Two-round n-out-of-n and Multi-Signatures and Trapdoor Commitment from Lattices[⋆]

Ivan Damgård[1], Claudio Orlandi[1], Akira Takahashi[1], and Mehdi Tibouchi[2]

[1] Department of Computer Science and DIGIT, Aarhus University, Denmark
{ivan,orlandi,takahashi}@cs.au.dk
[2] NTT Corporation, Japan
mehdi.tibouchi.br@hco.ntt.co.jp

November 16, 2020

Abstract. Although they have been studied for a long time, distributed signature protocols have garnered renewed interest in recent years in view of novel applications to topics like blockchains. Most recent works have focused on distributed versions of ECDSA and over variants of Schnorr signatures, however, and in particular, little attention has been given to constructions based on post-quantum secure assumptions like the hardness of lattice problems. A few lattice-based threshold signature and multi-signature schemes have been proposed in the literature, but they either rely on hash-and-sign lattice signatures (which tend to be comparatively inefficient), use expensive generic transformations, or only come with incomplete security proofs.

In this paper, we construct several lattice-based distributed signing protocols with low round complexity following the Fiat–Shamir with Aborts (FSwA) paradigm of Lyubashevsky (Asiacrypt 2009). Our protocols can be seen as distributed variants of the fast Dilithium-G signature scheme and the full security proof can be made assuming the hardness of module SIS and LWE problems. A key step to achieve security (unexplained in some earlier papers) is to prevent the leakage that can occur when parties abort after their first message—which can inevitably happen in the Fiat–Shamir with Aborts setting. We manage to do so using homomorphic commitments.

Exploiting the similarities between FSwA and Schnorr-style signatures, our approach makes the most of observations from recent advancements in the discrete log setting, such as Drijvers et al.'s seminal work on two-round multi-signatures (S&P 2019). In particular, we observe that the use of commitment not only resolves the subtle issue with aborts, but also makes it possible to realize secure *two-round n-out-of-n distributed signing* and *multi-signature in the plain public key model*, by equipping the commitment with a trapdoor feature. The construction of suitable trapdoor commitment from lattices is a side contribution of this paper.

Keywords: threshold signatures, n-out-of-n distributed signatures, multi-signatures, lattice-based cryptography, Fiat–Shamir with aborts, trapdoor commitments.

⋆ This research was supported by: the Concordium Blockchain Research Center, Aarhus University, Denmark; the Carlsberg Foundation under the Semper Ardens Research Project CF18-112 (BCM); the European Research Council (ERC) under the European Unions's Horizon 2020 research and innovation programme under grant agreement No 803096 (SPEC); the Danish Independent Research Council under Grant-ID DFF-6108-00169 (FoCC).

Table of Contents

1 Introduction

In recent years, distributed signing protocols have been actively researched, motivated by many new applications for instance in the blockchain domain. One of the main motivations to construct a distributed signature is reducing the risk of compromising the secret key, which could occur in various ways, for instance as a result of attacks on cryptographic devices. In this paper, we study two similar classes of distributed signing protocols that can be constructed from standard lattice-based computational hardness assumptions, namely n-out-of-n distributed signature schemes and multi-signature schemes.

n-out-of-n signature. An n-out-of-n signature is a special case of general t-out-of-n threshold signature. At a high level, the parties involved in n-out-of-n signature first invoke a key generation protocol in a way that each individual party P_j learns a share sk_j of the single signing key sk, but sk is unknown to anyone, and then they interact with each other to sign the message of interest. The required security property can be informally stated as follows: If all n parties agree to sign the message then they always produce a single signature that can be verified with a single public key pk; if at most $n-1$ parties are corrupted, it is not possible for them to generate a valid signature. Several recent works have studied threshold versions of ECDSA, arguably the most widely deployed signature scheme, both in the 2-out-of-2 variant [Lin17, DKLs18, CCL+19] and in the more general t-out-of-n case [GGN16, GG18, LN18, DKLs19, DKO+19, CCL+20, CMP20, GKSS20, DJN+20, GG20].

However it is well known that ECDSA does not withstand quantum attacks since it is based on discrete log, and it is therefore important to study post-quantum alternatives which support threshold signing. Despite this, very few works have considered n-out-of-n (or t-out-of-n) lattice-based signatures. Bendlin et al. [BKP13] proposed a threshold protocol to generate Gentry–Peikert–Vaikuntanathan signature [GPV08]; Boneh et al. [BGG+18] developed a universal thresholdizer that turns any signature scheme into a non-interactive threshold one, at the cost of using relatively heavy threshold fully homomorphic encryption.

Fiat–Shamir with aborts. Neither of the above previous papers investigated signatures following the *Fiat–Shamir with Aborts (FSwA)* paradigm due to Lyubashevsky [Lyu09, Lyu12], which was specifically designed for lattice-based signatures and is nowadays one of the most efficient and popular approaches to constructing such schemes. Recall that in standard Fiat-Shamir signatures, the scheme is based on an underlying 3-move Σ-protocol where transcripts are of form (w, c, z) and where c is a random challenge. This interactive protocol is then turned into a non-interactive signature scheme by choosing the challenge as the hash of the first message w and the message to be signed. The FSwA paradigm follows the same approach, with the important difference that the signer (prover) is allowed to abort the protocol after seeing the challenge. Only the non-aborting instances are used for signatures, and it turns out that this allows the signer to reduce the size of the randomness used and hence reduces signature size. This comes at the cost of marginally larger signing time because some (usually quite small) fraction of the protocol executions are lost due to aborts.

Examples of single-user schemes based on FSwA include Dilithium [LDK+19] and qTESLA [BAA+19], which are round-3 and round-2 candidates of the NIST Post-Quantum Cryptography Standardization process. Cozzo and Smart [CS19] recently estimated the concrete communication and computational costs required to build a distributed version of these schemes by computing Dilithium and qTESLA with generic multi-party computation. They pointed out inherent performance issue with MPC due to the mixture of both linear and non-linear operations within the FSwA framework.

Given all this, an obvious open question is to construct secure n-out-of-n protocols specifically tailored to the FSwA, while also achieving small round complexity.

Multi-signature. A multi-signature protocol somewhat resembles n-out-of-n signature and allows a group of n parties holding a signing key $sk_1, \ldots, sk_n$ to collaboratively sign the same message to obtain a single signature. However, multi-signature protocols differ from n-out-of-n signing in the following ways: (1) there is no dedicated key generation protocol, and instead each party P_j locally generates its own key pair (pk_j, sk_j) and publish pk_j before signing (so-called the *plain public-key model* [BN06]), (2) the group of signing parties is usually not fixed, and each party can initiate the signing protocol with a dynamically chosen set of parties associated with $L = \{pk_1, \ldots, pk_n\}$, and (3) the verification algorithm usually doesn't take a single fixed public key, and instead takes a particular set of public keys L that involved in the signing protocol. Hence, roughly speaking, multi-signatures have more flexibility than n-out-of-n signatures in terms of the choice of co-signers, at the cost of larger joint public key size and verification time (unless more advanced feature like key aggregation [MPSW19] is supported).

Schnorr vs FSwA. There is a long line of research that starts from Schnorr's signature scheme [Sch90] and follows the standard Fiat–Shamir paradigm to build distributed signatures [SS01, NKDM03, AF04, GJKR07, KG20] and multi-signatures [MOR01, BN06, BCJ08, MWLD10, STV+16, MPSW19, NRSW20, NRS20]. In particular, Drijvers et al. [DEF+19] recently discovered a flaw of the existing *two-round* Schnorr-based multi-signatures, with a novel concurrent attack relying on the generalized birthday algorithm of Wagner [Wag02]. They accordingly proposed mBCJ scheme, a provably secure variant of Bagherzhandi et al.'s BCJ scheme [BCJ08].

Unlike distributed n-out-of-n signatures, several *three* or *four-round* multi-signatures based on FSwA are already present in the literature. Bansarkhani and Sturm [BS16] extended the Güneysu–Lyubashevsky–Pöppelmann (GLP) [GLP12] signature and proposed the first multi-signature following the FSwA paradigm, which was recently followed by multiple similar variants [MJ19, TLT19, TE19, FH19, FH20]. Relying on the syntactic similarities between Schnorr and FSwA-style signatures, these protocols essentially borrow the ideas of Schnorr-based counterparts; for instance, [BS16] can be considered as a direct adaptation of Bellare–Neven's three-round Schnorr-like multi-signature [BN06]. However, as explained below, the security proofs of all these protocols are either incomplete or relying on a non-standard hardness assumption, where the underlying problem only emerges in the Fiat–Shamir *with aborts* setting. Therefore, we are also motivated to construct a provably secure multi-signature protocol within this paradigm, while making the most of useful observations from the discrete log setting.

Issue with "aborts". We first observe that there is an inherent issue when constructing distributed FSwA signatures. Just like earlier constructions in the discrete log setting [BN06, NKDM03] previous FSwA multi-signatures ask all parties to start doing what is essentially a single-user FSwA signature, and always reveal the first "commit" message of the underlying Σ-protocol. Then all these messages are added up and the sum is hashed, together with the message to be signed, in order to obtain the challenge. This means that all executions are revealed, whether they abort or not. An important issue with the FSwA approach is that, currently there is no known general way to prove the underlying Σ-protocol zero-knowledge in case of aborts [BCK+14, §3.2], [ESS+19, §4], [BBE+18, BBE+19], [Lyu19, p.26]. As a result, the signer should not reveal any of the aborted executions since otherwise the scheme cannot be proved secure. This issue is not serious in a single-user scheme, since the Σ-protocol is made non-interactive in the random oracle model anyway and there is no reason why the signer would reveal aborted executions.

In an interactive setting, the standard approach to circumvent the issue is to send a commitment to the first Σ-protocol message and only reveal it if the rejection sampling is successful. However, the previous FSwA multi-signatures skipped this subtle step. Thus the concurrent work by Fukumitsu and Hasegawa [FH20] (who constructed a FSwA-style multi-signature proven secure in QROM) had to rely on an additional non-standard assumption (which they call "rejected LWE"), while publicly available security proofs of other similar constructions [BS16, FH19, TLT19, MJ19, TE19] do not explain how to simulate the aborted executions. Despite the lack of such discussion in the proofs there are no known concrete attacks against the existing schemes, and it may be that one could patch the problem by making additional non-standard assumptions, or by carefully choosing the parameter such that the additional assumptions hold unconditionally. Still, it is paramount to strive to find protocols which can be proven secure relying on well-established computational hardness assumptions like LWE and SIS.

1.1 Contributions

FSwA-based distributed signatures with full security proof. In this paper we construct FSwA-type n-out-of-n distributed and multi-signature protocols solely relying on the hardness of learning with errors (LWE) and short integer solution (SIS) problems. Our constructions can be seen as distributed variants of the fast Dilithium-G signature scheme [DLL+18]. As a first step, we circumvent the aborts issue mentioned above by utilizing Baum et al.'s *additively homomorphic* commitment scheme [BDL+18], which is currently the most efficient construction based on lattices and relies on the hardness of Module-LWE and Module-SIS problems. This results in a provably secure, three-round n-out-of-n signature protocol DS_3[3].

[3] It is still an open question whether the aborts issue can instead be resolved by careful parameter choice, allowing to simulate the rejected transcripts without any additional assumptions. But we are aware of on-going work in this direction. If the question is answered in the affirmative our three-round protocol could be proven secure even without a commitment. However, the use of homomorphic commitment is crucial for constructing our new two-round protocols, which is our main contribution.

First two-round protocols. Previous FSwA-based multi-signatures required at least three rounds of interaction. On the other hand, as most recent discrete log-based solutions indicate [DEF+19, KG20, NRSW20, NRS20], two rounds is a natural goal because this clearly seems to be minimal for a distributed signature protocol based on the Fiat-Shamir paradigm: we first need to determine what should be hashed in order to get the challenge in the underlying Σ-protocol. This must (for security) include randomness from several players and hence requires at least one round of interaction. After this we need to determine the prover's answer in the Σ-protocol. This cannot be computed until the challenge is known and must (for security) require contributions from several players, and we therefore need at least one more round.

In this paper, we show that the application of homomorphic commitment not only resolves the issue with aborts, but also makes it possible to reduce the round complexity to two rounds. We do this by adding a *trapdoor* feature to the commitment scheme (a separate contribution that we discuss in more detail below). This results in a two-round, n-out-of-n signature protocol DS_2 presented in Section 3. With a slight modification this n-out-of-n protocol can be also turned into a two-round multi-signature scheme in the plain public key model. We describe a multi-signature variant MS_2 in Appendix D.

Our main two-round result highlights several important similarities and differences which emerge when translating a discrete log-based protocol to lattice-based one. The approaches taken in our two-round protocols are highly inspired by mBCJ discrete log-based multi-signature by Drijvers et al. [DEF+19] In particular, we observe that it is crucial for two-round protocols to use per-message commitment keys (as in mBCJ) instead of a single fixed key for all signing attempts (as in the original BCJ [BCJ08]), because otherwise the proof doesn't go through. Drijvers et al. only presented a full security proof for the protocol in the *key verification model*, in which each co-signer has to submit a zero-knowledge proof of knowledge of the secret key. Our protocols confirm that a similar approach securely transfers to the lattice setting even under different security models: distributed n-out-of-n signature with dedicated key generation phase, and multi-signature in the plain public key model.

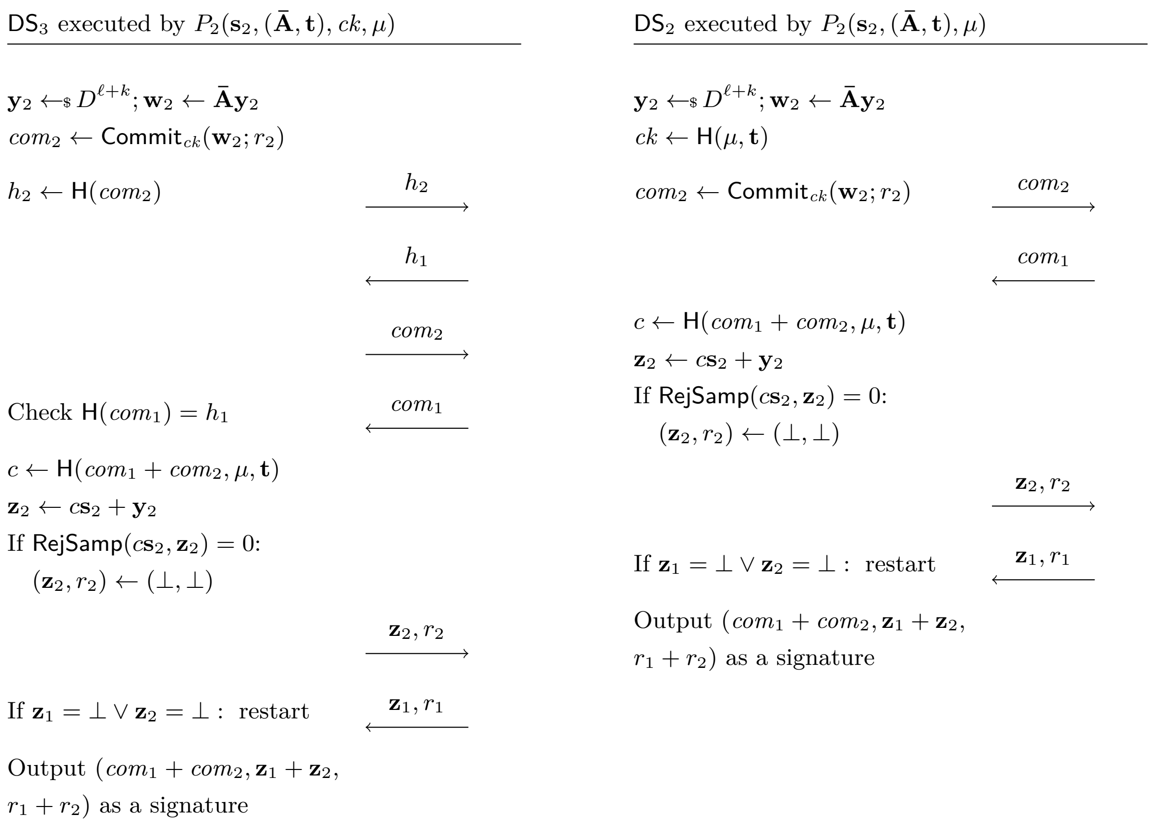
Lattice-based trapdoor commitment. As mentioned above, we turn Baum et al.'s scheme into a trapdoor commitment in Section 4, so that the two-round protocols DS_2 and MS_2 are indeed instantiable with only lattice-based assumptions. We make use of the lattice trapdoor by Micciancio and Peikert [MP12] to generate a trapdoor commitment key in the ring setting. The only modification required is that the committer now samples randomness from the discrete Gaussian distribution instead of the uniform distribution. This way, the committer holding a trapdoor of the commitment key can equivocate a commitment to an arbitrary message by sampling a small randomness vector from the Gaussian distribution. Such randomness is indeed indistinguishable from the actual randomness used in the committing algorithm. Since only a limited number of lattice-based trapdoor commitment schemes are known [GSW13, CHKP10, DM14, GVW15, LNTW19] our technique may be of independent interest.

1.2 Technical Overview

Our protocols are based on Dilithium signature scheme, which works over rings $R = \mathbb{Z}[X]/(f(X))$ and $R_q = \mathbb{Z}_q[X]/(f(X))$ defined with an appropriate irreducible polynomial $f(X)$ (see preliminaries for more formal details). Here we go over the core ideas of our construction by considering simple 2-out-of-2 signing protocols. The protocols below can be generalized to an n-party setting in a straightforward manner. We assume that each party P_j for $j = 1, 2$ owns a secret signing key share $\mathbf{s}_j \in R^{\ell+k}$ which has small coefficients, and a public random matrix $\bar{\mathbf{A}} = [\mathbf{A}|\mathbf{I}] \in R_q^{k\times(\ell+k)}$. The joint public verification key is defined as $\mathbf{t} = \bar{\mathbf{A}}(\mathbf{s}_1 + \mathbf{s}_2) \bmod q$. In the actual protocols the public key $\mathbf{t}$ also needs to be generated in a distributed way, but here we omit the key generation phase for brevity's sake.

Naive approach. We first present a naive (insecure) way to construct a 2-party signing protocol from FSwA. If the reader is familiar with CoSi Schnorr multi-signature [STV+16] this construction is essentially its lattice-based, 2-out-of-2 variant. In this protocol the parties P_j for $j = 1, 2$ involved in signing the message μ work as follows.

1. P_j samples a randomness $\mathbf{y}_j$ from some distribution $D^{\ell+k}$ defined over $R^{\ell+k}$ (which is typically the uniform distribution over a small range or discrete Gaussian), and then sends out the first message of FSwA $\mathbf{w}_j = \bar{\mathbf{A}}\mathbf{y}_j \bmod q$.
2. P_j locally derives a joint challenge $c \leftarrow \mathsf{H}(\mathbf{w}_1 + \mathbf{w}_2, \mu, \mathbf{t})$ and performs the rejection sampling $\mathsf{RejSamp}(c\mathbf{s}_j, \mathbf{z}_j)$ with $\mathbf{z}_j = c\mathbf{s}_j + \mathbf{y}_j$; if the result of $\mathsf{RejSamp}(c\mathbf{s}_j, \mathbf{z}_j)$ is "reject" then P_j sets $\mathbf{z}_j := \bot$. After exchanging $\mathbf{z}_j$'s if $\mathbf{z}_1 = \bot$ or $\mathbf{z}_2 = \bot$ (i.e., either of the parties aborts), then the protocol restarts from the step 1.



DS_3 executed by $P_2(\mathbf{s}_2, (\bar{\mathbf{A}}, \mathbf{t}), ck, \mu)$
$\mathbf{y}_2 \leftarrow_{\$} D^{\ell+k}; \mathbf{w}_2 \leftarrow \bar{\mathbf{A}}\mathbf{y}_2$
$com_2 \leftarrow \mathsf{Commit}_{ck}(\mathbf{w}_2; r_2)$
$h_2 \leftarrow \mathsf{H}(com_2)$
$\xrightarrow{h_2}$
$\xleftarrow{h_1}$
$\xrightarrow{com_2}$
Check $\mathsf{H}(com_1) = h_1$
$\xleftarrow{com_1}$
$c \leftarrow \mathsf{H}(com_1 + com_2, \mu, \mathbf{t})$
$\mathbf{z}_2 \leftarrow c\mathbf{s}_2 + \mathbf{y}_2$
If $\mathsf{RejSamp}(c\mathbf{s}_2, \mathbf{z}_2) = 0$:
$(\mathbf{z}_2, r_2) \leftarrow (\perp, \perp)$
$\xrightarrow{\mathbf{z}_2, r_2}$
If $\mathbf{z}_1 = \perp \vee \mathbf{z}_2 = \perp :$ restart
$\xleftarrow{\mathbf{z}_1, r_1}$
Output $(com_1 + com_2, \mathbf{z}_1 + \mathbf{z}_2,$
$r_1 + r_2)$ as a signature
DS_2 executed by $P_2(\mathbf{s}_2, (\bar{\mathbf{A}}, \mathbf{t}), \mu)$
$\mathbf{y}_2 \leftarrow_{\$} D^{\ell+k}; \mathbf{w}_2 \leftarrow \bar{\mathbf{A}}\mathbf{y}_2$
$ck \leftarrow \mathsf{H}(\mu, \mathbf{t})$
$com_2 \leftarrow \mathsf{Commit}_{ck}(\mathbf{w}_2; r_2)$
$\xrightarrow{com_2}$
$\xleftarrow{com_1}$
$c \leftarrow \mathsf{H}(com_1 + com_2, \mu, \mathbf{t})$
$\mathbf{z}_2 \leftarrow c\mathbf{s}_2 + \mathbf{y}_2$
If $\mathsf{RejSamp}(c\mathbf{s}_2, \mathbf{z}_2) = 0$:
$(\mathbf{z}_2, r_2) \leftarrow (\perp, \perp)$
$\xrightarrow{\mathbf{z}_2, r_2}$
If $\mathbf{z}_1 = \perp \vee \mathbf{z}_2 = \perp :$ restart
$\xleftarrow{\mathbf{z}_1, r_1}$
Output $(com_1 + com_2, \mathbf{z}_1 + \mathbf{z}_2,$
$r_1 + r_2)$ as a signature

Fig. 1. Comparison of different instantiations of FSwA-based 2-party signing protocols

3. Each party outputs $(\mathbf{w}, \mathbf{z}) := (\mathbf{w}_1 + \mathbf{w}_2, \mathbf{z}_1 + \mathbf{z}_2)$ as a signature on μ.

Note that the rejection sampling step is needed to make the distribution of $\mathbf{z}_j$ independent of a secret $\mathbf{s}_j$. The verification algorithm checks that the norm of $\mathbf{z}$ is small, and that $\bar{\mathbf{A}}\mathbf{z} - c\mathbf{t} = \mathbf{w} \pmod q$ holds, where the challenge is recomputed as $c \leftarrow \mathsf{H}(\mathbf{w}, \mu, \mathbf{t})$. One can easily check that the signature generated as above satisfies correctness, thanks to the linearity of the SIS function $f_{\bar{\mathbf{A}}}(\mathbf{x}) = \bar{\mathbf{A}}\mathbf{x} \bmod q$. However, we observe that an attempt to give a security proof fails due to two problems. Suppose the first party $\widetilde{P}_1$ is corrupt and let us try to simulate the values returned by honest P_2, whenever queried by the adversary.

First, since the protocol reveals $\mathbf{w}_2$ whether P_2 aborts or not, the joint distribution of rejected transcript $(\mathbf{w}_2, c, \perp)$ has to be simulated. As mentioned earlier, there is no known way to simulate it; in fact, the honest verifier zero knowledge (HVZK) of FSwA is only proven for "non-aborting" cases in the original work by Lyubashevsky [Lyu09, Lyu12, Lyu19] and its successors. Note that the obvious fix where players hash the initial messages and only reveal them if there is no abort will not work here: the protocols need to *add* the initial messages together before obtaining the challenge c in order to reduce signature size, only the sum is included in the signature. So with this approach the initial messages must be known in the clear before the challenge can be generated.

The second problem is more generic and could also occur in the standard Fiat–Shamir-style two party signing: if P_2 sends out $\mathbf{w}_2$ first, then the simulator does not know $\mathbf{w}_1$. In FS-style constructions, the usual strategy for signing oracle query simulation is to first sample a challenge c by itself, generate a simulated transcript $(\mathbf{w}_2, c, \mathbf{z}_2)$ by invoking a special HVZK simulator on c, and then program the random oracle H such that its output is fixed to a predefined challenge c. In the two-party setting, however, derivation of the *joint* challenge $c = \mathsf{H}(\mathbf{w}_1 + \mathbf{w}_2, \mu, \mathbf{t})$ requires contribution from $\widetilde{P}_1$ and thus there is no way for the simulator to program H in advance. Not only the proof doesn't go through, but also this naive construction is amenable to a concrete attack, which allows malicious $\widetilde{P}_1$ to create a valid forgery by adaptively choosing $\mathbf{w}_1$ after seeing $\mathbf{w}_2$. In Appendix E we describe this attack relying on a variant of Wagner's generalized birthday problem [Wag02, HJ10].

Homomorphic commitment to simulate aborts. We now present an intermediate provably secure protocol that circumvents the above issues. See DS_3 in Fig. 1. To address the first issue with aborts, each player P_j now commits to an initial Σ-protocol message $\mathbf{w}_j$ using an *additively homomorphic* commitment

com_j. Thanks to the hiding property, each party leaks no useful information about $\mathbf{w}_j$ until the rejection sampling is successful, and thus it is now possible to simulate a rejected transcript $(com_j, c, \perp)$. Then P_j broadcasts a hash based commitment to com_j, to deal with the second issue. Once all parties have done this, com_j's are revealed in the next round and checked against the hashes. Once the com_j's are known, they can be added together in a meaningful way by the homomorphic property, and we then hash the sum and the message to get the challenge. The verification now receives a signature consisting of three elements $(com, \mathbf{z}, r)$ and simply checks that $\bar{\mathbf{A}}\mathbf{z} - c\mathbf{t} \pmod q$ and r form a correct opening to com, where the challenge is recomputed as $c \leftarrow \mathsf{H}(com, \mu, \mathbf{t})$.

We note that the extra round for hash commitment is a standard technique, previously used in multiple three-round protocols, such as Nicolosi et al. [NKDM03] and Bellare and Neven [BN06] in the discrete log setting, and Bansarkhani and Sturm [BS16] in their FSwA-based instantiation. This way, the simulator for honest P_2 can successfully extract corrupt $\widetilde{P}_1$'s share com_1 by keeping track of incoming queries to H (when modeled as a random oracle), and program H such that $\mathsf{H}(com_1 + com_2, \mu, \mathbf{t}) := c$ before revealing com_2. For the completeness, in Appendix F we provide a formal security proof for DS_3 by showing a reduction to Module-LWE *without* relying on the forking lemma [PS00, BN06]. This is made possible by instantiating the construction with unconditionally binding commitment, which allows us to avoid rewinding the adversary and apply the *lossy identification* technique by Abdalla et al. [AFLT16].

One efficiency issue is, that the protocol has to be restarted until *all* parties pass the rejection sampling step simultaneously. All previous FSwA-based multi-signatures also had the same issues, but we can mitigate by running sufficiently many parallel executions of the protocol at once, or by carefully choosing the parameters for rejection sampling. To further reduce the number of aborts, we chose to instantiate the protocol with Dilithium-"G" [DLL+18] instead of the one submitted to NIST competition [LDK+19].

Trapdoor commitment to avoid the extra round. Although DS_3 is secure, the first round of interaction may seem redundant, since the parties are essentially "committing to a commitment". We show that the extra hash commitment round can be indeed dropped by adding a trapdoor feature to the commitment scheme, which allows the so-called *straight-line simulation* technique by Damgård [Dam00]. We present our main two-round protocol DS_2 in Fig. 1. This way, the simulation of honest P_2 does not require the knowledge of corrupt $\widetilde{P}_1$'s commitment share; instead, the simulator can now simply send a commitment com_2 (to some random value) and then later equivocate to an arbitrary value using the known trapdoor td associated with a trapdoored commitment key tck. Concretely, the simulator need not program the random oracle this time, and instead derives a challenge $c \leftarrow \mathsf{H}(com_1 + com_2, \mu, \mathbf{t})$ as the real honest party would do. Now the simulator invokes a (special) HVZK simulator with c as input, to obtain a transcript $(\mathbf{w}_2, c, \mathbf{z}_2)$. With some constant probability it equivocates com_2 to $\mathbf{w}_2$, or otherwise sends out $\perp$ to simulate aborts. We also stress that the *per-message commitment key* $ck \leftarrow \mathsf{H}(\mu, \mathbf{t})$ is crucial in the two-round protocol; if a single ck is used across all signing attempts, then a concurrent attack similar to the one against the naive construction becomes applicable (see Appendix E). Unlike the three-round protocol, we present a security proof relying on the forking lemma and we reduce the security to both Module-SIS and Module-LWE assumptions; since a trapdoor commitment can at most be computationally binding, we must extract from the adversary two different openings to the same commitment in order to be able to reduce security to the binding property. We leave for future work a tighter security proof for the two-round protocol.

Two-round multi-signature. We can now convert to a two-round multi-signature scheme in the plain public key model: following Bellare–Neven [BN06] the protocol now generates *per-user challenges* $c_j = \mathsf{H}(\mathbf{t}_j, \sum_j com_j, \mu, L)$ for each user's public key $\mathbf{t}_j \in L$, instead of interactively generating the fixed joint public key $\mathbf{t}$ in advance. The verification algorithm is adjusted accordingly: given $(com, \mathbf{z}, r)$ and a list of public keys L, the verifier checks that $\bar{\mathbf{A}}\mathbf{z} - \sum_j c_j\mathbf{t}_j \pmod q$ and r form a correct opening to com, where c_j's are recomputed as in the signing protocol. Appendix D formally describes our MS_2 protocol with security proof.

1.3 Related Work

The FSwA paradigm was first proposed by Lyubashevsky [Lyu09, Lyu12] and many efficient signature schemes following this framework have been devised, such as GLP [GLP12], BLISS [DDLL13], Dilithium [LDK+19] and qTESLA [BAA+19]. Bansarkhani and Sturm [BS16] extended GLP signature and proposed the first multi-signature following the FSwA paradigm. Since then several variants appeared in the literature: four-round protocol with public key aggregation [MJ19], three-round protocol with tight security proof [FH19] and proof in QROM [FH20], ID-based blind multi-signature [TLT19] and ID-based proxy multi-signature [TE19]. However, as mentioned earlier the security proofs for all these multi-signatures are either incomplete or rely on a non-standard heuristic assumption. Choi and Kim [CK16] proposed a linearly homomorphic multi-signature from lattices trapdoors. Kansal and Dutta [KD20] constructed a single-round multi-signature scheme relying on the hardness of SIS, which was soon after broken by Liu et al. [LTT20]. Several lattice-based threshold *ring signatures* exist in the literature, such as Cayrel et al. [CLRS10], Bettaieb and Schrek [BS13], and Torres et al. [TSSK20]. Döroz et al. [DHSS20] devised lattice-based *aggregate signature schemes* relying on rejection sampling. Very recently, Esgin et al. [EEE20] developed FSwA-based *adaptor signatures* with application to blockchains.

Our two-round protocols rely on trapdoor commitment to enable the straight-line simulation of ZK. The trick is originated in a concurrent ZK proof by Damgård [Dam00] and similar ideas have been extensively used in the ZK literature [BKLP15,CPS+16,COSV17b,COSV17a], to turn honest verifier ZK proof into full-fledged ZK. Moreover, recent efficient lattice-based ZK proofs [dLS18,ESS+19,YAZ+19,BLS19, ESLL19] also make use of Baum et al.'s additively homomorphic commitment. The issue of revealing the first "commit" message in the FSwA framework has been also discussed by Barthe et al. [BBE+18] in the context of masking countermeasure against side-channel attacks, and they used Baum et al.'s commitment to circumvent the issue. The homomorphic lattice-based trapdoor commitment could also be instantiated with GSW-FHE [GSW13], homomorphic trapdoor functions [GVW15], Chameleon hash [CHKP10,DM14] or mercurial commitment [LNTW19].

Comparison with Bendlin et al. [BKP13] An entirely different approach to constructing threshold signatures based on lattices relies not on the Fiat–Shamir with aborts paradigm, but on GPV hash-and-sign signatures [GPV08]. This approach was introduced by Bendlin et al. in [BKP13], who described how to implement Peikert's hash-and-sign signatures [Pei10] in a multiparty setting. Compared to the approach in this paper, it has the advantage of realizing the same distributed signature scheme (e.g., with the same size bound for verification) independently of the number of parties, and in particular, signature size does not grow with the number of parties. Moreover, it supports more general access structure than the full threshold considered in this paper (although their protocol does not withstand dishonest majority for the sake of information-theoretic security, while our protocol does tolerate up to $n-1$ corrupt parties). Its main downside, however, is that the most expensive part of Peikert's signing algorithm, namely the offline lattice Gaussian sampling phase, is carried out using *generic multiparty computation* (this is the first step of the protocol π_{Perturb} described in [BKP13, Fig. 23]). This makes it difficult to estimate the concrete efficiency of Bendlin et al.'s protocol, but since the Peikert signature scheme is fairly costly even in a single-user setting, the protocol is unlikely to be practical.

In contrast, while our protocols do use discrete Gaussian sampling, it is only carried out *locally by each party*, and it is Gaussian sampling over $\mathbb{Z}$ rather than a lattice, which is considerably less costly. Furthermore, while we also use lattice trapdoors as a proof technique in the trapdoor commitment scheme of our two-round protocol, trapdoor Gaussian sampling is never carried out in the actual protocol, only in the simulation (the actual protocol has no trapdoor). Thus, our protocols entirely avoid the expensive machinery present in Bendlin et al.'s scheme, and have a fully concrete instantiation (at the cost of signatures increasing in size with the number of parties).

2 Preliminaries

Notations. For positive integers a and b such that $a < b$ we use the integer interval notation $[a, b]$ to denote $\{a, a+1, \dots, b\}$; we use $[b]$ as shorthand for $[1, b]$. If S is a set we write $s \leftarrow_{\$} S$ to indicate sampling s from the uniform distribution defined over S; if D is a probability distribution we write $s \leftarrow_{\$} D$ to indicate sampling s from the distribution D; if we are explicit about the set S over which the distribution D is defined then we write $D(S)$; if $\mathcal{A}$ is an algorithm we write $s \leftarrow \mathcal{A}$ to indicate assigning an output from $\mathcal{A}$ to s.

2.1 Polynomial Rings and Discrete Gaussian Distribution

In this paper most operations work over rings $R = \mathbb{Z}[X]/(f(X))$ and $R_q = \mathbb{Z}_q[X]/(f(X))$, where q is a modulus, N is a power of two defining the degree of $f(X)$, and $f(X) = X^N + 1$ is the $2N$-th cyclotomic polynomial. Following [DLL+18], we consider *centered modular reduction* $\bmod {}^{\pm}q$: for any $v \in \mathbb{Z}_q$, $v' = v \bmod {}^{\pm}q$ is defined to be a unique integer in the range $[-\lfloor q/2 \rfloor, \lfloor q/2 \rfloor]$ such that $v' = v \mod q$. We define the norm of $v \in \mathbb{Z}_q$ such that $\|v\| := |v \mod {}^{\pm}q|$. Now we define the L^p-norm for a (vector of) ring element $\mathbf{v} = (\sum_{i=0}^{N-1} v_{i,1}X^i, \dots, \sum_{i=0}^{N-1} v_{i,m}X^i)^T \in R^m$ as follows.

$$\|\mathbf{v}\|_p := \left\|(v_{0,1}, \ldots, v_{N-1,1}, \ldots, v_{0,m}, \ldots, v_{N-1,m})^T\right\|_p.$$

We rely on the following *key set* $S_\eta \subseteq R$ parameterized by $\eta \geq 0$ consisting of small polynomials.

$$S_\eta = \{x \in R : \|x\|_\infty \leq \eta\}$$

Moreover the *challenge set* $C \subseteq R$ parameterized by $\kappa \geq 0$ consists of small and sparse polynomials, which will be used as the image of random oracle H_0.

$$C = \{c \in R : \|c\|_\infty = 1 \wedge \|c\|_1 = \kappa\}$$

The discrete Gaussian distribution over R^m is defined as follows.

Definition 1 (Discrete Gaussian Distribution over R^m). *For* $\mathbf{x} \in R^m$, *let* $\rho_{\mathbf{v},s}(\mathbf{x}) = \exp\left(-\pi \|\mathbf{x} - \mathbf{v}\|_2^2 / s^2\right)$ *be a* Gaussian function *of parameters* $\mathbf{v} \in R^m$ *and* $s \in \mathbb{R}$. *The* discrete Gaussian distribution $D^m_{\mathbf{v},s}$ *centered at* $\mathbf{v}$ *is*

$$D^m_{\mathbf{v},s}(\mathbf{x}) := \rho_{\mathbf{v},s}(\mathbf{x}) / \rho_{\mathbf{v},s}(R^m)$$

where $\rho_{\mathbf{v},s}(R^m) = \sum_{\mathbf{x} \in R^m} \rho_{\mathbf{v},s}(\mathbf{x})$.

In what follows we omit the subscript $\mathbf{v}$ if $\mathbf{v} = \mathbf{0}$ and write D^m_s as a shorthand. When s exceeds the so-called *smoothing parameter* $\eta(R^m) \leq \omega(\sqrt{\log(mN)})$ of the ambient space, then the discrete Gaussians $D_{R^m - \mathbf{v},s} = D^m_{\mathbf{v},s} - \mathbf{v}$ supported on all cosets of R^m are statistically close, and hence D^m_s behaves qualitatively like a continuous Gaussian of standard deviation $\sigma = s/\sqrt{2\pi}$. The condition on s will be satisfied for all the discrete Gaussians in this paper, and hence $\sigma = s/\sqrt{2\pi}$ will be called the standard deviation (even though it technically holds only up to negligible error). For the same reason, we will always be in a setting where the following fact [MP13, Theorem 3.3] [ESLL19, Lemma 9] holds.

Lemma 1 (Sum of Discrete Gaussian Samples). *Suppose s exceeds the smoothing parameter by a factor $\geq \sqrt{2}$. Let $\mathbf{x}_i$ for $i \in [n]$ be independent samples from the distribution D^m_s. Then the distribution of $\mathbf{x} = \sum_i \mathbf{x}_i$ is statistically close to $D^m_{s\sqrt{n}}$.*

2.2 Lattice Problems

Below we define two standard lattice problems over rings: module short integer solution (MSIS) and learning with errors (MLWE). We also call them MSIS/MLWE *assumption* if for any probabilistic polynomial-time adversaries the probability that they can solve a given problem is negligible.

Definition 2 ($\mathsf{MSIS}_{q,k,\ell,\beta}$ problem). *Given a random matrix $\mathbf{A} \leftarrow_\$ R_q^{k\times\ell}$ find a vector $\mathbf{x} \in R_q^{\ell+k} \setminus \{\mathbf{0}\}$ such that $[\mathbf{A}|\mathbf{I}] \cdot \mathbf{x} = \mathbf{0}$ and $\|\mathbf{x}\|_2 \leq \beta$.*

Definition 3 ($\mathsf{MLWE}_{q,k,\ell,\eta}$ problem). *Given a pair $(\mathbf{A}, \mathbf{t}) \in R_q^{k\times\ell} \times R_q^k$ decide whether it was generated uniformly at random from $R_q^{k\times\ell} \times R_q^k$, or it was generated in a way that $\mathbf{A} \leftarrow_\$ R_q^{k\times\ell}, \mathbf{s} \leftarrow_\$ S_\eta^{\ell+k}$ and $\mathbf{t} := [\mathbf{A}|\mathbf{I}] \cdot \mathbf{s}$.*

2.3 Fiat–Shamir with Aborts Framework and Dilithium-G

Algorithm 1 Key generation

```
Require: $pp = (R_q, k, \ell, \eta, B, \sigma, M)$
Output: $(sk, pk)$
1: $\mathbf{A} \leftarrow_\$ R_q^{k\times\ell}$
2: $\bar{\mathbf{A}} := [\mathbf{A}|\mathbf{I}] \in R_q^{k\times(\ell+k)}$
3: $(\mathbf{s}_1, \mathbf{s}_2) \leftarrow_\$ S_\eta^\ell \times S_\eta^k; \mathbf{s} := \begin{bmatrix}\mathbf{s}_1\\ \mathbf{s}_2\end{bmatrix}$
4: $\mathbf{t} := \bar{\mathbf{A}}\mathbf{s}$
5: $sk := \mathbf{s}$
6: $pk := (\bar{\mathbf{A}}, \mathbf{t})$
7: return $(sk, pk)$
```

Algorithm 2 Signature verification

```
Require: $pk, (\mathbf{z}, c), \mu, pp$
1: If $\|\mathbf{z}\|_2 \leq B$ and $c = \mathsf{H}_0(\bar{\mathbf{A}}\mathbf{z} - c\mathbf{t}, \mu, \mathbf{t})$:
2:     return 1
3: Otherwise: return 0
```

Algorithm 3 Signature generation

```
Require: $sk, \mu, pp = (R_q, k, \ell, \eta, B, \sigma, M)$
Output: valid signature pair $(\mathbf{z}, \mathbf{c})$
1: $(\mathbf{y}_1, \mathbf{y}_2) \leftarrow_\$ D_s^\ell \times D_s^k; \mathbf{y} := \begin{bmatrix}\mathbf{y}_1\\ \mathbf{y}_2\end{bmatrix}$
2: $\mathbf{w} := \bar{\mathbf{A}}\mathbf{y}$
3: $c \leftarrow \mathsf{H}_0(\mathbf{w}, \mu, \mathbf{t})$
4: $\mathbf{z} := c\mathbf{s} + \mathbf{y}$
5: With prob. $\min\left(1, D_s^{\ell+k}(\mathbf{z})/(M \cdot D_{c\mathbf{s},\sigma}^{\ell+k}(\mathbf{z}))\right)$:
6:     return $(\mathbf{z}, c)$
7: Restart otherwise
```

Algorithm 4 RS
```
1: $\mathbf{v} \leftarrow_\$ h$
2: $\mathbf{z} \leftarrow_\$ D^m_{\mathbf{v},s}$
3: With prob. $\min\left(1, D^m_s(\mathbf{z})/(M \cdot D^m_{\mathbf{v},s}(\mathbf{z}))\right)$:
4:     return $(\mathbf{z}, \mathbf{v})$
5: Otherwise:
6:     return $(\perp, \perp)$
```

Algorithm 5 SimRS
```
1: $\mathbf{v} \leftarrow_\$ h$
2: $\mathbf{z} \leftarrow_\$ D^m_s$
3: With prob. $1/M$:
4:     return $(\mathbf{z}, \mathbf{v})$
5: Otherwise:
6:     return $(\perp, \perp)$
```

Algorithm 6 Trans(sk, c)
```
1: $\mathbf{y} \leftarrow_\$ D^{\ell+k}_s$
2: $\mathbf{w} := \bar{\mathbf{A}}\mathbf{y}$
3: $\mathbf{z} := c\mathbf{s} + \mathbf{y}$
4: With prob. $\min\left(1, D^{\ell+k}_s(\mathbf{z})/(M \cdot D^{\ell+k}_{c\mathbf{s},s}(\mathbf{z}))\right)$:
5:     return $(\mathbf{w}, c, \mathbf{z})$
6: Otherwise:
7:     return $(\perp, c, \perp)$
```

Algorithm 7 SimTrans(pk, c)
```
1: $\mathbf{z} \leftarrow_\$ D^{\ell+k}_s$
2: $\mathbf{w} := \bar{\mathbf{A}}\mathbf{z} - c\mathbf{t}$
3: With prob. $1/M$:
4:     return $(\mathbf{w}, c, \mathbf{z})$
5: Otherwise:
6:     return $(\perp, c, \perp)$
```

We present a non-optimized version of Dilithium-G signature scheme in Algorithms 1 to 3, on which we base our distributed signing protocols. The random oracle is defined as $\mathsf{H}_0 : \{0,1\}^* \to C$. Due to Lemma 2 below the maximum L^2-norm of the signature $\mathbf{z} \in R^{\ell+k}$ is set to $B = \gamma\sigma\sqrt{(\ell+k)N}$, where the parameter $\gamma > 1$ is chosen such that the probability $\gamma^{(\ell+k)N} e^{(\ell+k)N(1-\gamma^2)/2}$ is negligible.

Lemma 2 ([Lyu12]). *For any $\gamma > 1$, $\Pr[\|\mathbf{z}\|_2 > \gamma\sigma\sqrt{mN} : \mathbf{z} \leftarrow_\$ D^m_s] < \gamma^{mN} e^{mN(1-\gamma^2)/2}$.*

The following claim by Lyubashevsky (adapted from [Lyu12, Thoerem 4.6]) is crucial for the signing oracle of FSwA to be simulatable, and also to decide the standard deviation σ as well as the expected number of repetitions M. For instance, setting $\alpha = 11$ and $t = 12$ leads to $M \approx 3$. Although M is asymptotically superconstant, t increases very slowly in practice, and hence M behaves essentially like a constant for practical security parameters (in the literature, it is often taken as 12 to ensure $e^{-t^2/2} < 2^{-100}$, thereby ensuring > 100 bits of security).

Lemma 3 (Rejection Sampling Lemma [Lyu12]). *Let $V \subseteq R^m$ be a set of polynomials such that for all $\mathbf{v} \in V$, it holds that $\|\mathbf{v}\|_2 \le T$, and let $h : V \to \mathbb{R}$ be a probability distribution. Fix some t such that $t = \omega(\sqrt{\log(mN)})$ and $t = o(\log(mN))$. For any $\alpha > 0$ if $\sigma = \alpha T$ and $M = e^{t/\alpha + 1/(2\alpha^2)}$ then the statistical distance between two output distributions of* RS *(Algorithm 4) and* SimRS *(Algorithm 5) is at most $e^{-t^2/2}/M$, which is negligible. Moreover, the probability that* RS *outputs $(\mathbf{z}, \mathbf{v}) \neq (\perp, \perp)$ is at least $(1 - e^{-t^2/2})/M$, which is noticeable.*

We now present a supporting lemma which is required for Dilithium-G to be UF-CMA secure. This is almost a direct consequence of Lemma 3 and a similar result appears in [KLS18, Lemma 4.3] to prove the security of Dilithium signature instantiated with the uniform distribution. We remark that the simulator in Algorithm 7 can only simulate transcripts of non-abort executions in the underlying *interactive Σ-*protocol; in fact, if Trans output $\mathbf{w}$ in case of rejection as it's done in the interactive protocol then there is no known method to simulate the *joint distribution* of $(\mathbf{w}, c, \perp)$ [BCK+14, Lyu19] (without assuming some ad-hoc assumptions like rejection-DCK [BBE+18] or rejected-LWE [FH20]).

Lemma 4 (Non-abort Special Honest Verifier Zero Knowledge). *Suppose T is defined as follows: if $\mathbf{s} \in S^{\ell+k}_\eta$ is sampled uniformly at random, it holds that $T \ge \|c\mathbf{s}\|_2$ with overwhelming probability for any $c \in C$. Fix some t such that $t = \omega(\sqrt{\log(mN)})$ and $t = o(\log(mN))$. For any $\alpha > 0$ if $\sigma = \alpha T$ and $M = e^{t/\alpha + 1/(2\alpha^2)}$, then for any $c \in C$ the output distributions of* Trans(sk, c) *(Algorithm 6) and* SimTrans(pk, c) *(Algorithm 7) are statistically indistinguishable, where $(pk, sk) = ((\bar{\mathbf{A}}, \mathbf{t}), \mathbf{s})$ is output from Algorithm 1.*

Proof. Suppose a set $V_\mathbf{s} = \left\{\mathbf{v} \in R^{\ell+k} : \mathbf{v} = c\mathbf{s} \text{ for } c \in C\right\}$ for a fixed signing key $\mathbf{s} \in S^{\ell+k}_\eta$ chosen uniformly at random. Due to the choice of T Lemma 3 ensures that the statistical distance would be negligible if both algorithms output $\mathbf{v} \in V_\mathbf{s}$ instead of c. Since for every $\mathbf{v} \in V_\mathbf{s}$ there exists a unique $c \in C$ such that $c\mathbf{s} = \mathbf{v}$ holds [Lyu12, DLL+18], replacing the output with c doesn't increase the statistical distance. Finally, outputting $\mathbf{w}$ doesn't affect the statistical distance either since the underlying identification protocol is *commitment recoverable* [KLS18] and therefore $\mathbf{w}$ can be simply reconstructed using c, $\mathbf{z}$ and pk anyway.

2.4 Trapdoor Homomorphic Commitment Scheme

Below we formally define a trapdoor commitment scheme. In Appendix B.1 we also define standard security requirements like hiding and binding, as well as the two additional properties required by our protocols: additive homomorphism and uniform key. The lattice-based commitments described in Section 4 indeed satisfy all of them. The uniform property is required since our protocols rely on commitment key derivation via random oracles mapping to a key space S_{ck}, and thus its output distribution should look like the one from CGen. Many other standard schemes like Pedersen commitment [Ped92] trivially satisfy this property. The additive homomorphism is also needed to preserve the algebraic structure of the first "commit" message of FSwA.

Definition 4 (Trapdoor Commitment Scheme). *A trapdoor commitment scheme* TCOM *consists of the following algorithms.*

- $\mathsf{CSetup}(1^\lambda) \to cpp$: *The setup algorithm outputs a public parameter* cpp *defining sets* $S_{ck}, S_{msg}, S_r, S_{com}$ *and the distribution* $D(S_r)$ *from which the randomness is sampled.*
- $\mathsf{CGen}(cpp) \to ck$: *The key generation algorithm that samples a commitment key from* S_{ck}.
- $\mathsf{Commit}_{ck}(msg; r) \to com$: *The committing algorithm that takes a message* $msg \in S_{msg}$ *and randomness* $r \in S_r$ *as input and outputs* $com \in S_{com}$. *We simply write* $\mathsf{Commit}_{ck}(msg)$ *when it uses* r *sampled from* $D(S_r)$.
- $\mathsf{Open}_{ck}(com, r, msg) \to b$: *The opening algorithm outputs* $b = 1$ *if the input tuple is valid, and* $b = 0$ *otherwise.*
- $\mathsf{TCGen}(cpp) \to (tck, td)$: *The trapdoor key generation algorithm that outputs* $tck \in S_{ck}$ *and the trapdoor* $td \in S_{td}$.
- $\mathsf{TCommit}_{tck}(td) \to com$: *The trapdoor committing algorithm that outputs a commitment* $com \in S_{com}$.
- $\mathsf{Eqv}_{tck}(td, com, msg) \to r$: *The equivocation algorithm that outputs randomness* $r \in S_r$.

A trapdoor commitment is said to be secure if it is unconditionally hiding, computationally binding, and for any $msg \in S_{msg}$, *the statistical distance* ϵ_{td} *between* (ck, msg, com, r) *and* (tck, msg, com', r') *is negligible, where* $cpp \leftarrow \mathsf{CSetup}(1^\lambda); ck \leftarrow \mathsf{CGen}(cpp); r \leftarrow_\$ D(S_r); com \leftarrow \mathsf{Commit}_{ck}(msg; r)$ *and* $(tck, td) \leftarrow \mathsf{TCGen}(cpp); com' \leftarrow \mathsf{TCommit}_{tck}(td); r' \leftarrow \mathsf{Eqv}_{tck}(td, com', msg)$.

2.5 Security Notions for n-out-of-n Signature and Multi-Signature

We first define the n-out-of-n distributed signature protocol and its security notion. The game-based security notion below is based on the one presented by Lindell [Lin17] for two-party signing protocol. Our definition can be regarded as its generalization to n-party setting. Following Lindell, we assume that the key generation can be invoked only once, while many signing sessions can be executed concurrently. The main difference is that, in our protocols all players have the same role, and therefore we fix wlog the index of honest party and challenger to n, who has to send out the message first in each round of interaction. This way, we assume that the adversary $\mathcal{A}$ who corrupts $P_1, \ldots, P_{n-1}$ is *rushing* by default (i.e., $\mathcal{A}$ is allowed to choose their own messages based on P_n's message).

Definition 5 (Distributed Signature Protocol). *A distributed signature protocol* DS *consists of the following algorithms.*

- $\mathsf{Setup}(1^\lambda) \to pp$: *The set up algorithm that outputs public parameters* pp *on a security parameter* λ *as input.*
- $\mathsf{Gen}_j(pp) \to (sk_j, pk)$ *for each* $j \in [n]$: *The interactive key generation algorithm that is run by party* P_j. *Each* P_j *runs the protocol on public parameters* pp *as input. At the end of the protocol* P_j *obtains a secret key share* sk_j *and public key* pk.
- $\mathsf{Sign}_j(sid, sk_j, pk, \mu) \to \sigma$ *for each* $j \in [n]$: *The interactive signing algorithm that is run by by party* P_j. *Each* P_j *runs the protocol on session ID* sid, *its signing key share* sk_j, *public key* pk, *and message to be signed* μ *as input. We also assume that the algorithm can use any state information obtained during the key generation phase. At the end of the protocol* P_j *obtains a signature* σ *as output.*
- $\mathsf{Ver}(\sigma, \mu, pk) \to b$: *The verification algorithm that takes a signature, message, and a single public key* pk *and outputs* $b = 1$ *if the signature is valid and otherwise* $b = 0$.

```
$\mathsf{Exp}^{\mathsf{DS\text{-}UF\text{-}CMA}}_{\mathsf{DS}}(\mathcal{A})$
1 : $Mset \leftarrow \varnothing$
2 : $pp \leftarrow \mathsf{Setup}(1^\lambda)$
3 : $(\mu^*, \sigma^*) \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{DS}}_n(\cdot,\cdot)}(pp)$
4 : $b \leftarrow \mathsf{Ver}(\mu^*, \sigma^*, pk)$
5 : $\mathbf{return}\ (b = 1) \land \mu^* \notin Mset$
```

```
$\mathsf{Exp}^{\mathsf{MS\text{-}UF\text{-}CMA}}_{\mathsf{MS}}(\mathcal{A})$
1 : $Mset \leftarrow \varnothing$
2 : $pp \leftarrow \mathsf{Setup}(1^\lambda)$
3 : $(sk, pk) \leftarrow \mathsf{Gen}(pp)$
4 : $(\mu^*, \sigma^*, L^*) \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{MS}}(\cdot,\cdot)}(pk, pp)$
5 : $b \leftarrow \mathsf{Ver}(\mu^*, \sigma^*, L^*)$
6 : $\mathbf{return}\ (b = 1) \land pk \in L^* \land (\mu^*, L^*) \notin Mset$
```

Fig. 2. DS-UF-CMA and MS-UF-CMA experiments. The oracles $\mathcal{O}^{\mathsf{DS}}_n$ and $\mathcal{O}^{\mathsf{MS}}$ are described in Figs. 3 and 4. In the left (resp. right) experiment, *Mset* is the set of all inputs μ (resp. (μ, L)) such that (sid, μ) (resp. $(sid, (\mu, L))$) was queried by $\mathcal{A}$ to its oracle as the first query with identifier $sid \neq 0$ (resp with any identifier sid). Note that pk in the left experiment is the public verification key output by P_n when it completes $\mathsf{Gen}_n(pp)$.

Oracle $\mathcal{O}^{\mathsf{DS}}_n(sid, m)$

The oracle is initialized with public parameters pp generated by Setup algorithm. The variable *flag* is initially set to false.

Key Generation Upon receiving $(0, m)$, if $flag = \mathsf{true}$ then return $\perp$. Otherwise do the following:

- If the oracle is queried with $sid = 0$ for the first time then it initializes a machine $\mathcal{M}_0$ running the instructions of party P_n in the distributed key generation protocol $\mathsf{Gen}_n(pp)$.
- If $\mathcal{M}_0$ has been already initialized then the oracle hands the machine $\mathcal{M}_0$ the next incoming message m and returns $\mathcal{M}_0$'s reply. If $\mathcal{M}_0$ concludes with local output (sk_n, pk), then set $flag = \mathsf{true}$.

Signature Generation Upon receiving (sid, m) with $sid \neq 0$, if $flag = \mathsf{false}$ then return $\perp$. Otherwise do the following:

- If the oracle is queried with sid for the first time then parse the incoming message m as μ. It initializes a machine $\mathcal{M}_{sid}$ running the instructions of party P_n in the distributed signing protocol $\mathsf{Sign}_n(sid, sk_n, pk, \mu)$. The machine $\mathcal{M}_{sid}$ is initialized with the key share and any state information stored by $\mathcal{M}_0$ at the end of the key generation phase. The message μ to be signed is included in *Mset*. If P_n sends the first message in the signing protocol, then this message is the oracle reply.
- If $\mathcal{M}_{sid}$ has been already initialized then the oracle hands the machine $\mathcal{M}_{sid}$ the next incoming message m and returns the next message sent by $\mathcal{M}_{sid}$. If $\mathcal{M}_{sid}$ concludes with local output σ, then the output obtained by $\mathcal{M}_{sid}$ is returned.

Fig. 3. Honest party oracle for the distributed signing protocol.

Oracle $\mathcal{O}^{\mathsf{MS}}(sid, m)$

The oracle is initialized with public parameters pp generated by Setup algorithm.

Signature Generation Upon receiving (sid, m) do the following:

- If the oracle is queried with sid for the first time then parse the incoming message m as (μ, L). If $pk \notin L$ then it returns $\perp$. Otherwise it initializes a machine $\mathcal{M}_{sid}$ running the instructions of party P in the multi-signature protocol $\mathsf{Sign}(sid, sk, pk, \mu, L)$. The machine $\mathcal{M}_{sid}$ is initialized with the key pair (sk, pk) and any state information obtained during $\mathsf{Gen}(pp)$. The pair (μ, L) is included in *Mset*. If P sends the first message in the signing protocol, then this message is the oracle reply.
- If $\mathcal{M}_{sid}$ has been already initialized then the oracle hands the machine $\mathcal{M}_{sid}$ the next incoming message m and returns the next message sent by $\mathcal{M}_{sid}$. If $\mathcal{M}_{sid}$ concludes, then the output obtained by $\mathcal{M}_{sid}$ is returned.

Fig. 4. Honest party oracle for the multi-signature protocol.

Definition 6 (DS-UF-CMA **Security).** *A distributed signature protocol* DS *is said to be* DS-UF-CMA *(distributed signature unforgeability against chosen message attacks) secure, if for any probabilistic polynomial time adversary* $\mathcal{A}$*, its advantage*

$$\mathbf{Adv}_{\mathsf{DS}}^{\mathsf{DS\text{-}UF\text{-}CMA}}(\mathcal{A}) := \Pr\left[\mathsf{Exp}_{\mathsf{DS}}^{\mathsf{DS\text{-}UF\text{-}CMA}}(\mathcal{A}) \to 1\right]$$

is negligible in λ*, where* $\mathsf{Exp}_{\mathsf{DS}}^{\mathsf{DS\text{-}UF\text{-}CMA}}(\mathcal{A})$ *is described in Fig. 2.*

Next we define the standard security notion of multi-signature protocol in the plain public-key model. The following definitions are adapted from [BN06], but the syntax is made consistent with n-out-of-n signing. The main difference from the distributed signature is, that there is no interactive key generation protocol anymore and the adversary is not required to fix its key pair at the beginning of the game. Accordingly, the adversary can dynamically choose a set of public keys involving the challenger's key, and query the signing oracle to receive signatures. On the other hand, assuming that key aggregation is not always supported the verification algorithm takes a set of public keys, instead of a single combined public key as in the prior case. We also note that n is now the number of *maximum* number of parties involved in a single execution of signing protocol, since the size of L may vary depending on a protocol instance.

Definition 7 (Multi-signature Protocol). *A multisignature protocol* MS *consists of the following algorithms.*

- $\mathsf{Setup}(1^\lambda) \to pp$*: The set up algorithm that outputs a public parameter* pp *on a security parameter* λ *as input.*
- $\mathsf{Gen}(pp) \to (sk, pk)$*: The non-interactive key generation algorithm that outputs a key pair on a public parameter* pp *as input.*
- $\mathsf{Sign}(sid, sk, pk, \mu, L) \to \sigma$*: The interactive signing algorithm that is run by a party* P *holding a key pair* (sk, pk)*. Each* P *runs the protocol on session ID* sid*, its signing key* sk*, public key* pk*, message to be signed* μ*, and a set of co-signers' public keys* L *as input. At the end of the protocol* P *obtains a signature* σ *as output.*
- $\mathsf{Ver}(\sigma, \mu, L) \to b$*: The verification algorithm that takes a signature, message, and a set of public keys and outputs* $b = 1$ *if the signature is valid and otherwise* $b = 0$*.*

Definition 8 (MS-UF-CMA **Security).** *A multisignature protocol* MS *is said to be* MS-UF-CMA *(multisignature unforgeability against chosen message attacks) secure, if for any probabilistic polynomial time adversary* $\mathcal{A}$*, its advantage*

$$\mathbf{Adv}_{\mathsf{MS}}^{\mathsf{MS\text{-}UF\text{-}CMA}}(\mathcal{A}) := \Pr\left[\mathsf{Exp}_{\mathsf{MS}}^{\mathsf{MS\text{-}UF\text{-}CMA}}(\mathcal{A}) \to 1\right]$$

is negligible in λ*, where* $\mathsf{Exp}_{\mathsf{MS}}^{\mathsf{MS\text{-}UF\text{-}CMA}}(\mathcal{A})$ *is described in Fig. 2.*

3 DS_2: Two-round n-out-of-n Signing from Module-LWE and Module-SIS

3.1 Protocol specification and overview

This section presents our main construction: provably secure two-round n-out-of-n protocol $\mathsf{DS}_2 = (\mathsf{Setup}, (\mathsf{Gen}_j)_{j\in[n]}, (\mathsf{Sign}_j)_{j\in[n]}, \mathsf{Ver})$, formally specified in Fig. 5. As mentioned in Section 2.5 all players have the same role and hence we only present n-th player's behavior. The protocol is built on top of additively homomorphic trapdoor commitment scheme TCOM with uniform key (see Definition 4 and Appendix B.1 for the formal definitions), and we will describe concrete instances of TCOM later in Section 4. We go over high-level ideas for each step below.

Parameter setup. We assume that a trusted party invokes $\mathsf{DS}_2.\mathsf{Setup}(1^\lambda)$ that outputs a set of public parameters described in Table 1 as well as the parameter for commitment scheme cpp (which is obtained by internally invoking $\mathsf{TCOM.CSetup}(1^\lambda)$). Most parameters commonly appear in the literature about the Fiat–Shamir with aborts paradigm (e.g. [DLL+18, Lyu12]) and we therefore omit the details here. The bit length l_1 and l_2 should be sufficiently long for the random oracle commitments to be secure. The only additional parameters are B_n and M_n, which we describe below in Section 3.2.

Fig. 5. Distributed n-out-of-n signature scheme.

Protocol $\mathsf{DS}_2.\mathsf{Gen}_n(pp)$

The protocol is parameterized by public parameters described in Table 1 and relies on the random oracles $\mathsf{H}_1 : \{0,1\}^* \to \{0,1\}^{l_1}$ and $\mathsf{H}_2 : \{0,1\}^* \to \{0,1\}^{l_2}$.

Matrix Generation

1. Sample a random matrix share $\mathbf{A}_n \leftarrow\$ R_q^{k\times\ell}$ and generate a random oracle commitment $g_n \leftarrow \mathsf{H}_1(\mathbf{A}_n, n)$. Send out g_n.
2. Upon receiving g_j for all $j \in [n-1]$ send out $\mathbf{A}_n$.
3. Upon receiving $\mathbf{A}_j$ for all $j \in [n-1]$:
   a. If $\mathsf{H}_1(\mathbf{A}_j, j) \neq g_j$ for some j then send out ABORT.
   b. Otherwise set public random matrix $\bar{\mathbf{A}} := [\mathbf{A}|\mathbf{I}] \in R_q^{k\times(\ell+k)}$, where $\mathbf{A} := \sum_{j\in[n]} \mathbf{A}_j$.

Key Pair Generation

1. Sample a secret key share $\mathbf{s}_n \leftarrow\$ S_\eta^{\ell+k}$ and compute a public key share $\mathbf{t}_n := \bar{\mathbf{A}}\mathbf{s}_n$, respectively, and generate a random oracle commitment $g'_n \leftarrow \mathsf{H}_2(\mathbf{t}_n, n)$. Send out g'_n.
2. Upon receiving g'_j for all $j \in [n-1]$ send out $\mathbf{t}_n$.
3. Upon receiving $\mathbf{t}_j$ for all $j \in [n-1]$:
   a. If $\mathsf{H}_2(\mathbf{t}_j, j) \neq g'_j$ for some j then send out ABORT.
   b. Otherwise set a combined public key $\mathbf{t} := \sum_{j\in[n]} \mathbf{t}_j$

If the protocol does not abort, P_n obtains $(sk_n, pk) = (\mathbf{s}_n, (\bar{\mathbf{A}}, \mathbf{t}))$ as local output.

Protocol $\mathsf{DS}_2.\mathsf{Sign}_n(sid, sk_n, pk, \mu)$

The protocol is parameterized by public parameters described in Table 1 and relies on the random oracles $\mathsf{H}_0 : \{0,1\}^* \to C$ and $\mathsf{H}_3 : \{0,1\}^* \to S_{ck}$. The protocol assumes that $\mathsf{DS}_2.\mathsf{Gen}_n(pp)$ has been previously invoked. If a party halts with ABORT at any point, then all $\mathsf{Sign}_n(sid, sk_n, pk, \mu)$ executions are aborted.

Inputs

1. P_n receives a unique session ID sid, $sk_n = \mathbf{s}_n$, $pk = (\bar{\mathbf{A}}, \mathbf{t})$ and message $\mu \in \{0,1\}^*$ as input.
2. P_n verifies that sid has not been used before (if it has been, the protocol is not executed).
3. P_n locally computes a per-message commitment key $ck \leftarrow \mathsf{H}_3(\mu, \mathbf{t})$.

Signature Generation P_n works as follows:

1. Compute the first message as follows.
   a. Sample $\mathbf{y}_n \leftarrow\$ D_s^{\ell+k}$ and compute $\mathbf{w}_n := \bar{\mathbf{A}}\mathbf{y}_n$.
   b. Compute $com_n \leftarrow \mathsf{Commit}_{ck}(\mathbf{w}_n; r_n)$ with $r_n \leftarrow\$ D(S_r)$.
   c. Send out com_n.
2. Upon receiving com_j for all $j \in [n-1]$ compute the signature share as follows.
   a. Set $com := \sum_{j\in[n]} com_j$.
   b. Derive a challenge $c \leftarrow \mathsf{H}_0(com, \mu, \mathbf{t})$.
   c. Computes a signature share $\mathbf{z}_n := c\mathbf{s}_n + \mathbf{y}_n$.
   d. Run the rejection sampling on input $(c\mathbf{s}_n, \mathbf{z}_n)$, i.e., with probability
   $$\min\left(1, D_s^{\ell+k}(\mathbf{z}_n)/(M \cdot D_{c\mathbf{s}_n,s}^{\ell+k}(\mathbf{z}_n))\right)$$
   send out $(\mathbf{z}_n, r_n)$; otherwise send out RESTART and go to 1.
3. Upon receiving RESTART from some party go to 1. Otherwise upon receiving $(\mathbf{z}_j, r_j)$ for all $j \in [n-1]$ compute the combined signature as follows
   a. For each $j \in [n-1]$ reconstruct $\mathbf{w}_j := \bar{\mathbf{A}}\mathbf{z}_j - c\mathbf{t}_j$ and validate the signature share:
   $$\|\mathbf{z}_j\|_2 \leq B \quad \text{and} \quad \mathsf{Open}_{ck}(com_j, r_j, \mathbf{w}_j) = 1.$$
   If the check fails for some j then send out ABORT.
   b. Compute $\mathbf{z} := \sum_{j\in[n]} \mathbf{z}_j$ and $r := \sum_{j\in[n]} r_j$.

If the protocol does not abort, P_n obtains a signature $(com, \mathbf{z}, r)$ as local output.

Algorithm $\mathsf{DS}_2.\mathsf{Ver}(com, \mathbf{z}, r, \mu, pk)$

Upon receiving a message μ, signature $(com, \mathbf{z}, r)$, and combined public key $pk = (\bar{\mathbf{A}}, \mathbf{t})$, generate a commitment key $ck \leftarrow \mathsf{H}_3(\mu, \mathbf{t})$, derive a challenge $c \leftarrow \mathsf{H}_0(com, \mu, \mathbf{t})$ and reconstruct $\mathbf{w} := \bar{\mathbf{A}}\mathbf{z} - c\mathbf{t}$. Then accept if $\|\mathbf{z}\|_2 \leq B_n$ and $\mathsf{Open}_{ck}(com, r, \mathbf{w}) = 1$.

Parameter	Description
n	Number of parties
N	A power of two defining the degree of $f(X)$
$f(X) = X^N + 1$	The $2N$-th cyclotomic polynomial
q	Prime modulus
$R = \mathbb{Z}[X]/(f(X))$	Cyclotomic ring
$R_q = \mathbb{Z}_q[X]/(f(X))$	Ring
k	The height of random matrices $\mathbf{A}$
ℓ	The width of random matrices $\mathbf{A}$
γ	Parameter defining the tail-cut bound
$B = \gamma\sigma\sqrt{N(\ell+k)}$	The maximum L^2-norm of signature share $\mathbf{z}_j \in R^{\ell+k}$ for $j \in [n]$
$B_n = \sqrt{n}B$	The maximum L^2-norm of combined signature $\mathbf{z} \in R^{\ell+k}$
κ	The maximum L^1-norm of challenge vector c
$C = \left\{c \in R : \|c\|_\infty = 1 \wedge \|c\|_1 = \kappa\right\}$	Challenge space where $\lvert C\rvert = \binom{N}{\kappa}2^\kappa$
$S_\eta = \left\{x \in R : \|x\|_\infty \leq \eta\right\}$	Set of small secrets
$T = \kappa\eta\sqrt{N(\ell+k)}$	Chosen such that Lemma 4 holds
α	Parameter defining σ and M
$\sigma = s/\sqrt{2\pi} = \alpha T$	Standard deviation of the Gaussian distribution
$t = \omega(\sqrt{\log(mN)}) \wedge t = o(\log(mN))$	Parameter defining M such that Lemma 3 holds
$M = e^{t/\alpha + 1/(2\alpha^2)}$	The expected number of restarts until a single party can proceed
$M_n = M^n$	The expected number of restarts until all n parties proceed simultaneously
cpp	Parameters for commitment scheme honestly generated with CSetup
l_1, l_2, l_4	Output bit lengths of random oracles $\mathsf{H}_1, \mathsf{H}_2$ and H_4

Table 1. Parameters for our distributed signature protocols.

Key generation. The key generation $\mathsf{DS}_2.\mathsf{Gen}_n$ essentially follows the approach by Nicolosi et al. [NKDM03] for two-party Schnorr signing. Upon receiving public parameters, all participants first interactively generate a random matrix $\mathbf{A} \in R_q^{k\times\ell}$, a part of Dilithium-G public key. This can be securely done with simple random oracle commitments[4]; as long as there is at least one honest party sampling a matrix share correctly, the resulting combined matrix is guaranteed to follow the uniform distribution. For the exact same reason, the exchange of public key shares is done with random oracle. This way, we can prevent the adversary from choosing some malicious public key share depending on the honest party's share (the so-called *rogue key attack* [MOR01]). Furthermore, the party's index j is concatenated with the values to be hashed for the sake of "domain separation" [BDG20]. This way, we prevent rushing adversaries from simply sending back the hash coming from the honest party and claiming that they know the preimage after seeing the honest party's opening.

Signature generation. The first crucial step of $\mathsf{DS}_2.\mathsf{Sign}_n$ in Fig. 5 is commitment key generation at **Inputs** 3; in fact, if instead some fixed key ck was used for all signing attempts, one could come up with a sub-exponential attack that outputs a valid forgery with respect to the joint public key $\mathbf{t}$. In Appendix E we sketch a variant of the concurrent attack due to Drijvers et al. [DEF+19]. The original attack was against two-round Schnorr multi-signatures including BCJ scheme [BCJ08], but due to the very similar structure of FSwA-based lattice signatures an attack would become feasible against a fixed-key variant of DS_2. This motivates us to derive a message-dependent commitment key, following Drivers et al.'s mBCJ scheme.

Then the signing protocol starts by exchanging the first "commit" messages of Σ-protocol, from which all parties derive a single joint challenge $c \in C$ via a random oracle. As we discussed earlier no participants are allowed to reveal $\mathbf{w}_j$ until the rejection sampling phase, and instead they send its commitment com_j, which is to be opened only if the signature share $\mathbf{z}_j$ passes the rejection sampling. Finally, the com_j's and r_j's are added together in a meaningful way, thanks to the homomorphic property of commitment scheme.

Verification and correctness. Thanks to the linearity of underlying scheme and homomorphism of the commitment, the verifier only needs to validate the sum of signature shares, commitments and

[4] We remark that the "commitments" generated by H_1 and H_2 in Fig. 5 are not randomized, and therefore they are not hiding. In our protocol, however, all committed values have high min-entropy and this is indeed sufficient for the security proof to hold. Alternatively, one could cheaply turn them into full-fledged secure and extractable commitments by additionally hashing random strings that are to be sent out during the opening phase [Pas03].

randomness. Here the Euclidean-norm bound B_n is set according to Lemma 1; if all parties honestly follow the protocol then the sum of n Gaussian shares is only $\sqrt{n}$ times larger (while if we employed the plain Dilithium as a base scheme then the bound would grow almost linearly). Hence together with the tail-cut bound of Lemma 2 it is indeed sufficient to set $B_n = \sqrt{n}B$ for the correctness to hold with overwhelming probability. To guarantee perfect correctness, the bound check can be also done during the signing protocol so that it simply restarts when the generated signature is too large (which of course only happens with negligible probability and shouldn't matter in practice).

3.2 Asymptotic efficiency analysis

Number of aborts and signature size. As indicated in Table 1 the probability that all participants simultaneously proceed is $1/M_n = 1/M^n$, where $1/M$ is the probability that each party asks to proceed. To make M_n reasonably small, say $M_n = 3$, we should set $\alpha \geq 11n$ [DLL+18], leading to $\sigma \geq 11nT$. This already increases the bound B of each signature share linearly compared to a non-distributed signature like Dilithium-G. In addition, we should set the bound B_n for combined signature to $\sqrt{n}B$ for the correctness to hold, and thus the SIS solution that we find in the security reduction grows by a factor of $n^{3/2}$.

This translates to a signature size increase of a factor of roughly[5] $O(\log n)$, so the scaling in terms of the number of parties is reasonable. In addition, when using the trapdoor commitment scheme of Section 4, one can substantially reduce the signature size by using the common Fiat–Shamir trick of expressing the signature as $(c, \mathbf{z}, r)$ instead of $(com, \mathbf{z}, r)$, and carrying out the verification by first re-computing the commitment using the randomness r, and then checking the consistency of the challenge: $c \overset{?}{=} \mathsf{H}_0(com, \mu, \mathbf{t})$. This keeps signature size close to the original Dilithium-G, despite the relatively large size of commitments.

We expect that a number of further optimizations are possible to improve the efficiency of this protocol in both asymptotic and concrete terms (e.g., by relying on stronger assumptions like (Mod-)NTRU), although this is left for further work. Accordingly, we also leave for further work the question of providing concrete parameters for the protocol, since the methodology for setting parameters is currently a moving target (e.g., the original parameters for Dilithium-G are not considered up-to-date), there is arguably no good point of comparison in the literature (in particular, no previous lattice-based two-round protocol), and again, a concrete instantiation would likely rely on stronger assumptions to achieve better efficiency anyway.

Round complexity. If this protocol is used as is, it only outputs a signature after the three rounds with probability $1/M_n$ (which is 1/3 with the parameters above). As a result, to effectively compute a signature, it has to be repeated M_n times on average, and so the expected number of rounds is in fact larger than 2 ($2M_n = 6$ in this case). One can of course adjust the parameters to reduce M_n to any constant greater than 1, or even to $1 + o(1)$ by picking e.g. $\alpha = \Theta(n^{1+\epsilon})$; this results in an expected number of rounds arbitrarily close to 2. Alternatively, one can keep a 2-round protocol while ensuring that the parties output a signature with overwhelming probability, simply by running sufficiently many parallel executions of the protocol at once: $\lambda \cdot \log \frac{M_n}{M_n - 1}$ parallel executions suffice if λ is the security parameter. The same remark applies analogously to the three-round protocol DS_3 of Appendix F.

3.3 Security

The formal security claim for our DS_2 protocol is given below.

Theorem 1. *Suppose the trapdoor commitment scheme* TCOM *is secure, additively homomorphic and has uniform keys. For any probabilistic polynomial-time adversary* $\mathcal{A}$ *that initiates a single key generation protocol by querying* $\mathcal{O}_n^{\mathsf{DS}_2}$ *with* $sid = 0$*, initiates* Q_s *signature generation protocols by querying* $\mathcal{O}_n^{\mathsf{DS}_2}$ *with* $sid \neq 0$*, and makes* Q_h *queries to the random oracle* $\mathsf{H}_0, \mathsf{H}_1, \mathsf{H}_2, \mathsf{H}_3$*, the protocol* DS_2 *of Fig. 5 is* $\mathsf{DS\text{-}UF\text{-}CMA}$ *secure under* $\mathsf{MSIS}_{q,k,\ell+1,\beta}$ *and* $\mathsf{MLWE}_{q,k,\ell,\eta}$ *assumptions, where* $\beta = 2\sqrt{B_n^2 + \kappa}$.

[5] To be more precise, since the verification bound scales as $n^{3/2}$, one should also increase q by the same bound to avoid arithmetic overflow. This makes the MSIS problem harder, but the MLWE easier if the dimension is kept unchanged. To keep the same security level, one should therefore also increase N by a factor of $1 + O(\frac{\log n}{\log q_0})$ where q_0 is the value of q in the single-user setting. Therefore, one could in principle argue that signature size actually scales as $O(\log^2 n)$. However, one typically chooses $q_0 > 2^{20}$, and therefore even in settings with billions of parties, $\frac{\log n}{\log q_0} < 2$. Thus, one can effectively regard N as independent of n.

We give a sketch of the security proof. The full proof with concrete security bound is given in Appendix C. We also remark that its multi-signature variant MS_2 (Appendix D) can be proven secure relying on essentially the same idea. We show that given any efficient adversary $\mathcal{A}$ that creates a valid forgery with non-negligible probability, one can break either $\mathsf{MSIS}_{q,k,\ell+1,\beta}$ assumption or computational binding of TCOM.

Key generation simulation. For the key generation phase, since the public key share of the honest signer $\mathbf{t}_n$ is indistinguishable from the vector sampled from R_q^k uniformly at random due to $\mathsf{MLWE}_{q,k,\ell,\eta}$ assumption, the honest party oracle simulator can replace $\mathbf{t}_n$ with such a vector. Therefore, the distribution of combined public key $\mathbf{t} = \sum_{j\in[n]} \mathbf{t}_j$ is also indistinguishable from the uniform distribution. Thanks to the random oracle commitment, after the adversary has submitted g'_j for each $j \in [n-1]$ one can extract the adversary's public key share $\mathbf{t}_j$, with which the simulator sets its share a posteriori $\mathbf{t}_n := \mathbf{t} - \sum_{j\in[n-1]} \mathbf{t}_j$ and programs the random oracle accordingly $\mathsf{H}_2(\mathbf{t}_n, n) := g'_n$. Using the same argument, one can set a random matrix share $\mathbf{A}_n := \mathbf{A} - \sum_{j\in[n-1]} \mathbf{A}_j$ given a resulting random matrix $\mathbf{A} \leftarrow_\$ R_q^{k\times\ell}$. Now we can embed an instance of $\mathsf{MSIS}_{q,k,\ell+1,\beta}$, which is denoted as $[\mathbf{A}'|\mathbf{I}]$ with $\mathbf{A}' \leftarrow_\$ R_q^{k\times(\ell+1)}$. Due to the way we simulated the joint public key $(\mathbf{A}, \mathbf{t})$ is uniformly distributed in $R_q^{k\times\ell} \times R_q^k$, so replacing it with a $\mathsf{MSIS}_{q,k,\ell+1,\beta}$ instance doesn't change the view of adversary at all, if $\mathbf{A}'$ is regarded as $\mathbf{A}' = [\mathbf{A}|\mathbf{t}]$.

Signature generation simulation. The oracle simulation closely follows the one for mBCJ [DEF+19]. Concretely, the oracle simulator programs H_3 so that for each signing query it returns tck generated via $(\mathit{tck}, \mathit{td}) \leftarrow \mathsf{TCGen}(\mathit{cpp})$, and for the specific crucial query that is used to create a forgery it returns an actual commitment key $\mathit{ck} \leftarrow \mathsf{CGen}(\mathit{cpp})$, which has been received by the reduction algorithm as a problem instance of the binding game. This way, upon receiving signing queries the oracle simulator can send out a "fake" commitment $\mathit{com}_n \leftarrow \mathsf{TCommit}_{\mathit{tck}}(\mathit{td})$ at the first round, and then the known trapdoor td allows to later equivocate to a simulated first message of the Σ-protocol *after* the joint random challenge $c \in C$ has been derived; formally, it samples a simulated signature share $\mathbf{z}_n \leftarrow_\$ D_s^{\ell+k}$ and then derives randomness as $r_n \leftarrow \mathsf{Eqv}_{\mathit{tck}}(\mathit{td}, \mathit{com}_n, \mathbf{w}_n := \bar{\mathbf{A}}\mathbf{z}_n - c\mathbf{t}_n)$. On the other hand, when the reduction obtains two openings after applying the forking lemma it can indeed break the binding property with respect to a real commitment key ck.

Forking lemma. Our proof is relying on the forking lemma [PS00, BN06]. This is mainly because we instantiated the protocol with a trapdoor commitment, which inevitably implies that the binding is only computational. Hence to construct a reduction that breaks binding, we do have to make the adversary submit two valid openings for a single commitment under the same key, which seems to require some kind of rewinding technique. After applying the forking lemma, the adversary submits two forgeries with distinct challenges $c^* \neq \hat{c}^*$, with which we can indeed find a solution to $\mathsf{MSIS}_{q,k,\ell+1,\beta}$, or otherwise break computational binding wrt ck. Concretely, after invoking the forking lemma, we obtain two forgeries $(\mathit{com}^*, \mathbf{z}^*, r^*, \mu^*)$ and $(\hat{\mathit{com}}^*, \hat{\mathbf{z}}^*, \hat{r}^*, \hat{\mu}^*)$ such that $c^* = \mathsf{H}(\mathit{com}^*, \mu, \mathbf{t}) \neq \mathsf{H}(\hat{\mathit{com}}^*, \hat{\mu}^*, \mathbf{t}) = \hat{c}^*$, $\mathit{com}^* = \hat{\mathit{com}}^*$, $\mu^* = \hat{\mu}^*$, and $\mathsf{H}(\mu^*, \mathbf{t}) = \mathsf{H}(\hat{\mu}^*, \mathbf{t}) = \mathit{ck}$. Since both forgeries are verified, we have $\|\mathbf{z}^*\|_2 \leq B_n \wedge \|\hat{\mathbf{z}}^*\|_2 \leq B_n$, and

$$\mathsf{Open}_{\mathit{ck}}(\mathit{com}^*, r^*, \bar{\mathbf{A}}\mathbf{z}^* - c^*\mathbf{t}) = \mathsf{Open}_{\mathit{ck}}(\mathit{com}^*, \hat{r}^*, \bar{\mathbf{A}}\hat{\mathbf{z}}^* - \hat{c}^*\mathbf{t}) = 1.$$

If $\bar{\mathbf{A}}\mathbf{z}^* - c^*\mathbf{t} \neq \bar{\mathbf{A}}\hat{\mathbf{z}}^* - \hat{c}^*\mathbf{t}$ then it means that computational binding is broken wrt a commitment key ck. Suppose $\bar{\mathbf{A}}\mathbf{z}^* - c^*\mathbf{t} = \bar{\mathbf{A}}\hat{\mathbf{z}}^* - \hat{c}^*\mathbf{t}$. Rearranging it leads to

$$[\mathbf{A}|\mathbf{I}|\mathbf{t}] \begin{bmatrix} \mathbf{z}^* - \hat{\mathbf{z}}^* \\ \hat{c}^* - c^* \end{bmatrix} = \mathbf{0}.$$

Recalling that $[\mathbf{A}'|\mathbf{I}] = [\mathbf{A}|\mathbf{t}|\mathbf{I}]$ is an instance of $\mathsf{MSIS}_{q,k,\ell+1,\beta}$ problem, we have found a valid solution if $\beta = \sqrt{(2B_n)^2 + 4\kappa}$, since $\|\mathbf{z}^* - \hat{\mathbf{z}}^*\|_2 \leq 2B_n$ and $0 < \|\hat{c}^* - c^*\|_2 \leq \sqrt{4\kappa}$.

4 Lattice-Based Commitments

In this section, we describe possible constructions for the lattice-based commitment schemes used in our protocols. The three-round protocol DS_3 of Appendix F requires a statistically binding, computationally hiding homomorphic commitment scheme, whereas the two-round protocol DS_2 of Section 3 needs a statistically hiding *trapdoor* homomorphic commitment scheme. We show that both types of commitments can be obtained using the techniques of Baum et al. [BDL+18]. More precisely, the first type of commitment scheme is a simple variant of the scheme of [BDL+18], in a parameter range that ensures statistical instead of just computational binding. The fact that such a parameter choice is possible is folklore, and does in fact appear in an earlier version of [BDL+18], so we do not claim any novelty in that regard.

The construction of a lattice-based trapdoor commitment scheme does not seem to appear in the literature, but we show that it is again possible by combining [BDL+18] with Micciancio–Peikert style trapdoors [MP12]. To prevent statistical learning attacks on the trapdoor sampling, however, it is important to sample the randomness in the commitment according to a discrete Gaussian distribution, in contrast with Baum et al.'s original scheme.

4.1 Statistically Binding Commitment Scheme

We first describe a statistically binding commitment scheme from lattices. The scheme, described in Fig. 7, is a simple variant of the scheme from [BDL+18], that mainly differs in the choice of parameter regime: we choose parameters so as to make the underlying SIS problem vacuously hard, and hence the scheme statistically binding. Another minor change is the reliance on discrete Gaussian distributions, for somewhat more standard and compact LWE parameters. The correctness and security properties, as well as the constraints on parameters, are obtained as follows.

Correctness. By construction. We select the bound B as $\Omega(s \cdot \sqrt{m' \cdot N})$. By [MP12, Lemma 2.9], this ensures that the probability to retry in the committing algorithm is negligible.

Statistically binding. Suppose that an adversary can construct a commitment $\mathbf{f}$ on two distinct messages $x \neq x'$, with the associated randomness $\mathbf{r}, \mathbf{r}'$. Since $x \neq x'$, the correctness condition ensures that $\mathbf{r}$ and $\mathbf{r}'$ are distinct and of norm $\leq B$, and satisfy $\hat{\mathbf{A}}_1 \cdot (\mathbf{r} - \mathbf{r}') \equiv 0 \pmod{q}$. This means in particular that there are non zero elements in the Euclidean ball $B_{m'}(0, 2B)$ of radius $2B$ in $R_q^{m'}$ that map to $\mathbf{0}$ in R_q^m. But this happens with negligible probability on the choice of $\hat{\mathbf{A}}_1$ when $\left|B_{m'}(0,2B)\right|/q^{mN} = 2^{-\Omega(N)}$. Now $\left|B_{m'}(0,2B)\right| = o\big((2\pi e/m'N)^{m'N/2} \cdot (2B)^{m'N}\big)$. Hence, picking for example $m' = 2m$, we get:

$$\frac{\left|B_{m'}(0,2B)\right|}{q^{mN}} \ll \left(\frac{4\pi e \cdot B^2}{mNq}\right)^{mN},$$

and the condition is satisfied for example with $q > 8\pi e B^2/mN$.

Computationally hiding. The randomness $\mathbf{r}$ can be written in the form $\begin{bmatrix}\mathbf{r}_1 & \mathbf{r}_2 & \mathbf{s}\end{bmatrix}^T$ where $\mathbf{r}_1 \in R_q^m$, $\mathbf{r}_2 \in R_q^k$, $\mathbf{s} \in R_q^{m'-m-k}$ are all sampled from discrete Gaussians of parameter s. The commitment elements then become:

$$\mathbf{f}_1 = \mathbf{r}_1 + \hat{\mathbf{A}}_1' \cdot \begin{bmatrix}\mathbf{r}_2 \\ \mathbf{s}\end{bmatrix} \qquad\qquad \mathbf{f}_2 = \mathbf{r}_2 + \hat{\mathbf{A}}_2' \cdot \mathbf{s} + \mathbf{x},$$

and distinguishing those values from uniform are clearly instances of decision MLWE. Picking $k = m$, $m' = 2m$, $s = \Theta(\sqrt{mN})$, $B = \Theta(mN)$, $q = \Theta\big((mN)^{3/2}\big)$ yields a simple instatiation with essentially standard security parameters.

4.2 Trapdoor Commitment Scheme

We now turn to the construction of a trapdoor commitment scheme with suitable homomorphic properties for our purposes. Our proposed scheme is described in Fig. 6. It is presented as a commitment for a *single* ring element $x \in R_q$. It is straightforward to extend it to support a vector $\mathbf{x} \in R_q^k$, but the efficiency gain from doing so is limited compared to simply committing to each coefficient separately, so we omit the extension.

We briefly discuss the various correctness and security properties of the scheme, together with the constraints that the various parameters need to satisfy. In short, we need to pick the standard deviation of the coefficients of the trapdoor matrix $\mathbf{R}$ large enough to ensure that the trapdoor key is statistically close to a normal commitment key; then, the randomness $\mathbf{r}$ in commitments should have large enough standard deviation to make commitments statistically close to uniform (and in particular statistically hiding), and also be sampleable using the trapdoor. These are constraints on $\bar{s}$ and s respectively. Finally, the bound B for verification should be large enough to accomodate valid commitments, and small enough compared to q to still make the scheme computationally binding (which corresponds to the hardness of an underlying Ring-SIS problem). Let us now discuss the properties one by one.

Correctness. By construction. We select the bound B as $C \cdot s \cdot \sqrt{N}(\sqrt{\ell + 2w} + 1)$ where $C \approx 1/\sqrt{2\pi}$ is the constant in [MP12, Lemma 2.9]. By that lemma, this ensures that the probability to retry in the committing algorithm is negligible (and in particular, the distribution of $\mathbf{r}$ after the rejection sampling is statistically close to the original Gaussian).

Computationally binding. Suppose that an adversary can construct a commitment $\mathbf{f}$ on two distinct messages $x \neq x'$, with the associated randomness $\mathbf{r}, \mathbf{r}'$. Since $x \neq x'$, the correctness condition ensures

Protocol Lattice-based Commitment Scheme

$\mathsf{CSetup}(1^\lambda)$ takes a security parameter and outputs $cpp = (N, q, \bar{s}, s, B, \ell, w)$.

$\mathsf{CGen}(cpp)$ takes a commitment parameter and samples $\hat{a}_{1,1} \leftarrow\!\!\$\, R_q^\times$ (a uniform invertible element of R_q) and $\hat{a}_{1,j} \leftarrow\!\!\$\, R_q$ for $j = 2, \dots, \ell + 2w$, $\hat{a}_{2,j} \leftarrow\!\!\$\, R_q$ for $j = 3, \dots, \ell + 2w$. It then outputs:

$$\hat{\mathbf{A}} = \begin{bmatrix} \hat{a}_{1,1} & \hat{a}_{1,2} & \hat{a}_{1,3} & \cdots & \hat{a}_{1,\ell+2w} \\ 0 & 1 & \hat{a}_{2,3} & \cdots & \hat{a}_{2,\ell+2w} \end{bmatrix}$$

as ck.

$\mathsf{Commit}_{ck}(x)$ takes $x \in R_q$ and samples a discrete Gaussian vector of randomness $\mathbf{r} \leftarrow\!\!\$\, D_s^{\ell+2w}$. It then outputs

$$\mathbf{f} = \hat{\mathbf{A}} \cdot \mathbf{r} + \begin{bmatrix} 0 \\ x \end{bmatrix} \in R_q^2.$$

To ensure perfect correctness, retry unless $\|\mathbf{r}\|_2 \le B$.

$\mathsf{Open}_{ck}(\mathbf{f}, \mathbf{r}, x)$ takes commitments, randomness and message, and checks that

$$\mathbf{f} = \hat{\mathbf{A}} \cdot \mathbf{r} + \begin{bmatrix} 0 \\ x \end{bmatrix} \quad \text{and} \quad \|\mathbf{r}\|_2 \le B.$$

$\mathsf{TCGen}(cpp)$ takes a commitment parameter and samples $\bar{\mathbf{A}} \in R_q^{2\times\ell}$ of the form:

$$\bar{\mathbf{A}} = \begin{bmatrix} \bar{a}_{1,1} & \bar{a}_{1,2} & \bar{a}_{1,3} & \cdots & \bar{a}_{1,\ell} \\ 0 & 1 & \bar{a}_{2,3} & \cdots & \bar{a}_{2,\ell} \end{bmatrix}$$

where all the $\bar{a}_{i,j}$ are uniform in R_q, except $\bar{a}_{1,1}$ which is uniform in $R_q^\times$. It also samples $\mathbf{R} \leftarrow\!\!\$\, D_{\bar{s}}^{\ell\times 2w}$ with discrete Gaussian entries. It then outputs $\mathbf{R}$ as the trapdoor td and $\hat{\mathbf{A}} = \left[\bar{\mathbf{A}} | \mathbf{G} - \bar{\mathbf{A}}\mathbf{R}\right]$ as the commitment key tck, where $\mathbf{G}$ is given by:

$$\mathbf{G} = \begin{bmatrix} 1 & 2 & \cdots & 2^{w-1} & 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 & 1 & 2 & \cdots & 2^{w-1} \end{bmatrix} \in R^{2\times 2w}.$$

$\mathsf{TCommit}_{tck}(td)$ simply returns a uniformly random commitment $\mathbf{f} \leftarrow\!\!\$\, R_q^{2\times 1}$. There is no need to keep a state.

$\mathsf{Eqv}_{tck}(\mathbf{R}, \mathbf{f}, x)$ uses the trapdoor discrete Gaussian sampling algorithm of Micciancio–Peikert [MP12, Algorithm 3] (or faster variants such as the one described in [GM18]) to sample $\mathbf{r} \leftarrow\!\!\$\, D_{\Lambda_{\mathbf{u}}^\perp(\hat{\mathbf{A}}),s}$ according to the discrete Gaussian of parameter s supported on the lattice coset:

$$\Lambda_{\mathbf{u}}^\perp(\hat{\mathbf{A}}) = \left\{\mathbf{z} \in R^{\ell+2w} \;:\; \hat{\mathbf{A}} \cdot \mathbf{z} \equiv \mathbf{u} \pmod q\right\} \quad \text{where} \quad \mathbf{u} = \mathbf{f} - \begin{bmatrix} 0 \\ x \end{bmatrix}.$$

Fig. 6. Equivocable variant of the commitment from [BDL⁺18].

that $\mathbf{r}$ and $\mathbf{r}'$ are distinct and of norm $\le B$, and satisfy $\hat{\mathbf{A}}_1 \cdot (\mathbf{r} - \mathbf{r}') \equiv 0 \pmod q$ where $\hat{\mathbf{A}}_1$ is the first row of $\hat{\mathbf{A}}$. Therefore, the vector $\mathbf{z} = \mathbf{r} - \mathbf{r}'$ is a solution of the Ring-SIS problem with bound $2B$ associated with $\hat{\mathbf{A}}_1$ (or equivalently, to the $\mathsf{MSIS}_{q,1,\ell+2w-1,2B}$ problem), and finding such a solution is hard.

Note that since the first entry of $\hat{\mathbf{A}}_1$ is invertible, one can put it in the form $[\mathbf{A}|\mathbf{I}]$ without loss of generality to express it directly as an MSIS problem in the sense of Definition 2. It also reduces tightly to standard Ring-SIS, because a random row vector in $R_q^{\ell+2w}$ contains an invertible entry except with probability at most $(N/q)^{\ell+2w} = 1/N^{\Omega(\log N)}$, which is negligible.

Statistically hiding. It suffices to make sure that

$$\hat{\mathbf{A}} \cdot D_s^{\ell+2w} \approx_s U(R_q^2)$$

with high probability on the choice of $\hat{\mathbf{A}}$. This is addressed by [LPR13, Corollary 7.5], which shows that it suffices to pick $s > 2N \cdot q^{(2+2/N)/(\ell+2w)}$.

Indistinguishability of the trapdoor. To ensure that the commitment key $\hat{\mathbf{A}}$ generated by TCGen is indistinguishable from a regular commitment key, it suffices to ensure that $\bar{\mathbf{A}}\mathbf{R}$ is statistically close to uniform. Again by [LPR13, Corollary 7.5], this is guaranteed for $\bar{s} > 2N \cdot q^{(2+2/N)/\ell}$. By setting $\ell = w = \lceil \log_2 q \rceil$, we can thus pick $\bar{s} = \Theta(N)$.

Equivocability. It is clear that an $\mathbf{r}$ sampled according to the given lattice coset discrete Gaussian is distributed as in the regular commitment algorithm (up to the negligible statistical distance due to rejection sampling). The only constraint is thus on the Gaussian parameter that can be achieved by the trapdoor Gaussian sampling algorithm. By [MP12, §5.4], the constraint on s is as follows:

$$s \geq \|\mathbf{R}\| \cdot \omega(\sqrt{\log N})$$

where $\|\mathbf{R}\| \leq C \cdot \bar{s}\sqrt{N}(\sqrt{\ell} + \sqrt{2w} + 1)$ by [MP12, Lemma 2.9]. Thus, one can pick $s = \Theta(N^{3/2} \log^2 N)$.

From the previous paragraphs, we can in particular see that the trapdoor commitment satisfies the security requirements of Definition 4. Thus, to summarize, we have proved the following theorem.

Theorem 2. *The trapdoor commitment scheme of Fig. 6, with the following choice of parameters:*

$$\begin{array}{ccc} \bar{s} = \Theta(N) & s = \Theta(N^{3/2} \log^2 N) & B = \Theta(N^2 \log^3 N) \\ \ell = w = \lceil \log_2 q \rceil & q = N^{2+\varepsilon} & (\varepsilon > 0,\ q \text{ prime}). \end{array}$$

is a secure trapdoor commitment scheme assuming that the $\mathsf{MSIS}_{q,1,\ell+2w-1,2B}$ *problem is hard.*

Note that we did not strive for optimality in the parameter selection; a finer analysis is likely to lead to a more compact scheme.

Furthermore, although the commitment has a linear structure that gives it homomorphic features, we need to increase parameters slightly to support additive homomorphism: this is because the standard deviation of the sum of n randomness vectors $\mathbf{v}$ is $\sqrt{n}$ times larger. Therefore, B (and accordingly q) should be increased by a factor of $\sqrt{n}$ to accomodate for n-party additive homomorphism. For constant n, of course, this does not affect the asymptotic efficiency.

References

AF04. M. Abe and S. Fehr. Adaptively secure feldman VSS and applications to universally-composable threshold cryptography. In *CRYPTO 2004*, vol. 3152 of *LNCS*, pp. 317–334. Springer, Heidelberg, 2004.

AFLT16. M. Abdalla, P.-A. Fouque, V. Lyubashevsky, and M. Tibouchi. Tightly secure signatures from lossy identification schemes. *Journal of Cryptology*, 29(3):597–631, 2016.

BAA$^+$19. N. Bindel, S. Akleylek, E. Alkim, P. S. L. M. Barreto, J. Buchmann, E. Eaton, G. Gutoski, J. Kramer, P. Longa, H. Polat, J. E. Ricardini, and G. Zanon. qTESLA. Technical report, National Institute of Standards and Technology, 2019.

BBE$^+$18. G. Barthe, S. Belaïd, T. Espitau, P.-A. Fouque, B. Grégoire, M. Rossi, and M. Tibouchi. Masking the GLP lattice-based signature scheme at any order. In *EUROCRYPT 2018, Part II*, vol. 10821 of *LNCS*, pp. 354–384. Springer, Heidelberg, 2018.

BBE$^+$19. G. Barthe, S. Belaïd, T. Espitau, P.-A. Fouque, M. Rossi, and M. Tibouchi. GALACTICS: Gaussian sampling for lattice-based constant- time implementation of cryptographic signatures, revisited. In *ACM CCS 2019*, pp. 2147–2164. ACM Press, 2019.

BCJ08. A. Bagherzandi, J. H. Cheon, and S. Jarecki. Multisignatures secure under the discrete logarithm assumption and a generalized forking lemma. In *ACM CCS 2008*, pp. 449–458. ACM Press, 2008.

BCK$^+$14. F. Benhamouda, J. Camenisch, S. Krenn, V. Lyubashevsky, and G. Neven. Better zero-knowledge proofs for lattice encryption and their application to group signatures. In *ASIACRYPT 2014, Part I*, vol. 8873 of *LNCS*, pp. 551–572. Springer, Heidelberg, 2014.

BDG20. M. Bellare, H. Davis, and F. Günther. Separate your domains: NIST PQC KEMs, oracle cloning and read-only indifferentiability. In *EUROCRYPT 2020, Part II*, vol. 12106 of *LNCS*, pp. 3–32. Springer, Heidelberg, 2020.

BDL$^+$18. C. Baum, I. Damgård, V. Lyubashevsky, S. Oechsner, and C. Peikert. More efficient commitments from structured lattice assumptions. In *SCN 18*, vol. 11035 of *LNCS*, pp. 368–385. Springer, Heidelberg, 2018.

BGG$^+$18. D. Boneh, R. Gennaro, S. Goldfeder, A. Jain, S. Kim, P. M. R. Rasmussen, and A. Sahai. Threshold cryptosystems from threshold fully homomorphic encryption. In *CRYPTO 2018, Part I*, vol. 10991 of *LNCS*, pp. 565–596. Springer, Heidelberg, 2018.

BKLP15. F. Benhamouda, S. Krenn, V. Lyubashevsky, and K. Pietrzak. Efficient zero-knowledge proofs for commitments from learning with errors over rings. In *ESORICS 2015, Part I*, vol. 9326 of *LNCS*, pp. 305–325. Springer, Heidelberg, 2015.

BKP13. R. Bendlin, S. Krehbiel, and C. Peikert. How to share a lattice trapdoor: Threshold protocols for signatures and (H)IBE. In *ACNS 13*, vol. 7954 of *LNCS*, pp. 218–236. Springer, Heidelberg, 2013.

BLS19. J. Bootle, V. Lyubashevsky, and G. Seiler. Algebraic techniques for short(er) exact lattice-based zero-knowledge proofs. In *CRYPTO 2019, Part I*, vol. 11692 of *LNCS*, pp. 176–202. Springer, Heidelberg, 2019.

BN06. M. Bellare and G. Neven. Multi-signatures in the plain public-key model and a general forking lemma. In *ACM CCS 2006*, pp. 390–399. ACM Press, 2006.

BS13. S. Bettaieb and J. Schrek. Improved lattice-based threshold ring signature scheme. In *Post-Quantum Cryptography - 5th International Workshop, PQCrypto 2013*, pp. 34–51. Springer, Heidelberg, 2013.

BS16. R. E. Bansarkhani and J. Sturm. An efficient lattice-based multisignature scheme with applications to bitcoins. In *CANS 16*, vol. 10052 of *LNCS*, pp. 140–155. Springer, Heidelberg, 2016.

CCL$^+$19. G. Castagnos, D. Catalano, F. Laguillaumie, F. Savasta, and I. Tucker. Two-party ECDSA from hash proof systems and efficient instantiations. In *CRYPTO 2019, Part III*, vol. 11694 of *LNCS*, pp. 191–221. Springer, Heidelberg, 2019.

CCL$^+$20. G. Castagnos, D. Catalano, F. Laguillaumie, F. Savasta, and I. Tucker. Bandwidth-efficient threshold EC-DSA. In *PKC 2020, Part II*, vol. 12111 of *LNCS*, pp. 266–296. Springer, Heidelberg, 2020.

CHKP10. D. Cash, D. Hofheinz, E. Kiltz, and C. Peikert. Bonsai trees, or how to delegate a lattice basis. In *EUROCRYPT 2010*, vol. 6110 of *LNCS*, pp. 523–552. Springer, Heidelberg, 2010.

CK16. R. Choi and K. Kim. Lattice-based multi-signature with linear homomorphism. In *2016 Symposium on Cryptography and Information Security (SCIS 2016)*, 2016.

CLRS10. P. Cayrel, R. Lindner, M. Rückert, and R. Silva. A lattice-based threshold ring signature scheme. In *LATINCRYPT 2010*, vol. 6212 of *LNCS*, pp. 255–272. Springer, 2010.

CMP20. R. Canetti, N. Makriyannis, and U. Peled. Uc non-interactive, proactive, threshold ecdsa. Cryptology ePrint Archive, Report 2020/492, 2020. https://eprint.iacr.org/2020/492.

COSV17a. M. Ciampi, R. Ostrovsky, L. Siniscalchi, and I. Visconti. Delayed-input non-malleable zero knowledge and multi-party coin tossing in four rounds. In *TCC 2017, Part I*, vol. 10677 of *LNCS*, pp. 711–742. Springer, Heidelberg, 2017.

COSV17b. M. Ciampi, R. Ostrovsky, L. Siniscalchi, and I. Visconti. Four-round concurrent non-malleable commitments from one-way functions. In *CRYPTO 2017, Part II*, vol. 10402 of *LNCS*, pp. 127–157. Springer, Heidelberg, 2017.

CPS$^+$16. M. Ciampi, G. Persiano, A. Scafuro, L. Siniscalchi, and I. Visconti. Improved OR-composition of sigma-protocols. In *TCC 2016-A, Part II*, vol. 9563 of *LNCS*, pp. 112–141. Springer, Heidelberg, 2016.

CS19. D. Cozzo and N. P. Smart. Sharing the LUOV: Threshold post-quantum signatures. In *17th IMA International Conference on Cryptography and Coding*, vol. 11929 of *LNCS*, pp. 128–153. Springer, Heidelberg, 2019.

Dam00. I. Damgård. Efficient concurrent zero-knowledge in the auxiliary string model. In *EUROCRYPT 2000*, vol. 1807 of *LNCS*, pp. 418–430. Springer, Heidelberg, 2000.

DDLL13. L. Ducas, A. Durmus, T. Lepoint, and V. Lyubashevsky. Lattice signatures and bimodal Gaussians. In *CRYPTO 2013, Part I*, vol. 8042 of *LNCS*, pp. 40–56. Springer, Heidelberg, 2013.

DEF$^+$19. M. Drijvers, K. Edalatnejad, B. Ford, E. Kiltz, J. Loss, G. Neven, and I. Stepanovs. On the security of two-round multi-signatures. In *2019 IEEE Symposium on Security and Privacy*, pp. 1084–1101. IEEE Computer Society Press, 2019.

DHSS20. Y. Doröz, J. Hoffstein, J. H. Silverman, and B. Sunar. Mmsat: A scheme for multimessage multiuser signature aggregation. Cryptology ePrint Archive, Report 2020/520, 2020. https://eprint.iacr.org/2020/520.

DJN$^+$20. I. Damgård, T. P. Jakobsen, J. B. Nielsen, J. I. Pagter, and M. B. Østergård. Fast threshold ecdsa with honest majority. Cryptology ePrint Archive, Report 2020/501, 2020. https://eprint.iacr.org/2020/501.

DKLs18. J. Doerner, Y. Kondi, E. Lee, and a. shelat. Secure two-party threshold ECDSA from ECDSA assumptions. In *2018 IEEE Symposium on Security and Privacy*, pp. 980–997. IEEE Computer Society Press, 2018.

DKLs19. J. Doerner, Y. Kondi, E. Lee, and a. shelat. Threshold ECDSA from ECDSA assumptions: The multiparty case. In *2019 IEEE Symposium on Security and Privacy*, pp. 1051–1066. IEEE Computer Society Press, 2019.

DKO$^+$19. A. Dalskov, M. Keller, C. Orlandi, K. Shrishak, and H. Shulman. Securing dnssec keys via threshold ecdsa from generic mpc. Cryptology ePrint Archive, Report 2019/889, 2019. https://eprint.iacr.org/2019/889.

DLL$^+$18. L. Ducas, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, and D. Stehlé. Crystals–dilithium: Digital signatures from module lattices. 2018.

dLS18. R. del Pino, V. Lyubashevsky, and G. Seiler. Lattice-based group signatures and zero-knowledge proofs of automorphism stability. In *ACM CCS 2018*, pp. 574–591. ACM Press, 2018.

DM14. L. Ducas and D. Micciancio. Improved short lattice signatures in the standard model. In *CRYPTO 2014, Part I*, vol. 8616 of *LNCS*, pp. 335–352. Springer, Heidelberg, 2014.

EEE20. M. F. Esgin, O. Ersoy, and Z. Erkin. Post-quantum adaptor signatures and payment channel networks. Cryptology ePrint Archive, Report 2020/845, 2020. https://eprint.iacr.org/2020/845.

ESLL19. M. F. Esgin, R. Steinfeld, J. K. Liu, and D. Liu. Lattice-based zero-knowledge proofs: New techniques for shorter and faster constructions and applications. In *CRYPTO 2019, Part I*, vol. 11692 of *LNCS*, pp. 115–146. Springer, Heidelberg, 2019.

ESS$^+$19. M. F. Esgin, R. Steinfeld, A. Sakzad, J. K. Liu, and D. Liu. Short lattice-based one-out-of-many proofs and applications to ring signatures. In *ACNS 19*, vol. 11464 of *LNCS*, pp. 67–88. Springer, Heidelberg, 2019.

FH19. M. Fukumitsu and S. Hasegawa. A tightly-secure lattice-based multisignature. In *APKC@AsiaCCS 2019*, pp. 3–11. ACM, 2019.

FH20. M. Fukumitsu and S. Hasegawa. A lattice-based provably secure multisignature scheme in quantum random oracle model. In *ProvSec 2020*, LNCS, pp. xxx–xxx. Springer, 2020.

GG18. R. Gennaro and S. Goldfeder. Fast multiparty threshold ECDSA with fast trustless setup. In *ACM CCS 2018*, pp. 1179–1194. ACM Press, 2018.

GG20. R. Gennaro and S. Goldfeder. One round threshold ecdsa with identifiable abort. Cryptology ePrint Archive, Report 2020/540, 2020. https://eprint.iacr.org/2020/540.

GGN16. R. Gennaro, S. Goldfeder, and A. Narayanan. Threshold-optimal DSA/ECDSA signatures and an application to bitcoin wallet security. In *ACNS 16*, vol. 9696 of *LNCS*, pp. 156–174. Springer, Heidelberg, 2016.

GJKR07. R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Secure distributed key generation for discrete-log based cryptosystems. *Journal of Cryptology*, 20(1):51–83, 2007.

GKSS20. A. Gagol, J. Kula, D. Straszak, and M. Swietek. Threshold ecdsa for decentralized asset custody. Cryptology ePrint Archive, Report 2020/498, 2020. https://eprint.iacr.org/2020/498.

GLP12. T. Güneysu, V. Lyubashevsky, and T. Pöppelmann. Practical lattice-based cryptography: A signature scheme for embedded systems. In *CHES 2012*, vol. 7428 of *LNCS*, pp. 530–547. Springer, Heidelberg, 2012.

GM18. N. Genise and D. Micciancio. Faster Gaussian sampling for trapdoor lattices with arbitrary modulus. In *EUROCRYPT 2018, Part I*, vol. 10820 of *LNCS*, pp. 174–203. Springer, Heidelberg, 2018.

GPV08. C. Gentry, C. Peikert, and V. Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *40th ACM STOC*, pp. 197–206. ACM Press, 2008.

GSW13. C. Gentry, A. Sahai, and B. Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *CRYPTO 2013, Part I*, vol. 8042 of *LNCS*, pp. 75–92. Springer, Heidelberg, 2013.

GVW15. S. Gorbunov, V. Vaikuntanathan, and D. Wichs. Leveled fully homomorphic signatures from standard lattices. In *47th ACM STOC*, pp. 469–477. ACM Press, 2015.

HJ10. N. Howgrave-Graham and A. Joux. New generic algorithms for hard knapsacks. In *EUROCRYPT 2010*, vol. 6110 of *LNCS*, pp. 235–256. Springer, Heidelberg, 2010.

KD20. M. Kansal and R. Dutta. Round optimal secure multisignature schemes from lattice with public key aggregation and signature compression. In *AFRICACRYPT 20*, vol. 12174 of *LNCS*, pp. 281–300. Springer, Heidelberg, 2020.

KG20. C. Komlo and I. Goldberg. FROST: Flexible round-optimized schnorr threshold signatures. Cryptology ePrint Archive, Report 2020/852, 2020. https://eprint.iacr.org/2020/852.

KLS18. E. Kiltz, V. Lyubashevsky, and C. Schaffner. A concrete treatment of Fiat-Shamir signatures in the quantum random-oracle model. In *EUROCRYPT 2018, Part III*, vol. 10822 of *LNCS*, pp. 552–586. Springer, Heidelberg, 2018.

LDK$^+$19. V. Lyubashevsky, L. Ducas, E. Kiltz, T. Lepoint, P. Schwabe, G. Seiler, and D. Stehlé. CRYSTALS-DILITHIUM. Technical report, National Institute of Standards and Technology, 2019.

Lin17. Y. Lindell. Fast secure two-party ECDSA signing. In *CRYPTO 2017, Part II*, vol. 10402 of *LNCS*, pp. 613–644. Springer, Heidelberg, 2017.

LN18. Y. Lindell and A. Nof. Fast secure multiparty ECDSA with practical distributed key generation and applications to cryptocurrency custody. In *ACM CCS 2018*, pp. 1837–1854. ACM Press, 2018.

LNTW19. B. Libert, K. Nguyen, B. H. M. Tan, and H. Wang. Zero-knowledge elementary databases with more expressive queries. In *PKC 2019, Part I*, vol. 11442 of *LNCS*, pp. 255–285. Springer, Heidelberg, 2019.

LPR13. V. Lyubashevsky, C. Peikert, and O. Regev. A toolkit for ring-LWE cryptography. In *EUROCRYPT 2013*, vol. 7881 of *LNCS*, pp. 35–54. Springer, Heidelberg, 2013.

LS18. V. Lyubashevsky and G. Seiler. Short, invertible elements in partially splitting cyclotomic rings and applications to lattice-based zero-knowledge proofs. In *EUROCRYPT 2018, Part I*, vol. 10820 of *LNCS*, pp. 204–224. Springer, Heidelberg, 2018.

LTT20. Z.-Y. Liu, Y.-F. Tseng, and R. Tso. Cryptanalysis of a round optimal lattice-based multisignature scheme. Cryptology ePrint Archive, Report 2020/1172, 2020. https://eprint.iacr.org/2020/1172.

Lyu09. V. Lyubashevsky. Fiat-Shamir with aborts: Applications to lattice and factoring-based signatures. In *ASIACRYPT 2009*, vol. 5912 of *LNCS*, pp. 598–616. Springer, Heidelberg, 2009.

Lyu12. V. Lyubashevsky. Lattice signatures without trapdoors. In *EUROCRYPT 2012*, vol. 7237 of *LNCS*, pp. 738–755. Springer, Heidelberg, 2012.

Lyu19. V. Lyubashevsky. Lattice-based zero-knowledge and applications. CIS 2019, 2019. https://crypto.sjtu.edu.cn/cis2019/slides/Vadim.pdf.

MJ19. C. Ma and M. Jiang. Practical lattice-based multisignature schemes for blockchains. *IEEE Access*, 7:179765–179778, 2019.

MOR01. S. Micali, K. Ohta, and L. Reyzin. Accountable-subgroup multisignatures: Extended abstract. In *ACM CCS 2001*, pp. 245–254. ACM Press, 2001.

MP12. D. Micciancio and C. Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In *EUROCRYPT 2012*, vol. 7237 of *LNCS*, pp. 700–718. Springer, Heidelberg, 2012.

MP13. D. Micciancio and C. Peikert. Hardness of SIS and LWE with small parameters. In *CRYPTO 2013, Part I*, vol. 8042 of *LNCS*, pp. 21–39. Springer, Heidelberg, 2013.

MPSW19. G. Maxwell, A. Poelstra, Y. Seurin, and P. Wuille. Simple schnorr multi-signatures with applications to bitcoin. *Des. Codes Cryptogr.*, 87(9):2139–2164, 2019.

MWLD10. C. Ma, J. Weng, Y. Li, and R. H. Deng. Efficient discrete logarithm based multi-signature scheme in the plain public key model. *Des. Codes Cryptogr.*, 54(2):121–133, 2010.

Ngu19. N. K. Nguyen. On the non-existence of short vectors in random module lattices. In *ASIACRYPT 2019, Part II*, vol. 11922 of *LNCS*, pp. 121–150. Springer, Heidelberg, 2019.

NKDM03. A. Nicolosi, M. N. Krohn, Y. Dodis, and D. Mazières. Proactive two-party signatures for user authentication. In *NDSS 2003*. The Internet Society, 2003.

NRS20. J. Nick, T. Ruffing, and Y. Seurin. MuSig2: Simple two-round schnorr multi-signatures. Cryptology ePrint Archive, Report 2020/1261, 2020. https://eprint.iacr.org/2020/1261.

NRSW20. J. Nick, T. Ruffing, Y. Seurin, and P. Wuille. MuSig-DN: Schnorr multi-signatures with verifiably deterministic nonces. Cryptology ePrint Archive, Report 2020/1057, 2020. https://eprint.iacr.org/2020/1057.

Pas03. R. Pass. On deniability in the common reference string and random oracle model. In *CRYPTO 2003*, vol. 2729 of *LNCS*, pp. 316–337. Springer, Heidelberg, 2003.

Ped92. T. P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO'91*, vol. 576 of *LNCS*, pp. 129–140. Springer, Heidelberg, 1992.

Pei10. C. Peikert. An efficient and parallel Gaussian sampler for lattices. In *CRYPTO 2010*, vol. 6223 of *LNCS*, pp. 80–97. Springer, Heidelberg, 2010.

PS00. D. Pointcheval and J. Stern. Security arguments for digital signatures and blind signatures. *Journal of Cryptology*, 13(3):361–396, 2000.

Sch90. C.-P. Schnorr. Efficient identification and signatures for smart cards. In *CRYPTO'89*, vol. 435 of *LNCS*, pp. 239–252. Springer, Heidelberg, 1990.

SS01. D. R. Stinson and R. Strobl. Provably secure distributed Schnorr signatures and a (t, n) threshold scheme for implicit certificates. In *ACISP 01*, vol. 2119 of *LNCS*, pp. 417–434. Springer, Heidelberg, 2001.

STV$^+$16. E. Syta, I. Tamas, D. Visher, D. I. Wolinsky, P. Jovanovic, L. Gasser, N. Gailly, I. Khoffi, and B. Ford. Keeping authorities "honest or bust" with decentralized witness cosigning. In *2016 IEEE Symposium on Security and Privacy*, pp. 526–545. IEEE Computer Society Press, 2016.

TE19. R. Toluee and T. Eghlidos. An efficient and secure ID-based multi-proxy multi-signature scheme based on lattice. Cryptology ePrint Archive, Report 2019/1031, 2019. https://eprint.iacr.org/2019/1031.

TLT19. R. Tso, Z. Liu, and Y. Tseng. Identity-based blind multisignature from lattices. *IEEE Access*, 7:182916–182923, 2019.

TSSK20. W. A. Torres, R. Steinfeld, A. Sakzad, and V. Kuchta. Post-quantum linkable ring signature enabling distributed authorised ring confidential transactions in blockchain. Cryptology ePrint Archive, Report 2020/1121, 2020. https://eprint.iacr.org/2020/1121.

Wag02. D. Wagner. A generalized birthday problem. In *CRYPTO 2002*, vol. 2442 of *LNCS*, pp. 288–303. Springer, Heidelberg, 2002.

YAZ$^+$19. R. Yang, M. H. Au, Z. Zhang, Q. Xu, Z. Yu, and W. Whyte. Efficient lattice-based zero-knowledge arguments with standard soundness: Construction and applications. In *CRYPTO 2019, Part I*, vol. 11692 of *LNCS*, pp. 147–175. Springer, Heidelberg, 2019.

Protocol Lattice-based statistically binding commitment

$\mathsf{CSetup}(1^\lambda)$ takes a security parameter and outputs $cpp = (q, N, k, m, m', \eta)$.

$\mathsf{CGen}(cpp)$ takes a commitment parameter and outputs ck consisting of $\hat{\mathbf{A}}_1 \in R_q^{m \times m'}$ and $\hat{\mathbf{A}}_2 \in R_q^{k \times m'}$.

$$\hat{\mathbf{A}}_1 = [\mathbf{I}_m | \hat{\mathbf{A}}_1'] \text{ where } \hat{\mathbf{A}}_1' \leftarrow\!\!\$\, R_q^{m \times (m'-m)}$$
$$\hat{\mathbf{A}}_2 = [\mathbf{0}^{k \times m} | \mathbf{I}_k | \hat{\mathbf{A}}_2'] \text{ where } \hat{\mathbf{A}}_2' \leftarrow\!\!\$\, R_q^{k \times (m'-m-k)}$$

$\mathsf{Commit}_{ck}(\mathbf{x})$ takes $\mathbf{x} \in R_q^k$, samples the randomness vector $\mathbf{r} \leftarrow\!\!\$\, D_s^{m'}$ and outputs

$$\begin{bmatrix} \mathbf{f}_1 \\ \mathbf{f}_2 \end{bmatrix} = \begin{bmatrix} \hat{\mathbf{A}}_1 \\ \hat{\mathbf{A}}_2 \end{bmatrix} \cdot \mathbf{r} + \begin{bmatrix} \mathbf{0}^m \\ \mathbf{x} \end{bmatrix}.$$

$\mathsf{Open}_{ck}(\mathbf{f}_1, \mathbf{f}_2, \mathbf{x}, \mathbf{r})$ checks that

$$\begin{bmatrix} \mathbf{f}_1 \\ \mathbf{f}_2 \end{bmatrix} = \begin{bmatrix} \hat{\mathbf{A}}_1 \\ \hat{\mathbf{A}}_2 \end{bmatrix} \cdot \mathbf{r} + \begin{bmatrix} \mathbf{0}^m \\ \mathbf{x} \end{bmatrix} \text{ and } \|\mathbf{r}\|_2 \leq B$$

Fig. 7. Statistically binding homomorphic commitment from [BDL+18]

A Lattice-based Statistically Binding Commitment

In Fig. 7, we give a formal specification of our statistically binding variant Baum et al.'s commitment scheme [BDL+18].

B Additional Preliminaries

B.1 Additional Properties for Commitment Scheme

In this subsection, we formally define the standard security requirements for a commitment scheme and two additional properties required by our protocols.

Definition 9 (Security of Commitment Scheme). *A commitment scheme* COM *is said to be secure if the following properties hold.*

- Correctness *It is correct if for any $msg \in S_{msg}$*

$$\Pr\left[\mathsf{Open}_{ck}(com, r, msg) \to 1 : \begin{array}{l} cpp \leftarrow \mathsf{CSetup}(1^\lambda);\, ck \leftarrow \mathsf{CGen}(cpp) \\ r \leftarrow\!\!\$\, D(S_r);\, com \leftarrow \mathsf{Commit}_{ck}(msg; r) \end{array}\right] = 1.$$

- Hiding *It is unconditionally (resp. computationally) hiding if the following probability is negligible for any probabilistic adversary (resp. probabilistic polynomial-time adversary) $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$.*

$$\epsilon_{hide} := \left| \Pr\left[b = b' : \begin{array}{l} cpp \leftarrow \mathsf{CSetup}(1^\lambda);\, ck \leftarrow \mathsf{CGen}(cpp) \\ (msg_0, msg_1) \leftarrow \mathcal{A}_1(ck, cpp) \\ b \leftarrow\!\!\$\, \{0,1\};\, com \leftarrow \mathsf{Commit}_{ck}(msg_b) \\ b' \leftarrow \mathcal{A}_2(com) \end{array}\right] - \frac{1}{2} \right|$$

- Binding *It is unconditionally (resp. computationally) binding if the following probability is negligible for any probabilistic adversary (resp. probabilistic polynomial-time adversary) $\mathcal{A}$.*

$$\epsilon_{bind} := \Pr\left[\begin{array}{l} msg \neq msg' \\ \mathsf{Open}_{ck}(com, r, msg) \to 1 \\ \mathsf{Open}_{ck}(com, r', msg') \to 1 \end{array} : \begin{array}{l} cpp \leftarrow \mathsf{CSetup}(1^\lambda) \\ ck \leftarrow \mathsf{CGen}(cpp) \\ (com, msg, r, msg', r') \leftarrow \mathcal{A}(ck) \end{array}\right]$$

  In particular, unconditionally binding implies that the following probability is also negligible, since otherwise unbounded adversaries can simply check all possible values in S_{com}, S_{msg} and S_r to find a tuple that breaks binding.

$$\epsilon_{ubind} := \Pr\left[\begin{array}{l} \exists (com, r, msg, r', msg') : \\ msg \neq msg' \\ \mathsf{Open}_{ck}(com, r, msg) \to 1 \\ \mathsf{Open}_{ck}(com, r', msg') \to 1 \end{array} : \begin{array}{l} cpp \leftarrow \mathsf{CSetup}(1^\lambda) \\ ck \leftarrow \mathsf{CGen}(cpp) \end{array}\right]$$

Definition 10 (Uniform Key). *A commitment key is said to be* uniform *if the output of* $\mathsf{CGen}(cpp)$ *follows the uniform distribution over the key space* S_{ck}.

Definition 11 (Additive Homomorphism). *A commitment is said to be* additively homomorphic *if for any* $msg, msg' \in S_{msg}$

$$\Pr\left[\mathsf{Open}_{ck}(com + com', r + r', msg + msg') \to 1 : \begin{array}{l} cpp \leftarrow \mathsf{CSetup}(1^\lambda) \\ ck \leftarrow \mathsf{CGen}(cpp) \\ r \leftarrow_\$ D(S_r); r' \leftarrow_\$ D(S_r) \\ com \leftarrow \mathsf{Commit}_{ck}(msg; r) \\ com' \leftarrow \mathsf{Commit}_{ck}(msg'; r') \end{array}\right] = 1.$$

B.2 General Forking Lemma

We restate the general forking lemma from [BN06].

Lemma 5 (General Forking Lemma). *Let* Q *be a number of queries and* C *be a set of size* $|C| > 2$. *Let* $\mathcal{B}$ *be a randomized algorithm that on input* x, $h_1, \ldots, h_Q$ *returns an index* $i \in [0, Q]$ *and a side output out. Let* IGen *be a randomized algorithm that we call the input generator. Let* $\mathcal{F}_\mathcal{B}$ *be a forking algorithm that works as in Fig. 8 given* x *as input and given black-box access to* $\mathcal{B}$. *Suppose the following probabilities.*

$$\begin{aligned} \mathrm{acc} &:= \Pr[i \neq 0 : x \leftarrow \mathsf{IGen}(1^\lambda); h_1, \ldots, h_Q \leftarrow_\$ C; (i, out) \leftarrow \mathcal{B}(x, h_1, \ldots, h_Q)] \\ \mathrm{frk} &:= \Pr[b = 1 : x \leftarrow \mathsf{IGen}(1^\lambda); (b, out, \hat{out}) \leftarrow \mathcal{F}_\mathcal{B}(x)] \end{aligned}$$

Then

$$\mathrm{frk} \geq \mathrm{acc} \cdot \left(\frac{\mathrm{acc}}{Q} - \frac{1}{|C|}\right).$$

Alternatively,

$$\mathrm{acc} \leq \frac{Q}{|C|} + \sqrt{Q \cdot \mathrm{frk}}.$$

```
Algorithm $\mathcal{F}_\mathcal{B}(x)$
Upon receiving $x$
  1. Pick a random coin $\rho$ for $\mathcal{B}$.
  2. Generate $h_1, \ldots, h_Q \leftarrow_\$ C$.
  3. $(i, out) \leftarrow \mathcal{B}(x, h_1, \ldots, h_Q; \rho)$.
  4. If $i = 0$ then return $(0, \perp, \perp)$.
  5. Regenerate $\hat{h}_i, \ldots, \hat{h}_Q \leftarrow_\$ C$.
  6. $(\hat{i}, \hat{out}) \leftarrow \mathcal{B}(x, h_1, \ldots, h_{i-1}, \hat{h}_i, \ldots, \hat{h}_Q; \rho)$.
  7. If $i = \hat{i}$ and $h_i \neq \hat{h}_i$ then return $(1, out, \hat{out})$
  8. Else return $(0, \perp, \perp)$.
```

Fig. 8. The forking algorithm $\mathcal{F}_\mathcal{B}$

C Security Proof for DS_2

Here we give a full security proof for our two-round protocol DS_2. See Fig. 5 for the protocol specification.

Theorem 1. *Suppose the trapdoor commitment scheme* TCOM *is secure, additively homomorphic and has uniform keys. For any probabilistic polynomial-time adversary* $\mathcal{A}$ *that initiates a single key generation protocol by querying* $\mathcal{O}_n^{\mathsf{DS}_2}$ *with* $sid = 0$*, initiates* Q_s *signature generation protocols by querying* $\mathcal{O}_n^{\mathsf{DS}_2}$ *with* $sid \neq 0$*, and makes* Q_h *queries to the random oracle* $\mathsf{H}_0, \mathsf{H}_1, \mathsf{H}_2, \mathsf{H}_3$*, the protocol* DS_2 *of Fig. 5 is* $\mathsf{DS\text{-}UF\text{-}CMA}$ *secure under* $\mathsf{MSIS}_{q,k,\ell+1,\beta}$ *and* $\mathsf{MLWE}_{q,k,\ell,\eta}$ *assumptions, where* $\beta = 2\sqrt{B_n^2 + \kappa}$*. Concretely, using other parameters specified in Table 1, the advantage of* $\mathcal{A}$ *is bounded as follows.*

$$\mathbf{Adv}_{\mathsf{DS}_2}^{\mathsf{DS\text{-}UF\text{-}CMA}}(\mathcal{A}) \leq e \cdot (Q_h + Q_s + 1) \cdot \Bigg((Q_h + Q_s)\epsilon_{td} + Q_s \cdot \frac{e^{-t^2/2}}{M} + \mathbf{Adv}_{\mathsf{MLWE}_{q,k,\ell,\eta}}$$
$$+ \frac{(Q_h+1)Q_h}{2^{l_1+1}} + \frac{Q_h}{q^{k\ell N}} + \frac{n}{2^{l_1}} + \frac{(Q_h+1)Q_h}{2^{l_2+1}} + \frac{Q_h}{q^{kN}} + \frac{n}{2^{l_2}}$$
$$+ \frac{Q_h + Q_s + 1}{|C|} + \sqrt{(Q_h + Q_s + 1) \cdot \left(\epsilon_{bind} + \mathbf{Adv}_{\mathsf{MSIS}_{q,k,\ell+1,\beta}}\right)} \Bigg)$$

Proof. Given $\mathcal{A}$ against DS_2 we show that its advantage $\mathbf{Adv}_{\mathsf{DS}_2}^{\mathsf{DS\text{-}UF\text{-}CMA}}(\mathcal{A})$ is negligible by constructing a reduction. Without loss of generality we assume that P_n is an honest party. Our first goal is to construct an algorithm $\mathcal{B}$ around $\mathcal{A}$ that simulates the behaviors of P_n without using honestly generated key pairs. Then we invoke the forking algorithm $\mathcal{F}_{\mathcal{B}}$ from Lemma 5 to obtain two forgeries with distinct challenges, which allow to construct a solution to $\mathsf{MSIS}_{q,k,\ell+1,\beta}$ or to break computational binding of commitment scheme TCOM. We present the resulting $\mathcal{B}$ in Fig. 9, together with its subroutines Figs. 10 to 14. Below we discuss how to realize this via several intermediate hybrids.

$\mathbf{G}_0$ **Random oracle simulation.** We assume that $\mathcal{B}$ receives random samples $h_i \leftarrow\!\!\$\, C$ for each $i \in [Q_h + Q_s + 1]$ as input. The random oracles $\mathsf{H}_0 : \{0,1\}^* \to C$, $\mathsf{H}_1 : \{0,1\}^* \to \{0,1\}^{l_1}$, $\mathsf{H}_2 : \{0,1\}^* \to \{0,1\}^{l_2}$ and $\mathsf{H}_3 : \{0,1\}^* \to S_{ck}$ are simulated as follows. The table HT_i is initially empty. The $\mathcal{B}$ also maintains a counter ctr which is initially set to 0. Note that the slightly involved simulation of H_0 below will come into play when the forking lemma is applied; this way, the adversary $\mathcal{A}$'s view is indeed identical until the forking point in two executions.

$\mathsf{H}_0(x)$ 1. Parse x as $(com, \mu, \mathbf{t})$; 2. Make a query $\mathsf{H}_3(\mu, \mathbf{t})$, so that $\mathrm{HT}_3[\mu, \mathbf{t}]$ is immediately set; 3. If $\mathrm{HT}_0[com, \mu, \mathbf{t}] = \perp$ then increment ctr and set $\mathrm{HT}_0[com, \mu, \mathbf{t}] := h_{ctr}$; 4. Return $\mathrm{HT}_0[com, \mu, \mathbf{t}]$.

$\mathsf{H}_1(x)$ If $\mathrm{HT}_1[x]$ is not set let $\mathrm{HT}_1[x] \leftarrow\!\!\$\, \{0,1\}^{l_1}$. Return $\mathrm{HT}_1[x]$.

$\mathsf{H}_2(x)$ If $\mathrm{HT}_2[x]$ is not set let $\mathrm{HT}_2[x] \leftarrow\!\!\$\, \{0,1\}^{l_2}$. Return $\mathrm{HT}_2[x]$.

$\mathsf{H}_3(\mu, \mathbf{t})$ If $\mathrm{HT}_3[\mu, \mathbf{t}]$ is not set let $\mathrm{HT}_3[\mu, \mathbf{t}] \leftarrow\!\!\$\, S_{ck}$. Return $\mathrm{HT}_3[\mu, \mathbf{t}]$.

Honest party oracle simulation. In this game $\mathcal{B}$ behaves exactly like a single honest party in DS_2; concretely, it simulates an oracle $\mathcal{O}_n^{\mathsf{DS}_2}$ (Fig. 3) which internally invokes instructions of Gen_n and Sign_n according to Fig. 5, respectively.

Forgery. When $\mathcal{A}$ outputs a forgery $(com^*, \mathbf{z}^*, r^*, \mu^*)$ at the end $\mathcal{B}$ proceeds as follows.

1. If $\mu^* \in Mset$ then $\mathcal{B}$ halts with output $(0, \perp)$
2. Make queries $ck^* \leftarrow \mathsf{H}_3(\mu^*, \mathbf{t})$ and $c^* \leftarrow \mathsf{H}_0(com^*, \mu^*, \mathbf{t})$
3. If $\mathsf{Open}_{ck^*}(com^*, \bar{\mathbf{A}}\mathbf{z}^* - c^*\mathbf{t}, r^*) \neq 1$ or $\|\mathbf{z}^*\|_2 > B_n$ then $\mathcal{B}$ halts with output $(0, \perp)$.
4. Find $i_f \in [Q_h + Q_s + 1]$ such that $c^* = h_{i_f}$ and $\mathcal{B}$ halts with output $(i_f, out = (com^*, c^*, \mathbf{z}^*, r^*, \mu^*, ck^*))$

Let $\Pr[\mathbf{G}_i]$ denote a probability that $\mathcal{B}$ doesn't output $(0, \perp)$ at the game $\mathbf{G}_i$. Then we have

$$\Pr[\mathbf{G}_0] = \mathbf{Adv}_{\mathsf{DS}_2}^{\mathsf{DS\text{-}UF\text{-}CMA}}(\mathcal{A}).$$

$\mathbf{G}_1$ This game is identical to $\mathbf{G}_0$ except at the following points.

Random oracle simulation. Simulation of the random oracle H_3 is analogous to Drijvers et al. [DEF+19] The core idea is to make sure that all sign queries can be responded with trapdoor commitments, which can be equivocated to an arbitrary plaintext later, and that the forgery submitted by $\mathcal{A}$ involves the actual commitment key. In this game $\mathcal{B}$ initially generates a single commitment key $ck \leftarrow\!\!\$\, S_{ck}$. Then upon receiving a query $(\mu, \mathbf{t})$ to H_3, $\mathcal{B}$ tosses a biased coin that comes out heads with probability ϖ and tails with $1 - \varpi$. If the coin comes out heads, then $\mathcal{B}$ invokes TCGen to generate a commitment key–trapdoor pair (tck, td), stores tck and td in tables $\mathrm{HT}_3[\mu, \mathbf{t}]$ and $\mathrm{TDT}[\mu, \mathbf{t}]$, respectively, and returns tck; if it comes out tails, then $\mathcal{B}$ stores a predefined ck in $\mathrm{HT}_3[\mu, \mathbf{t}]$ and returns ck. The complete description of random oracle simulation is presented in Fig. 11.

Honest party oracle simulation. The $\mathcal{B}$ differs from the prior one at the following steps in $\mathsf{DS}_2.\mathsf{Sign}_n$.

Inputs 3 Call $\mathsf{H}_3(\mu, \mathbf{t})$ to obtain tck. If $\mathrm{TDT}[\mu, \mathbf{t}] = \perp$ (i.e., TCGen was not called) then set a flag bad_4 and halt with output $(0, \perp)$. Otherwise obtain the trapdoor $td \leftarrow \mathrm{TDT}[\mu, \mathbf{t}]$

Signature Generation 1.b. Call $com_n \leftarrow \mathsf{TCommit}_{tck}(td)$ instead of committing to $\mathbf{w}_n$.

Signature Generation 2.c. After computing $\mathbf{z}_n := c\mathbf{s}_n + \mathbf{y}_n$ derive randomness $r_n \leftarrow \mathsf{Eqv}_{tck}(td, com_n, \mathbf{w}_n)$.

Forgery. When $\mathcal{A}$ outputs a successful forgery $(com^*, \mathbf{z}^*, r^*, \mu^*)$ at the end, we modify the step 3 of $\mathbf{G}_0$ as follows.

Forgery 3 If $\mathsf{Open}_{ck^*}(com^*, \bar{\mathbf{A}}\mathbf{z}^* - c^*\mathbf{t}, r^*) \neq 1$ or $\|\mathbf{z}^*\|_2 > B_n$ then $\mathcal{B}$ halts with output $(0, \perp)$. *If* $\mathrm{TDT}[\mu^*, \mathbf{t}] \neq \perp$ *(i.e.,* TCGen *was called for a query* $\mathsf{H}_3(\mu^*, \mathbf{t})$*)* then set bad_5 and $\mathcal{B}$ halts with output $(0, \perp)$.

Note that due to the way H_3 is simulated, if $\mathcal{B}$ does not output $(0, \perp)$ it is now guaranteed that $ck^* = ck = \mathsf{H}_3(\mu^*, \mathbf{t})$. Recalling that TCOM is secure (Definition 4) we have

$$\Pr[\mathbf{G}_1] \geq \varpi^{Q_h + Q_s} \cdot (1 - \varpi) \cdot \Pr[\mathbf{G}_0] - (Q_h + Q_s) \cdot \epsilon_{\mathrm{td}}$$

because the simulation is only successful if the random oracle H_3 internally uses TCGen for all but one queries to H_3 (both directly and indirectly via H_0 and Sign_n) *and* if H_3 uses a predefined ck for a single crucial query $(\mu^*, \mathbf{t})$ associated with forgery; in other words, it is only successful if neither bad_4 nor bad_5 is set above. Note that by setting $\varpi = (Q_h + Q_s)/(Q_h + Q_s + 1)$ since $(1/(1 + 1/(Q_h + Q_s)))^{Q_h + Q_s} \geq 1/e$ for $Q_h + Q_s \geq 0$ we obtain

$$\Pr[\mathbf{G}_1] \geq \frac{\Pr[\mathbf{G}_0]}{e(Q_h + Q_s + 1)} - (Q_h + Q_s) \cdot \epsilon_{\mathrm{td}}.$$

$\mathbf{G}_2$ This game is identical to $\mathbf{G}_1$ except at the following points.

Honest party oracle simulation. The $\mathcal{B}$ doesn't honestly generate $\mathbf{z}_n$ anymore and instead simulates the rejection sampling as follows.

Signature Generation 1.a. $\mathcal{B}$ does nothing here.

Signature Generation 2.c. Sample $\mathbf{z}_n \leftarrow_\$ D_s^{\ell+k}$ and derive randomness $r_n \leftarrow \mathsf{Eqv}_{tck}(td, com_n, \mathbf{w}_n := \bar{\mathbf{A}}\mathbf{z}_n - c\mathbf{t}_n)$.

Signature Generation 2.d. With probability $1/M$ send out $(\mathbf{z}_n, r_n)$. Otherwise send out RESTART.

The signature share $\mathbf{z}_n$ simulated this way is statistically indistinguishable from the real one because of special HVZK property of the underlying identification scheme. In other words, we can directly apply the result of Lemmas 3 and 4. Hence we have

$$|\Pr[\mathbf{G}_2] - \Pr[\mathbf{G}_1]| \leq Q_s \cdot \frac{e^{-t^2/2}}{M}.$$

$\mathbf{G}_3$ At this stage, notice that the signing queries are simulated according to $\mathsf{SimSign}_n$ in Fig. 14, and it doesn't rely on the actual secret key share $\mathbf{s}_n$ anymore. So our next goal is to simulate the generation of $\mathbf{t}_n$ without using $\mathbf{s}_n$. In this game $\mathcal{B}$ first picks the resulting random matrix $\mathbf{A} \in R_q^{k \times \ell}$ and defines its own share $\mathbf{A}_n$ a posteriori, after extracting adversary's committed shares $\mathbf{A}_1, \ldots, \mathbf{A}_{n-1}$. This can be done by searching the recorded random oracle queries in HT_1. Note that the distributions of $\mathbf{A}$ and $\mathbf{A}_n$ haven't changed from the previous game. The formal simulation strategy is described in **Matrix Generation** section of Fig. 13. Since $\mathbf{G}_3$ is identical to $\mathbf{G}_2$ from adversary $\mathcal{A}$'s point of view except at the *bad* events marked there, we have

$$|\Pr[\mathbf{G}_3] - \Pr[\mathbf{G}_2]| \leq \Pr[bad_1] + \Pr[bad_2] + \Pr[bad_3] \leq \frac{(Q_h + 1)Q_h}{2^{l_1+1}} + \frac{Q_h}{q^{k\ell N}} + \frac{n}{2^{l_1}}$$

where $\Pr[bad_1]$ corresponds to the probability that at least one collision occurs during at most Q_h queries to H_1 made by $\mathcal{A}$ or $\mathcal{B}$, which is at most $((Q_h + 1)Q_h/2)/2^{l_1}$; $\Pr[bad_2]$ is the probability that programming the random oracle H_1 fails, which happens only if $\mathsf{H}_1(\mathbf{A}_n, n)$ has been previously asked by $\mathcal{A}$ during at most Q_h queries to H_1, and the probability that guessing a uniformly random $\mathbf{A}_n$ by chance is at most $1/q^{k\ell N}$ for each query; $\Pr[bad_3]$ is the probability that $\mathcal{A}$ has predicted one of the $n-1$ outputs of random oracle H_1 without making a query to it, which could only happen with probability at most $n/2^{l_1}$.

$\mathbf{G}_4$ This game is identical to $\mathbf{G}_3$ except that $\mathcal{B}$ simply picks the random public key share $\mathbf{t}_n \leftarrow\!\!\$\, R_q^k$ during the key generation phase, instead of computing $\mathbf{t}_n = \bar{\mathbf{A}}\mathbf{s}_n$ where $\mathbf{s}_n \leftarrow\!\!\$\, S_\eta^{\ell+k}$. As $\mathbf{A}$ follows the uniform distribution over $R_q^{k\times\ell}$, if the adversary $\mathcal{A}$ can distinguish $\mathbf{G}_3$ and $\mathbf{G}_4$ then we can use $\mathcal{A}$ as a distinguisher that breaks $\mathsf{MLWE}_{q,k,\ell,\eta}$ assumption; hence we have

$$|\Pr[\mathbf{G}_4] - \Pr[\mathbf{G}_3]| \leq \mathbf{Adv}_{\mathsf{MLWE}_{q,k,\ell,\eta}}.$$

$\mathbf{G}_5$ In this game $\mathcal{B}$ first picks the resulting public key $\mathbf{t}$ randomly from R_q^k and defines its own share $\mathbf{t}_n$ a posteriori, after extracting adversary's committed shares $\mathbf{t}_1, \ldots, \mathbf{t}_{n-1}$. This can be done by searching the recorded random oracle queries in HT_2. Note that the distributions of $\mathbf{t}$ and $\mathbf{t}_n$ haven't changed from the previous game. The formal simulation strategy is described in **Key Pair Generation** section of Fig. 13. Since $\mathbf{G}_5$ is identical to $\mathbf{G}_4$ from adversary $\mathcal{A}$'s point of view except at the bad' events marked there, we have

$$|\Pr[\mathbf{G}_5] - \Pr[\mathbf{G}_4]| \leq \Pr[bad'_1] + \Pr[bad'_2] + \Pr[bad'_3] \leq \frac{(Q_h+1)Q_h}{2^{l_2+1}} + \frac{Q_h}{q^{kN}} + \frac{n}{2^{l_2}}$$

where the bounds are calculated just as in $\mathbf{G}_3$.

Forking lemma. At this stage, notice that the key generation query is simulated according to SimGen_n in Fig. 13. Our goal is to embed a challenge commitment key $ck \leftarrow \mathsf{CGen}(cpp)$ and an instance of $\mathsf{MSIS}_{q,k,\ell+1,\beta}$, which is denoted as $[\mathbf{A}'|\mathbf{I}]$ with $\mathbf{A}' \leftarrow\!\!\$\, R_q^{k\times(\ell+1)}$. As in $\mathbf{G}_5$ the combined public key $(\mathbf{A}, \mathbf{t})$ is uniformly distributed in $R_q^{k\times\ell} \times R_q^k$, replacing it with $\mathsf{MSIS}_{q,k,\ell+1,\beta}$ instance doesn't change the view of adversary at all, if $\mathbf{A}'$ is regarded as $\mathbf{A}' = [\mathbf{A}|\mathbf{t}]$. Moreover, thanks to the simulation of H_3 it is guaranteed that ck follows the uniform distribution over S_{ck} which is perfectly indistinguishable from honestly generated $ck \leftarrow \mathsf{CGen}(cpp)$ (since the keys are uniform). Hence we define the input generator IGen of forking lemma such that it outputs the instance $(ck, \mathbf{A}, \mathbf{t})$.

Now we prove the theorem by constructing $\mathcal{B}'$ around $\mathcal{B}$ in Fig. 9 that either (1) breaks binding of commitment wrt ck, or (2) finds a solution to $\mathsf{MSIS}_{q,k,\ell+1,\beta}$ on input $\mathbf{A}' = [\mathbf{A}|\mathbf{t}]$. The $\mathcal{B}'$ invokes the forking algorithm $\mathcal{F}_\mathcal{B}$ on input $(ck, \mathbf{A}, \mathbf{t})$ from Lemma 5. Then with probability frk we immediately get two forgeries $out = (com^*, c^*, \mathbf{z}^*, r^*, \mu^*, ck^*)$ and $\hat{out} = (\hat{com}^*, \hat{c}^*, \hat{\mathbf{z}}^*, \hat{r}^*, \hat{\mu}^*, \hat{ck}^*)$, where frk satisfies

$$\Pr[\mathbf{G}_5] = \mathrm{acc} \leq \frac{Q_h + Q_s + 1}{|C|} + \sqrt{(Q_h + Q_s + 1) \cdot \mathrm{frk}}.$$

By construction of $\mathcal{B}$ and $\mathcal{F}_\mathcal{B}$ we have $com^* = \hat{com}^*$, $\mu^* = \hat{\mu}^*$ and $c^* \neq \hat{c}^*$; until the forking point $ctr = i_\mathrm{f}$ the adversary $\mathcal{A}$'s view is identical in two executions. Moreover, due to the simulation of H_0 we also guarantee that $ck^* = \hat{ck}^* = ck$ since $\mathsf{H}_3(\mu^*, \mathbf{t})$ and $\mathsf{H}_3(\hat{\mu}^*, \mathbf{t})$ should have been invoked right before the fork. Since both forgeries are verified under the same commitment key ck, we have $\|\mathbf{z}^*\|_2 \leq B_n \wedge \|\hat{\mathbf{z}}^*\|_2 \leq B_n$ and

$$\mathsf{Open}_{ck}(com^*, r^*, \bar{\mathbf{A}}\mathbf{z}^* - c^*\mathbf{t}) = \mathsf{Open}_{ck}(com^*, \hat{r}^*, \bar{\mathbf{A}}\hat{\mathbf{z}}^* - \hat{c}^*\mathbf{t}) = 1.$$

If $\bar{\mathbf{A}}\mathbf{z}^* - c^*\mathbf{t} \neq \bar{\mathbf{A}}\hat{\mathbf{z}}^* - \hat{c}^*\mathbf{t}$ then $\mathcal{B}'$ can break computational binding with respect to ck, which succeeds with probability at most ϵ_bind. If $\bar{\mathbf{A}}\mathbf{z}^* - c^*\mathbf{t} = \bar{\mathbf{A}}\hat{\mathbf{z}}^* - \hat{c}^*\mathbf{t}$, rearranging it leads to

$$[\mathbf{A}|\mathbf{I}|\mathbf{t}] \begin{bmatrix} \mathbf{z}^* - \hat{\mathbf{z}}^* \\ \hat{c}^* - c^* \end{bmatrix} = \mathbf{0}.$$

Recalling that $[\mathbf{A}'|\mathbf{I}] = [\mathbf{A}|\mathbf{t}|\mathbf{I}]$ is an instance of $\mathsf{MSIS}_{q,k,\ell+1,\beta}$ problem, we have found a valid solution if $\beta = \sqrt{(2B_n)^2 + 4\kappa}$, since $\|\mathbf{z}^* - \hat{\mathbf{z}}^*\|_2 \leq 2B_n$ and $0 < \|\hat{c}^* - c^*\|_2 \leq \sqrt{4\kappa}$. Putting two cases together, we get

$$\mathrm{frk} \leq \epsilon_\mathrm{bind} + \mathbf{Adv}_{\mathsf{MSIS}_{q,k,\ell+1,\beta}}.$$

Algorithm $\mathcal{B}((ck, \mathbf{A}, \mathbf{t}), h_1, \ldots, h_{Q_h+Q_s+1})$

The algorithm is initialized with empty hash tables HT_i for $i = 0, \ldots, 3$, trapdoor table TDT, a set of queried messages $Mset = \varnothing$, and a counter $ctr = 0$.

Honest party oracle simulation Upon receiving a query of the form (sid, m) from $\mathcal{A}$, reply the query as described in $\mathsf{Sim}\mathcal{O}_n^{\mathsf{DS}_2}(sid, m)$ (Fig. 10). If $\mathsf{Sim}\mathcal{O}_n^{\mathsf{DS}_2}$ halts with output $(0, \perp)$ then $\mathcal{B}$ also halts with output $(0, \perp)$.

Random oracle simulation Upon receiving a query to the random oracles from $\mathcal{A}$, reply the query as described in Fig. 11.

Forgery Upon receiving a forgery $(com^*, \mathbf{z}^*, r^*, \mu^*)$ from $\mathcal{A}$:

```
1. If $\mu^* \in Mset$ then $\mathcal{B}$ halts with output $(0, \perp)$
2. Make queries $ck^* \leftarrow \mathsf{H}_3(\mu^*, \mathbf{t})$ and $c^* \leftarrow \mathsf{H}_0(com^*, \mu^*, \mathbf{t})$
3. If $\mathsf{Open}_{ck^*}(com^*, \bar{\mathbf{A}}\mathbf{z}^* - c^*\mathbf{t}, r^*) \neq 1$ or $\|\mathbf{z}^*\|_2 > B_n$ then $\mathcal{B}$ halts with output $(0, \perp)$. If $\mathrm{TDT}[\mu^*, \mathbf{t}] \neq \perp$ (i.e., $\mathsf{TCGen}$ was called for a query $\mathsf{H}_3(\mu^*, \mathbf{t})$) then set $bad_5$ and $\mathcal{B}$ halts with output $(0, \perp)$.
4. Find $i_f \in [Q_h + Q_s + 1]$ such that $c^* = h_{i_f}$ and $\mathcal{B}$ halts with output $(i_f, out = (com^*, c^*, \mathbf{z}^*, r^*, \mu^*, ck^*))$
```

Fig. 9. The algorithm simulating the view of $\mathcal{A}$ in $\mathsf{Exp}_{\mathsf{DS}_2}^{\mathsf{DS\text{-}UF\text{-}CMA}}(\mathcal{A})$ experiment

Oracle $\mathsf{Sim}\mathcal{O}_n^{\mathsf{DS}_2}(sid, m)$

The simulator is initialized with public parameters pp generated by Setup algorithm. The variable $flag$ is initially set to false.

Key Generation Upon receiving $(0, m)$, if $flag = \mathsf{true}$ then return $\perp$. Otherwise do the following:

- If the oracle is queried with $sid = 0$ for the first time then it initializes a machine $\mathcal{M}_0$ running the instructions $\mathsf{SimGen}_n(pp, \mathbf{A}, \mathbf{t})$ (Fig. 13).
- If $\mathcal{M}_0$ has been already initialized then the oracle hands the machine $\mathcal{M}_0$ the next incoming message m and returns $\mathcal{M}_0$'s reply. If $\mathcal{M}_0$ fails with output $(0, \perp)$ at any point then the oracle stops the simulation with output $(0, \perp)$. If $\mathcal{M}_0$ concludes $\mathsf{SimGen}_n(pp, \mathbf{A}, \mathbf{t})$ with local output $(\mathbf{t}_n, pk)$, then set $flag = \mathsf{true}$.

Signature Generation Upon receiving (sid, m) with $sid \neq 0$, if $flag = \mathsf{false}$ then return $\perp$. Otherwise do the following:

- If the oracle is queried with sid for the first time then parse the incoming message m as μ. It initializes a machine $\mathcal{M}_{sid}$ running the instructions of $\mathsf{SimSign}_n(sid, \mathbf{t}_n, pk, \mu)$ (Fig. 14). The machine $\mathcal{M}_{sid}$ is initialized with the key share and any state information stored by $\mathcal{M}_0$. The message μ to be signed is included in $Mset$. If P_n sends the first message in the signing protocol, then this message is the oracle reply.
- If $\mathcal{M}_{sid}$ has been already initialized then the oracle hands the machine $\mathcal{M}_{sid}$ the next incoming message m and returns the next message sent by $\mathcal{M}_{sid}$. If $\mathcal{M}_{sid}$ fails with output $(0, \perp)$ at any point then the oracle stops the simulation with output $(0, \perp)$. If $\mathcal{M}_{sid}$ concludes with local output σ, then the output obtained by $\mathcal{M}_{sid}$ is returned.

Fig. 10. Honest party oracle simulator for DS_2.

Algorithm Random Oracle Simulation

```
$\mathsf{H}_0(x)$
  1. Parse $x$ as $(com, \mu, \mathbf{t})$
  2. Make a query $\mathsf{H}_3(\mu, \mathbf{t})$
  3. If $\mathrm{HT}_0[com, \mu, \mathbf{t}] = \perp$ then increment $ctr$ and set $\mathrm{HT}_0[com, \mu, \mathbf{t}] := h_{ctr}$
  4. Return $\mathrm{HT}_0[com, \mu, \mathbf{t}]$
$\mathsf{H}_1(x)$
  1. If $\mathrm{HT}_1[x] = \perp$ then set $\mathrm{HT}_1[x] \leftarrow\$ \{0,1\}^{l_1}$
  2. Return $\mathrm{HT}_1[x]$
$\mathsf{H}_2(x)$
  1. If $\mathrm{HT}_2[x] = \perp$ then set $\mathrm{HT}_2[x] \leftarrow\$ \{0,1\}^{l_2}$
  2. Return $\mathrm{HT}_2[x]$
$\mathsf{H}_3(x)$
  1. Parse $x$ as $(\mu, \mathbf{t})$
  2. If $\mathrm{HT}_3[\mu, \mathbf{t}] = \perp$:
     – With probability $\varpi$ compute $(tck, td) \leftarrow \mathsf{TCGen}(cpp)$, store the trapdoor in $\mathrm{TDT}[\mu, \mathbf{t}] := td$ and set $\mathrm{HT}_3[\mu, \mathbf{t}] := tck$.
     – With probability $1 - \varpi$, set $\mathrm{HT}_3[\mu, \mathbf{t}] := ck$.
  3. Return $\mathrm{HT}_3[\mu, \mathbf{t}]$
```

Fig. 11. Random oracle simulator for DS_2.

```
Algorithm SearchHashTable

Upon receiving the hash table HT together with hash values $(h_1, \dots, h_{n-1})$:
  1. If for some $j \in [n-1]$ the preimage of $h_j$ doesn't exist in HT then set the flag $alert$.
  2. If for some $j \in [n-1]$ more than one preimages of $h_j$ exist in HT then set the flag $bad$.
  3. Return $(alert, bad, m_1, \dots, m_{n-1})$, where for each $j \in [n-1]$ if there is no $m_j$ such that $\mathrm{HT}[m_j] = h_j$ then
     $m_j = \perp$, and otherwise $m_j$ is defined such that $\mathrm{HT}[m_j] = h_j$.
```

Fig. 12. Routine for searching hash tables.

```
Algorithm SimGen_n$(pp, \mathbf{A}, \mathbf{t})$

Matrix Generation
  1. Sample $g_n \leftarrow\!\!\$\, \{0,1\}^{l_1}$ and send out $g_n$.
  2. Upon receiving $g_j$ for all $j \in [n-1]$ proceed as follows:
     a. Invoke SearchHashTable in Fig. 12 on input $\mathrm{HT}_1$ and $(g_1, \dots, g_{n-1})$ to obtain
        $(alert, bad_1, (\mathbf{A}_1, 1), \dots, (\mathbf{A}_{n-1}, n-1))$.
     b. If the flag $bad_1$ is set then simulation fails with output $(0, \perp)$.
     c. If the flag $alert$ is set then pick $\mathbf{A}_n \leftarrow\!\!\$\, R_q^{k \times \ell}$. Otherwise using a predefined matrix $\mathbf{A}$ define $\mathbf{A}_n := \mathbf{A} -$
        $\sum_{j=1}^{n-1} \mathbf{A}_j$.
        – If $\mathrm{HT}_1[\mathbf{A}_n, n]$ has been already set then set $bad_2$ and simulation fails with output $(0, \perp)$.
        – Otherwise program the random oracle $\mathrm{HT}_1[\mathbf{A}_n, n] := g_n$ and send out $\mathbf{A}_n$.
  3. Upon receiving $\mathbf{A}_j$ for all $j \in [n-1]$:
     a. If $\mathsf{H}_1(\mathbf{A}_j, j) \neq g_j$ for some $j$ then send out ABORT.
     b. If $alert$ is set and $\mathsf{H}_1(\mathbf{A}_j, j) = g_j$ for all $j$ then set $bad_3$ and simulation fails with output $(0, \perp)$.
     c. Otherwise set a public random matrix $\bar{\mathbf{A}} := [\mathbf{A}|\mathbf{I}] \in R_q^{k \times (\ell + k)}$.
Key Pair Generation
  1. Sample $g'_n \leftarrow\!\!\$\, \{0,1\}^{l_2}$ and send out $g'_n$.
  2. Upon receiving $g'_j$ for all $j \in [n-1]$ proceed as follows:
     a. Invoke SearchHashTable in Fig. 12 on input $\mathrm{HT}_2$ and $(g'_1, \dots, g'_{n-1})$ to obtain
        $(alert', bad'_1, (\mathbf{t}_1, 1), \dots, (\mathbf{t}_{n-1}, n-1))$.
     b. If the flag $bad'_1$ is set then simulation fails with output $(0, \perp)$.
     c. If the flag $alert'$ is set then pick $\mathbf{t}_n \leftarrow\!\!\$\, R_q^k$. Otherwise using a predefined public key $\mathbf{t}$ define $\mathbf{t}_n := \mathbf{t} -$
        $\sum_{j=1}^{n-1} \mathbf{t}_j$.
        – If $\mathrm{HT}_2[\mathbf{t}_n, n]$ has been already set then set $bad'_2$ and simulation fails with output $(0, \perp)$.
        – Otherwise program the random oracle $\mathrm{HT}_2[\mathbf{t}_n, n] := g'_n$ and send out $\mathbf{t}_n$.
  3. Upon receiving $\mathbf{t}_j$ for all $j \in [n-1]$:
     a. If $\mathsf{H}_2(\mathbf{t}_j, j) \neq g'_j$ for some $j$ then send out ABORT.
     b. If $alert'$ is set and $\mathsf{H}_2(\mathbf{t}_j, j) = g'_j$ for all $j$ then set $bad'_3$ and simulation fails with output $(0, \perp)$.
If neither the protocol aborts nor the simulation fails, the simulator obtains public key share $\mathbf{t}_n$ and $pk = (\bar{\mathbf{A}}, \mathbf{t})$ as
  local output.
```

Fig. 13. Key generation simulator for DS_2

Algorithm $\mathsf{SimSign}_n(sid, \mathbf{t}_n, pk, \mu)$

The simulator is parameterized by public parameters described in Table 1 and relies on the random oracles $\mathsf{H}_0 : \{0,1\}^* \to C$ and $\mathsf{H}_3 : \{0,1\}^* \to S_{ck}$. The simulator assumes that $\mathsf{SimGen}_n(pp, \mathbf{A}, \mathbf{t})$ (Fig. 13) has been previously invoked. If the simulator halts with ABORT at any point, then all $\mathsf{SimSign}_n(sid, \mathbf{t}_n, pk, \mu)$ executions are aborted.

Inputs

1. The simulator receives a unique session ID sid, $\mathbf{t}_n$, $pk = (\bar{\mathbf{A}}, \mathbf{t})$ and message $\mu \in \{0,1\}^*$ as input.
2. The simulator verifies that sid has not been used before (if it has been, the protocol is not executed).
3. The simulator locally computes a per-message commitment key by querying a random oracle $tck \leftarrow \mathsf{H}_3(\mu, \mathbf{t})$. If $\mathrm{TDT}[\mu, \mathbf{t}] = \bot$ (i.e., TCGen was not called) then set bad_4 and simulation fails with output $(0, \bot)$. Otherwise obtain the trapdoor $td \leftarrow \mathrm{TDT}[\mu, \mathbf{t}]$.

Signature Generation

1. Compute the first message as follows.
   a. Compute $com_n \leftarrow \mathsf{TCommit}_{tck}(td)$.
   b. Send out com_n.
2. Upon receiving com_j for all $j \in [n-1]$ compute the signature share as follows.
   a. Set $com := \sum_{j \in [n]} com_j$.
   b. Derive a challenge $c \leftarrow \mathsf{H}_0(com, \mu, \mathbf{t})$.
   c. Computes a simulated signature share $\mathbf{z}_n \leftarrow_\$ D_s^{\ell+k}$ and derive randomness $r_n \leftarrow \mathsf{Eqv}_{tck}(td, com_n, \mathbf{w}_n = \bar{\mathbf{A}}\mathbf{z}_n - c\mathbf{t}_n)$.
   d. With probability $1/M$ send out $(\mathbf{z}_n, r_n)$; otherwise send out RESTART and go to 1.
3. Upon receiving RESTART from some party go to 1. Otherwise upon receiving $(\mathbf{z}_j, r_j)$ for all $j \in [n-1]$ compute the combined signature as follows
   a. For each $j \in [n-1]$ reconstruct $\mathbf{w}_j := \bar{\mathbf{A}}\mathbf{z}_j - c\mathbf{t}_j$ and validate the signature share:
   $$\|\mathbf{z}_j\|_2 \leq B \quad \text{and} \quad \mathsf{Open}_{tck}(com_j, r_j, \mathbf{w}_j) = 1.$$
   If the check fails for some j then send out ABORT.
   b. Compute $\mathbf{z} := \sum_{j \in [n]} \mathbf{z}_j$ and $r := \sum_{j \in [n]} r_j$.

If neither the protocol aborts nor the simulation fails, the simulator obtains a signature $(com, \mathbf{z}, r)$ as local output.

Fig. 14. Signature generation simulator for DS_2.

D MS_2: Two-round Multi-signature in the Plain Public Key Model

In this section we describe our two-round multi-signature scheme MS_2. The main difference from n-out-of-n signature is that, the protocol requires no interactive key generation at all, and instead for each signing execution a party receives a set of public keys L together with a message to be signed. As the number of participants may change for each signing attempt, in this section we define n to be the *maximum number of signers allowed in a single execution of signing protocol*, i.e., only L of cardinalty at most n is a valid input.

Now we present concrete specifications of $\mathsf{MS}_2 = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$. The Setup works just like the one for DS_2, but it additionally outputs a matrix $\bar{\mathbf{A}} = [\mathbf{A}|\mathbf{I}]$ as part of public parameters, so we assume that $\bar{\mathbf{A}}$ is generated by a trusted third party (if the generation of $\bar{\mathbf{A}}$ has to be done in a distributed way then parties can invoke a matrix generation protocol in Fig. 5 instead and a signer only uses $\bar{\mathbf{A}}$ as long as it participated in the generation of $\bar{\mathbf{A}}$). Then Gen algorithm is the same as Algorithm 1, except that it takes $\bar{\mathbf{A}}$ as input and outputs $sk = \mathbf{s}$ and $pk = \mathbf{t}$. In a multi-signature scheme, the indices assigned to the signers are just local references to the cosigners participating in a particular protocol instance [BN06], and therefore we wlog assume that each signer assign the index n to itself, and consider other signers' indices as $1, \ldots, n'-1$, where $n' = |L| \leq n$. The signing protocol Sign and verification Ver are described in Fig. 15. The only difference from $\mathsf{DS}_2.\mathsf{Sign}_n$ and $\mathsf{DS}_2.\mathsf{Ver}$ is that signature shares are now constructed from per-user challenges, instead of a single common challenge for all co-signers (just as Bellare–Neven [BN06] or Bagherzandi et al. [BCJ08] did). Therefore, the random oracle simulation below is more involved than in the proof for Theorem 1. On the other hand, as MS_2 has no interactive key generation, the proof only requires much simpler key generation simulation. Therefore, the concrete security bound in the following theorem is slightly better than the previous case.

Theorem 3. *Suppose the trapdoor commitment scheme* TCOM *is secure, additively homomorphic and has uniform keys. For any probabilistic polynomial-time adversary $\mathcal{A}$ that initiates Q_s signature generation protocols by querying $\mathcal{O}_n^{\mathsf{MS}_2}$, and makes Q_h queries to the random oracle $\mathsf{H}_0, \mathsf{H}_3$, the protocol* MS_2 *of Fig. 15 is* $\mathsf{MS}\text{-}\mathsf{UF}\text{-}\mathsf{CMA}$ *secure under* $\mathsf{MSIS}_{q,k,\ell+1,\beta}$ *and* $\mathsf{MLWE}_{q,k,\ell,\eta}$ *assumptions, where $\beta = 2\sqrt{B_n^2 + \kappa}$. Concretely, using other parameters specified in Table 1, the advantage of $\mathcal{A}$ is bounded as follows.*

$$\mathbf{Adv}_{\mathsf{MS}_2}^{\mathsf{MS\text{-}UF\text{-}CMA}}(\mathcal{A}) \leq e \cdot (Q_h + Q_s + 1) \cdot \Bigg((Q_h + Q_s)\epsilon_{td} + Q_s \cdot \frac{e^{-t^2/2}}{M} + \mathbf{Adv}_{\mathsf{MLWE}_{q,k,\ell,\eta}} + \frac{Q_h + Q_s + 1}{|C|} + \sqrt{(Q_h + Q_s + 1) \cdot \left(\epsilon_{bind} + \mathbf{Adv}_{\mathsf{MSIS}_{q,k,\ell+1,\beta}}\right)} \Bigg)$$

Proof. As this proof has a significant overlap with the one for DS_2, we highlight the differences in purple. To avoid redundancy we do not present the resulting reduction algorithm in a separate box, but it should be clear from the hybrid argument below.

Given $\mathcal{A}$ against MS_2 we show that its advantage $\mathbf{Adv}_{\mathsf{MS}_2}^{\mathsf{MS\text{-}UF\text{-}CMA}}(\mathcal{A})$ is negligible by constructing a reduction. Without loss of generality we assume that P_n is an honest party. Our first goal is to construct an algorithm $\mathcal{B}$ around $\mathcal{A}$ that simulates the behaviors of P_n without using honestly generated key pairs. Then we invoke the forking algorithm $\mathcal{F}_{\mathcal{B}}$ from Lemma 5 to obtain two forgeries with distinct challenges, which allow to construct a solution to $\mathsf{MSIS}_{q,k,\ell+1,\beta}$ or to break computational binding of commitment scheme TCOM. Below we discuss how to realize this via several intermediate hybrids.

G_0 **Key generation.** The $\mathcal{B}$ first generates its key pair by invoking $pp \leftarrow \mathsf{Setup}(1^\lambda)$ and $(sk_n, pk_n) := (\mathbf{s}_n, \mathbf{t}_n) \leftarrow \mathsf{Gen}(pp)$. Then $\mathcal{A}$ is given (pp, pk_n) as input.

Random oracle simulation. We assume that $\mathcal{B}$ receives random samples $h_i \leftarrow\!\!\$\, C$ for each $i \in [Q_h + Q_s + 1]$ as input. The random oracles $\mathsf{H}_0 : \{0,1\}^* \to C$ and $\mathsf{H}_3 : \{0,1\}^* \to S_{ck}$ are simulated as follows. The table HT_i is initially empty. The $\mathcal{B}$ also maintains a counter ctr which is initially set to 0. Note that the simulation of H_0 below will come into play when the forking lemma is applied; this way, the adversary $\mathcal{A}$'s view is indeed identical until the forking point in two executions. Concretely, the simulation below follows the proof for Bagherzandi et al. [BCJ08]'s discrete log-based multi-signature BCJ, except that a query to H_3 is interleaved.

```
H_0(x)
    1. Parse $x$ as $(\mathbf{t}, com, \mu, L)$
    2. Make a query $\mathsf{H}_3(\mu, L)$
```

Protocol $\mathsf{MS}_2.\mathsf{Sign}(sid, sk_n, pk_n, \mu, L)$

The protocol is parameterized by public parameters described in Table 1 and matrix $\bar{\mathbf{A}}$, and relies on the random oracles $\mathsf{H}_0 : \{0,1\}^* \to C$ and $\mathsf{H}_3 : \{0,1\}^* \to S_{ck}$. The protocol assumes that $\mathsf{MS}_2.\mathsf{Gen}(pp)$ has been previously invoked. If a party halts with ABORT at any point, then all $\mathsf{Sign}(sid, sk_n, pk_n, \mu, L)$ executions are aborted.

Inputs

1. P_n receives a unique session ID sid, $sk_n = \mathbf{s}_n$, $pk = \mathbf{t}_n$, message $\mu \in \{0,1\}^*$ and a list of public keys L as input. If $n' := |L| > n$ or $\mathbf{t}_n \notin L$ then send out ABORT. Otherwise parse L as $\{\mathbf{t}_1, \ldots, \mathbf{t}_{n'-1}, \mathbf{t}_n\}$.
2. P_n verifies that sid has not been used before (if it has been, the protocol is not executed).
3. P_n locally computes a per-message commitment key $ck \leftarrow \mathsf{H}_3(\mu, L)$.

Signature Generation P_n works as follows:

1. Compute the first message as follows.
   a. Sample $\mathbf{y}_n \leftarrow\$ D_s^{\ell+k}$ and compute $\mathbf{w}_n := \bar{\mathbf{A}}\mathbf{y}_n$.
   b. Compute $com_n \leftarrow \mathsf{Commit}_{ck}(\mathbf{w}_n; r_n)$ with $r_n \leftarrow\$ D(S_r)$.
   c. Send out com_n.
2. Upon receiving com_j for all $j \in [n'-1]$ compute the signature share as follows.
   a. Set $com := \sum_{j \in [n'-1]} com_j + com_n$.
   b. Derive a challenge $c_n \leftarrow \mathsf{H}_0(\mathbf{t}_n, com, \mu, L)$.
   c. Computes a signature share $\mathbf{z}_n := c_n \mathbf{s}_n + \mathbf{y}_n$.
   d. Run the rejection sampling on input $(c_n \mathbf{s}_n, \mathbf{z}_n)$, i.e., with probability
   $$\min\left(1, D_s^{\ell+k}(\mathbf{z}_n)/(M \cdot D_{c_n \mathbf{s}_n, s}^{\ell+k}(\mathbf{z}_n))\right)$$
   send out $(\mathbf{z}_n, r_n)$; otherwise send out RESTART and go to 1.
3. Upon receiving RESTART from some party go to 1. Otherwise upon receiving $(\mathbf{z}_j, r_j)$ for all $j \in [n'-1]$ compute the combined signature as follows
   a. For each $j \in [n'-1]$ derive a per-user challenge $c_j \leftarrow \mathsf{H}_0(\mathbf{t}_j, com, \mu, L)$, reconstruct $\mathbf{w}_j := \bar{\mathbf{A}}\mathbf{z}_j - c_j \mathbf{t}_j$ and validate the signature share:
   $$\|\mathbf{z}_j\|_2 \leq B \quad \text{and} \quad \mathsf{Open}_{ck}(com_j, r_j, \mathbf{w}_j) = 1.$$
   If the check fails for some j then send out ABORT.
   b. Compute $\mathbf{z} := \sum_{j \in [n'-1]} \mathbf{z}_j + \mathbf{z}_n$ and $r := \sum_{j \in [n'-1]} r_j + r_n$.

If the protocol does not abort, P_n obtains a signature $(com, \mathbf{z}, r)$ as local output.

Algorithm $\mathsf{MS}_2.\mathsf{Ver}(com, \mathbf{z}, r, \mu, L)$

Upon receiving a message μ, signature $(com, \mathbf{z}, r)$, and a set of public keys L, if $|L| > n$ then reject the signature. Otherwise for each j such that $\mathbf{t}_j \in L$ derive a per-user challenge $c_j \leftarrow \mathsf{H}_0(\mathbf{t}_j, com, \mu, L)$ and reconstruct $\mathbf{w} := \bar{\mathbf{A}}\mathbf{z} - \sum_j c_j \mathbf{t}_j$. Then accept if $\|\mathbf{z}\|_2 \leq B_n$ and $\mathsf{Open}_{ck}(com, r, \mathbf{w}) = 1$.

Fig. 15. Two-round multi-signature secure in the plain public key model.

3. If $\mathrm{HT}_0[\mathbf{t}, com, \mu, L] = \bot$:
   (a) If $\mathbf{t}, \mathbf{t}_n \in L$:
      * For each $\mathbf{t}_j \in L \setminus \{\mathbf{t}_n\}$, set $\mathrm{HT}_0[\mathbf{t}_j, com, \mu, L] \leftarrow_\$ C$.
      * Increment ctr and $\mathrm{HT}_0[\mathbf{t}_n, com, \mu, L] := h_{ctr}$.
   (b) Otherwise, set $\mathrm{HT}_0[\mathbf{t}, com, \mu, L] \leftarrow_\$ C$
4. Return $\mathrm{HT}_0[\mathbf{t}, com, \mu, L]$

$\underline{\mathsf{H}_3(x)}$ If $\mathrm{HT}_3[\mu, L]$ is not set let $\mathrm{HT}_3[\mu, L] \leftarrow_\$ S_{ck}$. Return $\mathrm{HT}_3[\mu, L]$.

Honest party oracle simulation. In this game $\mathcal{B}$ behaves exactly like a single honest party in MS_2; concretely, it simulates an oracle $\mathcal{O}^{\mathsf{MS}_2}$ (Fig. 4) which internally invokes instructions of Sign according to Fig. 15.

Forgery. When $\mathcal{A}$ outputs a forgery $(com^*, \mathbf{z}^*, r^*, \mu^*, L^*)$ at the end $\mathcal{B}$ proceeds as follows.

1. If $(\mu^*, L^*) \in Mset$ or $|L^*| > n$ then $\mathcal{B}$ halts with output $(0, \bot)$
2. If $\mathbf{t}_n \notin L^*$ then $\mathcal{B}$ halts with output $(0, \bot)$
3. Make queries $ck^* \leftarrow \mathsf{H}_3(\mu^*, L^*)$. Let $n^* := |L^*| \leq n$ and parse L^* as $\{\mathbf{t}_1^*, \ldots, \mathbf{t}_{n^*-1}^*, \mathbf{t}_n\}$. For each $j \in [n^*-1]$, make queries $c_j^* \leftarrow \mathsf{H}_0(\mathbf{t}_j, com^*, \mu^*, L^*)$ and $c_n^* \leftarrow \mathsf{H}_0(\mathbf{t}_n, com^*, \mu^*, L^*)$.
4. If $\mathsf{Open}_{ck}(com^*, r^*, \bar{\mathbf{A}}\mathbf{z}^* - \sum_{j\in[n^*-1]} c_j^* \mathbf{t}_j^* - c_n^* \mathbf{t}_n) \neq 1$ or $\|\mathbf{z}^*\|_2 > B_n$ then $\mathcal{B}$ halts with output $(0, \bot)$.
5. Find $i_\mathrm{f} \in [Q_h + Q_s + 1]$ such that $c_n^* = h_{i_\mathrm{f}}$ and $\mathcal{B}$ halts with output $(i_\mathrm{f}, out = (com^*, \{c^*\}_{j\in[n^*-1]}, c_n^*, \mathbf{z}^*, r^*, \mu^*, L^*, ck^*))$

Let $\Pr[\mathbf{G}_i]$ denote a probability that $\mathcal{B}$ doesn't output $(0, \bot)$ at the game $\mathbf{G}_i$. Then we have

$$\Pr[\mathbf{G}_0] = \mathbf{Adv}^{\mathsf{MS\text{-}UF\text{-}CMA}}_{\mathsf{MS}_2}(\mathcal{A}).$$

$\mathbf{G}_1$ This game is identical to $\mathbf{G}_0$ except at the following points.

Random oracle simulation.

```
H_3(x)  1. Parse $x$ as $(\mu, L)$
        2. If $\mathrm{HT}_3[\mu, L] = \bot$:
           (a) With probability $\varpi$ compute $(tck, td) \leftarrow \mathsf{TCGen}(cpp)$, store the trapdoor in $\mathrm{TDT}[\mu, L] := td$ and set $\mathrm{HT}_3[\mu, L] := tck$.
           (b) With probability $1 - \varpi$, set $\mathrm{HT}_3[\mu, L] := ck$.
        3. Return $\mathrm{HT}_3[\mu, L]$
```

Honest party oracle simulation. The $\mathcal{B}$ differs from the prior one at the following steps in $\mathsf{MS}_2.\mathsf{Sign}_n$.

Inputs 3 Call $\mathsf{H}_3(\mu, L)$ to obtain tck. If $\mathrm{TDT}[\mu, L] = \bot$ (i.e., TCGen was not called) then set a flag bad_4 and halt with output $(0, \bot)$. Otherwise obtain the trapdoor $td \leftarrow \mathrm{TDT}[\mu, L]$

Signature Generation 1.b. Call $com_n \leftarrow \mathsf{TCommit}_{tck}(td)$ instead of committing to $\mathbf{w}_n$.

Signature Generation 2.c. After computing $\mathbf{z}_n := c_n \mathbf{s}_n + \mathbf{y}_n$ derive randomness $r_n \leftarrow \mathsf{Eqv}_{tck}(td, com_n, \mathbf{w}_n)$.

Forgery. When $\mathcal{A}$ outputs a successful forgery $(com^*, \mathbf{z}^*, r^*, \mu^*, L^*)$ at the end, we modify the step 4 of $\mathbf{G}_0$ as follows.

Forgery 4 If $\mathsf{Open}_{ck}(com^*, r^*, \bar{\mathbf{A}}\mathbf{z}^* - \sum_{j\in[n^*-1]} c_j^* \mathbf{t}_j^* - c_n^* \mathbf{t}_n) \neq 1$ or $\|\mathbf{z}^*\|_2 > B_n$ then $\mathcal{B}$ halts with output $(0, \bot)$. *If* $\mathrm{TDT}[\mu^*, L^*] \neq \bot$ *(i.e.,* TCGen *was called for a query* $\mathsf{H}_3(\mu^*, L^*)$*)* then set bad_5 and $\mathcal{B}$ halts with output $(0, \bot)$.

Note that due to the way H_3 is simulated, if $\mathcal{B}$ does not output $(0, \bot)$ it is now guaranteed that $ck^* = ck = \mathsf{H}_3(\mu^*, L^*)$. Recalling that TCOM is secure (Definition 4) we have

$$\Pr[\mathbf{G}_1] \geq \varpi^{Q_h+Q_s} \cdot (1 - \varpi) \cdot \Pr[\mathbf{G}_0] - (Q_h + Q_s) \cdot \epsilon_{\mathrm{td}}$$

because the simulation is only successful if the random oracle H_3 internally uses TCGen for all but one queries to H_3 (both directly and indirectly via H_0 and Sign_n) *and* if H_3 uses a predefined ck

for a single crucial query (μ^*, L^*) associated with forgery; in other words, it is only successful if neither bad_4 nor bad_5 is set above. Note that by setting $\varpi = (Q_h + Q_s)/(Q_h + Q_s + 1)$ since $(1/(1 + 1/(Q_h + Q_s)))^{Q_h + Q_s} \geq 1/e$ for $Q_h + Q_s \geq 0$ we obtain

$$\Pr[\mathbf{G}_1] \geq \frac{\Pr[\mathbf{G}_0]}{e(Q_h + Q_s + 1)} - (Q_h + Q_s) \cdot \epsilon_{\mathrm{td}}.$$

$\mathbf{G}_2$ This game is identical to $\mathbf{G}_1$ except at the following points.

Honest party oracle simulation. The $\mathcal{B}$ doesn't honestly generate $\mathbf{z}_n$ anymore and instead simulates the rejection sampling as follows.

Signature Generation 1.a. $\mathcal{B}$ does nothing here.

Signature Generation 2.c. Sample $\mathbf{z}_n \leftarrow_{\$} D_s^{\ell+k}$ and derive randomness $r_n \leftarrow \mathsf{Eqv}_{tck}(td, com_n, \mathbf{w}_n := \bar{\mathbf{A}}\mathbf{z}_n - c_n \mathbf{t}_n)$.

Signature Generation 2.d. With probability $1/M$ send out $(\mathbf{z}_n, r_n)$. Otherwise send out RESTART.

The signature share $\mathbf{z}_n$ simulated this way is statistically indistinguishable from the real one because of special HVZK property of the underlying identification scheme. In other words, we can directly apply the result of Lemmas 3 and 4. Hence we have

$$|\Pr[\mathbf{G}_2] - \Pr[\mathbf{G}_1]| \leq Q_s \cdot \frac{e^{-t^2/2}}{M}.$$

$\mathbf{G}_3$ At this stage, notice that the simulated signing query responses don't rely on the actual secret key share $\mathbf{s}_n$ anymore. So our next goal is to simulate the generation of $\mathbf{t}_n$ without using $\mathbf{s}_n$.

This game is identical to $\mathbf{G}_3$ except that $\mathcal{B}$ simply picks the random public key share $\mathbf{t}_n \leftarrow_{\$} R_q^k$ during the key generation phase, instead of computing $\mathbf{t}_n = \bar{\mathbf{A}}\mathbf{s}_n$ where $\mathbf{s}_n \leftarrow_{\$} S_\eta^{\ell+k}$. As $\mathbf{A}$ follows the uniform distribution over $R_q^{k\times\ell}$, if the adversary $\mathcal{A}$ can distinguish $\mathbf{G}_2$ and $\mathbf{G}_3$ then we can use $\mathcal{A}$ as a distinguisher that breaks $\mathsf{MLWE}_{q,k,\ell,\eta}$ assumption; hence we have

$$|\Pr[\mathbf{G}_3] - \Pr[\mathbf{G}_2]| \leq \mathbf{Adv}_{\mathsf{MLWE}_{q,k,\ell,\eta}}.$$

Forking lemma. Our goal is to embed a challenge commitment key $ck \leftarrow \mathsf{CGen}(cpp)$ and an instance of $\mathsf{MSIS}_{q,k,\ell+1,\beta}$, which is denoted as $[\mathbf{A}'|\mathbf{I}]$ with $\mathbf{A}' \leftarrow_{\$} R_q^{k\times(\ell+1)}$. As in $\mathbf{G}_3$ the matrix and public key $(\mathbf{A}, \mathbf{t}_n)$ is uniformly distributed in $R_q^{k\times\ell} \times R_q^k$, replacing it with $\mathsf{MSIS}_{q,k,\ell+1,\beta}$ instance doesn't change the view of adversary at all, if $\mathbf{A}'$ is regarded as $\mathbf{A}' = [\mathbf{A}|\mathbf{t}_n]$. Moreover, thanks to the simulation of H_3 it is guaranteed that ck follows the uniform distribution over S_{ck} which is perfectly indistinguishable from honestly generated $ck \leftarrow \mathsf{CGen}(cpp)$ (since the keys are uniform). Hence we define the input generator IGen of forking lemma such that it outputs the instance $(ck, \mathbf{A}, \mathbf{t}_n)$.

Now we prove the theorem by constructing $\mathcal{B}'$ around $\mathcal{B}$ that either (1) breaks binding of commitment wrt ck, or (2) finds a solution to $\mathsf{MSIS}_{q,k,\ell+1,\beta}$ on input $\mathbf{A}' = [\mathbf{A}|\mathbf{t}_n]$. The $\mathcal{B}'$ invokes the forking algorithm $\mathcal{F}_{\mathcal{B}}$ on input $(ck, \mathbf{A}, \mathbf{t}_n)$ from Lemma 5. Then with probability frk we immediately get two forgeries $out = (com^*, \{c^*\}_{j\in[n^*-1]}, c_n^*, \mathbf{z}^*, r^*, \mu^*, L^*, ck^*)$ and $\hat{out} = (\hat{com}^*, \{\hat{c}^*\}_{j\in[\hat{n}^*-1]}, \hat{c}_n^*, \hat{\mathbf{z}}^*, \hat{r}^*, \hat{\mu}^*, \hat{L}^*, \hat{ck}^*)$, where frk satisfies

$$\Pr[\mathbf{G}_3] = \mathrm{acc} \leq \frac{Q_h + Q_s + 1}{|C|} + \sqrt{(Q_h + Q_s + 1) \cdot \mathrm{frk}}.$$

By construction of $\mathcal{B}$ and $\mathcal{F}_{\mathcal{B}}$ we have $com^* = \hat{com}^*$, $\mu^* = \hat{\mu}^*$, $n^* = \hat{n}^*$, $L^* = \hat{L}^* = \{\mathbf{t}_1^*, \ldots, \mathbf{t}_{n^*-1}^*, \mathbf{t}_n\}$, $c_j^* = \hat{c}_j^*$ for each $j \in [n^* - 1]$, and $c_n^* \neq \hat{c}_n^*$; until the forking point $ctr = i_{\mathrm{f}}$ the adversary $\mathcal{A}$'s view is identical in two executions. Moreover, due to the simulation of H_0 we also guarantee that $ck^* = \hat{ck}^* = ck$ since $\mathsf{H}_3(\mu^*, L^*)$ and $\mathsf{H}_3(\hat{\mu}^*, \hat{L}^*)$ should have been invoked right before the fork. Since both forgeries are verified under the same commitment key ck, we have $\|\mathbf{z}^*\|_2 \leq B_n \wedge \|\hat{\mathbf{z}}^*\|_2 \leq B_n$ and

$$\begin{aligned}&\mathsf{Open}_{ck}(com^*, r^*, \bar{\mathbf{A}}\mathbf{z}^* - \sum_{j\in[n^*-1]} c_j^*\mathbf{t}_j^* - c_n^*\mathbf{t}_n)\\ =&\mathsf{Open}_{ck}(com^*, \hat{r}^*, \bar{\mathbf{A}}\hat{\mathbf{z}}^* - \sum_{j\in[n^*-1]} c_j^*\mathbf{t}_j^* - \hat{c}_n^*\mathbf{t}_n) = 1.\end{aligned}$$

If $\bar{\mathbf{A}}\mathbf{z}^* - c_n^*\mathbf{t}_n \neq \bar{\mathbf{A}}\hat{\mathbf{z}}^* - \hat{c}_n^*\mathbf{t}_n$ then $\mathcal{B}'$ can break computational binding with respect to ck, which succeeds with probability at most ϵ_{bind}. If $\bar{\mathbf{A}}\mathbf{z}^* - c_n^*\mathbf{t}_n = \bar{\mathbf{A}}\hat{\mathbf{z}}^* - \hat{c}_n^*\mathbf{t}_n$, rearranging it leads to

$$[\mathbf{A}|\mathbf{I}|\mathbf{t}_n]\begin{bmatrix}\mathbf{z}^* - \hat{\mathbf{z}}^* \\ \hat{c}_n^* - c_n^*\end{bmatrix} = \mathbf{0}.$$

Recalling that $[\mathbf{A}'|\mathbf{I}] = [\mathbf{A}|\mathbf{t}_n|\mathbf{I}]$ is an instance of $\mathsf{MSIS}_{q,k,\ell+1,\beta}$ problem, we have found a valid solution if $\beta = \sqrt{(2B_n)^2 + 4\kappa}$, since $\|\mathbf{z}^* - \hat{\mathbf{z}}^*\|_2 \leq 2B_n$ and $0 < \|\hat{c}_n^* - c_n^*\|_2 \leq \sqrt{4\kappa}$. Putting two cases together, we get

$$\mathrm{frk} \leq \epsilon_{\mathrm{bind}} + \mathbf{Adv}_{\mathsf{MSIS}_{q,k,\ell+1,\beta}}.$$

E Potential Wagner-like Attack on Naive Two-round Protocols

Below we sketch a variant of the concurrent attack originally described by Drijvers et al. [DEF+19]. The original attack was against two-round discrete log-based multi-signatures including CoSi [STV+16] and BCJ [BCJ08], but due to the very similar structure of FSwA lattice signatures an attack would become feasible against naive two-round instantiations (albeit with sub-exponential computational costs due to reliance on a K-list sum algorithm). Since such naive FSwA-based constructions do not exist in the literature, we do not go into details of the efficiency analysis of the concurrent attack. The attack sketched here should be treated as a motivating discussion about why our two-round protocols rely on a message-dependent commitment key in Figs. 5 and 15.

Attack on a naive construction from Section 1.2. For simplicity we consider the attack on two-party signing, but the same strategy also works similarly in a general n-party setting. Let $\tilde{\mathbf{s}}_1$ and $\mathbf{s}_2$ be the key shares of adversary and honest party, respectively, and let $\mathbf{t} = \bar{\mathbf{A}}(\tilde{\mathbf{s}}_1 + \mathbf{s}_2)$ be the combined public key. The adversary initiates $K-1$ concurrent signing sessions on the same message μ. Then for each session $i \in [K-1]$, the honest party submits $\mathbf{w}_2^{(i)} = \bar{\mathbf{A}}\mathbf{y}_2^{(i)}$. Here the adversary does not immediately send back its own commitment share. Instead, by only interacting with the random oracle H_0 the adversary tries to find a message μ^* and $\tilde{\mathbf{w}}_1^{(1)}, \ldots, \tilde{\mathbf{w}}_1^{(K-1)} \in R_q^k$ such that the following holds.

$$c^* := \mathsf{H}_0(\mathbf{w}^*, \mu^*, \mathbf{t}) = \mathsf{H}_0(\tilde{\mathbf{w}}_1^{(1)} + \mathbf{w}_2^{(1)}, \mu, \mathbf{t}) + \ldots + \mathsf{H}_0(\tilde{\mathbf{w}}_1^{(K-1)} + \mathbf{w}_2^{(K-1)}, \mu, \mathbf{t})$$

where the adversary defines $\mathbf{w}^* := \mathbf{w}_2^{(1)} + \ldots + \mathbf{w}_2^{(K-1)}$. Because the random oracle outputs consist of $C = \left\{c \in \mathbb{Z}^N : \|c\|_1 = \kappa \wedge \|c\|_\infty = 1\right\}$, finding such inputs amounts to solving a variant of Wagner's generalized birthday problem (GBP) [Wag02, HJ10] instantiated over $(C, +)$, when K is chosen to be a power of two. (Note that in the discrete log setting GBP is instantiated over a group $(\mathbb{Z}_q, +)$.) Then the adversary resumes the pending sessions by sending back such $\tilde{\mathbf{w}}_1^{(i)}$ for $i \in [K-1]$. The honest signer for each session returns its signature share

$$\mathbf{z}_2^{(i)} = \mathbf{y}_2^{(i)} + c^{(i)}\mathbf{s}_2$$

where $c^{(i)} = \mathsf{H}_0(\tilde{\mathbf{w}}_1^{(i)} + \mathbf{w}_2^{(i)}, \mu, \mathbf{t})$. Finally the adversary outputs

$$\mathbf{z}^* = \mathbf{z}_2^{(1)} + \ldots + \mathbf{z}_2^{(K-1)} + c^*\tilde{\mathbf{s}}_1 = \mathbf{y}_2^{(1)} + \ldots + \mathbf{y}_2^{(K-1)} + c^*(\tilde{\mathbf{s}}_1 + \mathbf{s}_2)$$

as a forgery on μ^* together with $\mathbf{w}^*$. Now let us check that $(\mathbf{w}^*, \mathbf{z}^*)$ satisfies the verification condition. Thanks to the collision found by a GBP solver, and by construction of $\mathbf{w}^*$ and $\mathbf{z}^*$, it holds that $\bar{\mathbf{A}}\mathbf{z}^* - c^*\mathbf{t} = \mathbf{w}^*$. Note that the adversary should take extra care of the norm of $\mathbf{z}^*$ by bounding the number of sessions $K-1$; the small $\|\mathbf{z}^*\|$ is part of the verification condition, while $\|\mathbf{z}^*\|$ grows for large K. For this reason there should be some tradeoffs for K, since the larger the K is, the lower the complexity of GBP algorithm becomes. This means that, since the norm bound in verification has to be increased according to n (see Section 3.2), the attack also becomes efficient in an n-party setting, which allows to choose larger K when $n-1$ parties are corrupt.

We also remark that the attack can be completed only when the honest party passes the rejection sampling simultaneously in all $K-1$ concurrent sessions, because otherwise the attacker doesn't receive all $\mathbf{z}_2^{(i)}$ values required for forgery. Hence there is another tradeoff here: if the success rate of rejection sampling is set low then the protocol has more round complexity, while it mitigates the concurrent attack, and vice versa.

Attack on a variant of DS_2 with fixed commitment key. When a single commitment key ck is reused for all signing attempts in DS_2 (Fig. 5) then a similar concurrent attack becomes applicable. This time for each session $i \in [K-1]$, the honest party submits $com_2^{(i)} = \mathsf{Commit}_{ck}(\mathbf{w}_2^{(i)}; r_2^{(i)})$ where $\mathbf{w}_2^{(i)} = \bar{\mathbf{A}}\mathbf{y}_2^{(i)}$. Then the adversary interacts with the random oracle H_0 to find a message μ^* and $\tilde{com}_1^{(1)}, \ldots, \tilde{com}_1^{(K-1)} \in S_{com}$ such that the following holds (with a GBP solver).

$$c^* := \mathsf{H}_0(com^*, \mu^*, \mathbf{t}) = \mathsf{H}_0(\tilde{com}_1^{(1)} + com_2^{(1)}, \mu, \mathbf{t}) + \ldots + \mathsf{H}_0(\tilde{com}_1^{(K-1)} + com_2^{(K-1)}, \mu, \mathbf{t})$$

where the adversary defines $com^* := com_2^{(1)} + \ldots + com_2^{(K-1)}$.

Then the adversary resumes the pending sessions by sending back such $\tilde{com}_1^{(i)}$ for $i \in [K-1]$. The honest signer for each session returns its signature share together with commitment opening $r_2^{(i)}$

$$\mathbf{z}_2^{(i)} = \mathbf{y}_2^{(i)} + c^{(i)}\mathbf{s}_2$$

where $c^{(i)} = \mathsf{H}_0(\tilde{com}_1^{(i)} + com_2^{(i)}, \mu, \mathbf{t})$. Finally the adversary outputs

$$\begin{aligned}\mathbf{z}^* &= \mathbf{z}_2^{(1)} + \ldots + \mathbf{z}_2^{(K-1)} + c^*\tilde{\mathbf{s}}_1 = \mathbf{y}_2^{(1)} + \ldots + \mathbf{y}_2^{(K-1)} + c^*(\tilde{\mathbf{s}}_1 + \mathbf{s}_2)\\ r^* &= r_2^{(1)} + \ldots + r_2^{(K-1)}\end{aligned}$$

as a forgery on μ^* together with com^*. Thanks to the collision found by a GBP solver and due to the additive homomorphism of commitment, it holds that $\mathsf{Open}_{ck}(com^*, r^*, \bar{\mathbf{A}}\mathbf{z}^* - c^*\mathbf{t}) = 1$.

If the protocol derives a per-message commitment key via random oracle $\mathsf{H}_3 : \{0,1\}^* \to S_{ck}$ as our protocols (as well as mBCJ) do, the attack becomes nontrivial; now the tuple $(com^*, r^*, \bar{\mathbf{A}}\mathbf{z}^* - c^*\mathbf{t})$ has to be verified with respect to the message-dependent key $ck^* \leftarrow \mathsf{H}_3(\mu^*, \mathbf{t})$, which of course shouldn't collide with $ck \leftarrow \mathsf{H}_3(\mu, \mathbf{t})$ thanks to the random oracle.

F DS_3: Three-round Distributed Signature Protocol from Module-LWE

F.1 Protocol Specification and Overview

As an important stepping stone towards our main two-round constructions, we give a detailed description of provably secure three-round, n-out-of-n distributed signature protocol $\mathsf{DS}_3 = (\mathsf{Setup}, (\mathsf{Gen}_j)_{j\in[n]}, (\mathsf{Sign})_{j\in[n]}, \mathsf{Ver})$, formally specified in Fig. 16. Key generation and verification are identical to DS_2 (see Fig. 5). The protocol is built on top of additively homomorphic commitment scheme $\mathsf{COM} = (\mathsf{CSetup}, \mathsf{CGen}, \mathsf{Commit}, \mathsf{Open})$ with uniform key (see Section 2 for the formal definition), and we describe concrete instances of COM in Section 4.

The only difference from DS_2 is that, the signing protocol now involves an extra round where participants exchange a hash of com_j, and later check that everyone knows the correct preimage. This is a standard technique used in Bellare and Neven [BN06] (or its GLP-based variant [BS16]) The intuition behind this seemingly redundant step is analogous to the rogue key attack; without this step the adversary might be able to adaptively choose a malicious $\widetilde{com}$ after seeing the honest party's share. However, this extra round can be indeed dropped by instantiating the protocol with a trapdoor commitment scheme (see Section 3). We remark that generating a per-message commitment key as in 3 is not necessary for the three-round protocol and one could alternatively use a single fixed $ck \leftarrow \mathsf{CGen}(cpp)$ generated by the trusted party. However, this step becomes crucial for the two-round protocols to be secure.

F.2 Security

We give a security proof for DS_3 by instantiating the protocol with an unconditionally binding commitment scheme and by setting the parameters so that the underlying SIS problem becomes vacuously hard. Here we give a sketch of our proof. First, thanks to the computational hiding of COM, the oracle simulator can replace the commitment share com_n of honest party P_n with a commitment to some random vector in R_q^k in case it wants to abort. For non-abort executions we can essentially invoke the special HVZK simulator of Algorithm 7 to answer the oracle queries from the adversary. Hence we can indeed simulate the honest execution of P_n.

The core idea for proving the soundness essentially follows the lossy identification technique by Abdalla et al. [AFLT16]; since the public key share of the honest signer $\mathbf{t}_n$ is indistinguishable from the vector

Protocol $\mathsf{DS}_3.\mathsf{Sign}_n(sid, sk_n, pk, \mu)$

The protocol is parameterized by public parameters described in Table 1 and relies on the random oracles $\mathsf{H}_0 : \{0,1\}^* \to C$, $\mathsf{H}_3 : \{0,1\}^* \to S_{ck}$ and $\mathsf{H}_4 : \{0,1\}^* \to \{0,1\}^{l_4}$. The protocol assumes that $\mathsf{DS}_3.\mathsf{Gen}_n(pp)$ has been previously invoked. If a party halts with ABORT at any point, then all $\mathsf{Sign}_n(sid, sk_n, pk, \mu)$ executions are aborted.

Inputs

1. P_n receives a unique session ID sid, $sk_n = \mathbf{s}_n$, $pk = (\bar{\mathbf{A}}, \mathbf{t})$ and message $\mu \in \{0,1\}^*$ as input.
2. P_n verifies that sid has not been used before (if it has been, the protocol is not executed).
3. P_n locally computes a per-message commitment key $ck \leftarrow \mathsf{H}_3(\mu, \mathbf{t})$.

Signature Generation P_n works as follows:

1. Compute the first message as follows.
   a. Sample $\mathbf{y}_n \leftarrow\!\!\$\, D_s^{\ell+k}$ and compute $\mathbf{w}_n := \bar{\mathbf{A}}\mathbf{y}_n$.
   b. Compute $com_n \leftarrow \mathsf{Commit}_{ck}(\mathbf{w}_n; r_n)$ with $r_n \leftarrow\!\!\$\, D(S_r)$, and $h_n \leftarrow \mathsf{H}_4(com_n)$.
   c. Send out h_n.
2. Upon receiving h_j for all $j \in [n-1]$ send out com_n.
3. Upon receiving com_j for all $j \in [n-1]$ compute the signature share as follows.
   a. If $\mathsf{H}_4(com_j) \neq h_j$ for some $j \in [n-1]$ send out ABORT.
   b. Set $com := \sum_{j\in[n]} com_j$.
   c. Derive a challenge $c \leftarrow \mathsf{H}_0(com, \mu, \mathbf{t})$.
   d. Computes a signature share $\mathbf{z}_n := c\mathbf{s}_n + \mathbf{y}_n$.
   e. Run the rejection sampling on input $(c\mathbf{s}_n, \mathbf{z}_n)$, i.e., with probability

   $$\min\left(1, D_s^{\ell+k}(\mathbf{z}_n)/(M \cdot D_{c\mathbf{s}_n,s}^{\ell+k}(\mathbf{z}_n))\right)$$

   send out $(\mathbf{z}_n, r_n)$; otherwise send out RESTART and go to 1.
4. Upon receiving RESTART from some party go to 1. Otherwise upon receiving $(\mathbf{z}_j, r_j)$ for all $j \in [n-1]$ compute the combined signature as follows
   a. For each $j \in [n-1]$ reconstruct $\mathbf{w}_j := \bar{\mathbf{A}}\mathbf{z}_j - c\mathbf{t}_j$ and validate the signature share:

   $$\|\mathbf{z}_j\|_2 \leq B \quad \text{and} \quad \mathsf{Open}_{ck}(com_j, r_j, \mathbf{w}_j) = 1.$$

   If the check fails for some j then send out ABORT.
   b. Compute $\mathbf{z} := \sum_{j\in[n]} \mathbf{z}_j$ and $r := \sum_{j\in[n]} r_j$.

If the protocol does not abort, P_n obtains a signature $(com, \mathbf{z}, r)$ as local output.

Fig. 16. Three-round distributed signing protocol.

sampled from R_q^k uniformly at random due to the LWE assumption, the oracle simulator can replace $\mathbf{t}_n$ with such a vector (i.e. a *lossy key*). Moreover, thanks to the programmability and extractability of random oracle commitments in the key generation phase, the oracle simulator can even sample the resulting combined public key $\mathbf{t}$ from the uniform distribution *in advance* and set its share $\mathbf{t}_n$ a posteriori depending on the other shares. Now, the unconditional binding of COM guarantees that there cannot *exist* commitments having two openings except a negligible fraction on the random choice of ck. (See Definition 9. Also recall that we defined a uniform key in Definition 10, so the keys given by the random oracle are perfectly indistinguishable from the ones from CGen.) Finally, we argue that on the random choice of joint public key $(\mathbf{A}, \mathbf{t})$ there cannot *exist* two valid transcripts that share the first message of the underlying Σ-protocol (i.e. $\mathbf{w} \in R_q^k$) except a negligible fraction. In that case, the only way for an adversary to come up with a forgery is to luckily receive a specific challenge $c \in C$ from the random oracle H_0, which can only happen with probability $1/|C|$.

The last step of the security proof essentially follows the one for Dilithium-QROM given by Kiltz, Lyubashevsky and Schaffner [KLS18], and we impose an additional condition on the modulus q so that the polynomials of small norm are invertible in the ring R_q [LS18][6]. We also remark that Fukumitsu and Hasegawa [FH19] also attempted a tight security reduction for their Dilithium-like three-round multi-signature scheme, although the proof disregards the simulation of rejected transcripts. Since the rest of their proof seems sound, by applying an additively homomorphic commitment as we do one could patch the proof, while losing the reduction tightness due to the use of computational hiding of commitment scheme.

Theorem 4. *Suppose the commitment scheme* COM *is unconditionally binding, computationally hiding, uniform, additively homomorphic, and the output of committing algorithm* Commit *has ξ-bit min-entropy. Assume the modulus q satisfies $q = 5 \mod 8$, $2B_n < \sqrt{q/2}$ and $2\kappa < \sqrt{q/2}$. For any probabilistic polynomial-time adversary $\mathcal{A}$ that initiates a single key generation protocol by querying $\mathcal{O}_n^{\mathsf{DS}_3}$ with $sid = 0$, initiates Q_s signature generation protocols by querying $\mathcal{O}_n^{\mathsf{DS}_3}$ with $sid \neq 0$, and makes Q_h queries to the random oracle $\mathsf{H}_0, \mathsf{H}_1, \mathsf{H}_2, \mathsf{H}_3, \mathsf{H}_4$, the protocol $\mathsf{DS}_3 = (\mathsf{Setup}, (\mathsf{Gen}_j)_{j\in[n]}, (\mathsf{Sign})_{j\in[n]}, \mathsf{Ver})$ is* $\mathsf{DS\text{-}UF\text{-}CMA}$ *secure under* $\mathsf{MLWE}_{q,k,\ell,\eta}$ *assumption.*

Proof. Suppose we are given $\mathcal{A}$ that breaks DS_3 with advantage $\mathbf{Adv}_{\mathsf{DS}_3}^{\mathsf{DS\text{-}UF\text{-}CMA}}(\mathcal{A})$. Without loss of generality we assume that P_n is an honest party. Our first goal is to construct an algorithm $\mathcal{B}$ around $\mathcal{A}$ that simulates the behaviors of P_n without using honestly generated key pairs. In Fig. 17 we present the resulting oracle simulator $\mathsf{SimSign}_n$ which are eventually invoked by $\mathcal{B}$. The simulation of key generation is done just as in the proof for DS_2 (see Fig. 13). Below we discuss how these are derived via several intermediate hybrid games.

$\mathbf{G}_0$ **Random Oracle simulation.** The random oracles $\mathsf{H}_0 : \{0,1\}^* \to C$, $\mathsf{H}_1 : \{0,1\}^* \to \{0,1\}^{l_1}$, $\mathsf{H}_2 : \{0,1\}^* \to \{0,1\}^{l_2}$, $\mathsf{H}_3 : \{0,1\}^* \to S_{ck}$ and $\mathsf{H}_4 : \{0,1\}^* \to \{0,1\}^{l_4}$ are are simulated as follows.

$\mathsf{H}_i(x)$ The table HT_i is initially empty. When queried with x, if $\mathrm{HT}_i[x]$ is set then return $\mathrm{HT}_i[x]$. Otherwise sample y from H_i's image uniformly at random and return $\mathrm{HT}_i[x] := y$.

Honest party oracle simulation. In this game $\mathcal{B}$ behaves exactly like a single honest party in DS_3; concretely, it simulates an oracle $\mathcal{O}_n^{\mathsf{DS}_3}$ (Fig. 3) which internally invokes instructions of Gen_n and Sign_n according to Fig. 5 and Fig. 16, respectively.

Forgery. When $\mathcal{A}$ outputs a forgery $(\mu^*, com, \mathbf{z}, r)$ at the end $\mathcal{B}$ first generates a commitment key $ck \leftarrow \mathsf{H}_3(\mu^*, \mathbf{t})$, derives a challenge $c \leftarrow \mathsf{H}_0(com, \mu^*, \mathbf{t})$ and reconstructs $\mathbf{w} = \bar{\mathbf{A}}\mathbf{z} - c\mathbf{t}$. Then $\mathcal{B}$ checks $\mu^* \notin Mset$ and the verification condition

$$\|\mathbf{z}\|_2 \leq B_n \quad \text{and} \quad \mathsf{Open}_{ck}(com, r, \mathbf{w}) = 1.$$

If the forgery is verified then $\mathcal{B}$ outputs 1. Otherwise $\mathcal{B}$ outputs 0. Let $\Pr[\mathbf{G}_i]$ denote a probability that $\mathcal{B}$ returns 1 at the game $\mathbf{G}_i$. Then we have

$$\Pr[\mathbf{G}_0] = \mathbf{Adv}_{\mathsf{DS}_3}^{\mathsf{DS\text{-}UF\text{-}CMA}}(\mathcal{A}).$$

$\mathbf{G}_1$ In this game we modify $\mathcal{B}$ from the prior game so that it first picks a random challenge $c \leftarrow_\$ C$ and computes its own signature share $\mathbf{z}_n$ without interacting with adversary. Then the oracle proceeds as in the previous game and sends out hash commitment h_n. Upon receiving $h_1, \ldots, h_{n-1}$, the oracle

[6] This condition could be actually relaxed somewhat by applying the result due to Nguyen [Ngu19]

searches the hash table HT_0 to check if there exists the corresponding preimages $com_1, \ldots, com_{n-1}$. If it's successful, then let $com = \sum_j com_j$ and program the random oracle so that $\mathrm{HT}_0[com, \mu, \mathbf{t}] := c$. Otherwise simulation fails. Since $\mathbf{G}_1$ is identical to $\mathbf{G}_0$ from adversary $\mathcal{A}$'s point of view except at the *bad* events marked in Fig. 17, we have

$$\begin{aligned} |\Pr[\mathbf{G}_1] - \Pr[\mathbf{G}_0]| &\leq \Pr[bad_4] + \Pr[bad_5] + \Pr[bad_6] \\ &\leq \frac{(Q_h + nQ_s + 1)^2}{2^{l_4+1}} + Q_s \left(\frac{Q_h + nQ_s}{2^\xi} + \frac{Q_h + Q_s}{2^\xi} + \frac{n}{2^{l_4}} \right) \end{aligned}$$

where $\Pr[bad_4]$ corresponds to the probability that at least one collision occurs during at most $Q_h + nQ_s$ queries to H_4 made by $\mathcal{A}$ or $\mathcal{B}$; $\Pr[bad_5]$ is the probability that programming the random oracle H_0 fails at least once during Q_s trials due to either of two cases: 1) $\mathsf{H}_4(com_n)$ has been asked by $\mathcal{A}$ during at most $Q_h + nQ_s$ queries to H_4 (and therefore $\mathcal{A}$ knows com and could query $\mathsf{H}_0(com, \mu, \mathbf{t})$ deliberately), which could succeed with probability at most $1/2^\xi$ for each query, or 2) $\mathrm{HT}_0[com, \mu, \mathbf{t}]$ has been set by $\mathcal{A}$ or $\mathcal{B}$ by chance during at most $Q_h + Q_s$ prior queries to H_0, which could happen with probability at most $(Q_h + Q_s)/2^\xi$; $\Pr[bad_6]$ is the probability that $\mathcal{A}$ has predicted one of the $n-1$ outputs of random oracle H_4 without making a query to it, which could only happen with probability at most $n/2^{l_4}$ for each sign query. We remark that the above probability bound is essentially a special case of the one given by [BN06].

$\mathbf{G}_2$ In this game we modify $\mathcal{B}$ from the prior game so that if $\mathbf{z}_n$ gets rejected then it commits to some uniformly random vector $\mathbf{w}_n \in R_q^k$ and sends out hash of corresponding commitment $h_n = \mathsf{H}_4(com_n)$, where $com_n \leftarrow \mathsf{Commit}_{ck}(\mathbf{w}_n; r_n)$ and $r_n \leftarrow\!\!\$\, D(S_r)$. Note that the adversary cannot distinguish this simulated com_n from the real one due to the hiding property of commitment. In other words, we have

$$|\Pr[\mathbf{G}_2] - \Pr[\mathbf{G}_1]| \leq Q_s \cdot \epsilon_{\mathrm{hide}}.$$

$\mathbf{G}_3$ In this game $\mathcal{B}$ doesn't honestly generate $\mathbf{z}_n$ anymore and instead simulates the rejection sampling as follows. With probability $1 - 1/M$ (i.e., simulation of rejection), it generates commitment com_n to $\mathbf{w}_n \leftarrow\!\!\$\, R_q^k$ as before. Otherwise it samples $\mathbf{z}_n$ from $D_s^{\ell+k}$ and computes $\mathbf{w}_n = \bar{\mathbf{A}}\mathbf{z}_n - c\mathbf{t}_n$. The signature share $\mathbf{z}_n$ generated this way is indistinguishable from the real one because of the special HVZK property of the underlying identification scheme. In other words, we can directly apply the result of Lemmas 3 and 4. Hence we have

$$|\Pr[\mathbf{G}_3] - \Pr[\mathbf{G}_2]| \leq Q_s \cdot \frac{e^{-t^2/2}}{M}.$$

At this point $\mathcal{B}$ simulates the honest party's behavior during signature generation by following $\mathsf{SimSign}_n$ in Fig. 17.

$\mathbf{G}_4$ Now notice that signing phase doesn't rely on the actual secret key share $\mathbf{s}_n$ anymore. So the next step is to simulate the key generation phase without using $\mathbf{s}_n$. This can be done just as in Fig. 13 used for the security proof of two-round protocol, and hence

$$\begin{aligned} |\Pr[\mathbf{G}_4] - \Pr[\mathbf{G}_3]| \leq \mathbf{Adv}_{\mathsf{MLWE}_{q,k,\ell,\eta}} &+ \frac{(Q_h+1)Q_h}{2^{l_1+1}} + \frac{Q_h}{q^{k\ell N}} + \frac{n}{2^{l_1}} \\ &+ \frac{(Q_h+1)Q_h}{2^{l_2+1}} + \frac{Q_h}{q^{kN}} + \frac{n}{2^{l_2}}. \end{aligned}$$

Now $\mathcal{B}$ entirely simulates the behaviors of honest party by invoking SimGen_n (Fig. 13) and $\mathsf{SimSign}_n$ (Fig. 17), which don't rely on the secret key share $\mathbf{s}_n$. We would like to evaluate the upper bound of $\Pr[\mathbf{G}_4]$. We first argue that the following probability is negligible.

$$\Pr_{\mathbf{A} \leftarrow\!\!\$\, R_q^{k\times\ell}, \mathbf{t} \leftarrow\!\!\$\, R_q^k, ck \leftarrow\!\!\$\, S_{ck}} \left[\exists (com, c, \mathbf{z}, r, c', \mathbf{z}', r') : \begin{array}{c} c \neq c' \wedge \|\mathbf{z}\|_2 \leq B_n \wedge \|\mathbf{z}'\|_2 \leq B_n \\ \wedge\ \mathsf{Open}_{ck}(com, r, \bar{\mathbf{A}}\mathbf{z} - c\mathbf{t}) \\ = \mathsf{Open}_{ck}(com, r', \bar{\mathbf{A}}\mathbf{z}' - c'\mathbf{t}) = 1 \end{array} \right] \quad (1)$$

Case 1: $\bar{\mathbf{A}}\mathbf{z} - c\mathbf{t} \neq \bar{\mathbf{A}}\mathbf{z}' - c'\mathbf{t}$. In this case, (1) is bounded by the probability that there exists some commitment having two openings over random choice of ck, which should be bounded by negligible $\epsilon_{\mathrm{ubind}}$ if the commitment key is uniform (and hence $ck \leftarrow\!\!\$\, S_{ck}$ can be regarded as if it was generated from CGen) and if unconditionally binding holds (see Definition 9).

Case 2: $\bar{\mathbf{A}}\mathbf{z} - c\mathbf{t} = \bar{\mathbf{A}}\mathbf{z}' - c'\mathbf{t}$. Let $\mathbf{z}_1 \in R_q^\ell, \mathbf{z}_2 \in R_q^k, \mathbf{z}_1' \in R_q^\ell, \mathbf{z}_2' \in R_q^k$ be such that $\mathbf{z} = \begin{bmatrix}\mathbf{z}_1\\ \mathbf{z}_2\end{bmatrix}$ and $\mathbf{z}' = \begin{bmatrix}\mathbf{z}_1'\\ \mathbf{z}_2'\end{bmatrix}$, we have

$$\mathbf{A}(\mathbf{z}_1 - \mathbf{z}_1') + \mathbf{z}_2 - \mathbf{z}_2' = (c - c')\mathbf{t}$$

where we used the fact that $\bar{\mathbf{A}} = [\mathbf{A}|\mathbf{I}]$. Hence, the probability (1) in this case is bounded by

$$2 \cdot |\bar{C}| \cdot \frac{(4B_n + 1)^{(\ell+k)N}}{q^{kN}}$$

by applying Lemma 6 with $\bar{\mathbf{z}} = \mathbf{z} - \mathbf{z}'$, $\bar{c} = c - c'$, $\beta = 2B_n$, and

$$\bar{C} = \{\bar{c} \in R : \bar{c} = c - c' \wedge c \in C \wedge c' \in C \wedge c \neq c'\}.$$

If the event for (1) doesn't occur, then it means that for given $com \in S_{com}$ there exists at most one transcript that verifies. In that case $\mathcal{A}$ has at most a $1/|C|$ chance of obtaining the correct challenge for each query to H_0 with input $(com, \mu^*, \mathbf{t})$ if $\mu^* \notin Mset$.

Since $\mathcal{A}$ makes at most Q_h queries to H_0 and H_3 in total and $\mathcal{B}$ makes a single query to H_0 and H_3 at the forgery phase, we have

$$\Pr[\mathbf{G}_4] \leq (Q_h + 1)\left(\epsilon_{\text{ubind}} + 2 \cdot |\bar{C}| \cdot \frac{(4B_n + 1)^{(\ell+k)N}}{q^{kN}} + \frac{1}{|C|}\right).$$

The following lemma is a slightly modified version of Lemma 4.6 of [KLS18]. The main difference is that we use the Euclidean norm instead of ∞-norm.

Lemma 6. *Let β be a positive integer less than $\sqrt{q/2}$ and $\bar{C}$ be a set of elements in $R \setminus \{\mathbf{0}\}$ with coefficients less than $\sqrt{q/2}$. If $q = 5 \mod 8$ then*

$$\Pr_{\mathbf{A} \leftarrow\$ R_q^{k\times\ell}, \mathbf{t} \leftarrow\$ R_q^k}[\exists(\bar{\mathbf{z}}_1, \bar{\mathbf{z}}_2, \bar{c}) \in R_q^\ell \times R_q^k \times \bar{C} : \mathbf{A}\bar{\mathbf{z}}_1 + \bar{\mathbf{z}}_2 = \bar{c}\mathbf{t} \wedge \|\bar{\mathbf{z}}\|_2 \leq \beta] \leq 2 \cdot |\bar{C}| \cdot \frac{(2\beta+1)^{(\ell+k)N}}{q^{kN}}$$

where $\bar{\mathbf{z}} = \begin{bmatrix}\bar{\mathbf{z}}_1\\ \bar{\mathbf{z}}_2\end{bmatrix}$.

Proof. **Case $\bar{\mathbf{z}}_1 = \mathbf{0}$.** Since $0 \leq \|\bar{c}\|_\infty \leq \sqrt{q/2}$ and $q = 5 \mod 8$, Lemma 2.2 by Lyubashevsky and Seiler [LS18] guarantees that $\bar{c}$ is invertible in R_q. In this case the probability is upper-bounded by

$$\begin{aligned}
&\Pr_{\mathbf{t} \leftarrow\$ R_q^k}[\exists(\bar{\mathbf{z}}_2, \bar{c}) \in R_q^k \times \bar{C} : \bar{\mathbf{z}}_2 = \bar{c}\mathbf{t} \wedge \|\bar{\mathbf{z}}_2\|_2 \leq \beta]\\
=&\Pr_{\mathbf{t} \leftarrow\$ R_q^k}[\exists(\bar{\mathbf{z}}_2, \bar{c}) \in R_q^k \times \bar{C} : \bar{c}^{-1}\bar{\mathbf{z}}_2 = \mathbf{t} \wedge \|\bar{\mathbf{z}}_2\|_2 \leq \beta]\\
\leq&\sum_{\bar{\mathbf{z}}_2 \in R_q^k, \bar{c} \in \bar{C}} \Pr_{\mathbf{t} \leftarrow\$ R_q^k}[\bar{c}^{-1}\bar{\mathbf{z}}_2 = \mathbf{t} \wedge \|\bar{\mathbf{z}}_2\|_2 \leq \beta]\\
\leq&\sum_{\bar{\mathbf{z}}_2 \in R_q^k, \bar{c} \in \bar{C}} \Pr_{\mathbf{t} \leftarrow\$ R_q^k}[\bar{c}^{-1}\bar{\mathbf{z}}_2 = \mathbf{t} \wedge \|\bar{\mathbf{z}}_2\|_\infty \leq \beta]\\
=&|\bar{C}| \cdot \left(\frac{2\beta+1}{q}\right)^{kN}
\end{aligned}$$

Case $\bar{\mathbf{z}}_1 \neq \mathbf{0}$. Let $\mathbf{a} \in R_q^k, \mathbf{A}' \in R_q^{k\times(\ell-1)}$ be such that $[\mathbf{a}|\mathbf{A}'] = \mathbf{A}$ and $\bar{z} \in R_q, \bar{\mathbf{z}}_1' \in R_q^{\ell-1}$ be such that $\begin{bmatrix}\bar{z}\\ \bar{\mathbf{z}}_1'\end{bmatrix} = \bar{\mathbf{z}}_1$. Assuming wlog that $\bar{z}$ is non-zero, it is guaranteed that $\bar{z}$ is invertible in R_q since

$\|\bar{z}\|_\infty \leq \|\bar{z}\|_2 \leq \beta \leq \sqrt{q/2}$. Hence we obtain the following upper-bound.

$$\begin{aligned}
&\Pr_{\mathbf{A} \leftarrow\$ R_q^{k\times\ell}, \mathbf{t} \leftarrow\$ R_q^k}[\exists(\bar{\mathbf{z}}_1, \bar{\mathbf{z}}_2, \bar{c}) \in R_q^\ell \times R_q^k \times \bar{C} : \mathbf{A}\bar{\mathbf{z}}_1 + \bar{\mathbf{z}}_2 = \bar{c}\mathbf{t} \wedge \|\bar{\mathbf{z}}\|_2 \leq \beta] \\
=& \Pr_{\mathbf{a} \leftarrow\$ R_q, \mathbf{A}' \leftarrow\$ R_q^{k\times(\ell-1)}, \mathbf{t} \leftarrow\$ R_q^k}[\exists(\bar{z}, \bar{\mathbf{z}}_1', \bar{\mathbf{z}}_2, \bar{c}) \in R_q \times R_q^{\ell-1} \times R_q^k \times \bar{C} : \bar{z}\mathbf{a} + \mathbf{A}'\bar{\mathbf{z}}_1' + \bar{\mathbf{z}}_2 = \bar{c}\mathbf{t} \wedge \|\bar{\mathbf{z}}\|_2 \leq \beta] \\
=& \Pr_{\mathbf{a} \leftarrow\$ R_q}[\exists(\bar{z}, \bar{\mathbf{z}}_1', \bar{\mathbf{z}}_2, \bar{c}) \in R_q \times R_q^{\ell-1} \times R_q^k \times \bar{C} : \mathbf{a} = \bar{z}^{-1}(\bar{c}\mathbf{t} - \mathbf{A}'\bar{\mathbf{z}}_1' - \bar{\mathbf{z}}_2) \wedge \|\bar{\mathbf{z}}\|_2 \leq \beta] \\
\leq& \sum_{\bar{\mathbf{z}}_1 \in R_q^\ell \setminus \{\mathbf{0}\}, \bar{\mathbf{z}}_2 \in R_q^k, \bar{c} \in \bar{C}} \Pr[\mathbf{a} = \bar{z}^{-1}(\bar{c}\mathbf{t} - \mathbf{A}'\bar{\mathbf{z}}_1' - \bar{\mathbf{z}}_2) \wedge \|\bar{\mathbf{z}}\|_2 \leq \beta] \\
\leq& \sum_{\bar{\mathbf{z}}_1 \in R_q^\ell \setminus \{\mathbf{0}\}, \bar{\mathbf{z}}_2 \in R_q^k, \bar{c} \in \bar{C}} \Pr[\mathbf{a} = \bar{z}^{-1}(\bar{c}\mathbf{t} - \mathbf{A}'\bar{\mathbf{z}}_1' - \bar{\mathbf{z}}_2) \wedge \|\bar{\mathbf{z}}\|_\infty \leq \beta] \\
\leq& \sum_{\bar{\mathbf{z}}_1 \in R_q^\ell \setminus \{\mathbf{0}\}, \bar{\mathbf{z}}_2 \in R_q^k, \bar{c} \in \bar{C}} \Pr[\mathbf{a} = \bar{z}^{-1}(\bar{c}\mathbf{t} - \mathbf{A}'\bar{\mathbf{z}}_1' - \bar{\mathbf{z}}_2) \wedge \|\bar{\mathbf{z}}\|_\infty \leq \beta] \\
\leq& |\bar{C}| \cdot \frac{(2\beta+1)^{(\ell+k)N}}{q^{kN}}
\end{aligned}$$

Putting the two cases together we obtain the result.

Algorithm $\mathsf{DS}_3.\mathsf{SimSign}_n(\mathit{sid}, \mathbf{t}_n, pk, \mu)$

The simulator is parameterized by public parameters described in Table 1 and relies on the random oracles $\mathsf{H}_0 : \{0,1\}^* \to C, \mathsf{H}_3 : \{0,1\}^* \to S_{ck}$ and $\mathsf{H}_4 : \{0,1\}^* \to \{0,1\}^{l_4}$. The simulator assumes that $\mathsf{SimGen}_n(pp, \mathbf{A}, \mathbf{t})$ (Fig. 13) has been previously invoked. If the simulator halts with ABORT at any point, then all $\mathsf{SimSign}_n(\mathit{sid}, \mathbf{t}_n, pk, \mu)$ executions are aborted.

Inputs

1. The simulator receives a unique session ID sid, $\mathbf{t}_n$, $pk = (\bar{\mathbf{A}}, \mathbf{t})$ and message $\mu \in \{0,1\}^*$ as input.
2. The simulator verifies that sid has not been used before (if it has been, the protocol is not executed).
3. The simulator locally computes a per-message commitment key by querying a random oracle $ck \leftarrow \mathsf{H}_3(\mu, \mathbf{t})$.

Signature Generation

1. Compute the first message as follows.
   a. With probability $1 - 1/M$, sample $\mathbf{w}_n \leftarrow\!\!\$\, R_q^k$; otherwise sample $\mathbf{z}_n \leftarrow\!\!\$\, D_\sigma^{\ell+k}$, define $\mathbf{w}_n := \bar{\mathbf{A}}\mathbf{z}_n - c\mathbf{t}_n$ and set the flag *accept*. Then generate a commitment $com_n \leftarrow \mathsf{Commit}_{ck}(\mathbf{w}_n; r_n)$ with $r_n \leftarrow\!\!\$\, D(S_r)$, and its hash $h_n \leftarrow \mathsf{H}_4(com_n)$.
   b. Send out h_n.
2. Upon receiving h_j for all $j \in [n-1]$:
   a. Invoke $\mathsf{SearchHashTable}$ in Fig. 12 on input HT_4 and $(h_1, \ldots, h_{n-1})$ to obtain $(\mathit{alert}, \mathit{bad}_4, com_1, \ldots, com_{n-1})$.
   b. If the flag bad_4 is set then simulation fails.
   c. If the flag *alert* is set then send out com_n.
   d. Otherwise define $com := \sum_{j\in[n]} com_j$ and
      - If $\mathrm{HT}_0[com, \mu, \mathbf{t}]$ has been already set then simulation fails and set the flag bad_5.
      - Otherwise program the random oracle $\mathrm{HT}_0[com, \mu, \mathbf{t}] := c$ and send out com_n.
3. Upon receiving com_j for all $j \in [n-1]$:
   a. If $\mathsf{H}_4(com_j) \neq h_j$ for some j, then send out ABORT.
   b. If *alert* is set and $\mathsf{H}_4(com_j) = h_j$ holds for all j then simulation fails and set the flag bad_6.
   c. Otherwise send out $(\mathbf{z}_n, r_n)$ if the flag *accept* is set; otherwise send out RESTART.
4. Upon receiving RESTART from some party go to 1. Otherwise upon receiving $(\mathbf{z}_j, r_j)$ for all $j \in [n-1]$ compute the combined signature as follows.
   a. For each $j \in [n-1]$ reconstruct $\mathbf{w}_j := \bar{\mathbf{A}}\mathbf{z}_j - c\mathbf{t}_j$ and validate the signature share:
   $$\|\mathbf{z}_j\|_2 \leq B \quad \text{and} \quad \mathsf{Open}_{ck}(com_j, r_j, \mathbf{w}_j) = 1.$$
   If the check fails for some j then send out ABORT.
   b. Compute $\mathbf{z} := \sum_{j\in[n]} \mathbf{z}_j$ and $r := \sum_{j\in[n]} r_j$.

If neither the protocol aborts nor the simulation fails, the simulator obtains a signature $(com, \mathbf{z}, r)$ as local output.

Fig. 17. Simulated signature generation for DS_3