Collaborative, Segregated NIZK (CoSNIZK) and More Efficient Lattice-Based Direct Anonymous Attestation

Liqun Chen[2], Patrick Hough[1], and Nada El Kassem[2]

[1] Mathematical Institute
Oxford University
patrick.hough@maths.ox.ac.uk
[2] Surrey Centre for Cyber Security, Guildford, UK
University of Surrey
{liqun.chen, nada.elkassem}@surey.ac.uk

Abstract. Direct Anonymous Attestation (DAA) allows a (host) device with a Trusted Platform Module (TPM) to prove that it has a certified configuration of hardware and software whilst preserving the privacy of the device. All deployed DAA schemes are based on classical security assumptions. Despite a long line of works proposing post-quantum designs, the vast majority give only theoretical schemes and where concrete parameters are computed, their efficiency is far from practical.

Our first contribution is to define *collaborative, segregated, non-interactive zero knowledge* (CoSNIZK). This notion generalizes the property of collaborative zero-knowledge as formalised by Ozdemir and Boneh [OB22] so that the zero-knowledge property need only apply to a subset of provers during collaborative proof generation. This if of particular interest for proxy-cryptography in which part of an expensive but sensitive computation may be delegated to another party. We give a lattice-based CoSNIZK based on the highly efficient proof framework in [LNP22].

Our main contribution is the construction of a DAA based on the hardness of problems over module lattices as well as the ISIS_f assumption recently introduced in [BLNS23]. A key component of our work is the CoSNIZK construction which allows the TPM and host to jointly create attestations whilst protecting TPM key material from a potentially corrupt host. We prove the security of our DAA scheme according to the well-established UC definition of [CDL16a]. Our design achieves DAA signatures more than 1.5 orders of magnitude smaller than previous works at only 38KB making it the first truly practical candidate for post-quantum DAA.

1 INTRODUCTION

Direct Anonymous Attestation (DAA) is a cryptographic protocol which allows a Trusted Platform Module (TPM) to serve as a root of trust for a host device in which it is embedded. The TPM does so by creating attestations about the state of the host system e.g., by certifying the boot sequence of the host system. Such attestations convince a remote verifier that the device it is communicating with comprises trusted hardware and is running correct software. A core designs principle of DAA is that such attestations are made in a *privacy-preserving* manner. That is, a verifier can check that such attestations originate from a certified device without learning anything about the identity of the TPM. Furthermore, DAA supports *user-controlled-linkability* through choice of a basename bsn. Attestations (signatures) made with a fresh basename or $\mathsf{bsn} = \perp$ are unlinkable while repeatedly signing with the same basename creates attestations that are linkable.

One can view DAA as a variant of group signatures in which a single issuer controls the admission of members to the group. Moreover, the opening property is replaced by user-controlled-linkability. Importantly, DAA signatures cannot be created without the approval and involvement of the TPM so that a potentially corrupt host (e.g., running malicious software) cannot produce attestations alone.

Conceived by Brickell, Camenisch, and Chen [BCC04] in 2004 and standardized by the Trusted Computing Group (TCG) as TPM 1.2 (2004) [Gro04] and TPM 2.0 (2014) [Gro14], DAA is deployed in billions [Gro19] of devices worldwide including embedded devices, servers, and the majority of enterprise PCs. TPM 2.0 supports a suite of elliptic-curve based DAA designs [EB, BCL08, CPS09] which are further included in the ISO standard ISO/IEC 20008-2 [Int13, Int15]. Owing to its wide deployment and standardization, there has been a huge body of additional work relating to DAA aiming to improve its security and efficiency [BFG+11, CCD+17, CDL16a, CDL15, CDL17, CU15, Xi14, SRC12].

DAA has also been used as a basis for a number of other privacy-enhancing protocols and for an increasing number of practical applications in a society for which awareness of private authentication is on the rise. A variant of DAA called Enhanced Privacy ID (EPID) is used in Intel SGX [BL07] and DAA is also in the Fast IDentity Online (FIDO) [CDE+17] authentication framework in which a TPM produces attestations that new authentication keys are securely stored on the TPM. More recently, DAA has been proposed for authentication in vehicle-to-vehicle communication [CTY+21] wherein tracking of drivers must be prevented.

The security of all DAA schemes currently supported by TPMs rely on classical hardness assumptions; factorization in the RSA setting or finding discrete logarithms in the elliptic curve setting. Upon the arrival of quantum computers, all cryptography built on such problems will be deemed insecure. Moreover, prompted by the selection of post-quantum secure NIST standards for key-exchange and digital signature algorithms [Nat22], there is a rapidly increasing adoption of post-quantum secure alternative designs. Indeed, the EU Horizon2020 FutureTPM project [Con18] was established with the aim of designing a 'FutureTPM' with post-quantum security. The project is planning to consider the adoption of the Kyber [BDK+17], Dilithium [DKL+18] and SPHINCS+ [BHK+19] algorithms of key exchange and plain signatures. However, there is still much more work to be done in designing efficiency *privacy-preserving* designs with post-quantum security. Regarding DAA, there have been a number of works aiming at post-quantum security [EE17, ECE$^+$19, ECE$^+$18, CDE$^+$23, CKLL19]. As with many post-quantum designs, most are built using computational problems over lattices.

Unfortunately, these works largely provide only a theoretical framework for post-quantum DAA. In particular, all but [CKLL19] give only asymptotic parameters and thus no concrete efficiency can be ascertained. Regarding, [CKLL19], the authors give a rough estimate for the size of their DAA signatures at 2MB without providing justification for their parameter selection. Comparing this to deployed DAA schemes based on classical hardness assumptions, this signature size is over 1000 times larger. As a general rule-of-thumb, post-quantum secure schemes being adopted to replace classical ones come with a communication cost roughly 25-30X higher. The gap in post-quantum DAA efficiency to this range and the number of works attempting to cross that gap evidence the challenge of instantiating such a complex protocol from post-quantum assumptions. It is clear that a design of significant novelty is needed to bring the efficiency of post-quantum DAA into the realm of practicality.

A key technical question in designing a DAA scheme is how the TPM and Host can jointly create attestations in a way that (1) requires the TPM's involvement so as to stop a malicious host making false attestations and (2) that protects the sensitive material stored on the TPM from the possibly corrupt host. A naive solution to this would be to require the TPM to store all credentials and create all attestations itself, only passing them to the host for broadcasting to verifiers. However, in reality the TPM is a highly constrained device, both in terms of storage and computation. Thus, in order to realize a practical DAA, some of this storage and computation must be delegated to the host. These two entities must now work collaboratively to prove the integrity of the platform (i.e. knowledge of valid credentials).

This kind of scenario is not uncommon in cryptography. Indeed, distributed provers were considered by Pedersen as early as 1991 [Ped91] in the context of undeniable signatures. Here, the prover's interaction is required to validate a given signature. Recently, a number of works have considered the distributed prover setting [DPP$^+$22, DFK$^+$23, WZC$^+$18]. A recent work by Odzemir and Boneh [OB22] explores collaborative zk-SNARKS and their potential applications to healthcare, credit-score computation, and private audits of multiple parties.

In DAA, we have two parties (a TPM and host) who must work collaboratively to create zero-knowledge proofs (of credentials). However, the setting is subtly different to the standard multi-prover setting in that the host's share of the witness need not be cryptographically hidden from the TPM. That is, while the zero-knowledge property of the final proof is required, the host's witness share does not need to be hidden to the TPM *during proof generation*. In fact, this one-sided zero-knowledge property is not unique to DAA. Indeed, so-called 'proxy-cryptography' [ID03] is often employed for reasons of efficiency, usability, and scalability. In one example, the president delegates part of her signing key to her assistant. This can be used to sign in the president's name only when the vice president has already added his signature to a document. This allows the president to sign while absent but only in combination with the vice-president's signature.

For constrained devices such as a TPMs, Intel's SGX, or Apple's 'T-series' hardware chips (macbooks) and secure enclaves (iphones), the natural questions arise: (1) can their crypographic key material be partially delegated to allow for greater efficiency or even new functionality and (2) does one-sided witness privacy allow for more efficient solutions than treating all parties as equals in a standard multi-prover collaborative proof and (3) can one construct a more efficient DAA using these combined ideas?

1.1 Our Results

We present a lattice-based DAA scheme based on the module learning with errors (MLWE), short integer solution (MSIS), and MNTRU problems. Additionally, inspired by the anonymous credentials work of Bootle et al. in [BLNS23], we make use of the ISIS_f assumption introduced there. Along the way we formalise the the notion of collaborative, segregated NIZK ($\mathsf{CoSNIZK}$), giving a construction based on the highly efficient LNP proof framework [LNP22]. Finally, we provide concrete parameters to securely instantiate our DAA design which lead to a signature size (38KB) 1.5 orders of magnitude smaller than the state of the art and a 4-fold reduction in TPM key size. This ends a long line of work seeking to provide post-quantum secure DAA with truly practical efficiency.

Collaborative, Segregated NIZK. We begin by introducing the notion of collaborative, segregated non-interactive zero-knowledge ($\mathsf{CoSNIZK}$). This generalizes the concept of collaborative zero-knowledge as formalized by Ozdemir and Boney in [OB22] so that the zero-knowledge property need only apply to a subset of (segregated) provers during collaborative proof generation. We give a lattice-based $\mathsf{CoSNIZK}$ based on the highly efficient proof framework in [LNP22]. Moreover, this construction allows one party to delegate part of its witness to another (freeing up storage) and the resulting proof size is unchanged from the single-prover setting.

Module NTRU-ISIS_f *for hash-and-sign blind-signatures.* In [BLNS23], the authors introduce a new family of lattice problems whose members are conducive to efficient zero-knowledge proofs for proving knowledge of a solution. The authors show how this is a natural extension of the SIS and ISIS assumptions upon which hash-and-sign GPV signatures [GPV08] are based. Where the authors of [BLNS23] use this assumption in conjunction with *ring* variants of LWE and SIS, we show how it can be made compatible with *module* lattices and so make use of the more efficient trapdoor sampling techniques used in [CPS+19].

A More Efficient LDAA. We combine the building blocks above to present a new lattice-based DAA scheme and provide concrete parameters. The efficiency significantly improves upon the state-of-the-art in [CKLL19], reducing the DAA signature size by 52 times to 38KB. As compared to deployed classical schemes, this lands within the range often considered for practical use.

2 Preliminary

2.1 The Ring R_q

We define $\Phi : R_q \to (\mathbb{Z}_q^d)^\top$ to be the map that sends a polynomial $a \in R_q$ to its (column) coefficient vector. We define $\mathsf{Rot} : R_q \to \mathbb{Z}_q^{d \times d}$ to be the map that sends a polynomial $a \in R_q$ to a matrix whose i'th column is $\Phi(a \cdot X^i \mod (X^d + 1))$. It is easily checked that $\mathsf{Rot}(a)\Phi(b) = \Phi(a \cdot b)$ and that this notation extends naturally to vectors and matrices of polynomials. We will often use $\vec{a}$ in place of $\Phi(a)$ to ease presentation. When using hash functions, we will denote their image space using subscripts. For example, H_{R_q} maps an input to an element of R_q Viewing ring elements in their coefficient embedding we can consider their ℓ_1, ℓ_2, and ℓ_∞-norms, extending their definitions in the natural way for k-dimensional vectors $\mathbf{a} \in R^k$. For an elements $a \in R_q$, we may sometimes write $a \mod p$ to mean that a's coefficients are reduced modulo p so that $(a \mod p) \in R_p$. We denote by H_S, a hash function with target space S. In zero-knowledge arguments, this will also be used to denote the random oracle. For a polynomial $x \in R_q$, we denote by $\tilde{x}$ its constant coefficient. We denote by $\vec{x}$ a column vector of integers and for $\mathbf{x} \in R_q^n$, we denote by $\vec{\mathbf{x}}$ the nd-length coefficient column vector.

2.2 Module-LWE and Module-SIS Problems

The security of our scheme relies on the lattices problems Module-SIS (MSIS) and Module-LWE (MLWE)

Definition 1 **($\mathsf{MLWE}_{n,m,\chi}$).** *Let n, m be positive integers, and χ be an error distribution over R_q*[3]*. The* Module-LWE *problem then asks an adversary $\mathcal{A}$ to distinguish between the following two cases: 1) $(\mathbf{A}, \mathbf{A}\mathbf{s} \mod q)$ for $\mathbf{A} \leftarrow R_q^{n \times (n+m)}$, and secret vector $\mathbf{s} \leftarrow \chi^{n+m}$ and 2) $(\mathbf{A}, \mathbf{b}) \leftarrow R_q^{n \times (n+m)} \times R_q^n$.*

Definition 2 **($\mathsf{MSIS}_{n,m,\mathcal{B}}$).** *Let n, m be positive integers, and $0 < \mathcal{B} < q$. Then, given $\mathbf{A} \in R_q^{n \times m}$, the* Module-SIS *problem asks an adversary $\mathcal{A}$ to find $\mathbf{z} \in R_q^m$ such that $\mathbf{A}\mathbf{z} = \mathbf{0} \mod q$ and $0 < \|z\|_2 \le \mathcal{B}$.*

Definition 3 **($\mathsf{MNTRU}_{n,m,\chi,q}$ [CPS+20]).** *Let n be a positive integer and χ be a bounded distribution over R_q. The Module-NTRU problem then asks an adversary $\mathcal{A}$ to distinguish between the following two cases:* (1) *$\mathbf{F}^{-1}\mathbf{g} \in R_q^n$ for secret $(\mathbf{F}, \mathbf{g}) \leftarrow \chi^{n \times n} \times \chi^n$, and* (2) *$\mathbf{h} \in R_q^n$ for uniform $\mathbf{h} \xleftarrow{\$} R_q^n$.*

2.3 NTRU Lattices

In this section, and wherever NTRU trapdoor sampling is referred to in this work, we consider a ring $\hat{R} := \mathbb{Z}[X]/(X^{\hat{d}} + 1)$ of dimension $\hat{d}$. This is to distinguish

[3] Note, we do not include q and d as parameters in the subscript of $\mathsf{MLWE}_{n,m,\chi}$ for cleaner notation. It is thus implicit that all computational assumptions are defined with respect to R_q in this work.

between the ring over which MLWE and MSIS are defined. Given a basis B, we denote the Gram-Schmidt orthogonalisation of B by $\tilde{B}$. Let $\hat{d}$ be a power of two, q be a positive integer, $\mathbf{g} \in \hat{R}_q^{\hat{n}}$, and $\mathbf{F} \in \hat{R}_q^{\hat{n} \times \hat{n}}$ be invertible. Let $\mathbf{h} = \mathbf{F}^{-1}\mathbf{g} \in \hat{R}_q^{\hat{n}}$ and define the *NTRU* module as

$$\mathcal{L}_{\mathsf{NTRU}} := \{(u, \mathbf{v}) \in \hat{R}^{\hat{n}+1} \ : \ u + \mathbf{v}^T \mathbf{h} = \quad \bmod q\}.$$

$\mathcal{L}_{\mathsf{NTRU}}$ admits bases in the form of

$$\mathbf{B}_{\mathsf{NTRU}} := \begin{pmatrix} -\mathbf{h} & \mathbf{I}_{\hat{n}} \\ q & \mathbf{0}_{\hat{n}} \end{pmatrix} \text{ and } \mathbf{B}_{\mathbf{F},\mathbf{g}} := \begin{pmatrix} \mathbf{g} & -\mathbf{F} \\ g_0 & -\mathbf{f}_0^T \end{pmatrix}.$$

As with any module, $\mathcal{L}_{\mathsf{NTRU}}$ can be seen as a $\mathbb{Z}$-lattice over $\mathbb{Q}^{(\hat{n}+1)\hat{d}}$.

Lemma 1 ([CPS+20]). *There is an efficient algorithm* $\mathsf{NTRU.TrapGen}(q, \hat{d}, \hat{n})$ *which outputs* $\mathbf{h}$ *and a basis* $\mathbf{B}$ *of* $\mathcal{L}_{\mathsf{NTRU}}$ *such that* $\|\tilde{\mathbf{B}}\|_2 \leq \gamma \cdot q^{1/(\hat{n}+1)}$, *where* $\gamma = 1.17$ *for* $\hat{n} = 1, 2$ *and* $\gamma = 1.24$ *for* $\hat{n} = 3$. *Furthermore, let* $\epsilon = 1/(4\sqrt{\lambda})$. *There is a PPT algorithm* $\mathsf{GSampler}$, *which takes* $(\mathbf{h}, \mathbf{B}_{\mathbf{F},\mathbf{g}}) \leftarrow \mathsf{NTRU.TrapGen}(\hat{d}, \hat{n}, q)$, *standard deviation* $\mathfrak{s}_{\mathsf{pre}} > 0$ *and a target vector* $c \in \hat{R}_q$ *as input and outputs* $\mathbf{e} \in \hat{R}_q^{\hat{n}+1}$ *such that*

$$\Delta\left(\left[1 \ \mathbf{h}^T\right]^{-1}_{\mathfrak{s}_{\mathsf{pre}}}(c), \mathsf{GSampler}(\mathbf{h}, \mathbf{B}_{\mathbf{F},\mathbf{g}}, \mathfrak{s}_{\mathsf{pre}}, c)\right) \leq 2^{-\lambda},$$

as long as

$$\mathfrak{s}_{\mathsf{pre}} \geq \gamma q^{1/(\hat{n}+1)} \cdot \eta_\epsilon(\mathbb{Z}^{(\hat{n}+1)\hat{d}}) \text{ where } \eta_\epsilon(\mathbb{Z}^{(\hat{n}+1)\hat{d}}) \approx \frac{1}{\pi} \cdot \sqrt{\frac{1}{2} \log\left(2\hat{d}(\hat{n}+1)(1+\frac{1}{\epsilon})\right)}.$$

2.4 The $\mathsf{Int\text{-}NTRU\text{-}ISIS}_f$ Problem

We recall the falsifiable (interactive) $\mathsf{Int\text{-}NTRU\text{-}ISIS}_f$ assumption as introduced by Bootle et. al in [BLNS23]. Note the slight modification we make to the relation a winning output must satisfy. In particular, the vector $\mathbf{1}^\top$ is used to compress the right hand side of the relation into a single ring element. This is needed in order to apply the trapdoor sampling techniques of [CPS+20] in the *module* NTRU setting.

Definition 4 (The $\mathsf{Int\text{-}MNTRU\text{-}ISIS}_f$ Problem (Interactive)). *Define the system parameters* $\mathsf{sp} = (q, d, n, m, N, \mathfrak{s}, \mathcal{B}_s, \mathcal{B}_m)$ *as a tuple of functions of the security parameter* λ. *Let* $f : [N] \to R_q^n$ *be an efficiently computable function. We say an adversary* $\mathcal{A}$ *solves the* $\mathsf{Int\text{-}MNTRU\text{-}ISIS}_f$ *if it wins the experiment defined in Figure 1.*

Concrete instantiation of f. We instantiate f, as in [BLNS23], using a binary encoding function parameterized by an integer $t \in \mathbb{N}$ and a matrix $B \in \mathbb{Z}_q^{nd \times t}$. The function f is then defined as

$$f(x) := \Phi^{-1}(B \cdot \mathsf{bin}(x)) \in R_q^n,$$

where $\mathsf{bin} \in \{0,1\}^t$ is a binary decomposition of $x - 1$. Hence, $N = 2^t$.

```
$\mathsf{Exp}_{\mathsf{sp}}^{\mathsf{Int\text{-}MNTRU\text{-}ISIS}_f}(\mathcal{A})$
1: $(\mathbf{h}, \mathbf{B}_{\mathbf{F},\mathbf{g}}) \leftarrow \mathsf{NTRU.TrapGen}(q, n, d)$
2: $\mathbf{A} := \begin{bmatrix} 1 & \mathbf{h}^T \end{bmatrix}$
3: $\mathbf{C} \leftarrow R_q^{n \times (n+m)}$
4: $\mathcal{M} = \emptyset$
5: $(x^*, \mathbf{s}^*, \mathbf{m}^*, \mathbf{r}^*) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{pre}}}(\mathbf{A}, \mathbf{C})$
6: if $(\mathbf{m}^* \notin \mathcal{M})$
       $\wedge \left( \mathbf{A}\mathbf{s}^* = \mathbf{1}^\top \left( f(x^*) + \mathbf{C} \begin{bmatrix} \mathbf{m}^* \\ \mathbf{r}^* \end{bmatrix} \right) \right)$
       $\wedge (0 < \|\mathbf{s}^*\| \leq \mathcal{B}_s) \wedge (\|\mathbf{m}^*\|, \|\mathbf{r}^*)\| \leq \mathcal{B}_m)$
7:     then return 1
8: else return 0

$\mathcal{O}_{\mathsf{pre}}(\mathbf{m}, \mathbf{r})$
1: if $\|\mathbf{m}, \mathbf{r}\| < \mathcal{B}_m$
2:     then return $\perp$
3: $x \leftarrow [N]$
4: $\mathbf{s} \leftarrow \mathbf{A}_{\mathsf{s}}^{-1} \left( \mathbf{1}^\top \left( f(x) + \mathbf{C} \begin{bmatrix} \mathbf{m} \\ \mathbf{r} \end{bmatrix} \right) \right)$
5: $\mathcal{M} \leftarrow \mathcal{M} \cup \{\mathbf{m}\}$
6: return $(x, \mathbf{s})$
```

Fig. 1. The interactive $\mathsf{ISIS_f}$ experiment

2.5 The LNP Proof System

Throughout this work, as in many privacy-enhancing protocols, parties will prove that they have behaved according to the protocol 'in zero-knowledge'. That is, without exposing any information intended to be kept secret.

We employ the lattice-based proof system in [LNP22]. Herein, we may refer to this proof system as the 'LNP' proof. The authors provide a highly efficient framework for proving a wide range of lattice relations. In particular, one can prove *exact* norm bounds of secret vectors satisfying some given relation. In this work we will apply these techniques to relations of the form

$$\mathbf{P}\mathbf{s} = \mathbf{v} \text{ over } R_q, \tag{1}$$

where $\mathbf{P}$ and $\mathbf{v}$ are a public matrix and vector of elements in R_q whilst $\mathbf{s}$ is a secret vector of such elements. Given $\mathbf{s}$ such that (1) $\mathbf{s}$ satisfies Equation (1) and (2) $\|\mathbf{s}\| \leq \mathcal{B}$ for some bound $\mathcal{B}$, one can prove both (1) and (2). Furthermore, the authors provide a SAGE tool for computing secure parameters for such a proof.

The techniques of [LNP22] are highly involved and a full presentation of the core proof structure would not be practical. Instead we give a high-level overview of the key ideas. For a full exposition of the details, we direct the reader to [LNP22, Section 5.1]. One key observation used in the LNP proof system is that the inner product of two vectors $\vec{r}$ and $\vec{s}$ can be made to appear as the constant coefficient in a polynomial product of two polynomials which are functions of $\vec{r}$ and $\vec{s}$. Then, by using a proof of polynomial product and hiding all but one coefficient, it is possible to prove vector inner products modulo q. Noting that the inner product of a vector with itself is by definition the ℓ_2-norm squared of that vector, one has a method for proving vector norm values modulo q. Secondly, using an inexpensive 'range proof' one can prove that the norm of a vector is not bigger than q. Thus, by combining these two strategies, one can lift the norm proof over $\mathbb{Z}_q$ to one over $\mathbb{Z}$ as required. The first step in employing the techniques of [LNP22] is to write the relation to be proven as a relation over

$\mathbb{Z}_q$ so that one is always proving a relation of the following form

$$\mathcal{R}_{\mathsf{Gen}} := \{\mathsf{X} := (P, \vec{\mathbf{v}}), \mathsf{W} := \vec{\mathbf{s}}_1 \mid P \in \mathbb{Z}_q^{nd \times m_1 d}, \vec{\mathbf{v}} \in \mathbb{Z}_q^{nd}, \vec{\mathbf{s}}_1 \in \mathbb{Z}_q^{m_1 d} : P\vec{\mathbf{s}_1} = \vec{\mathbf{v}} \wedge \|\vec{\mathbf{s}}_1\|_2 \leq \mathcal{B}_s\}. \quad (2)$$

This can be done by using rotation matrices and coefficient vectors defined by Rot and Φ.

Challenge Space. For reference, we recall the challenge space used by the LNP proof system [LNP22]. Let $\xi, \mu > 0$ and k be a power-of-two. Define the challenge space $\mathcal{C}$ as $\mathcal{C} := \{c \in R_q : \|c\|_\infty \leq \xi \wedge \sigma(c) = c \wedge \sqrt[2k]{\|c^{2k}\|_1} \leq \nu\}$. We use the experimental values from [BLNS23], ensuring that for c sampled uniformly from the space of infinity-norm ξ-bounded elements, it holds that $\sqrt[2k]{\|c^{2k}\|_1} \leq \nu$ with probability $> 99\%$. We use $d = 128$, $\xi = 2$, $\nu = 59$, $k = 32$, which give $|\mathcal{C}| > 2^{147}$.

3 Collaborative Segregated NIZK

In this section we consider a special case of distributed-prover zero-knowledge, often referred to as distributed, interactive zero-knowledge (DIZK) or collaborative, non-interactive zero-knowledge (CoNIZK). We will use the name CoNIZK as coined by Ozdemir and Boneh in [OB22] (though there the authors consider collaborative zk-SNARKs). In the CoNIZK setting, a set of participants $\mathcal{U} := \{\mathcal{P}_i\}_{i \in [N]}$ each hold a share W_i in some witness $\mathsf{W} := \{\mathsf{W}_i\}_{i \in [N]}$. Their aim, through collaboration, is to prove that $(\mathsf{X}, \mathsf{W}) \in \mathcal{R}$ for some public statement X and NP language $\mathcal{L}$ defined by the relation $\mathcal{R}$. The resulting proof should be correct, sound, and zero-knowledge in the usual sense. Furthermore, the proof generation procedure should not reveal any information about a participant's witness share to the other participants. One can thus think of zero-knowledge as being both *local* (protecting witness shares between provers) and *global* (protecting the full witness and its shares from any external party).

We consider a generalization of CoNIZK in which the *zero-knowledge property may not protect the witness shares of some parties involved in the proof generation process*. That is, witness share information of some parties may be leaked *internally* during the proof creation but the resulting proof is still zero-knowledge with respect to the full witness W. To formalize this idea we consider that parties are *segregated* into two groups; those which we call *closed* lie in $\mathcal{U}_c$ whilst *open* parties lie in $\mathcal{U}_o$. Closed parties are those for whom local zero-knowledge applies (their witness shares remain hidden) whilst open parties may leak some information about their witness shares through the proof generation procedure.[4] Herein, we will refer to this primitive as collaborative, segregated non-interactive zero-knowledge, denoted CoSNIZK which we now formally define.

3.1 Syntax

A *segregated, non-interactive collaborative proof* CoSNIZK is defined for a set $\mathcal{U}$ of N provers and a relation $\mathcal{R}(\mathsf{X}, \mathsf{W}_1, \ldots, \mathsf{W}_N)$. Furthermore, $\mathcal{U} = \mathcal{U}_c \sqcup \mathcal{U}_o$ is

[4] Setting $\mathcal{U}_o = \emptyset$ recovers the CoNIZK setting defined in [OB22].

the disjoint union of a set of N_c closed users $\mathcal{U}_c$ and N_o open users $\mathcal{U}_o$ so that $N = N_c + N_o$. A $\mathsf{CoSNIZK}$ is a tuple $(\mathsf{CoSetup}, \mathsf{CoProve}, \mathsf{CoVerify})$:

- $\mathsf{CoSetup}(\lambda, \mathcal{R}) \rightarrow \mathsf{pp}$: On input security parameter λ and relation $\mathcal{R}$, outputs public parameters pp.
- $\mathsf{CoProve}(\mathsf{pp}, \mathsf{X}, \mathsf{W}_1, \ldots, \mathsf{W}_N) \rightarrow \pi$: On input the public parameters, statement X and N private witness shares $\mathsf{W}_1, \ldots, \mathsf{W}_N$, outputs a proof π.
- $\mathsf{CoVerify}(\mathsf{pp}, \mathsf{X}, \pi) \rightarrow 0/1$: On input public parameters pp, statement X, and proof π, returns 0 or 1 indicating reject or accept respectively.

Informally, a $\mathsf{CoSNIZK}$ proof system is secure if it satisfies the following properties. *Completeness*: Honest provers with valid witness produce a valid proof. *Knowledge Soundness*: Only provers that know a valid witness can construct a valid proof. *t-Zero-Knowledge*: Up to t colluding dishonest provers learn nothing about witnesses of the other *closed* users through $\mathsf{CoProve}$, other than the validity of the whole witness.

3.2 Security

Definition 5 formally states the security properties required of a $\mathsf{CoSNIZK}$ proof system[5]. Note, the definition is given in the random oracle model.

Definition 5 (Collaborative, Segregated NIZK). *Let H be a secure hash function. A collaborative, segregated NIZK, or $\mathsf{CoSNIZK}$, for a relation $\mathcal{R}$, secure against t malicious provers is a collaborative proof $\Pi_{\mathsf{CoSNIZK}} := (\mathsf{CoSetup}, \mathsf{CoProve}, \mathsf{CoVerify})$ with the following properties:*

- *Completeness: For all $(\mathsf{X}, \mathsf{W}) \in \mathcal{R}$, the following is negligible in λ:*

$$\Pr\left[\mathsf{CoVerify}^{\mathsf{H}}(\mathsf{pp}, \mathsf{X}, \pi) = 0 \ : \ \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{CoSetup}^{\mathsf{H}}(\lambda, \mathcal{R}) \\ \pi \leftarrow \mathsf{CoProve}^{\mathsf{H}}(\mathsf{pp}, \mathsf{X}, \mathsf{W}) \end{array}\right].$$

- *Knowledge Soundness: For all X, for all efficient provers $\mathcal{U} := \{\mathcal{P}_1^*, \ldots, \mathcal{P}_N^*\}$, there exists an efficient extractor CoExt such that*

$$\Pr\left[(\mathsf{X}, \mathsf{W}) \in \mathcal{R} \ : \ \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{CoSetup}^{\mathsf{H}}(\lambda, \mathcal{R}) \\ \mathsf{W} \leftarrow \mathsf{CoExt}^{\mathsf{H}, \mathcal{U}^{\mathsf{H}}}(\mathsf{pp}, \mathsf{X}) \end{array}\right] \geq \Pr\left[\mathsf{CoVerify}^{\mathsf{H}}(\mathsf{pp}, \mathsf{X}, \pi) = 1 \ : \ \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{CoSetup}^{\mathsf{H}}(\lambda, \mathcal{R}) \\ \pi \leftarrow \mathcal{U}^{\mathsf{H}}(\mathsf{pp}, \mathsf{X}) \end{array}\right] - \epsilon,$$

  for negligible ϵ.
- *t-Zero-Knowledge: For all efficient $\mathcal{A}$ controlling a set $\mathcal{U}_{\mathcal{A}} := \{\mathcal{P}_1, \ldots, \mathcal{P}_k\}$ of $k \leq t$ provers containing at most $N_c - 1$ provers, there exists an efficient simulator Sim such that for all X, W, and for all efficient distinguishers $\mathcal{D}$,*

$$\left|\Pr\left[\mathcal{D}^{\mathsf{H}'}(\mathsf{tr}) = 1 \ : \ \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{CoSetup}^{\mathsf{H}}(\lambda, \mathcal{R}) \\ b \leftarrow \mathcal{R}(\mathsf{X}, \mathsf{W}) \in \{0, 1\} \\ \mathsf{tr} \leftarrow \mathsf{CoSim}^{\mathsf{H}}(\mathsf{pp}, \mathsf{X}, \{\mathsf{W}_j\}_{j \in \mathcal{U}_{\mathcal{A}} \cup (\mathcal{U}_o \setminus \mathcal{U}_{\mathcal{A}})}, b) \end{array}\right] - Pr\left[\mathcal{D}^{\mathsf{H}}(\mathsf{tr}) = 1 \ : \ \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{CoSetup}(\lambda, \mathcal{R}) \\ \mathsf{tr} \leftarrow \mathsf{View}_{\mathcal{A}}^{\mathsf{H}}[\mathsf{X}, \mathsf{W}] \end{array}\right]\right|,$$

[5] Since soundness only considers efficient provers, the protocol is an *argument* of knowledge. Nevertheless, we refer to it as a ‘proof’ for convenience in this work.

is negligible where tr *is a transcript,* $\mathsf{View}^{\mathsf{H}}_{\mathcal{A}}[\mathsf{X}, \mathsf{W}]$ *denotes the view of* $\mathcal{A}$ *when all provers interact with input* X *and* W*.* H' *denotes the (possibly) re-programmed random oracle created by* CoSim*.*

As is true for the collaborative zk-SNARK definitions in [OB22], some properties in Definition 5 differ from their NIZK analogues in a few ways. Importantly, in the distributed setting, prover i could choose its W_i arbitrarily and then later learn if the resulting combined witness is valid. This is unavoidable information learned about the other witness which we model by explicitly providing the validity of the combined witness to the simulator. Thus, only the validity of the final witness is revealed.

Knowledge soundness establishes only *distributed knowledge* in the sense defined by Halpern and Moses in [HM84]. That is, provers could pool their information to determine the N witnesses. It does not however prove that $\mathcal{P}_i$ knows W_i. This is an inherent feature of the non-interactive nature of the proof which obscures the number of provers from the verifier.

The most significant modification, and where our definition generalizes those in [OB22], is in the t-zero-knowledge property. Here, the adversary may corrupt any set of k provers (either open or closed) and must not be able to learn any information about the witnesses held by the remaining honest *closed* provers. We thus provide the simulator with witnesses of all corrupted provers *and* those of any honest open provers. This models that the local zero-knowledge property only holds for the un-corrupted open parties. Again, considering a setup without any open provers, we recover the definition of [OB22].

3.3 Two-Party CoSNIZK from Lattices

We present a two-party (dual-prover) CoSNIZK protocol Π_{CoSNIZK} for the standard lattice relation

$$\mathbf{P}\mathbf{s}_1 = \mathbf{v},$$

defined over R_q for some public matrix $\mathbf{P} \in R_q^{n \times m_1}$, secret $\mathbf{s}_1$[6] $\in R_q^{m_1}$, and pubic $\mathbf{v} \in R_q^n$. We would additionally like to prove something about the norm of $\mathbf{s}_1$. At this point, we note that it suffices to prove the equation $\mathsf{Rot}(\mathbf{P})\vec{\mathbf{s}}_1 = \vec{\mathbf{v}}$ over $\mathbb{Z}_q$, where $\vec{\mathbf{s}}$ and $\vec{\mathbf{v}}$ are the image of $\mathbf{s}_1$ and $\mathbf{v}$ under Φ respectively i.e. their column vectors of coefficients. Indeed, as in the proof system of [LNP22], our first step is always to define the statement/relation over $\mathbb{Z}_q$ and proceed from there. In what follows, it is assumed that $\mathcal{P}_1$ is the open prover and $\mathcal{P}_2$ is closed[7]. We consider that $\mathcal{P}_1$ holds a witness $\mathsf{W}_1 = \mathbf{s}_{11}$ and $\mathcal{P}_2$ holds a witness $\mathsf{W}_2 = \mathbf{s}_{12}$. Moreover we consider the setting in which these are additive shares in the full witness so

[6] The suffix here is to aid in the proof presentation where we distinguish it from the commitment randomness $\mathbf{s}_2$.

[7] This is to aid presentation by removing 'c' and 'o' labels throughout.

that $\mathbf{s}_1 = \mathbf{s}_{11} + \mathbf{s}_{12}$[8]. Then, without loss of generality, and denoting $P := \mathsf{Rot}(\mathbf{P})$, we define a proof system with respect to the following relation.

$$\mathcal{R} := \left\{ \begin{matrix} \mathsf{X} := (P, \vec{\mathbf{v}}), \\ \mathsf{W} := \vec{\mathbf{s}}_1 := \begin{bmatrix} \vec{\mathbf{s}}_{11} \\ \vec{\mathbf{s}}_{12} \end{bmatrix} \end{matrix} \middle| \begin{matrix} P \in \mathbb{Z}_q^{nd \times m_1 d}, \\ \vec{\mathbf{v}} \in \mathbb{Z}_q^{nd}, \vec{\mathbf{s}}_1 \in \mathbb{Z}_q^{m_1 d} \end{matrix} : \begin{matrix} P\vec{\mathbf{s}}_1 = \vec{\mathbf{v}} \wedge \|\vec{\mathbf{s}}_{11}\|_2 = \mathcal{B}_{s_{11}}, \\ \wedge \|\vec{\mathbf{s}}_{12}\|_2 = \mathcal{B}_{\mathbf{s}_{12}} \end{matrix} \right\}. \quad (3)$$

We consider exact norm arguments on the witness since, as shown in [LNP22], one can always apply Lagrange's sum-of-four-squares theorem to create a modified witness and linear relation with exact norm claims whose validity implies the original norm inequalities. In the following protocol description, we will use $\mathbf{A}, \mathbf{a}, a$ for matrices of polynomials, vectors of polynomials, and single polynomials and we will use $A, \vec{\mathbf{a}}, \vec{a}$ when considering the rotation matrix of $\mathbf{A}$, and the coefficient vectors of $\mathbf{a}$ and a respectively.

$\mathsf{CoSetup}(\lambda, \mathcal{R})$*:* We define the common *random* string consisting of uniformly random matrices which are used for the ABDLOP commitment [LNP22]:

$$\mathsf{crs} := (\mathbf{A}_1, \mathbf{A}_2, \mathbf{B}_y, \mathbf{B}_r, \mathbf{b}) \in R_q^{k_{\mathsf{MSIS}} \times m_1} \times R^{k_{\mathsf{MSIS}} \times m_2} \times R^{256/d \times m_2} \times R^{\tau \times k_{\mathsf{MLWE}}} \times R^{m_2},$$

where k_{MSIS} and m_2 are parameters set so that MSIS and MLWE are hard. We are now ready to describe the $\mathsf{CoProve}$ algorithm.

$\mathsf{CoProve}(\mathsf{pp}, \mathsf{X}, \mathsf{W}_1, \mathsf{W}_2)$*:*

Round 1. In this round, the two proves need to create a commitment to the vector $\mathbf{s}_1$ as well as commitments to masking vectors $\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3$ and to a vector $\mathbf{r}$ with constant coefficients equal to zero. This work is split between $\mathcal{P}_1$ and $\mathcal{P}_2$ as follows.

Let $\mathbf{s}_{11}$ and $\mathbf{s}_{12}$ to be $\mathcal{P}_1$ and $\mathcal{P}_2$'s additive shares of $\mathbf{s}_1$ respectively. $\mathcal{P}_1$ then samples $\mathbf{s}_2 \leftarrow \chi^{m_2}$ where χ is a probability distribution on ternary polynomials in R_q. It also samples $\mathbf{y}_i \leftarrow D_{\mathfrak{s}_i}^{m_i}$ for $i = 1, 2$ and $\mathbf{y}_3 \leftarrow D_{\mathfrak{s}_3}^{256}$. Finally it samples a polynomial vector $\mathbf{r} \leftarrow \{x \in R_q : \tilde{x} = 0\}^\tau$ and computes the commitments

$$\begin{aligned} \mathbf{t}_{A1} &:= \mathbf{A}_1\mathbf{s}_{11} + \mathbf{A}_2\mathbf{s}_2, & \mathbf{w} &:= \mathbf{A}_1\mathbf{y}_1 + \mathbf{A}_2\mathbf{y}_2, \\ \mathbf{t}_y &:= \mathbf{B}_y\mathbf{s}_2 + \mathbf{y}_3, & \mathbf{t}_r &:= \mathbf{B}_r\mathbf{s}_2 + \mathbf{r}. \end{aligned}$$

$\mathcal{P}_1$ then sends these to $\mathcal{P}_2$ who computes $\mathbf{t}_A := \mathbf{t}_{A1} + \mathbf{A}_1\mathbf{s}_{12}$. The first message and corresponding challenge are then $a_1 := (\mathbf{t}_A, \mathbf{t}_y, \mathbf{t}_r, \mathbf{w})$ and

$$(\mathfrak{R}_0, \mathfrak{R}_1) := \mathsf{H}(1, \mathsf{crs}, \mathsf{X}, a_1) \in \{0,1\}^{256 \times m_1 d} \times \{0,1\}^{256 \times m_1 d}.$$

[8] Additive shares are the most common kind of shares considered in the distributed setting. We note that e.g. multiplicative shares could be converted to additive shares if needed.

Round 2. $\mathcal{P}_1$ and $\mathcal{P}_2$ now jointly create the response for the approximate proof of shortness for $\mathbf{s}_1$. $\mathcal{P}_2$ begins by passing $\mathcal{P}_1$ the challenge $\mathfrak{R} := \mathfrak{R}_0 - \mathfrak{R}_1$. $\mathcal{P}_1$ then computes $\vec{z}_{31} := \vec{\mathbf{y}}_3 + \mathfrak{R}\vec{\mathbf{s}}_{11}$ and applies rejection sampling on $\vec{z}_{31}$ before passing it to $\mathcal{P}_2$. $\mathcal{P}_2$ then computes $\vec{z}_3 := \vec{z}_{31} + \mathfrak{R}\vec{\mathbf{s}}_{12}$, performing a second round of rejection sampling, returning to the beginning of the round upon rejection. The second message and corresponding challenge are then $a_2 := \vec{z}_3$ and

$$(\gamma_{i,j})_{i\in[\tau],j\in[256+nd+2]} := \mathsf{H}(2, \mathsf{crs}, \mathsf{X}, a_1, a_2) \in \mathbb{Z}_q^{\tau\times(256+nd+2)}.$$

Round 3. The goal of the provers is now to jointly prove the following relations over $\mathbb{Z}_q$:

$$(1)\ \vec{z}_3 = \vec{\mathbf{y}}_3 + \mathfrak{R}\vec{\mathbf{s}}_1,\quad (2)\ P\vec{\mathbf{s}}_1 = \vec{\mathbf{v}},\quad (3)\ \langle\vec{\mathbf{s}}_{11}, \vec{\mathbf{s}}_{11}\rangle = \mathcal{B}^2_{\mathbf{s}_{11}},\quad (4)\ \langle\vec{\mathbf{s}}_{12}, \vec{\mathbf{s}}_{12}\rangle = \mathcal{B}^2_{\mathbf{s}_{12}}.$$

By way of example, let us consider relation (3). Proving this equation over $\mathbb{Z}_q$ is equivalent to proving that the constant coefficient of $\sigma(\mathbf{s}_{11})^\top\mathbf{s}_{11} - \mathcal{B}^2_{\mathbf{s}_{11}}$ is equal to zero. In a similar way, the provers now aim to create a polynomial h_i for which, if its constant coefficient is zero, the relations above hold with high probability over $\mathbb{Z}_q$. This is done as follows. For each $i \in [\tau]$, $\mathcal{P}_1$ creates the polynomial

$$\begin{aligned} h_{i1} := r_i &+ \sum_{j=1}^{256} \gamma_{i,j} \cdot (\sigma(\mathbf{r}_j)^\top \mathbf{s}_{11}) \\ &+ \sum_{j=1}^{nd} \gamma_{i,256+j} \cdot (\sigma(\mathbf{p}_j)^\top \mathbf{s}_{11} - \sigma(\hat{\mathbf{e}}_j)^\top \mathbf{v}) \\ &+ \gamma_{i,256+nd+1} \cdot (\sigma(\mathbf{s}_{11})^\top \mathbf{s}_{11} - \mathcal{B}^2_{\mathbf{s}_{11}}), \end{aligned}$$

where $\mathbf{r}_j$ is the polynomial vector such that its coefficient vector is the j-th row of $\mathfrak{R}$. Similarly, $\mathbf{p}_j$ corresponds to the j'th row of P and $\hat{\mathbf{e}}_j$ is the polynomial vector so that $\Phi(\hat{\mathbf{e}}_j)$ is a unit vector with its j-th element being 1. $\mathcal{P}_1$ then sends h_{i1} to $\mathcal{P}_2$. $\mathcal{P}_2$ then computes

$$\begin{aligned} h_{i2} := &\sum_{j=1}^{256} \gamma_{i,j} \cdot (\sigma(\mathbf{r}_j)^\top \mathbf{s}_{12} + \sigma(\hat{\mathbf{e}}_j)^\top \mathbf{y}_3 - z_j) \\ &+ \sum_{j=1}^{nd} \gamma_{i,256+j} \cdot (\sigma(\mathbf{p}_j)^\top \mathbf{s}_{12} - \sigma(\hat{\mathbf{e}}_j)^\top \mathbf{v}) \\ &+ \gamma_{i,256+nd+2} \cdot (\sigma(\mathbf{s}_{12})^\top \mathbf{s}_{12} - \mathcal{B}^2_{\mathbf{s}_{12}}), \end{aligned}$$

where z_j is simply the j-th component of $\vec{z}_3$. $\mathcal{P}_2$ then computes $h_i := h_{i1} + h_{i2}$ for each $i \in [\tau]$. By definition of $\mathbf{r}$, the constant coefficients of $h_1, ..., h_\tau$ are all zero and the other coefficients are masked by r_i. The third message and the corresponding challenge are then $a_3 := (h_1, \ldots, h_\tau)$ and $\mu := (\mu_i)_{i\in[\tau]} = \mathsf{H}(3, \mathsf{crs}, \mathsf{X}, a_1, a_2, a_3) \in R_q^\tau$.

Round 4. The goal of the proovers is to now jointly prove the τ quadratic equations defined by the h_i above. With high probability, it suffices to prove the following equation, constructed by linearly combining the τ equations

$$\begin{aligned} 0 = \sum_{i=1}^{\tau} \Big(& \sum_{j=1}^{256} \gamma_{i,j} \cdot (\sigma(\mathbf{r}_j)^\top \mathbf{s}_1 + \sigma(\hat{\mathbf{e}}_j)^\top \mathbf{y}_3 - z_j) \\ & + \sum_{j=1}^{nd} \gamma_{i,256+j} \cdot (\sigma(\mathbf{p}_j)^\top \mathbf{s}_1 - \sigma(\hat{\mathbf{e}}_j)^\top \mathbf{v}) \\ & + \gamma_{i,256+nd+1} \cdot (\sigma(\mathbf{s}_{11})^\top \mathbf{s}_{11} - \mathcal{B}^2_{\mathbf{s}_{11}}) \\ & + \gamma_{i,256+nd+2} \cdot (\sigma(\mathbf{s}_{12})^\top \mathbf{s}_{12} - \mathcal{B}^2_{\mathbf{s}_{12}}) \\ & + r_i - h_i \Big) \end{aligned} \tag{4}$$

Let us define

$$\mathbf{B} := \begin{bmatrix} \mathbf{B}_y \\ \mathbf{B}_g \end{bmatrix}, \ \mathbf{t}_B := \begin{bmatrix} \mathbf{t}_y \\ \mathbf{t}_r \end{bmatrix}, \ \mathbf{m} := \begin{bmatrix} \mathbf{y}_3 \\ \mathbf{r} \end{bmatrix}, \hat{\mathbf{s}}_1 := \begin{bmatrix} \mathbf{s}_{11} \\ \sigma(\mathbf{s}_{11}) \\ \mathbf{m} \\ \sigma(\mathbf{m}) \end{bmatrix}, \text{ and } \hat{\mathbf{s}}_2 := \begin{bmatrix} \mathbf{s}_{12} \\ \sigma(\mathbf{s}_{12}) \\ 0 \\ 0 \end{bmatrix}.$$

Further, defining $\hat{\mathbf{s}} := \hat{\mathbf{s}}_1 + \hat{\mathbf{s}}_2$, we note that Equation (4) may be written as

$$\hat{\mathbf{s}}^\top \mathbf{D}_2 \hat{\mathbf{s}} + \mathbf{d}_1^\top \hat{\mathbf{s}} + d_0 = 0,$$

where $\mathbf{D}_2$, $\mathbf{d}_1$, and d_0 are public matrices (see [BLNS23, Section 6] for how they are constructed). Now, we run the sub-protocol for proving a single quadratic equation with automorphisms as described in [LNP22] but we split the work between $\mathcal{P}_1$ and $\mathcal{P}_2$.

$\mathcal{P}_2$ begins by sending $(\hat{\mathbf{s}}_2)^\top \mathbf{D}_2$ and $\mathbf{D}_2 \hat{\mathbf{s}}_2$ to $\mathcal{P}_1$. $\mathcal{P}_1$ can now compute the garbage term

$$f_1 := (\hat{\mathbf{s}}_1)^\top \mathbf{D}_2 \mathbf{y} + (\hat{\mathbf{s}}_2)^\top \mathbf{D}_2 \mathbf{y} + \mathbf{y}^\top \mathbf{D}_2 \hat{\mathbf{s}}_1 + \mathbf{y}^\top \mathbf{D}_2 \hat{\mathbf{s}}_2 + \mathbf{d}_1 \mathbf{y},$$

where $\mathbf{y} := [\mathbf{y}_1 \mid \sigma(\mathbf{y}_1) \mid -\mathbf{B}\mathbf{y}_2 \mid -\sigma(\mathbf{B}\mathbf{y}_2)]^\top$. $\mathcal{P}_1$ can then further create the commitment $t := \mathbf{b}^\top \mathbf{s}_2 + f_1$ to f_1. Then it sets $f_0 := \mathbf{y}^\top \mathbf{D}_2 \mathbf{y} + \mathbf{b}^\top \mathbf{y}_2$. These can then be passed to $\mathcal{P}_2$ who computes the challenge. Thus the fourth message and corresponding challenge are $a_4 := (t, f_0)$ and $c := \mathsf{H}(4, \mathsf{crs}, \mathsf{X}, a_1, a_2, a_3, a_4) \in \mathcal{C}$.

Final round. Given the challenge c, $\mathcal{P}_1$ begins by computing responses $\mathbf{z}_{i1} := c\mathbf{s}_{i1} + \mathbf{y}_i$ for $i = 1, 2$, applying rejection sampling to them and passing them to $\mathcal{P}_2$. The host then computes $\mathbf{z}_i := \mathbf{z}_{i1} + c\mathbf{s}_{i2}$. The final message is then $a_5 := (\mathbf{z}_1, \mathbf{z}_2)$ and the proof output by provers is then

$$\pi := (a_1, a_2, a_3, a_4, a_5).$$

When made non-interactive via the Fiat-Shamir transform, we note that $\mathbf{w}$ need not be sent, nor any of the challenges except c since these are all computable from the other proof elements.

$\mathsf{CoVerify}(\mathsf{pp}, \mathsf{X}, \pi)$*:* Given a proof π the verifier recomputes the corresponding challenges as well as $\mathbf{D}_2$, $\mathbf{d}_1$, and d_0. Denoting

$$\mathbf{z} := \begin{bmatrix} \mathbf{z}_1 \\ \sigma(\mathbf{z}_1) \\ c\mathbf{t}_B - \mathbf{B}\mathbf{z}_2 \\ \sigma(c\mathbf{t}_B - \mathbf{B}\mathbf{z}_2) \end{bmatrix},$$

the verifier outputs 1 if *all* the following relations hold:

$$\begin{cases} \|\mathbf{z}_1\|_2 \overset{?}{\leq} \mathcal{B}_1, \|\mathbf{z}_2\|_2 \overset{?}{\leq} \mathcal{B}_2, \|\vec{z_3}\|_2 \overset{?}{\leq} \mathcal{B}_3, \\ \tilde{h}_i \overset{?}{=} 0 \text{ for } i \in [\tau], \\ \mathbf{A}_1\mathbf{z}_1 + \mathbf{A}_2\mathbf{z}_2 \overset{?}{=} \mathbf{w} + c\mathbf{t}_A, \\ \mathbf{z}^\top \mathbf{D}_2 \mathbf{z} + c\mathbf{d}_1^\top \mathbf{z} + c^2 d_0 - (ct - \mathbf{b}^\top \mathbf{z}_2) \overset{?}{=} f_0, \end{cases}$$

and 0 otherwise.

We now show that $\mathit{\Pi}_{\mathsf{CoSNIZK}}$ is a secure $\mathsf{CoSNIZK}$ in the sense of Definition 5. Since these security properties are both heavily related and rely on those proven in [LNP22], we give only a proof sketch.

Theorem 1. *Let* H *be a secure hash function. Then* $\mathit{\Pi}_{\mathsf{CoSNIZK}} := (\mathsf{CoSetup}, \mathsf{CoProve}, \mathsf{CoVerify})$ *is a complete, knowledge sound, and* 1*-zero-knowledge* $\mathsf{CoSNIZK}$ *for the relation* $\mathcal{R}$*.*

Proof sketch. Completeness follows directly from the completeness of the LNP proof system. Knowledge soundness also follows from the LNP framework: from the perspective of an extractor, properties that hold for a malicious prover also hold for a malicious collection of provers.

Zero knowledge also follows from the LNP proof system since this property is only intended to protect $\mathcal{P}_1$'s witness and one can view $\mathcal{P}_1$'s messages in $\mathsf{CoProve}$ as an LNP transcript for proving knowledge of $\mathbf{s}_{11}$. In the case that $b = 1$ (W is a valid witness), then the LNP zero-knowledge property implies that π can be simulated from X. Thus the adversary's view can be simulated from π and the witness of the corrupt prover ($\mathcal{P}_2$). If $b = 0$ (the witness is invalid) then the adversary's view can be directly simulated from the witness of the corrupt prover and $\perp$. $\square$

4 A New DAA Scheme

4.1 DAA Functionality and Properties

In a DAA scheme there are four main entities: a number of *trusted platform modules* (TPMs), a number of *hosts*, an *issuer* and a number of *verifiers*. We denote TPMs, hosts, and the issuer by $\mathcal{M}$, $\mathcal{H}$, and $\mathcal{I}$ respectively. A TPM and its corresponding host together form a platform which can engage in a join protocol with the issuer who decides whether the platform can become a member. Once a member, TPM and host can jointly creates signatures with respect to basenames bsn. Signing with respect to fresh basenames or $\mathsf{bsn} = \perp$ yield anonymous, unlinkable signatures. That is, any verifier can check, via a deterministic Verify algorithm, that the signature stems from a legitimate platform without learning anything about the identity of the signer. If a platform signs repeatedly using the same basename $\mathsf{bsn} \neq \perp$, it should be publicly checkable, via a deterministic link algorithm, that those signatures stem from the same platform. DAA also supports *key-based revocation* where signatures fail verification if they stem from a TPM whose secret key is registered in a revocation list RL of rogue TPMs.

At a high level, DAA must satisfy the following security and privacy properties. These properties are captured by the formal security model of DAA in [CDL16b] in which the authors define the ideal functionality $\mathcal{F}_{\mathsf{daa}}$ for DAA in the UC framework.

Anonymity: An adversary given two signatures with respect to two different basenames or $\mathsf{bsn} = \perp$ cannot tell whether the signatures originate from the same or two different honest platforms.

(One More) Unforgeability: When the issuer is honest, an adversary controlling n TPMs cannot create more than n unlinkable DAA signatures for the same basename $\mathsf{bsn} \neq \perp$.

Non-Frameability No adversary can create a signature on a message m and basename bsn which links to another signature created by an honest TPM who never signed m w.r.t. bsn.

4.2 The Protocol

We begin by giving a high-level overview of our new DAA scheme suppressing, for now, the artefacts needed for proving its security in the UC model such as session identifiers. In the same vein, we present the join and signing protocols in their complete, interactive form without separating them into sub-stages. This is done to aid the reader's intuition and again we leave the syntactic rearrangements needed for UC compatibility to Section 5. We present the join and signing phases in Figures 2 and 3 respectively.

Setup. Let $q, n, \hat{n}, m$, and N be positive integers such that $q = p_1 p_2$, a product of two primes of the form $p_i = 5 \mod 8$, $p_1 < p_2$. Let d be a power of two [9] and let $\mathfrak{s}$, $\mathfrak{s}_{\mathsf{NTRU}}$ $\mathcal{B}_s$, and $\mathcal{B}_m$ be positive reals. Let $\hat{d}$ be a positive multiple of d. Define the system parameters $\mathsf{sp} = (q, d, n, \hat{n}, m, N, B, U, \mathfrak{s}_{\mathsf{NTRU}}, \mathfrak{s}_{\mathsf{pre}}, \mathcal{B}_s, \mathcal{B}_{\mathsf{tsk}})$. $B \in \mathbb{Z}^{nd \times t}$ is sampled uniformly and used to define the function f as described in Section 2.4 and N is the maximum number of platforms that can join. U is sampled uniformly from $\mathbb{Z}_q^{n^2 d \times 256}$. The issuer's public key is $\mathsf{ipk} = (\mathbf{C}_1, \mathbf{C}_2, \mathbf{h}, \mathsf{bsn}_{\mathcal{I}})$ where $\mathbf{C}_1, \mathbf{C}_2 \in R^{n \times n}$, $\mathbf{h} \in R_q^n$, $\mathsf{bsn}_{\mathcal{I}}$ is basename unique to the issuer, and its secret key is $\mathsf{isk} = \mathbf{B}_{\mathbf{F},\mathbf{g}}$ which is a short basis for the NTRU lattice $\mathcal{L}_{\mathsf{NTRU}}$ defined by $\mathbf{h}$. Note, $\mathbf{h}$, and $\mathbf{B}_{\mathbf{F},\mathbf{g}}$ are generated using $\mathsf{NTRU.TrapGen}$ as described in Lemma 1.

[9] for the purposes of this work, the reader can think of d as being 64 or 128.

Join. The TPM samples $\mathbf{e} = (\mathbf{e}_1, \mathbf{e}_2) \leftarrow R_3^n \times R_3^n$ and $e_3 \leftarrow \{0,1\}^{256}$ and sets its secret key $\mathsf{tsk} = (\mathbf{e}_1, \mathbf{e}_2, e_3)$. Next, the TPM computes

$$\mathbf{u}_1 := \mathbf{C}_1\mathbf{e}_1 + \mathbf{C}_2\mathbf{e}_2 \in R_q^n, \tag{5}$$

and sends $\mathbf{u}_1$ along with a proof of knowledge $\pi_{\mathsf{join}} \leftarrow \mathsf{Prove}^{\mathsf{H}}_{\mathsf{join}}(\mathbf{u}_1; (\mathbf{e}_1, \mathbf{e}_2))$ of $(\mathbf{e}_1, \mathbf{e}_2)$ satisfying Equation (5) such that $\|\mathbf{e}_1\|_2, \|\mathbf{e}_2\|_2 \le \mathcal{B}_{\mathsf{tsk}}$. The TPM then computes the value $\mathbf{D} := \mathsf{H}_{R_q^{n\times n}}(\mathsf{bsn}_{\mathcal{I}})$. It then computes the error term $\mathbf{e}' := \mathsf{H}_{R_3^n}(e_3, \mathsf{bsn}_{\mathcal{I}})$ and outputs the pseudonym $\mathsf{nym} = \mathbf{D}\mathbf{e}_1 + \mathbf{e}' \in R_q^n$.

The issuer, upon receiving $(\mathbf{u}_1, \pi_{\mathsf{join}}, \mathsf{nym})$ will check the validity of the proof, aborting if it does not pass. Furthermore, the issuer checks that the TPM hasn't joined previously i.e. is not in the members list ML, and that it isn't on the revocation list RL of rogue TPMs. To execute the first check, we define the $\mathsf{CheckML}$ algorithm, which takes as input the members list ML containing pseudonyms nym_i submitted in previous successful join requests and the TPM's nym. The issuer rejects the TPM's join request if $\|\mathsf{nym} - \mathsf{nym}_i\|_2 \le 2\mathcal{B}_{\mathsf{tsk}}$ for any i. For the revocation check, $\mathsf{CheckRL}$, the issuer checks that for each TPM secret key $\mathbf{e}_1 \in \mathsf{RL}$, $\|\mathsf{nym} - \mathbf{D}\mathbf{e}_1\|_2$ is not less than $\mathcal{B}_{\mathsf{tsk}}$ rejecting the join request if so.

Next the issuer samples $x \xleftarrow{\$} [N]$, computes $f(x) := \mathsf{Coeffs}^{-1}(B \cdot \mathsf{bin}(x)) \in R_q^n$. Next, the issuer constructs $c := \mathbf{1}^\top(f(x) + \mathbf{u}_1) \in R_q$, where $\mathbf{1}$ is the column vector containing n entries of the constant polynomial 1. The issuer then samples a credential as $\mathbf{s} \leftarrow \mathsf{GSampler}(\mathbf{h}, \mathbf{B}_{\mathbf{F},\mathbf{g}}, \mathfrak{s}_{\mathsf{pre}}, c)$, such that $\|\mathbf{s}\|_2 \le \mathcal{B}_s$ and

$$\left[1\ \mathbf{h}^T\right]\mathbf{s} = c \mod R_q, \tag{6}$$

It then passes $(\mathbf{s}, x)$ to the host who checks that $\mathbf{s}$ is short and satisfies Equation (6). It then stores the credential $\mathsf{cred} = (\mathbf{s}, x)$.

$\mathcal{M}_i(\mathsf{tsk} := (\mathbf{e}_1, \mathbf{e}_2, e_3))$		$\mathcal{H}_i(\mathsf{ipk} := (\mathbf{C}_1, \mathbf{C}_2, \mathbf{h}))$		$\mathcal{I}(\mathsf{isk} := \mathbf{B}_{\mathbf{F},\mathbf{g}}, \mathsf{ML})$
$\mathbf{u}_1 := \mathbf{C}_1\mathbf{e}_1 + \mathbf{C}_2\mathbf{e}_2$				
$\pi_{\mathsf{join}} \leftarrow \mathsf{Prove}^{\mathsf{H}}_{\mathsf{join}}(\mathbf{u}_1; (\mathbf{e}_1, \mathbf{e}_2))$				
	$\xrightarrow{\mathbf{u}_1, \pi_{\mathsf{join}}}$		$\xrightarrow{\mathbf{u}_1, \pi_{\mathsf{join}}}$	
				$0/1 \leftarrow \mathsf{CheckML}(\mathsf{nym}, \mathsf{ML})$
				$0/1 \leftarrow \mathsf{CheckRL}(\mathsf{nym}, \mathsf{RL})$
				$0/1 \leftarrow \mathsf{Verify}^{\mathsf{H}}_{\mathsf{join}}(\mathbf{u}_1; \mathbf{C}_1, \mathbf{C}_2)$
				abort if any output 0
				$x \xleftarrow{\$} [N]$
				$c := \mathbf{1}^\top(f(x) + \mathbf{u}_1) \in R_q$
				$\mathbf{s} \leftarrow \mathsf{GSampler}(\mathbf{h}, \mathbf{B}_{\mathbf{F},\mathbf{g}}, \sigma_{\mathsf{cred}}, c)$
		$\mathsf{cred} := (\mathbf{s}, x)$	$\xleftarrow{(\mathbf{s}, x)}$	

Fig. 2. The Join protocol for Π_{LDAA}.

Constructing π_{join}. We now detail the non-interactive zero-knowledge proof framework used by the platform in the join protocol.

It suffices for the TPM to prove knowledge of short $\mathbf{e}_1, \mathbf{e}_2 \in R_q^n$ satisfying the module-LWE instance

$$\begin{bmatrix} \mathbf{C}_2^{-1}\mathbf{C}_1 \ \mathbf{I}_n \end{bmatrix} \begin{bmatrix} \mathbf{e}_1 \\ \mathbf{e}_2 \end{bmatrix} = \mathbf{C}_2^{-1}\mathbf{u}_1. \tag{7}$$

More precisely, the TPM proves that $\|\mathbf{e}_1\|_2, \|\mathbf{e}_2\|_2 \leq \mathcal{B}_{\mathsf{tsk}}$ for $\mathcal{B}_{\mathsf{tsk}} = \sqrt{nd}$. Prove this relation is exactly dealt with in [LNP22, Section 6.2] where the authors show how one only needs to commit to $\mathbf{e}_1$ and prove that

$$\left\| \begin{bmatrix} \mathbf{e}_1 \\ \mathbf{C}_2^{-1}\mathbf{C}_1\mathbf{e}_1 - \mathbf{C}_2^{-1}\mathbf{u} \end{bmatrix} \right\|_2 = \left\| \begin{bmatrix} \mathbf{I}_n \\ \mathbf{C}_2^{-1}\mathbf{C}_1 \end{bmatrix} \mathbf{e}_1 - \begin{bmatrix} \mathbf{0} \\ \mathbf{C}_2^{-1}\mathbf{u}_1 \end{bmatrix} \right\|_2 \leq \mathcal{B}_{\mathsf{tsk}}.$$

We adopt the same parameters as given in [LNP22, Section 6.2] which yield a proof size for π_{join} of 14.4KB. Finally, the proof inherits the security properties of correctness, zero-knowledge, and knowledge soundness. For now, we refer to [BLNS23, Lemmata 5.5 and 5.6] and [LNP22, Theorem B.7] for the formal statement of these results respectively. We will state these properties explicitly for π_{sign} in the next section.

Sign. To sign a message m with respect to a basename bsn (if $\mathsf{bsn} = \perp$, the TPM chooses one uniformly at random from the space of basenames, $\mathfrak{B}$), the TPM creates the value $\mathbf{D} := \mathsf{H}_{R_q^{n \times n}}(\mathsf{bsn})$. It then computes the error term $\mathbf{e}' := \mathsf{H}_{R_3^n}(e_3, \mathsf{bsn})$ and outputs the pseudonym $\mathsf{nym} = \mathbf{D}\mathbf{e}_1 + \mathbf{e}' \in R_q^n$.

At this point, the platform (TPM and Host) collectively know short $\mathbf{e}_1, \mathbf{e}_2$, $\mathbf{e}', \mathbf{s}$, and x, satisfying

$$\mathsf{nym} = \mathbf{D}\mathbf{e}_1 + \mathbf{e}', \tag{8}$$

$$\begin{bmatrix} 1 \ \mathbf{h}^T \end{bmatrix} \mathbf{s} = \mathbf{1}^\top \left(f(x) + \mathbf{C}_1\mathbf{e}_1 + \mathbf{C}_2\mathbf{e}_2 \right). \tag{9}$$

In order to create a signature on m with respect to bsn, the platform creates a proof of knowledge $\pi_{\mathsf{sign}} \leftarrow \mathit{\Pi}^{\mathsf{H}}_{\mathsf{CoSNIZK}}((\mathsf{nym}, \mathbf{D}, \mathbf{h}, \mathbf{C}_1, \mathbf{C}_2, f), (\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}', \mathbf{s}, x))$ of these short elements satisfying Equations (8) and (9) (where the message is included in the hash input of the non-interactive proof). That is, the TPM and host engage in a collaborative, segregated non-interactive proof (CoSNIZK) procedure. In particular, they follow the two-party CoSNIZK design presentedx in Section 3.3 where the TPM plays the role of a 'closed' prover ($\mathcal{P}_1$) and the host is an open prover ($\mathcal{P}_2$). The host then outputs the signature $\mathsf{sig} = (\mathsf{nym}, \mathsf{bsn}, \pi_{\mathsf{sign}})$. We give more details of the corresponding relation and proof below.

Verify. Given a message m, signature $\mathsf{sig} = (\mathsf{nym}, \mathsf{bsn}, \pi_{\mathsf{sign}})$, and issuer public key ipk, the verifier checks the proof π_{sign} (recovering $\mathbf{D}$ from bsn). Further, the verifier checks the signature against the list of revoked TPMs by running $\mathsf{CheckRL}(\mathsf{nym}, \mathsf{RL})$ as described above. If all checks pass, it outputs 1 and 0 otherwise.

$\mathcal{M}_i(\mathsf{tsk} := (\mathbf{e}_1, \mathbf{e}_2, e_3), m, \mathsf{bsn})$		$\mathcal{H}_i(\mathsf{cred} := (\mathbf{s}, x), m, \mathsf{bsn})$
if $\mathsf{bsn} = \perp, \mathsf{bsn} \overset{\$}{\leftarrow} \mathfrak{B}$		
$\mathbf{D} := \mathsf{H}_{R_q^{n \times n}}(\mathsf{bsn})$		
$\mathbf{e}' := \mathsf{H}_{R_3^n}(e_3, \mathsf{bsn})$		
$\mathsf{nym} := \mathbf{D}\mathbf{e}_1 + \mathbf{e}'$		
	$\xrightarrow{\mathsf{nym}}$	
	$\langle \mathsf{Prove}^{\mathsf{H}}_{\mathsf{sign}}((\mathsf{nym}, \mathbf{D}, \mathbf{h}, \mathbf{C}_1, \mathbf{C}_2), (\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}', \mathbf{s}, x)) \rangle$	
		$\mathsf{sig} := (\mathsf{nym}, \mathsf{bsn}, \pi_{\mathsf{sign}})$

Fig. 3. The Sign protocol for Π_{LDAA}. The penultimate line denotes the interaction between the TPM and Host to produce π_{sign} (see below for π_{sign} construction details).

Link. On input two verifying signatures $(\mu_1, \mathsf{bsn}, \mathsf{sig}_1)$ and $(\mu_2, \mathsf{bsn}, \mathsf{sig}_2)$ on messages μ_1 and μ_2 for the same basename, the linking algorithm proceeds as follows:

1. If $\mathsf{nym}_1 = \mathsf{nym}_2$, it outputs 1
2. Otherwise it checks that $\|\mathsf{nym}_1 - \mathsf{nym}_2\|_2 \leq 2\mathcal{B}_{\mathsf{tsk}}$, outputting 1 if so. Otherwise it returns 0.[10]

Constructing π_{sign}. Here, we give provide some further details of how the platform uses the $\mathsf{CoSNIZK}$ protocol in Section 3.3 to generate π_{sign}. As mentioned above, the TPM plays the role of a closed prover (no information about its witness share is learned by the host or verifier) and the host is an open prover (its witness share, the credential, can be learnt by the TPM but not by the verifier).

We begin by determining the statement and witness to be proven, and thus the relation which defined the $\mathsf{CoSNIZK}$.
We first combine Equations (8) and (9) into an equation of the form defined by the relation in Equation (2). It is easily derived that, after some rearrangement, Equations (8) and (9) can be written as one equation over $\mathbb{Z}_q$ of the form

$$\begin{bmatrix} I_d & \Phi(\mathbf{h}^\top) & -J_1 & -J_2 & -J_3 & 0 \\ 0 & 0 & 0 & \mathsf{Rot}(\mathbf{D}) & 0 & I_{nd} \end{bmatrix} \begin{bmatrix} \vec{\mathbf{s}} \\ \vec{u} \\ \vec{\mathbf{e_1}} \\ \vec{\mathbf{e_2}} \\ \vec{\mathbf{e}'} \end{bmatrix} = \begin{bmatrix} \vec{0} \\ \vec{\mathsf{nym}} \end{bmatrix} \in \mathbb{Z}_q^{(n+1)d}, \tag{10}$$

where $J_1 \in \mathbb{Z}_q^{d \times t}$ and $J_2, J_3 \in \mathbb{Z}_q^{d \times nd}$ are matrices resulting from the product of $\mathbf{1}^\top$ with $(f(x) + \mathbf{C}_1\mathbf{e}_1 + \mathbf{C}_2\mathbf{e}_2)$ on the righ-hand side of Equation (9). Note that the apparent mismatch of dimensions on the left hand side of Equation (10) because the first two columns of the public matrix multiply with $\mathbf{s}$ (which has length 2, see Lemma 1 for the NTRU trapdoor structure). As well as the linear

[10] Though Step 1 is subsumed by Step 2, equality is much easier to check and so this flow makes linking more efficient for honestly generated signatures.

relation Equation (10), we must further prove the following quadratic relations: $\|\vec{\mathbf{s}}\|_2 \le \mathcal{B}_s$, $\langle \vec{u}, \vec{u} - 1 \rangle = 0$, and $\|\vec{\mathbf{e}}_1\|_2, \|\vec{\mathbf{e}}_2\|_2, \|\vec{\mathbf{e}}'\|_2 \le \mathcal{B}_{\mathsf{tsk}}$, where $B\vec{u} = \Phi(f(x))$[11]. Note, the two-norm of a ring element can be equivalently expressed as the inner product of its coefficient vector with itself.

We now observe that, by Lagrange's sum-of-four-squares Theorem, there exists $a := (a_1, a_2, a_3, a_4, 0, \ldots, 0)^\top \in \mathbb{Z}_q^d$ such that $\left\|[\vec{\mathbf{s}} \mid \vec{a}]^\top\right\|_2 = \mathcal{B}_s$. Let us redefine, via abuse of notation, $\vec{\mathbf{s}} := \left[\vec{\mathbf{s}}\ \vec{a}\right]^\top$. A similar elongation and reassignment can be performed on $\vec{\mathbf{e}}_1, \vec{\mathbf{e}}_2$, and $\vec{\mathbf{e}}'$. Further, note that by inserting columns of zeros in the correct places to first matrix on the left-hand side of Equation (10), we can replace the secret vectors in the second matrix with their elongated versions whilst preserving the linear relation. The advantage now is that all norm bounds (quadratic relations) to be proven are *exact* rather than inequalities. This makes the proof structure much cleaner. Finally, we extend the vector $\vec{u}$ with zeros to have length d so that the whole secret vector has length a multiple of d. Again we insert $(d - t)$ columns of zeros into the public matrix to preserve the original equation. At this point, let us note the length of the secret vector in Equation (10): $(3n + 1)d + (\hat{n} + 1)\hat{d}$. In the following, we define $m_1 := (3n + 1)d + (\hat{n} + 1)\hat{d}/d$, where we assume $\hat{d}$ is a multiple of d. Thus, m_1 is the number of ring elements in the secret vector.

In order to refer to objects in Equation (10), let us label the matrices and vectors (left to right) as P, $\mathbf{s}_1$, and $\mathbf{v}$ respectively. At this point the proof proceeds exactly as described in Section 3.3. To help the reader, we fill in a few details of selected proof rounds to help connect that proof flow with the particular relation considered here.

Round 1. Let us define $\mathbf{s}_1^{(t)}$ and $\mathbf{s}_1^{(h)}$ to be the TPM and Host's additive shares of $\mathbf{s}_1$ respectively so that $\mathbf{s}_1^{(t)} := [\mathbf{0} \mid \mathbf{0} \mid \mathbf{e}_1 \mid \mathbf{e}_2 \mid \mathbf{e}']^\top$ and $\mathbf{s}_1^{(h)} := [\mathbf{s} \mid \mathbf{u} \mid \mathbf{0} \mid \mathbf{0} \mid \mathbf{0}]^\top$, where $\mathbf{u} := \Phi^{-1}(\vec{u})$.

Round 3. Here, the goal of the platform is now to jointly prove the following relations over $\mathbb{Z}_q$:

$$\begin{aligned}&(1)\ \vec{z}_3 = \vec{\mathbf{y}}_3 + \mathfrak{R}\vec{\mathbf{s}}_1, \quad (2)\ P\vec{\mathbf{s}}_1 = \vec{\mathbf{v}}, \quad (3)\ \langle \vec{\mathbf{s}}, \vec{\mathbf{s}} \rangle = \mathcal{B}_s^2, \quad (4)\ \langle \vec{u}, \vec{u} - \vec{1} \rangle = 0,\\ &(5)\ \langle \vec{e}_1, \vec{e}_1 \rangle = \mathcal{B}_{\mathsf{tsk}}^2, \qquad (6)\ \langle \vec{e}_2, \vec{e}_2 \rangle = \mathcal{B}_{\mathsf{tsk}}^2, \qquad (7)\ \langle \vec{e}', \vec{e}' \rangle = \mathcal{B}_{\mathsf{tsk}}^2\end{aligned}$$

The corresponding τ equations created by the TPM are, for $i \in [\tau]$

$$\begin{aligned}h_i^{(t)} &:= r_i + \sum_{j=1}^{256} \gamma_{i,j} \cdot (\sigma(\mathbf{r}_j)^\top \mathbf{s}_1^{(t)} + \sigma(\hat{\mathbf{e}}_j)^\top \mathbf{y}_3 - z_j^{(t)})\\ &+ \sum_{j=1}^{(n+1)d} \gamma_{i,256+j} \cdot (\sigma(\mathbf{p}_j)^\top \mathbf{s}_1^{(t)} - \sigma(\hat{\mathbf{e}}_j)^\top \mathbf{v}) + \gamma_{i,256+(n+1)d+3} \cdot (\sigma(\mathbf{e}_1)^\top \mathbf{e}_1 - \mathcal{B}_{\mathsf{tsk}}^2)\end{aligned}$$

[11] Note that proving knowledge of a preimage of $f(x) = \mathbf{y}$ is equivalent to proving knowledge of a binary $\vec{u}$ such that $B\vec{u} = \Phi(\mathbf{y})$.

$$+ \gamma_{i,256+(n+1)d+4} \cdot (\sigma(\mathbf{e}_2)^\top \mathbf{e}_2 - \mathcal{B}^2_{\mathsf{tsk}}) + \gamma_{i,256+(n+)d+5} \cdot (\sigma(\mathbf{e}')^\top \mathbf{e}' - \mathcal{B}^2_{\mathsf{tsk}})$$

The host then computes the following before combining with $h_i^{(t)}$ and completing the round.

$$\begin{aligned}
h_i^{(h)} &:= \sum_{j=1}^{256} \gamma_{i,j} \cdot (\sigma(\mathbf{r}_j)^\top \mathbf{s}_1^{(h)} + \sigma(\hat{\mathbf{e}}_j)^\top \mathbf{y}_3 - z_j^{(h)}) \\
&+ \sum_{j=1}^{(n+1)d} \gamma_{i,256+j} \cdot (\sigma(\mathbf{p}_j)^\top \mathbf{s}_1^{(h)} - \sigma(\hat{\mathbf{e}}_j)^\top \mathbf{v}) + \gamma_{i,256,(n+1)d+1} \cdot (\sigma(\mathbf{s})^\top \mathbf{s} - \mathcal{B}_s^2) \\
&+ \gamma_{i,256,(n+1)d+2} \cdot (\sigma(\mathbf{u})^\top (\mathbf{u} - \mathbf{1})).
\end{aligned}$$

Round 4. The goal of the platform is to now jointly prove the τ quadratic equations defined by the h_i above. With high probability, it suffices to prove the following equation, constructed by linearly combining the τ equations

$$\begin{aligned}
0 = \sum_{i=1}^{\tau} \Big(&\sum_{j=1}^{256} \gamma_{i,j} \cdot (\sigma(\mathbf{r}_j)^\top \mathbf{s}_1 + \sigma(\hat{\mathbf{e}}_j)^\top \mathbf{y}_3 - z_j) \\
&+ \sum_{j=1}^{(n+1)d} \gamma_{i,256+j} \cdot (\sigma(\mathbf{p}_j)^\top \mathbf{s}_1 - \sigma(\hat{\mathbf{e}}_j)^\top \mathbf{v}) + \gamma_{i,256,(n+1)d+1} \cdot (\sigma(\mathbf{s})^\top \mathbf{s} - \mathcal{B}_s^2) \\
&+ \gamma_{i,256,(n+1)d+2} \cdot (\sigma(\mathbf{u})^\top (\mathbf{u} - \mathbf{1})) + \gamma_{i,256,(n+1)d+3} \cdot (\sigma(\mathbf{e}_1)^\top \mathbf{e}_1 - \mathcal{B}^2_{\mathsf{tsk}}) \\
&+ \gamma_{i,256,(n+1)d+4} \cdot (\sigma(\mathbf{e}_2)^\top \mathbf{e}_2 - \mathcal{B}^2_{\mathsf{tsk}}) + \gamma_{i,256,(n+)d+5} \cdot (\sigma(\mathbf{e}')^\top \mathbf{e}' - \mathcal{B}^2_{\mathsf{tsk}}) \\
&+ r_i - h_i \Big)
\end{aligned} \tag{11}$$

Security. We now give lemmas for the security of the $\mathsf{CoSNIZK}$ π_{sign}. Though its security is already inherited from Theorem 1, we present lemmas which make explicit the parameter constraints which must hold for the security of π_{sign} to inherit its $\mathsf{CoSNIZK}$ security from [LNP22]. These results are due to [LNP22] and so we only state them here. For their proof we refer the reader to the aforementioned work. Since we assume an honest TPM in this work, any added security concerns that might arise from the splitting of proof work between TPM and host could only come from elements sent from the TPM to the host. It is easy to see that, if parameters are chosen according to the following lemmas, no information about the TPM secret key is leaked to the host. Observe that if one considers only objects passed from TPM to host, they constitute an 'LNP transcript' for a proof of only the TPMs share which is thus hidden from the host. Furthermore, any objects passed from the host to the TPM need not hide the credential from the TPM. Recall that the storage of the credential on the host is purely to reduce the storage and computational requirements of the TPM. Moreover, the TPM need not check that the host is using the correct credential since if not, resulting DAA signatures will not verify.

Lemma 2 (Correctness of underlying LNP proof). *Let $\alpha_1, \alpha_2, \alpha_3 \in O(\sqrt{\lambda})$, $d \in O(\lambda)$. Recall ν, ω_{max} from Section 2.5 and define the following standard deviations:*

$$\mathfrak{s}_1 := \alpha_1 \nu \|\mathbf{s}_1\|_2,\ \mathfrak{s}_2 := \alpha_2 \nu \sqrt{m_2 d},\ \mathfrak{s}_3 := \alpha_3 \|\mathbf{s}_1\|_2 \omega_{max},$$

and the corresponding rejection rates for $i \in \{1,2,3\}$:

$$M_i := \exp\left(\sqrt{\frac{2(\lambda+1)}{\log e}} \cdot \frac{1}{\alpha_i} + \frac{1}{2\alpha_i^2}\right).$$

Then the underlying LNP proof system is correct.

Lemma 3 (Zero-Knowledge of underlying LNP proof). *$\alpha_1, \alpha_2, \alpha_3 \in O(\sqrt{\lambda})$, $d \in O(\lambda)$. Recall ν, ω_{max} from Section 2.5 and define the following standard deviations:*

$$\mathfrak{s}_1 := \alpha_1 \nu \|\mathbf{s}_1\|_2,\ \mathfrak{s}_2 := \alpha_2 \nu \sqrt{m_2 d},\ \mathfrak{s}_3 := \alpha_3 \|\mathbf{s}_1\|_2 \omega_{max},$$

and the corresponding rejection rates for $i \in \{1,2,3\}$:

$$M_i := \exp\left(\sqrt{\frac{2(\lambda+1)}{\log e}} \cdot \frac{1}{\alpha_i} + \frac{1}{2\alpha_i^2}\right).$$

Then the underlying LNP proof system is zero-knowledge under the $\mathsf{MLWE}_{(k_{\mathsf{MSIS}}+256/d+\tau+1),m_2,\chi,q}$ assumption.

Finally, we come to soundness. The proof of this lemma can be found in [LNP22, Theorem B.7].

Lemma 4 (Knowledge Soundness of underlying LNP proof). *Let $B := 8\nu\sqrt{\mathcal{B}_1^2 + \mathcal{B}_2^2}$ and*

$$q > \max\left(b, 16 m_1 d \mathcal{B}_3, \frac{2}{\omega_{max}(\lambda)} \mathcal{B}_3^2\right).$$

Then there exists an extractor $\mathcal{E}$ with the following properties. Given oracle access to any (potentially dishonest) prover $\mathcal{P}^*$, which outputs a valid LNP proof with probability ϵ, the extractor runs in expected polynomial (in statement) number of steps and with probability at least

$$\epsilon - \left(\frac{2}{|\mathcal{C}|} + p_1^{-d/2} + p_1^{-\lambda} + 2^{-128}\right),$$

it either outputs $\mathbf{s}_1 \in R_q^{m_1}$ such that $P\vec{\mathbf{s}_1} = \vec{\mathbf{v}}$ and $\|\mathbf{s}\|_2 = B$, or an $\mathsf{MSIS}_{k_{\mathsf{MSIS}},m_1+m_2,B}$ solution for the matrix $[\mathbf{A}_1|\mathbf{A}_2]$.

5 Security Proof

We now show that our DAA protocol satisfies the desired DAA security guarantees captured through the ideal functionality $\mathcal{F}_{\mathsf{daa}}$ [CDL16b]. Before presenting our proof sketch, we first discuss how the protocols and algorithms presented in Section 4 have to be "UC-fied".

5.1 UC Wrapper for our Protocol

To date, the only sound security notion for DAA is an ideal functionality in the Universal Composability framework. Describing protocols in the UC framework requires some extra care to include session identifiers, explicit party inputs in interactive protocols, as well as to reflect the abstract modeling of keys and secure channels. We start by describing the necessary sub-functionalities our protocol relies on, and then discuss how to map our protocols to the interfaces required by the DAA ideal functionality $\mathcal{F}_{\mathsf{daa}}$. In the following discussion, the reader may find it useful to refer to the full $\mathcal{F}_{\mathsf{daa}}$ functionality which we enclose in Appendix A.

Sub-Functionalities. We assume a common reference string functionality $\mathcal{F}_{\mathsf{crs}}^{D}$ and a certificate authority functionality $\mathcal{F}_{\mathsf{ca}}$ available to all parties. The later allows the issuer to register his public key, and $\mathcal{F}_{\mathsf{crs}}^{D}$ is used to provide all entities with the system parameters.

For the communication between TPM and issuer (via the host) in the join protocol, we use the semi-authenticated channel $\mathcal{F}_{auth^*}$ introduced in [CDL16b]. For all communication between a host and TPM we assume the secure message transmission functionality $\mathcal{F}_{smt}$ (enabling authenticated and encrypted communication). In practice, $\mathcal{F}_{smt}$ is naturally guaranteed by the physical proximity of the host and TPM forming the platform.

In the description of the protocol, we assume that parties call $\mathcal{F}_{\mathsf{crs}}^{D}$ and $\mathcal{F}_{\mathsf{ca}}$ to retrieve the public key of another party. Further, if any of the checks in the protocol fails, the protocol ends with a failure message $\perp$. The protocol also outputs $\perp$ whenever a party receives an input or message it does not expect (e.g., protocol messages arriving in the wrong order.)

$\mathcal{F}_{\mathsf{daa}}$ *Interfaces.* The $\mathcal{F}_{\mathsf{daa}}$ functionality considers an issuer $\mathcal{I}$ and the platform consisting of a TPM $\mathcal{M}_i$ and a host $\mathcal{H}_i$. In UC, different instances of a protocol are separated through unique session identifiers $\mathsf{sid} = (\mathcal{I}, \mathsf{sid}')$. In the real-world these are mapped to the issuer public key, and all parties use the sid to link their stored key material to the particular issuer.

Setup. $\mathcal{I}$ upon input $(\mathsf{SETUP}, \mathsf{sid})$ generates his key pair $(\mathsf{ipk}, \mathsf{isk})$. It registers the public key $(\mathsf{sid}, \mathsf{ipk})$ at $\mathcal{F}_{\mathsf{ca}}$, stores the secret key as $(\mathsf{sid}, \mathsf{isk})$ and ends with output $(\mathsf{SETUPDONE}, \mathsf{sid})$.

Join. To distinguish several join sessions that might run in parallel, we use a unique sub-session identifier jsid that is given as additional input to all parties. The join protocol starts when $\mathcal{H}_j$ receives the input $(\mathsf{JOIN}, \mathsf{sid}, \mathsf{jsid}, \mathcal{M}_i)$ upon which it triggers $\mathcal{M}_i$ to generate $\mathbf{u}_1$ along with the proof π_{join}, and send it via $\mathcal{F}_{auth^*}$ to $\mathcal{I}$. When $\mathcal{I}$ receives the message it outputs $(\mathsf{JOINPROCEED}, \mathsf{sid}, \mathsf{jsid}, \mathcal{M}_i)$. The join session is complete when the issuer receives an input telling him to proceed with join session jsid, upon which it returns $\mathsf{cred} = (\mathbf{s}, x)$. This explicit interaction with the issuer allows the issuer to perform some additional check to decide whether $\mathcal{M}_i$ is allowed to join. The host stores the credential as $(\mathsf{sid}, \mathcal{M}_i, \mathsf{cred})$ and the TPM stores its secret key as $(\mathsf{sid}, \mathcal{H}_j, \mathsf{tsk})$ i.e., both "remember" with whom they joined. The join ends with $\mathcal{M}_i$ outputting $(\mathsf{JOINCOMPLETE}, \mathsf{sid}, \mathsf{jsid})$.

Sign. Signing is a protocol run between a TPM $\mathcal{M}_i$ and a host $\mathcal{H}_j$. Again, we use a unique sub-session identifier ssid to allow for multiple sign sessions and unique identification of the particular session in the UC interfaces. The host $\mathcal{H}_j$ upon input $(\mathsf{SIGN}, \mathsf{sid}, \mathsf{ssid}, \mathcal{M}_i, m, \mathsf{bsn})$, retrieves its join record $(\mathsf{sid}, \mathcal{M}_i, \mathsf{cred})$ and aborts if no such record is found. It sends $(\mathsf{ssid}, m, \mathsf{bsn})$ to the TPM which then checks that a key record $(\mathsf{sid}, \mathcal{H}_j, \mathsf{tsk})$ exists and outputs $(\mathsf{SIGNPROCEED}, \mathsf{sid}, \mathsf{ssid}, m, \mathsf{bsn})$. The signature is completed when $\mathcal{M}_i$ receives the input $(\mathsf{SIGNPROCEED}, \mathsf{sid}, \mathsf{ssid})$, upon which it computes nym. The explicit input from the TPM is necessary to ensure that the TPM in fact "approved" the attestation of m and bsn. Finally, the host outputs the jointly computed signature as $(\mathsf{SIGNATURE}, \mathsf{sid}, \mathsf{ssid}, \sigma)$.

Verify and Link. Here both algorithms are simply re-labeled as UC interfaces with the ipk being replaced with sid, i.e. both algorithms are made available through interfaces $(\mathsf{VERIFY}, \mathsf{sid}, m, \mathsf{bsn}, \sigma, \mathsf{RL})$ and $(\mathsf{LINK}, \mathsf{sid}, \sigma, m, \sigma', m', \mathsf{bsn})$ respectively.

Theorem 2. *The protocol Π_{LDAA} presented in Section 4 securely realizes $\mathcal{F}_{\mathsf{daa}}$ [CDL16b] in the $(\mathcal{F}_{auth^*}, \mathcal{F}_{\mathsf{ca}}, \mathcal{F}_{smt}, \mathcal{F}_{\mathsf{crs}}^{D})$-hybrid and random oracle model under static corruptions, if the* MLWE*,* MSIS*,* MNTRU*, and* $\mathsf{Int\text{-}MNTRU\text{-}ISIS}_f$ *assumptions hold.*

Owing to the long proof inherent in the UC framework, we include a proof sketch of Theorem 2 in **??**. The proof follows a similar argument to the ones presented in existing DAA works.

6 Performance

In this section we analyze the performance of our new lattice-based DAA scheme.

Let us begin by setting the target bit-security parameter $\lambda = 128$. Throughout this section we aim for an implicit lattice dimension of at least 1024 for all lattice assumptions. As is common for achieving the above security level for this dimension, we use a prime 32-bit modulus $q = 2^{32} - 99$. We then ensure a negligible soundness error as prescribed Lemma 4 by choosing the soundness boosting parameter $\tau = 4$, which ensures that $q^{-\tau}$ and q^{-d} are negligible for all sensible values of d. Let us denote by $m_1^{\mathsf{join}}, m_2^{\mathsf{join}}$ the lengths of the vectors $\mathbf{s}_1$ and $\mathbf{s}_2$ in the join protocol. Similarly, we define $m_1^{\mathsf{join}}, m_2^{\mathsf{join}}$ to be the corresponding quantities in the sign protocol.

We now choose values of $k_{\mathsf{MSIS}}^{\mathsf{Join}}$ and m_2^{join} so that MLWE is hard with parameters $((k_{\mathsf{MSIS}}^{\mathsf{sign}} + 256/d + \tau + 1), m_2^{\mathsf{join}} - (k_{\mathsf{MSIS}}^{\mathsf{sign}} + 256/d + \tau + 1), \chi, q)$ and MSIS is hard w.r.t. $(k_{\mathsf{MSIS}}^{\mathsf{sign}}, m_1^{\mathsf{join}} + m_2^{\mathsf{join}}, B_{\mathsf{join}})$.

Similarly, we choose $k_{\mathsf{MSIS}}^{\mathsf{sign}}$ and m_2^{sign}, so that MLWE is hard w.r.t $((k_{\mathsf{MSIS}}^{\mathsf{sign}} + 256/d + \tau + 1), m_2^{\mathsf{sign}} - (k_{\mathsf{MSIS}}^{\mathsf{sign}} + 256/d + \tau + 1), \chi, q)$ and MSIS is hard w.r.t. $(k_{\mathsf{MSIS}}^{\mathsf{sign}}, m_1^{\mathsf{sign}} + m_2^{\mathsf{sign}}, B_{\mathsf{sign}})$ where χ is the uniform distribution of ternary elements of R and B_{join} and B_{sign} are the two-norms of the vector $[\mathbf{s}_1\ \mathbf{s}_2]^\top$ in each phase.

In order to estimate the hardness of MSIS we use the relation due to Micciancio and Regev [MR09], which states that LLL will recover a short vector of 2-norm $2^{(2\sqrt{d \log_2 q \log_2 \delta})}$. δ is the root Hermite factor and $\delta < 1.0045$ gives rise to at least 128 bits of security.

For MLWE hardness estimation we follow convention by using the estimator [APS15]. This estimates the cost of BKZ conservatively by focusing only on the cost of a single uSVP oracle call, a core operation in BKZ. We assume $8d$ uSVP calls required for a lattice dimension d, following convention.

We must also estimate the security of the issuer's NTRU public key. Here, we use an implicit lattice dimension of $\hat{n} \cdot \hat{d}$ and modulus q. We use standard deviation

$$\sigma_{\mathsf{NTRU}} = \gamma \cdot \frac{1}{\sqrt{\hat{d}(\hat{n} + 2 - i)}} \cdot q^{1/(\hat{d}+1)}, \tag{12}$$

for constructing the i-th row of $\mathbf{h} := \mathbf{F}^{-1} \cdot \mathbf{g}$ as prescribed by the pre-image sampling techniques of [CPS+20], which requires $\gamma = 1.24$ for $\hat{d} = 3$. We then run the estimator of [Dv21] for the hardness of NTRU with these parameters. Note, $\hat{n} > 3$ would lead us into so-called 'overstretched' parameters since larger $\hat{n}$ forces smaller NTRU keys by Equation (12). We set the standard deviations σ_1, σ_2 and σ_3 needed for the correctness and zero-knowledge properties of the LNP proof system, taking $\sigma_1 := \nu\|\mathbf{s}_1\|_2$, $\sigma_2 := \nu\sqrt{m_2 d}$, $\sigma_3 := \|\mathbf{s}_1\|_2 \omega_{\max}$.

Finally, we set the bounds $\mathcal{B}_1$, $\mathcal{B}_2$, and $\mathcal{B}_3$ for the verification checks using the standard tail bound of $\sigma\sqrt{2d}$ for the two-norm of an elements of length d, sampled from a discrete Gaussian of parameter σ. The resulting parameter set is shown in Table 2.

Total sizes. We now compute the total communication cost for Π_{LDAA}. Beginning with the proofs, we inherit the proof size for π_{join} from [LNP22, Section 6.2] which yield a proof size for π_{join} of 14.4KB, where one can use their Huffman encoding technique on $\mathbf{z}_1$ since the committed vector $\mathbf{e}_1$ can be seen as coming from a discrete Gaussian on parameter $\sqrt{2/3}$ (for ternary $\mathbf{e}_1$).

Next, using the parameters of Table 2, we have that the size of π_{sign} is 33.6KB. A signature also comprises a pseudonym nym and basename of size $8d \log q$ and 128 bits respectively. Thus, the final signature size is 37.7 KB.

Scheme	TPM key size	signature size
[CKLL19]	3KB	$>$ 2MB
Ours	0.77KB	37.7KB

Table 1. Key and signatures sizes for the previous state-of-the-art lattice DAA scheme [CKLL19] as compared to this work.

parameters	description	value
λ	security parameter	128
q	modulus	$2^{32} - 99$
d	ring dimension for R	128
$\hat{d}$	ring dimension for NTRU ring $\hat{R}$	384
ξ	max coeff. of chal. space element	2
ν	k-bound on chal. space element	59
τ	# gargage terms r_j for boosting soundness	4
m_2	len. of $\mathbf{s}_2$ in ABDlOP commitment	25
$k_{\mathsf{MSIS}}^{\mathsf{join}}$	height of $\mathbf{A}_1, \mathbf{A}_2$ in ABDLOP for join	9
$k_{\mathsf{MSIS}}^{\mathsf{sign}}$	height of $\mathbf{A}_1, \mathbf{A}_2$ in ABDLOP for sign	9
χ	distribution from which $\mathbf{s}_2$ is sampled	$\mathcal{B}_{S_1}$
N	max # of platforms that can join	2^{40}
n	dim. of TPM keys $\mathbf{e}_1, \mathbf{e}_2 \in R_q$	8
γ	Gram-Schmidt slack for falcon sampling	1.5
$\hat{n}$	dimension of NTRU public key $\mathbf{h}$	3
$\mathfrak{s}_{\mathsf{NTRU}}$	s.d. for sampling NTRU trapdoor	8.1
m_1^{join}	len. of witness $\mathbf{s}_1$ in join	8
m_1^{sign}	len. of witness $\mathbf{s}_1$ in sign	34
$\omega_{\max}(128)$	approx. rang proof param.	$\sqrt{337}$
$\mathfrak{s}_1^{\mathsf{join}}$	s.d. for sampling y_1 in join	4480
$\mathfrak{s}_2^{\mathsf{join}}$	s.d. for sampling y_2 in join	7920
$\mathfrak{s}_3^{\mathsf{join}}$	s.d. for sampling y_3 in join	587
$\mathfrak{s}_1^{\mathsf{sign}}$	s.d. for sampling y_1 in sign	1929375
$\mathfrak{s}_2^{\mathsf{sign}}$	s.d. for sampling y_2 in sign	7920
$\mathfrak{s}_3^{\mathsf{sign}}$	s.d. for sampling y_3 in sign	252990
$\mathcal{B}_1^{\mathsf{join}}$	verification bound for $\mathbf{z}_1$ in join	202742
$\mathcal{B}_2^{\mathsf{join}}$	verification bound for $\mathbf{z}_2$ in join	633568
$\mathcal{B}_3^{\mathsf{join}}$	verification bound for $\mathbf{z}_3$ in join	$1.7\sqrt{256}$
$\mathcal{B}_1^{\mathsf{sign}}$	verification bound for $\mathbf{z}_1$ in sign	87313590
$\mathcal{B}_2^{\mathsf{sign}}$	verification bound for $\mathbf{z}_2$ in sign	633568
$\mathcal{B}_3^{\mathsf{sign}}$	verification bound for $\mathbf{z}_3$ in sign	$1.7\sqrt{256}$

Table 2. Example parameters for Π_{LDAA}.

Comparison with existing works. Table 1 displays the TPM key sizes and DAA signature sizes for our work and we include the estimates provided for the same quantities in [CKLL19] which represents the current state-of-the-art. Furthermore, we note that the signature size provided in [CKLL19] is a very rough lower bound since the authors calculate this figure by arguing that 'most polynomials in R_q have coefficients smaller than q'. Our numbers are based on the actual storage needed for fully-fledged ring elements. All previous works give only asymptotic parameters.

Thus, our work represents a four-fold reduction in TPM key size and importantly a reduction by 1.5 orders of magnitude (53 times) in signature size over the state-of-the-art.

7 Future Work

An obvious next step would be to implement our proposed DAA protocol to asses its practical efficiency. At present, the TPM specification [CCD+17] provides interfaces optimized for operations over classical groups (e.g. group exponentiation). The EU Horizon2020 FutureTPM project [Con18] has ongoing work to design a specification appropriate for post-quantum DAA designs to be efficiently implemented and we intend to inform that process with the design in this work. Unfortunately it is hard to assess the running time of any design from post-quantum assumptions owing to the lack of an updated specification but a proof-of concept implementation would be a good first step.

References

APS15. Martin R Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. *Journal of Mathematical Cryptology*, 9(3):169–203, 2015. 24

BCC04. Ernest F. Brickell, Jan Camenisch, and Liqun Chen. Direct anonymous attestation. In Vijayalakshmi Atluri, Birgit Pfitzmann, and Patrick McDaniel, editors, *ACM CCS 2004*, pages 132–145. ACM Press, October 2004. 2

BCL08. Ernie Brickell, Liqun Chen, and Jiangtao Li. Simplified security notions of direct anonymous attestation and a concrete scheme from pairings. Cryptology ePrint Archive, Report 2008/104, 2008. https://eprint.iacr.org/2008/104. 2

BDK$^+$17. Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, and Damien Stehlé. CRYSTALS – Kyber: a CCA-secure module-lattice-based KEM. Cryptology ePrint Archive, Report 2017/634, 2017. https://eprint.iacr.org/2017/634. 2

BFG$^+$11. D. Bernhard, G. Fuchsbauer, E. Ghadafi, N.P. Smart, and B. Warinschi. Anonymous attestation with user-controlled linkability. Cryptology ePrint Archive, Report 2011/658, 2011. https://eprint.iacr.org/2011/658. 2

BHK$^+$19. Daniel J. Bernstein, Andreas Hülsing, Stefan Kölbl, Ruben Niederhagen, Joost Rijneveld, and Peter Schwabe. The SPHINCS$^+$ signature framework. In *ACM CCS*, pages 2129–2146, 2019. 2

BL07. Ernie Brickell and Jiangtao Li. Enhanced privacy ID: A direct anonymous attestation scheme with enhanced revocation capabilities. Cryptology ePrint Archive, Report 2007/194, 2007. https://eprint.iacr.org/2007/194. 2

BLNS23. Jonathan Bootle, Vadim Lyubashevsky, Ngoc Khanh Nguyen, and Alessandro Sorniotti. A framework for practical anonymous credentials from lattices. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, 2023, Proceedings, Part II*, volume 14082 of *Lecture Notes in Computer Science*, pages 384–417. Springer, 2023. 1, 4, 6, 8, 13, 17

CCD$^+$17. Jan Camenisch, Liqun Chen, Manu Drijvers, Anja Lehmann, David Novick, and Rainer Urian. One TPM to bind them all: Fixing TPM 2.0 for provably secure anonymous attestation. In *2017 IEEE Symposium on Security and Privacy*, pages 901–920. IEEE Computer Society Press, May 2017. 2, 26

CDE$^+$17. J. Camenisch, M. Drijvers, A. Edgington, A. Lehmann, R. Lindemann, and R. Urian. Fido ecdaa algorithm, implementation draft, 2017. 2

CDE$^+$23. Liqun Chen, Changyu Dong, Nada El Kassem, Christopher J. P. Newton, and Yalan Wang. Hash-based direct anonymous attestation. Springer, PQ Crypto, 2023. https://link.springer.com/chapter/10.1007/978-3-031-40003-2_21. 3

CDL15. Jan Camenisch, Manu Drijvers, and Anja Lehmann. Universally composable direct anonymous attestation. Cryptology ePrint Archive, Report 2015/1246, 2015. https://eprint.iacr.org/2015/1246. 2

CDL16a. Jan Camenisch, Manu Drijvers, and Anja Lehmann. Anonymous attestation using the strong Diffie Hellman assumption revisited. Cryptology ePrint Archive, Report 2016/663, 2016. https://eprint.iacr.org/2016/663. 1, 2

CDL16b. Jan Camenisch, Manu Drijvers, and Anja Lehmann. Universally composable direct anonymous attestation. In Chen-Mou Cheng, Kai-Min Chung, Giuseppe Persiano, and Bo-Yin Yang, editors, *PKC 2016, Part II*, volume 9615 of *LNCS*, pages 234–264. Springer, Heidelberg, March 2016. 15, 22, 23, 31, 33

CDL17. Jan Camenisch, Manu Drijvers, and Anja Lehmann. Anonymous attestation with subverted TPMs. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part III*, volume 10403 of *LNCS*, pages 427–461. Springer, Heidelberg, August 2017. 2

CKLL19. Liqun Chen, Nada El Kassem, Anja Lehmann, and Vadim Lyubashevsky. A framework for efficient lattice-based DAA. In Liqun Chen, Chris J. Mitchell, Thanassis Giannetsos, and Daniele Sgandurra, editors, *Proceedings of the 1st ACM Workshop on Workshop on Cyber-Security Arms Race, CYSARM@CCS 2019, London, UK, November 15, 2019*, pages 23–34. ACM, 2019. 3, 4, 25, 26

Con18. FutureTPM Consortium. Future proofing the connected world: A quantum-resistant trusted platform module. https://futuretpm-project.technikon.com/, 2018. Accessed: 2024-05-25. 2, 26

CPS09. L. Chen, D. Page, and N.P. Smart. On the design and implementation of an efficient DAA scheme. Cryptology ePrint Archive, Report 2009/598, 2009. https://eprint.iacr.org/2009/598. 2

CPS$^+$19. Chitchanok Chuengsatiansup, Thomas Prest, Damien Stehlé, Alexandre Wallet, and Keita Xagawa. ModFalcon: compact signatures based on module NTRU lattices. Cryptology ePrint Archive, Report 2019/1456, 2019. https://eprint.iacr.org/2019/1456. 4

CPS$^+$20. Chitchanok Chuengsatiansup, Thomas Prest, Damien Stehlé, Alexandre Wallet, and Keita Xagawa. ModFalcon: Compact signatures based on module-NTRU lattices. In *ASIACCS 20*, pages 853–866. ACM Press, 2020. 5, 6, 24

CTY$^+$21. Liquan Chen, Tianyang Tu, Kunliang Yu, Mengnan Zhao, and Yingchao Wang. V-LDAA: A new lattice-based direct anonymous attestation scheme for vanets system. *Secur. Commun. Networks*, 2021:4660875:1–4660875:13, 2021. 2

CU15. Liqun Chen and Rainer Urian. DAA-A: direct anonymous attestation with attributes. In Mauro Conti, Matthias Schunter, and Ioannis G. Askoxylakis, editors, *Trust and Trustworthy Computing - 8th International Conference, TRUST 2015, Heraklion, Greece, August 24-26, 2015, Proceedings*, volume 9229 of *Lecture Notes in Computer Science*, pages 228–245. Springer, 2015. 2

DFK$^+$23. Sourav Das, Rex Fernando, Ilan Komargodski, Elaine Shi, and Pratik Soni. Distributed-prover interactive proofs. In Guy N. Rothblum and Hoeteck Wee, editors, *Theory of Cryptography - 21st International Conference, TCC 2023, Taipei, Taiwan, November 29 - December 2, 2023, Proceedings, Part I*, volume 14369 of *Lecture Notes in Computer Science*, pages 91–120. Springer, 2023. 3

DKL$^+$18. Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS-Dilithium: A lattice-based digital signature scheme. *IACR TCHES*, 2018(1):238–268, 2018. https://tches.iacr.org/index.php/TCHES/article/view/839. 2

DPP$^+$22. Pankaj Dayama, Arpita Patra, Protik Paul, Nitin Singh, and Dhinakaran Vinayagamurthy. How to prove any NP statement jointly? Efficient distributed-prover zero-knowledge protocols. *PoPETs*, 2022(2):517–556, April 2022. 3

Dv21. Léo Ducas and Wessel P. J. van Woerden. NTRU fatigue: How stretched is overstretched? LNCS, pages 3–32. Springer, Heidelberg, 2021. 24

EB. Jiangtao Li Ernie Brickell, Liqun Chen. *A New Direct Anonymous Attestation Scheme from Bilinear Maps.* 2

ECE$^+$18. Nada El Kassem, Liqun Chen, Rachid El Bansarkhani, Ali El Kaafarani, Jan Camenisch, and Patrick Hough. L-DAA: Lattice-based direct anonymous attestation. Cryptology ePrint Archive, Report 2018/401, 2018. `https://eprint.iacr.org/2018/401`. 3

ECE$^+$19. Nada El Kassem, Liqun Chen, Rachid El Bansarkhani, Ali El Kaafarani, Jan Camenisch, Patrick Hough, Paulo Martins, and Leonel Sousa. More efficient, provably-secure direct anonymous attestation from lattices. Future Generation Computer Systems Journal, 2019. `https://www.sciencedirect.com/science/article/abs/pii/S0167739X19300536`. 3

EE17. Rachid El Bansarkhani and Ali El Kaafarani. Direct anonymous attestation from lattices. Cryptology ePrint Archive, Report 2017/1022, 2017. `https://eprint.iacr.org/2017/1022`. 3

GPV08. Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In Richard E. Ladner and Cynthia Dwork, editors, *40th ACM STOC*, pages 197–206. ACM Press, May 2008. 4

Gro04. Trusted Computing Group. Trusted platform module (tpm) library specification, 2004. 2

Gro14. Trusted Computing Group. Trusted platform module (tpm) library specification, 2014. 2

Gro19. Trusted Computing Group. Tpm 2.0: A brief overview, 2019. 2

HM84. Joseph Y. Halpern and Yoram Moses. Knowledge and common knowledge in a distributed environment. In Robert L. Probert, Nancy A. Lynch, and Nicola Santoro, editors, *3rd ACM PODC*, pages 50–61. ACM, August 1984. 10

ID03. Anca-Andreea Ivan and Yevgeniy Dodis. Proxy cryptography revisited. In *Proceedings of the Network and Distributed System Security Symposium, NDSS 2003, San Diego, California, USA*. The Internet Society, 2003. 3

Int13. International Organization for Standardization (ISO). Iso/iec 20008-2:2013, security techniques, anonymous digital signatures, part 2: Mechanisms using a group public key, 2013. 2

Int15. International Organization for Standardization (ISO). Iso/iec 11889-1:2015, trusted platform module library, part 1: Architecture, Y2015. 2

LNP22. Vadim Lyubashevsky, Ngoc Khanh Nguyen, and Maxime Plançon. Lattice-based zero-knowledge proofs and applications: Shorter, simpler, and more general. LNCS, pages 71–101. Springer, Heidelberg, 2022. 1, 4, 7, 8, 10, 11, 13, 14, 17, 20, 21, 24

MR09. Daniele Micciancio and Oded Regev. *Lattice-based Cryptography*, pages 147–191. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009. 24

Nat22. National Institute of Standards and Technology (NIST). Nist post-quantum cryptography standardization - selected algorithms, 2022. 2

OB22. Alex Ozdemir and Dan Boneh. Experimenting with collaborative zk-SNARKs: Zero-knowledge proofs for distributed secrets. pages 4291–4308. USENIX Association, 2022. 1, 3, 4, 8, 10

Ped91. Torben Pryds Pedersen. Distributed provers with applications to undeniable signatures. In Donald W. Davies, editor, *EUROCRYPT'91*, volume 547 of *LNCS*, pages 221–242. Springer, Heidelberg, April 1991. 3

SRC12. Ben Smyth, Mark D. Ryan, and Liqun Chen. Formal analysis of privacy in direct anonymous attestation schemes. Cryptology ePrint Archive, Report 2012/650, 2012. https://eprint.iacr.org/2012/650. 2

WZC$^+$18. Howard Wu, Wenting Zheng, Alessandro Chiesa, Raluca Ada Popa, and Ion Stoica. DIZK: A distributed zero knowledge proof system. *IACR Cryptol. ePrint Arch.*, page 691, 2018. 3

Xi14. Li Xi. Daa-related apis in tpm2.0 revisited. Cryptology ePrint Archive, Paper 2014/052, 2014. https://eprint.iacr.org/2014/052. 2

A The DAA Ideal Functionality

Here we recall the DAA ideal functionality defined by Camenisch, Drijvers, and Lehmann [CDL16b] in the universal composability model.

Setup

1. `Issuer Setup`. On input $(\mathsf{SETUP}, \mathsf{sid})$ from $\mathcal{I}$,
   - Verify that $\mathsf{sid} = (\mathcal{I}, \mathsf{sid}')$ and output $(\mathsf{SETUP}, \mathsf{sid})$ to $\mathcal{S}$.
2. `Set Algorithms`. On input $(\mathsf{ALG}, \mathsf{sid}, \mathsf{sig}, \mathsf{ver}, \mathsf{link}, \mathsf{identify}, \mathsf{ukgen})$ from $\mathcal{S}$,
   - Check that ver, link, and $\mathsf{identify}$ are deterministic (**i**).
   - Store $(\mathsf{sid}, \mathsf{sig}, \mathsf{ver}, \mathsf{link}, \mathsf{identify}, \mathsf{ukgen})$.

Join

1. `Join Request`. On input $(\mathsf{JOIN}, \mathsf{sid}, \mathsf{jsid}, \mathcal{M}_i)$ from host $\mathcal{H}_j$,
   - Create a join session record $(\mathsf{jsid}, \mathcal{M}_i, \mathcal{H}_j, \mathit{status})$ with $\mathit{status} \leftarrow \mathit{request}$.
   - Output $(\mathsf{JOINSTART}, \mathsf{sid}, \mathsf{jsid}, \mathcal{M}_i, \mathcal{H}_j)$ to $\mathcal{S}$.
2. `Join Request Delivery`. On input $(\mathsf{JOINSTART}, \mathsf{sid}, \mathsf{jsid},)$ from $\mathcal{S}$,
   - Update the session record $(\mathsf{jsid}, \mathcal{M}_i, \mathcal{H}_j, \mathit{status})$ to $\mathit{status} \leftarrow \mathit{delivered}$.
   - Output $(\mathsf{JOINPROCEED}, \mathsf{sid}, \mathsf{jsid}, \mathcal{M}_i)$ to $\mathcal{I}$.
3. `Complete Join`. On input $(\mathsf{JOINPROCEED}, \mathsf{sid}, \mathsf{jsid})$ from $\mathcal{I}$,
   - Update session record $(\mathsf{jsid}, \mathcal{M}_i, \mathcal{H}_j, \mathit{status})$ to $\mathit{status} \leftarrow \mathit{complete}$.
   - Output $(\mathsf{JOINCOMPLETE}, \mathsf{sid}, \mathsf{jsid}, \mathcal{S})$.
4. `KeyGeneration`. On input $(\mathsf{JOINCOMPLETE}, \mathsf{sid}, \mathsf{jsid}, \mathsf{sk})$ from $\mathcal{S}$.
   - Look up record $(\mathsf{jsid}, \mathcal{M}_i, \mathcal{H}_j, \mathit{status})$ with $\mathit{status} = \mathit{complete}$.
   - Abort if $\mathcal{I}$ or $\mathcal{M}_i$ is honest and a record $(\mathcal{M}_i, *, *) \in$ `Members` already exists (**ii**).
   - If $\mathcal{M}_i$ and $\mathcal{H}_j$ are honest, set $\mathsf{sk} \leftarrow \perp$.
   - Else, verify that the provided sk is eligible by checking
   - $\mathsf{CheckSkHonest}(\mathsf{sk}) = 1$ (**iii**) if $\mathcal{H}_j$ is corrupt and $\mathcal{M}_i$ is honest, or
   - $\mathsf{CheckSkCorrupt}(\mathsf{sk} = 1$ (**iv**) if $\mathcal{M}_i$ is corrupt.
   - Insert $\langle \mathcal{M}_i, \mathcal{H}_j, \mathsf{sk} \rangle$ into `Members` and output $(\mathsf{JOINED}, \mathsf{sid}, \mathsf{jsid})$ to $\mathcal{H}_j$.

Fig. 4. The Setup and Join related interfaced of $\mathcal{F}_{\mathsf{daa}}$.

Sign

1. $\texttt{SignRequest}$. On input $(\mathsf{SIGN}, \mathsf{sid}, \mathsf{ssid}, \mathcal{M}_i, \mathsf{m}, \mathsf{bsn})$ from $\mathcal{H}_j$,
   - If $\mathcal{I}$ is honest and no entry $\langle \mathcal{M}_i, \mathcal{H}_j, * \rangle$ exists in $\texttt{Members}$, abort.
   - Create a sign session record $(\mathsf{ssid}, \mathcal{M}_i, \mathcal{H}_j, \mathsf{m}, \mathsf{bsn}, \mathit{status})$ with $\mathit{status} \leftarrow \mathit{request}$.
   - Output $(\mathsf{SIGNSTART}, \mathsf{sid}, \mathsf{ssid}, \mathsf{m}, \mathsf{bsn}, \mathcal{M}_i, \mathcal{H}_j)$ to $\mathcal{S}$.
2. $\texttt{Sign Request Delivery}$. On input $(\mathsf{SIGNSTART}, \mathsf{sid}, \mathsf{ssid})$ from $\mathcal{S}$,
   - Update the session record $(\mathsf{ssid}, \mathcal{M}_i, \mathcal{H}_j, \mathsf{m}, \mathsf{bsn}, \mathit{status})$ to $\mathit{status} \leftarrow \mathit{delivered}$.
   - Output $(\mathsf{SIGNPROCEED}, \mathsf{sid}, \mathsf{ssid}, \mathsf{m}, \mathsf{bsn})$ to $\mathcal{M}_i$.
3. $\texttt{SignProceed}$. On input $(\mathsf{SIGNPROCEED}, \mathsf{sid}, \mathsf{ssid})$ from $\mathcal{M}_i$,
   - Look up record $(\mathsf{ssid}, \mathcal{M}_i, \mathcal{H}_j, \mathsf{m}, \mathsf{bsn}, \mathit{status})$ with $\mathit{status} = \mathit{delivered}$.
   - Output $(\mathsf{SIGNCOMPLETE}, \mathsf{sid}, \mathsf{ssid})$ to $\mathcal{S}$.
4. $\texttt{Signature Generation}$. On input $(\mathsf{SIGNCOMPLETE}, \mathsf{sid}, \mathsf{ssid}, \sigma)$ from $\mathcal{S}$,
   - If $\mathcal{M}_i$ and $\mathcal{H}_j$ are honest, ignore he adversary's signature and internally generate the signature for a fresh or established sk:
   - If $\mathsf{bsn} \neq \perp$, retrieve sk from $\langle \mathcal{M}_i, \mathcal{H}_j, \mathsf{bsn}, \mathsf{sk} \rangle \in \texttt{DomainKeys}$ for $(\mathcal{M}_i, \mathsf{bsn})$. If no such sk exists or $\mathsf{bsn} = \perp$, set $\mathsf{sk} \leftarrow \mathsf{ukgen}()$. Check $\mathsf{CheckSkHonest}(\mathsf{sk}) = 1$ **(v)** and store $\langle \mathcal{M}_i, \mathcal{H}_j, \mathsf{bsn}, \mathsf{sk} \rangle$ in $\texttt{DomainKeys}$.
   - Compute signature as $\sigma \leftarrow \mathsf{sig}(\mathsf{sk}, \mathsf{m}, \mathsf{bsn})$ and check $\mathsf{ver}(\sigma, \mathsf{m}, \mathsf{bsn}) = 1$ **(vi)**.
   - Check $\mathsf{identify}(\sigma, \mathsf{m}, \mathsf{bsn}, \mathsf{sk}) = 1$ **(vii)** and check that there is no $\mathcal{M}_i' \neq \mathcal{M}_i$ with signing key sk' registered in $\texttt{Members}$ or $\texttt{DomainKeys}$ with $\mathsf{identify}(\sigma, \mathsf{m}, \mathsf{bsn}, \mathsf{sk}') = 1$ **(viii)**.
   - If $\mathcal{M}_i$ is honest, store $(\sigma, \mathsf{m}, \mathsf{bsn}, \mathcal{M}_i)$ in $\texttt{Signed}$.
   - Output $(\mathsf{SIGNATURE}, \mathsf{sid}, \mathsf{ssid}, \sigma)$ to $\mathcal{H}_j$.

Verify

1. On input $(\mathsf{VERIFY}, \mathsf{sid}, \mathsf{m}, \mathsf{bsn}, \sigma, \mathsf{RL})$ from some party $\mathcal{V}$,
   - Retrieve all pairs $\langle \mathcal{M}_i, *, \mathsf{sk} \rangle \in \texttt{Members}$ such that $\mathsf{identify}(\sigma, \mathsf{m}, \mathsf{bsn}, \mathsf{sk}_i) = 1$ and all $\langle \mathcal{M}_i, \mathcal{H}_j, *, *, \mathsf{sk}_i \rangle \in \texttt{DomainKeys}$ where $\mathsf{identify}(\sigma, \mathsf{m}, \mathsf{bsn}, \mathsf{sk}_i) = 1$. Set $f \leftarrow 0$ if at least one of the following conditions hold:
     - More than one such value sk_i was found **(ix)**.
     - $\mathcal{I}$ is honest an no identifying value sk_i was found **(x)**.
     - $\mathcal{I}$ is honest and there is an honest $\mathcal{M}_i$ but no entry $\langle *, \mathsf{m}, \mathsf{bsn}, \mathcal{M}_i \rangle \in \texttt{Signed}$ exists **(xi)**.
     - There is an honest platform $(\mathcal{M}_i, \mathcal{H}_j)$, but no entry $\langle *, \mathsf{m}, \mathsf{bsn}, \mathcal{M}_i \rangle \in \texttt{Signed}$ exists **(xii)**.
     - There is a $\mathsf{sk}' \in \mathsf{RL}$ where $\mathsf{identify}(\sigma, \mathsf{m}, \mathsf{bsn}, \mathsf{sk}') = 1$ and no identifying value sk_i was found for an honest platform $(\mathcal{M}_i, \mathcal{H}_j)$ **(xiii)**.
   - If $f \neq 0$, set $f \leftarrow \mathsf{ver}(\sigma, \mathsf{m}, \mathsf{bsn})$ **(xiv)**.
   - Add $\langle \sigma, \mathsf{m}, \mathsf{bsn}, \mathsf{RL}, f \rangle$ to $\texttt{VerResults}$, output $(\mathsf{VERIFIED}, \mathsf{sid}, f)$ to $\mathcal{V}$.

Link

1. On input $(\mathsf{LINK}, \mathsf{sid}, \sigma, \mathsf{m}, \sigma', \mathsf{m}', \mathsf{bsn})$ from $\mathcal{V}$ with $\mathsf{bsn} \neq \perp$,
   - Output $\perp$ to $\mathcal{V}$ if at least one signature tuple $(\sigma, \mathsf{m}, \mathsf{bsn})$ or $(\sigma', \mathsf{m}', \mathsf{bsn})$ is not valid (verified via the Verify interface with $\mathsf{RL} = \emptyset$) **(xv)**.
   - For each sk_i in $\texttt{Members}$ and $\texttt{DomainKeys}$ compute $b_i \leftarrow \mathsf{identify}(\sigma, \mathsf{m}, \mathsf{bsn}, \mathsf{sk}_i)$ and $b_i' \leftarrow \mathsf{identify}(\sigma', \mathsf{m}', \mathsf{bsn}, \mathsf{sk}_i)$ and do the following:
   - Set $f \leftarrow 0$ if $b_i \neq b_i'$ for some i **(xvi)**.
   - Set $f \leftarrow 1$ if $b_i = b_i' = 1$ for some i **(xvii)**.
   - If f is not yet defined, set $f \leftarrow \mathsf{link}(\sigma, \mathsf{m}, \sigma', \mathsf{m}', \mathsf{bsn})$.
   - Output $(\mathsf{LINK}, \mathsf{sid}, f)$ to $\mathcal{V}$.

Fig. 5. The Sign, Verify, and Link interfaces of $\mathcal{F}_{\mathsf{daa}}$.

B Proof sketch of Theorem 2

Here we give a detailed proof sketch for Theorem 2.

To show that no environment, $\mathcal{E}$, can distinguish the real world, in which it is working with Π_{LDAA} and adversary $\mathcal{A}$, from the ideal world, in which it uses $\mathcal{F}_{\mathsf{daa}}$ with simulator $\mathcal{S}$, we use a sequence of games. We start with the real world protocol execution. In the next game we construct one entity C that runs the real world protocol for all honest parties. Then we split C into two pieces, a functionality $\mathcal{F}$ and a simulator $\mathcal{S}$, where $\mathcal{F}$ receives all inputs from honest parties and sends the outputs to honest parties. We start with a useless functionality, and gradually change $\mathcal{F}$ and update $\mathcal{S}$ accordingly, to end up with the full $\mathcal{F}_{\mathsf{daa}}$ and a satisfying simulator.

The proof closely follows the structure of the UC proof by Camenisch et al. in [CDL16b], with the crucial steps occurring in Game 7, where the signatures of honest platforms are replaced by signatures on "dummy" keys (guaranteeing *anonymity*), and Games 12–15 where we let the functionality enforce the expected *unforgeability* and *non-frameability* properties. For unforgeability we rely on the security of credentials created by the issuer. This is exactly the $\mathsf{Int\text{-}MNTRU\text{-}ISIS}_f$ assumption which guarantees that only the issuer can produce the preimage $\mathbf{s}$ which forms part of the credential.

Game 1. This is the real world protocol.

Game 2. The challenger $\mathcal{C}$ now receives all inputs and simulates the real world protocol for honest parties. Since $\mathcal{C}$ gets all inputs, it can simply run the real world protocol. It also simulates all hybrid functionalities, but does so honestly, so $\mathcal{E}$ does not see any difference. By construction, this is equivalent to the previous game.

Game 3. We now split $\mathcal{C}$ into a "dummy functionality" $\mathcal{F}$ and simulator $\mathcal{S}$. $\mathcal{F}$ receives all inputs, and simply forwards them to $\mathcal{S}$. $\mathcal{S}$ simulates the real world protocol and sends the outputs it generates to $\mathcal{F}$, who then outputs them to $\mathcal{E}$. This game only restructures the previous game.

Game 4. In this game we let our intermediate $\mathcal{F}$ handle the setup related interfaces using the procedure specified in $\mathcal{F}$. Consequently, $\mathcal{F}$ expects to receive the algorithms ($\mathsf{ukgen}, \mathsf{sig}, \mathsf{ver}, \mathsf{link}, \mathsf{identify}$) from the simulator. ukgen is used to generate the secret keys of all honest TPMs by sampling them as described in the join phase of Section 4.2. For ver, and link, $\mathcal{S}$ can simply provide the algorithms from the real-world protocol, where it omits the revocation check from ver. The sig algorithm will be a combination of the join and sign procedure though, as it will be used to create anonymous signatures for honest platforms for which it uses a fresh TPM key whenever the platform signs w.r.t. a new basename. Thus, to internally create signatures via sig, the algorithm must first create a valid membership credential for the freshly chosen tsk and then sign with this new credential. So sig must contain the issuer's private key, which the simulator $\mathcal{S}$ has to be able to get.

When $\mathcal{I}$ is honest, $\mathcal{S}$ is running the issuer, i.e., it knows its secret key and sets the sig algorithm accordingly. When $\mathcal{I}$ is corrupt, $\mathcal{S}$ starts the simulation when the issuer registers his key with $\mathcal{F}_{\mathsf{ca}}$ that is controlled by the simulator. Since the public key comes with a proof of knowledge of the issuer's secret key, $\mathcal{S}$ can extract the secret key from there and define the sig algorithm accordingly. By the zero-knowledge property of the proof system, this game hop is indistinguishable for the adversary.

Finally, for $\mathsf{identify}$ which is used to check whether a signature $(\sigma, m, \mathsf{bsn})$ belongs to a certain tsk, we use roughly the same procedure as for revocation checks. That is, the algorithm parses $\sigma = (\mathsf{nym}, d, \mathsf{bsn})$, $\mathsf{tsk} = (\mathbf{e}_1, *)$, and checks that $\|(\mathsf{nym} - \mathbf{D}\mathbf{e}_1)\|$ is small. If so it outputs 1, and 0 otherwise. Recall that we compute pseudonyms for random d when $\mathsf{bsn} = \perp$, so this check works for all cases.

Game 5. $\mathcal{F}$ now handles the verify and link queries using the provided algorithms ver and link from the previous game, rather than forwarding the queries to $\mathcal{S}$. We do not let $\mathcal{F}$ perform the additional checks (Checks (ix) – (xvi)) done by $\mathcal{F}$, though, but add these only later. For Check (xii), $\mathcal{F}$ rejects a signature when a matching $\mathsf{tsk}' \in \mathsf{RL}$ is found, but does not exclude honest TPMs from this check yet.

Because verify and link do not involve network traffic, the simulator does not have to simulate traffic either, we must only make sure the outputs do not change. $\mathcal{F}$ executes the algorithms that $\mathcal{S}$ supplied, and $\mathcal{S}$ supplied them in such a way that they are equivalent to the real world algorithms, so the outcome will clearly be equivalent.

Game 6. In this step we change $\mathcal{F}$ to also handle the join-related interfaces, meaning it will receive the inputs and generate the outputs. We let $\mathcal{F}$ run the same procedure as $\mathcal{F}$, but again omit the additional checks (Checks (ii)–(iv)).

In the final join interface $\mathsf{JOINCOMPLETE}$, the simulator has to provide the secret key tsk of the TPM. When the TPM is honest, $\mathcal{S}$ knows the key anyway and uses it towards $\mathcal{F}$. If the TPM is corrupt and either the issuer or host is honest, $\mathcal{S}$ extracts the vector $\mathbf{e}_1$ from the proof π_{join} that it receives in the role of the honest $\mathcal{I}$ or $\mathcal{H}_j$ using the knowledge soundness of the proof system and sets $\mathsf{tsk} \leftarrow (\mathbf{e}_1, \perp)$. Note that we do not extract nor set the part e_3 of the TPM's secret key. This has no impact though, as tsk will only be used for internal checks by $\mathsf{identify}$ for which only $\mathbf{e}_1$ is used.

Finally, note that $\mathcal{F}$ sets $\mathsf{tsk} := \perp$ when both the TPM and host are honest. However, this has no impact yet, as the signatures are still created by the simulator and the verify and link interfaces of $\mathcal{F}$ do not run the additional checks that make use of the internally stored records and keys.

Overall, this game hop is indistinguishable by the knowledge soundness of π_{join}.

Game 7. We now transform $\mathcal{F}$ such that it internally handles the signing queries of *honest platforms* instead of merely forwarding them to $\mathcal{S}$. Thus, this game hop proves the **anonymity** of our DAA scheme.

Again, $\mathcal{F}$ uses the sign interfaces from $\mathcal{F}$, with the difference that it does not perform the Check (v) – (vii) which we only add in a later game.

When both the TPM and the host are honest, $\mathcal{F}$ creates the signatures internally in an unlinkable way: It chooses a new tsk per basename and TPM, or per signature when $\mathsf{bsn} = \perp$ and then runs the sig algorithm for that fresh key. As described earlier, sig starts by internally "issuing" a membership credential on tsk using the issuer's secret key that is included in sig. $\mathcal{F}$ keeps the internally chosen keys $\langle \mathcal{M}_i, \mathsf{bsn}, \mathsf{tsk} \rangle$ in a list `DomainKeys` to ensure consistency if a TPM wishes to reuse the basename.

This change is indistinguishable by the zero-knowledge of π_{sign}. The reduction can be done in a straightforward manner, using a hybrid argument to replace the signatures one-by-one.

Game 8. We change $\mathcal{F}$ such that it no longer informs $\mathcal{S}$ which message and basename are being signed. Thus, when the TPM and host are both honest, $\mathcal{S}$ does not learn m or bsn but only its leakage $l(m, \mathsf{bsn})$. Recall, that signatures for honest platforms are generated by the functionality now, so $\mathcal{S}$ merely as to mimic communication between the honest TPM and host.

Game 9. We now add the constraint that when $\mathcal{I}$ is honest, $\mathcal{F}$ only allows platforms that joined to sign, which is checked via the list `Members`. Note that for our simulation we only care about platforms that are at least "partially" honest, i.e., the host and/or TPM are honest, as otherwise there is nothing to simulate. For such platforms, this check will not change the view of $\mathcal{E}$ using the simulator $\mathcal{S}$ from the previous game: in the real world, an honest host and TPM both check that they have a joined before signing. In the ideal world, $\mathcal{S}$ makes join queries towards $\mathcal{F}$ ensuring that the joined platforms (with honest entities) are in `Members`, and thus $\mathcal{F}$ still allows any signing that could take place in the real world.

Game 10. In this game we let $\mathcal{F}$ additionally check the validity of every new tsk that is generated or received in the join and sign interface.

If the TPM is corrupt, $\mathcal{F}$ checks that $\mathsf{CheckSkCorrupt}(\mathsf{tsk}) = 1$ for the tsk that the simulator extracted from π_{join} (Check (iv)). This check prevents the adversary from choosing different keys $\mathsf{tsk} \neq \mathsf{tsk}'$ that both fit to the same signature. By uniqueness of solutions to MLWE only one secret $\mathbf{e}_1$ satisfying the nym-part of the signature (also for the "random" pseudonyms when $\mathsf{bsn} = \perp$). As there is only a single tsk for every valid signature where $\mathsf{identify}(\sigma, m, \mathsf{bsn}, \mathsf{tsk}) = 1$, this check will never fail.

For keys of honest TPMs, $\mathcal{F}$ verifies that $\mathsf{CheckSkHonest}(\mathsf{tsk}) = 1$ whenever it receives or generates a new tsk (Check (iii) and (v)). With these checks we avoid the registration of keys for which matching signatures already exist. Since keys for honest TPMs are chosen uniformly at random from an exponentially large group and every signature has exactly one matching key, the chance that a signature under that key already exists is negligible.

Game 11. We now add the checks to $\mathcal{F}$ that $\mathcal{F}$ runs in the sign interfaces when internally generating signatures for honest platforms. After creating a signature, $\mathcal{F}$ checks whether the signature verifies and matches the right key (Check (vi) and (vii)). As $\mathcal{S}$ supplied proper algorithms and the signature scheme is complete, these checks will obviously always succeed.

$\mathcal{F}$ also checks with the help of its internal key records `Members` and `DomainKeys` that no one else already has a key which would match this newly generated signature (Check viii). As signatures match only a single TPM key and we choose keys of honest platforms at random from a large domain, this can happen only with negligible probability.

In the next four game hops, we let $\mathcal{F}$ perform the four additional checks that are done by $\mathcal{F}$ in the verification interface, i.e., we rely on $\mathcal{F}$ to enforce the desired **unforgeability** and **non-frameability** guarantees. We now show that this check does not change the verification outcome, as any signature that would previously pass will still pass.

Game 12. $\mathcal{F}$ now performs an additional check during verification, it checks whether it finds multiple tsk values matching this signature, and if so, it rejects the signature. It is easy to see that there is only one tsk per signature for which $\mathsf{identify}$ will output 1, as there is only one $\mathbf{e}_1$ that can lead to the pseudonym nym (by the hardness of MLWE). Moreover, if the proof π_{sign} is valid, multiple matching tsk values breaks the soundness of the proof system.

Game 13. When $\mathcal{I}$ is honest, $\mathcal{F}$ now only accepts signatures on tsk values that $\mathcal{I}$ has issued a membership credential on. Under the existential unforgeability of the membership credential, this check changes the verification outcome only with negligible probability. Valid signatures using tsk values not certified by the issuer break the soundness of π_{sign} (and thus break the $\mathsf{Int\text{-}NTRU\text{-}ISIS}_f$ problem).

Game 14. $\mathcal{F}$ now prevents forging signatures using an honest TPM's tsk. If the environment can distinguish this game hop, i.e., it can create a valid signature σ for message m and basename bsn that traces via $\mathsf{identify}$ to an honest TPM, but that TPM has never signed m, bsn. We can use this to break the MLWE problem. Again, the reduction can be done in a straight-forward manner: We use the MLWE challenge as $\mathbf{u}_1$ for a randomly chosen honest TPM during the join protocol (with the possibly corrupt issuer). DAA signatures for this honest TPM are merely simulated. If we see a valid signature that we never created, we extract $\mathbf{e}_1$ from the accompanying π_{sign} and output that as the MLWE solution.

Game 15. Check (xii) is added to $\mathcal{F}$, this ensures that honest TPMs are not being revoked. If an honest TPM is simulated by means of the Ring-LWE problem instance, if a proper key RL is found, it must be the secret key of the target instance. This is again equivalent to solving the MLWE problem.

Game 16. We now let $\mathcal{F}$ perform all the additional checks $\mathcal{F}$ makes for link queries. The output of $\mathcal{F}$ based on these checks is still consistent with the output which the link algorithm would give: If there is a tsk that matches one signatures but not the other, by the knowledge soundness of π_{sign} we have that the pseudonyms are not based on the same tsk, and thus must differ which results in link outputting 0. If there is a tsk that matches both signatures, again by knowledge soundness of π_{sign} we have that the pseudonyms are based on the same tsk and must be equal, resulting in link outputting 1.

Now $\mathcal{F}$ is equal to $\mathcal{F}_{\mathsf{daa}}$, concluding our proof sketch.