Efficient Post-Quantum SNARKs for RSIS and RLWE and their Applications to Privacy*

Cecilia Boschini[1,2], Jan Camenisch[3], Max Ovsiankin[4], and Nicholas Spooner[4]

[1] IBM Research - Zurich
[2] Università della Svizzera Italiana
[3] Dfinity
[4] UC Berkeley

Abstract. In this paper we give efficient statistical zero-knowledge proofs (SNARKs) for Module/Ring LWE and Module/Ring SIS relations, providing the remaining ingredient for building efficient cryptographic protocols from lattice-based hardness assumptions. We achieve our results by exploiting the linear-algebraic nature of the statements supported by the Aurora proof system (Ben-Sasson et al.), which allows us to easily and efficiently encode the linear-algebraic statements that arise in lattice schemes and to side-step the issue of "relaxed extractors", meaning extractors that only recover a witness for a larger relation than the one for which completeness is guaranteed. We apply our approach to the example use case of partially dynamic group signatures and obtain a lattice-based group signature that protects users against corrupted issuers, and that produces signatures smaller than the state of the art, with signature sizes of less than 300 KB for the comparably secure version of the scheme. To obtain our argument size estimates for proof of knowledge of RLWE secret, we implemented the NIZK using libiop.

Keywords: zero-knowledge proofs, group signatures, lattice-based cryptography, post-quantum cryptography

1 Introduction

Zero knowledge (ZK) proofs allow a prover P to convince a verifier V of the truth of a statement without revealing any information except that the statement is true. Since their introduction [25], they have been widely used in cryptography, both as a tool in their own right and as a building block in more complex protocols [19, 10, 18, 17]. A central difficulty in building efficient lattice-based privacy-preserving protocols is constructing post-quantum secure zero knowledge proofs for the types of relations which arise in the constructions. As most schemes are based on some version of the Ring/Module-SIS and Ring/Module-LWE problems [37, 41, 32, 35, 28], constructing efficient ZK proofs for these relations is of great importance. However, until now efficient zero-knowledge proof for lattices (cf. [22, 12]) came with the drawback of having "relaxed extractors", meaning extractors that only recover a witness for a relation which is a strict superset of the one for which completeness is guaranteed.

1.1 Our Results

We present non-interactive zero knowledge (NIZK) proofs for Module/Ring-LWE and Module/Ring-SIS relations, that have size of the order of 70 kB for 128 bits of security. These proofs rely on Aurora, a SNARK designed by Ben-Sasson et al. [7]. From it, our proofs inherit statistical zero-knowledge and soundness, post-quantum security, exact extractability (that is, the extraction guarantee is for the same relation as the protocol completeness), and transparent setup (no need for a trusted authority to generate the system parameters). Such proofs support algebraic circuits, and therefore can be combined with lattice based building

* The original paper (with the same title and by the same authors) was presented at PQCrypto 2020 [14]. This is the full version.

	Partially Dynamic	Anonymous	Traceable	Non-Frameable	Users	δ_{HRF}	Signature(MB)
[22]	✓	✓	✓		2^{80}	1.002	0.581
$\mathcal{GS}$	✓	✓	✓		2^{26}	1.0007	0.3
$\mathcal{GS}_{\text{full}}$	✓	✓	✓	✓	2^{26}	1.0007	1.44

Table 1. Comparison for around 90 bits of security.

blocks. We show that it is possible to combine this protocol with (the ring version of) Boyen's signature [15], to prove knowledge of a signature on a publicly known message, or knowledge of a valid pair message-signature, and an RLWE-based encryption scheme [35], to prove knowledge of a valid decryption of a given ciphertext.

To showcase their efficiency we construct a (partially) dynamic group signature [6], and we compare it with the most efficient NIZK-based group signature to date [22] in Table 1.1. Differently from ours, the scheme by del Pino et al. does not protect honest users from framing attempts by corrupted issuers (the non-frameability property). Therefore, we compare it with two variants of our scheme: $\mathcal{GS}$, that does not guarantee non-frameability, and $\mathcal{GS}_{\text{full}}$ (cf. Section 5), that also has non-frameability. To compare the security levels of the schemes we consider the Hermite Root Factors (denoted by δ_{HRF}); a smaller delta implies higher security guarantees. The choice of the parameters setting is motivated in Section 5.5. In both cases, the NIZK proof is of size less than 250 KB, improving upon the state of the art.

The group signature is proven secure in the ROM under RSIS and RLWE. Security in the QROM follows also from [20]; to achieve 128 bits of QROM security requires a three-fold increase in proof size.

We demonstrate the effectiveness of our NIZKs with an implementation. We are able to produce a Ring-LWE proof in around 40 seconds on a laptop (cf. Section 3.7). In comparison, the scheme of [22] produces proofs in under a second. Nonetheless, we consider our NIZK and group signature a benchmark for evaluating efficiency claims for (existing and future) NIZK proofs for lattice relations. In particular, it shows what can be achieved using 'generic' tools.

1.2 Our Techniques

In their simplest form, lattice problems can be generically described as finding a vector $s \in \mathbb{Z}_q^n$ with small coefficients (i.e., $|s_i| \leq \beta$ for all i) such that $Ms = u \bmod q$ for given matrix $M \in \mathbb{Z}_q^{m \times n}$ and vector $u \in \mathbb{Z}_q^m$, q being a prime.

Until now, in the most efficient protocols (cf. for example [3]) which allow proving knowledge of the vector s, the prover has to mask s with a previously chosen random vector, and prove that it satisfies some linear relation. For the norm bound, the prover would prove that with high probability s lies in an interval centered about the origin (cf. [31, 30]). This method has two main problems: first, the challenge set has low entropy, hence, to get a low soundness error, the proof needs to be repeated multiple times, and second, the length of the proof is linear in n, the dimension of s.

The SNARK Aurora allows to prove knowledge of a witness for a given instance of the *Rank-1 Constraint Satisfaction* (R1CS), i.e., of a vector $z \in \mathbb{F}^{n+1}$ such that, given a vector $v \in \mathbb{F}^k$ and three matrices $A, B, C \in \mathbb{F}^{m \times (n+1)}$, $k < n$, the vector z extends v to satisfy $Az \circ Bz = Cz$, where $\circ$ denotes the entry-wise product. The entries of v are the unknowns of the problem, while the equations they satisfy are called constraints (and are derived from the general equation $Az \circ Bz = Cz$). Hence, we say that the previous R1CS system has n unknowns and m constraints. Aurora provides proofs of length $O_\lambda(\log^2 N)$, where $f = O_\lambda(\log^2 N)$ means $f = O(\lambda^c \log^2 N)$ for some $c > 0$, and N is the total number of nonzero entries of A, B, C.

The conversion of an instance (s, M, u) of a lattice problem to an instance of R1CS is quite natural. We set $\mathbb{F} := \mathbb{Z}_q$ so that to prove that $Ms = u \bmod q$ holds, it is enough to set $A := [0_{m \times 1}\ M]$, $B := [1_{m \times 1}\ 0_{m \times n}]$, $C := [u\ 0_{m \times n}]$, and $z := [1\ s^T]^T$, where $0_{m \times n}$ (resp. $1_{m \times n}$) is a matrix with m rows and n columns with all components equal to 0 (resp. 1), and the parameter k of the R1CS problem is set to be $k = n$. The number of constraints of this system is m, and the number of variables is n.

To prove that the secret vector s has also a small norm, we use binary decomposition. In particular, to prove that a component s_j of s is smaller than $\beta = 2^h$, it is enough to verify that its binary representation

is at most h bits long, i.e.,

$$s_j = c_j \sum_{i=0}^{h-1} 2^i b_{i,j} \ \text{, with } c_j \in \{\pm 1\} \text{ and } b_{i,j} \in \{0,1\} \ \forall i \ ,$$

where c is the bit representing the sign of a. This is equivalent to proving that $b_{0,j}, \ldots, b_{h-1,j}, s_j$ satisfy the following constraints:

$$b_{i,j}(1 - b_{i,j}) = 0 \quad \forall i \quad \wedge \quad \left(\sum_{i=0}^{h-1} b_{i,j} 2^i - s_j\right)\left(\sum_{i=0}^{h-1} b_{i,j} 2^i + s_j\right) = 0 \ .$$

These correspond to the R1CS instance (A_j, B_j, C_j) and witness z_j, with

$$A_j := \begin{bmatrix} 0 & & 0 \\ \vdots & \mathbb{I}_h & \vdots \\ 0 & & 0 \\ 0\ 1\ 2 \ldots 2^{h-1} & & -1 \end{bmatrix} \ , \quad B_j := \begin{bmatrix} 1 & & 0 \\ \vdots & -\mathbb{I}_h & \vdots \\ 1 & & 0 \\ 0\ 1\ 2 \ldots 2^{h-1} & & 1 \end{bmatrix} \ , \quad z_j := \begin{bmatrix} 1 \\ b_{0,j} \\ \vdots \\ b_{h-1,j} \\ s_j \end{bmatrix} \ ,$$

and C_j the all-zero matrix, where $\mathbb{I}_h$ is the identity matrix of dimension h. Thus proving that s has a small norm adds $n(h+1)$ constraints and nh unknowns to the proof (i.e., the coefficients of the bit decomposition of each component of s). Hence, expanding $A,\ B,\ C,\ z$ with all the $A_j,\ B_j,\ C_j, z_j$ (taking care not to repeat entries in z) yields the full instance. This includes $m + n(h+1)$ constraints, and the nonzero entries of the matrices $A,\ B,\ C$ are $N = nm + 2m + (5h+1)n$, and outputs proofs of length $O(\log^2(n(m+5h+1)+2m))$ (where we recall that h is the logarithm of the bound on the norm of the solution to the lattice problem).

The R1CS formalism allows us to prove knowledge of a message-signature pair in Boyen's signature scheme adapted to the polynomial ring setting (cf. [15] and Section 2.3) in a natural way (cf. Section 3.4). As this case requires to work with vectors of polynomials instead of elements of $\mathbb{F}_q$ as in the original definition of R1CS, particular care has to be taken with multiplications of ring elements (which are represented as a vector of their coefficients). This can be implemented in an R1CS instance by first applying a linear transformation to the coefficients to place them in the Fourier (NTT) basis, and then multiplying point-wise.

1.3 Related Work

Both Libert et al. [29] and Baum et al. [3] introduce ZK proof to prove knowledge of solutions of lattice problems that are linear in the length of the secret and in $\log \beta$ respectively (where β is the bound of the norm of the secret vector). Our scheme improves these in that the proof length depends *polylogarithmically* on the length of the secret vector and $\log \beta$. Moreover, we implemented our scheme (cf. Section 4), and give concrete estimates for parameters that guarantee 128 bits of security.

The lattice-based SNARK of [24] relies on the qDH assumption (among others), hence unlike our scheme this is not post-quantum secure, and needs a trusted setup, which prevents to use it to build group signatures with the non-frameability property.

Regarding group signature construction, a new construction was published by Katsumata and Yamada [26], that builds group signatures without using NIZK proofs in the standard model. Their construction is of a different form, and, in particular, sidesteps the problem of building NIZKs for lattices, hence we can only compare the signature lengths. Differently from ours, their signature sizes still depend linearly on the number of users (while ours depend polylogarithmically on the number of users) when security is based on standard LWE/SIS. They are able to remove this dependency assuming subexponential hardness for SIS.

2 Preliminaries

We denote vectors and matrices with upper-case letters. Column vectors are denoted as $V = \left[v_1 \,;\, \ldots \,;\, v_n\right]$ and row vectors as $V = \left[v_1 \,\ldots\, v_n\right]$. Sampling and element x from a distribution $\mathcal{D}$ are denoted as $x \xleftarrow{\$} \mathcal{D}$. If x is sampled uniformly over a set A, we write $x \xleftarrow{\$} A$. With $x \leftarrow a$ we denote that x is assigned the value a. When necessary, we denote the uniform distribution over a set S as $\mathcal{U}(S)$. We denote by log the logarithm with base 2. We use the standard Landau notation (i.e., $O(\cdot)$, $\omega(\cdot), \ldots$) plus the notation $O_\lambda(\cdot)$, where $f = O_\lambda(g)$ means that there exists $c > 0$ such that $f = O(\lambda^c g)$.

2.1 Preliminaries: Ideal Lattices

Let $\mathbb{Z}[X]$ be the ring of polynomials with integer coefficients, $\mathbf{f} \in \mathbb{Z}[X]$ be a monic, irreducible polynomial of degree n, and $\mathcal{R}$ be the quotient ring $\mathcal{R} := \mathbb{Z}[X]/\langle \mathbf{f} \rangle$. Let $\deg(\mathbf{a})$ be the degree of the polynomial $\mathbf{a}$. Ring elements are represented with the standard set of representatives $\{\mathbf{g} \bmod \mathbf{f} : \ \mathbf{g} \in \mathbb{Z}[X]\}$, corresponding to vectors in $\mathbb{Z}^n$ through the standard group homomorphism h that sends $\mathbf{a} = \sum_{i=0}^{n-1} a_i x^i$ to the vector of its coefficients $(a_0, \ldots, a_{n-1})$. Let $\mathcal{R}_q = \mathbb{Z}_q[X]/\langle X^n + 1 \rangle$ for a prime q. Elements in the ring are polynomials of degree at most $n-1$ with coefficients in $[0, q-1]$ and operations between them are done modulo q [1]. For an element $\mathbf{a} = \sum_{i=0}^{n-1} a_i \mathbf{x}^i$, the norms are computed as $\|\mathbf{a}\|_1 = \sum_i |a_i|$, $\|\mathbf{a}\| = \sqrt{\sum_i a_i^2}$ and $\|\mathbf{a}\|_\infty = \max |a_i|$. For a vector $\mathbf{S} = [\mathbf{s}_1, \ldots, \mathbf{s}_m] \in \mathcal{R}^m$, the norm $\|\mathbf{S}\|_p$ is defined as $\max_{i=1}^m \|\mathbf{s}_i\|_p$. Let $\mathcal{S}_1$ be the subset of elements of $\mathcal{R}_q$ with coefficients in $\{0, \pm 1\}$. $\mathsf{BitD}(\mathbf{a})$ is an algorithm that on input elements $\mathbf{a}_i \in \mathcal{R}_q$, outputs vectors $\vec{a}_i$ containing the binary expansion of the coefficients of $\mathbf{a}_i$.

Remark that h is a group homomorphism w.r.t. componentwise addition, but it is not a ring homomorphism because $h(\mathbf{fg}) \neq h(\mathbf{f})h(\mathbf{g})$. Combining the standard set of representatives with the group homomorphism we obtain that ideals in $\mathcal{R}$ and $\mathcal{R}_q$ corresponds to lattices in $\mathbb{Z}^n$ and $\mathbb{Z}_q^n$ respectively. Polynomials can also be converted into vectors through the NTT transformation [21, 35], that sends each polynomial to the vector of its evaluation over the n-th roots of unity. This transformation is a ring homomorphism w.r.t. componentwise addition *and* multiplication, and is very useful when building ZK proofs (cf. Section 3).

A sample $\mathbf{z}$ from a discrete Gaussian $\mathcal{D}_{\mathcal{R}_q, \mathbf{u}, \sigma}$ centered in $\mathbf{u}$ and with std. deviation σ, is generated as a sample from a discrete Gaussian over $\mathbb{Z}^n$ and then map it into $\mathcal{R}_q$ using the obvious embedding of coordinates into coefficients of the polynomials. Similarly, we omit the $\mathbf{0}$ and write $\left[\mathbf{y}_1 \ldots \mathbf{y}_k\right] \xleftarrow{\$} \mathcal{D}^k_{\mathcal{R}_q, \sigma}$ to mean that a vector $\mathbf{y}$ is generated according to $\mathcal{D}_{\mathbb{Z}^{kn}, \mathbf{0}, \sigma}$ and then gets interpreted as k polynomials $\mathbf{y}_i$. With an abuse of notation, we denote by $\mathcal{D}^\perp_{\mathbf{A}, \mathbf{u}, s}$ the distribution of the vectors $\mathbf{V} \in \mathcal{R}^m$ such that $\mathbf{V} \sim \mathcal{D}^m_{\mathcal{R}, \mathbf{0}, s}$ conditioned on $\mathbf{AV} = \mathbf{u} \bmod q$.

Lemma 2.1 (cf. [2, Lemma 1.5], [31, Lemma 4.4]). *Let $m > 0$. The following bounds hold:*

1. $\mathsf{Pr}_{\mathbf{S} \xleftarrow{\$} \mathcal{D}^m_\sigma}(\|\mathbf{S}\| > 1.05\sigma\sqrt{m}) < (0.998)^m$
2. $\mathsf{Pr}_{\mathbf{S} \xleftarrow{\$} \mathcal{D}^m_\sigma}(\|\mathbf{S}\|_\infty > 8\sigma) < m2^{-47}$

We recall two well-studied lattice problems over rings: RSIS and RLWE. We present RLWE in the "normal form", i.e., where the secret and the error are chosen from the same distribution. This version is as hard as the standard one (cf. Lemma 2.24 in the full version of [36]).

Definition 2.2. ($\mathrm{RSIS}_{m,q,\beta}$ problem [35]) *The $RSIS_{m,q,\beta}$ problem asks given a vector $\mathbf{A} \xleftarrow{\$} \mathcal{R}_q^{1 \times m}$ to find a vector $\mathbf{S} \in \mathcal{R}_q^m$ such that $\mathbf{AS} = \mathbf{0}$ mod q and $\|\mathbf{S}\| \leq \beta$.*

The inhomogeneous *version of RSIS asks to find $\mathbf{S} \in \mathcal{R}_q^m$ such that $\mathbf{AS} = \mathbf{u}$, and $\|\mathbf{S}\| \leq \beta$ for given uniformly random $\mathbf{A}$ and $\mathbf{u}$.*

[1] This is because $\mathbf{f}$ is monic: by the polynomial division algorithm, if the leading coefficient of $\mathbf{f}$ is invertible in the ring of coefficients, then there exists and are uniquely determined $\mathbf{q}$ and $\mathbf{r}$ such that for all $\mathbf{g} \in \mathcal{R}$, $\mathbf{g} = \mathbf{fq} + \mathbf{r}$, $\deg(\mathbf{r}) < \deg(\mathbf{f})$, cf. Thm 1.1 on page 173 in [27].

Definition 2.3. ($\mathrm{RLWE}_{k,\chi}$ problem, normal form, cf. [36]) *The* $\mathrm{RLWE}_{\chi,\mathbf{s}}$ distribution *(resp., the* $RLWE_\chi$ *distribution in the normal form) outputs pairs* $(\mathbf{a},\mathbf{b}) \in \mathcal{R}_q \times \mathcal{R}_q$ *such that* $\mathbf{b} = \mathbf{as}+\mathbf{e}$ *for a uniformly random* $\mathbf{a}$ *from* $\mathcal{R}_q$*,* $\mathbf{s} \in \mathcal{R}_q$ *and* $\mathbf{e}$ *sampled from distribution* χ *(resp.,* $\mathbf{a} \xleftarrow{\$} \mathcal{R}_q$*,* $\mathbf{s},\mathbf{e} \xleftarrow{\$} \chi$*).*

The $\mathrm{RLWE}_{k,\chi}$ decisional problem *on ring* $\mathcal{R}_q$ *with distribution* χ *is to distinguish whether* k *pairs* $(\mathbf{a}_1,\mathbf{b}_1),\ldots,(\mathbf{a}_k,\mathbf{b}_k)$ *were sampled from the* $RLWE_\chi$ *distribution or from the uniform distribution over* $\mathcal{R}_q^2$*.*

The $\mathrm{RLWE}_{k,\chi}$ search problem *on ring* $\mathcal{R}_q$ *with distribution* χ *is given* k *pairs* $(\mathbf{a}_1,\mathbf{b}_1),\ldots,(\mathbf{a}_k,\mathbf{b}_k)$ *sampled from the* $RLWE_\chi$ *distribution, find* $\mathbf{s}$*.*

Module-RSIS and Module-RLWE [28] are a more general formulation of RSIS and RLWE. *Module-RSIS* asks to find a short vector $\mathbf{S} \in \mathcal{R}_q^{m_2}$ such that $\mathbf{AS} = \mathbf{0}$ given a *matrix* $\mathbf{A} \xleftarrow{\$} \mathcal{R}_q^{m_1 \times m_2}$ (the inhomogeneous version is defined analogously). The *Module-RLWE* distribution outputs pairs $(\mathbf{A}, \langle \mathbf{A},\mathbf{S}\rangle + \mathbf{e}) \in \mathcal{R}_q^k \times \mathcal{R}_q$, where the secret $\mathbf{S}$ and the error $\mathbf{e}$ are drawn from $\mathcal{R}_q^k$ and $\mathcal{R}_q$ respectively.

2.2 RLWE Encryption Scheme

Let n be a power of 2, p, and q be two primes such that $q \gg p$, and χ be an error distribution. The RLWE encryption scheme $(\mathsf{EParGen}, \mathsf{EKeyGen}, \mathsf{Enc}, \mathsf{Dec})$ [36] to encrypt a binary message $\mu \in \mathcal{S}_1$ works as follows. On input the security parameter λ, the parameters generation $\mathsf{EParGen}$ outputs (n,p,q). The key generator $\mathsf{EKeyGen}$ samples $\mathbf{a} \xleftarrow{\$} \mathcal{R}_q$, $\mathbf{s} \xleftarrow{\$} \mathcal{R}_q$ and $\mathbf{d} \leftarrow \chi$, and sets $\mathbf{b} = \mathbf{as}+\mathbf{d} \bmod q$. The encryption key is $epk = (\mathbf{a},\mathbf{b})$, the decryption key is $esk = \mathbf{s}$. On input a message μ, the encryption algorithm Enc generates the ciphertext $(\mathbf{v},\mathbf{w})$ as

$$\begin{aligned} \mathbf{v} &= p(\mathbf{ar}+\mathbf{e}) \bmod q \\ \mathbf{w} &= p(\mathbf{br}+\mathbf{f}) + \mu \bmod q\ , \end{aligned}$$

where $\mathbf{e},\mathbf{f} \xleftarrow{\$} \chi$ and $\mathbf{r} \xleftarrow{\$} \mathcal{R}_q$. Decryption amounts to computing $(\mathbf{w} - \mathbf{sv} \bmod q) \bmod p$.

Theorem 2.4 (Lemma 8.3 and 8.4 in [36]). *The above scheme is IND-CPA secure under* $RLWE_{2,\chi}$*.*

Moreover, if χ *outputs elements with norm bounded by* N *with probability* $1-\nu(n)$*,* $p \cdot \chi$ *is* δ*-subgaussian with parameter* s *for some* $\delta = O(1)$*, and* $q \geq s\sqrt{2(N)^2 + n} \cdot \omega(\sqrt{\log n})$*, then the decryption is correct with probability* $1-\nu(n)$*.*

Observe that we adapted the parameters bounds to work with the particular polynomial ring we have chosen.

This encryption scheme can be made IND-CCA2 secure combining it with a non-malleable NIZK proof system following Naor-Yung construction [40]. In our instantiation we choose the error distribution χ to be a Gaussian distribution with standard deviation $s_{RLWE} = \omega(\sqrt{\log q})$ (cf. Theorem 1 in [16]).

We remark that this encryption scheme encrypts plaintexts that are polynomials of degree n with binary coefficients. In case it would be necessary to encrypt a bit string $\vec{b} = (b_1,\ldots,b_k)$, we assume the encryption algorithm first converts it to an element of $\mathcal{S}_1$ (or more than one, if $k > n$) by setting $b_i = 0$ for $k < i \leq n$ and constructing the polynomial $\mathbf{b} = \sum_{i=1}^{n} b_i x^{i-1}$ (the case $k > n$ is analogous).

2.3 Boyen's signature on ideal lattices

A digital signature scheme is composed by 4 PPT algorithms $(\mathsf{SParGen}, \mathsf{SKeyGen}, \mathsf{Sign}, \mathsf{SVerify})$. Existential unforgeability against adaptive chosen-message attacks (*eu-acma*) requires that the adversary should not be able to forge a signature on some message μ^* of her choice, even if she has access to a signing oracle. In this section we describe the variant of Boyen's signature [15] by Micciancio and Peikert [38], adapted to have security based on hardness assumptions on ideal lattices. Such variant has been claimed to be secure since long time, but, to the best of our knowledge, this is the first time in which a security proof is given explicitly (cf. Appendix 2.3). In particular, we prove the signature secure when defined over the $(2n)$-th cyclotomic ring. We choose to work over ideal lattices because they allow for a more efficient representation of their elements, thus improving the storage requirements of the scheme (cf. [37]).

Theorem 2.5 (Trapdoor generation, from [38]). *Let $\mathcal{R}_q$ be a power of 2 cyclotomic ring and set parameters $m = 2$, $k = \lceil \log q \rceil$, $\bar{m} = m + k$. There exists an algorithm* GenTrap *that outputs a vector $\bar{\mathbf{A}} \in \mathcal{R}_q^{1\times \bar{m}}$ and a trapdoor $\mathbf{R} \in \mathcal{R}_q^{m\times k}$ with tag $\mathbf{h} \in \mathcal{R}_q$ such that:*

- $\bar{\mathbf{A}} = [\mathbf{A}|\mathbf{AR} + \mathbf{hG}]$*, where* $\mathbf{G}$ *is the* gadget matrix*,* $\mathbf{G} = [1\, 2\, 4\, \ldots\, 2^{k-1}]$*, and* $\mathbf{A} = [\mathbf{a}|\mathbf{1}] \in \mathcal{R}_q^{1\times 2}$*,* $\mathbf{a} \xleftarrow{\$} \mathcal{R}_q$*.*
- $\mathbf{R}$ *is distributed as a Gaussian* $\mathcal{D}_{\mathcal{R},s}^{2\times k}$ *for some* $s = \alpha q$*, where* $\alpha > 0$ *is a RLWE error term,* $\alpha q > \omega(\sqrt{\log n})$ *(cf [36, Theorem 2.22]).*
- $\mathbf{h}$ *is an invertible element in* $\mathcal{R}_q$*.*
- $\bar{\mathbf{A}}$ *is computationally pseudrandom (ignoring the component set to* $\mathbf{1}$*) under (decisional) RLWE$_D$ where* $D = \mathcal{D}_{\mathcal{R},s}$*.*

Genise and Micciancio [23] give an optimal sampling algorithm for the previous trapdoor construction.

Theorem 2.6 (Gaussian sampler, adapted from [38] and [23]). *Let $\mathcal{R}_q$, m, k, $\bar{m}$ be as in Theorem 2.5,* $\mathbf{G}$ *be the* gadget matrix $\mathbf{G} = [1\, 2\, 4\, \ldots\, 2^{k-1}]$*,* $\mathbf{A} \in \mathcal{R}_q^{1\times m}$ *and* $\mathbf{R} \in \mathcal{R}_q^{2\times k}$ *be the output of GenTrap, and* $\mathbf{B}$ *a vector in* $\mathcal{R}_q^{1\times d}$ *for some* $d \geq 0$*. Then, there is an algorithm that can sample from the distribution* $\mathcal{D}^{\perp}_{[\mathbf{A}\ |\ \mathbf{AR}+\mathbf{G}\ |\ \mathbf{B}],\mathbf{u},s}$ *for any* $s = O(\sqrt{n \log q}) \cdot \omega(\sqrt{\log n})$ *for any* $\mathbf{u} \in \mathcal{R}_q$ *in time* $\tilde{O}(n \log q)$ *for the offline phase and* $\tilde{O}(n^2)$ *for the online phase.*

The original signature was proved existentially unforgeable against adaptive chosen-message attacks *eu-acma* under SIS. Micciancio and Peikert proved their variant to be strongly unforgeable against static chosen-message attack (*su-scma*) under SIS with a tighter reduction, and then made it strongly unforgeable against adaptive chosen-message attacks *su-acma* using chameleon hash functions [43]. For our purposes adaptive existential unforgeability is enough, so our aim is to prove the scheme *eu-acma* under RSIS combining the techniques used in the proofs of these two papers.

Parameters: $spar \leftarrow \mathsf{SParGen}(1^\lambda)$
Let $\mathbf{f}$ be the $(2n)$-th cyclotomic polynomial, $\mathbf{f} = \mathbf{x}^n + 1$. and q be a prime. Construct the polynomial rings $\mathcal{R} = \mathbb{Z}[X]/\langle \mathbf{f} \rangle$ and $\mathcal{R}_q = \mathbb{Z}_q[X]/\langle \mathbf{f} \rangle$. Let $k = \lceil \log_2 q \rceil$, $m = 2$, and $\bar{m} = m + k = 2 + \lceil \log q \rceil$ be the length of the public matrices, and ℓ be the length of the message. Let $s_{ssk} = \sqrt{\log(n^2)} + 1$ and $s_\sigma = \sqrt{n \log n} \cdot \sqrt{\log n^2}$ be the standard deviations of the distributions of the signing key and of the signature respectively (their values are determined following Theorem 2.5 and 2.6 respectively).

Key Generation: $(svk, ssk) \leftarrow \mathsf{SKeyGen}(spar)$
Run the algorithm GenTrap from Theorem 2.5 to get a vector $[\mathbf{A} \mid \mathbf{B}] = [\mathbf{A} \mid \mathbf{AR} + \mathbf{G}]$ and a trapdoor $\mathbf{R}$. The public key is composed by $\ell + 1$ random matrices $\mathbf{A}_0, \ldots, \mathbf{A}_\ell \xleftarrow{\$} \mathcal{R}_q^{1\times k}$, a random vector $\mathbf{u} \xleftarrow{\$} \mathcal{R}_q$ and the vector $[\mathbf{A} \mid \mathbf{B}] \in \mathcal{R}_q^{1\times \bar{m}}$. i.e., $svk = (\mathbf{A}, \mathbf{B}, \mathbf{A}_0, \ldots, \mathbf{A}_\ell, \mathbf{u})$, and the (secret) signing key is $ssk = \mathbf{R}$. Remark that the probability distribution of $\mathbf{R}$ is $\mathcal{D}_{\mathcal{R},s_{ssk}}^{2\times k}$.

Signing: $\sigma \leftarrow \mathsf{Sign}(\mu, ssk)$
To sign a message $\mu = (\mu_1, \ldots, \mu_\ell) \in \{0,1\}^\ell$, the signer constructs a message-dependent public vector $\mathbf{A}_\mu = [\mathbf{A} \mid \mathbf{B} \mid \mathbf{A}_0 + \sum_{i=1}^{\ell} (-1)^{\mu_i} \mathbf{A}_i]$ and then it samples a short vector $\mathbf{S} \in \mathcal{R}_q^{\bar{m}+k}$ running the algorithm SampleD from Theorem 2.6 on input $(\mathbf{A}_\mu, \mathbf{u}, \mathbf{R})$. The algorithm outputs the signature $\sigma = \mathbf{S}$. Remark that the probability distribution of the signature $\mathbf{S}$ is $\mathcal{D}^{\perp}_{\mathbf{A}_\mu,\mathbf{u},s_\sigma}$.

Verification: $\{0,1\} \leftarrow \mathsf{SVerify}(\sigma, \mu, svk)$
The verifier checks that the vector $\mathbf{S}$ has small norm, i.e., $\|\mathbf{S}\|_\infty \leq 8 s_\sigma$. Then, he constructs $\mathbf{A}_\mu = [\mathbf{A} \mid \mathbf{B} \mid \mathbf{A}_0 + \sum_{i=1}^{\ell} (-1)^{\mu_i} \mathbf{A}_i]$ and checks that $\mathbf{S}$ satisfies the verification equation, i.e., $\mathbf{A}_\mu \mathbf{S} = \mathbf{u} \bmod q$.

Correctness follows from Theorem 2.5 and 2.6 and from Lemma 2.1. We prove the *eu-acma* security of the scheme under RSIS by proving that if there exists a PPT adversary A that can break the signature scheme we can construct an algorithm B that can solve RSIS exploiting A. The proof is obtained combining the message guessing technique in the proof of Theorem 25 in [15] with the proof of Theorem 6.1 in [38].

Theorem 2.7. *(eu-acma security) If there exists a PPT adversary* A *that can break the eu-acma security of the signature scheme* $(\mathsf{SParGen}, \mathsf{SKeyGen}, \mathsf{Sign}, \mathsf{SVerify})$ *in time* t_{A} *with probability* ϵ_{A} *asking* q_{A} *queries to the signing oracle, then there exists a PPT algorithm* B *that can solve* $RSIS_{\bar{m}+1,q,\beta}$ *for a large enough* $\beta = 8s_\sigma + (\ell+1)kn8s_\sigma$ *exploiting* A *in time* $t_{\mathsf{B}} \sim t_{\mathsf{A}}$ *with probability* $\epsilon_{\mathsf{B}} = \epsilon_{\mathsf{A}} \cdot (1 - \epsilon_{RLWE}) \cdot \frac{1}{q}\left(1 - \frac{q_{\mathsf{A}}}{q}\right)$ *or a PPT algorithm that solves* $RLWE_{(\ell+1)k,\mathcal{U}(\mathcal{S}_1)}$ *with probability* ϵ_{A} *in time* t_{A}.

Proof. The algorithm B has access to a RSIS oracle that outputs a RSIS instance $\mathbf{A}_{RSIS}$ when prompted. To exploit A to solve RSIS, B has to plug in the RSIS instance in the verification key of the signature, and to exploit the forgery that A produces to build a solution for RSIS. The verification key generated by B should be indistinguishable from a honestly generated key. Moreover, B should also implement a signing oracle that, on input a message from A, outputs a signature on that message.

Upon receiving the instance $\mathbf{A}_{RSIS}$ from the RSIS oracle, B generates the public parameters for the signature scheme as it follows. First, it parses $\mathbf{A}_{RSIS}$ as $\mathbf{A}_{RSIS} = [\mathbf{A} \mid \mathbf{B} \mid \mathbf{u}] = [\mathbf{a} \mid \mathbf{1} \mid \mathbf{B} \mid \mathbf{u}]$ [2]. Rearranging its components, this corresponds to the normal form of RSIS (cf. beginning of Section 4 in [36]). In the eu-acma scenario, the adversary chooses the message μ^* she will forge a signature for after receiving the public key of the scheme and having possibly queried the signing oracle. Hence, the game is as follows. The simulator B generates the verification key $(\mathbf{A}, \mathbf{B}, \mathbf{A}_0, \ldots, \mathbf{A}_\ell, \mathbf{u})$ from the RSIS instance $\mathbf{A}_{RSIS}$ as follows. First, it samples random $\mathbf{R}_i \xleftarrow{\$} \mathcal{S}_1^{2\times k}$ for $i = 0, \ldots, \ell$, the random integers $h_i \xleftarrow{\$} \mathcal{U}(\mathbb{Z}_q)$ for $i = 1, \ldots, \ell$, and sets $h_0 := 1$. Then, it sets:

$$\begin{aligned} &[\mathbf{A} \mid \mathbf{B} \mid \mathbf{u}] := \mathbf{A}_{RSIS} \ , \\ &\mathbf{A}_i := \mathbf{A}\mathbf{R}_i + h_i\mathbf{G} \text{ for } i = 1, \ldots, \ell \ . \end{aligned} \tag{1}$$

Then, B sends $svk = (\mathbf{A}, \mathbf{B}, \mathbf{A}_0, \ldots, \mathbf{A}_\ell, \mathbf{u})$ to A. The key svk generated by the simulator is indistinguishable from a honestly generated one under $\mathrm{RLWE}_{(\ell+1)k,\mathcal{U}(\mathcal{S}_1)}$ (cf. Lemma 2.8).

The adversary is allowed to query signatures on at most q_{A} messages μ of her choice. Upon receiving the message μ, B constructs

$$\begin{aligned} h_\mu &:= h_0 + \sum_{i=1}^{\ell} (-1)^{\mu_i} h_i \\ \mathbf{R}_\mu &:= \mathbf{R}_0 + \sum_{i=1}^{\ell} (-1)^{\mu_i} \mathbf{R}_i \\ \mathbf{A}_\mu &= [\mathbf{A} \mid \mathbf{B} \mid \mathbf{A}_0\mathbf{R}_\mu + h_\mu\mathbf{G}] \\ &= \left[\mathbf{A} \mid \mathbf{B} \mid (\mathbf{A}\mathbf{R}_0 + \sum_{i=1}^{\ell} (-1)^{\mu_i}\mathbf{R}_i) + (h_0 + \sum_{i=1}^{\ell} (-1)^{\mu_i} h_i)\mathbf{G}\right] \\ &= [\mathbf{A} \mid \mathbf{B} \mid \mathbf{A}_0 + \sum_{i=1}^{\ell} (-1)^{\mu_i}\mathbf{A}_i] \ . \end{aligned}$$

If $h_\mu = 0$, B aborts. The product $h_\mu\mathbf{G}$ can be written as $\mathbf{H}_\mu\mathbf{G}$, where $\mathbf{H}_\mu = h_\mu\mathbb{I}_k$ is an invertible matrix, as $h_\mu \in \mathbb{Z}_q$ and q is a prime. Hence $\mathbf{G}$ is a $\mathbf{H}_\mu$-trapdoor for $\mathbf{A}$, and B can use it to sample a short element $\mathbf{S}$ in the lattice $\Lambda^{\perp}_{\mathbf{u}}(\mathbf{A}_\mu)$ thanks to Theorem 2.6. In fact, B samples an element of $\Lambda^{\perp}_{\mathbf{u}}(\mathbf{A}_\mu)$ distributed as a

[2] This can be done with high probability, if the ring $\mathcal{R}_q$ contains enough invertible elements (cf. for example [34, Lemma 2.2]). Indeed, assume that $\mathcal{R}_q$ contains N invertible elements. Then with probability $(m+2)N/q^n$ at least one of the components of $\mathbf{A}_{RSIS} = [\mathbf{a}_1 \ \ldots \ \mathbf{a}_{m+2}]$ is invertible. Assume w.l.o.g. such component to be $b := a_{m+2}$. Then $\mathbf{b}^{-1}\mathbf{a}_i$ are iid (uniform) random variables over $\mathcal{R}_q$, and solving the RSIS instance $[\mathbf{b}^{-1}\mathbf{a}_1 \ \ldots \ \mathbf{b}^{-1}\mathbf{a}_{m+1} \ \mathbf{1}]$ implies finding a solution for the $\mathbf{A}_{RSIS}$ instance of RSIS. In case no component of $\mathbf{A}_{RSIS}$ is invertible, B aborts. This happens with probability $1 - (m+2)N/q^n$, that is negligible if the number N of invertible elements in $\mathcal{R}_q$ is large enough.

Gaussian with standard deviation α, hence the norm of the coefficients of the elements of $\mathbf{S}$ is bounded by $B = 8\alpha$. Hence, the vector $\mathbf{S}$ is a valid signature on μ, and B can send it back to A.

Upon receiving a forgery (μ^*, σ^*), B aborts if $0 \leftarrow \mathsf{SVerify}(\sigma^*, \mu^*, svk)$ or if $h_{\mu^*} \not\equiv 0 \bmod q$. Otherwise, B can extract a solution to RSIS from σ, as it can be seen from the verification equation:

$$\begin{aligned}
\mathbf{A}_{\mu^*}\mathbf{S}^* &= \mathbf{u} \bmod q \\
&\Rightarrow [\mathbf{A} \mid \mathbf{B} \mid \mathbf{A}_0 + \sum_{i=1}^{\ell}(-1)^{\mu_i^*}\mathbf{A}_i] \begin{bmatrix}\mathbf{S}_1^* \\ \mathbf{S}_2^* \\ \mathbf{S}_3^*\end{bmatrix} = \mathbf{u} \bmod q \\
&\Rightarrow [\mathbf{A} \mid \mathbf{B} \mid \mathbf{A}_0\mathbf{R}_{\mu^*}] \begin{bmatrix}\mathbf{S}_1^* \\ \mathbf{S}_2^* \\ \mathbf{S}_3^*\end{bmatrix} = \mathbf{u} \bmod q \\
&\Rightarrow [\mathbf{A} \mid \mathbf{B} \mid \mathbf{u}] \underbrace{\begin{bmatrix}\mathbf{S}_1^* + \mathbf{R}_{\mu^*}\mathbf{S}_3^* \\ \mathbf{S}_2^* \\ -1\end{bmatrix}}_{\mathbf{S}_{RSIS}} = \mathbf{0} \bmod q
\end{aligned}$$

where $\mathbf{R}_{\mu^*} := \mathbf{R}_0 + \sum_{i=1}^{\ell}(-1)^{\mu_i^*}\mathbf{R}_i$. The norm of the vector $\mathbf{S}_{RSIS}$ is dominated by

$$\|\mathbf{S}_1^* + \mathbf{R}_{\mu^*}\mathbf{S}_3^*\|_\infty \leq 8s_\sigma + (\ell+1)\max_i \|\mathbf{R}_i\mathbf{S}_3^*\|_\infty \leq 8s_\sigma + (\ell+1)kn8s_\sigma \ ,$$

as the norm $\max_i \|\mathbf{R}_i\mathbf{S}_3^*\|_\infty \leq k\max_{i,j,h} \|\mathbf{r}_{h,j}^i\mathbf{s}_j^*\|_\infty \leq kn8s_\sigma$, where $\mathbf{r}_{h,j}^i$ and $\mathbf{s}_j^*$ are the components of $\mathbf{R}_i$ and $\mathbf{S}_3^*$ respectively, and the last inequality follows by standard bounds on the infinity norm of the product of polynomials (cf. for example [13, Lemma 1]). The success probability of B is $\epsilon_\mathsf{A} \times (1 - \epsilon_{RLWE}) \times \mathsf{Pr}[\mathsf{B}$ does not abort], where ϵ_{RLWE} is the probability that the adversary can distinguish the verification key generated by B from a honestly generated one.The abort probability of algorithm B can be bounded from below as

$$\mathsf{Pr}[\mathsf{B} \text{ does not abort}] \geq \frac{1}{q}\left(1 - \frac{q_\mathsf{A}}{q}\right)$$

following the same augment in the proof of [15, Lemma 27]. Hence the success probability of B is

$$\epsilon_\mathsf{B} = \epsilon_\mathsf{A} \times (1 - \epsilon_{RLWE}) \times \frac{1}{q}\left(1 - \frac{q_\mathsf{A}}{q}\right)$$

and it is negligible assuming that $q \gg 2q_\mathsf{A}$ (where recall that q_A is the number of queries that A is allow to ask the signing oracle). □

Lemma 2.8. *Assume there exist a PPT algorithm* A *playing the eu-acma experiment that can distinguish the verification key as generated in Equation* (1) *from a honestly generated one with probability* ϵ_A *in time* t_A. *Then there exists an algorithm* B *that can solve* $RLWE_{(\ell+1)k,\mathcal{U}(\mathcal{S}_1)}$ *with probability* $\epsilon_B =$ *in time* $t_B = poly(t_\mathsf{A})$ *exploiting* A.

Proof. The proof is essentially equal to the proof of Lemma 10 in [11, Appendix E], hence we omit it.

Remark 2.9. Observe that to have a non-negligible success probability it should hold that $q \gg 2q_\mathsf{A}$, where q_A is the number of signing queries the adversary is allowed to do. When using this signature as a building block for a group signature, this is actually not limiting. Recall that in the group signature this signature scheme is used to authenticate an encryption of a user's identity. Hence, a limit in the number of users, e.g., assuming 2^{32} users (i.e., more than half of the Earth population), implies that the adversary can make at most 2^{32} queries to the signing oracle. Therefore, choosing $q \gg 2^{33}$ would be enough to ensure a non-negligible success probability. The value of q could be improved using the technique shown in Section 3.5 of [15], which allows to relax the requirement to $q^t \gg 2q_\mathsf{A}$, for some divisor t of n. We decided against the use of this technique

as the improvement would not be significant (the parameter q has to be quite large anyway to ensure the hardness of the RSIS instance underlying the security proof). Finally, remark that for $q \gg 2^\ell$, the security proof can also be done using complexity leveraging, along the lines of the proofs of Theorems 7,8,9 in [11], and results in better parameters.

2.4 The Aurora Protocol

Aurora is a Interactive Oracle Proof for R1CS relations by Ben-Sasson et al. [7].

Definition 2.10 (R1CS relation). *The relation $\mathcal{R}_{R1CS}$ consists of the set of all pairs $((\mathbb{F}, k, m, n, A, B, C, v), w)$ where $\mathbb{F}$ is a finite field, k is the number of inputs, n is the number of variables, m is the number of constraints, A, B, C are matrices in $\mathbb{F}^{m\times(n+1)}$, $v \in \mathbb{F}^k$, and $w \in \mathbb{F}^{n-k}$ such that $Az \circ Bz = Cz$ where $z = (1, v, w) \in \mathbb{F}^{n+1}$ and $\circ$ denotes entry-wise (Hadamard) product.*

The following theorem summarizes the properties of Aurora when compiled to a SNARK via the transform by Ben-Sasson et al. (cf. Theorem 7.1 in [8]). In the statement below, $N := \max(m, n)$; generally n and m will be of roughly the same magnitude.

Theorem 2.11 (informal, cf. Theorem 1.2 in [7]). *There exists a non-interactive zero-knowledge argument for R1CS that is unconditionally secure in the random oracle model with proof length $O(\lambda^2 \log^2 N)$ and one-time simulation soundness error $2^{-\lambda}$ against adversaries making at most 2^λ queries to the oracle. The prover runs in time $O_\lambda(N \log N)$ and the verifier in time $O_\lambda(N)$.*

Remark 2.12 (Simulation soundness). To use the above construction in the Naor–Yung paradigm, as we later do, requires one-time simulation soundness (OTSS). This is shown as follows; we assume some familiarity with [9]. Let π be a proof output by the simulator for a statement x supplied by the adversary. First recall that to achieve adaptive soundness and zero knowledge, the oracle queries of the verifier and honest prover are prefixed with the statement x and a fresh random string $r \in \{0,1\}^\lambda$. Since with high probability no efficient adversary can find $x' \neq x, q, q'$ such that $\rho(x\|r\|q) = \rho(x'\|r\|q')$, if the adversary in the OTSS game chooses an instance different from that of the simulated proof, the success probability of the extractor is affected only by a negligible amount.

Now suppose that an adversary generates a different proof $\pi' \neq \pi$ of the same statement x. In the Aurora IOP, the query locations for the first oracle are a uniformly random subset of $[\ell]$ (where ℓ is the oracle length, $\ell = \Omega(N)$) of size $\Omega(\lambda)$. This is determined by the verifier's final randomness, which in the compiled NIZK depends on all of the Merkle tree roots; these are all included in π. Moreover, these collectively depend on every symbol of π; hence no efficient adversary can find a valid $\pi' \neq \pi$ whose query set is the same as that of π. In particular, the Merkle tree root corresponding to the *first* round has some query in π' which is not in π; since it is infeasible to find an accepting authentication path for this query relative to the root provided by the simulator, the value of this root must differ between π and π'. It follows that, with high probability, the extractor only 'programs' queries which were not already programmed by the simulator, and so one-time simulation soundness holds.

3 NIZKs for Lattices from R1CS

We build the NIZKs from simple, reusable building blocks. When composing these building blocks, it is often necessary to make explicit inputs private. Generally this involves no additional complication; if changes are needed to ensure soundness, we will point them out. When we construct R1CS instances (cf. Definition 2.10), we typically write down a list of variables and constraints, rather than explicitly constructing the matrices.

3.1 Basic operations

We describe how to express some basic lattice operations in $\mathcal{R}_q$ as arithmetic operations over $\mathbb{F}_q \cong \mathbb{Z}_q$ for prime q.

Representation of ring elements. We represent ring elements as vectors in $\mathbb{F}_q^n$ w.r.t. some basis of $\mathcal{R}_q$. Note that regardless of the choice of basis, addition in $\mathcal{R}_q$ corresponds exactly to component-wise addition of vectors. An $\mathcal{R}_q$-element is denoted by a lowercase bold letter (e.g. $\mathbf{a}$) and the corresponding vector in $\mathbb{F}_q^n$ by an arrow (e.g. $\vec{a}$). A vector in $\mathcal{R}_q^m$ is denoted by an uppercase bold letter (e.g. $\mathbf{A}$) and the corresponding matrix in $\mathbb{F}_q^{m \times n}$, whose rows are the coefficients of the elements of the vector, is denoted by an uppercase letter (e.g. A).

Bases. We will use two bases: the coefficient basis and the evaluation or number-theoretic transform (NTT) basis. The NTT basis, which is the discrete Fourier basis over $\mathbb{F}_q$, allows polynomial multiplication to be expressed as pointwise multiplication of vectors. Transforming from the coefficient basis to the NTT basis is a linear transformation $T \in \mathbb{F}_q^{n \times n}$. The choice of basis depends on the type of constraint we wish to check; generally we will represent inputs in the coefficient basis.

An issue with the NTT basis is that to multiply ring elements $\mathbf{a}, \mathbf{b} \in \mathcal{R}_q$ naively requires us to compute the degree-$2n$ polynomial $\mathbf{ab} \in \mathbb{F}_q[X]$ and then reduce modulo $X^n + 1$. This would make multiplying ring elements quite expensive. For our choice of $\mathcal{R}_q$, however, so long as q has $2n$-th roots of unity we can employ the *negative wrapped convolution* [33], which is a linear transform T such that if $\vec{a}, \vec{b}, \vec{c}$ represent the coefficients of $\mathbf{a}, \mathbf{b}, \mathbf{c} \in \mathcal{R}_q$ respectively, $T\vec{a} \circ T\vec{b} = T\vec{c}$ if and only if $\mathbf{c} = \mathbf{ab}$ in $\mathcal{R}_q$. From here on, T is the negative wrapped convolution.

Addition and multiplication. Following the above discussions, addition is (always) componentwise over $\mathbb{F}_q$ and multiplication is componentwise in the NTT basis. Hence to check that $\mathbf{a} + \mathbf{b} = \mathbf{c}$ or $\mathbf{a} \cdot \mathbf{b} = \mathbf{c}$ in $\mathcal{R}_q$ when $\mathbf{a}, \mathbf{b}, \mathbf{c}$ are represented in the coefficient basis as $\vec{a}, \vec{b}, \vec{c}$, we use the constraint systems $\vec{a} + \vec{b} = \vec{c}$ or $T\vec{a} \circ T\vec{b} = T\vec{c}$ respectively. Each of these 'constraints' is a shorthand for a set of n constraints, one for each dimension; i.e., $a_i + b_i = c_i$ for all $i \in [n]$, or $\langle T_i, \vec{a} \rangle \circ \langle T_i, \vec{b} \rangle = \langle T_i, \vec{c} \rangle$ for all $i \in [n]$ where T_i is the i-th row of T.

Decomposition. A simple but very important component of many primitives is computing the subset-sum decomposition of a $\mathbb{Z}_q$-element a with respect to a list of $\mathbb{Z}_q$-elements $(e_1, \ldots, e_\ell)$; that is, finding $b_1, \ldots, b_\ell$ such that $b_i \in \{0, 1\}$ and $\sum_{i=1}^{\ell} b_i e_i = a$. For example, when $e_i = 2^{i-1}$ for each i, this is the bit decomposition of a. The following simple constraint system enforces that $b_1, \ldots, b_\ell$ is the subset-sum decomposition of $a \in \mathbb{F}_q$ with respect to $(e_1, \ldots, e_\ell)$.

$$b_i(1 - b_i) = 0 \quad \forall i \in \{0, \ldots, \ell - 1\} \quad \wedge \quad \sum_{i=0}^{\ell-1} b_i e_i - a = 0$$

For the case of $e_i = 2^{i-1}$ we will use the notation $\vec{b} = \mathsf{BitDec}(a)$ to represent this constraint system. For a vector $\vec{a} \in \mathbb{F}_q^n$ and matrix $B \in \mathbb{F}_q^{n \times \ell}$ we write $B = \mathsf{BitDec}(\vec{a})$ for the constraint system "$B_j = \mathsf{BitDec}(a_j) \quad \forall j \in [k]$", for B_j the j-th row of B.

Proof of shortness. Showing that $a \in \mathbb{Z}_q$ is bounded by $\beta < (p-1)/2$, i.e. $-\beta < a < \beta$, can be achieved using its decomposition. It was observed in [30] that taking $e_1 = \lceil \beta/2 \rceil, e_2 = \lceil (\beta - b_1)/2 \rceil, \ldots, e_\ell = 1$ for $\ell = \lceil \log \beta \rceil$ yields a set of integers whose subset sums are precisely $\{0, \ldots, \beta - 1\}$. We then have that $|a| < \beta$ if and only if there exist $b_1, \ldots, b_\ell \in \{0, 1\}, c \in \{-1, 1\}$ such that $c \sum_{i=1}^{\ell} b_i e_i = a$. The prover will supply $b_1, \ldots, b_\ell$ as part of the witness. This introduces the following constraints:

$$b_i(1 - b_i) = 0 \quad \forall i \quad \wedge \quad (\sum_{i=1}^{\ell} b_i e_i - a)(\sum_{i=0}^{k-1} b_i e_i + a) = 0$$

The number of new variables is k; the number of constraints is $k + 1$. When we describe R1CS instances we will write the above constraint system as "$|a| < \beta$". For $\vec{a} \in \mathbb{Z}_q^n$, we will write "$\|\vec{a}\|_\infty < \beta$" for the constraint

system "$|a_i| < \beta \quad \forall i \in [n]$", i.e. n independent copies of the above constraint system, one for each entry of $\vec{a}$.

3.2 Proof of knowledge of RLWE secret key

We give a proof of knowledge for the relation $\mathcal{R} = \{(\mathbf{c}, \mathbf{d}; \mathbf{t}, \mathbf{e}) \in \mathcal{R}_q^4 \ : \ \mathbf{d} = \mathbf{ct} + \mathbf{e} \bmod q \ \wedge \ \|\mathbf{e}\|_\infty < \beta\}$. Let $\vec{c}, \vec{d}, \vec{t}, \vec{e} \in \mathbb{F}_q^n$ encode $\mathbf{c}, \mathbf{d}, \mathbf{t}, \mathbf{e}$ in the coefficient basis. The condition is encoded by the following constraint system:

$$T\vec{c} \circ T\vec{t} = T\vec{f} \quad \wedge \quad \vec{f} + \vec{e} = \vec{d} \quad \wedge \quad \|\vec{e}\|_\infty \leq \beta$$

where $\vec{f} \in \mathbb{F}_q^n$ should be the coefficient representation of $\mathbf{ct}$. The number of variables and constraints are bounded by $n(\log \beta + 6)$. We write $\mathsf{RLWE}_\beta(\vec{c}, \vec{d}, \vec{t}, \vec{e})$ as shorthand for the above system of constraints. Note that we did not use the fact that the verifier knows $\vec{c}, \vec{d}$; this will allow us to later use the same constraint system when $\vec{c}, \vec{d}$ are also secret. Hence, applying Theorem 2.11 yields the following.

Lemma 3.1. *There is a NIZK proof (SNARK) for the relation $\mathcal{R}$, secure and extractable in the random oracle model, with proof length $O(\lambda^2 \log^2 (n \log \beta) \log q)$.*

With our parameters as given in Section 5.5, the size of a NIZK for a single proof of knowledge of an RLWE secret key is 72 kB (obtained from our implementation Section 4 using libiop). Constraint systems for RSIS, Module-RSIS and Module-RLWE can be derived similarly.

3.3 Proof of knowledge of plaintext

We give a proof of knowledge for the relation $\mathcal{R} = \{(\mathbf{a}, \mathbf{b}, \mathbf{v}, \mathbf{w}; \mathbf{e}, \mathbf{f}, \mathbf{r}, \mu) \in \mathcal{R}_q^7 \times \mathcal{S}_1 : \ \mathbf{v} = p(\mathbf{ar} + \mathbf{e}) \wedge \mathbf{w} = p(\mathbf{br} + \mathbf{f}) + \mu \wedge \|\mathbf{e}\|_\infty, \|\mathbf{f}\|_\infty < \beta\}$. Recall that $\mathcal{S}_1 \subseteq \mathcal{R}_q$ is the set of all polynomials of degree less than n whose coefficients are in $\{0, 1\}$, which is in natural bijection with the set $\{0,1\}^n$.

Let $\vec{a}, \vec{b}, \vec{v}, \vec{w}, \vec{e}, \vec{f}, \vec{r}, \vec{\mu} \in \mathbb{F}_q^n$ be the coefficient representations of the corresponding ring elements. The condition is encoded by the following constraint system:

$$\mathsf{RLWE}_\beta(\vec{g}, \vec{a}, \vec{r}, \vec{e}) \ \wedge \ \mathsf{RLWE}_\beta(\vec{h}, \vec{b}, \vec{r}, \vec{f}) \ \wedge \ \vec{w} = p \cdot \vec{g} \ \wedge \ \vec{v} = p \cdot \vec{h} + \vec{\mu} \ \wedge \ \mu_i(\mu_i - 1) = 0 \ \forall i.$$

The number of variables is $n(2 \log \beta + 10)$; the number of constraints is $n(2 \log \beta + 15)$. This constraint system (repeated twice) is also used to build the NIZK required for the Naor-Yung construction. We write "$\vec{v}, \vec{w} = \mathsf{Enc}_p(\vec{a}, \vec{b}, \vec{r}, \vec{\mu})$" to denote the above system of constraints; $\vec{e}$ and $\vec{f}$ will be fresh variables for each instance of the system. Once again, we do not use the fact that the verifier knows $\vec{a}, \vec{b}, \vec{v}, \vec{w}$, which will be useful later.

To encrypt tn bits, we simply encrypt t n-bit blocks separately. The constraint system is then given by t copies of the above system. We will use the notation $V, W = \mathsf{Enc}_p(\vec{a}, \vec{b}, \vec{r}, \vec{\mu})$ to represent this, where V, W are $n \times k$ matrices whose rows are the encryptions of each n-bit block.

3.4 Proof of valid signature

An important component of the group signature scheme is proving knowledge of a message $\mu \in \{0,1\}^\ell$ together with a Boyen signature on μ (see Section 2.3). We first consider a simpler relation, where we prove knowledge of a signature on a publicly-known message. In the Boyen signature scheme, this corresponds to checking an inner product of ring elements, along with a proof of shortness for the signature. This corresponds to checking the relation $\mathcal{R} = \{(\mathbf{A}_\mu, \mathbf{u}; \mathbf{S}) \in (\mathcal{R}_q^{1 \times k} \times \mathcal{R}_q) \times \mathcal{R}_q^k : \mathbf{A}_\mu \mathbf{S} = \mathbf{u} \wedge \|\mathbf{S}\|_\infty < \beta\}$. Let $A, S \in \mathbb{F}_q^{k \times n}$ be the matrices whose rows are the coefficients of the entries of $\mathbf{A}_\mu, \mathbf{S}$, and let $\vec{u} \in \mathbb{F}_q^n$ be the coefficient representation of $\mathbf{u}$. We obtain the following constraint system:

$$TA_i \circ TS_i = TF_i \quad \forall i \in [k] \ , \quad \wedge \quad \sum_{i=1}^{k} F_i = \vec{u} \ , \quad \wedge \quad \|S_i\|_\infty < \beta \quad \forall i \in [k]$$

where $F \in \mathbb{F}_q^{k\times n}$, and A_i, S_i, F_i are the i-th rows of the corresponding matrices.

Now we turn to the more complex task of proving knowledge of a (secret) message and a signature on that message. Here the verifier can no longer compute $\mathbf{A}_\mu$ by itself, and so the work must be done in the proof. In particular, we check the following relation.

$$\mathcal{R} = \left\{ \begin{array}{c} ([\mathbf{A} \mid \mathbf{B}], \mathcal{A}, \mathbf{u}; \mu, \mathbf{S}) \in \mathcal{R}_q^{1\times m} \times \\ \times (\mathcal{R}_q^{1\times k})^{\ell+1} \times \mathcal{R}_q \times \{0,1\}^\ell \times \mathcal{R}_q^{m+k} \end{array} : \mathbf{A}_\mu \mathbf{S} = \mathbf{u} \wedge \|\mathbf{S}\|_\infty < \beta \right\},$$

where $\mathcal{A} = (\mathbf{A}_0, \ldots, \mathbf{A}_\ell)$ and $\mathbf{A}_\mu = [\mathbf{A} \mid \mathbf{B} \mid \mathbf{A}_0 + \sum_{i=1}^{\ell} (-1)^{\mu_i} \mathbf{A}_i]$. Let $M \in \mathbb{F}^{m\times n}$ be the matrix whose rows are the coefficients of the entries of $[\mathbf{A} \mid \mathbf{B}]$, and let $A_0, \ldots, A_\ell \in \mathbb{F}^{k\times n}$ be matrices whose rows are the coefficients of the entries of $\mathbf{A}_0, \ldots, \mathbf{A}_\ell$ respectively. Let $\mu' = ((-1)^{\mu_1}, \ldots, (-1)^{\mu_\ell})$ be the string in $\{\pm 1\}^\ell$ corresponding to the message μ. Clearly, the transform from μ to μ' is bijective. Let $A'_i \in \mathbb{F}^{n\times(\ell+1)}$ be such that the j-th column of A'_i is the i-th row of A_j (i.e., the coefficients of the i-th entry of A_j). Observe that $A'_i \cdot (1, \mu')$ is the coefficient representation of the i-th entry of $\mathbf{A}_0 + \sum_{j=1}^{\ell} \mu'_j \mathbf{A}_j$. Given this, the following constraint system captures the relation we need:

$$\begin{gathered} TM_i \circ TS_i = TF_i \quad \forall i \in [m] \ , \quad \wedge \quad (TA'_i)(1,\mu) \circ TS_{m+i} = TF_{m+i} \quad \forall i \in [k] \\ \sum_{i=1}^{k+m} F_i = \vec{u} \ , \quad \wedge \quad (1+\mu_i)\cdot(1-\mu_i) = 0 \quad \forall i \in [\ell] \ , \quad \wedge \quad \|S_i\|_\infty < \beta \quad \forall i \in [m+k] \end{gathered}$$

with $F \in \mathbb{F}_q^{(m+k)\times n}$. We will denote the above constraint system by $\mathsf{SVerify}_\beta(M, \mathcal{A}, \vec{u}, S, \mu)$, with $\mathcal{A} = (A_0, \ldots, A_\ell)$. The number of variables and constraints are bounded by $(4+\log\beta)(m+k)n + k(\max(n, \ell+1))$.

3.5 Signature generation

Here we specify the relation whose proof constitutes a signature for our group signature scheme; see Section 5.2 for details. We repeat its formal description below.

$$\mathcal{R}_S = \left\{ \begin{array}{l} (\mathbf{A}, \mathbf{B}, \mathcal{A}, \mathbf{u}, (\mathbf{a}_0, \mathbf{b}_0, \mathbf{a}_1, \mathbf{b}_1), \\ (\mathbf{V}_0, \mathbf{W}_0), (\mathbf{V}_1, \mathbf{W}_1);\ \mathbf{t}, i, \mathbf{c}, \mathbf{d}, \mathbf{e}, \mathbf{S}) \end{array} \text{ s.t. } \begin{array}{c} 1 \leftarrow \mathsf{SVerify}(\mathbf{S}, (\mathbf{c}, \mathbf{d}, i), \mathbf{A}, \mathbf{B}, \mathbf{A}_0, \ldots, \mathbf{A}_\ell, \mathbf{u}) \\ \wedge\ \mathbf{d} = \mathbf{ct} + \mathbf{e} \ \wedge\ \|\mathbf{e}\| \le \beta' \\ \wedge\ (\mathbf{V}_0, \mathbf{W}_0) \leftarrow \mathsf{Enc}(i, \mathbf{c}, \mathbf{d}, (\mathbf{a}_0, \mathbf{b}_0)) \\ \wedge\ (\mathbf{V}_1, \mathbf{W}_1) \leftarrow \mathsf{Enc}(i, \mathbf{c}, \mathbf{d}, (\mathbf{a}_1, \mathbf{b}_1)) \end{array} \right\}$$

We now describe the constraint system which represents this relation. The variables $\vec{c}$, $\vec{d}$, $\vec{e}$, i, A, B, $\mathcal{A}$, $\vec{u}$, S, $\vec{a}_0$, $\vec{a}_1$, $\vec{b}_0$, $\vec{b}_1$, V_0, W_0, V_1, W_1 are the coefficient representations of the corresponding variables in the relation. Using the notation defined in the previous subsections, the constraint system is as follows.

$$\begin{gathered} C = \mathsf{BitDec}(\vec{c}) \quad \wedge \quad D = \mathsf{BitDec}(\vec{d}) \quad \wedge \quad \vec{i} = \mathsf{BitDec}(i) \quad \wedge \\ \mathsf{SVerify}_\beta([A|B], \mathcal{A}, \vec{u}, S, (C, D, \vec{i})) \quad \wedge \quad \mathsf{RLWE}_{\beta'}(\vec{c}, \vec{d}, \vec{e}) \ , \quad \wedge \\ V_0, W_0 = \mathsf{Enc}_p(\vec{a}_0, \vec{b}_0, \vec{r}, (C, D, \vec{i})) \quad \wedge \quad V_1, W_1 = \mathsf{Enc}_p(\vec{a}_1, \vec{b}_1, \vec{r}, (C, D, \vec{i})) \end{gathered}$$

The number of variables and constraints are bounded by $(4+\log\beta)(m+k)n + 2kn\log q + 5n\log\beta + 30n + 6$. With our parameters this yields approximately 10 million variables and constraints. By applying the proof system of [7], we obtain the following lemma

Lemma 3.2. *There is a NIZK proof (SNARK) for the relation $\mathcal{R}_S$, secure and extractable in the random oracle model, with proof length $O(\log^2\left((m+k)n\log\beta + n^2\log q\right)\log q)$.*

3.6 Proof of valid decryption

The relation $\mathcal{R} = \{(\mathbf{v}, \mathbf{w}, \mu, \mathbf{a}, \mathbf{b}; \mathbf{s}, \mathbf{e}) \ : \ (\mathbf{w} - \mathbf{sv}) \bmod p = \mu \ \wedge \ \mathbf{b} = \mathbf{as} + \mathbf{e} \ \wedge \ \|\mathbf{s}\|_\infty, \|\mathbf{e}\|_\infty \leq \beta\}$, captures the statement that the prover knows the RLWE secret key corresponding to a given public key, and that a given ciphertext decrypts to a given message under this key. The constraint system is as follows.

$$\mathsf{RLWE}_\beta(\vec{a}, \vec{b}, \vec{s}, \vec{e}) \quad \wedge \quad T\vec{w} - T\vec{s} \circ T\vec{v} = T(\vec{\mu} + p\vec{h}) \quad \wedge \quad \|\vec{h}\|_\infty < (q-1)/2p$$

The final constraint ensures that $\vec{\mu} + p\vec{h}$ does not 'wrap around' modulo q. Since $\vec{v}, \vec{w}$ are public, the verifier can incorporate them into the constraint system. The number of variables and constraints is bounded by $n(\log \beta + \log(q/p) + 5)$.

3.7 Parameter choices

In this section we discuss how the parameter choices in Section 5.5 relate to the relations described in the above sections, and the resulting constraint system sizes given by our implementation (Section 4). Throughout we let q be a prime with $\log_2 q \approx 65$, and $\mathcal{R}_q = \mathbb{F}_q / \langle X^n + 1 \rangle$ with $n = 1024$. We have $\log \beta = 10$.

Proof of knowledge of RLWE secret key. Our implementation yields a constraint system with 16,383 variables and 15,361 constraints for the parameters specified. The resulting proof is 72kB in size, and is produced in roughly 40 seconds on a consumer laptop (MacBook Pro).

Proof of knowledge of plaintext. Our implementation yields a constraint system with 32,769 variables and 29,696 constraints for the parameters specified. The resulting proof is 87kB in size, and is produced in roughly three minutes.

Proof of valid signature. Here $k = \bar{m} = 67$, $m = 2\bar{m} = 134$. Proving knowledge of a message $\mu \in \{0,1\}^\ell$ and signature on μ yields at most $3 \times 10^6 + 67\ell$ constraints, for $\ell > n$. Our message size is $\ell = 2nk + \log N$, where N is the number of users in the system; we obtain roughly $12 \times 10^6 + 67 \log N$ constraints. Since the number of users will always be bounded by (say) 2^{40}, the number of constraints is bounded by 12 million.

Our implementation yields a constraint system of 2,663,451 variables and 2,530,330 constraints. This is too large to produce a proof for on our Google Cloud instance, but extrapolating from known proof sizes we expect this to be at most 150kB.

Signature generation. Our implementation yields a constraint system with 10,196,994 variables and 10,460,226 constraints. This is too large to produce a proof for, but extrapolating from known proof sizes we expect at most 250kB.

4 Implementation

The implementation was written in C++, primarily using the following libraries:

- `libff` (`https://github.com/scipr-lab/libfff`)
- `libiop` (`https://github.com/scipr-lab/libiop`)

`libff` is a C++ implementation of finite fields, and libiop includes a C++ implementation of the Aurora IOP and SNARK. The implementation took advantage of the `libiop`-provided APIs to construct the R1CS encodings of the various relations detailed in 3. Once these R1CS constraint systems were constructed, `libiop` was used to construct the Aurora IOPs, which where then compiled to zkSNARKs. Finally, the proof size of these SNARKs was measured directly.

`libiop` does not currently provide primitives to organize very large constraint systems as in this paper. To prevent the constraint systems from getting unwieldy, an additional class `ring_poly` was created to represent ring elements 3.1 as vectors of R1CS variables. This class also contains an implementation of the negative wrapped convolution (along with its inverse), which was tested by comparing with multiplication of polynomials in the 'long-form' method. In addition, now polynomial multiplication using the negative

wrapped convolution could be represented as a basic constraint and be composed as part of a larger constraint system. Similarly, bit decompositions and proofs of shortness were also represented as basic constraints.

Mirroring the definition of the relations themselves, the implementations for 3.1, 3.2 were composed by referencing the relevant smaller relations.

Several small utilites were also created in order to compute the parameters `libff` requires for the specific prime fields used in this paper, and to generate other prime fields to test how proof sizes varied with number of bits of the underlying prime field.

The constraint systems were compiled and run on a consumer-grade 2016 Macbook Pro, when running the prover and verifier could fit in memory. For the larger constraint systems such as for 3.2, a Google Cloud large-memory compute instance was used to finish constructing the proofs.

5 Group signatures

We present a dynamic group signature $\mathsf{GS} = (\mathsf{GKg}, \mathsf{UKg}, \mathsf{Join}, \mathsf{Iss}, \mathsf{GSign}, \mathsf{GVerify}, \mathsf{GOpen}, \mathsf{GJudge})$ that supports N users and guarantees non-frameability in the ROM under post-quantum assumptions. Being *dynamic* means that users can join at any time during the lifespan of the group. Our construction follows the framework by Bellare, Shi and Zhang [6]and is built from a lattice-based hash-and-sign signature $(\mathsf{SParGen}, \mathsf{SKeyGen}, \mathsf{Sign}, \mathsf{SVerify})$ (cf. Section 2.3), SNARKs (P, V), a post-quantum one-time signature scheme $(\mathsf{OTSGen}, \mathsf{OTSSign}, \mathsf{OTSVf})$ (e.g., Lamport's signature scheme with key length 2λ bits) and a CCA2-secure encryption scheme $(\mathsf{EParGen}, \mathsf{EKeyGen}, \mathsf{Enc}, \mathsf{Dec})$ (the RLWE encryption scheme [36] made CCA2-secure via the Naor-Yung paradigm [40], cf. Section 2.2). Correctness of our construction trivially follows from the correctness of the building blocks. Security can be proved along the lines of the proofs in [6] (cf. Section 5.4).

5.1 Key Generation and Joining Protocol

Let N be the maximum number of users supported by the scheme. We assume there exists a publicly available list $\mathbf{upk}$ containing the personal (OTS) verification keys of the users, i.e., $\mathbf{upk}[i] = vk_i$.

GKg*:* A trusted third party generates the parameters $spar \leftarrow \mathsf{SParGen}(1^\lambda)$ and $epar \leftarrow \mathsf{EParGen}(1^\lambda)$. The error distribution of the RLWE encryption scheme is a Gaussian distribution with standard deviation $\sigma_{RLWE} = 2\sqrt{\log q}$. Then it sets $\ell = 2n\lceil \log q \rceil + \lceil \log N \rceil$, and checks that $q \geq 4p\sqrt{\log q \log n}\sqrt{64 \log q + n}$. If that's not the case, it aborts and restarts the parameter generation. It generates the group manager's secret signing key $T_{\mathbf{A}}$ with corresponding public key $(\mathbf{A}, \mathbf{B}, \mathbf{A}_0, \ldots, \mathbf{A}_\ell, \mathbf{u})$ running the key generation algorithm $\mathsf{SKeyGen}$ of the signature scheme. Finally, it generates the opener's keys by first generating two pairs of encryption and decryption keys of the encryption scheme, $((\mathbf{a}_i, \mathbf{b}_i), \mathbf{s}_i) \leftarrow \mathsf{EKeyGen}(epar)$ for $i = 0, 1$, and then setting $opk = (\mathbf{a}_0, \mathbf{b}_0, \mathbf{a}_1, \mathbf{b}_1)$ and $osk = \mathbf{s}_0$; $\mathbf{s}_1$ is discarded. Recall that the RLWE error distribution χ is set to be a discrete Gaussian with standard deviation s_{RLWE}. By Lemma 2.1 an element $\mathbf{e} \xleftarrow{\$} \chi$ has norm bounded by $B_I = 8s_{RLWE}$.

UKg*:* The i-th user generates her OTS keys running $(sk_i, vk_i) \leftarrow \mathsf{OTSGen}(1^\lambda)$. The verification key vk_i is added as the i-th entry to the public list $\mathbf{upk}$. The keys of the user are $(usk_i, upk_i) = (sk_i, vk_i)$.

Join *and* Iss*:* The joining protocol is composed by a pair of algorithms $(\mathsf{Join}, \mathsf{Iss})$ run by the user and the group manager respectively, as showed in Figure 5.1.

- The user starts by running Join on input her key pair $((\mathbf{c}_i, \mathbf{d}_i), \mathbf{t}_i)$. The algorithm ends outputting $(\mathbf{c}_i, \mathbf{d}_i, \sigma_i, vk_i)$ to M along with a proof Π_i that the user knows $\mathbf{t}_i, \mathbf{e}_i$, i.e., a proof that $(\mathbf{c}_i, \mathbf{d}_i)$ is a RLWE pair. The signature is generated running $\mathsf{OTSSign}((\mathbf{c}_i, \mathbf{d}_i), sk_i)$, while the proof is generated running $\mathsf{P}_I(\mathbf{c}_i, \mathbf{d}_i; \mathbf{t}_i, \mathbf{e}_i)$ that is the prover algorithm of a SNARK $(\mathsf{P}_I, \mathsf{V}_I)$ for the following relation:

$$\mathcal{R}_I = \{(\mathbf{c}_i, \mathbf{d}_i; \mathbf{t}_i, \mathbf{e}_i) \in \mathcal{R}_q^4 \ : \ \mathbf{d}_i = \mathbf{c}_i\mathbf{t}_i + \mathbf{e}_i \bmod q \ \wedge \ \|\mathbf{e}_i\|_\infty \leq \beta'\}$$

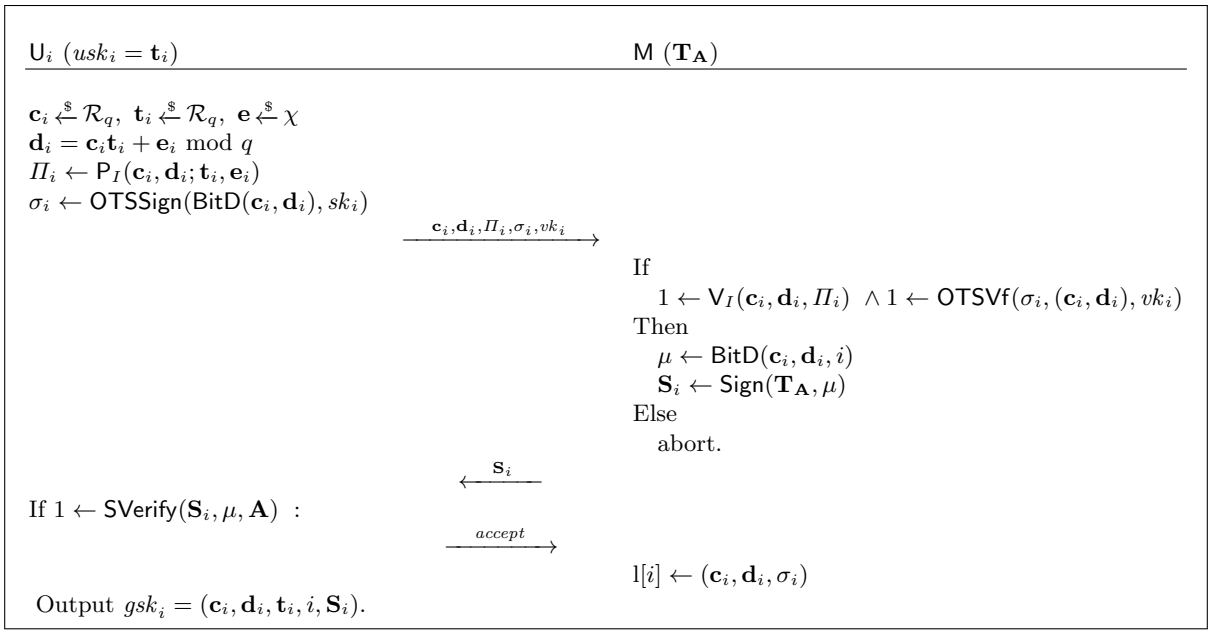

Fig. 1. Joining protocol.

where β' is an upper bound on the absolute value of the coefficients of $\mathbf{e}_i$ computed in the parameters generation phase.

- M runs $\mathsf{V}_I(\mathbf{c}_i, \mathbf{d}_i, \Pi_i)$ and $\mathsf{OTSVf}(\sigma_i, (\mathbf{c}_i, \mathbf{d}_i), vk_i)$. If any of them outputs 0, the group manager aborts. Otherwise, he signs $(\mathbf{c}_i, \mathbf{d}_i, i)$ using the signature scheme, i.e., he generates $\mathbf{S}_i$ with small norm such that $[\mathbf{A} \mid \mathbf{B} \mid \mathbf{A}_0 + \sum_{j=1}^{\ell}(-1)^{\mu_j}\mathbf{A}_j]\mathbf{S}_i = \mathbf{u} \bmod q$ where $\mu = (\mu_1, \ldots, \mu_\ell)$ is the binary expansion of $(\mathbf{c}_i, \mathbf{d}_i, i)$. Then, M sends $\mathbf{S}_i$ to U_i.
- The user verifies that $\mathbf{S}_i$ is a valid signature on $(\mathbf{c}_i, \mathbf{d}_i, i)$. If this is the case, she sends *accept* to the issuer, and sets her signing key to be $(\mathbf{t}_i, \mathbf{c}_i, \mathbf{d}_i, i, \mathbf{S}_i)$. Otherwise, she aborts.
- On input *accept*, the issuer stores in the list $\mathrm{l}[i] = (\mathbf{c}_i, \mathbf{d}_i, \sigma_i)$ and concludes the protocol.

At the end of this process, the user obtains a credential $cred_i = \mathbf{S}_i$ linked to her public key $(\mathbf{c}_i, \mathbf{d}_i)$. The user's public key $(\mathbf{c}_i, \mathbf{d}_i)$ is a RLWE sample: RLWE guarantees that the group manager cannot recover the user's secret key $\mathbf{s}$ from it.

5.2 Signing Algorithm

The signature algorithm is shown in Figure 5.3.

To produce a valid signature, a user has to prove that she has a valid credential. This means she has to prove that she has a signature by M on her user public key and group identity $(\mathbf{c}_i, \mathbf{d}_i, i)$. Moreover, to allow the opener to output a proof of honest opening, it is necessary that he can extract $\mathbf{c}_i$ and $\mathbf{d}_i$ from the signature. Hence, the user attaches to the NIZK proof also two encryptions $(\mathbf{V}_0, \mathbf{W}_0), (\mathbf{V}_1, \mathbf{W}_1)$ of the user's identity i and of the RLWE sample $(\mathbf{c}_i, \mathbf{d}_i)$ w.r.t the two RLWE encryption keys in the opener public key. Remark that this does not compromise the user, as the opener never gets the user's secret key nor the user's signing key. To guarantee that the user is not cheating by encrypting a fake credential or by encrypting different plaintexts in the two ciphertexts, the user has to prove that the two ciphertexts encrypt the same

$\mathsf{GSign}(gsk_i, gpk, opk, \mu)$
Parse $gsk_i = (\mathbf{c}_i, \mathbf{d}_i, \mathbf{t}_i, i, \mathbf{S}_i)$ and $opk = (\mathbf{a}_0, \mathbf{b}_0, \mathbf{a}_1, \mathbf{b}_1)$.
For $b = 0, 1$
$(\mathbf{V}_i, \mathbf{W}_i) \leftarrow \mathsf{Enc}(\mathsf{BitD}(\mathbf{c}_i, \mathbf{d}_i, i), (\mathbf{a}_b, \mathbf{b}_b))$
$\Pi_S \leftarrow \mathsf{P}_S(\mu; gpk, opk, (\mathbf{V}_0, \mathbf{W}_0), (\mathbf{V}_1, \mathbf{W}_1); \mathbf{t}_i, i, \mathbf{c}_i, \mathbf{d}_i, \mathbf{e}_i, \mathbf{S}_i)$
Return $\sigma = (\Pi_S, \mathbf{V}_0, \mathbf{W}_0, \mathbf{V}_1, \mathbf{W}_1)$.

$\mathsf{GVerify}(gsk_i, gpk, opk, \mu)$
Parse $gsk_i = (\mathbf{c}_i, \mathbf{d}_i, \mathbf{t}_i, i, \mathbf{S}_i)$ and $opk = (\mathbf{a}_0, \mathbf{b}_0, \mathbf{a}_1, \mathbf{b}_1)$.
For $b = 0, 1$
$(\mathbf{V}_i, \mathbf{W}_i) \leftarrow \mathsf{Enc}(\mathsf{BitD}(\mathbf{c}_i, \mathbf{d}_i, i), (\mathbf{a}_b, \mathbf{b}_b))$
$\Pi_S \leftarrow \mathsf{P}_S(\mu; gpk, opk, (\mathbf{V}_0, \mathbf{W}_0), (\mathbf{V}_1, \mathbf{W}_1); \mathbf{t}_i, i, \mathbf{c}_i, \mathbf{d}_i, \mathbf{e}_i, \mathbf{S}_i)$
Return $\sigma = (\Pi_S, \mathbf{V}_0, \mathbf{W}_0, \mathbf{V}_1, \mathbf{W}_1)$.

Fig. 2. Signing and verification algorithms of the group signature.

$(i, \mathbf{c}_i, \mathbf{d}_i)$ on which she proved she has a credential. The relation becomes:

$$\mathcal{R}_S = \left\{ \begin{array}{l} (\mathbf{A}, \mathbf{B}, \mathbf{A}_0, \ldots, \mathbf{A}_\ell, \mathbf{u}, \\ opk, (\mathbf{V}_0, \mathbf{W}_0), \\ (\mathbf{V}_1, \mathbf{W}_1); \\ \mathbf{t}_i, i, \mathbf{c}_i, \mathbf{d}_i, \mathbf{e}_i, \mathbf{S}_i) \end{array} : \begin{array}{c} 1 \leftarrow \mathsf{SVerify}(\mathbf{S}_i, \mathsf{BitD}(i, \mathbf{c}_i, \mathbf{d}_i), \mathbf{A}, \mathbf{B}, \mathbf{A}_0, \ldots, \mathbf{A}_\ell, \mathbf{u}) \\ \wedge\ \mathbf{d}_i = \mathbf{c}_i \mathbf{t}_i + \mathbf{e}_i \ \wedge\ \|\mathbf{e}_i\| \leq \beta' \\ \wedge\ (\mathbf{V}_0, \mathbf{W}_0) \leftarrow \mathsf{Enc}(\mathsf{BitD}(i, \mathbf{c}_i, \mathbf{d}_i), (\mathbf{a}_0, \mathbf{b}_0)) \\ \wedge\ (\mathbf{V}_1, \mathbf{W}_1) \leftarrow \mathsf{Enc}(\mathsf{BitD}(i, \mathbf{c}_i, \mathbf{d}_i), (\mathbf{a}_1, \mathbf{b}_1)) \end{array} \right\} \tag{2}$$

and $(\mathsf{P}_S, \mathsf{V}_S)$ is a non-interactive SNARK for $\mathcal{R}_S$ (cf. Section 3.5). The user outputs the signature $\sigma = (\mathbf{V}_0, \mathbf{W}_0, \mathbf{V}_1, \mathbf{W}_1, \Pi_S)$.

Remark 5.1. In Bellare et al.[6] the user has to add to the signature the encryption of the credential to allow the simulator to recover a forgery for the signature scheme from a valid forgery in the security proof of traceability and non-frameability. Instead, in the security proofs of our scheme the credential is extracted through extraction. This is not possible in the construction by Bellare et al., as their NIZK proof is not required to be a proof of knowledge.

5.3 Signature Verification, Opening, and the Judge Algorithm

To *verify* a signature σ on a message μ, the algorithm $\mathsf{GVerify}$ checks Π_S by outputting what $\mathsf{V}_S(\Pi_S, \mu, \mathbf{A}, \mathbf{B}, \mathbf{A}_0, \ldots, \mathbf{A}_\ell, \mathbf{u}, opk, (\mathbf{V}_0, \mathbf{W}_0, \mathbf{V}_1, \mathbf{W}_1))$ outputs.

The *opener* first runs $\mathsf{GVerify}$ on the signature. If $\mathsf{GVerify}$ returns 0 the opener outputs $(0, \epsilon)$. Otherwise he decrypts the ciphertext $(\mathbf{V}_0, \mathbf{W}_0)$ using his secret key $\mathbf{s}_0$ to recover the identity i and public key $(\mathbf{c}'_i, \mathbf{d}'_i)$ of the signer using his secret key $\mathbf{s}$. Then, to prove that the user's identity he extracted is valid, he recovers the i-th entry of the list $\mathrm{l}[i] = (\mathbf{c}_i, \mathbf{d}_i, \sigma_i)$ and checks that $(\mathbf{c}'_i, \mathbf{d}'_i) = (\mathbf{c}_i, \mathbf{d}_i)$. If that is true, he outputs $\mathrm{l}[i]$ along the $(\mathbf{c}'_i, \mathbf{d}'_i)$ he recovered from the signature. Finally, the opener produces a proof that the opening procedure was performed honestly using the decryption key *osk* corresponding to the opener's public key *opk*, i.e., he outputs a proof Π_O for the following relation:

$$\mathcal{R}_O = \left\{ \begin{array}{c} (\mathbf{V}_0, \mathbf{W}_0, i, \mathbf{c}'_i, \mathbf{d}'_i, \\ \mathbf{a}_0, \mathbf{b}_0;\ \mathbf{s}_0, \mathbf{e}_0) \end{array} : (\mathbf{W}_0 - \mathbf{s}_0 \mathbf{V}_0) \bmod p = \begin{pmatrix} \hat{i} \\ \hat{\mathbf{c}}'_i \\ \hat{\mathbf{d}}'_i \end{pmatrix} \wedge \begin{array}{c} \mathbf{b}_0 = \mathbf{a}_0 \mathbf{s}_0 + \mathbf{e}_0 \bmod q \\ \|\mathbf{s}_0\|_\infty, \|\mathbf{e}_0\|_\infty \leq \beta' \end{array} \right\}, \tag{3}$$

where $\hat{i}, \hat{\mathbf{c}}'_i, \hat{\mathbf{d}}'_i$ are the binary polynomials obtained from the binary expansions of $i, \mathbf{c}'_i, \mathbf{d}'_i$. If every check and the decryption go through, the output of the opener is $(i, \tau) = (i, (\mathbf{c}'_i, \mathbf{d}'_i, \mathbf{c}_i, \mathbf{d}_i, \sigma_i, \Pi_O))$. Otherwise, the opener outputs $(i, \tau) = (0, \epsilon)$.

The *Judge* algorithm verifies the opener claims of having opened correctly a signature. Hence, it has to verify Π_O and that the decrypted public key, the entry in the list and the certified public key of the user coincides. It takes as input $(gpk, \sigma, \mu, (i, \tau))$, i.e., the group public key, the signature $\sigma = (\Pi_S, (\mathbf{V}_0, \mathbf{W}_0, \mathbf{V}_1, \mathbf{W}_1))$ and the respective message μ, and the output of the opener $(i, \tau) = (i, (\mathbf{c}'_i, \mathbf{d}'_i, \mathbf{c}_i, \mathbf{d}_i, \sigma_i, \Pi_O))$. It recovers the public key upk_i of user i from the public list, and outputs 1 if all of the following conditions hold:

- $(i, \tau) \neq (0, \epsilon)$
- $1 \leftarrow \mathsf{GVerify}(\sigma, \mu, gpk)$

– $(\mathbf{c}, \mathbf{d}) = (\mathbf{c}', \mathbf{d}')$
– $1 \leftarrow \mathsf{V}_O(\Pi_O, \mathbf{V}_0, \mathbf{W}_0, i, \mathbf{c}'_i, \mathbf{d}'_i, \mathbf{a}_0, \mathbf{b}_0)$
– $1 \leftarrow \mathsf{OTSVf}(\sigma_i, (\mathbf{c}_i, \mathbf{d}_i), vk_i)$.

Otherwise, the algorithm outputs 0.

5.4 Correctness, Security and Parameters

Correctness follows from the correctness of the building blocks, as shown in the proof of Theorem 5.2.

Theorem 5.2 (Correctness). *If the signature, OTS, RLWE encryption and NIZK proof system are correct, the group signature described above is correct.*

Proof (Sketch of the proof of Theorem 5.2.). The proof consists of 5 steps

– proving that the joining protocol results in the group manager producing a signature on the user's RLWE pair,
– proving that a user can produce a signature through the NIZK and the RLWE encryption,
– proving that verification accepts honestly generated signatures,
– proving that the decryption of the ciphertext contained in a honestly generated signature outputs the identity of the signer and a NIZK,
– proving that the judge always accepts the output of a honest opener.

We start from the joining procedure. A user can prove it has a RLWE pair $(\mathbf{c}_i, \mathbf{d}_i)$ running P_I as $\mathbf{e}_i$ is sampled from a Gaussian (as specified in the Parameter Generation), hence it has infinity norm less than $8\sigma_{RLWE}$ thanks to Lemma 2.1. The signing algorithm succeeds as all the parameters are generated according to specifications. On the user's side, the correctness of the signature guarantees that the verification algorithm outputs 1 with high probability.

During the signing procedure we only need to make sure that the prover P_S can in fact output a NIZK for relation $\mathcal{R}_S$. This follows from the correctness of the signature, encryption and NIZK proof system. The bounds in relation $\mathcal{R}_S$ are set in Section 3 and follow from Lemma 2.1.

As the NIZK is generated honestly, the verification is guaranteed to output 1 with overwhelming probability by the correctness of the NIZK proof system. The correctness of the decryption of the RLWE ciphertext holds as long as $q \geq p\sigma_{RLWE}\sqrt{2 \cdot 16\sigma^2_{RLWE} + n} \cdot 2\sqrt{logn}$ by Theorem 2.4. As $\sigma_{RLWE} = 2\sqrt{\log q}$, q as chosen in Section 5.1 satisfies the inequality. Finally, the opener can generate a NIZK proof to guarantee the opening was performed correctly, as the secret opening key $\mathbf{s}_O$ and $\mathbf{s}_O$ are again sampled from a Gaussian with standard deviation σ_{RLWE}, hence they have infinity norm bound by $8\sigma_{RLWE}$ w.h.p. by Lemma 2.1.

The judge outputs 1 when receiving as input a honestly generated opening thanks to the correctness of the OTS and of the NIZK proof system.

Our group signature guarantees anonymity, traceability and non-frameability, meaning that it protects also against a corrupted group manager trying to frame a honest user. More precisely, the scheme is proven secure in the Random Oracle Model under quantum-safe assumptions. This essentially means that the scheme is provably secure against a classical adversary (i.e., an adversary that can only ask classical queries to the random oracle) that has access to a quantum computer.

Anonymity relies on simulation soundness and zero-knowledge of the proof system, and on the IND-CPA property of the encryption. The construction does not require the IND-CCA2 security of the encryption, as in the proof we exploit a step similar to the Sahai extension [42] of the Naor-Yung approach [40].

Theorem 5.3 (Anonymity). *The group signature scheme* GS *is anonymous in the Random Oracle Model under the zero-knowledge and simulation soundness property of the NIZK proof system and under the IND-CPA security of the encryption scheme.*

Proof. Assume A is an adversary against anonymity interacting with a simulator B. We prove through game hops that the experiment $\mathsf{Exp}^{an}_{\mathsf{A},0}(1^\lambda)$ is indistinguishable from the experiment $\mathsf{Exp}^{an}_{\mathsf{A},1}(1^\lambda)$ (cf. Figure A for the anonymity experiment). The success probability of A is:

$$\begin{aligned}\epsilon_A &= \left|\mathsf{Pr}\left[b = b' \ : \ b \xleftarrow{\$} \{0,1\}, b' \leftarrow \mathsf{Exp}^{an}_{\mathsf{A},b}(1^\lambda)\right] - \frac{1}{2}\right| \\ &= \frac{1}{2}\left|\mathsf{Pr}\left[b' = 0 \ : \ b' \leftarrow \mathsf{Exp}^{an}_{\mathsf{A},0}(1^\lambda)\right] + \mathsf{Pr}\left[b' = 1 \ : \ b' \leftarrow \mathsf{Exp}^{an}_{\mathsf{A},1}(1^\lambda)\right] - 1\right| \\ &= \frac{1}{2}\left|\mathsf{Pr}\left[b' = 0 \ : \ b' \leftarrow \mathsf{Exp}^{an}_{\mathsf{A},0}(1^\lambda)\right] + \mathsf{Pr}\left[b' = 0 \ : \ b' \leftarrow \mathsf{Exp}^{an}_{\mathsf{A},1}(1^\lambda)\right]\right| \ .\end{aligned}$$

We proof that this probability is negligible if the rNIZK is zero-knowledge through a standard sequence of game hops. Let $\mathbf{Game_i}$ the probability that A outputs 0 at the end of the i-th game.

Game 0. Game 0 executes $\mathsf{Exp}^{an}_{\mathsf{A},0}(1^\lambda)$. Hence,

$$\mathsf{Pr}[\mathbf{Game_0}] = \mathsf{Pr}\left[b' = 0 \ : \ b' \leftarrow \mathsf{Exp}^{an}_{\mathsf{A},0}(1^\lambda)\right] \ .$$

Game 1. In game 1 B simulates the NIZK proof in the oracle Chall using the simulator Σ_S of the proof system $(\mathsf{P}_S, \mathsf{V}_S)$. An adversary that can distinguish this game from the previous one can be used to break the ZK property of the NIZK proof system. Indeed, an algorithm B_1 with access to an oracle $\mathcal{O}_{NIZK}$ that outputs either simulated or honestly generated proofs can exploit A to distinguish the outputs of such oracle as follows. B_1 runs the anonymity experiment honestly but Chall. When it has to generate the challenge signature on μ_0, B_1 queries it to the oracle instead. It is clear that if the oracle outputs a simulated proof, this is exactly Game 1 and if the proof is honestly generated, A is playing exactly Game 0. At the end of the interaction, B_1 outputs exactly the same bit b' output by A. Hence, the success probability ϵ_1 of B_1 is

$$\begin{aligned}\epsilon_1 &= \left|\mathsf{Pr}\left[b' = b \ : \ b \xleftarrow{\$} \{0,1\}, \ b' \leftarrow \mathsf{Exp}^{ZK-b}_{\mathsf{B}_1}(1^\lambda)\right] - \tfrac{1}{2}\right| \\ &= \tfrac{1}{2}\left|\mathsf{Pr}\left[b' = 0 \ : \ b' \leftarrow \mathsf{Exp}^{ZK-0}_{\mathsf{B}_1}(1^\lambda)\right] + \mathsf{Pr}\left[b' = 1 \ : \ b' \leftarrow \mathsf{Exp}^{ZK-1}_{\mathsf{B}_1}(1^\lambda)\right] - 1\right| \\ &= \tfrac{1}{2}\left|\mathsf{Pr}\left[b' = 0 \wedge b' \leftarrow \mathsf{Exp}^{ZK-0}_{\mathsf{B}_1}(1^\lambda)\right] - \mathsf{Pr}\left[b' = 0 \wedge b' \leftarrow \mathsf{Exp}^{ZK-1}_{\mathsf{B}_1}(1^\lambda)\right]\right| \\ &= \tfrac{1}{2}\left|\mathsf{Pr}\left[\mathbf{Game_0}\right] - \mathsf{Pr}\left[\mathbf{Game_1}\right]\right| \ ,\end{aligned}$$

where we denoted by $\mathsf{Exp}^{ZK-b}_{\mathsf{B}_1}$ the experiment in which B_1 has access to an oracle $\mathcal{O}_{NIZK}$ that implements either the honest prover when $b = 0$, or the simulator when $b = 1$.

Game 2. In Game 2 B does everything as in Game 1, except that now it generates $(\mathbf{V}_1, \mathbf{W}_1)$ as an encryption of $1^{\bar{\ell}}$ (where $\bar{\ell}$ is the length of the plaintext) while $(\mathbf{V}_0, \mathbf{W}_0)$ is still an encryption of $id^*_0 = (i^*_0, \mathbf{c}^*_0, \mathbf{d}^*_0)$. The IND-CPA property of the encryption guarantees the indistinguishability of the games. Namely, let A be an adversary such that $\mathsf{Pr}[\mathbf{Game_1}] - \mathsf{Pr}[\mathbf{Game_2}]$ is non-negligible. Then B_2 can win the IND-CPA experiment exploiting A as follows. Upon receiving $(\mathbf{a}_1, \mathbf{b}_1)$ from the oracle, B_2 generates $(\mathbf{a}_0, \mathbf{b}_0)$ honestly and sends $opk = (\mathbf{a}_0, \mathbf{b}_0, \mathbf{a}_1, \mathbf{b}_1)$ to A. When A sends back the identities (id^*_0, id^*_1), B_2 generates $(\mathbf{V}_0, \mathbf{W}_0)$ as an encryption of $id^*_0 = (i^*_0, \mathbf{c}^*_0, \mathbf{d}^*_0)$, sends id^*_1 to the encryption oracle and generates the proof Π_S using the simulator. B_2 outputs the same bit b' output by A. Remark that if the encryption outputs an encryption of id^*_1, then B_2 is implementing Game 1, otherwise this is exactly Game 2. Hence, the success probability ϵ_2 of B_2 is

$$\epsilon_2 = \frac{1}{2}\left|\mathsf{Pr}[\mathbf{Game_1}] - \mathsf{Pr}[\mathbf{Game_2}]\right| \ .$$

Game 3. In Game 3 B does everything as in Game 2, except that now it generates $(\mathbf{V}_1, \mathbf{W}_1)$ as an encryption of $id^*_1 = (i^*_1, \mathbf{c}^*_1, \mathbf{d}^*_1)$ ($(\mathbf{V}_0, \mathbf{W}_0)$ is still an encryption of $id^*_0 = (i^*_0, \mathbf{c}^*_0, \mathbf{d}^*_0)$). Again, the IND-CPA property of the encryption guarantees the indistinguishability of the games. Indeed, an adversary A such

that $\mathsf{Pr}[\mathbf{Game_2}] - \mathsf{Pr}[\mathbf{Game_3}]$ is non-negligible can be exploited by B_3 to win the IND-CPA experiment exactly as before. Hence, the success probability ϵ_3 of B_3 is

$$\epsilon_3 = \frac{1}{2} \left| \mathsf{Pr}[\mathbf{Game_2}] - \mathsf{Pr}[\mathbf{Game_3}] \right| \ .$$

Game 4. In Game 4 the simulator does everything as in Game 3 except the generation of the opening keys and of the opening oracle $\mathcal{O}_{\mathsf{GOpen}}$. Indeed, when generating the opening keys, B preserves $\mathbf{s}_1$ instead of $\mathbf{s}_0$, and then it performs the decryption in $\mathcal{O}_{\mathsf{GOpen}}$ w.r.t. $\mathbf{s}_1$. The only way that an adversary can distinguish the games is if it can submit a valid signature σ whose two RLWE ciphertexts encrypt different messages. This would break the simulation soundness of the NIZK proof system. Indeed, an algorithm B_4 would break the soundness exploiting A as follows. When generating the opening keys, it would keep $\mathbf{s}_0$ as well. Whenever it receives a decryption query, it would decrypt both $(\mathbf{V}_0, \mathbf{W}_0)$ and $(\mathbf{V}_1, \mathbf{W}_1)$, checking that the resulting plaintexts are equal. If that is not the case, then B_4 can return $((\mu, vk, gpk, opk, (\mathbf{V}_0, \mathbf{W}_0), (\mathbf{V}_1, \mathbf{W}_1), \mathbf{t}_i, i, \mathbf{c}_i, \mathbf{d}_i, \mathbf{e}_i, \mathbf{S}_i), \Pi_S)$ as a proof of false statement. Since making this query s the only way A can distinguish the two games, the algorithm B_3 has success probability

$$\epsilon_4 = \left| \mathsf{Pr}[\mathbf{Game_3}] - \mathsf{Pr}[\mathbf{Game_4}] \right| \ .$$

Game 5. In Game 5 B does everything as in Game 4, except that now it generates $(\mathbf{V}_0, \mathbf{W}_0)$ as an encryption of $1^{\bar{\ell}}$ (where $\bar{\ell}$ is the length of the plaintext) while $(\mathbf{V}_1, \mathbf{W}_1)$ is an encryption of $id_1^* = (i_1^*, \mathbf{c}_1^*, \mathbf{d}_1^*)$. As before, the IND-CPA property of the encryption guarantees the indistinguishability of the games. Remark that a simulator B_5 trying to win the IND-CPA experiment exploiting A never has to decrypt $(\mathbf{V}_0, \mathbf{W}_0)$, thanks to the key switching in the previous game. The success probability ϵ_5 of B_5 is

$$\epsilon_5 = \frac{1}{2} \left| \mathsf{Pr}[\mathbf{Game_4}] - \mathsf{Pr}[\mathbf{Game_5}] \right| \ .$$

Game 6. In Game 6 B does everything as in Game 5, except that now it generates $(\mathbf{V}_0, \mathbf{W}_0)$ as an encryption of $id_1^* = (i_1^*, \mathbf{c}_1^*, \mathbf{d}_1^*)$. As before, a simulator B_5 trying to win the IND-CPA experiment exploiting A has success probability

$$\epsilon_6 = \frac{1}{2} \left| \mathsf{Pr}[\mathbf{Game_5}] - \mathsf{Pr}[\mathbf{Game_6}] \right| \ .$$

Game 7. In Game 7, B reverts back to decrypt using $\mathbf{s}_0$ in the opening oracle. As in Game 4, an algorithm B_7 breaks the soundness exploiting an adversary such that $\mathsf{Pr}[\mathbf{Game_6}] - \mathsf{Pr}[\mathbf{Game_7}$ is non negligible has success probability

$$\epsilon_7 = \left| \mathsf{Pr}[\mathbf{Game_6}] - \mathsf{Pr}[\mathbf{Game_7}] \right| \ .$$

Game 8. In Game 8 B reverts back to generating the proof Π_S^* in the challenge signature honestly. This is equivalent to B running Chall_1, hence Game 8 is exactly $\mathsf{Exp}_{\mathsf{A},1}^{an}(1^\lambda)$, and it holds that $\mathsf{Pr}[\mathbf{Game_8}] = \mathsf{Pr}\left[b' = 0 \ : \ b' \leftarrow \mathsf{Exp}_{\mathsf{A},1}^{an}(1^\lambda)\right]$. Moreover, analogously to Game 1, an adversary distinguishing Game 8 from Game 7 allows to construct a distinguisher B_8 that has advantage

$$\epsilon_8 = \frac{1}{2} \left| \mathsf{Pr}\left[\mathbf{Game_7}\right] - \mathsf{Pr}\left[\mathbf{Game_8}\right] \right| \ ,$$

in breaking the zero-knowledge property of the proof system.

This yields that the advantage of A in breaking anonymity is bounded by

$$\begin{aligned}
\epsilon &= \frac{1}{2}\left|\mathsf{Pr}\left[b'=0 \ : \ b' \leftarrow \mathsf{Exp}^{an}_{\mathsf{A},0}(1^\lambda)\right] + \mathsf{Pr}\left[b'=0 \ : \ b' \leftarrow \mathsf{Exp}^{an}_{\mathsf{A},1}(1^\lambda)\right]\right| \\
&= \frac{1}{2}\left|\mathsf{Pr}\left[\mathbf{Game_0}\right] - \mathsf{Pr}\left[\mathbf{Game_8}\right]\right| \\
&= \frac{1}{2}\left|\sum_{i=0}^{7} \mathsf{Pr}\left[\mathbf{Game_i}\right] - \mathsf{Pr}\left[\mathbf{Game_{i+1}}\right]\right| \\
&\leq \sum_{i=0}^{7} \frac{1}{2}\left|\mathsf{Pr}\left[\mathbf{Game_i}\right] - \mathsf{Pr}\left[\mathbf{Game_{i+1}}\right]\right| \\
&= \epsilon_1 + \epsilon_2 + \epsilon_3 + \frac{1}{2}\epsilon_4 + \epsilon_5 + \epsilon_6 + \frac{1}{2}\epsilon_7 + \epsilon_8 \\
&= 2\epsilon_{ZK} + \epsilon_{SS} + 4\epsilon_{CPA} \ ,
\end{aligned}$$

where ϵ_{ZK}, ϵ_{SS}, and ϵ_{CPA} are the advantage in breaking the zero-knowledge property of the NIZK proof system, the simulation soundness property of the NIZK proof system and the IND-CPA security of the encryption scheme. □

Theorem 5.4 (Traceability). *The group signature scheme* GS *satisfies traceability in the Random Oracle Model if the signature scheme is eu-acma secure and the proof system is a sound argument of knowledge.*

Proof. Assume A is a PPT algorithm that breaks the traceability of the GS with non-negligible advantage $\mathsf{Adv}^{\mathsf{A}}_{trac}(\lambda)$. We define an adversary B (shown in Figure 3) that breaks the eu-acma security of the signature with non negligible probability, assuming the NIZK proof is a sound argument of knowledge.

The simulator B runs the eu-acma experiment in Figure A. At the beginning of the experiment, B receives the signature verification key svk, and access to a signing oracle $\mathcal{O}_{\mathsf{Sig}}$.

B instantiates the group signature GS as follows. It generates honestly the opener's keys (opk, osk) running $\mathsf{EKeyGen}$, and sets the issuer's public key to be $gpk = svk$. Then it implements the oracles according to their definition (cf. Appendix A), except for $\mathcal{O}_{\mathsf{lss}}$ and AddU. When answering these oracles, B produces the signature $\mathbf{S}$ by querying the signature oracle $\mathcal{O}_{\mathsf{Sig}}$ as shown in Figure 3.

At the end of the experiment, the adversary A outputs a valid pair message-signature (μ^*, σ^*). Consider the following events:

E_1**:** the signature is valid but the opener cannot recover a valid signer's identity:
$1 \leftarrow \mathsf{GVerify}(\mu^*, \sigma^*, gpk, opk) \ \wedge \ (i^* = 0)$;
E_2**:** the signature is valid, the opener can recover a valid identity but it cannot prove that the opening was performed correctly:
$1 \leftarrow \mathsf{GVerify}(\mu^*, \sigma^*, gpk, opk) \ \wedge \ (i^* \neq 0) \ \wedge \ 0 \leftarrow \mathsf{GJudge}(gpk, opk, upk_{i^*}, \mu^*, \sigma^*, i^*, \tau^*)$;
E**:** $E_1 \vee E_2$;
S**:** the signature is valid and the opening is correct: $(gpk, opk, c_0, c_1, gsk_i, \mathbf{e}) \in \mathcal{R}_S$;

where $(i^*, \tau^*) \leftarrow \mathsf{GOpen}(\mu^*, \sigma^*, gpk, osk)$. Notice that E_1 and E_2 are disjoint. Then we can write the advantage of A in breaking the traceability of GS as:

$$\mathsf{Adv}^{\mathsf{A}}_{trac}(\lambda) = \mathsf{Pr}[E] = \mathsf{Pr}[E \wedge \bar{S}] + \mathsf{Pr}[E_1 \wedge S] + \mathsf{Pr}[E_2 \wedge S] \ .$$

We now compute the three probabilities.

The event $E \wedge \bar{S}$ (remark that $\bar{S}$ means that the signature is valid, as we assume A outputs a signature that passes verification, and the opening is *not* correct) corresponds to A breaking the soundness of the SNARK. Indeed, if the opening is not correct either there is no entry $\mathrm{l}[i]$ or $\mathrm{l}[i]$ is not equal to the output of the decryption (i is the decrypted identity). In both cases B can parse $\sigma^* = (\Pi^*_S, \mathbf{V}^*_0, \mathbf{W}^*_0, \mathbf{V}^*_1, \mathbf{W}^*_1)$ and send $(\Pi^*_S, \mu^*, gpk, opk, (\mathbf{V}^*_0, \mathbf{W}^*_0, \mathbf{V}^*_1, \mathbf{W}^*_1))$ as a proof of false statement. From the previous analysis it holds:

$$\begin{aligned}
\mathsf{Pr}[E \wedge \bar{S}] &= \mathsf{Pr}[1 \leftarrow \mathsf{GVerify}(\mu^*, \sigma^*, gpk, opk) \ \wedge \ (gpk, opk, c_0, c_1, gsk_i, \mathbf{e}) \notin \mathcal{R}_S] \\
&\leq 2^{-\lambda}
\end{aligned}$$

$\mathsf{B}^{\mathcal{O}_{\mathsf{Sig}},\ \mathsf{A}}(gpk)$
$(opk, osk) \leftarrow \mathsf{EKeyGen}(1^\lambda)$
$\mathrm{CU} \leftarrow \emptyset$
$\mathrm{HU} \leftarrow \emptyset$
$\mathrm{GSig} \leftarrow \emptyset$
$(\mu^*, sig^*, sig') \leftarrow \mathsf{GF}_{\mathsf{A}}^{\mathcal{O}_{\mathsf{Iss}},\ \mathsf{AddU},\ \mathsf{USK},\ \mathsf{CrptU},\ \mathsf{RReg}}(gpk, opk, osk)$
If $0 \leftarrow \mathsf{GVerify}(\mu^*, \sigma^*, gpk, opk)$, abort.
Else $(i^*, \tau^*) \leftarrow \mathsf{GOpen}(\mu^*, \sigma^*, gpk, osk)$.
If $(i^* = \epsilon) \ \vee \ 0 \leftarrow \mathsf{GJudge}(gpk, opk, upk_{i^*}, \mu^*, \sigma^*, i^*, \tau^*)$ abort.
Else $((\bar{\mathbf{c}}_i, \bar{\mathbf{d}}_i, \bar{i}), \bar{\mathbf{S}}) \leftarrow \mathsf{E}(sig^*, sig')$.
Return $((\bar{\mathbf{c}}_i, \bar{\mathbf{d}}_i, \bar{i}), \bar{\mathbf{S}})$.

$\mathcal{O}_{\mathsf{Iss}}(\mathbf{c}_i, \mathbf{d}_i, \Pi_i, ots_i, vk_i)$
$b \leftarrow \mathsf{V}_I(\mathbf{c}_i, \mathbf{d}_i, \Pi_i)$
$b' \leftarrow \mathsf{OTSVf}(ots_i, (\mathbf{c}_i, \mathbf{d}_i), vk_i)$
If $(b = 0 \ \vee \ b' = 0)$ abort.
$\mu \leftarrow \mathsf{BitD}(\mathbf{c}_i, \mathbf{d}_i, i)$
$\mathbf{S}_i \leftarrow \mathcal{O}_{\mathsf{Sig}}(\mu)$
$\mathrm{l}[i] \leftarrow (\mathbf{c}_i, \mathbf{d}_i, ots_i, vk_i)$
$l[i] \leftarrow (\mathbf{c}_i, \mathbf{d}_i, ots_i, vk_i)$
Return $i, \mathbf{S}_i$

$\mathsf{AddU}(i)$
If $i \in \mathrm{CU} \cup \mathrm{HU}$ abort.
$(sk, vk) \leftarrow \mathsf{OTSGen}(1^\lambda)$
$\mathrm{HU}[i] \leftarrow (vk, sk)$
$\mathbf{c}_i \xleftarrow{\$} \mathcal{R}_q, \ \mathbf{t}_i \xleftarrow{\$} \mathcal{R}_q, \ \mathbf{e} \xleftarrow{\$} \chi$
$\mathbf{d}_i = \mathbf{c}_i \mathbf{t}_i + \mathbf{e}_i \bmod q$
$\Pi_i \leftarrow \mathsf{P}_I(\mathbf{c}_i, \mathbf{d}_i; \mathbf{t}_i, \mathbf{e}_i)$
$ots_i \leftarrow \mathsf{OTSSign}(\mathsf{BitD}(\mathbf{c}_i, \mathbf{d}_i), sk_i)$
$\mu \leftarrow \mathsf{BitD}(\mathbf{c}_i, \mathbf{d}_i, i)$
$\mathbf{S}_i \leftarrow \mathcal{O}_{\mathsf{Sig}}(\mu)$
$\mathrm{l}[i] \leftarrow (\mathbf{c}_i, \mathbf{d}_i, ots_i)$
$l[i] \leftarrow (\mathbf{c}_i, \mathbf{d}_i, ots_i, vk_i)$
$gsk_i = (\mathbf{c}_i, \mathbf{d}_i, \mathbf{t}_i, i, \mathbf{S}_i)$.
$\mathrm{HU}[i] \leftarrow gsk_i$
Return $upk_i = vk$.

Fig. 3. Simulator for the proof of traceability.

In the event $E_1 \wedge S$ the opener decrypts the ciphertext contained in the signature to obtain a user's identity and public key $(i, \mathbf{c}'_i, \mathbf{d}'_i)$. Then, it recovers the i-th entry of the list $\mathrm{l}[i]$. There can be two cases: either $\mathrm{l}[i] = \perp$ or $\mathrm{l}[i] = (\mathbf{c}_i, \mathbf{d}_i, ots_i)$ and $(\mathbf{c}_i, \mathbf{d}_i) \neq (\mathbf{c}'_i, \mathbf{d}'_i)$. Both cases implies that A did not query the signing oracle on $(\mathbf{c}'_i, \mathbf{d}'_i)$. Hence, B can break the unforgeability of the signature scheme using the extractor of the SNARK to obtain a valid signature $\bar{\mathbf{S}}$ on $(\mathbf{c}'_i, \mathbf{d}'_i)$. Therefore,

$$\mathsf{Pr}[E_1 \wedge S] \leq 8(1 - \nu(\lambda))\mathsf{Adv}_{\mathsf{B}}^{eu-acma}(\lambda) \ ,$$

where $1 - \nu(\lambda)$ comes from the success probability of the extractor and the factor 8 from the Generalized Forking Lemma [5], that is needed to get the input for the extractor. The runtime of B is $t_{\mathsf{A}} \cdot 8n^2 q_H / \epsilon_{\mathsf{A}} \cdot \ln(8n/\epsilon_{\mathsf{A}})$, where t_{A} is the runtime of A, $\epsilon_{\mathsf{A}} = \mathsf{Pr}[E_1 \wedge S]$, and $q_H = poly(\lambda)$ as an algorithm that runs in polynomial time can query the random oracle at most a polynomial number of times. Remark that B makes N queries to the signing oracle as N is the bound on the number of users supported by the scheme.

Finally, it holds that $\mathsf{Pr}[E_2 \wedge S] = 0$. To verify this claim, let us analyze the algorithm GJudge. Upon receiving $(gpk, opk, upk_i, \mu, \sigma, i, \tau)$, the algorithm parses $\tau = (\mathbf{V}_0, \mathbf{W}_0, i, \mathbf{c}'_i, \mathbf{d}'_i, \mathbf{a}_0, \mathbf{b}_0, \Pi_O)$, and recovers $\mathrm{l}[i]$. Given that the opener outputs $(i, \tau) \neq (0, \epsilon)$, the user's public key obtained by the opener from the decryption $(\mathbf{c}'_i, \mathbf{d}'_i)$ should be equal to the entry $\mathrm{l}[i] = (\mathbf{c}, \mathbf{d})$ output by the Opener, i.e., $(\mathbf{c}'_i, \mathbf{d}'_i) = (\mathbf{c}_i, \mathbf{d}_i)$. All the verification algorithms go through, as the opener executed honestly the verification of the one-time signature ots, and the event S implies that $(gpk, opk, c_0, c_1, gsk_i, \mathbf{e}) \in \mathcal{R}_S$, hence the verification of the proof Π_O produced by the opener outputs 1. Therefore, if the opener is honest the algorithm GJudge outputs 1 and the claim follows.

In conclusion, the advantage of A against GS is

$$\mathsf{Adv}_{\mathsf{A}}^{trac}(\lambda) \leq 2^{-\lambda} + 8(1 - \nu(\lambda))\mathsf{Adv}_{\mathsf{B}}^{eu-acma}(\lambda) + \nu(\lambda) \ ,$$

that is negligible if the signature is eu-acma secure and the proof system is sound and a proof of knowledge. □

Theorem 5.5 (Non-Frameability). *The group signature scheme* GS *satisfies non-frameability in the Random Oracle Model if the proof system is a zero-knowledge argument of knowledge, the OTS is a OTS, and* $RLWE_{1,\mathcal{U}(\mathcal{S}_1)}$ *is hard.*

Proof. Let A be a PPT algorithm which wins the non-frameability experiment in Figure A with advantage $\epsilon_{\mathsf{A}} = \mathsf{Adv}^{non-fr}_{\mathsf{A}}(\lambda)$. We start analyzing the event $E =$"A succeeds". Consider the following events:

F**:** $1 \leftarrow \mathsf{GVerify}(\mu^*, \sigma^*, gpk, opk), \mathrm{HU}[i^*] \neq \perp$, A did not query $\mathsf{USK}(i), 1 \leftarrow \mathsf{GJudge}(gpk, opk, upk_{i^*}, \mu^*, \sigma^*, i^*, \tau^*)$, $(i^*, \mu^*) \notin \mathrm{GSig}$, where $(i^*, \tau^*) \leftarrow \mathsf{GOpen}(\mu^*, \sigma^*, gpk, osk)$;
S_1**:** $(\mu^*, \sigma^*, gpk, opk) \in \mathcal{R}_S$;
S_2**:** $(gpk, opk, \mu^*, \sigma^*, \tau^*, i^*) \in \mathcal{R}_O$;
P**:** $(\mathbf{c}'_i, \mathbf{d}'_i) = (\mathbf{c}_i, \mathbf{d}_i)$, where $(\mathbf{c}_i, \mathbf{d}_i)$ is obtained from $\mathrm{l}[i]$ and $(\mathbf{c}'_i, \mathbf{d}'_i)$ from the opening.

Then the advantage of the adversary in winning the non-frameability experiment is

$$\mathsf{Adv}^{nf}_{\mathsf{A}}(\lambda) = \mathsf{Pr}[F] \leq \mathsf{Pr}[F \wedge \bar{S}_1] + \mathsf{Pr}[F \wedge \bar{S}_2] + \mathsf{Pr}[F \wedge P \wedge S_1 \wedge S_2] + \mathsf{Pr}[F \wedge \bar{P} \wedge S_1 \wedge S_2]$$

because the events S_1 and S_2 are not disjoint[3]. In the following we compute the probabilities of these events.

The soundness of the proof systems yields that $\mathsf{Pr}[F \wedge \bar{S}_1] \leq 2^{-\lambda}$ and $\mathsf{Pr}[F \wedge \bar{S}_1] \leq 2^{-\lambda}$.

If the event $F \wedge P \wedge S_1 \wedge S_2$ happens, we can construct an algorithm B_1 that solves the Search version of RLWE, i.e., that given $(\bar{\mathbf{c}}, \bar{\mathbf{d}})$ finds $\bar{\mathbf{t}}$ such that $\bar{\mathbf{d}} = \bar{\mathbf{c}}\bar{\mathbf{s}} + \bar{\mathbf{e}}$ for some small error $\bar{\mathbf{e}}$.

Lemma 5.6. $\mathsf{Pr}[F \wedge P \wedge S_1 \wedge S_2] \leq \mathsf{Adv}^{RLWE}_{\mathsf{B}}(\lambda) \cdot 8N(\lambda)(1 - \nu(\lambda))$.

Proof. The algorithm B_1 (cf. Figure 4) is given access to an oracle $\mathcal{O}_{RLWE}$ that outputs RLWE pairs $(\mathbf{c}, \mathbf{b})$ when prompted, and simulates all oracles according to their definitions but USK, $\mathcal{O}_{\mathsf{Iss}}$, SndToU and $\mathcal{O}_{\mathsf{GSign}}$. Indeed, B samples a random u and simulates all honest users honestly but the u-th, whose RLWE pair is set to be the pair $(\bar{\mathbf{c}}, \bar{\mathbf{d}})$ output by $\mathcal{O}_{RLWE}$. Then, B simulates USK honestly unless A queries for the user u; in such case, B aborts. If the adversary runs $\mathcal{O}_{\mathsf{Iss}}$ with user identity u (where u has not been assigned yet), B_1 samples another u and then runs the algorithm Iss honestly. Whenever A queries the oracle SndToU for the user u, B sends $(\bar{\mathbf{c}}, \bar{\mathbf{d}})$ as user's keys, and simulates the SNARK of $\bar{\mathbf{t}}$. In a similar way, whenever the user queries $\mathcal{O}_{\mathsf{GSign}}$ for a signature by the user u, B simulates the proof Π_S. If the adversary successfully outputs a forgery, this means that A was able to generate a SNARK of, among other things, small $\bar{\mathbf{t}}$ and $\bar{\mathbf{e}}$ such that $\bar{\mathbf{d}} = \bar{\mathbf{c}}\bar{\mathbf{s}} + \bar{\mathbf{e}}$. Hence, B can recover $\bar{\mathbf{t}}$ rewinding A and exploiting the extractor of the proof system $(\mathsf{P}_S, \mathsf{V}_S)$. Given that solving the search version of RLWE is equivalent to solving the decisional version (cf. [35]), the advantage of B in solving RLWE is $\mathsf{Pr}[F \wedge P \wedge S_1 \wedge S_2] \leq \mathsf{Adv}^{RLWE}_{\mathsf{B}}(\lambda) \cdot 8N(\lambda)(1 - \nu(\lambda))$ where the factor 8 comes from the Generalized Forking Lemma, $N(\lambda)$ is the number of users and $(1 - \nu(\lambda))$ comes from the success probability of the extractor of the SNARK.

Finally, consider the event $F \wedge \bar{P} \wedge S_1 \wedge S_2$. We construct an algorithm B_2 that breaks the unforgeability of the OTS exploiting A.

Lemma 5.7. $\mathsf{Adv}^{OTS}_{\mathsf{B}}(\lambda) \geq \mathsf{Pr}[F \wedge \bar{P} \wedge S_1 \wedge S_2] \cdot 1/N(\lambda)$ *where* $N(\lambda)$.

[3] Indeed, if S_1 and S_2 are not disjoint events it holds that, for all sets P in the set space Ω:

$$\bar{S}_1 \cup \bar{S}_2 \cup \big[(P \cap S_1 \cap S_2) \cup (\bar{P} \cap S_1 \cap S_2)\big] = \bar{S}_1 \cup \bar{S}_2 \cup (S_1 \cap S_2) = \Omega \ ,$$

hence $\{\bar{S}_1, \bar{S}_2, (P \cap S_1 \cap S_2), (\bar{P} \cap S_1 \cap S_2)\}$ covers the space Ω.

$\mathsf{B}_1^{\mathcal{O}_{RLWE},\,\mathsf{A}}(1^\lambda)$
$(gpk, gsk, opk, osk) \leftarrow \mathsf{GKg}(1^\lambda)$
$\mathrm{CU} \leftarrow \emptyset$, $\mathrm{HU} \leftarrow \emptyset$, $\mathrm{GSig} \leftarrow \emptyset$
$u \xleftarrow{\$} \{1, \ldots, N\}$
$(\mu^*, \sigma_1^*, \sigma_2^*) \leftarrow \mathsf{GF}_{\mathsf{A}}^{\mathcal{O}_{\mathsf{GSign}},\ \mathsf{AddU},\ \mathsf{USK},\ \mathsf{CrptU},\ \mathsf{RReg},\ \mathsf{WReg},\ \mathsf{SndToU}}(gpk, gsk, opk, osk)$
$(i, \tau) \leftarrow \mathsf{GOpen}(\sigma_1^*, osk)$
If $i \neq u$ abort.
For $j = 1, 2$, parse $\sigma_j^* = (\Pi_{S,j}^*, \mathbf{V}_{0,j}^*, \mathbf{W}_{0,j}^*, \mathbf{V}_{1,j}^*, \mathbf{W}_{1,j}^*)$
$(\bar{\mathbf{t}}_u, \bar{u}, \bar{\mathbf{c}}_u, \bar{\mathbf{d}}_u, \bar{\mathbf{e}}_u, \bar{\mathbf{S}}_u) \leftarrow \mathsf{E}_S(\Pi_{S,1}^*, \Pi_{S,2}^*)$
Return $(\bar{\mathbf{t}}_u, \bar{\mathbf{e}}_u)$.

$\mathcal{O}_{\mathsf{SndToU}}(i, aux)$
If $\mathrm{HU}[i] = \perp$ abort.
Parse $\mathrm{HU}[i] = (upk_i, usk_i)$.
If $aux = \perp$
If $i = u$
$(\bar{\mathbf{c}}, \bar{\mathbf{d}}) \leftarrow \mathcal{O}_{RLWE}$
$\Pi_u \leftarrow \Sigma_I(\bar{\mathbf{c}}, \bar{\mathbf{d}})$
$ots \leftarrow \mathsf{OTSSign}(\mathsf{BitD}(\bar{\mathbf{c}}, \bar{\mathbf{d}}), usk_u)$
Return $\bar{\mathbf{c}}, \bar{\mathbf{d}}, \Pi_u, ots_u, vk_u$.
Else
$\mathbf{c}_i \xleftarrow{\$} \mathcal{R}_q,\ \mathbf{t}_i \xleftarrow{\$} \mathcal{R}_q,\ \mathbf{e} \xleftarrow{\$} \chi$
$\mathbf{d}_i = \mathbf{c}_i \mathbf{t}_i + \mathbf{e}_i \bmod q$
$\Pi_i \leftarrow \mathsf{P}_I(\mathbf{c}_i, \mathbf{d}_i; \mathbf{t}_i, \mathbf{e}_i)$
$ots_i \leftarrow \mathsf{OTSSign}(\mathsf{BitD}(\mathbf{c}_i, \mathbf{d}_i), usk_i)$
Return $\mathbf{c}_i, \mathbf{d}_i, \Pi_i, ots_i, vk_i$.
Else
Parse $aux = \mathbf{S}_i$
$\mathsf{BitD}(\mathbf{c}_i, \mathbf{d}_i, i)$
If $0 \leftarrow \mathsf{SVerify}(\mathbf{S}_i, \mu, \mathbf{A})$ abort.
$\mathrm{l}[i] \leftarrow (\mathbf{c}_i, \mathbf{d}_i, ots_i)$
$gsk_i = (\mathbf{c}_i, \mathbf{d}_i, \mathbf{t}_i, i, \mathbf{S}_i)$
$\mathrm{HU}[i] \leftarrow (upk_i, usk_i)$, $\mathrm{HU}[i] \leftarrow gsk_i$
Return *accept*

$\mathsf{USK}(i)$
If $i = u$ abort.
If $\mathrm{HU}[i] = \perp \ \vee \ \mathrm{HU}[i] = \perp$ abort.
Return usk_i, gsk_i.

$\mathcal{O}_{\mathsf{GSign}}(i, \mu)$
If $\mathrm{HU}[i] = \perp$ abort.
Parse $\mathrm{HU}[i] = gsk_i$.
If $i = u$
For $b = 0, 1$
$(\mathbf{V}_i, \mathbf{W}_i) \leftarrow \mathsf{Enc}(\mathsf{BitD}(\mathbf{c}_i, \mathbf{d}_i, i), (\mathbf{a}_b, \mathbf{b}_b))$
$\Pi_S \leftarrow \Sigma_S(\mu; gpk, opk, (\mathbf{V}_0, \mathbf{W}_0), (\mathbf{V}_1, \mathbf{W}_1))$
Else
For $b = 0, 1$
$(\mathbf{V}_i, \mathbf{W}_i) \leftarrow \mathsf{Enc}(\mathsf{BitD}(\mathbf{c}_i, \mathbf{d}_i, i), (\mathbf{a}_b, \mathbf{b}_b))$
$\Pi_S \leftarrow \mathsf{P}_S(\mu; gpk, opk, (\mathbf{V}_0, \mathbf{W}_0), (\mathbf{V}_1, \mathbf{W}_1), \mathbf{t}_i, i, \mathbf{c}_i, \mathbf{d}_i, \mathbf{e}_i, \mathbf{S}_i)$
$\mathrm{GSig} \leftarrow \mathrm{GSig} \cup \{(i, \mu)\}$
Return $\sigma = (\Pi_S, \mathbf{V}_0, \mathbf{W}_0, \mathbf{V}_1, \mathbf{W}_1)$.

Fig. 4. Simulator that solves RLWE exploiting an adversary against non-frameability.

Proof. B_2 (cf. Figure 5) has access to an oracle $\mathcal{O}_{OTS}$ that, when prompted, outputs a verification key vk, and that allows querying (only) one signature w.r.t. such key pair on a message of B_2's choice. Again, B_2 samples a random user identity u, and simulates all the oracle honestly but USK, $\mathcal{O}_{\mathsf{lss}}$, SndToU, and AddU. If A queries USK for the user u, B_2 aborts. Otherwise, B_2 simulates all other users honestly (according to the definitions in Appendix A). If it runs $\mathcal{O}_{\mathsf{lss}}$ with user identity u (where u has not been assigned yet), B_2 samples another u and then runs the algorithm lss honestly. If A prompts SndToU for user u, B_2 generates the RLWE pair $(\mathbf{c}_u, \mathbf{d}_u)$, and prompts $\mathcal{O}_{OTS}$ to get the key pair (vk, sk) and a signature on $(\mathbf{c}_u, \mathbf{d}_u)$. Then it generates the proof Π_I and sends everything to A. Otherwise, B_2 executes SndToU according to the definition in Appendix A. Finally, if A asks to add a honest user i to AddU, B_2 behaves honestly but in case $i = u$, when it gets the one-time signature from $\mathcal{O}_{OTS}$.

When A outputs the forged signature σ^*, B runs the opening algorithm on it to find the targeted identity and aborts if the target user is not u. Otherwise, according to the definition of the event $F \wedge \bar{P} \wedge S_1 \wedge S_2$, the pair $(\mathbf{c}'_u, \mathbf{d}'_u)$ output by the opening is not equal to the pair $(\mathbf{c}_u, \mathbf{d}_u)$ contained in $\mathrm{l}[u]$. Therefore, the one-time signature ots'_u output by the opener (which is valid w.r.t. , as GJudge outputs 1) is a valid signature w.r.t. the user public key $upk_u = vk$ on a message $(\mathbf{c}'_u, \mathbf{d}'_u)$ that was not queried to the signing oracle (as

$\mathsf{B}_2^{\mathcal{O}_{OTS}}(1^\lambda)$
$(gpk, gsk, opk, osk) \leftarrow \mathsf{GKg}(1^\lambda)$
$\mathrm{CU} \leftarrow \emptyset$, $\mathrm{HU} \leftarrow \emptyset$, $\mathrm{GSig} \leftarrow \emptyset$
$u \xleftarrow{\$} \{1, \ldots, N\}$
$(\mu^*, \sigma^*) \leftarrow \mathsf{A}^{\mathcal{O}_{\mathsf{GSign}},\ \mathsf{AddU},\ \mathsf{USK},\ \mathsf{CrptU},\ \mathsf{RReg},\ \mathsf{WReg},\ \mathsf{SndToU}}(gpk, gsk, opk, osk)$
$(i, \tau) \leftarrow \mathsf{GOpen}(\sigma_1^*, osk)$
Parse $\tau = (\bar{\mathbf{c}}_u, \bar{\mathbf{d}}_u, \mathbf{c}'_u, \mathbf{d}'_u, ots'_u, vk_u, \Pi_O))$
Parse $\mathrm{l}[u] = (\mathbf{c}_u, \mathbf{d}_u, ots_u)$.
If $i \neq u \ \vee \ (\mathbf{c}_u, \mathbf{d}_u) = (\mathbf{c}'_u, \mathbf{d}'_u) \ \vee \ 0 \leftarrow \mathsf{GJudge}(gpk, opk, upk_i, \mu^*, \sigma^*, i, \tau)$ abort.
Return $(\mathbf{c}'_u, \mathbf{d}'_u, ots'_u)$.

$\mathcal{O}_{\mathsf{SndToU}}(i, aux)$
If $\mathrm{HU}[i] = \perp$ abort.
Parse $\mathrm{HU}[i] = (upk_i, usk_i)$.
If $aux = \perp$
$\mathbf{c}_i \xleftarrow{\$} \mathcal{R}_q, \ \mathbf{t}_i \xleftarrow{\$} \mathcal{R}_q, \ \mathbf{e} \xleftarrow{\$} \chi$
$\mathbf{d}_i = \mathbf{c}_i \mathbf{t}_i + \mathbf{e}_i \bmod q$
$\Pi_i \leftarrow \mathsf{P}_I(\mathbf{c}_i, \mathbf{d}_i; \mathbf{t}_i, \mathbf{e}_i)$
If $i = u$
$ots_u \leftarrow \mathcal{O}_{OTS}(\mathsf{BitD}(\mathbf{c}_u, \mathbf{d}_u))$
Else
$ots_i \leftarrow \mathsf{OTSSign}(\mathsf{BitD}(\mathbf{c}_i, \mathbf{d}_i), usk_i)$
Return $\mathbf{c}_i, \mathbf{d}_i, \Pi_i, ots_i, vk_i$.
Else
Parse $aux = \mathbf{S}_i$
$\mathsf{BitD}(\mathbf{c}_i, \mathbf{d}_i, i)$
If $0 \leftarrow \mathsf{SVerify}(\mathbf{S}_i, \mu, \mathbf{A})$ abort.
$\mathrm{l}[i] \leftarrow (\mathbf{c}_i, \mathbf{d}_i, ots_i)$
$gsk_i = (\mathbf{c}_i, \mathbf{d}_i, \mathbf{t}_i, i, \mathbf{S}_i)$
$\mathrm{HU}[i] \leftarrow (upk_i, usk_i)$, $\mathrm{HU}[i] \leftarrow gsk_i$
Return *accept*

$\mathsf{USK}(i)$
If $i = u$ abort.
If $\mathrm{HU}[i] = \perp \ \vee \ \mathrm{HU}[i] = \perp$ abort.
Return usk_i, gsk_i.

$\mathsf{AddU}(i)$
If $i \in \mathrm{HU} \cup \mathrm{CU}$ abort.
If $i = u$
$usk_u \leftarrow \perp$
$upk_i \leftarrow \mathcal{O}_{OTS}$
Else
$(sk_i, vk_i) \leftarrow \mathsf{OTSGen}(1^\lambda)$
$gsk_i \leftarrow \langle \mathsf{Iss}(gsk), \mathsf{SndToU}(i, aux) \rangle$
$\mathrm{HU}[i] \leftarrow (upk_i, usk_i)$
$\mathrm{HU}[i] \leftarrow gsk_i$
Return upk_i.

Fig. 5. Simulator that breaks the unforgeability of the OTS exploiting an adversary against non-frameability.

$\mathcal{O}_{OTS}$ was only queried for a signature on $(\mathbf{c}_u, \mathbf{d}_u)$). Hence, B can output $((\mathbf{c}'_u, \mathbf{d}'_u), ots'_u)$ to win the eu-acma experiment, and it holds $\mathsf{Adv}_{\mathsf{B}}^{OTS}(\lambda) \geq \mathsf{Pr}[F \wedge \bar{P} \wedge S_1 \wedge S_2] \cdot 1/N(\lambda)$ where $N(\lambda)$.

Hence, the advantage of A is

$$\mathsf{Adv}_{\mathsf{A}}^{nf}(\lambda) \leq 2^{-\lambda+1} + 8N(k)(\mathsf{Adv}_{\mathsf{B}}^{RLWE}(\lambda)(1 - \nu(\lambda)) + N(\lambda)\mathsf{Adv}_{\mathsf{B}'}^{OTS}(\lambda)) \ .$$

□

5.5 Parameters and Storage Requirements

We compute parameters for $\lambda \geq 128$ bits of security in the "paranoid" framework of Alkim et al. [1], that in particular requires $\delta \leq 1.00255$. We intend "security" here as the claim that the underlying hardness assumptions are hard to solve for a quantum computer. We choose as ring the polynomial ring $\mathcal{R}_q$ defined by $n = 2^{10}$ and a prime $2^{64} < q < 2^{65}$. Such choice of degree guarantees that the set $\mathcal{S}_1$ contains more than 2^{256} elements, hence finding the user's secret $\mathbf{t}_i$ through a brute-force attack is not possible. The number N of supported users is 2^{26}. For technical reasons, Aurora requires that $\mathbb{F}_q$ has a large power-of-2 multiplicative subgroup, and so we choose q accordingly (most choices of q satisfy this requirement). This implies that the unforgeability of the signature scheme is based on a $\mathrm{RSIS}_{d,\beta}$ instance where $d = 68$ and $\beta \leq 2^{46}$, and on a

$\mathrm{RLWE}_{l,\chi}$ instance with $l < 2^{25}$. To estimate their hardness, we use the root Hermite factor δ (cf. [39]), and we obtained a $\delta_{RSIS} \leq 1.00062$ and $\delta_{RLWE} \leq 1.00001$.

We now compute the length of the keys and of a signature output by the group signature. An element in $\mathcal{R}_q$ can be stored in $nk \leq 8.32$ KB. The opener's secret key is composed by one ring element, hence it can be stored in 8.32 KB, while the opener's public key in 33.28 KB (as it is composed by 4 ring elements).

The group manager's public key requires a bit of care. Indeed, the key $(\mathbf{A}, \mathbf{B}, \mathbf{A}_0, \ldots, \mathbf{A}_\ell, \mathbf{u})$ includes $\mathbf{A} = [\mathbf{a}, \mathbf{1}]$, $\mathbf{B} \in \mathcal{R}_q^{1 \times \bar{m}}$ that are generated with the trapdoor (cf. Section 2.3), ℓ random vectors with $\bar{m} = 67$ components in $\mathcal{R}_q$, where $\ell = 2nk + \lceil \log N \rceil$, and a random element $\mathbf{u} \in \mathcal{R}_q$. Storing these would require $nk \cdot (1 + \bar{m} + \bar{m} \cdot \ell + 1) = 2^{10} \cdot 65 \cdot (1 + 67 + 67 \cdot 2^{18} + 1) = 146$ GB, and it is clearly infeasible. Instead, the issuer can send a condensed (pseudorandom) representation of the random elements $\mathbf{A}_0, \ldots, \mathbf{A}_\ell, \mathbf{u}$, having considerably smaller size. The size of the public key then becomes the size of such a representation plus $(\bar{m} + 1)nk \leq 0.57$ MB.

The group manager's secret key is the trapdoor $\mathbf{T}_\mathbf{A}$, that has components with coefficients smaller than $8s_{ssk} = 8\sqrt{\log(n^2)} + 1$ (cf Lemma 2.1 and Theorem 2.5). Hence the size of $\mathbf{T}_\mathbf{A}$ is $2kn \log(8s_{ssk}) \leq 91$ KB.

At the end of the joining phase the user obtains the credential $(\mathbf{c}_i, \mathbf{d}_i, \mathbf{t}_i, i, \mathbf{S}_i)$, where the vector $\mathbf{S}_i$ is composed by $2\bar{m} + 2$ ring elements with coefficients smaller than $8s_\sigma = 8\sqrt{n \log n \log n^2}$ (cf. Section 2.3). Hence it has size $3nk + \lceil \log N \rceil + (2\bar{m} + 2)n \log(8\sqrt{n \log n \cdot \log n^2}) \leq 231$ KB. The secret signing key of the OTS can be discarded after the joining phase.

Finally, a signature is composed by the NIZK proof Π_S, and 4 vectors of elements in the ring. The proof length is around 250 KB (estimate from [7]). The vectors $\mathbf{V}_0, \mathbf{W}_0, \mathbf{V}_1, \mathbf{W}_1$ are the encryptions of two ring elements $(\mathbf{c}_i, \mathbf{d}_i)$ and a number $i < N$. As the encryption algorithm converts them into polynomials in $\mathcal{S}_1$ whose coefficients are the bits of their binary expansions, each vector is composed by $\lceil (2nk + \lceil \log N \rceil)/n \rceil = 2k + \lceil \lceil \log N \rceil / n \rceil$ elements in $\mathcal{R}_q$, hence $(\mathbf{V}_0, \mathbf{W}_0, \mathbf{V}_1, \mathbf{W}_1)$ has size $(2k + \lceil \lceil \log N \rceil / n \rceil) \cdot nk \leq (2 \cdot 65 + \lceil 26 \cdot 2^{-10} \rceil) \cdot 2^{10} \cdot 65 = 131 \cdot 1024 \cdot 65 = 1.09$ MB. Hence a signature is roughly 1.34 MB long.

To compare our scheme with previous ones (such as del Pino et al. [22] or Boschini et al. [12]), we compute the length of the signature for the case in which the group manager is assumed to be honest (by default our scheme guarantees a stronger notion of security). Modifying our signature to have a honest group manager essentially means that it is enough that during issuance the user gets a signature by the group manager on the user identity i. Hence, opening only requires the signature to contain an encryption of the user's identity i, whose bit decomposition can be encoded as one element of $\mathcal{S}_1$. Therefore, the vectors $\mathbf{V}_0, \mathbf{W}_0, \mathbf{V}_1, \mathbf{W}_1$ actually are just ring elements, hence the size of the signature is at most $250 + 4 \cdot 2^{10} \cdot 65 \leq 300$ KB (obviously, the size of the proof should shrink too, as the number of variables is smaller, but we mean this number as a rough upper bound).

References

1. Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. Post-quantum key exchange - A new hope. In *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016.*, pages 327–343, 2016.
2. Wojciech Banaszczyk. New bounds in some transference theorems in the geometry of numbers. *Mathematische Annalen*, 296(1):625–635, 1993.
3. Carsten Baum, Jonathan Bootle, Andrea Cerulli, Rafaël del Pino, Jens Groth, and Vadim Lyubashevsky. Sub-linear lattice-based zero-knowledge arguments for arithmetic circuits. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 669–699. Springer, Heidelberg, August 2018.
4. Mihir Bellare, Daniele Micciancio, and Bogdan Warinschi. Foundations of group signatures: Formal definitions, simplified requirements, and a construction based on general assumptions. In Eli Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 614–629. Springer, Heidelberg, May 2003.
5. Mihir Bellare and Gregory Neven. Multi-signatures in the plain public-key model and a general forking lemma. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *ACM CCS 06*, pages 390–399. ACM Press, October / November 2006.

6. Mihir Bellare, Haixia Shi, and Chong Zhang. Foundations of group signatures: The case of dynamic groups. In Alfred Menezes, editor, *CT-RSA 2005*, volume 3376 of *LNCS*, pages 136–153. Springer, Heidelberg, February 2005.
7. Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward. Aurora: Transparent succinct arguments for R1CS. *IACR Cryptology ePrint Archive*, 2018:828, 2018.
8. Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. Interactive oracle proofs. *IACR Cryptology ePrint Archive*, 2016:116, 2016.
9. Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. Interactive oracle proofs. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B, Part II*, volume 9986 of *LNCS*, pages 31–60. Springer, Heidelberg, October / November 2016.
10. David Bernhard, Véronique Cortier, David Galindo, Olivier Pereira, and Bogdan Warinschi. SoK: A comprehensive analysis of game-based ballot privacy definitions. In *2015 IEEE Symposium on Security and Privacy*, pages 499–516. IEEE Computer Society Press, May 2015.
11. Cecilia Boschini, Jan Camenisch, and Gregory Neven. Relaxed lattice-based signatures with short zero-knowledge proofs. Cryptology ePrint Archive, Report 2017/1123, 2017. `https://eprint.iacr.org/2017/1123`.
12. Cecilia Boschini, Jan Camenisch, and Gregory Neven. Floppy-sized group signatures from lattices. In Bart Preneel and Frederik Vercauteren, editors, *ACNS 18*, volume 10892 of *LNCS*, pages 163–182. Springer, Heidelberg, July 2018.
13. Cecilia Boschini, Jan Camenisch, and Gregory Neven. Relaxed lattice-based signatures with short zero-knowledge proofs. In Liqun Chen, Mark Manulis, and Steve Schneider, editors, *ISC 2018*, volume 11060 of *LNCS*, pages 3–22. Springer, Heidelberg, September 2018.
14. Cecilia Boschini, Jan Camenisch, Max Ovsiankin, and Nicholas Spooner. Efficient post-quantum snarks for RSIS and RLWE and their applications to privacy. In Jintai Ding and Jean-Pierre Tillich, editors, *Post-Quantum Cryptography - 11th International Conference, PQCrypto 2020, Paris, France, April 15-17, 2020, Proceedings*, volume 12100 of *Lecture Notes in Computer Science*, pages 247–267. Springer, 2020.
15. Xavier Boyen. Lattice mixing and vanishing trapdoors: A framework for fully secure short signatures and more. In Phong Q. Nguyen and David Pointcheval, editors, *PKC 2010*, volume 6056 of *LNCS*, pages 499–517. Springer, Heidelberg, May 2010.
16. Zvika Brakerski and Vinod Vaikuntanathan. Fully homomorphic encryption from ring-LWE and security for key dependent messages. In Phillip Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 505–524. Springer, Heidelberg, August 2011.
17. Jan Camenisch and Anna Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In Birgit Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 93–118. Springer, Heidelberg, May 2001.
18. David Chaum. Security without identification: Transaction systems to make big brother obsolete. *Commun. ACM*, 28(10):1030–1044, 1985.
19. David Chaum and Eugène van Heyst. Group signatures. In Donald W. Davies, editor, *EUROCRYPT'91*, volume 547 of *LNCS*, pages 257–265. Springer, Heidelberg, April 1991.
20. Alessandro Chiesa, Peter Manohar, and Nicholas Spooner. Succinct arguments in the quantum random oracle model. In *Theory of Cryptography - 17th International Conference, TCC 2019, Nuremberg, Germany, December 1-5, 2019, Proceedings, Part II*, pages 1–29, 2019.
21. James W Cooley and John W Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of computation*, 19(90):297–301, 1965.
22. Rafaël del Pino, Vadim Lyubashevsky, and Gregor Seiler. Lattice-based group signatures and zero-knowledge proofs of automorphism stability. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 18*, pages 574–591. ACM Press, October 2018.
23. Nicholas Genise and Daniele Micciancio. Faster gaussian sampling for trapdoor lattices with arbitrary modulus. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part I*, volume 10820 of *LNCS*, pages 174–203. Springer, Heidelberg, April / May 2018.
24. Rosario Gennaro, Michele Minelli, Anca Nitulescu, and Michele Orrù. Lattice-based zk-SNARKs from square span programs. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 18*, pages 556–573. ACM Press, October 2018.
25. Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208, 1989.
26. Shuichi Katsumata and Shota Yamada. Group signatures without NIZK: from lattices in the standard model. In *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and*

Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part III, pages 312–344, 2019.
27. Serge Lang. *Algebra*, volume 211 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, 2002.
28. Adeline Langlois and Damien Stehlé. Worst-case to average-case reductions for module lattices. *Des. Codes Cryptography*, 75(3):565–599, 2015.
29. Benoît Libert, San Ling, Khoa Nguyen, and Huaxiong Wang. Lattice-based zero-knowledge arguments for integer relations. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 700–732. Springer, Heidelberg, August 2018.
30. San Ling, Khoa Nguyen, Damien Stehlé, and Huaxiong Wang. Improved zero-knowledge proofs of knowledge for the ISIS problem, and applications. In Kaoru Kurosawa and Goichiro Hanaoka, editors, *PKC 2013*, volume 7778 of *LNCS*, pages 107–124. Springer, Heidelberg, February / March 2013.
31. Vadim Lyubashevsky. Lattice signatures without trapdoors. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 738–755. Springer, Heidelberg, April 2012.
32. Vadim Lyubashevsky and Daniele Micciancio. Generalized compact Knapsacks are collision resistant. In Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener, editors, *ICALP 2006, Part II*, volume 4052 of *LNCS*, pages 144–155. Springer, Heidelberg, July 2006.
33. Vadim Lyubashevsky, Daniele Micciancio, Chris Peikert, and Alon Rosen. SWIFFT: A modest proposal for FFT hashing. In Kaisa Nyberg, editor, *FSE 2008*, volume 5086 of *LNCS*, pages 54–72. Springer, Heidelberg, February 2008.
34. Vadim Lyubashevsky and Gregory Neven. One-shot verifiable encryption from lattices. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part I*, volume 10210 of *LNCS*, pages 293–323. Springer, Heidelberg, April / May 2017.
35. Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 1–23. Springer, Heidelberg, May / June 2010.
36. Vadim Lyubashevsky, Chris Peikert, and Oded Regev. A toolkit for ring-LWE cryptography. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 35–54. Springer, Heidelberg, May 2013.
37. Daniele Micciancio. Generalized compact knapsacks, cyclic lattices, and efficient one-way functions from worst-case complexity assumptions. In *43rd FOCS*, pages 356–365. IEEE Computer Society Press, November 2002.
38. Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 700–718. Springer, Heidelberg, April 2012.
39. Daniele Micciancio and Oded Regev. *Lattice-based Cryptography*, pages 147–191. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.
40. Moni Naor and Moti Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In *22nd ACM STOC*, pages 427–437. ACM Press, May 1990.
41. Chris Peikert and Alon Rosen. Efficient collision-resistant hashing from worst-case assumptions on cyclic lattices. In Shai Halevi and Tal Rabin, editors, *TCC 2006*, volume 3876 of *LNCS*, pages 145–166. Springer, Heidelberg, March 2006.
42. Amit Sahai. Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In *40th FOCS*, pages 543–553. IEEE Computer Society Press, October 1999.
43. Adi Shamir and Yael Tauman. Improved online/offline signature schemes. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 355–367. Springer, Heidelberg, August 2001.

A Formal Definition of Group Signature

In this section we present and discuss the Bellare-Shi-Zhang (BSZ) model for such group signature [6]. All the results and observations presented in the section are taken from their work, (unless otherwise stated).

A (dynamic) group signature is a set of algorithms ($\mathsf{GKg}, \mathsf{UKg}, \mathsf{Join}, \mathsf{Iss}, \mathsf{GSign}, \mathsf{GVerify}, \mathsf{GOpen}, \mathsf{GJudge}$) between a group manager, an opener and users. It is called *dynamic* because users can join at any time during the lifespan of the group (while in static group signatures where joining can only happen during the setup phase at the beginning).

To exclude man-in-the-middle attacks, we assume the existence of a PKI that allows to obtain certified copies of the public key of the entities. To include this in our model, we assume there exists a public list **upk** whose i-th entry contains the public key of user i. The position i is the user identity. Finally, the group manager keeps another list called l, whose i-th entry contains the public key of the i-th user and her signing key.

Key Generation. A trusted third party generates the group manager's and opener's keys running $(gpk, gsk, opk, osk) \leftarrow \mathsf{GKg}(1^\ell)$ (where ℓ is the security parameter). For the sake of clarity, we distinguish the group public key gpk from the opener's public key opk. A user generates her user secret key as $(usk, upk) \leftarrow \mathsf{UKg}(gpk)$.

Joining Phase. The joining phase is an interactive protocol between a user i (running algorithm Join) and the group manager (running Iss). Each takes as input an incoming message (ϵ if it is the first step of the interaction) and the current state, and outputs an outgoing message, an updated state, and a decision ($accept$, $reject$, $continue$). At the end of the interaction, the user obtains a signing key gsk_i that includes her keys usk, upk and her identity i as group member. If at the end of the interaction the user accepts, the group manager creates a new entry $\mathrm{l}[i]$ in the list l containing the identity and signing key of user i. The list is needed to allow the opener to prove that the identity he has extract from the signature corresponds to an existing group member.

Signing. A user can sign a message μ on behalf of the group using her signing key with the algorithm $\mathsf{GSign}(gsk_i, gpk, opk, \mu)$.

Verification. A signature σ on a message μ can be verified with the algorithm $\{1, 0\} \leftarrow \mathsf{GVerify}(\mu, \sigma, gpk, opk)$.

Opening. The opener can recover the identity i' of the group member that signed a message μ running $(i', \tau) \leftarrow \mathsf{GOpen}(\mu, \sigma, gpk, osk, \mathrm{l})$. The algorithm outputs the identity of the signer with a proof that the opening procedure was performed honestly (contained in τ). The list is needed to prove that the identity i' that the opener outputs corresponds to an existing group member. If opening fails, it outputs $(i', \tau) = (0, \epsilon)$

Judge. A third party can check whether the opener honestly opened a signature by running the algorithm $0, 1 \leftarrow \mathsf{GJudge}(gpk, opk, \mu, \sigma, \tau, i', \mathrm{l})$. Differently from Bellare et al., we allow the judge to have access to the list instead of being given the public key $gpk_{i'}$. We think this choice is a better interpretation of the role of safeguard that this algorithm has. Otherwise, the correctness of the output of GJudge depends on it being given the right public key. This implies that the algorithm GJudge cannot be invoked by random users, as the list l is not public (nor accessible to users).

The security requirements of the group signature are correctness, anonymity, traceability, and non frameability.

The formal definitions require the definition of some lists. In the following, we clarify what are the lists for the BSZ model and how are they used. To keep track of honest users, a list HU is created. Such list contains as i-th entry the public and secret key of the user with identity i. Hence, by "adding i to the list HU" we mean we set the i-th entry of the list to be (upk_i, usk_i). The list CU contains all the users that were corrupted *before* joining the group (i.e., corrupted by A through CrptU). The adversary is allowed to corrupt users after they become group members (through USK), but such users are not into this list. In fact, such list contains the users that might have colluded with the group manager (in case M is corrupted) to tamper with the list l (impacting honest opening and J). To detect when the adversary forges a signature by simply

	Group Manager	User	Opener	Judge
Correctness		✓(the adversary can add users)		
Anonymity	✓	✓(all *usk* can be leaked)		
Traceability		✓(all *usk* can be leaked)	✓ (partially)	
Non frameability	✓	✓(there should exist at least one honest user)	✓	

Table 2. Corrupted entities depending on the security property.

querying a user's signing key, the non-frameability experiment checks A's queries to USK before accepting a forgery. Finally, we need a list of the challenge message-signature pairs CH generated during the anonymity game, and a list GSig of the signatures output by the signing oracle.

$\mathcal{O}_{\mathsf{Iss}}$**:** this oracle (denoted SndToI in [6]) performs honestly the issuer's side of the joining phase. During the interaction, the adversary impersonates a user i willing to join the group. The oracle first checks that the user i is a corrupted user; if $i \notin$ CU the algorithm aborts. At the end of the interaction it updates the list of users l with the new user credential.

$\mathcal{O}_{\mathsf{GOpen}}$**:** on input a pair (μ, σ), if $(\mu, \sigma) \notin$ CH this oracle (denoted Open in [6]) outputs the honest opening of the signature, otherwise it outputs $\perp$.

$\mathcal{O}_{\mathsf{GSign}}$**:** on input i and a message μ, it aborts if $i \notin$ HU. Otherwise, it recovers the corresponding signing key gsk_i (it aborts if such key does not exist), then it outputs a signature $\sigma \leftarrow \mathsf{GSign}(gsk_i, gpk, opk, \mu)$. It stores (i, μ) in a list GSig.

Chall_b**:** on input a message μ and two identities i_0, i_1, it recovers gsk_{i_b} and outputs a signature σ on μ using the signing key of user i_b if $i_0, i_1 \in$ HU (otherwise it aborts). The algorithm adds the entry (μ, σ) to CH.

RReg**:** on input i, outputs the i-th entry of the list l[i] to the adversary.

WReg**:** on input i and a string B (that is a valid list entry), sets l[i] to B.

SndToU**:** this algorithm allows the adversary to choose a user i and to run with i the joining protocol impersonating the issuer. The result of this interaction is that there is a new honest group member, hence the algorithm adds i to HU.

CrptU**:** the adversary uses this algorithm whenever she wants to corrupt a user before it has joined the group. On input i, upk', it first checks whether i is in HU or CU. If $i \in \text{CU} \cup \text{HU}$, it returns ϵ. Then it looks up the user with keys (upk_i, usk_i), and sets her key to be (upk', usk_i). Finally, if i is in HU, it removes it and adds (upk', usk_i) to CU it is in CU, it updates it to be (upk', usk_i).

USK**:** on input the user's public key upk_i, outputs the corresponding secret key usk_i and secret signing key gsk_i.

AddU**:** on input a user identity i, it creates a new honest user with identity i. If $i \in \text{CU} \cup \text{HU}$ it aborts. Otherwise, it generates usk_i, upk_i, adds i to HU, and executes honestly the group joining protocol by running Join and Iss on behalf of user i and of the issuer respectively, where both algorithms are initialized with the necessary keys. If the protocol ends successfully, it returns upk_i.

We slightly modified the BSZ model to clarify the definitions. Mainly, we introduced a list GSig, to make it easy to verify whether a message was signed with a particular *usk* by the honest signing algorithm $\mathcal{O}_{\mathsf{GSign}}$, and we do not assume that access to WReg implies that A is able to read the list too. The result is that, whenever the adversary can corrupt the issuer, she is given access to both RReg and WReg.

To understand why we need these oracles, we analyze the different degrees of corruption of the entities depending on the security property we want to guarantee. We briefly explain this in the following; our observations are summarized in Table A. Note that the Judge is always honest by construction.

The algorithms $\mathcal{O}_{\mathsf{Iss}}$, $\mathcal{O}_{\mathsf{GOpen}}$ and $\mathcal{O}_{\mathsf{GSign}}$ model the honest behavior of the issuer, the opener and a user respectively. The other algorithms model the various hostile scenarios. There are different degrees of corruption, but the distinction Bellare et al. are concerned with is between *partial* and *full* corruption. In both cases the secret key is leaked to the adversary. However, a partially corrupt entity still performs its task honestly, while a fully corrupt one will give the adversary full control, allowing deviations from the protocols. Note that this distinction is not relevant in the case of the issuer. In fact, knowing the issuer secret key is not useful for the adversary if she cannot interact with users. Hence in the definitions a *corrupted* issuer is always assumed to be *fully* corrupted (i.e., when the adversary gets *gsk* she is also given access to the oracle

Experiment $\mathsf{Exp}^{corr}_{\mathsf{A}}(1^\lambda)$
$\mathrm{CU} \leftarrow \emptyset$, $\mathrm{HU} \leftarrow \emptyset$
$(gpk, gsk, opk, osk) \leftarrow \mathsf{GKg}(1^\lambda)$
$(i, \mu) \leftarrow \mathsf{A}^{\mathsf{AddU},\ \mathsf{RReg}}(gpk, opk)$
If $(i \notin \mathrm{HU} \ \vee \ gsk_i = \epsilon)$ return 0.
$\sigma \leftarrow \mathsf{GSign}(gsk_i, gpk, opk, \mu)$
If $0 \leftarrow \mathsf{GVerify}(\mu, \sigma, gpk, opk)$ return 1.
$(i', \tau) \leftarrow \mathsf{GOpen}(\mu, \sigma, gpk, osk)$
If $(i' \neq i \ \vee \ 0 \leftarrow \mathsf{GJudge}(gpk, opk, \mu, \sigma, \tau, i', \mathrm{l}))$ return 1, else return 0.

Fig. 6. Correctness experiment for dynamic group signature.

SndToU). The case of the opener is different. In fact, in this case knowing the secret opening key is useful to the adversary, as it is enough to de-anonymize signatures produced by group members.

Finally, regarding users, the adversary has multiple options. She can either create users (querying $\mathcal{O}_{\mathsf{Iss}}$ or exploiting knowledge of gsk), ask users to sign messages of her choice (interacting with $\mathcal{O}_{\mathsf{GSign}}$), or corrupt a user to either change her public key (CrptU), or reveal her secret key (USK).

The algorithm Chall_b is used in the anonymity experiment to model the generation of the challenge signature. In this way, it is not necessary to distinguish in the experiment the adversary's capabilities before and after receiving the challenge.

Correctness requires that honestly generated signatures satisfy the verification checks, can be opened to the correct identity of the signer, and that the proof generated by a honest opener is always accepted by the GJudge algorithm.

Definition A.1 (Correctness). *A group signature* (GKg, UKg, Join, Iss, GSign, GVerify, GOpen, GJudge) *satisfies* correctness *if every adversary* A *has no advantage in winning the experiment* $\mathsf{Exp}^{corr}_{\mathsf{A}}(1^\lambda)$ *in Figure A, i.e.,*

$$\mathsf{Adv}^{corr}_{\mathsf{A}}(\lambda) := \mathsf{Pr}(1 \leftarrow \mathsf{Exp}^{corr}_{\mathsf{A}}(1^\lambda)) = 0 \ .$$

Anonymity informally requires the adversary not to be able to distinguish which user has signed a message of her choice. For the property to be meaningful, the adversary should not be allowed to corrupt the opener. On the other hand, the adversary is allowed to *fully corrupt* (i.e., to know the secret key of) both the users and the issuer. In particular, the adversary should not be able to recognize the signatures generated by a user even if she recovers the secret key of the user.

Definition A.2 (Anonymity). *A group signature satisfies anonymity if*

$$\mathsf{Adv}^{anon}_{\mathsf{A}}(\lambda) := \mathsf{Pr}(1 \leftarrow \mathsf{Exp}^{an}_{\mathsf{A},1}(1^\lambda)) - \mathsf{Pr}(1 \leftarrow \mathsf{Exp}^{an}_{\mathsf{A},0}(1^\lambda))$$

is negligible in the security parameter for all PPT adversaries A*, where the experiment can be found in Figure A.*

Coherently with Bellare et al., in $\mathsf{Exp}^{an}_{\mathsf{A}}$ the adversary is not given access to a signing oracle, as it would be redundant.

Experiment $\mathsf{Exp}^{an}_{\mathsf{A},b}(1^\lambda)$
$(gpk, gsk, opk, osk) \leftarrow \mathsf{GKg}(1^\lambda)$
$\mathrm{CU} \leftarrow \emptyset$, $\mathrm{HU} \leftarrow \emptyset$, $\mathrm{CH} \leftarrow \emptyset$
$b' \leftarrow \mathsf{A}^{\mathcal{O}_{\mathsf{GOpen}},\ \mathsf{CrptU},\ \mathsf{USK},\ \mathsf{RReg},\ \mathsf{WReg},\ \mathsf{SndToU},\ \mathsf{Chall}_b}(gpk, gsk, opk)$
Return b'.

Fig. 7. Anonymity experiment for dynamic group signature.

Experiment $\mathsf{Exp}^{tr}_{\mathsf{A}}(1^\lambda)$
$(gpk, gsk, opk, osk) \leftarrow \mathsf{GKg}(1^\lambda)$
$\mathrm{CU} \leftarrow \emptyset$, $\mathrm{HU} \leftarrow \emptyset$, $\mathrm{GSig} \leftarrow \emptyset$
$(\mu^*, \sigma^*) \leftarrow \mathsf{A}^{\mathcal{O}_{\mathsf{Iss}},\ \mathsf{AddU},\ \mathsf{USK},\ \mathsf{CrptU},\ \mathsf{RReg}}(gpk, opk, osk)$
If $0 \leftarrow \mathsf{GVerify}(\mu^*, \sigma^*, gpk, opk)$, return 0
Else $(i^*, \tau^*) \leftarrow \mathsf{GOpen}(\mu^*, \sigma^*, gpk, osk)$.
If $(i^* = \epsilon) \ \vee \ 0 \leftarrow \mathsf{GJudge}(gpk, opk, \mu^*, \sigma^*, \tau^*, i^*, \mathrm{l})$ return 1.
Else return 0.

Fig. 8. Traceability experiment for dynamic group signature.

The notion of traceability for group signature essentially means that the opener can always link a signature back to the signer (cf. [4]). When the group signature has blinded joining protocol, such notion is split in two, as both the issuer and the opener can be corrupted. First, an adversary A should not be able to produce a signature that cannot be opened by an honest opener, or such that a honest opener is unable to prove that the opening was correctly executed, even if A corrupts the opener (*traceability*). If A could corrupt the issuer this would be trivial to achieve, as the issuer could issue a signing key gsk_i to a user i, without adding the user keys to the list l. In this way, A can produce valid signatures using gsk_i, but the opener cannot recover the user's public key, as $\mathrm{l}[i]$ does not exist. Analogously, if A could fully corrupt the opener the definition would be meaningless, as the adversary would win it always by simply forcing the opener to declare itself unable to open the forged signature A outputs.

Definition A.3 (Traceability). *A group signature satisfies traceability if all PPT adversaries* A *have negligible advantage in the experiment* $\mathsf{Exp}^{tr}_{\mathsf{A}}$*, i.e.,*

$$\mathsf{Adv}^{trac}_{\mathsf{A}}(\lambda) := \mathsf{Pr}(1 \leftarrow \mathsf{Exp}^{tr}_{\mathsf{A}}) = \nu(\lambda) \ .$$

Remark that again there is no need to give to the adversary explicit access to an oracle that produces signatures of honest users, as the adversary could just query the user's secret keys through USK.

On the other hand, it should be impossible for A to generated a signature that links back to a honest user who did not produce it, even if she corrupted both the issuer and the opener (*non-frameability*).

Definition A.4 (Non-frameability). *A group signature satisfies non-framea- bility if all PPT adversaries* A *have negligible advantage in the experiment* $\mathsf{Exp}^{nf}_{\mathsf{A}}$ *in Figure A, i.e.,*

$$\mathsf{Adv}^{non-fr}_{\mathsf{A}}(\lambda) := \mathsf{Pr}(1 \leftarrow \mathsf{Exp}^{tr}_{\mathsf{A}}) = \nu(\lambda) \ .$$

Experiment $\mathsf{Exp}^{nf}_{\mathsf{A}}(1^\lambda)$
$(gpk, gsk, opk, osk) \leftarrow \mathsf{GKg}(1^\lambda)$
$\mathrm{CU} \leftarrow \emptyset$, $\mathrm{HU} \leftarrow \emptyset$, $\mathrm{GSig} \leftarrow \emptyset$
$(\mu^*, \sigma^*) \leftarrow \mathsf{A}^{\mathcal{O}_{\mathsf{GSign}},\ \mathsf{AddU},\ \mathsf{USK},\ \mathsf{CrptU},\ \mathsf{RReg},\ \mathsf{WReg},\ \mathsf{SndToU}}(gpk, gsk, opk, osk)$
If $0 \leftarrow \mathsf{GVerify}(\mu^*, \sigma^*, gpk, opk)$, return 0
Else $(i^*, \tau^*) \leftarrow \mathsf{GOpen}(\mu^*, \sigma^*, gpk, osk)$.
If $i^* \in \mathrm{HU} \ \wedge \ 1 \leftarrow \mathsf{GJudge}(gpk, opk, \mu^*, \sigma^*, \tau^*, i^*, \mathrm{l}) \ \wedge \ (i^*, \mu^*) \notin \mathrm{GSig}$
return 1.
Else return 0.

Fig. 9. Non-frameability experiment for dynamic group signature.