ \wedge \|\boldsymbol{x}_1\| \leq \beta \wedge \mathsf{nym}_{\mathcal{I}} = \mathcal{H}(bsn_{\mathcal{I}})\boldsymbol{x}_1 + \boldsymbol{e}_{\mathcal{I}} \mod q\ \wedge\ \|\boldsymbol{e}_{\mathcal{I}}\|_\infty \leq \beta\Big\}(\rho).$$

3. It sends $(\mathsf{nym}_{\mathcal{I}}, \boldsymbol{u}_t, \pi_{\boldsymbol{u}_t})$ to the issuer by giving $\mathcal{F}^*_{\mathsf{auth}}$ an input $(\text{SEND}, \mathsf{nym}_{\mathcal{I}}, \pi_{\boldsymbol{u}_t}, sid, jsid)$.

$\mathcal{I}$, upon receiving $(\text{SENT}, \mathsf{nym}_{\mathcal{I}}, \pi_{\boldsymbol{u}_t}, sid, jsid, \mathcal{M})$ from $\mathcal{F}^*_{\mathsf{auth}}$, verifies the proof $\pi_{\boldsymbol{u}_t}$ and makes sure that the signer $\mathcal{M} \notin \texttt{ML}$. $\mathcal{I}$ stores $(jsid, \mathsf{nym}_{\mathcal{I}}, \pi_{\boldsymbol{u}_t}, \mathcal{M})$, and generates the message $(\text{JOINPROCEED}, sid, jsid, \mathsf{id}, \pi_{\boldsymbol{u}_t})$, for some identity $\mathsf{id} \in \{0,1\}^{\ell}$ assigned to $\mathcal{M}$, and not used before by any joined member.

Procedure 3 (Join Proceed). If the signer chooses to proceed with the Join session, the message $(\text{JOINPROCEED}, sid, jsid)$ is sent to the issuer, who performs as follows:

1. It checks the record $(jsid, \mathsf{nym}_{\mathcal{I}}, \mathsf{id}, \mathcal{M}, \pi_{\boldsymbol{u}_t})$. For all $\mathsf{nym}'_{\mathcal{I}}$ from the previous Join records, the issuer checks whether $\|\mathsf{nym}_{\mathcal{I}} - \mathsf{nym}'_{\mathcal{I}}\|_\infty \leq 2\beta$ holds; if yes, the issuer further checks if $\boldsymbol{u}_t = \boldsymbol{u}'_t$. If the equality $\boldsymbol{u}_t = \boldsymbol{u}'_t$ holds, the issuer will jump to Step 4 returning $\hat{X}_h = \hat{X}'_h$, if not the issuer will abort. Note that this double check will make sure that no two EPID keys will include the same $\boldsymbol{x}_1$ value.
2. For all $\mathsf{nym}^*_{\mathcal{I}}$ in the Issuer's Revocation record $\texttt{IR}$, the issuer checks whether the equation
$$\|\mathsf{nym}_{\mathcal{I}} - \mathsf{nym}^*_{\mathcal{I}}\|_\infty \leq 2\beta$$
holds, if yes the issuer aborts.
3. It calculates the vector of polynomials $\hat{A}_h = [\hat{A}_{\mathcal{I}}|\hat{A}_0 + \sum_{i=1}^{\ell} \mathsf{id}_i \hat{A}_i] \in \mathcal{R}_q^{2m}$.
4. It samples, using the issuer's private key $\hat{T}_{\mathcal{I}}$, a preimage $\hat{X}_h = [\hat{X}_{h_1}|\hat{X}_{h_2}] = (\boldsymbol{y}_2, ..., \boldsymbol{y}_{2m+1}) \in \mathcal{D}_r^m \times \mathcal{D}_s^m$ of $\boldsymbol{u} - \boldsymbol{u}_t$ such that $\hat{A}_h\hat{X}_h = \boldsymbol{u}_h = \boldsymbol{u} - \boldsymbol{u}_t \mod q$ and $\|\hat{X}_{h_1}\|_\infty \leq \beta/2$ and $\|\hat{X}_{h_2}\|_\infty \leq \beta$.

5. The issuer adds $(\mathsf{nym}_{\mathcal{I}}, \mathsf{id}, \mathcal{M}, \pi_{\boldsymbol{u}_t})$ to his data base, and sends $(sid, jsid, \hat{X}_h)$ to $\mathcal{M}$ via $\mathcal{F}^*_{\mathsf{auth}}$.

When $\mathcal{M}$ receives the message $(sid, jsid, \hat{X}_h)$, it checks that the equations $\hat{A}_h \hat{X}_h = \boldsymbol{u}_h \mod q$ and $\boldsymbol{u} = \boldsymbol{u}_t + \boldsymbol{u}_h$ are satisfied with $\|\hat{X}_{h_1}\|_\infty \leq \beta/2$ and $\|\hat{X}_{h_2}\|_\infty \leq \beta$. It stores $(sid, \mathcal{M}, \mathsf{id}, \hat{X}_h, \boldsymbol{u}_t)$ and outputs (JOINED, $sid, jsid$). $\mathcal{M}$ then computes $\hat{X} = (\boldsymbol{x}_1, \forall_{i=(2,\ldots,m+1)}\ \boldsymbol{x}_i := \boldsymbol{x}_i + \boldsymbol{y}_i, \forall_{i=(m+2,\ldots,2m+1)}\ \boldsymbol{x}_i := \boldsymbol{y}_i)$, where $\|\hat{X}\|_\infty \leq \beta$.

Procedure 4 (Sign Request). Upon input (SIGN, $sid, ssid, \mathcal{M}, \mu$), the signer does the following:

1. It makes sure to have a Join record $(sid, \mathsf{id}, \hat{X}, \mathcal{M})$.
2. It generates a sign entry $(sid, ssid, \mu)$ in its record.
3. Finally it outputs (SIGNPROCEED, $sid, ssid, \mu$).

Procedure 5 (Sign Proceed). When $\mathcal{M}$ gets permission to proceed for $ssid$, the signer proceeds as follows:

1. It retrieves the records $(sid, \mathsf{id}, \pi_{\boldsymbol{u}_t})$ and $(sid, ssid, \mu)$.
2. $\mathcal{M}$ samples a random polynomial $\boldsymbol{p}$ and computes the polynomial $\mathsf{nym} = \boldsymbol{p}\boldsymbol{x}_1 + \boldsymbol{e} \mod q$, for an error term $\boldsymbol{e} \leftarrow \mathcal{D}_s$ such that $\|\boldsymbol{e}\|_\infty \leq \beta$. $\mathcal{M}$ then generates a signature based knowledge proof π.
$$\pi = \mathsf{SPK}\Big\{\mathsf{public} := \{\mathsf{pp}, \mathsf{nym}, \boldsymbol{p}\},$$
$$\mathsf{witness} := \{\hat{X} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{2m+1}), \mathsf{id}, \boldsymbol{e}\} :$$
$$[\boldsymbol{b}|\hat{A}_h]\hat{X} = \boldsymbol{u} \ \wedge\ \|\hat{X}\|_\infty \leq \beta \ \wedge\ \mathsf{nym} = \boldsymbol{p}\boldsymbol{x}_1 + \boldsymbol{e} \ \wedge\ \|\boldsymbol{e}\|_\infty \leq \beta\Big\}(\mu).$$
The details of the proof π are presented in Appendix A.1.
3. The signer proves that it is not using any of the keys that produced a revoked signature $(\sigma_i^*, \boldsymbol{p}_i^*, \mathsf{nym}_i^*)$ in the signature revocation list (more details about the proof can be found in Appendix A.2).
   - Let $\mathsf{nym}_i^* = \boldsymbol{p}_i^* \boldsymbol{f}_i + \boldsymbol{l}_i$, where $(\boldsymbol{f}_i, \boldsymbol{l}_i)$ were used before to create nym_i^* by some $\mathcal{M}_i{}^*$ that generated a revoked signature $\sigma_i^* \in \mathtt{SRL}$. $\mathcal{M}$ proceeds as follows:
     - $\boldsymbol{q}_i, \boldsymbol{l}'_i, \boldsymbol{l}''_i, \boldsymbol{l}'''_i \leftarrow \mathcal{D}_s$
     - $\boldsymbol{o}_i = \boldsymbol{p}_i^* \boldsymbol{q}_i + \boldsymbol{l}'_i$, $\boldsymbol{k}_i = \boldsymbol{o}_i \boldsymbol{x}_1 + \boldsymbol{l}''_i$, $\boldsymbol{d}_i = \mathsf{nym}_i^* \boldsymbol{q}_i + \boldsymbol{l}'''_i$
     - $\boldsymbol{r}_{x_1}, \boldsymbol{r}_e, \boldsymbol{r}_{q_i}, \boldsymbol{r}_{l'_i}, \boldsymbol{r}_{l''_i}, \boldsymbol{r}_{l'''_i} \leftarrow \mathcal{D}_s$
     - $\boldsymbol{t}_{\mathsf{nym}} = \boldsymbol{p}\boldsymbol{r}_{x_1} + \boldsymbol{r}_e$, $\boldsymbol{t}_{o_i} = \boldsymbol{p}_i^* \boldsymbol{r}_{q_i} + \boldsymbol{r}_{l'_i}$,
       $\boldsymbol{t}_{k_i} = \boldsymbol{o}_i \boldsymbol{r}_{x_1} + \boldsymbol{r}_{l''_i}$, $\boldsymbol{t}_{d_i} = \mathsf{nym}_i^* \boldsymbol{r}_{q_i} + \boldsymbol{r}_{l'''_i}$.
   - Calculates the challenge $c_v = H(\boldsymbol{t}_{\mathsf{nym}} | \boldsymbol{t}_{o_i} | \boldsymbol{t}_{k_i} | \boldsymbol{t}_{d_i} | \mu) \in \{0, 1, 2, \ldots, 2n-1\}$.
   - The following responses are computed:
     - $\boldsymbol{s}_{x_1} = \boldsymbol{r}_{x_1} + X^{c_v} \boldsymbol{x}_1$, $\boldsymbol{s}_e = \boldsymbol{r}_e + X^{c_v} \boldsymbol{e}$, $\boldsymbol{s}_{q_i} = \boldsymbol{r}_{q_i} + X^{c_v} \boldsymbol{q}_i$,
       $\boldsymbol{s}_{l'_i} = \boldsymbol{r}_{l'_i} + X^{c_v} \boldsymbol{l}'_i$, $\boldsymbol{s}_{l''_i} = \boldsymbol{r}_{l''_i} + X^{c_v} \boldsymbol{l}''_i$, $\boldsymbol{s}_{l'''_i} = \boldsymbol{r}_{l'''_i} + X^{c_v} \boldsymbol{l}'''_i$.

     Abort if any of these rejection samples outputs 1:
     - $\mathtt{rej}(\boldsymbol{s}_{x_1}, X^{c_v} \boldsymbol{x}_1, \xi)$, $\mathtt{rej}(\boldsymbol{s}_e, X^{c_v} \boldsymbol{e}, \xi)$, $\mathtt{rej}(\boldsymbol{s}_{q_i}, X^{c_v} \boldsymbol{q}_i, \xi)$,
       $\mathtt{rej}(\boldsymbol{s}_{l'_i}, X^{c_v} \boldsymbol{l}'_i, \xi)$, $\mathtt{rej}(\boldsymbol{s}_{l''_i}, X^{c_v} \boldsymbol{l}''_i, \xi)$ or $\mathtt{rej}(\boldsymbol{s}_{l'''_i}, X^{c_v} \boldsymbol{l}'''_i, \xi)$.

4. Finally, $\mathcal{M}$ outputs $\sigma = (\pi, \mathsf{nym}, \boldsymbol{o}_i, \boldsymbol{k}_i, \boldsymbol{d}_i, \boldsymbol{s}_{x_1}, \boldsymbol{s}_e, \boldsymbol{s}_{q_i}, \boldsymbol{s}_{l'_i}, \boldsymbol{s}_{l''_i}, \boldsymbol{s}_{l'''_i}, c_v, \texttt{KRL}, \texttt{SRL})$.

Procedure 6 (Verify). Let KRL denotes the revocation list with all the rogue signer's secret keys $\boldsymbol{x}_1^*$. Upon input (VERIFY, $sid, \sigma, \mu, \texttt{KRL}, \texttt{SRL}$), the verifier proceeds as follows:

1. It checks the zero-knowledge proof regarding the statement: $\{[\boldsymbol{b}|\ \hat{A}_h]\ \hat{X} = \boldsymbol{u} \wedge \|\ \hat{X}\|_\infty \leq \beta \wedge \mathsf{nym} = \boldsymbol{p}\ \boldsymbol{x}_1 + \boldsymbol{e} \mod q \wedge \|\ \boldsymbol{e}\ \|_\infty \leq \beta.\ \}$
2. For all $\boldsymbol{x}_1^* \in \texttt{KRL}$, if $\|\boldsymbol{p}\boldsymbol{x}_1^* - \mathsf{nym}\|_\infty \leq \beta$ the verifier outputs 0.
3. For all $\sigma_i^* = (\pi_{\mathsf{nym}_i^*}, \mathsf{nym}_i^*, \boldsymbol{p}_i^*) \in \texttt{SRL}$, the verifier
   (a) computes:
      – $\boldsymbol{t'}_{k_i}{=}\boldsymbol{o}_i\boldsymbol{s}_{x_1} + \boldsymbol{s}_{l''_i} - X^{c_v}\boldsymbol{k}_i$, $\boldsymbol{t'}_{d_i}{=}\mathsf{nym}_i^*\boldsymbol{s}_{q_i} + \boldsymbol{s}_{l'''_i} - X^{c_v}\boldsymbol{d}_i$, $\boldsymbol{t'}_{o_i}{=}\boldsymbol{p}_i^*\boldsymbol{s}_{q_i} + \boldsymbol{s}_{l'_i} - X^{c_v}\boldsymbol{o}_i$, $\boldsymbol{t'}_{\mathsf{nym}}{=}\boldsymbol{p}\boldsymbol{s}_{x_1} + \boldsymbol{s}_e - X^{c_v}\mathsf{nym}$.
   (b) checks $c_v \overset{?}{=} H(\boldsymbol{t'}_{\mathsf{nym}}|\boldsymbol{t'}_{o_i}|\boldsymbol{t'}_{k_i}|\boldsymbol{t'}_{d_i}|\mu)$ and that all the following norms satisfy $\|\boldsymbol{s}_{\boldsymbol{x}_1}\|_\infty, \|\boldsymbol{s}_e\|_\infty, \|\boldsymbol{s}_{q_i}\|_\infty, \|\boldsymbol{s}_{l'_i}\|_\infty, \|\boldsymbol{s}_{l''_i}\|_\infty, \|\boldsymbol{s}_{l'''_i}\|_\infty \leq \beta + \sqrt{n}\beta$.
4. For all $\sigma_i^* = (\pi_{\mathsf{nym}_i^*}, \mathsf{nym}_i^*, \boldsymbol{p}_i^*)$, the verifier checks $2\|\boldsymbol{d}_i - \boldsymbol{k}_i\| < \Gamma$, where Γ is a function of β. If $2\|\boldsymbol{d}_i - \boldsymbol{k}_i\| < \Gamma$ the verifier outputs 0, otherwise 1.

Procedure 7 (Revoke). On input (Revoke, $sid, \boldsymbol{x}_1^*, \texttt{KRL}$) or (Revoke, sid, σ^*, $\mu^*, \texttt{SRL}$), the revocation manager adds $\boldsymbol{x}_1^*$ to KRL or σ^* to SRL after verifying σ^*.

5 A Sketched Security Proof for LEPID

In this section, we provide a sketch of the security proof of the LEPID scheme. A detailed security proof is presented in Appendix B. A variant of the sequence of games of [13] is presented, showing that no environment $\mathcal{E}$ can distinguish the real world protocol Π with an adversary $\mathcal{A}$, from the ideal world $\mathcal{F}^l_{\mathsf{EPID}}$ 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.

Game 1. This is the real world protocol.

Game 2. An entity C is introduced, that receives all inputs from the honest parties and simulates Π for them. This is equivalent to Game 1.

Game 3. C is split into $\mathcal{F}$ and $\mathcal{S}$. $\mathcal{F}$ behaves as an ideal functionality, receiving all inputs and forwarding them to $\mathcal{S}$, who simulates the real world protocol for honest parties. $\mathcal{S}$ sends the outputs to F, who forwards them to $\mathcal{E}$. This game is similar to Game 2, but with a different structure.

Game 4. $\mathcal{F}$ now behaves differently in the setup interface. It stores the algorithms for the issuer $\mathcal{I}$, and checks that the structure of sid is correct for an honest $\mathcal{I}$, aborting if not. In case $\mathcal{I}$ is corrupt, $\mathcal{S}$ extracts the secret key for $\mathcal{I}$ and proceeds in the setup interface on behalf of $\mathcal{I}$. Clearly $\mathcal{E}$ will notice no change.

Game 5. $\mathcal{F}$ now performs the verification and key revokation checks instead of forwarding them to $\mathcal{S}$. There are no protocol messages and the outputs are exactly as the real world protocol. However, the verification algorithm that $\mathcal{F}$

uses does not contain any key or signature revocation checks. $\mathcal{F}$ can perform this check separately, so the outcomes are equal.

Game 6. $\mathcal{F}$ stores in its records the members that have joined. If $\mathcal{I}$ is honest, $\mathcal{F}$ stores the secret key tsk, extracted from $\mathcal{S}$, for corrupt platforms. $\mathcal{S}$ always has enough information to simulate the real world protocol except when the issuer is the only honest party. In this case, $\mathcal{S}$ does not know who initiated the join, and so cannot make a join query with $\mathcal{F}$ on the signer's behalf. Thus, to deal with this case, $\mathcal{F}$ can safely choose any corrupt signer and put it into Members. The identities of signers are only used for creating signatures for honest signers, so corrupted signers do not matter. In the case that the signer is already registered in Members, $\mathcal{F}$ would abort the protocol, but $\mathcal{I}$ will have already tested this case before continuing with the query JOINPROCEED. Hence $\mathcal{F}$ will not abort. Thus in all cases, $\mathcal{F}$ and $\mathcal{S}$ can interact to simulate the real world protocol.

Game 7. (Anonymity). In this game, $\mathcal{F}$ 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 $\mathcal{F}$ handles the first k' signing inputs of $\mathcal{M}_i$ for $i < k$ using algorithms, and subsequent inputs are forwarded to $\mathcal{S}$ who creates signatures as before. We note that Game 7.0.0=Game 6. For increasing k', Game $7.k.k'$ will be at some stage equal to Game $7.k+1.0$, this is because there can only be a polynomial number of signing queries to be processed. Therefore, for large enough k and k', $\mathcal{F}$ handles all the signing queries of all signers, and Game 7 is indistinguishable from Game $7.k.k'$. To prove that Game $7.k.k'+1$ is indistinguishable from Game $7.k.k'$, suppose that there exists an environment that can distinguish a signature of an honest party using $tsk = \boldsymbol{x}_1$ from a signature using a different $tsk^j = \boldsymbol{x}_1^j$, then the environment can solve the Decision Ring -LWE Problem.

The first $j \leq k'$ signing queries on behalf of $\mathcal{M}_k$ are handled by $\mathcal{F}$ using the algorithms, and subsequent inputs are then forwarded to $\mathcal{S}$ as before. Now suppose that F outputs the tuples $(\mathsf{nym}^j, \boldsymbol{p}^j, \boldsymbol{o}_i^j, \boldsymbol{k}_i^j, \boldsymbol{d}_i^j, \boldsymbol{s}_{x_1}^j, \boldsymbol{s}_e^j, \boldsymbol{s}_{q_i}^j, \boldsymbol{s}_{l_i'}^j, \boldsymbol{s}_{l_i''}^j, \boldsymbol{s}_{l_i'''}^j, c_v^j, \mathtt{SRL})$ for $j \leq k'$, with $\mathsf{nym}^j = \boldsymbol{p}^j \boldsymbol{x}_1 + \boldsymbol{e}^j$, for an error term $\boldsymbol{e}_j \leftarrow \mathcal{D}_s$, and the remaining proofs are honestly generated. The $j = k' + 1$-th query for $\mathcal{M}_k$ is as follows: $(\mathsf{nym}^{\mathcal{S}}, \boldsymbol{p}^{\mathcal{S}}, \boldsymbol{o}_i^{\mathcal{S}}, \boldsymbol{k}_i^s, \boldsymbol{d}_i^{\mathcal{S}}, \boldsymbol{s}_{x_1}^{\mathcal{S}}, \boldsymbol{s}_e^{\mathcal{S}}, \boldsymbol{s}_{q_i}^{\mathcal{S}}, \boldsymbol{s}_{l_i'}^{\mathcal{S}}, \boldsymbol{s}_{l_i''}^{\mathcal{S}}, \boldsymbol{s}_{l_i'''}^{\mathcal{S}}, c_v^{\mathcal{S}}, \mu^{\mathcal{S}}, \mathtt{SRL})$. $\mathcal{S}$ is challenged to decide if $(\mathsf{nym}^{\mathcal{S}}, \boldsymbol{p}^{\mathcal{S}}, \boldsymbol{o}_i^{\mathcal{S}}, \boldsymbol{k}_i^s, \boldsymbol{d}_i^{\mathcal{S}}, \boldsymbol{s}_{x_1}^{\mathcal{S}}, \boldsymbol{s}_e^{\mathcal{S}}, \boldsymbol{s}_{q_i}^{\mathcal{S}}, \boldsymbol{s}_{l_i'}^{\mathcal{S}}, \boldsymbol{s}_{l_i''}^{\mathcal{S}}, \boldsymbol{s}_{l_i'''}^{\mathcal{S}}, c_v^{\mathcal{S}}, \mu^{\mathcal{S}}, \mathtt{SRL})$ is chosen from a Ring LWE distribution for some secret $\boldsymbol{x}_1$ or uniformly at random. $\mathcal{S}$ proceeds in simulating the signer without knowing the secret $\boldsymbol{x}_1$. $\mathcal{S}$ can answer all the H queries, as $\mathcal{S}$ is controlling $\mathcal{F}_{\mathsf{CRS}}$. $\mathcal{S}$ sets: $\boldsymbol{t}_{k_i}^{\mathcal{S}} = \boldsymbol{o}_i^{\mathcal{S}} \boldsymbol{s}_{x_1}^{\mathcal{S}} + \boldsymbol{s}_{l_i''}^{\mathcal{S}} - X^{c_v^{\mathcal{S}}} \boldsymbol{k}_i^{\mathcal{S}}$; $\boldsymbol{t}_{d_i}^{\mathcal{S}} = \mathsf{nym}_i^* \boldsymbol{s}_{q_i}^{\mathcal{S}} + \boldsymbol{s}_{l_i'''}^{\mathcal{S}} - X^{c_v^{\mathcal{S}}} \boldsymbol{d}_i^{\mathcal{S}}$; $\boldsymbol{t}_{o_i}^{\mathcal{S}} = \boldsymbol{p}_i^* \boldsymbol{s}_{q_i}^{\mathcal{S}} + \boldsymbol{s}_{l_i'}^{\mathcal{S}} - X^{c_v^{\mathcal{S}}} \boldsymbol{o}_i^{\mathcal{S}}$; $\boldsymbol{t}_{\mathsf{nym}}^{\mathcal{S}} = \boldsymbol{p}^{\mathcal{S}} \boldsymbol{s}_{x_1}^{\mathcal{S}} + \boldsymbol{s}_e^{\mathcal{S}} - X^{c_v^{\mathcal{S}}} \mathsf{nym}^{\mathcal{S}}$; and, finally, $c_v^{\mathcal{S}} := H(\boldsymbol{t}_{\mathsf{nym}}^{\mathcal{S}} | \boldsymbol{t}_{o_i}^{\mathcal{S}} | \boldsymbol{t}_{k_i}^{\mathcal{S}} | \boldsymbol{t}_{d_i}^{\mathcal{S}} | \mu^{\mathcal{S}})$. For $i > k' + 1$, $\mathcal{S}$ outputs the tuples $(\mathsf{nym}^j, \boldsymbol{p}^j, \boldsymbol{o}_i^j, \boldsymbol{k}_i^j, \boldsymbol{d}_i^j, \boldsymbol{s}_{x_1}^j, \boldsymbol{s}_e^j, \boldsymbol{s}_{q_i}^j, \boldsymbol{s}_{l_i'}^j, \boldsymbol{s}_{l_i''}^j, \boldsymbol{s}_{l_i'''}^j, c_v^j, \mu^j, \mathtt{SRL})$, with $\mathsf{nym}^j = \boldsymbol{p}^j \boldsymbol{x}_1^j + \boldsymbol{e}^j \mod q$, for some freshly generated secret $\boldsymbol{x}_1^j$ and error term $\boldsymbol{e}^j \leftarrow \mathcal{D}_s$. For each case, $\mathcal{M}_k$ can provide a simulated proof as follows. $\mathcal{S}$ sets $\boldsymbol{t}_{k_i}^j = \boldsymbol{o}_i^j \boldsymbol{s}_{x_1}^j + \boldsymbol{s}_{l_i''}^j - X^{c_v^j} \boldsymbol{k}_i^j$; $\boldsymbol{t}_{d_i}^j = \mathsf{nym}_i^* \boldsymbol{s}_{q_i}^j + \boldsymbol{s}_{l_i'''}^j - X^{c_v^j} \boldsymbol{d}_i^j$; $\boldsymbol{t}_{o_i}^j = \boldsymbol{p}_i^* \boldsymbol{s}_{q_i}^j + \boldsymbol{s}_{l_i'}^j - X^{c_v^j} \boldsymbol{o}_i^j$; $\boldsymbol{t}_{\mathsf{nym}}^j = \boldsymbol{p}^j \boldsymbol{s}_{x_1}^j + \boldsymbol{s}_e^j - X^{c_v^j} \mathsf{nym}^j$; and, finally, $c_v^j := H(\boldsymbol{t}_{\mathsf{nym}}^j | \boldsymbol{t}_{o_i}^j | \boldsymbol{t}_{k_i}^j | \boldsymbol{t}_{d_i}^j | \mu^j)$.

Thus, any distinguisher between Game $7.k.k'$ and Game $7.k.k'+1$ can solve the Decision Ring LWE Problem.

Game 8. $\mathcal{F}$ now no longer informs $\mathcal{S}$ about the message and $\boldsymbol{p}$ that are being signed. If the signer $\mathcal{M}$ is honest, then $\mathcal{S}$ can learn nothing about the message μ and $\boldsymbol{p}$. Instead, $\mathcal{S}$ knows only the leakage $l(\mu, \boldsymbol{p})$. To simulate the real world, $\mathcal{S}$ chooses a pair $(\mu', \boldsymbol{p}')$ such that $l(\mu', \boldsymbol{p}')$=$l(\mu, \boldsymbol{p})$. An environment $\mathcal{E}$ observes no difference, and thus Game 8=Game 7.

Game 9. If $\mathcal{I}$ is honest, then $\mathcal{F}$ now only allows members that joined to sign. An honest signer will always check whether it has joined before signing in the real world protocol, so there is no difference for honest signers. Therefore Game 9=Game 8.

Game 10. When storing a new $tsk = \boldsymbol{x}_1$, $\mathcal{F}$ checks $\mathsf{CheckTskCorrupt}(tsk)$=1 or $\mathsf{CheckTskHonest}(tsk)$=1. We want to show that these checks will always pass. In fact, valid signatures always satisfy $\mathsf{nym} = \boldsymbol{p}\boldsymbol{x}_1 + \boldsymbol{e}$ where $\|\boldsymbol{x}_1\|_\infty \leq \beta$ and $\|\boldsymbol{e}\|_\infty \leq \beta$. By the unique Shortest Vector Problem, there exists only one tuple $(\boldsymbol{x}_1, \boldsymbol{e})$ such that $\|\boldsymbol{x}_1\|_\infty \leq \beta$ and $\|\boldsymbol{e}\|_\infty \leq \beta$ for small enough β. Thus, $\mathsf{CheckTskCorrupt}(tsk)$ will always give the correct output. Also, due to the large min-entropy of discrete Gaussians the probability of sampling $\boldsymbol{x}_1' = \boldsymbol{x}_1$, and thus of having a signature already using the same $tsk = \boldsymbol{x}_1$, is negligible, which implies that $\mathsf{CheckTskHonest}(tsk)$ will give the correct output with overwhelming probability. Hence Game 10=Game 9.

Game 11. (Completeness). In this game, $\mathcal{F}$ checks that honestly generated signatures are always valid. This is true as sig algorithm always produces signatures passing through verification checks. Those signatures satisfy $\mathsf{identify}(tsk, \sigma, \mu, \boldsymbol{p}) = 1$, which is checked via nym. $\mathcal{F}$ also makes sure, using its internal records Members and DomainKeys that honest users are not sharing the same secret key tsk. If there exists a key $tsk' = \boldsymbol{x}_1'$ in Members and DomainKeys such that $\|\mathsf{nym} - \boldsymbol{p}\boldsymbol{x}_1'\|_\infty \leq \beta$, then this breaks search Ring-LWE.

Game 12. Check-IX is added to ensure that there are no multiple tsk tracing back to the same signature. Since there exists only one pair $(\boldsymbol{x}_1, \boldsymbol{e}_\mathcal{I})$, $\|\boldsymbol{x}_1\|_\infty \leq \beta$, $\|\boldsymbol{e}_\mathcal{I}\|_\infty \leq \beta$, satisfying $\mathsf{nym}_\mathcal{I} = \mathcal{H}(bsn_\mathcal{I})\boldsymbol{x}_1 + \boldsymbol{e}_\mathcal{I}$, two different signers cannot share the same $\boldsymbol{x}_1$, thus any valid signature traces back to a single tsk.

Game 13. (Unforgeability). To prevent accepting signatures that were issued by the use of join credentials not issued by an honest issuer, $\mathcal{F}$ further adds Check-X. This is due to the unforgeability of Boyen signatures [15].

Game 14. (Unforgeability). Check-XI is added to $\mathcal{F}$, preventing the forging of signatures with honest tsk and credentials. If a valid signature is given on a message that the signer has never signed, the proof could not have been simulated. $\boldsymbol{x}_1$ would be extracted and Ring-LWE would be broken. So Game 14=Game 13.

Game 15. Check-XII is added to $\mathcal{F}$, ensuring that honest signers keys are not being revoked. If an honest signer is simulated by means of the Ring-LWE problem instance and a proper key KRL is found, it must be the secret key of the target instance. This is equivalent to solving the search Ring-LWE problem.

Game 16. $\mathcal{F}$ now performs signature based revocation when verifying signatures. $\mathcal{F}$ checks that there is no $(\sigma^*, \mathsf{nym}^*, \boldsymbol{p}^*) \in \mathtt{SRL}$ such that for some matching tsk_i and $(\sigma^*, \mu^*, \boldsymbol{p}^*) \in \mathtt{SRL}$, we have identify$(\sigma^*, \mu^*, \boldsymbol{p}^*, tsk_i) = 1$. By the soundness of the proof presented in Appendix A.2, this check will always pass with overwhelming probability. □

6 Experimental Results

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 $m = O(\log q)$ as in [10]. A concrete choice of parameters can be as follows: $n = 512,\ l = 32,\ q = 8380417,\ m = 24$, and $\beta = 275$.

Both LDAA and LEPID were implemented in C, emulating all entities in a single machine. The code was compiled with gcc 4.8.5 with the `-O3` and `-march=native` flags and executed on an Intel i9 7900X CPU with 64GB running at 3.3 GHz operated by CentOS 7.5. The obtained experimental results can be found in Table 1. Note that the measured times for signing and verification do not take into account transfer times between the entities or object creation and destruction.

By construing the signer as a single entity instead of two as in the LDAA, the proposed LEPID scheme achieves a reduction of the private-key size of 1.5 times. While the comparison in signatures sizes between both schemes yields favourably for the proposed LEPID scheme with a small amount of rejected users, as the number of users in the SRL increases, its signature size increases linearly at a rate of 18kB per rejected user (9 polynomials and an integer). When the SRL contains 500 users, the LEPID signature size closely matches that of the LDAA scheme. Should LEPID signing be implemented on a device with limited computational resources like the TPM, its constrained memory resources and the cost of data transfer might limit its application to small and medium-sized communities. In particular, if one considers a revocation rate of 0.1%, LEPID

Scheme	Private-key (kB)	Signature (MB)	Signing Time (s)	Verification Time (s)
LDAA	147	847	541	129
LEPID (no revoked users)	100	836	361	114
LEPID (100 users in SRL)	100	838	371	117
LEPID (500 users in SRL)	100	845	372	119
LEPID (1000 users in SRL)	100	854	374	121

Table 1: Experimental results for the proposed LEPID and LDAA [8] for $n = 512$, $q = 8380417$, $l = 32$, $m = 24$ and $\beta = 256$ obtained on an Intel i9 7900X

signatures will compare favourably in size to LDAA signatures for communities with fewer than 500,000 users.

The signing time in the LEPID scheme is dominated by the signature based knowledge proof π. The addition of the `SRL` , and consequently of 13 polynomial multiplications per rejected user, shows no meaningful impact in the final signing time, where LEPID maintains a speedup of 1.4 over the LDAA scheme. Likewise, in the verification time, the additional 2 polynomial multiplications per rejected user incurred by the `SRL` are negligible compared to the verification of π. Hence, the proposed LEPID scheme achieves a speedup of 1.1 when compared with the LDAA scheme across both small and medium rejection lists. For the computational complexity introduced by the `SRL` to be meaningful, the number of rejected users must be in the order of millions. Once more, the proposed LEPID scheme shows improved signature and verification times for small and medium communities when compared with the LDAA.

7 Related Work

A post-quantum EPID scheme has been proposed in [3] built on hash and pseudorandom functions. More concretely, the EPID credential corresponds to a hash-based signature generated by the issuer, and proofs-of-knowledge are constructed from the Multi-Party Computation (MPC) in the head technique from Ishai et al. [16]. While [3] achieves signature sizes in the order of MBs, execution times are not considered. The main goal of this article is not to outperform [16], but rather to ignite research on lattice-based EPID partially propelled by the National Institute of Standards and Technology (NIST)'s effort on post-quantum cryptography standardisation [17]. By basing our construction on lattices, future versions of EPID might leverage the research resulting from this standardisation process to improve their efficiency. Moreover, since post-quantum cryptography is still in its infancy, it might be useful for implementers to consider multiple security assumptions, to mitigate the effects of cryptanalysis against one of them.

8 Conclusion

While EPID plays a determinant role in the security of SGX, the scheme currently deployed by Intel will become insecure in the event that a large-scale quantum-computer is produced. Herein, a novel EPID scheme is proposed, supported on lattice-based security assumptions, and achieving presumed quantum resistance. A security model for EPID is presented for the first time in the UC framework, and the proposed scheme is proven secure under this model. When compared with a closely related LDAA scheme from related art, the proposed LEPID achieves a reduction in the private-key size of 1.5 times, and of the signature and verification times of 1.4 and 1.1 times, respectively, when no users have been revoked. It is furthermore shown, experimentally, that the overhead introduced by the more effective revocation method of LEPID is minimal for small to medium-sized communities. Finally, it is expected that the proposed LEPID may benefit from theoretical developments and hardware accelerators that result from the increased interest that lattice-based cryptography has gathered in the last few years.

Acknowledgements. This research was supported by European Unions Horizon 2020 research and innovation programme under grant agreement No. 779391 (FutureTPM), and by national funds through Fundação para a Ciência e a Tecnologia (FCT) with references UID/CEC/50021/2019 and FCT Grant No. SFRH/BD/145477/2019.

References

1. Simon Johnson, Vinnie Scarlata, Carlos Rozas, Ernie Brickel, and Frank Mckeen. Intel Software Guard Extensions: EPID Provisioning and Attestation Services Intel, 2016. https://software.intel.com/en-us/download/intel-sgx-intel-epid-provisioning-and-attestation-services.
2. 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.
3. Dan Boneh, Saba Eskandarian and Ben Fisch. Post-quantum EPID Signatures from Symmetric Primitives. In *Topics in Cryptology – CT-RSA 2019*, pages 251–271. Springer, 2019.
4. Vadim Lyubashevsky. *Towards practical lattice-based cryptography.* University of California, San Diego, 2008.
5. 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.
6. Fabrice Benhamouda, Jan Camenisch, Stephan Krenn, Vadim Lyubashevsky, and Gregory Neven. Better zero-knowledge proofs for lattice encryption and their application to group signatures. In *ASIACRYPT (1)*, pages 551–572. Springer, 2014.
7. Hamid Nejatollahi, Nikil Dutt, Indranil Banerjee and Rosario Cammarota Domain-specific Accelerators for Ideal Lattice-based Public Key Protocols *IACR Cryptology ePrint Archive*, 2018:608, 2018.
8. Nada Kassem, Liqun Chen, Rachid El Bansarkhani, Ali El Kaafarani, Jan Camenisch, Patrick Hough, Paulo Martins, and Leonel Sousa. More efficient, provably-secure direct anonymous attestation from lattices. *Future Generation Computer Systems*, 2019.
9. Vadim Lyubashevsky. Lattice signatures without trapdoors. In *EUROCRYPT*, volume 7237 of *LNCS*, 2012.
10. Chris Peikert et al. A decade of lattice cryptography. *Foundations and Trends® in Theoretical Computer Science*, 10(4):283–424, 2016.
11. Oded Regev. The learning with errors problem (invited survey). In *2010 IEEE 25th Annual Conference on Computational Complexity.*
12. 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.

13. 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.
14. 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.
15. 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.
16. Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge proofs fromsecure multiparty computation. In *SIAM J. Comput.*, 39(3):1121–1152, 2009.
17. National Institute of Standards and Technology Post-Quantum Cryptography. 2019. https://csrc.nist.gov/Projects/Post-Quantum-Cryptography.

A Zero Knowledge Proofs in the LEPID Scheme

A.1 The details of π

Now we explain the details on how to compute 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 a matrix vector product. We construct the matrices $\bar{A}_i = \mathsf{rot}(\boldsymbol{a}_i)$ for $i = (1, 2, ..., (\ell+2)m+1)$ for all polynomials $\boldsymbol{a}_i$ in $\boldsymbol{b}$, $\hat{A}_I$, $\hat{A}_0$, ..., $\hat{A}_\ell$ 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* $i = (2+\ell)m+1$.

We apply the techniques of decomposition and extension described in [12] on each of the vectors of $\hat{X}$ and the vector $\boldsymbol{e}$, to get the vectors:

$$\{\boldsymbol{e}_j\}_{j=1}^k, \{\boldsymbol{x}_1^j\}_{j=1}^k, \{\boldsymbol{x}_2^j\}_{j=1}^k, \ldots, \{\boldsymbol{x}_{2m+1}^j\}_{j=1}^k \in B_{3n}.$$

Let $\bar{A}^*_{i+(j+1)m+1} = \mathbf{0}$ for $j > \ell$, and let $\boldsymbol{x}_{(1+i)m+j} = \mathsf{id}^* \cdot \boldsymbol{x}_{m+1+j}$ for $1 \le i \le 2\ell$ and $1 \le j \le m$, and $\boldsymbol{x}_i = \mathbf{0} \in \mathbb{Z}^n$ for $2m+1 < i \le (2+2\ell)m+1$. Then,

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

Before proceeding with the proof, the prover:

1. Samples the following masking vectors: $\{\boldsymbol{r}_{\boldsymbol{e}}^j \leftarrow \mathbb{Z}_q^{3n}\}_{j=1}^k$, $\{\boldsymbol{r}_i^j \leftarrow \mathbb{Z}_q^{3n}\}_{j=1}^k$ for $i \in [2(1+\ell)m+1]$ and $j \in [k]$, and $\boldsymbol{r}_{\mathsf{id}^*} \leftarrow \mathbb{Z}_q^{2\ell}$.
2. Defines the following terms: $D = [\mathsf{rot}(\boldsymbol{p})|\boldsymbol{0}] \in \mathbb{Z}_q^{n\times 3n}$, $\boldsymbol{v}_i^j = \boldsymbol{x}_i^j + \boldsymbol{r}_i^j$, $\boldsymbol{v}_{\boldsymbol{e}}^j = \boldsymbol{e}^j + \boldsymbol{r}_{\boldsymbol{e}}^j$, and $\boldsymbol{v}_{\mathsf{id}^*} = \mathsf{id}^* + \boldsymbol{r}_{\mathsf{id}^*}$.
Note that the above selection needs to satisfy the following equation:
$$\boldsymbol{u} + \sum_{i=1}^{2(1+\ell)m+1} \bar{A}_i^* \cdot \left(\sum_{j=1}^{k} \beta_j \cdot \boldsymbol{r}_i^j\right) = \sum_{i=1}^{2(1+\ell)m+1} \bar{A}_i^* \cdot \left(\sum_{j=1}^{k} \beta_j \cdot \boldsymbol{v}_i^j\right)$$
3. Samples the permutations as follows: $\tau \leftarrow \mathsf{S}_{2\ell}$ for id^*, $\{\phi_j \leftarrow \mathsf{S}_{3n}\}_{j=1}^k$ for $\hat{X}_1$, $\{\psi_j \leftarrow \mathsf{S}_{3n}\}_{j=1}^k$ for $\hat{X}_2$, where $\hat{X} = [\hat{X}_1 \in \mathcal{R}_q^{m+1}|\hat{X}_2 \in \mathcal{R}_q^m]$,

Now, we are ready to explain the result.
To create π, $\mathcal{M}$ first generates the commitments $CMT = (\boldsymbol{C}_1, \boldsymbol{C}_2, \boldsymbol{C}_3)$ where:

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

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 $\{1,2,3\}^t$.

Challenge: The prover generates the challenges using Fiat-Shamir's hash-function transformation, which is based on a random oracle, which should only include CMT:
$$\{\boldsymbol{CH}_j\}_{j=1}^t = \mathcal{H}_0(\mu, \{CMT_j\}_{j=1}^t, \mathsf{pp}) = \{1,2,3\}^t.$$

Response: For each challenge, $\mathcal{M}$ sends its own response to to the verifier. The resulting responses are treated as follows:

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

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

A.2 Signature Proof of knowledge $\pi_{\boldsymbol{k}_i}$(same for $\pi_{\boldsymbol{d}_i}$, $\pi_{\boldsymbol{o}_i}$)

The proofs π_{k_i}, π_{d_i} and π_{o_i} allow the prover to efficiently demonstrate knowledge of small secrets $2\boldsymbol{q}_i, 2\boldsymbol{l}'_i, 2\boldsymbol{l}''_i, 2\boldsymbol{l}'''_i \leftarrow \mathcal{D}_{s'}$ such that:

– $2\boldsymbol{o}_i = 2\boldsymbol{p}^*_i\boldsymbol{q}_i + 2\boldsymbol{l}'_i$
– $2\boldsymbol{k}_i = 2\boldsymbol{o}_i\boldsymbol{x}_1 + 2\boldsymbol{l}''_i$
– $2\boldsymbol{d}_i = 2\mathsf{nym}^*_i\boldsymbol{q}_i + 2\boldsymbol{l}'''_i$

We now explain in detail the proof π_{k_i} and show its properties: completeness, honest-verifier zero-knowledge and special soundness.

The following lemma shows that there is always a subset of polynomials in the ring $\mathcal{R}_q$ that are invertible such that their inverses have only small coefficients.

Lemma 1. *[6] Let $0 < i, j < 2n-1$, then the polynomial $2(X^i - X^j)^{-1}$ has infinite norm at most 1 over the ring $\mathcal{R}_q$.*

Let $\boldsymbol{k}_i = \boldsymbol{o}_i\boldsymbol{x}_1 + \boldsymbol{l}''_i$, where $\boldsymbol{x}_1$ and $\boldsymbol{l}''_i$ are chosen from $\mathcal{D}_s$ and play the roles of the LWE-secrets. Our scheme convinces the verifier that the signer knows some secret $\bar{x_1}$ and a random polynomial $\bar{l''_i}$ with norms larger than β such that $\bar{c}_v\boldsymbol{k}_i = \boldsymbol{o}_i\bar{x_1} + \bar{l''_i}$. However $\bar{c}_v \in \bar{\mathcal{C}}$ is unknown to the verifier. On the other hand, to check that σ is not in SRL we need to measure the exact distance between the $\boldsymbol{k}_i$s and $\boldsymbol{d}_i$s, thus the verifier has to know $\bar{c}_v$. Therefore, we need the signer $\mathcal{M}$ to construct a proof that allows a verifier to check the distance even without knowing $\bar{c}_v$. We adopt the ZKP from [6] to let the signer prove that it actually knows small secrets $\bar{x_1}$ and $\bar{l''_i}$ with norms greater then β that satisfy the equation $2\boldsymbol{k}_i = 2\boldsymbol{o}_i\bar{x_1} + 2\bar{l''_i}$, i.e. the verifier ensures that the prover knows short secrets for twice $\boldsymbol{k}_i$ (same proof for $\boldsymbol{d}_i, \boldsymbol{o}_i$ applies).

Proof. We prove now that our modified SPK satisfies the following:

Lemma 2. *[4]: Let $\boldsymbol{a}$ and $\boldsymbol{b}$ be two polynomials in $\mathbb{Z}_q/(X^n+1)$. If a polynomial $\boldsymbol{b}$ is chosen randomly with mean 0, then with high probability $\|\boldsymbol{ab}\|_\infty \approx \sqrt{n}\|\boldsymbol{a}\|_\infty\|\boldsymbol{b}\|_\infty$.*

– Completeness: In fact the verifier can always verify the equation since:

$$X^{c_v}\boldsymbol{k}_i + \boldsymbol{t}_{k_i}$$

$$= X^{c_v}(\boldsymbol{o}_i\boldsymbol{x}_1 + \boldsymbol{l}''_i) + \boldsymbol{o}_i\boldsymbol{r}_{x_1} + \boldsymbol{r}_{l''_i}$$

$$= X^{c_v}(\boldsymbol{o}_i\boldsymbol{x}_1 + \boldsymbol{l}''_i) + \boldsymbol{o}_i(\boldsymbol{s}_{x_1} - x^{c_v}\boldsymbol{x}_1) + \boldsymbol{s}_{l''_i} - x^{c_v}\boldsymbol{l}''_i$$

$$= \boldsymbol{o}_i\boldsymbol{s}_{x_1} + \boldsymbol{s}_{l''_i}$$

For the norms we have that $\|\boldsymbol{s}_{x_1}\|_\infty \approx \beta + \sqrt{n}\beta$ with overwhelming probability using lemma 2, and the same applies for $\|\boldsymbol{s}_{l''_i}\|_\infty$.

- Honest-verifier zero-knowledge: Given a challenge c_v, a simulator $\mathcal{S}$ proceeds as follows: chooses $\boldsymbol{s}_{s_t}$ and $\boldsymbol{s}_{l_i''}$ randomly from $\mathcal{D}_{s'}$, computes $\boldsymbol{t}_{k_i} = \boldsymbol{o}_i \boldsymbol{s}_{s_t} + \boldsymbol{s}_{l_i''} - X^{c_v} \boldsymbol{k}_i$. Finally $\mathcal{S}$ outputs $(c_v, \boldsymbol{t}_{k_i}, \boldsymbol{k}_i, \boldsymbol{s}_{s_t}, \boldsymbol{s}_{l_i''})$. If no abort occurs, then the distribution of $(\boldsymbol{s}_{x_1}, \boldsymbol{s}_{l_i''})$ does not depend on $(\boldsymbol{x}_1, \boldsymbol{l}_i'')$, thus the simulated and real protocol transcripts are indistinguishable.

- Special soundness: Given two accepted transcripts $(c_v^1, \boldsymbol{s}_{x_1}^1, \boldsymbol{s}_{l_i''}^1)$ and $(c_v^2, \boldsymbol{s}_{x_1}^2, \boldsymbol{s}_{l_i''}^2)$, that share the same $\boldsymbol{t}_{k_i}$. We also proved the verification equality $X^{c_v}\boldsymbol{k}_i + \boldsymbol{t}_{k_i} = \boldsymbol{o}_i \boldsymbol{s}_{x_1} + \boldsymbol{s}_{l_i''}$. Therefore,
we have:

$$X^{c_v^1}\boldsymbol{k}_i + \boldsymbol{t}_{k_i} = \boldsymbol{o}_i \boldsymbol{s}_{x_1}^1 + \boldsymbol{s}_{l_i''}^1 \tag{4}$$

and

$$X^{c_v^2}\boldsymbol{k}_i + \boldsymbol{t}_{k_i} = \boldsymbol{o}_i \boldsymbol{s}_{x_1}^2 + \boldsymbol{s}_{l_i''}^2 \tag{5}$$

Subtracting the two above equations we get:

$$\boldsymbol{k}_i(X^{c_v^1} - X^{c_v^2}) = \boldsymbol{o}_i(\boldsymbol{s}_{x_1}^1 - \boldsymbol{s}_{x_1}^2) + (\boldsymbol{s}_{l_i''}^1 - \boldsymbol{s}_{l_i''}^2)$$

dividing both sides by $(X^{c_v^1} - X^{c_v^2})$ and multiplying by 2 we get:

$$2\boldsymbol{k}_i = o_i \frac{2(\boldsymbol{s}_{x_1}^1 - \boldsymbol{s}_{\boldsymbol{x}_1}^2)}{(X^{c_v^1} - X^{c_v^2})} + \frac{2(\boldsymbol{s}_{l_i''}^1 - \boldsymbol{s}_{l_i''}^2)}{(X^{c_v^1} - X^{c_v^2})}$$

The equation can be written in the form:

$$2\boldsymbol{k}_i = \boldsymbol{o}_i \bar{x_1} + \bar{l_i''} \tag{6}$$

where $\bar{x_1} = 2\frac{(\boldsymbol{s}_{x_1}^1 - \boldsymbol{s}_{x_1}^2)}{(X^{c_v^1} - X^{c_v^2})}$ and $\bar{l_i''} = 2\frac{(\boldsymbol{s}_{l_i''}^1 - \boldsymbol{s}_{l_i''}^2)}{(X^{c_v^1} - X^{c_v^2})}$.
Thus, relying on lemmas 1 and 2, the infinite norm of the extracted witness $\bar{x_1}$ is as follows:

$$\|\bar{x_1}\|_\infty \leq \sqrt{n}\|(\boldsymbol{s}_{x_1}^1 - \boldsymbol{s}_{x_1}^2)\|_\infty \|2(X^{c_v^1} - X^{c_v^2})^{-1}\|_\infty \leq 2\sqrt{n}(\beta + \sqrt{n}\beta)$$

Similarly, for the extracted $\bar{l_i''}$, we have

$$\|\bar{l_i''}\|_\infty \leq \sqrt{n}\|(\boldsymbol{s}_{l_i''}^1 - \boldsymbol{s}_{l_i''}^2)\|_\infty \|2(X^{c_v^1} - X^{c_v^2})^{-1}\|_\infty \leq 2\sqrt{n}(\beta + \sqrt{n}\beta)$$

since $\|(\boldsymbol{s}_{x_1}^1 - \boldsymbol{s}_{x_1}^2)\|_\infty \leq 2(\beta + \sqrt{n}\beta)$ and $\|2(X^{c_v^1} - X^{c_v^2})^{-1}\|_\infty = 1$ from lemma 1.

B Detailed Security Proof of the LEPID Scheme

- **S**ETUP

  On input (SETUP, sid) from $\mathcal{I}$, output (FORWARD, (SETUP, $sid, \mathcal{I}$) to $\mathcal{S}$.
- **J**OIN
  1. On input (JOIN, $sid, jsid$) from the platform $\mathcal{M}_i$, output (FORWARD, (JOIN, $sid, jsid, \mathcal{M}_i$)) to $\mathcal{S}$.
  2. On input (JOINPROCEED,$sid, jsid$) from $\mathcal{I}$, output (FORWARD, (JOINPROCEED,$sid, jsid$), $\mathcal{I}$) to $\mathcal{S}$.
- **S**IGN
  1. On input (SIGN,$sid, ssid, \boldsymbol{p}$) from $\mathcal{M}_i$, output (FORWARD, (SIGN,$sid, ssid, \mathcal{M}_i, \boldsymbol{p}$)) to $\mathcal{S}$.
  2. On input (SIGNPROCEED,$sid, ssid$) from $\mathcal{M}_i$, output (FORWARD, (SIGN-PROCEED, $sid, ssid$), $\mathcal{M}_i$) to $\mathcal{S}$.
- **V**ERIFY

  On input (VERIFY, $sid, \mu, \boldsymbol{p}, \sigma, \texttt{KRL}, \texttt{SRL}$) from V, output (FORWARD, (VERIFY, $sid, \mu, \boldsymbol{p}, \sigma, \texttt{KRL}, \texttt{SRL}$), V) to $\mathcal{S}$.
- **R**EVOKE: On the input tsk^* from a party R, output (FORWARD, (REVOKE, $sid, \mu, tsk^*, \texttt{KRL}, \texttt{SRL}$), R) to $\mathcal{S}$.
  On the input $(\sigma^*, \mu^*, \boldsymbol{p}^*)$ from a party R, (FORWARD, (REVOKE, $sid, \mu^*, \boldsymbol{p}^*, \sigma^*, \texttt{KRL}, \texttt{SRL}$), R) to $\mathcal{S}$.
- **O**UTPUT

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

Fig. 1: Game 3 for $\mathcal{F}$

- **K**eyGen
  Upon receiving input (FORWARD, (SETUP, $sid, \mathcal{I}$)from $\mathcal{F}$, give "$\mathcal{I}$" (SETUP, sid) .
- **J**OIN

  1. Upon receiving (FORWARD, (JOIN, $sid, jsid, \mathcal{M}_i$) from $\mathcal{F}$, give input (JOIN, $sid, jsid$) to "$\mathcal{M}_i$"
  2. Upon receiving input (FORWARD, (JOINPROCEED,$sid, jsid$), $\mathcal{I}$) from $\mathcal{F}$, give "$\mathcal{I}$" input (JOINPROCEED,$sid, jsid$).
- **S**IGN

  1. Upon receiving input (FORWARD, (SIGN, $sid, ssid, \mathcal{M}_i, \boldsymbol{p}$) from $\mathcal{F}$, give "$\mathcal{M}_i$" input (SIGN, $sid, ssid$, $\boldsymbol{p}$).
  2. Upon receiving input (FORWARD, (SIGNPROCEED,$sid, ssid$), $\mathcal{M}_i$) from $\mathcal{F}$, give "$\mathcal{M}_i$" input (SIGNPROCEED,$sid, ssid$).

- **VERIFY**

  Upon receiving input (FORWARD, (VERIFY, $sid, \mu, \boldsymbol{p}, \sigma, \mathtt{KRL}, \mathtt{SRL}$), V) from $\mathcal{F}$, give "V" input (VERIFY, $sid, \mu, \boldsymbol{p}, \sigma, \mathtt{KRL}, \mathtt{SRL}$).
- **REVOKE**

  Upon receiving (FORWARD, (REVOKE, $sid, \mu, tsk^*, \mathtt{KRL}, \mathtt{SRL}$), R) from $\mathcal{F}$, give "R" an input (REVOKE, $sid, \mu, tsk^*, \mathtt{KRL}, \mathtt{SRL}$).
  Upon receiving (FORWARD, (REVOKE, $sid, \mu^*, \boldsymbol{p}^*, \sigma^*, \mathtt{KRL}, \mathtt{SRL}$), R) from $\mathcal{F}$, give "R" an input (REVOKE, $sid, \mu^*, \boldsymbol{p}^*, \sigma^*, \mathtt{KRL}, \mathtt{SRL}$).
- **OUTPUT**

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

Fig. 2: Game 3 for $\mathcal{S}$

- **SETUP**

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

  1. On input (JOIN, $sid, jsid$) from the platform $\mathcal{M}_i$, output (FORWARD, (JOIN, $sid, jsid, \mathcal{M}_i$)) to $\mathcal{S}$.
  2. On input (JOINPROCEED,$sid, jsid$) from $\mathcal{I}$, output (FORWARD, (JOINPROCEED,$sid, jsid$), $\mathcal{I}$) to $\mathcal{S}$.
- **SIGN**

  1. On input (SIGN, $sid, ssid, \boldsymbol{p}$) from $\mathcal{M}_i$, output (FORWARD, (SIGN, $sid, ssid, \mathcal{M}_i, \boldsymbol{p}$)) to $\mathcal{S}$.
  2. On input (SIGNPROCEED,$sid, ssid$) from $\mathcal{M}_i$, output (FORWARD, (SIGNPROCEED,$sid, ssid$), $\mathcal{M}_i$) to $\mathcal{S}$.
- **VERIFY**

  On input (VERIFY, $sid, \mu, \boldsymbol{p}, \sigma, \mathtt{KRL}, \mathtt{SRL}$) from V, output (FORWARD, (VERIFY, $sid, \mu, \boldsymbol{p}, \sigma, \mathtt{KRL}, \mathtt{SRL}$), V) to $\mathcal{S}$.

– **R**EVOKE

  On the input tsk^* from a party R, output (FORWARD, (REVOKE, $sid, \mu, tsk^*, \texttt{KRL}$), R) to $\mathcal{S}$.
  On the input $(\sigma^*, \mu^*, \boldsymbol{p}^*, \texttt{KRL}, \texttt{SRL})$ from a party R, (FORWARD, (REVOKE, $sid, \mu^*, \boldsymbol{p}^*, \sigma^*, \texttt{KRL}, \texttt{SRL}$), R) to $\mathcal{S}$.

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

Fig. 3: Game 4 for $\mathcal{F}$

– **K**eyGen: Honest $\mathcal{I}$: On input (SETUP, sid) from $\mathcal{F}$
  - Check $sid = (\mathcal{I}, sid')$, output $\perp$ to $\mathcal{I}$ if the check fails.
  - Give "$\mathcal{I}$" input (SETUP, sid).
  - Upon receiving output (SETUPDONE, sid) from "$\mathcal{I}$", $\mathcal{S}$ takes its private key $\hat{T_{\mathcal{I}}}$.
  - Define Sign$(tsk, \mu, \boldsymbol{p}, \texttt{KRL}, \texttt{SRL})$ as follows:
    Define SamplePre$(\hat{T_{\mathcal{I}}}, \boldsymbol{u}_t, \boldsymbol{u}, \mathsf{id})$ that outputs a signature $\hat{X}_h = [\hat{X}_{h_1}|\hat{X}_{h_2}]$ as satisfying
    $$\hat{A}_h \cdot \hat{X}_h = \boldsymbol{u}_h \mod q,$$
    with $\|\hat{X}_{h_1}\|_\infty \leq \beta/2$ and $\|\hat{X}_{h_2}\|_\infty \leq \beta$, and id is a fresh tag will be the L-EPID credential.
    * $\mathsf{nym} = \boldsymbol{p} \cdot tsk + \boldsymbol{e} \mod q$, for an error term $\boldsymbol{e} \leftarrow \mathcal{D}_s$ such that $\|\boldsymbol{e}\|_\infty < \beta$.
    * $\forall(\sigma_i^*, \boldsymbol{p}_i^*, \mathsf{nym}_i^*) \in \texttt{SRL}$
      · $\boldsymbol{q}_i, \boldsymbol{l}_i', \boldsymbol{l}_i'', \boldsymbol{l}_i''' \leftarrow \mathcal{D}_s$
      · $\boldsymbol{o}_i = \boldsymbol{p}_i^* \boldsymbol{q}_i + \boldsymbol{l}_i'$
      · $\boldsymbol{k}_i = \boldsymbol{o}_i \boldsymbol{x}_1 + \boldsymbol{l}_i''$
      · $\boldsymbol{d}_i = \mathsf{nym}_i^* \boldsymbol{q}_i + \boldsymbol{l}_i'''$
      · $\boldsymbol{r}_{x_1}, \boldsymbol{r}_e, \boldsymbol{r}_{q_i}, \boldsymbol{r}_{l_i'}, \boldsymbol{r}_{l_i''}, \boldsymbol{r}_{l_i'''} \leftarrow \mathcal{D}_s$
      · $\boldsymbol{t}_{\mathsf{nym}} = \boldsymbol{p}\boldsymbol{r}_{x_1} + \boldsymbol{r}_e$
      · $\boldsymbol{t}_{o_i} = \boldsymbol{p}_i^* \boldsymbol{r}_{q_i} + \boldsymbol{r}_{l_i'}$.
      · $\boldsymbol{t}_{k_i} = \boldsymbol{o}_i \boldsymbol{r}_{x_1} + \boldsymbol{r}_{l_i''}$.
      · $\boldsymbol{t}_{d_i} = \mathsf{nym}_i^* \boldsymbol{r}_{q_i} + \boldsymbol{r}_{l_i'''}$.
      · $c_v = H(\boldsymbol{t}_{\mathsf{nym}}|\boldsymbol{t}_{o_i}|\boldsymbol{t}_{k_i}|\boldsymbol{t}_{d_i}|\mu) \in \{0, 1, 2, \cdots, 2n-1\}$.
      · $\boldsymbol{s}_{x_1} = \boldsymbol{r}_{x_1} + x^{c_v}\boldsymbol{x}_1$
      · $\boldsymbol{s}_e = \boldsymbol{r}_e + x^{c_v}\boldsymbol{e}$
      · $\boldsymbol{s}_{q_i} = \boldsymbol{r}_{q_i} + x^{c_v}\boldsymbol{q}_i$
      · $\boldsymbol{s}_{l_i'} = \boldsymbol{r}_{l_i'} + x^{c_v}\boldsymbol{l}_i'$
      · $\boldsymbol{s}_{l_i''} = \boldsymbol{r}_{l_i''} + x^{c_v}\boldsymbol{l}_i''$
      · $\boldsymbol{s}_{l_i'''} = \boldsymbol{r}_{l_i'''} + x^{c_v}\boldsymbol{l}_i'''$
    * $\texttt{rej}(\boldsymbol{s}_{x_1}, x^{c_v}\boldsymbol{x}_1, \xi)$, $\texttt{rej}(\boldsymbol{s}_e, x^{c_v}\boldsymbol{e}, \xi)$, $\texttt{rej}(\boldsymbol{s}_{q_i}, x^{c_v}\boldsymbol{q}_i, \xi)$, $\texttt{rej}(\boldsymbol{s}_{l_i'}, x^{c_v}\boldsymbol{l}_i', \xi)$, $\texttt{rej}(\boldsymbol{s}_{l_i''}, x^{c_v}\boldsymbol{l}_i'', \xi)$ or $\texttt{rej}(\boldsymbol{s}_{l_i'''}, x^{c_v}\boldsymbol{l}_i''', \xi)$.

  - $*$ $\sigma = (\ \pi,\ \mathsf{nym},\ \boldsymbol{o}_i,\ \boldsymbol{k}_i,\ \boldsymbol{d}_i,\ \boldsymbol{s}_{x_1},\ \boldsymbol{s}_e,\ \boldsymbol{s}_{q_i},\ \boldsymbol{s}_{l'_i},\ \boldsymbol{s}_{l''_i},\ \boldsymbol{s}_{l'''_i},\ c_v,\ \mathtt{KRL},\ \mathtt{SRL})$
- Define ver$(\sigma, \mu, \boldsymbol{p}, \mathtt{KRL}, \mathtt{SRL})$ as follows:
  - $*$ $\forall$ $tsk^* \in \mathtt{KRL}$, if $\|\boldsymbol{p} \cdot tsk^* - \mathsf{nym}\|_\infty \leq \beta$ outputs 0.
    1. $\forall$ $\sigma_i^* = (\pi_{\mathsf{nym}_i^*}, \mathsf{nym}_i^*, \boldsymbol{p}_i^*) \in \mathtt{SRL}$,
    2. compute:
    3. $\boldsymbol{t'}_{k_i}{=}\boldsymbol{o}_i\boldsymbol{s}_{x_1} + \boldsymbol{s}_{l''_i} - x^{c_v}\boldsymbol{k}_i$
    4. $\boldsymbol{t'}_{d_i}{=}\mathsf{nym}_i^*\boldsymbol{s}_{q_i} + \boldsymbol{s}_{l'''_i} - x^{c_v}\boldsymbol{d}_i$
    5. $\boldsymbol{t'}_{o_i}{=}\boldsymbol{p}_i^*\boldsymbol{s}_{q_i} + \boldsymbol{s}_{l'_i} - x^{c_v}\boldsymbol{o}_i$
    6. $\boldsymbol{t'}_{\mathsf{nym}}{=}\boldsymbol{p}\boldsymbol{s}_{x_1} + \boldsymbol{s}_e - x^{c_v}\mathsf{nym}$
    7. $c_v \stackrel{?}{=} H(\boldsymbol{t'}_{\mathsf{nym}}|\boldsymbol{t'}_{o_i}|\boldsymbol{t'}_{k_i}|\boldsymbol{t'}_{d_i}|\mu)$.
    8. check $\|\boldsymbol{s}_{\boldsymbol{x}_1}\|_\infty, \|\boldsymbol{s}_e\|_\infty, \|\boldsymbol{s}_{q_i}\|_\infty, \|\boldsymbol{s}_{l'_i}\|_\infty, \|\boldsymbol{s}_{l''_i}\|_\infty, \|\boldsymbol{s}_{l'''_i}\|_\infty \leq \beta + \sqrt{n}\beta$.
    9. check $2\|\boldsymbol{d}_i - \boldsymbol{k}_i\| < \mathit{\Gamma}$, where $\mathit{\Gamma}$ is a function of β. If $2\|\boldsymbol{d}_i - \boldsymbol{k}_i\| < \mathit{\Gamma}$ the verifier outputs 0, otherwise 1.
  - $*$ If all checks pass, output $(\mathsf{VERIFIED}, \mathit{sid}, 1)$, $(\mathsf{VERIFIED}, \mathit{sid}, 0)$ otherwise.
- Define revoke$(\ tsk^*, \sigma^*, \mu^*, \boldsymbol{p}^*)$, add tsk^* to $\mathtt{KRL}$ or σ^* to $\mathtt{SRL}$ after verifying σ^*.
- Define Identify$(\sigma, \mu, \boldsymbol{p}, tsk)$ as follows: It parses σ as $(nym, \boldsymbol{p})$ and checks that $tsk \in \mathcal{D}_s$, ver$(\sigma, \mu, \boldsymbol{p})$=1 and $\|nym - \boldsymbol{p} \cdot tsk\|_\infty \leq \beta$. If so output 1, otherwise output 0.
- Define Kgen, take $tsk \in \mathcal{D}_s$ and output tsk.
- $\mathcal{S}$ sends (KEYS, *sid*, Sign, Verify, Revoke, Identify, Kgen) to $\mathcal{F}$.

Corrupt I: $\mathcal{S}$ notices this setup as it notices $\mathcal{I}$ registering a public key with $\mathcal{F}_{CA}$ with $\mathit{sid} = (\mathcal{I}, \mathit{sid}')$.
- If the registered key is in the form $(\hat{A}_{\mathcal{I}}, \pi_{\mathcal{I}})$ and $\pi_{\mathcal{I}}$ is valid, then $\mathcal{S}$ extracts $\hat{T}_{\mathcal{I}}$ from $\pi_{\mathcal{I}}$.
- $\mathcal{S}$ defines the algorithms Sign, Verify, Revoke, and Identify as before, but now depending on the extracted key. $\mathcal{S}$ sends (SETUP, *sid*) to $\mathcal{F}$ on behalf of $\mathcal{I}$. On input (KEYGEN, *sid*) from $\mathcal{F}$, $\mathcal{S}$ sends (KEYS, *sid*, Sign, Verify, Revoke, Identify, Kgen) to $\mathcal{F}$.
- On input (SETUPDONE, *sid*) from $\mathcal{F}$. $\mathcal{S}$ continues simulating "$\mathcal{I}$".

– **J**OIN, **S**IGN, **V**ERIFY, **R**EVOKE: Unchanged.

Fig. 4: Game 4 for $\mathcal{S}$

– **S**ETUP
  1. On input (SETUP, *sid*) from $\mathcal{I}$, verify that $\mathit{sid} = (\mathcal{I}, \mathit{sid}')$ and output (SETUP, *sid*) to $\mathcal{S}$.
  2. On input (ALGORITHMS, *sid*, sign, ver, revoke, identify, Kgen) from $\mathcal{S}$, check that ver, revoke, and identify are deterministic. Store (*sid*, sign, ver, revoke, identify, Kgen) and output (SETUPDONE, *sid*) to $\mathcal{I}$.

– **J**OIN

1. On input (JOIN, $sid, jsid$) from the platform $\mathcal{M}_i$, output (FORWARD, (JOIN, $sid, jsid, \mathcal{M}_i$)) to $\mathcal{S}$.
2. On input (JOINPROCEED,$sid, jsid$) from $\mathcal{I}$, output (FORWARD, (JOINPROCEED,$sid, jsid$), $\mathcal{I}$) to $\mathcal{S}$.

– **S**IGN

1. On input (SIGN, $sid, ssid, \boldsymbol{p}$) from $\mathcal{M}_i$, output (FORWARD, (SIGN, $sid, ssid, \mathcal{M}_i, \boldsymbol{p}$)) to $\mathcal{S}$.
2. On input (SIGNPROCEED,$sid, ssid$) from $\mathcal{M}_i$, output (FORWARD, (SIGNPROCEED,$sid, ssid$), $\mathcal{M}_i$) to $\mathcal{S}$.

– **V**ERIFY
On input (VERIFY, $sid, \mu, \boldsymbol{p}, \sigma, \mathtt{KRL}, \mathtt{SRL}$) from V
  - Set $f = 0$ if
    - ∗ There is a $tsk^* \in \mathtt{KRL}$ such that Identify$(\sigma, \mu, \boldsymbol{p}, tsk^*) = 1$
  - If $f \neq 0$, set f=Verify$(\sigma, \mu, \boldsymbol{p})$.
  - Add $(\sigma, \mu, \boldsymbol{p}, \mathtt{KRL}, f)$ to VerResults, output (VERIFIED, sid, f) to V.

– **R**EVOKE
On input (REVOKE, $tsk^*, \sigma^*, \mu^*, \boldsymbol{p}^*$), the revocation manager adds tsk^* to $\mathtt{KRL}$ or σ^* to $\mathtt{SRL}$ after verifying σ^*.

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

Fig. 5: Game 5 for $\mathcal{F}$

– **K**eyGen, **J**OIN, **S**IGN : Unchanged.
– **V**ERIFY, **R**EVOKE: Nothing to simulate.

Fig. 6: Game 5 for $\mathcal{S}$

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

– **J**OIN
1. JOINREQUEST: On input (JOIN, $sid, jsid$) from $\mathcal{M}_i$
   - Create a join session $\langle jsid, \mathcal{M}_i$, request $\rangle$.
   - Output (JOINSTART, $sid, jsid, \mathcal{M}_i$) to $\mathcal{S}$.

2. JOIN REQUEST DELIVERY: Proceed upon receiving delivery notification from $\mathcal{S}$.
   - Update the session record to $\langle jsid, \mathcal{M}_i$, delivered$\rangle$.
   - If $\mathcal{I}$ or $\mathcal{M}_i$ is honest and $\langle \mathcal{M}_i, \star \rangle$ is already in Members, output $\bot$.
   - Output (JOINPROCEED,$sid, jsid, \mathcal{M}_i$) to $\mathcal{I}$.
3. JOIN PROCEED: Upon receiving (JOINPROCEED,$sid, jsid, \mathcal{M}_i$) from $\mathcal{I}$
   - Update the session record to $\langle jsid, sid, \mathcal{M}_i$, complete$\rangle$.
   - Output (JOINCOMPLETE, $sid, jsid$) to $\mathcal{S}$.
4. KEY GENERATION: On input (JOINCOMPLETE,$sid, jsid, tsk$) from $\mathcal{S}$.
   - Update the session record to $\langle jsid, \mathcal{M}_i$, complete$\rangle$
   - If $\mathcal{M}_i$ is honest, set $tsk = \bot$.
   - Insert $\langle \mathcal{M}_i, tsk \rangle$ into Members, and output (JOINED, $sid, jsid$) to $\mathcal{M}_i$.

- **S**IGN

  1. On input (SIGN, $sid, ssid, \boldsymbol{p}$) from $\mathcal{M}_i$, output (FORWARD, (SIGN, $sid, ssid, \mathcal{M}_i, \boldsymbol{p}$)) to $\mathcal{S}$.
  2. On input (SIGNPROCEED,$sid, ssid$) from $\mathcal{M}_i$, output (FORWARD, (SIGNPROCEED,$sid, ssid$), $\mathcal{M}_i$) to $\mathcal{S}$.
- **V**ERIFY
  On input (VERIFY, $sid, \mu, \boldsymbol{p}, \sigma, \mathtt{KRL}, \mathtt{SRL}$) from V
  - Set $f = 0$ if
    - $*$ There is a $tsk^* \in \mathtt{KRL}$ such that Identify$(\sigma, \mu, \boldsymbol{p}, tsk^*) = 1$
  - If $f \neq 0$, set f=Verify$(\sigma, \mu, \boldsymbol{p}, \mathtt{KRL}, \mathtt{SRL})$.
  - Add $(\sigma, \mu, \boldsymbol{p}, \mathtt{KRL}, f)$ to VerResults, output (VERIFIED, sid, f) to V.
- **R**EVOKE
  On input (REVOKE, $tsk^*, \sigma^*, \mu^*, \boldsymbol{p}^*$), the revocation manager adds tsk^* to $\mathtt{KRL}$ or σ^* to $\mathtt{SRL}$ after verifying σ^*.
- **O**UTPUT
  On input (OUTPUT, P, μ) from $\mathcal{S}$, output μ to P.

Fig. 7: Game 6 for $\mathcal{F}$

- **K**eyGen: Unchanged.
  JOIN: **H**onest $\mathcal{M}_i$, $\mathcal{I}$
  - When $\mathcal{S}$ receives (JOINSTART, $sid, jsid, \mathcal{M}_i$) from $\mathcal{F}$
  - It simulates the real world protocol by giving "$\mathcal{M}_i$" input (JOIN, $sid, jsid$) and waits for output (JOINPROCEED, $sid, jsid, \mathcal{M}_i$) from "$\mathcal{I}$".
  - As $\mathcal{M}_i$ is honest, $\mathcal{S}$ already knows tsk as it is simulating $\mathcal{M}_i$.
  - $\mathcal{S}$ sends (JOINSTART, $sid, jsid$) to $\mathcal{F}$.
  - Upon receiving input (JOINCOMPLETE, $sid, jsid$) from $\mathcal{F}$, $\mathcal{S}$ gives "$\mathcal{I}$" input (JOINPROCEED, $sid, jsid$) and waits for output (JOINED, $sid, jsid$) from "$\mathcal{M}_i$".

- Upon receiving input (JOINCOMPLETE, $sid, jsid$) from $\mathcal{F}$, $\mathcal{S}$ sends (JOIN-COMPLETE, $sid, jsid, \perp$) to $\mathcal{F}$.

Honest $\mathcal{I}$, Corrupt $\mathcal{M}_i$:
- $\mathcal{S}$ notices this join as "$\mathcal{I}$" receives (SENT, $sid', (\boldsymbol{u}_t, \pi_{\boldsymbol{u}_t})$) from $\mathcal{F}^*_{\mathsf{auth}}$.
- $\mathcal{S}$ doesn't know the identity of the signer that started this join, so $\mathcal{S}$ chooses any corrupt $\mathcal{M}^*$ and proceeds as if this signer initiated this join, although this may not be the correct signer. This makes no difference as when creating signatures we only look for corrupt signers as they are not considered in generating signatures.
- $\mathcal{S}$ then extracts tsk from the proof $\pi_{\boldsymbol{u}_t}$.
- $\mathcal{S}$ makes a join query with $\mathcal{M}^*$ by sending (JOIN, $sid, jsid, \mathcal{M}^*$) to $\mathcal{F}$.
- Upon receiving input (JOINSTART, $sid, jsid, \mathcal{M}^*$) from $\mathcal{F}$, $\mathcal{S}$ continues simulating "$\mathcal{I}$" until it outputs (JOINPROCEED, $sid, jsid, \mathcal{M}^*$).
- $\mathcal{S}$ sends (JOINSTART, $sid, jsid$) to $\mathcal{F}$.
- Upon receiving (JOINCOMPLETE, $sid, jsid$) from $\mathcal{F}$, $\mathcal{S}$ sends (JOINCOMPLETE, $sid, jsid, tsk$) to $\mathcal{F}$.
- Upon receiving (JOINED, $sid, jsid$) from $\mathcal{F}$ as $\mathcal{M}_i$ is corrupt, $\mathcal{S}$ gives "$\mathcal{I}$" input (JOINPROCEED, $sid, jsid$).

Honest $\mathcal{M}_i$, Corrupt $\mathcal{I}$:
- On input (JOINSTART, $sid, jsid, \mathcal{M}_i$) from $\mathcal{F}$, $\mathcal{S}$ gives "$\mathcal{M}_i$" input (JOIN, $sid, jsid$) and waits for output (JOINED, $sid, jsid, \mathcal{M}_i$) from "$\mathcal{M}_i$".
- $\mathcal{S}$ sends (JOINSTART, $sid, jsid$) to $\mathcal{F}$.
- Upon receiving input (JOINPROCEED, sid, $jsid$) from $\mathcal{F}$, $\mathcal{S}$ sends (JOIN-PROCEED, $sid, jsid$) to $\mathcal{F}$ on behalf of $\mathcal{I}$.
- Upon receiving input (JOINCOMPLETE, $sid, jsid$) from $\mathcal{F}$, $\mathcal{S}$ sends (JOIN-COMPLETE, $sid, jsid, \perp$) to $\mathcal{F}$.

– **S**IGN, **V**ERIFY, **L**INK: Unchanged.

Fig. 8: Game 6 for $\mathcal{S}$

– **S**ETUP, **J**OIN: Unchanged.
– **S**IGN
- SIGN REQUEST: On input (SIGN, $sid, ssid, \mu, \boldsymbol{p}$) from $\mathcal{M}_i$,
  - ∗ Create a sign session $\langle ssid, \mathcal{M}_i, \mu, \boldsymbol{p},\ request\rangle$.
  - ∗ Output (SIGNSTART, $sid, ssid, \mathcal{M}_i$) to $\mathcal{S}$.
- SIGN REQUEST DELIVERY: On input (SIGNSTART, $sid, ssid$) from $\mathcal{S}$, update the session to $\langle ssid, \mathcal{M}_i, \mu, \boldsymbol{p},\ delivered\rangle$.
- Output (SIGNPROCEED, $sid, ssid, \mu, \boldsymbol{p}$) to $\mathcal{M}_i$.
- SIGN PROCEED: On input (SIGNPROCEED, $sid, ssid$) from $\mathcal{M}_i$
  - ∗ Update the records $\langle ssid, \mathcal{M}_i, \mu, \boldsymbol{p}$, delivered$\rangle$.
  - ∗ Output (SIGNCOMPLETE, $sid, ssid$) to $\mathcal{S}$.
- SIGNATURE GENERATION: On the input (SIGNCOMPLETE, $sid, ssid, \sigma$) from $\mathcal{S}$, if $\mathcal{M}_i$ is honest then:

      * Ignore the adversary's signature σ.
      * Generate the signature $\sigma \leftarrow Sign(tsk, \mu, \boldsymbol{p})$.
      * For all $(\sigma^*, \mu^*, \boldsymbol{p}^*) \in \texttt{SRL}$, find all $(tsk^*, \mathcal{M}^*)$ from Members and DomainKeys such that identify$(\sigma^*, \mu^*, \boldsymbol{p}^*, \star, tsk^*) = 1$
        · Check that no two distinct keys tsk^* trace back to σ^*.
        · Check that no pair $(tsk^*, \mathcal{M}_i)$ was found.
      * If $\mathcal{M}_i$ is honest, then store $\langle \sigma, \mu, \mathcal{M}_i, \boldsymbol{p} \rangle$ in Signed and output (SIGNATURE, $sid, ssid, \sigma$) to $\mathcal{M}_i$.

– **V**ERIFY
  On input (VERIFY, $sid, \mu, \boldsymbol{p}, \sigma, \texttt{KRL}, \texttt{SRL}$) from V
  - Set $f = 0$ if
      * There is a $tsk^* \in \texttt{KRL}$ such that Identify$(\sigma, \mu, \boldsymbol{p}, tsk^*) = 1$
  - If $f \neq 0$, set f=Verify$(\sigma, \mu, \boldsymbol{p})$.
  - Add $(\sigma, \mu, \boldsymbol{p}, \texttt{KRL}, f)$ to VerResults, output (VERIFIED, sid, f) to V.

– **R**EVOKE
  On input (REVOKE, $tsk^*, \sigma^*, \mu^*, \boldsymbol{p}^*$), the revocation manager adds tsk^* to $\texttt{KRL}$ or σ^* to $\texttt{SRL}$ after verifying σ^*.

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

Fig. 9: Game 7 for $\mathcal{F}$

– **K**eyGen, **J**OIN: Unchanged.

– **S**IGN :**H**onest $\mathcal{M}_i$
  Upon receiving (SIGNSTART, $sid, ssid, \mathcal{M}_i, \boldsymbol{p}, \mu$) from $\mathcal{F}$.
  - $\mathcal{S}$ starts the simulation by giving "$\mathcal{M}_i$" input (SIGN, $sid, ssid,, \mu, \boldsymbol{p}$).
  - When "$\mathcal{M}_i$" outputs (SIGNPROCEED, $sid, ssid, \mu, \boldsymbol{p}$), $\mathcal{S}$ sends (SIGNSTART, $sid, ssid$) to $\mathcal{F}$.
  - Upon receiving (SIGNCOMPLETE, $sid, ssid$) from $\mathcal{F}$, output (SIGNPROCEED, $sid, ssid$) to "$\mathcal{M}_i$".
  - When "$\mathcal{M}_i$" outputs (SIGNATURE, $sid, ssid, \sigma$), send (SIGNCOMPLETE, $sid, ssid, \perp$) to $\mathcal{F}$.

  Corrupt $\mathcal{M}_i$
  Upon receiving (SIGNSTART, $sid, ssid, \mathcal{M}_i, \boldsymbol{p}, \mu$) from $\mathcal{F}$, send (SIGNSTART, $sid, ssid$) to $\mathcal{F}$.
  - Upon receiving (SIGNPROCEED, $sid, ssid, \mu, \boldsymbol{p}$) from $\mathcal{F}$ on behalf of $\mathcal{M}_i$, as $\mathcal{M}_i$ is corrupt, $\mathcal{S}$ sends (SIGN, $sid, ssid, \mathcal{M}_i, \mu, \boldsymbol{p}$) to $\mathcal{F}$ on behalf of $\mathcal{M}_i$.
  - Upon receiving (SIGNCOMPLETE, $sid, ssid$) from $\mathcal{F}$, $\mathcal{S}$ sends (SIGNCOMPLETE, $sid, ssid, \sigma$) to $\mathcal{F}$.

– **V**ERIFY, **L**INK: Nothing to simulate.

Fig. 10: Game 7 for $\mathcal{S}$

- **S**ETUP, **J**OIN: Unchanged.
- **S**IGN
  - SIGN REQUEST: On input (SIGN, $sid, ssid, \mu, \boldsymbol{p}$) from $\mathcal{M}_i$,
    - Create a sign session $\langle ssid, \mathcal{M}_i, \mu, \boldsymbol{p},\ request\rangle$.
    - Output (SIGNSTART, $sid, ssid, \mathcal{M}_i,\ l(\mu.bsn)$) to $\mathcal{S}$.
  - SIGN REQUEST DELIVERY: On input (SIGNSTART, $sid, ssid$) from $\mathcal{S}$, update the session to $\langle ssid, \mathcal{M}_i, \mu, \boldsymbol{p},\ delivered\rangle$.
  - Output (SIGNPROCEED, $sid, ssid, \mu, \boldsymbol{p}$) to $\mathcal{M}_i$.
  - SIGN PROCEED: On input (SIGNPROCEED, $sid, ssid$) from $\mathcal{M}_i$
    - Update the records $\langle ssid, \mathcal{M}_i, \mu, \boldsymbol{p}$, delivered$\rangle$.
    - Output (SIGNCOMPLETE, $sid, ssid$) to $\mathcal{S}$.
  - SIGNATURE GENERATION: On the input (SIGNCOMPLETE, $sid, ssid, \sigma$) from $\mathcal{S}$, if $\mathcal{M}_i$ is honest then:
    - Ignore the adversary's signature σ.
    - Generate the signature $\sigma \leftarrow Sign(tsk, \mu, \boldsymbol{p})$.
    - For all $(\sigma^*, \mu^*, \boldsymbol{p}^*) \in \texttt{SRL}$, find all $(tsk^*, \mathcal{M}^*)$ from Members and DomainKeys such that identify$(\sigma^*, \mu^*, \boldsymbol{p}^*, \star, tsk^*) = 1$
      - · Check that no two distinct keys tsk^* trace back to σ^*.
      - · Check that no pair $(tsk^*, \mathcal{M}_i)$ was found.
    - If $\mathcal{M}_i$ is honest, then store $\langle \sigma, \mu, \mathcal{M}_i, \boldsymbol{p}\rangle$ in Signed and output (SIGNATURE, $sid, ssid, \sigma$) to $\mathcal{M}_i$.
- **V**ERIFY
  On input (VERIFY, $sid, \mu, \boldsymbol{p}, \sigma, \texttt{KRL}, \texttt{SRL}$) from V
  - Set $f = 0$ if
    - There is a $tsk^* \in \texttt{KRL}$ such that Identify$(\sigma, \mu, \boldsymbol{p}, tsk^*) = 1$
  - If $f \neq 0$, set f=Verify$(\sigma, \mu, \boldsymbol{p}, \texttt{KRL}, \texttt{SRL})$.
  - Add $(\sigma, \mu, \boldsymbol{p}, \texttt{KRL}, f)$ to VerResults, output (VERIFIED, $sid,\ f$) to V.
- **R**EVOKE
  On input (REVOKE, $tsk^*, \sigma^*, \mu^*, \boldsymbol{p}^*$), the revocation manager adds tsk^* to $\texttt{KRL}$ or σ^* to $\texttt{SRL}$ after verifying σ^*.
- **O**UTPUT
  On input (OUTPUT, P, μ) from $\mathcal{S}$, output μ to P.

Fig. 11: Game 8 for $\mathcal{F}$

- **K**eyGen, **J**OIN: Unchanged.
- **S**IGN

  Honest $\mathcal{M}_i$:

  Upon receiving (SIGNSTART, $sid, ssid, \mathcal{M}_i, l$) from $\mathcal{F}$.
  - $\mathcal{S}$ takes a dummy pair (μ', bsn') such that $l(\mu', bsn') = l$.
  - $\mathcal{S}$ starts the simulation by giving "$\mathcal{M}_i$" input (SIGN, $sid, ssid, \mu', \boldsymbol{p}'$).

- When "$\mathcal{M}_i$" outputs (SIGNPROCEED, $sid, ssid, \mu', \boldsymbol{p}'$), $\mathcal{S}$ sends (SIGNSTART, $sid, ssid$) to $\mathcal{F}$.
- Upon receiving (SIGNCOMPLETE, $sid, ssid$) from $\mathcal{F}$, output (SIGNPROCEED, $sid, ssid$) to "$\mathcal{M}_i$".
- When "$\mathcal{M}_i$" outputs (SIGNATURE, $sid, ssid, \sigma$), send (SIGNCOMPLETE, $sid, ssid, \perp$) to $\mathcal{F}$.

Corrupt $\mathcal{M}_i$
Upon receiving (SIGNSTART, $sid, ssid, \mathcal{M}_i, l$) from $\mathcal{F}$.

- Send (SIGNSTART, $sid, ssid$) to $\mathcal{F}$.
- Upon receiving (SIGNPROCEED, $sid, ssid, \mu, \boldsymbol{p}$) from $\mathcal{F}$ on behalf of $\mathcal{M}_i$, as $\mathcal{M}_i$ is corrupt, $\mathcal{S}$ sends (SIGNPROCEED, $sid, ssid, \mu, \boldsymbol{p}$) to $\mathcal{F}$ on behalf of $\mathcal{M}_i$.
- Upon receiving (SIGNCOMPLETE, $sid, ssid$) from $\mathcal{F}$, send (SIGNCOMPLETE, $sid, ssid, \sigma$) to $\mathcal{F}$.

– **V**ERIFY, **L**INK: Nothing to simulate.

Fig. 12: Game 8 for $\mathcal{S}$

– **S**ETUP, **J**OIN: Unchanged.
– **S**IGN
- SIGN REQUEST: On input (SIGN, $sid, ssid, \mu, \boldsymbol{p}$) from $\mathcal{M}_i$,
- Abort if $\mathcal{I}$ is honest and no entry $\langle \mathcal{M}_i, \star \rangle$ exists $\mathtt{ML}$.
  * Create a sign session $\langle ssid, \mathcal{M}_i, \mu, \boldsymbol{p},\ request \rangle$.
  * Output (SIGNSTART, $sid, ssid, \mathcal{M}_i,\ l(\mu.bsn)$) to $\mathcal{S}$.
- SIGN REQUEST DELIVERY: On input (SIGNSTART, $sid, ssid$) from $\mathcal{S}$, update the session to $\langle ssid, \mathcal{M}_i, \mu, \boldsymbol{p},\ delivered \rangle$.
- Output (SIGNPROCEED, $sid, ssid, \mu, \boldsymbol{p}$) to $\mathcal{M}_i$.
- SIGN PROCEED: On input (SIGNPROCEED, $sid, ssid$) from $\mathcal{M}_i$
  * Update the records $\langle ssid, \mathcal{M}_i, \mu, \boldsymbol{p}$, delivered$\rangle$.
  * Output (SIGNCOMPLETE, $sid, ssid$) to $\mathcal{S}$.
- SIGNATURE GENERATION: On the input (SIGNCOMPLETE, $sid, ssid, \sigma$) from $\mathcal{S}$, if $\mathcal{M}_i$ is honest then:
  * Ignore the adversary's signature σ.
  * Generate the signature $\sigma \leftarrow Sign(tsk, \mu, \boldsymbol{p})$.
  * For all $(\sigma^*, \mu^*, \boldsymbol{p}^*) \in \mathtt{SRL}$, find all $(tsk^*, \mathcal{M}^*)$ from Members and DomainKeys such that identify$(\sigma^*, \mu^*, \boldsymbol{p}^*, \star, tsk^*) = 1$
    · Check that no two distinct keys tsk^* trace back to σ^*.
    · Check that no pair $(tsk^*, \mathcal{M}_i)$ was found.
  * If $\mathcal{M}_i$ is honest, then store $\langle \sigma, \mu, \mathcal{M}_i, \boldsymbol{p} \rangle$ in Signed and output (SIGNATURE, $sid, ssid, \sigma$) to $\mathcal{M}_i$.

– **V**ERIFY
On input (VERIFY, $sid, \mu, \boldsymbol{p}, \sigma, \mathtt{KRL}, \mathtt{SRL}$) from V

- Set $f = 0$ if
  - $*$ There is a $tsk^* \in \mathtt{KRL}$ such that Identify$(\sigma, \mu, \boldsymbol{p}, tsk^*) = 1$
- If $f \neq 0$, set f=Verify$(\sigma, \mu, \boldsymbol{p}, \mathtt{KRL}, \mathtt{SRL})$.
- Add $(\sigma, \mu, \boldsymbol{p}, \mathtt{KRL}, f)$ to VerResults, output (VERIFIED, *sid*, f) to V.

– **R**EVOKE
On input (REVOKE, $tsk^*, \sigma^*, \mu^*, \boldsymbol{p}^*$), the revocation manager adds tsk^* to $\mathtt{KRL}$ or σ^* to $\mathtt{SRL}$ after verifying σ^*.

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

Fig. 13: Game 9 for $\mathcal{F}$

– **S**ETUP: Unchanged.
– **J**OIN: Unchanged.
– **S**IGN: Unchanged.
– **V**ERIFY: Unchanged.
– **L**INK: Unchanged.

Fig. 14: Games 9-16 for $\mathcal{S}$

– **S**ETUP: Unchanged.
– **J**OIN
1. JOINREQUEST: On input (JOIN, $sid, jsid$) from $\mathcal{M}_i$
   - Create a join session $\langle jsid, \mathcal{M}_i,$ request $\rangle$.
   - Output (JOINSTART, $sid, jsid, \mathcal{M}_i$) to $\mathcal{S}$.
2. JJOIN REQUEST DELIVERY: Proceed upon receiving delivery notification from $\mathcal{S}$.
   - Update the session record to $\langle jsid, \mathcal{M}_i,$ delivered$\rangle$.
   - If $\mathcal{I}$ or $\mathcal{M}_i$ is honest and $\langle \mathcal{M}_i, \star\rangle$ is already in Members, output $\perp$.
   - Output (JOINPROCEED,$sid, jsid, \mathcal{M}_i$) to $\mathcal{I}$.
3. JOIN PROCEED: Upon receiving (JOINPROCEED,$sid, jsid, \mathcal{M}_i$) from $\mathcal{I}$
   - Update the session record to $\langle jsid, sid, \mathcal{M}_i,$ complete$\rangle$.
   - Output (JOINCOMPLETE, $sid, jsid$) to $\mathcal{S}$.
4. KEY GENERATION: On input (JOINCOMPLETE,$sid, jsid, tsk$) from $\mathcal{S}$.
   - Update the session record to $\langle jsid, \mathcal{M}_i,$ complete$\rangle$
   - If $\mathcal{M}_i$ is honest, set $tsk = \perp$.
   - Else, verify that the provided tsk is eligible by performing the following checks:
     - $*$ If $\mathcal{M}_i$ is honest, then CheckTskHonest(tsk)=1.
     - $*$ If $\mathcal{M}_i$ is corrupt, then CheckTskCorrupt(tsk)=1.
   - Insert $\langle \mathcal{M}_i, tsk\rangle$ into Members, and output (JOINED, $sid, jsid$) to $\mathcal{M}_i$.

- **S**IGN
  - SIGN REQUEST: On input (SIGN, $sid, ssid, \mu, \boldsymbol{p}$) from $\mathcal{M}_i$,
  - Abort if $\mathcal{I}$ is honest and no entry $\langle \mathcal{M}_i, \star \rangle$ exists ML.
    - ∗ Create a sign session $\langle ssid, \mathcal{M}_i, \mu, \boldsymbol{p},\ request \rangle$.
    - ∗ Output (SIGNSTART, $sid, ssid, \mathcal{M}_i,\ l(\mu.bsn)$) to $\mathcal{S}$.
  - SIGN REQUEST DELIVERY: On input (SIGNSTART, $sid, ssid$) from $\mathcal{S}$, update the session to $\langle ssid, \mathcal{M}_i, \mu, \boldsymbol{p},\ delivered \rangle$.
  - Output (SIGNPROCEED, $sid, ssid, \mu, \boldsymbol{p}$) to $\mathcal{M}_i$.
  - SIGN PROCEED: On input (SIGNPROCEED, $sid, ssid$) from $\mathcal{M}_i$
    - ∗ Update the records $\langle ssid, \mathcal{M}_i, \mu, \boldsymbol{p}$, delivered$\rangle$.
    - ∗ Output (SIGNCOMPLETE, $sid, ssid$) to $\mathcal{S}$.
  - SIGNATURE GENERATION: On the input (SIGNCOMPLETE, $sid, ssid, \sigma$) from $\mathcal{S}$, if $\mathcal{M}_i$ is honest then:
    - ∗ Ignore the adversary's signature σ.
    - ∗ Generate the signature $\sigma \leftarrow Sign(tsk, \mu, \boldsymbol{p})$.
    - ∗ Check CheckTskHonest(tsk)=1.
    - ∗ For all $(\sigma^*, \mu^*, \boldsymbol{p}^*) \in$ SRL, find all $(tsk^*, \mathcal{M}^*)$ from Members and DomainKeys such that identify$(\sigma^*, \mu^*, \boldsymbol{p}^*, \star, tsk^*) = 1$
      - · Check that no two distinct keys tsk^* trace back to σ^*.
      - · Check that no pair $(tsk^*, \mathcal{M}_i)$ was found.
    - ∗ If $\mathcal{M}_i$ is honest, then store $\langle \sigma, \mu, \mathcal{M}_i, \boldsymbol{p} \rangle$ in Signed and output (SIGNATURE, $sid, ssid, \sigma$) to $\mathcal{M}_i$.
- **V**ERIFY
  On input (VERIFY, $sid, \mu, \boldsymbol{p}, \sigma$, KRL, SRL) from V
  - Set $f = 0$ if
    - ∗ There is a $tsk^* \in$ KRL such that Identify$(\sigma, \mu, \boldsymbol{p}, tsk^*) = 1$.
  - If $f \neq 0$, set f=Verify($\sigma, \mu, \boldsymbol{p}$, KRL, SRL).
  - Add ($\sigma, \mu, \boldsymbol{p}$, KRL, f) to VerResults, output (VERIFIED, sid, f) to V.
- **R**EVOKE
  On input (REVOKE, $tsk^*, \sigma^*, \mu^*, \boldsymbol{p}^*$), the revocation manager adds tsk^* to KRL or σ^* to SRL after verifying σ^*.
- **O**UTPUT
  On input (OUTPUT, P, μ) from $\mathcal{S}$, output μ to P.

Fig. 15: Game 10 for $\mathcal{F}$

- **S**ETUP, **J**OIN: Unchanged.
- **S**IGN
  - SIGN REQUEST: On input (SIGN, $sid, ssid, \mu, \boldsymbol{p}$) from $\mathcal{M}_i$,
  - Abort if $\mathcal{I}$ is honest and no entry $\langle \mathcal{M}_i, \star \rangle$ exists ML.
    - ∗ Create a sign session $\langle ssid, \mathcal{M}_i, \mu, \boldsymbol{p},\ request \rangle$.
    - ∗ Output (SIGNSTART, $sid, ssid, \mathcal{M}_i,\ l(\mu.bsn)$) to $\mathcal{S}$.

- SIGN REQUEST DELIVERY: On input (SIGNSTART, $sid, ssid$) from $\mathcal{S}$, update the session to $\langle ssid, \mathcal{M}_i, \mu, \boldsymbol{p}, \ delivered\rangle$.
- Output (SIGNPROCEED, $sid, ssid, \mu, \boldsymbol{p}$) to $\mathcal{M}_i$.
- SIGN PROCEED: On input (SIGNPROCEED, $sid, ssid$) from $\mathcal{M}_i$
  - ∗ Update the records $\langle ssid, \mathcal{M}_i, \mu, \boldsymbol{p}$, delivered$\rangle$.
  - ∗ Output (SIGNCOMPLETE, $sid, ssid$) to $\mathcal{S}$.
- SIGNATURE GENERATION: On the input (SIGNCOMPLETE, $sid, ssid, \sigma$) from $\mathcal{S}$, if $\mathcal{M}_i$ is honest then:
  - ∗ Ignore the adversary's signature σ.
  - ∗ Generate the signature $\sigma \leftarrow Sign(tsk, \mu, \boldsymbol{p})$.
  - ∗ CheckTskHonest(tsk)=1.
  - ∗ Check Verify($\sigma, \mu, \boldsymbol{p}, \texttt{KRL}, \texttt{SRL}$)=1.
  - ∗ Check identify($\sigma, \mu, \ \boldsymbol{p}, tsk$)=1.
  - ∗ Check the is no signer other than $\mathcal{M}_i$ with key tsk' registered in Members or DomainKeys such that identify($\sigma, \mu, \boldsymbol{p}, tsk'$)=1.
  - ∗ For all $(\sigma^*, \mu^*, \boldsymbol{p}^*) \in \texttt{SRL}$, find all $(tsk^*, \mathcal{M}^*)$ from Members and DomainKeys such that identify$(\sigma^*, \mu^*, \boldsymbol{p}^*, \star, tsk^*) = 1$
    - · Check that no two distinct keys tsk^* trace back to σ^*.
    - · Check that no pair $(tsk^*, \mathcal{M}_i)$ was found.
  - ∗ If $\mathcal{M}_i$ is honest, then store $\langle \sigma, \mu, \mathcal{M}_i, \boldsymbol{p}\rangle$ in Signed and output (SIGNATURE, $sid, ssid, \sigma$) to $\mathcal{M}_i$.

– **V**ERIFY
On input (VERIFY, $sid, \mu, \boldsymbol{p}, \sigma, \texttt{KRL}, \texttt{SRL}$) from V
- Set $f = 0$ if
  - ∗ There is a $tsk^* \in \texttt{KRL}$ such that Identify$(\sigma, \mu, \boldsymbol{p}, tsk^*) = 1$
- If $f \neq 0$, set f=Verify($\sigma, \mu, \boldsymbol{p}, \texttt{KRL}, \texttt{SRL}$).
- Add $(\sigma, \mu, \boldsymbol{p}, \texttt{KRL}, f)$ to VerResults, output (VERIFIED, sid, f) to V.

– **R**EVOKE
On input (REVOKE, $tsk^*, \sigma^*, \mu^*, \boldsymbol{p}^*$), the revocation manager adds tsk^* to $\texttt{KRL}$ or σ^* to $\texttt{SRL}$ after verifying σ^*.

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

Fig. 16: Game 11 for $\mathcal{F}$

– **S**ETUP, **J**OIN: Unchanged.
– **S**IGN
- SIGN REQUEST: On input (SIGN, $sid, ssid, \mu, \boldsymbol{p}$) from $\mathcal{M}_i$,
- Abort if $\mathcal{I}$ is honest and no entry $\langle \mathcal{M}_i, \star\rangle$ exists $\texttt{ML}$.
  - ∗ Create a sign session $\langle ssid, \mathcal{M}_i, \mu, \boldsymbol{p}, \ request\rangle$.
  - ∗ Output (SIGNSTART, $sid, ssid, \mathcal{M}_i$, $l(\mu.bsn)$) to $\mathcal{S}$.
- SIGN REQUEST DELIVERY: On input (SIGNSTART, $sid, ssid$) from $\mathcal{S}$, update the session to $\langle ssid, \mathcal{M}_i, \mu, \boldsymbol{p}, \ delivered\rangle$.

- Output (SIGNPROCEED, $sid, ssid, \mu, \boldsymbol{p}$) to $\mathcal{M}_i$.
- SIGN PROCEED: On input (SIGNPROCEED, $sid, ssid$) from $\mathcal{M}_i$
  - Update the records $\langle ssid, \mathcal{M}_i, \mu, \boldsymbol{p}$, delivered$\rangle$.
  - Output (SIGNCOMPLETE, $sid, ssid$) to $\mathcal{S}$.
- SIGNATURE GENERATION: On the input (SIGNCOMPLETE, $sid, ssid, \sigma$) from $\mathcal{S}$, if $\mathcal{M}_i$ is honest then:
  - Ignore the adversary's signature σ.
  - Generate the signature $\sigma \leftarrow Sign(tsk, \mu, \boldsymbol{p})$.
  - Check $\mathsf{CheckTskHonest}(tsk)$=1.
  - Check Verify($\sigma, \mu, \boldsymbol{p}, \texttt{KRL}, \texttt{SRL}$)=1.
  - Check identify($\sigma, \mu, \boldsymbol{p}, tsk$)=1.
  - Check the is no signer other than $\mathcal{M}_i$ with key tsk' registered in Members or DomainKeys such that identify($\sigma, \mu, \boldsymbol{p}, tsk'$)=1.
  - For all $(\sigma^*, \mu^*, \boldsymbol{p}^*) \in \texttt{SRL}$, find all $(tsk^*, \mathcal{M}^*)$ from Members and DomainKeys such that identify$(\sigma^*, \mu^*, \boldsymbol{p}^*, \star, tsk^*) = 1$
    - Check that no two distinct keys tsk^* trace back to σ^*.
    - Check that no pair $(tsk^*, \mathcal{M}_i)$ was found.
  - If $\mathcal{M}_i$ is honest, then store $\langle \sigma, \mu, \mathcal{M}_i, \boldsymbol{p} \rangle$ in Signed and output (SIGNATURE, $sid, ssid, \sigma$) to $\mathcal{M}_i$.

– **V**ERIFY
On input (VERIFY, $sid, \mu, \boldsymbol{p}, \sigma, \texttt{KRL}, \texttt{SRL}$) from V
- Extract all pairs $(tsk_i, \mathcal{M}_i)$ from the DomainKeys and Members, for which Identify($\sigma, \mu, \boldsymbol{p}, tsk_i$)=1.
- Set $f = 0$ if
  - There is a $tsk^* \in \texttt{KRL}$ such that Identify$(\sigma, \mu, \boldsymbol{p}, tsk^*) = 1$
  - More than one key tsk_i was found.
- If $f \neq 0$, set f=Verify($\sigma, \mu, \boldsymbol{p}, \texttt{KRL}, \texttt{SRL}$).
- Add $(\sigma, \mu, \boldsymbol{p}, \texttt{KRL}, f)$ to VerResults, output (VERIFIED, sid, f) to V.

– **R**EVOKE
On input (REVOKE, $tsk^*, \sigma^*, \mu^*, \boldsymbol{p}^*$), the revocation manager adds tsk^* to $\texttt{KRL}$ or σ^* to $\texttt{SRL}$ after verifying σ^*.

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

Fig. 17: Game 12 for $\mathcal{F}$

– **S**ETUP, **J**OIN: Unchanged.
– **S**IGN
- SIGN REQUEST: On input (SIGN, $sid, ssid, \mu, \boldsymbol{p}$) from $\mathcal{M}_i$,
- Abort if $\mathcal{I}$ is honest and no entry $\langle \mathcal{M}_i, \star \rangle$ exists $\texttt{ML}$.
  - Create a sign session $\langle ssid, \mathcal{M}_i, \mu, \boldsymbol{p},\ request \rangle$.
  - Output (SIGNSTART, $sid, ssid, \mathcal{M}_i,\ l(\mu.bsn)$) to $\mathcal{S}$.

- SIGN REQUEST DELIVERY: On input (SIGNSTART, $sid, ssid$) from $\mathcal{S}$, update the session to $\langle ssid, \mathcal{M}_i, \mu, \boldsymbol{p},\ delivered \rangle$.
- Output (SIGNPROCEED, $sid, ssid, \mu, \boldsymbol{p}$) to $\mathcal{M}_i$.
- SIGN PROCEED: On input (SIGNPROCEED, $sid, ssid$) from $\mathcal{M}_i$
  - Update the records $\langle ssid, \mathcal{M}_i, \mu, \boldsymbol{p}$, delivered$\rangle$.
  - Output (SIGNCOMPLETE, $sid, ssid$) to $\mathcal{S}$.
- SIGNATURE GENERATION: On the input (SIGNCOMPLETE, $sid, ssid, \sigma$) from $\mathcal{S}$, if $\mathcal{M}_i$ is honest then:
  - Ignore the adversary's signature σ.
  - Generate the signature $\sigma \leftarrow Sign(tsk, \mu, \boldsymbol{p})$.
  - Check $\mathsf{CheckTskHonest}(tsk)$=1.
  - Check Verify$(\sigma, \mu, \boldsymbol{p}, \mathtt{KRL}, \mathtt{SRL})$=1.
  - Check identify$(\sigma, \mu,\ \boldsymbol{p}, tsk)$=1.
  - Check the is no signer other than $\mathcal{M}_i$ with key tsk' registered in Members or DomainKeys such that identify$(\sigma, \mu, \boldsymbol{p}, tsk')$=1.
  - For all $(\sigma^*, \mu^*, \boldsymbol{p}^*) \in \mathtt{SRL}$, find all $(tsk^*, \mathcal{M}^*)$ from Members and DomainKeys such that identify$(\sigma^*, \mu^*, \boldsymbol{p}^*, \star, tsk^*) = 1$
    - Check that no two distinct keys tsk^* trace back to σ^*.
    - Check that no pair $(tsk^*, \mathcal{M}_i)$ was found.
  - If $\mathcal{M}_i$ is honest, then store $\langle \sigma, \mu, \mathcal{M}_i, \boldsymbol{p} \rangle$ in Signed and output (SIGNATURE, $sid, ssid, \sigma$) to $\mathcal{M}_i$.

– **VERIFY**
On input (VERIFY, $sid, \mu, \boldsymbol{p}, \sigma, \mathtt{KRL}, \mathtt{SRL}$) from V
- Extract all pairs $(tsk_i, \mathcal{M}_i)$ from the DomainKeys and Members, for which Identify$(\sigma, \mu, \boldsymbol{p}, tsk_i)$=1.
- Set $f = 0$ if
  - $\mathcal{I}$ is honest and no pair $(tsk_i, \mathcal{M}_i)$ was found.
  - There is a $tsk^* \in \mathtt{KRL}$ such that Identify$(\sigma, \mu, \boldsymbol{p}, tsk^*) = 1$
  - More than one key tsk_i was found.
- If $f \neq 0$, set f=Verify$(\sigma, \mu, \boldsymbol{p}, \mathtt{KRL}, \mathtt{SRL})$.
- Add $(\sigma, \mu, \boldsymbol{p}, \mathtt{KRL}, f)$ to VerResults, output (VERIFIED, sid, f) to V.

– **REVOKE**
On input (REVOKE, $tsk^*, \sigma^*, \mu^*, \boldsymbol{p}^*$), the revocation manager adds tsk^* to $\mathtt{KRL}$ or σ^* to $\mathtt{SRL}$ after verifying σ^*.

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

Fig. 18: Game 13 for $\mathcal{F}$

– **SETUP**, **JOIN**: Unchanged.
– **SIGN**
- SIGN REQUEST: On input (SIGN, $sid, ssid, \mu, \boldsymbol{p}$) from $\mathcal{M}_i$,

- Abort if $\mathcal{I}$ is honest and no entry $\langle \mathcal{M}_i, \star \rangle$ exists $\texttt{ML}$.
  - Create a sign session $\langle ssid, \mathcal{M}_i, \mu, \boldsymbol{p},\ request \rangle$.
  - Output (SIGNSTART, $sid, ssid, \mathcal{M}_i,\ l(\mu.bsn)$) to $\mathcal{S}$.
- SIGN REQUEST DELIVERY: On input (SIGNSTART, $sid, ssid$) from $\mathcal{S}$, update the session to $\langle ssid, \mathcal{M}_i, \mu, \boldsymbol{p},\ delivered \rangle$.
- Output (SIGNPROCEED, $sid, ssid, \mu, \boldsymbol{p}$) to $\mathcal{M}_i$.
- SIGN PROCEED: On input (SIGNPROCEED, $sid, ssid$) from $\mathcal{M}_i$
  - Update the records $\langle ssid, \mathcal{M}_i, \mu, \boldsymbol{p}$, delivered$\rangle$.
  - Output (SIGNCOMPLETE, $sid, ssid$) to $\mathcal{S}$.
- SIGNATURE GENERATION: On the input (SIGNCOMPLETE, $sid, ssid, \sigma$) from $\mathcal{S}$, if $\mathcal{M}_i$ is honest then:
  - Ignore the adversary's signature σ.
  - Generate the signature $\sigma \leftarrow Sign(tsk, \mu, \boldsymbol{p})$.
  - Check $\mathsf{CheckTskHonest}(tsk)$=1.
  - Check Verify($\sigma, \mu, \boldsymbol{p}, \texttt{KRL}, \texttt{SRL}$)=1.
  - Check identify($\sigma, \mu,\ \boldsymbol{p}, tsk$)=1.
  - Check the is no signer other than $\mathcal{M}_i$ with key tsk' registered in Members or DomainKeys such that identify($\sigma, \mu, \boldsymbol{p}, tsk'$)=1.
  - For all $(\sigma^*, \mu^*, \boldsymbol{p}^*) \in \texttt{SRL}$, find all $(tsk^*, \mathcal{M}^*)$ from Members and DomainKeys such that identify$(\sigma^*, \mu^*, \boldsymbol{p}^*, \star, tsk^*) = 1$
    - Check that no two distinct keys tsk^* trace back to σ^*.
    - Check that no pair $(tsk^*, \mathcal{M}_i)$ was found.
  - If $\mathcal{M}_i$ is honest, then store $\langle \sigma, \mu, \mathcal{M}_i, \boldsymbol{p} \rangle$ in Signed and output (SIGNATURE, $sid, ssid, \sigma$) to $\mathcal{M}_i$.

– **V**ERIFY
On input (VERIFY, $sid, \mu, \boldsymbol{p}, \sigma, \texttt{KRL}, \texttt{SRL}$) from V
- Extract all pairs $(tsk_i, \mathcal{M}_i)$ from the DomainKeys and Members, for which Identify($\sigma, \mu, \boldsymbol{p}, tsk_i$)=1.
- Set $f = 0$ if
  - $\mathcal{I}$ is honest and no pair $(tsk_i, \mathcal{M}_i)$ was found.
  - An honest $\mathcal{M}_i$ was found, but no entry $\langle \star, \mu, \mathcal{M}_i, \boldsymbol{p} \rangle$ was found in Signed.
  - There is a $tsk^* \in \texttt{KRL}$ such that Identify$(\sigma, \mu, \boldsymbol{p}, tsk^*) = 1$
  - More than one key tsk_i was found.
- If $f \neq 0$, set f=Verify($\sigma, \mu, \boldsymbol{p}, \texttt{KRL}, \texttt{SRL}$).
- Add $(\sigma, \mu, \boldsymbol{p}, \texttt{KRL}, f)$ to VerResults, output (VERIFIED, $sid,\ f$) to V.

– **R**EVOKE
On input (REVOKE, $tsk^*, \sigma^*, \mu^*, \boldsymbol{p}^*$), the revocation manager adds tsk^* to $\texttt{KRL}$ or σ^* to $\texttt{SRL}$ after verifying σ^*.

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

Fig. 19: Game 14 for $\mathcal{F}$

- **S**ETUP, **J**OIN: Unchanged.
- **S**IGN
  - SIGN REQUEST: On input (SIGN, $sid, ssid, \mu, \boldsymbol{p}$) from $\mathcal{M}_i$,
  - Abort if $\mathcal{I}$ is honest and no entry $\langle \mathcal{M}_i, \star \rangle$ exists ML.
    - Create a sign session $\langle ssid, \mathcal{M}_i, \mu, \boldsymbol{p},\ request \rangle$.
    - Output (SIGNSTART, $sid, ssid, \mathcal{M}_i,\ l(\mu.bsn)$) to $\mathcal{S}$.
  - SIGN REQUEST DELIVERY: On input (SIGNSTART, $sid, ssid$) from $\mathcal{S}$, update the session to $\langle ssid, \mathcal{M}_i, \mu, \boldsymbol{p},\ delivered \rangle$.
  - Output (SIGNPROCEED, $sid, ssid, \mu, \boldsymbol{p}$) to $\mathcal{M}_i$.
  - SIGN PROCEED: On input (SIGNPROCEED, $sid, ssid$) from $\mathcal{M}_i$
    - Update the records $\langle ssid, \mathcal{M}_i, \mu, \boldsymbol{p}$, delivered$\rangle$.
    - Output (SIGNCOMPLETE, $sid, ssid$) to $\mathcal{S}$.
  - SIGNATURE GENERATION: On the input (SIGNCOMPLETE, $sid, ssid, \sigma$) from $\mathcal{S}$, if $\mathcal{M}_i$ is honest then:
    - Ignore the adversary's signature σ.
    - Generate the signature $\sigma \leftarrow Sign(tsk, \mu, \boldsymbol{p})$.
    - Check CheckTskHonest(tsk)=1.
    - Check Verify($\sigma, \mu, \boldsymbol{p}$, KRL, SRL)=1.
    - Check identify($\sigma, \mu,\ \boldsymbol{p}, tsk$)=1.
    - Check the is no signer other than $\mathcal{M}_i$ with key tsk' registered in Members or DomainKeys such that identify($\sigma, \mu, \boldsymbol{p}, tsk'$)=1.
    - For all $(\sigma^*, \mu^*, \boldsymbol{p}^*) \in$ SRL, find all $(tsk^*, \mathcal{M}^*)$ from Members and DomainKeys such that identify$(\sigma^*, \mu^*, \boldsymbol{p}^*, \star, tsk^*) = 1$
      - Check that no two distinct keys tsk^* trace back to σ^*.
      - Check that no pair $(tsk^*, \mathcal{M}_i)$ was found.
    - If $\mathcal{M}_i$ is honest, then store $\langle \sigma, \mu, \mathcal{M}_i, \boldsymbol{p} \rangle$ in Signed and output (SIGNATURE, $sid, ssid, \sigma$) to $\mathcal{M}_i$.
- **V**ERIFY
  On input (VERIFY, $sid, \mu, \boldsymbol{p}, \sigma$, KRL, SRL) from V
  - Extract all pairs $(tsk_i, \mathcal{M}_i)$ from the DomainKeys and Members, for which Identify($\sigma, \mu, \boldsymbol{p}, tsk_i$)=1.
  - Set $f = 0$ if
    - $\mathcal{I}$ is honest and no pair $(tsk_i, \mathcal{M}_i)$ was found.
    - An honest $\mathcal{M}_i$ was found, but no entry $\langle \star, \mu, \mathcal{M}_i, \boldsymbol{p} \rangle$ was found in Signed.
    - There is a $tsk^* \in$ KRL such that Identify$(\sigma, \mu, \boldsymbol{p}, tsk^*) = 1$, and no pair $(\mathcal{M}_i, tsk_i)$ for honest $\mathcal{M}_i$ was found.
    - More than one key tsk_i was found.
  - If $f \neq 0$, set f=Verify($\sigma, \mu, \boldsymbol{p}$, KRL, SRL).
  - Add ($\sigma, \mu, \boldsymbol{p}$, KRL, f) to VerResults, output (VERIFIED, sid, f) to V.
- **R**EVOKE
  On input (REVOKE, $tsk^*, \sigma^*, \mu^*, \boldsymbol{p}^*$), the revocation manager adds tsk^* to KRL or σ^* to SRL after verifying σ^*.
- **O**UTPUT
  On input (OUTPUT, P, μ) from $\mathcal{S}$, output μ to P.

Fig. 20: Game 15 for $\mathcal{F}$

- **S**ETUP, **J**OIN: Unchanged.
- **S**IGN
  - SIGN REQUEST: On input (SIGN, $sid, ssid, \mu, \boldsymbol{p}$) from $\mathcal{M}_i$,
  - Abort if $\mathcal{I}$ is honest and no entry $\langle \mathcal{M}_i, \star \rangle$ exists ML.
    - Create a sign session $\langle ssid, \mathcal{M}_i, \mu, \boldsymbol{p},\ request \rangle$.
    - Output (SIGNSTART, $sid, ssid, \mathcal{M}_i,\ l(\mu.bsn)$) to $\mathcal{S}$.
  - SIGN REQUEST DELIVERY: On input (SIGNSTART, $sid, ssid$) from $\mathcal{S}$, update the session to $\langle ssid, \mathcal{M}_i, \mu, \boldsymbol{p},\ delivered \rangle$.
  - Output (SIGNPROCEED, $sid, ssid, \mu, \boldsymbol{p}$) to $\mathcal{M}_i$.
  - SIGN PROCEED: On input (SIGNPROCEED, $sid, ssid$) from $\mathcal{M}_i$
    - Update the records $\langle ssid, \mathcal{M}_i, \mu, \boldsymbol{p}$, delivered$\rangle$.
    - Output (SIGNCOMPLETE, $sid, ssid$) to $\mathcal{S}$.
  - SIGNATURE GENERATION: On the input (SIGNCOMPLETE, $sid, ssid, \sigma$) from $\mathcal{S}$, if $\mathcal{M}_i$ is honest then:
    - Ignore the adversary's signature σ.
    - Generate the signature $\sigma \leftarrow Sign(tsk, \mu, \boldsymbol{p})$.
    - Check CheckTskHonest(tsk)=1.
    - Check Verify($\sigma, \mu, \boldsymbol{p}, \texttt{KRL}, \texttt{SRL}$)=1.
    - Check identify($\sigma, \mu,\ \boldsymbol{p}, tsk$)=1.
    - Check the is no signer other than $\mathcal{M}_i$ with key tsk' registered in Members or DomainKeys such that identify($\sigma, \mu, \boldsymbol{p}, tsk'$)=1.
    - For all $(\sigma^*, \mu^*, \boldsymbol{p}^*) \in \texttt{SRL}$, find all $(tsk^*, \mathcal{M}^*)$ from Members and DomainKeys such that identify$(\sigma^*, \mu^*, \boldsymbol{p}^*, \star, tsk^*) = 1$
      - Check that no two distinct keys tsk^* trace back to σ^*.
      - Check that no pair $(tsk^*, \mathcal{M}_i)$ was found.
    - If $\mathcal{M}_i$ is honest, then store $\langle \sigma, \mu, \mathcal{M}_i, \boldsymbol{p} \rangle$ in Signed and output (SIGNATURE, $sid, ssid, \sigma$) to $\mathcal{M}_i$.
- **V**ERIFY

  On input (VERIFY, $sid, \mu, \boldsymbol{p}, \sigma, \texttt{KRL}, \texttt{SRL}$) from V
  - Extract all pairs $(tsk_i, \mathcal{M}_i)$ from the DomainKeys and Members, for which Identify($\sigma, \mu, \boldsymbol{p}, tsk_i$)=1.
  - Set $f = 0$ if
    - $\mathcal{I}$ is honest and no pair $(tsk_i, \mathcal{M}_i)$ was found.
    - An honest $\mathcal{M}_i$ was found, but no entry $\langle \star, \mu, \mathcal{M}_i, \boldsymbol{p} \rangle$ was found in Signed.
    - There is a $tsk^* \in \texttt{KRL}$ such that Identify$(\sigma, \mu, \boldsymbol{p}, tsk^*) = 1$, and no pair $(\mathcal{M}_i, tsk_i)$ for honest $\mathcal{M}_i$ was found.
    - For some matching $tsk_i \in \texttt{ML}$ and $(\sigma^*, \mu^*, \boldsymbol{p}^*) \in \texttt{SRL}$, such that identify$(\sigma^*, \mu^*, \boldsymbol{p}^*, tsk_i) = 1$.
    - More than one key tsk_i was found.
  - If $f \neq 0$, set f=Verify($\sigma, \mu, \boldsymbol{p}, \texttt{KRL}, \texttt{SRL}$).
  - Add $(\sigma, \mu, \boldsymbol{p}, \texttt{KRL}, f)$ to VerResults, output (VERIFIED, $sid,\ f$) to V.
- **R**EVOKE

  On input (REVOKE, $tsk^*, \sigma^*, \mu^*, \boldsymbol{p}^*$), the revocation manager adds tsk^* to $\texttt{KRL}$ or σ^* to $\texttt{SRL}$ after verifying σ^*.

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

Fig. 21: Game 16 for $\mathcal{F}$