Compressed Linear Aggregate Signatures
Based on Module Lattices

Katharina Boudgoust and Adeline Roux-Langlois
katharina.boudgoust@irisa.fr, adeline.roux-langlois@irisa.fr
Univ Rennes, CNRS, IRISA

Abstract. The Fiat-Shamir with Aborts paradigm of Lyubashevsky (Asiacrypt’09) has given rise to efficient lattice-based signature schemes. One popular implementation is Dilithium which is a finalist in an ongoing standardization process run by the NIST. An interesting research question is whether it is possible to combine several unrelated signatures, issued from different signing parties on different messages, into one single aggregated signature. Of course, its size should be much smaller than the trivial concatenation of all signatures. Ideally, the aggregation can be done offline by a third party, called public aggregation. Doröz et al. (IACR eprint 2020/520) proposed a first lattice-based aggregate signature scheme allowing public aggregation. However, its security is based on the hardness of the Partial Fourier Recovery problem, a structured lattice problem which neither benefits from worst-to-average reductions nor wasn’t studied extensively from a cryptanalytic point of view. In this work we give a first instantiation of an aggregate signature allowing public aggregation whose hardness is proven in the aggregate independent-chosen-key model assuming the intractability of two well-studied problems on module lattices: The Module Learning With Errors problem (M-LWE) and the Module Short Integer Solution problem (M-SIS). Both benefit from worst-case to average-case hardness reductions. The security model we use is a more restricted variant of the original aggregate chosen-key model. Our protocol can be seen as an aggregated variant of Dilithium. Alternatively, it can be seen as a transformation of the protocol from Doröz et al. to the M-LWE/M-SIS framework.

Keywords. Lattice-Based Cryptography, Module Lattices, Signature Aggregation

1 Introduction

For a long time, the main focus of cryptology was on secure encryption. With the invention of public key cryptology in the 1970s and the spread of the internet, the need of secure key exchange and authentication of data became more and more important. This is why nowadays the focus of public key cryptology increasingly shifts towards digital signatures. A digital signature scheme $\Pi_S$ with message space $\mathcal{M}$ is composed of three algorithms $KGen, Sig$ and $Vf$. The algorithm $KGen$
generates a key pair \((sk, vk)\) for a given user, who can then use their secret key \(sk\) to generate a signature \(\sigma\) on a given message \(m \in M\) by running \(\text{Sig}(sk, m)\). Afterwards, this signature can be verified by anyone using the verification key \(vk\), which is publicly available, by running \(\{0, 1\} \leftarrow \text{Vf}(vk, m, \sigma)\). If the verification procedure outputs 1, the signature passes validation.

An interesting research question is whether it is possible to define an additional algorithm \(\sigma_{agg} \leftarrow \text{AggSig}(VK, M, \Sigma)\) which takes as input a vector of \(N \in \mathbb{Z}\) verification keys \(VK = (vk_j)_{j \in [N]}\), a vector of \(N\) messages \(M = (m_j)_{j \in [N]}\) and a vector of \(N\) signatures \(\Sigma = (\sigma_j)_{j \in [N]}\), that were generated by the \(N\) different signing parties with corresponding verification keys \(vk_j\), and outputs a single signature \(\sigma_{agg}\). We further require a way for others to verify that \(\sigma_{agg}\) is indeed an aggregation of valid signatures. Thus, we need to provide a second additional algorithm \(\{0, 1\} \leftarrow \text{AggVf}(VK, M, \sigma_{agg})\), that outputs 1 if \(\sigma_{agg}\) is a valid aggregation of \(N\) valid signatures. All five algorithms define a so-called aggregate signature scheme \(\Pi_{AS} = (\text{KGen}, \text{Sig}, \text{Vf}, \text{AggSig}, \text{AggVf})\), where we require that it must satisfy correctness and unforgeability properties. A trivial solution is to set \(\sigma_{agg}\) as the concatenation of all the \(N\) different signatures and verify one after the other. In the following we are searching for an aggregate scheme that produces a \(\sigma_{agg}\) which is significantly shorter than this trivial solution. Ideally, the aggregation algorithm \(\text{AggSig}\) can be performed by a third, even untrusted party without needing to communicate with the \(N\) signing parties. We call this public aggregation. The concept and a first realization of aggregate signatures with public aggregation were given by Boneh et al. [BGLS03] by using bilinear maps constructed over elliptic curves in the generic group model. Aggregate signatures are a useful tool to save communication costs in settings where different users have to authenticate their communication, for instance in consensus protocols or certificate chains. More recently, they attracted increased interest as they help to reduce the size of blockchains such as the Bitcoin blockchain.

A first attempt to build lattice-based aggregate signatures with public aggregation was recently made by Doröz et al. [DHSS20]. Their construction builds upon the signature scheme \(\text{PASS Sign}\), introduced by Hoffstein et al. [HPS+14]. As a warm-up, they introduce a simple linear aggregate signature, which they call \(\text{MMSA}\) (multi-message, multi-user signature aggregation). However, in this version, the aggregate signature is larger than the trivial concatenation of \(N\) different signatures. In order to improve the efficiency and thus to get something meaningful, they first compress the signature, leading to \(\text{MMSAT}\) (the \(T\) stands for a linear compression function \(T\)), and then compress the verification keys, leading to \(\text{MMSATK}\). Unfortunately, their construction has two disadvantages: First, the size of an aggregated signature is still linear in the number \(N\) of involved signatures, even though it is much smaller than simply concatenating them all thanks to the compression. Second, they only provide a security proof.

---

1 Throughout the paper, the neutral singular pronouns they/their are used in order to keep the language as inclusive as possible. See also https://www.acm.org/diversity-inclusion/words-matter
for the first variant MMSA\(^2\) by showing that it inherits the security of the underlying PASS Sign, and subsequently its security can be based on the hardness of the Partial Fourier Recovery problem (PFR). The PFR asks to recover a polynomial in the ring \(\mathbb{Z}[x]/(x^n - 1)\) of small norm having access only to a partial list of its Fourier transform. It can be formulated as a shortest vector problem over some structured lattices. However, up to date there are no connections to worst-case lattice problems, which may be seen as a security concern.

In a parallel line of research, aggregate signature schemes that only allow for private aggregation have been proposed. In this setting, the different signing parties interact with each other to generate an aggregated signature on one message, which can be the concatenation of different messages. Those are also known as multi-signature schemes and there have been several recent protocols following the Fiat-Shamir with Aborts (FSwA) paradigm \([\text{Lyu09}]\) providing lattice-based interactive aggregate signatures, see for instance \([\text{BS16}], [\text{DOTT20}]\) and \([\text{BK20}]\).

**Contributions.** In this paper, we propose an aggregate signature allowing public aggregation, whose security is proven assuming simultaneously the hardness of Module Learning With Errors (M-LWE) and Module Short Integer Solution (M-SIS) and therefore on worst-case module lattice problems \([\text{LS15}]\). Earlier proposals either only offered security based on (non-standard) average-case lattice problems, or didn’t allow for public aggregation. From a high level perspective, we take the practical signature from Güneysu et al. \([\text{GLP12}]\) as a starting point. It follows the FSwA paradigm for lattice-based schemes \([\text{Lyu09}, \text{Lyu12}]\), which is also used in the signature Dilithium \([\text{DKL}^+18]\), a finalist in the ongoing NIST standardization process\(^3\). The emphasis of our work lies on a proper security proof, in a security model we explain below, also for the compressed version of the aggregate signature. We think that it is important to adapt the interesting ideas of the MMSA(TK) aggregate signature to the setting of lattice-based signatures within the M-LWE/M-SIS framework to stimulate further research in this direction.

**Technical Details.** Let us quickly recall the FSwA paradigm for lattice-based signatures. In the following, all computations are done over the ring \(R_q = \mathbb{Z}_q[x]/(x^n + 1)\), where \(n\) is a power of two and \(q\) is some prime modulus. A verification key is given as \(t = [A|I] \cdot s \in R_q^k\), where \(s \in R_q^{k+\ell}\) is a vector of small norm (defining \(sk\)), \(A \in R_q^{k \times \ell}\) is a public uniform matrix and \(I\) the identity matrix of order \(k\). A signature is provided by \(\sigma = (u, z) \in R_q^k \times R_q^{k+\ell}\), where \(u\) is a commitment that via some hash function \(H_c\) defines a challenge \(c\), and \(z\) is the answer to this challenge. For verification, one checks that \(z\) is small and that \([A|I] \cdot z = t \cdot c + u\), where \(c = H_c(u, t, m)\) for the verification key \(t\) and a message \(m\). Adding \(t\) to the input of \(H_c\) is a simple countermeasure to prevent so-called rogue key attacks \([\text{BGLS03}, \text{Sec. 3.2}]\). The size of a single signature can be reduced by replacing \(u \in R_q^k\) by \(c \in R\). As we detail out later in Section 3 this

\(^2\)The authors announced that a security proof for MMSA(TK) will appear in a full version which was not available at the time of writing.

\(^3\)[https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/](https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/)
can’t be done in the aggregate setting. A naive way to aggregate $N$ different signatures $(\sigma_j)_{j \in [N]}$ with $\sigma_j = (u_j, z_j)$ into one signature $\sigma_{agg}$ would be to compute the sum of all components $(\sum_j u_j, \sum_j z_j)$. However, we wouldn’t be able to verify this aggregated signature as we can’t re-compute the different challenges $c_j$ as we don’t know the inputs $u_j$ to $H_c$, originally used by the signing parties. Thus, we can only sum up the $z_j$ parts and still have to transmit all the $u_j$, which produces an aggregate signature of the form $\sigma_{agg} = ((u_j)_j, \sum_j z_j)$. This is essentially how the aggregate signature in MMSA looks like [DHSS20] Sec. 4.

In order to gain efficiency, we use the following observation made in [DHSS20] Sec. 5. For the security of the signature scheme, we need the collision property of the random oracle $H_c$ to be negligible in our security parameter $\lambda$, say $2^{-\lambda}$. However, a commitment $u$ is an element of $R_q^n$ and thus the commitment space is much bigger than we need for combinatorial security. By taking a linear map $T: R_q^n \to \mathbb{Z}_q^{n_0}$ with $n_0 \log_2(q) \approx 2\lambda$, we can compress the input to $H_c$ to $T(u_j)$. For the verification, we only need the sum of the $u_j$, i.e., $\hat{u} = \sum_j u_j$.

Now, the aggregate signature is set to $\sigma_{agg} = (\hat{u}, T(u_j)_j, z)$, where $z = \sum_j z_j$. In order to pass verification, the norm of $z$ has to be small enough and the equation $\langle A[I] \cdot z = \sum_j t_j \cdot c_j + \hat{u} \rangle$ has to be true, where $c_j = H_c(T(u_j), t_j, m_j)$. Further, one checks that $T(\hat{u}) = \sum_j T(u_j)$, to make sure that the $T(u_j)$ and $\hat{u}$ are well defined. We formally present our aggregate signature scheme in Section 3.

An aggregated signature is composed of $N + 2$ elements, which is linear in the number of involved signatures. Still, it is much smaller than the trivial concatenation, as the size of the vector $T(u_j)_j$ is only linear in the security parameter $\lambda$ and not linear in the lattice dimension $kn$, where $\lambda \ll kn$.

To illustrate the gain in efficiency, we consider the concrete parameters of Dilithium for the security level III with security parameter $\lambda = 128$. In this setting, one signature $\sigma = (z, c) \in R_q^n \times R_q$ with $\|z\|_{\infty} \leq B$ and $\|c\|_{\infty} = 1$ is of size 2 701 Bytes. Taking the concatenation of $N = 1000$ signatures results in an aggregate signature of size 2 701 000 Bytes. In our construction, an aggregate signature $\sigma_{agg} = (\hat{u}, (T(u_j))_{j \in [N]}, z) \in R_q^n \times (\mathbb{Z}_q^{n_0})^N \times R_q^{k+t}$ with $\|z\|_{\infty} \leq \sqrt{N} \cdot B$ and $n_0 \cdot \log_2(q) \approx 2\lambda$. In order to keep the same ratio between the bound and the modulus, in our scheme $\log_2(q)$ has to be larger than in Dilithium by an additive factor $\log_2(\sqrt{N})$. The resulting $\sigma_{agg}$ is of size $\approx 43 700$ Bytes, which is more than 60 times smaller than the trivial solution.

In Section 4 we provide a rigorous security proof (Theorem 1), where the proof idea follows the one of Damgård et al. [DOTT20] for their inter-active multi-signature. It is composed of a sequence of indistinguishable games (assuming the hardness of M-LWE), where the starting one is the security game of our aggregate signature. The game is specified by the aggregate independent-chosen-key model, a restricted variant of the more general aggregate chosen-key model originally introduced by Boneh et al. [BGLS03]. The main difference between both notions is that in our security game the adversary has to generate a set of signatures before receiving the challenge in order to make sure that they are independent of it. In Section 4.3 we show a subtle attack against our aggregate signature in the more general chosen-key model, motivating the intro-
duction of our new model. It was observed by Zhao [Zha19] in the context of Schnorr-based aggregate signatures on elliptic curves. Note that Zhao proposed a modified protocol to prevent this type of attacks at the expense of introducing a new non-standard problem. This modified protocol was later attacked with a sub-exponential algorithm by Hanaoka et al. [HOS +20].

In the last game, the signing procedure is simulated by some algorithm that doesn’t depend on the secret key and the challenge verification key is sampled uniformly at random. By applying the General Forking Lemma from Bellare and Neven [BN06] we can use two different responses of a successful adversary against the scheme in the last game to solve an instance of M-SIS. As we don’t need trapdoor commitment schemes, the proof is less technical than the one in [DOTT20]. We use a Gaussian distribution for the masking vectors and the rejection sampling, as done in [DOTT20]. This leads to tighter norm bounds of an aggregate signature, see Remark 2.

Open Problems. The General Forking Lemma induces two problems: First, we currently don’t know how to extend it to the quantum setting. And second, it leads to non-tight security proofs. To circumvent both issues, one could try to adapt the techniques of Abdalla et al. [AFLT16] and of Kiltz et al. [KLS18]. As we point out in Remark 2 the signature compression impedes their straight-forward adaption. Furthermore, we leave as an open problem to provide an aggregate signature scheme based on standard lattice-problems which allows public aggregation and at the same time is provably secure in the aggregate chosen-key security model. More generally, the design of such a scheme which additionally produces aggregated signatures of length independent of the number N of aggregated signatures, is an important open problem.

2 Preliminaries

For $k \in \mathbb{N}$, we represent the set $\{1, \ldots, k\}$ by $[k]$. Vectors are denoted in bold lowercase and matrices in bold capital letters and the identity matrix of order $k$ is denoted by $I_k$. The concatenation of two matrices $A$ and $B$ with the same number of rows is denoted by $[A|B]$. For any set $S$, we denote by $U(S)$ the uniform distribution over $S$. We write $x \leftarrow D$ to denote the process of sampling an element $x$ following the distribution $D$. Throughout the paper $R = \mathbb{Z}[x]/(x^n+1)$ denotes the ring of integers of the $2n$-th cyclotomic field, where $n$ is a power of two. Further, $q$ is a prime such that $q = 1 \mod 2n$ defining the quotient ring $R_q = \mathbb{Z}_q[x]/(x^n+1)$. An element $a = \sum_{j=1}^n a_j x^{j-1}$ of $R$ is identified with its coefficient vector $a = (a_j)_{j \in [n]} \in \mathbb{Z}^n$. For any ring element $a \in R$, we set $\|a\|_\infty$, $\|a\|_2$ and $\|a\|_1$ as the infinity, the Euclidean and the $\ell_1$-norm of its coefficient vector, respectively. All norms can be generalized to vectors $a \in R^k$ for $k \in \mathbb{N}$, by considering the coefficient vector of dimension $kn$ defined by $a$. We rely on the key set $S_\beta = \{a \in R: \|a\|_\infty \leq \beta\}$ with $\beta \in \mathbb{N}$. We define continuous and discrete Gaussian distributions over $R^m$, for $m \in \mathbb{Z}$.

4 In an earlier version of this paper, we weren’t aware of this attack.
Definition 1. For \( z \in \mathbb{R}^m \) let \( \rho_{v,s}(z) = (1/\sqrt{2\pi s})^m \exp(-\|z-v\|^2/2s^2) \) be the continuous Gaussian distribution centered at \( v \in \mathbb{R}^m \) with standard deviation \( s > 0 \). Its discrete analogue is defined as \( D_{v,s}^m(z) = \sum_{x \in \mathbb{R}^m} \rho_{v,s}(x) \). If \( v = 0 \), we simply write \( D_s^m(z) \).

We restate a result on the distribution of the sum of discrete Gaussians over \( \mathbb{R}^m \). It uses the so-called smoothing parameter \( \eta(\mathbb{R}^m) \) of \( \mathbb{R}^m \), which was introduced in [MR07] and can be bounded by \( \eta(\mathbb{R}^m) \leq \sqrt{\omega(\log(nm))} \) [MR07, Lem. 3.3]. Informally, it describes a threshold above which many properties of the continuous Gaussian distribution also hold for its discrete analogue.

Lemma 1 (Thm. 3.3 [MP13]). Suppose that \( s \geq \eta(\mathbb{R}^m)/\sqrt{\pi} \). Let \( m,N \in \mathbb{Z} \) and for \( j \in [N] \) let \( z_j \leftarrow D_s^m \) be independent samples. Then the distribution of \( z = \sum_{j \in [N]} z_j \) is statistically close to \( D_s^{\sqrt{N}} \).

Finally, we use the following tail bound on discrete Gaussians.

Lemma 2 (Lem. 4.4 [Lyu12]). For any parameter \( \gamma > 1 \) it yields \( \Pr[\|z\|_2 > \gamma s\sqrt{mn}; z \leftarrow D_s^m] < \gamma^m \exp(mn(1-\gamma^2)/2) \).

Throughout the paper we choose \( \gamma \) such that this probability is negligibly small for a fixed \( m \).

2.1 Module Lattice Problems

We also recall two lattice problems and refer to [LS15] for more details. We state them in their discrete, primal and HNF form.

Definition 2 (M-LWE). Let \( k,\ell,\beta \in \mathbb{N} \). The Module Learning With Errors problem M-LWE\(_{k,\ell,\beta} \) is defined as follows. Given \( A \leftarrow U(\mathbb{R}^{k \times \ell}) \) and \( t \in \mathbb{R}^k \), Decide whether \( t \leftarrow U(\mathbb{R}^{k}) \) or if \( t = [A|I_k] \cdot s \), where \( s \leftarrow U(S_{\beta}^{\ell+k}) \).

The M-LWE assumption states that no PPT distinguisher can distinguish the two distributions with non-negligible advantage. Worst-case to average-case reductions guarantee that M-LWE is quantumly [LS15] and classically [BJRW20] at least as hard as the approximate shortest vector problem over module lattices.

Definition 3 (M-SIS). Let \( k,\ell,b \in \mathbb{N} \). The Module Short Integer Solution problem M-SIS\(_{k,\ell,b} \) is as follows. Given a uniformly random matrix \( A \leftarrow U(\mathbb{R}^{k \times \ell}) \). Find a non-zero vector \( s \in \mathbb{R}^{k+\ell} \) such that \( \|s\|_2^2 \leq b \) and \( [A|I_k] \cdot s = 0 \in \mathbb{R}^k \).

The M-SIS assumption states that no PPT adversary can solve this problem with non-negligible probability. Worst-case to average-case reductions guarantee that M-SIS is classically [LS15] at least as hard as the approximate shortest independent vector problem over module lattices.
2.2 Aggregate Signature Schemes

We present the formal definition of aggregate signature schemes and their property of correctness.

**Definition 4.** An aggregate signature scheme $\Pi_{AS}$ for a message space $M$ consists of a tuple of PPT algorithms $\Pi_{AS} = (KGen, \text{Sig}, Vf, \text{AggSig}, \text{AggVf})$, proceeding as specified in the following protocol:

- $KGen(1^\lambda) \rightarrow (sk, vk)$: On input a security parameter $\lambda$, the key generation algorithm returns a secret signing key $sk$ and a public verification key $vk$.
- $\text{Sig}(sk, m) \rightarrow \sigma$: On input a signing key $sk$ and a message $m \in M$, the signing algorithm returns a signature $\sigma$.
- $Vf(vk, m, \sigma) \rightarrow \{0, 1\}$: On input a verification key $vk$, a message $m \in M$ and a signature $\sigma$, the verification algorithm either accepts (i.e. outputs 1) or rejects (i.e. outputs 0).
- $\text{AggSig}(VK, M, \Sigma) \rightarrow \sigma_{agg}$: On input a vector of verification keys $VK = (vk_j)_{j \in [N]}$, a vector of messages $M = (m_j)_{j \in [N]}$ and a vector of signatures $\Sigma = (\sigma_j)_{j \in [N]}$, the signature aggregation algorithm returns an aggregated signature $\sigma_{agg}$.
- $\text{AggVf}(VK, M, \sigma_{agg}) \rightarrow \{0, 1\}$: On input a vector of verification keys $VK = (vk_j)_{j \in [N]}$, a vector of messages $M = (m_j)_{j \in [N]}$ and an aggregated signature $\sigma_{agg}$, the aggregated verification algorithm either accepts (i.e. outputs 1) or rejects (i.e. outputs 0).

**Definition 5.** Let $\Pi_{AS} = (KGen, \text{Sig}, Vf, \text{AggSig}, \text{AggVf})$ be an aggregate signature scheme for a message space $M$. It is called correct if for all security parameters $\lambda \in \mathbb{N}$ and number of signers $N \in \mathbb{N}$ it yields

$$\Pr[\text{AggVf}(VK, M, \text{AggSig}(VK, M, \Sigma)) = 1] = 1,$$

where $m_j \in M$, $(sk_j, vk_j) \leftarrow KGen(1^\lambda)$ and $\sigma_j \leftarrow \text{Sig}(sk_j, m_j)$ for $j \in [N]$ and $VK = (vk_j)_{j \in [N]}$, $M = (m_j)_{j \in [N]}$ and $\Sigma = (\sigma_j)_{j \in [N]}$.

2.3 General Forking Lemma

For the sake of completeness and to fix notations, we restate the General Forking Lemma from Bellare and Neven [BN06].

**Lemma 3 (General Forking Lemma).** Let $N_q \geq 1$ be an integer and let $C$ be a set of size $|C| \geq 2$. Let $B$ be a randomized algorithm that on input $x, h_1, \ldots, h_{N_q}$ returns a pair $(j, out)$, where $j \in \{0, \ldots, N_q\}$ and a side output $out$. Let $lGen$ be a randomized algorithm called the input generator, parameterized by some security parameter $\lambda$. We define the accepting probability of $B$ as

$$\text{acc} = \Pr[j \neq 0: x \leftarrow lGen(1^\lambda); h_1, \ldots, h_{N_q} \leftarrow U(\mathcal{H}); (j, out) \leftarrow B(x, h_1, \ldots, h_{N_q})].$$

Let $F_B$ be a forking algorithm that works as in Figure 7 given $x$ as input and black-box access to $B$. We define the forking probability of $F_B$ as

$$\text{frk} = \Pr[(\text{out}, \text{out}) \neq (\perp, \perp): x \leftarrow lGen(1^\lambda); (\text{out}, \text{out}) \leftarrow F_B(x)].$$

Then it yields $\text{acc} \leq N_q/|C| + \sqrt{N_q \cdot \text{frk}}$. 

Compressed Linear Aggregate Signatures Based on Module Lattices 7
Upon input $x$, the algorithm $F_B$ does the following:

1. Pick a random coin $\rho$ for $B$
2. Generate $h_1, \ldots, h_{N_q} \leftarrow U(C)$
3. $(j, \text{out}) \leftarrow B(x, h_1, \ldots, h_{N_q}, \rho)$
4. If $j = 0$, then return $(\bot, \bot)$
5. Regenerate $\tilde{h}_j, \ldots, \tilde{h}_{N_q} \leftarrow U(C)$
6. $(\tilde{j}, \tilde{\text{out}}) \leftarrow B(x, h_1, \ldots, h_{j-1}, \tilde{h}_j, \ldots, \tilde{h}_{N_q}, \rho)$
7. If $j = \tilde{j}$ and $h_j \neq \tilde{h}_j$, then return $(\text{out}, \tilde{\text{out}})$
8. Else return $(\bot, \bot)$

Fig. 1: The forking algorithm $F_B$.

3 Our Lattice-Based Aggregate Signature Scheme

In this section we first present the underlying single signature scheme (Section 3.1) before introducing our aggregate signature scheme in Section 3.2. From a high level perspective, we take the practical signature from Güneysu et al. [GLP12] as a starting point. The linear aggregation follows the idea of Doröz et al. [DHSS20], where the main difference is that we moved to the M-LWE/M-SIS framework instead of the Partial Fourier Recovery framework of the original scheme. We think that M-LWE and M-SIS are more standard lattice problems and thus they increase our confidence in the security of the proposed scheme.

3.1 The Single Signature Scheme

In the following we describe the underlying single signature scheme, which is essentially the signature scheme from Güneysu et al. [GLP12] with minor modifications. Let $R_q = \mathbb{Z}_q[x]/(x^n + 1)$, with $n$ a power of two and $q$ a prime such that $q = 1 \mod 2n$. For $k, \ell \in \mathbb{N}$, let $A \in R_{k \times \ell}^q$ follow the uniform distribution and be a public shared parameter of the system. The number of columns $\ell$ and the number of rows $k$ should be adapted to the required security level, but usually they are small constants. Let $H_c : \{0, 1\}^* \rightarrow C = \{c \in R : \|c\|_1 = d, \|c\|_\infty = 1\}$ be a random oracle with $d$ such that $|C| > 2^{2\lambda}$, where $\lambda$ denotes the required security level. Let $s, \beta, M \in \mathbb{Z}$ and the message space $\mathcal{M} = \{0, 1\}^*$. Further, let $T$ denote a linear map $T : R_{k}^q \rightarrow \mathbb{Z}_{n_0}^q$, such that $n_0 \cdot \log_2 q \approx 2\lambda$.

The signature scheme $\Pi_S = (\text{KGen}, \text{Sig}, \text{Vf})$ from [GLP12] with minor modifications and allowing signature compression is illustrated in Figure 2.

**Description.** The algorithm KGen samples a secret key vector $s$, composed of elements of $R$ with coefficients of size at most $\beta$, and sets the verification key to $t = [A][I_k] \cdot s \in R_k^k$. At the beginning of the signing procedure, a masking vector $y$ following the Gaussian distribution $D_{n+k}^q$ is sampled. The signing party then computes $u = [A][I_k] \cdot y \in R_{n_0}^q$, which serves together with the verification key $t$ and the message $m$ as input to the random oracle $H_c$. The output $c$ of $H_c$ is a polynomial in $R$ with exactly $d$ coefficients that are $\pm 1$ and the remaining coefficients are $0$. The second part of a potential signature is defined...
as \( z = s \cdot c + y \). In order to make the distribution of the signature independent of the secret key, the algorithm only outputs the potential signature with probability \( \min(1, D_s^{\ell+k}(z)/M \cdot D_{t,c}^{\ell+k}(z)) \), where \( M \) is a constant that depends on \( \beta \) (the Euclidean norm of the secret \( s \)) and \( d \) (the \( \ell_1 \)-norm of the challenge \( c \)). This step is called rejection sampling. In order to verify \( \sigma \), the verifier first re-constructs the hash value \( c = H_e(T(u), t, m) \) and then checks if the norm of \( z \) is smaller than \( B \), where \( B = \gamma s \sqrt{n(\ell+k)} \) is as in Lemma 2 with \( m = \ell + k \), and that \([A]c \cdot z = t \cdot c + u\). The parameters \( \beta, d, M \) and \( \gamma \) have to be set strategically such that the scheme is correct, efficient and secure, see \[\text{Lyu12}\text{DKL}^+,18\].

**Modifications.** A first difference to the signature scheme in \[\text{GLP12}\] is that we use a Gaussian distribution for \( y \) (and thus for \( z \) and the rejection sampling) as originally done in \[\text{Lyu12}\], instead of a bounded uniform distribution. This change is motivated by the fact that the sum of independent Gaussians provides better bounds on its norm than the sum of uniformly distributed elements. This becomes crucial when aggregating the signatures in Section 3.2.

Another minor difference is that instead of transmitting \( c \) in the signature, we send \( u \). For a single signature, both cases are equivalent, as \( u \) defines \( c \) via the hash function \( H_e \) (and the verification key \( t \) and the message \( m \)) and \( c \) defines \( u \) via the equation \( u = [A]c \cdot z - t \cdot c \) over \( R_q \). In Section 3.2 we see that this is not the case for an aggregate signature scheme and we thus need to transmit the information \( u \) (and some additional information on the compressed values, see Sec. 3.2). Note that the size of \( u \) is much larger than the one of \( c \), but for a large number of aggregated signatures, this additional cost will be amortized.

A third modification is that we add the verification key \( t \) to the input of the hash function \( H_e \) to compute the challenge \( c \). As proposed by Boneh et
scheme is given by aggregation with compression is illustrated in Figure 3. The aggregate signature $T$ values produce the two sums $\sum j \in [N] \sum z_j \in R_q^{n_0 \times k}$ and $\hat{u} = \sum j \in [N] u_j \in R_q^k$. If $\|z\|_2 < \sqrt{N \cdot B}$, then return $\sigma_{agg} = (\hat{u}, (T(u_j))_j, z)$; else return $\perp$.

$\text{AggVf(VK, M, } \sigma_{agg} :$ Re-construct $c_j = H_c(T(u_j), t_j, m_j)$ for all $j \in [N]$. If $\|z\|_2 < \sqrt{N \cdot B}$ and if $[A I_k] \cdot z = \sum j \in [N] (t_j \cdot c_j) + \hat{u}$ and if $T(\hat{u}) = \sum j \in [N] T(u_j)$ return 1; else return 0;

Fig. 3: Linear Signature Aggregation with Compression.

In order to aggregate $N$ signatures $(\sigma_j)_{j \in [N]}$ the algorithm $\text{AggSig}$ simply computes the two sums $z = \sum j \in [N] z_j$ and $\hat{u} = \sum j \in [N] u_j$, together with the compressed values $T(u_j)_{j \in [N]}$ and outputs the aggregated signature $\sigma_{agg} = (\hat{u}, (T(u_j))_j, z)$.
if the Euclidean norm of $z$ is bounded above by $\sqrt{N} \cdot B$. Else the algorithm outputs $\bot$. The probability that $\text{AggSig}$ outputs $\bot$ is negligible by Lemma 1. The aggregation can be done by anyone, even by untrusted parties, as long as they have access to the random oracle $H_c$. Thus, public aggregation is enabled.

To verify an aggregated signature $\sigma_{agg}$, $\text{AggVf}$ first re-constructs the challenges $c_j$ for $j \in [N]$ by using the compressed commitment $T(u_j)$ provided in $\sigma_{agg}$, the verification key $t_j$ and the message $m_j$. The algorithm then checks if the norm of $z$ lies within the correct bound. Finally it verifies the equations $[A, I_k] \cdot z = \sum_j (t_j \cdot c_j) + \hat{u}$ and $T(\hat{u}) = \sum_j T(u_j)$. If all checks go through, it outputs 1, else 0. It now becomes clear that transmitting $c_j$ wouldn’t be sufficient to verify the aggregated signature: as the verifier only knows the term $z$, but not all the $z_j$, they cannot reconstruct $u_j$ from $c_j$. Without knowing $u_j$, however, the verifier would not be able to verify the aggregated signature. Inter-active multi-signatures circumvent this issue by generating inter-active one common input $u$ to the random oracle that depends on all the $u_j$, before sending the multi-signature, see for instance [BS16] or [DOTT20]. As a public aggregation is our main objective, we accept a larger aggregated signature size.

The correctness of our protocol $\Pi_{AS}$ simply follows from the linearity of matrix-vector multiplication over $R_q$.

**Lemma 4 (Correctness).** Let $\Pi_{AS} = (KGen, \text{Sig}, Vf, \text{AggSig}, \text{AggVf})$ be the aggregate signature scheme for a message space $M$ with the algorithms as in Figures 3 and 4. Assuming the correctness of the corresponding single signature scheme $\Pi_S = (KGen, \text{Sig}, Vf)$, the aggregate signature is correct, i.e.,

$$\Pr[\text{AggVf}(VK, M, \text{AggSig}(VK, M, \Sigma)) = 1 | \text{AggSig}(VK, M, \Sigma) \neq \bot] = 1,$$

where $m_j \in M$, $(sk_j, vk_j) \leftarrow KGen(1^\lambda)$ and $\sigma_j \leftarrow \text{Sig}(sk_j, m_j)$ for $j \in [N]$ and $VK = (vk_j)_{j \in [N]}$, $M = (m_j)_{j \in [N]}$ and $\Sigma = (\sigma_j)_{j \in [N]}$.

**Proof.** Let $\sigma_{agg} \neq \bot$ be the aggregate signature produced by $\text{AggSig}(VK, M, \Sigma)$. The first two checks if $\|z\|_2 \leq \sqrt{N} \cdot B$ and if $T(\hat{u}) = \sum_{j \in [N]} T(u_j)$ succeed by the construction of $\text{AggSig}$. For the last check, we compute

$$[A, I_k] \cdot z = [A, I_k] \cdot \left( \sum_{j=1}^{N} z_j \right) = \sum_{j=1}^{N} [A, I_k] \cdot z_j$$

$$= \sum_{j=1}^{N} t_j \cdot c_j + u_j = \sum_{j=1}^{N} t_j \cdot c_j + \hat{u},$$

where we used the linearity over $R_q$ and the correctness of $\Pi_S$. \hfill $\Box$

**Remark 1.** In order to guarantee the correctness of the scheme, it is important that all key pairs $(sk, vk)$ share the same public matrix $A$. It can be computed inter-active by all, or by a reasonable large subset of all parties together during a setup-phase as in [DOTT20]. In order to gain in efficiency, it can instead also be computed by some compact random seed.
The key idea behind the compression is that \((\hat{u}, T(u_j))_{j \in [N]}\) is much smaller than transmitting the complete \((u_j)_{j \in [N]}\). At the same time, \(n_0\) is large enough to guarantee combinatorial security as we set \(n_0\) such that it is hard to find a collision of \(H_c\) for fixed verification key \(t\) and message \(m\). By verifying that \(T(\hat{u}) = \sum_j T(u_j)\), we know that \(\hat{u}\) was correctly generated. If we set \(T\) as the identity map (and \(n_0 = kn\)), then \(T(u_j) = u_j\) defines the full vector and we can delete the sum \(\hat{u}\) from the aggregate signature.

**Remark 2.** Throughout the paper we use the Gaussian distribution for the masking vectors \(y_j\) and thus the resulting signature components \(z_j\). Alternatively, one can use a bounded uniform distribution, as originally done in [GLP12], as well as in Dilithium. In this case, the bound on the norm of the aggregate signature component \(z\) increases by a factor of \(N\) (instead of a factor of \(\sqrt{N}\) in the Gaussian setting). To mitigate this loss, one can use a randomized weighted sum and the Central Limit Theorem as done for MMSA in [DHSS20, Sec. 4], which requires the introduction of a second hash function, increasing the aggregation complexity.

## 4 Security of Our Aggregate Signature Scheme

We present in Section 4.1 our security model for aggregate signatures before proving in Section 4.2 the security of our scheme from Section 3.

### 4.1 The Aggregate Independent-Chosen-Key Security Model

Informally speaking, the security notion we use within this paper of an aggregate signature scheme captures that there exists no efficient adversary who is able to existentially forge an aggregate signature, within a specified game. We call it the *aggregate independent-chosen-key* security model, which can be seen as a weaker variant of the more general *aggregate chosen-key* security model as originally introduced by Boneh et al. [BCLS03]. In Section 4.3 we show that this model is too strong for our aggregate signature scheme by presenting a subtle attack against it, first observed by Zhao [Zha19]. More generally, this attack applies to all Schnorr-type signatures which use a linear aggregation as we do. Let \(\Pi_{AS} = (\text{KGen, Sig, Vf, AggSig, AggVf})\) be an aggregate signature scheme with message space \(\mathcal{M}\) as in Definition 4 and let \(N\) be the number of signatures to aggregate. At the beginning of the game, an adversary \(\mathcal{A}\) attacking \(\Pi_{AS}\) is asked to output a vector of messages \(M' = (m_j)_{j \in [N-1]}\), of verification keys \(\mathcal{VK}' = (vk_j)_{j \in [N-1]}\) and of signatures \(\Sigma' = (\sigma_j)_{j \in [N-1]}\). Note that the signatures don’t necessarily have to be valid signatures on the messages in \(M'\) under the keys \(\mathcal{VK}'\). Only then \(\mathcal{A}\) is given a single verification key \(vk_N\), the *challenge key*. Their goal is the existential forgery of an aggregate signature \(\sigma_{agg}\) involving \(N\) signatures, where we oblige \(\mathcal{A}\) to include a signature that can be verified using the challenge key and to include the verification keys of \(\mathcal{VK}'\), the messages of \(M'\) and the signatures of \(\Sigma'\). The adversary is also given access to a signing oracle on the challenge key \(vk_N\).
The reason why this is called the independent-chosen-key model, is that we force the adversary to forge an aggregate signature, where \( N - 1 \) of the \( N \) signatures on verification keys and messages chosen by the adversary have to be fixed before receiving the challenge. With this restriction, we guarantee that the signatures are independent of the challenge. Their advantage, denoted by \( \text{Adv} \text{ AggSig}_A \), is defined to be their probability of success in the following game.

**Commit.** For \( j \in [N-1] \), the aggregate forger \( A \) outputs messages \( m_j \), verification keys \( vk_j \) and signatures \( \sigma_j \), defining \( M' = (m_j)_j \), \( V_K' = (vk_j)_j \) and \( \Sigma' = (\sigma_j)_j \).

**Setup.** Then, \( A \) is provided with a challenge verification key \( vk_N \).

**Queries.** Proceeding adaptively, \( A \) queries signatures on messages of their choice that can be verified using the challenge key \( vk_N \).

**Response.** Finally, \( A \) outputs an aggregate signature \( \sigma_{agg} \), together with an \( N \)-th signature \( \sigma_N = (z_N, u_N) \) and message \( m_N \).

**Result.** The forger \( A \) wins the game if \( \sigma_{agg} = \text{AggSig}(V_K, M, \Sigma) \), where \( V_K = [V_K'|t_N] \), \( M = [M'|m_N] \) and \( \Sigma = [\Sigma'|\sigma_N] \), and if \( 1 \leftarrow \text{AggVf}(V_K, M, \sigma_{agg}) \).

In order to avoid trivial solutions, \( A \) is not allowed to hand in a pair \( (vk_N, m_N) \), which was queried on the signing oracle before.

If we show the security in the Random Oracle Model (ROM), we also have to give \( A \) the possibility to query the used random oracles.

**Definition 6.** Let \( H \) denote a hash function that is modeled as a random oracle. An aggregate signature scheme \( \Pi_{\text{AS}} \) is called \((N_H, N_{\text{Sig}}, N)\)-secure against existential forgery in the aggregate independent-chosen-key model in the ROM, if there exists no PPT algorithm \( A \) that existentially forges an aggregate signature on \( N \) verification keys in the aggregate independent-chosen-key model, where \( A \) has non-negligible advantage, makes at most \( N_H \) queries to the random oracle \( H \) and at most \( N_{\text{Sig}} \) queries to the signing oracle on the challenge key.

### 4.2 Proof of Security

We now prove that our scheme is secure against existential forgery in the independent-chosen-key model. The proof follows the one of Damgård et al. \cite{DOTT20} for their interactive multi-signature. As we don’t need trapdoor commitment schemes, our proof is less technical. Let \( \Pi_{\text{AS}} = (KGen, \text{Sig}, Vf, \text{AggSig}, \text{AggVf}) \) be the aggregate signature from Section 3.

**Theorem 1.** Let \( H_c \) be the hash function from our aggregate signature, modeled as a random oracle, with image space \( C \). Assume the hardness of \( \text{M-LWE}_{k, l, \beta} \) and \( \text{M-SIS}_{k, l+1, b} \), where \( b = 2(\sqrt{N} \cdot B + \sqrt{d}) \) with \( B = \gamma s \sqrt{n(k + l)} \). Then, the aggregate signature \( \Pi_{\text{AS}} \) with parameters \((l, k, \beta, s, \gamma, d, N)\) is secure against existential forgery in the aggregate independent-chosen-key model in the ROM. The advantage of some PPT adversary \( A \) against \( \Pi_{\text{AS}} \) is bounded above by

\[
\text{Adv}_{\text{AggSig}_A} \leq \text{Adv}_{\text{M-LWE}_{k, l, \beta}} + N_q/|C| + \sqrt{N_q \cdot \text{Adv}_{\text{M-SIS}_{k, l+1, b}}} + \text{negl}(\lambda),
\]
where \( \lambda \) denotes the security parameter and \( A \) makes at most \( N_{H_c} \) queries to \( H_c \) and \( N_{\text{Sig}} \) queries to the signing oracle and we set \( N_q = N_{H_c} + N_{\text{Sig}} \).

**Proof.** Let \( A \) be an adversary against \( \Pi_{\text{AS}} \) with advantage \( \text{Adv}_{\text{AggSig}}A \). Our high level goal is to show that their advantage is negligible in the security parameter \( \lambda \) by providing a sequence of games \( G_0, G_1 \) and \( G_2 \), where \( G_0 \) is the original independent-chosen-key security game as in Section 4.1. Assuming the hardness of M-LWE, the adversary \( A \) can distinguish between those games only with negligible advantage. In the last game \( G_2 \) we apply the General Forking Lemma in order to obtain two forgeries \( \sigma_{agg}, \sigma_{agg} \), with distinct challenges for the challenge key \( \text{vk}_N \). The two forgeries then allow to construct a solution to M-SIS.

\( G_0 \): Set \( N_q = N_{H_c} + N_{\text{Sig}} \), where \( A \) makes at most \( N_{H_c} \) queries to \( H_c \) and \( N_{\text{Sig}} \) queries to the signing oracle on \( \text{vk}_N \). Recall that \( C \) denotes the challenge space from \( \text{Sig} \). Let \( B \) be a second algorithm that is provided with some randomly chosen \( h_j \leftarrow U(C) \) for \( j \in [N] \). For the random oracle \( H_c \), the algorithm \( B \) maintains a table \( HT_c \) which is empty at the beginning. Further \( B \) stores a counter \( ctr \), initially set to 0.

**Commit.** For \( j \in [N-1] \), the aggregate forger \( A \) outputs messages \( m_j \), verification keys \( t_j \) and signatures \( \sigma_j = (u_j, z_j) \), defining \( VK' = (t_j), M' = (m_j) \) and \( \Sigma' = (\sigma_j) \). After receiving this information, \( B \) checks if \( HT_c[x_j] \) was already set for \( x_j = (T(u_j), t_j, m_j) \) for \( j \in [N-1] \). If no, they sample \( HT_c[x_j] \leftarrow U(C) \). With this, \( B \) sets the random oracle’s output for the committed signatures (even if \( A \) queries them later).

**Setup.** \( B \) generates \((sk_N, \text{vk}_N) \leftarrow \text{KGen}(1^\lambda)\) and sends \( \text{vk}_N \) to \( A \).

**Queries on \( H_c \).** On input \( x = (T(u), t, m) \), if \( HT_c[x] \) is already set, \( B \) returns \( HT_c[x] \). Else, if \( t = t_N \), they increment \( ctr \) and set \( HT_c[x] = h_{ct} \). Else they sample \( HT_c[x] \leftarrow U(C) \). Finally, they output \( c = HT_c[x] \).

**Signing queries.** \( B \) follows the honest signing procedure \( \text{Sig} \) from \( \Pi_{\text{AS}} \) for \( sk_N \) on input message \( m \).

** Forgery.** Suppose that \( A \) outputs a forgery \( \sigma_{agg} = (u, (T(u)), z, ...) \) and the \( N \)-th signature \( \sigma_N = (u_N, z_N) \) together with a message \( m_N \), defining the message vector \( M = [M'm_N] \), the verification key vector \( VK = [VK][t_N] \) and the signature vector \( \Sigma = [\Sigma]\sigma_N \). Without loss of generality we assume that \( HT_c \) was programmed on \( x = (T(u_N), t_N, m_N) \) and thus \( c_N = h_j \) for some counter index \( j_f \). If not, \( A \) would only have a probability of \( 1 - 1/|C| \) to guess the correct \( c_N \). If \( \sigma_{agg} = \text{AggSig}(VK, M, \Sigma) \) and if \( \text{AggVf}(VK, M, \sigma_{agg}) = 1 \), then \( B \) outputs the tuple \((j_f, \text{out})\), where \( \text{out} = (VK, M, \sigma_{agg}, C) \), with \( C = (c_j)_{j \in [N]} \) such that \( c_j = H_c(T(u_j), t_j, m_j) \). Else, \( B \) outputs \((0, \bot)\).

For \( j = 0, 1, 2 \), let \( \Pr[G_j] \) denote the probability that \( B \) doesn’t output \((0, \bot)\) in game \( G_j \). It yields \( \Pr[G_0] = \text{Adv}_{\text{AggSig}}A \).

\( G_1 \): The game \( G_1 \) is identical to the previous game \( G_0 \) except that \( B \) doesn’t generate the signature honestly, but instead simulates the transcript without using the secret key \( sk_N \).
Signing queries. On input message \( m \), \( \mathcal{B} \) samples \( c \leftarrow U(C) \) and \( z \leftarrow D^{t+k}_s \). They compute \( u = [A|I_k] \cdot z - t_N \cdot c \) and program \( c = HT_c[x] \) with \( x = (T(u), t_N, m) \) afterwards. Finally, \( \mathcal{B} \) outputs \( \sigma = (u, z) \) with probability \( 1/M \).

Due to the rejection sampling, the distribution of \( z \) is identical in both signing variants (see [DOT12] Lem. 4) for more details). The only difference between the actual and the original signing algorithm is that now the output of \( H_c \) is programmed at the end, without checking whether it has already been set for \( x \). Following the same argument as in [Lyu12, Lem. 5.3] this happens only with negligible probability and thus \( |\Pr[G_2] - \Pr[G_1]| \leq \text{negl}(\lambda) \).

\( G_2 \): The game \( G_2 \) is identical to the previous game \( G_1 \) except how \( \mathcal{B} \) generates the \( v_{k_N} \) during the setup phase.

Setup. \( \mathcal{B} \) samples \( t_N \leftarrow U(R^k_q) \), sets \( v_{k_N} = t_N \) and outputs \( v_{k_N} \) to \( \mathcal{A} \).

As the signing queries are answered without using the corresponding secret key \( sk \), \( \mathcal{B} \) can replace \( v_{k_N} \) by a random vector without \( \mathcal{A} \) noticing, assuming the hardness of M-LWE\(_{k,\ell,b}\). Thus \( |\Pr[G_2] - \Pr[G_1]| \leq \text{Adv}_{\text{M-LWE}} \).

Now comes the final step of the proof. Note that in game \( G_2 \) the matrix \( A' = [A|t_N] \) follows the uniform distribution over \( R^N_q \).

We construct another algorithm \( \mathcal{B}' \) around \( \mathcal{B} \) who invokes the General Forking Lemma (Lemma 3), where the input generator \( \text{IGen} \) is defined to output \( A' \).

Let \( \text{acc} \) denote the accepting probability of \( \mathcal{B} \) and \( \text{frk} \) the forking probability of \( F_B \) as defined in Lemma 3. Thus, the forking algorithm \( F_B \) outputs with probability \( \text{frk} \) two different outputs \( \text{out}, \text{out}' \neq (\perp, \perp) \), where \( \Pr[F_B] = \text{acc} \leq N_q/|C| + \sqrt{N_q \cdot \text{frk}} \). Let \( \text{out} = (\mathcal{V}_M, M, \sigma_{\text{agg}}, C) \) and \( \text{out}' = (\mathcal{V}_M, \tilde{M}, \tilde{\sigma}_{\text{agg}}, C) \).

During the forking, neither \( \mathcal{B} \) not \( \mathcal{A} \) is aware of being rewinded. In particular, before arriving at the fork, they will behave exactly the same in both executions. As \( \mathcal{A} \) fixed \( (m_j, t_j, z_j, u_j)_{j \in [N-1]} \) before receiving the challenge key and \( \mathcal{B} \) verified that the same signatures where used in the aggregate signature, it is guaranteed that \( \mathcal{A} \) queried the random oracle \( H_c \) on the corresponding inputs before the fork. As the random coins of \( \mathcal{B} \) are the same in both executions and as the simulation of \( H_c \) is for \( t_j \neq t_N \) independent of the input \( (h_j)_{j \in [N_q]} \) to \( \mathcal{B} \), we have \( t_j = \tilde{t}_j \) and \( c_j = \tilde{c}_j \) for all \( j \in [N-1] \). Further it yields \( M = \tilde{M} \) and \( \tilde{u} = \tilde{u} \).

As in both cases, the forgery passes validation, it yields \( \|z\|_2, \|\tilde{z}\|_2 < \sqrt{N} \cdot B \).

Additionally, \( [A|I_k] \cdot z = \sum_{j \in [N]} t_j \cdot c_j + \tilde{u} \) and \( [A|I_k] \cdot \tilde{z} = \sum_{j \in [N]} \tilde{t}_j \cdot \tilde{c}_j + \tilde{u} \).

Hence, we can deduce that \( [A|I_k] \cdot (z - t_NC_N) = [A|I_k] \cdot (\tilde{z} - t_N \tilde{c}_N) \). In other words, \( \mathcal{B}' \) can compute the vector \( x = (z - \tilde{z}, \tilde{c}_N - c_N)^T \) which is a solution to the M-SIS problem for the matrix \( A' \). The Euclidean norm of the vector \( x \) is bounded above by \( \|x\|_2 \leq 2(\sqrt{N}B + \sqrt{d}) = b \). This implies that \( \text{frk} \leq \text{Adv}_{\text{M-SIS}} \).

Overall, we get

\[
\text{Adv}_{\text{AggSig}} \leq \text{Adv}_{\text{M-LWE}_{k,\ell,b}} + N_q/|C| + \sqrt{N_q \cdot \text{Adv}_{\text{M-SIS}_{k,\ell,b}}} + \text{negl}(\lambda).
\]

Implicitly, we assumed that \( \mathcal{A} \) queries \( H_c \) for the challenge key \( v_{k_N} \) and the message \( m_N \) using the same commitment \( u_N \) leading to the same \( T(u_N) \). If \( \mathcal{A} \)
would choose another pre-image $\tilde{u}_N$ such that $T(\tilde{u}_N) = T(u_N)$, then this would change the sum so that $\tilde{u} \neq u$. In this case, $\mathcal{B}'$ simply invokes the General Forking Lemma again. As there are only finitely many pre-images, after at most finitely many rewinds, $\mathcal{B}'$ gets the 'good' forgery. □

Remark 3. Using the General Forking Lemma introduces two disadvantages: On the one hand, it causes an important loss in the reduction. On the other hand, we currently don’t know how to extend it to the so-called quantum ROM, where an adversary has quantum access to the random oracle (and classical access to the signing oracle). Abdalla et al. [AFLT16] proposed a much tighter reduction for lattice-based signature schemes following the FSwA paradigm by introducing lossy identification schemes. Kiltz et al. [KLS18] used their techniques to construct a generic framework for tightly secure signatures in the quantum ROM.

The key idea behind lossy identification schemes is that verification keys can be replaced by random, so-called lossy, keys. Then, using a lossy key, for a fixed commitment (in our scheme this is the vector $u$) with overwhelming probability there exists at most one transcript that verifies. In our case, however, it is not clear that lossiness holds due to the compression. More precisely, fixing the commitments $T(u_j)$ to the hash function $H$, does not fix the underlying vectors $(u_j)$ as one can easily generate other vectors $(v_j)$ that are mapped to the same values by $T$, i.e., $T(u_j) = T(v_j)$ for all $j$. Hence, the sum $\tilde{u} = \sum_j u_j$ in general doesn’t equal the sum $\tilde{v} = \sum_j v_j$ and thus an adversary may find two valid transcripts.

4.3 Rogue-Attack in the Aggregate Chosen-Key Model

The aggregate signature scheme $\Pi_{AS}$ as presented in Section 3 is not secure in the aggregate chosen-key model as introduced by Boneh et al. [BGLS03]. In this security game, the adversary $\mathcal{A}$ is given a challenge key $t_N$ (representing the verification key of one honest signing party) and has access to a signing oracle with respect to this challenge key. Informally speaking, in order to win the game, $\mathcal{A}$ has to output a set of messages $M$, a set of verification keys $VK$ and a forgery $\sigma_{agg}$ which is a valid aggregate signature on $M$ using $VK$, where $VK$ contains $vk_N$. In the following we describe a simple attack in this security model, which was first observed by Zhao [Zha19] in the context of a Schnorr-based aggregate signature scheme on elliptic curves.

1. Let $t_N$ be the challenge key given to the adversary $\mathcal{A}$.
2. $\mathcal{A}$ generates the remaining key pairs $(s_j, t_j) \leftarrow KGen$ for $j \in [N-1]$ and selects arbitrary messages $m_j$ for $j \in [N]$.
3. They sample $y_N \leftarrow D_{s}^{k+k}$, set $u_N = [A][k] \cdot y_N$ and query the random oracle to obtain $c_N = H_e(T(u_N), t_N, m_N)$.
4. Now, $\mathcal{A}$ can prepare a 'rogue-commitment' $u_{N-1}$ by sampling $y_{N-1} \leftarrow D_{s}^{k+k}$ and setting $u_{N-1} = -t_N \cdot c_N + [A][k] \cdot y_{N-1}$, depending on the challenge $c_N$ for the challenge key $t_N$.
5. They compute $c_{N-1} = H_e(T(u_{N-1}), t_{N-1}, m_{N-1})$. 
6. They set $z_{N-1} = y_{N-1} + s_{N-1} \cdot c_{N-1}$ and apply the rejection sampling in order to make the distribution of $z_{N-1}$ independent of $s_{N-1}$.

7. For the remaining $j \in [N-2]$, they follow the honest signature procedure: they sample $y_j \leftarrow D_s^{k+\ell}$, set $u_j = [A|I_k] \cdot y_j$, compute $c_j = H_t(T(u_j), t_j, m_j)$ and set $z_j = s_j \cdot c_j + y_j$. Then they apply the rejection sampling in order to make the distribution of $z_j$ independent of $s_j$.

8. Finally, they output the forgery $\sigma_{agg} = (\hat{u}, T(u_j), z)$, where $\hat{u} = \sum_{j=1}^{N} u_j$ and $z = y_N + \sum_{j=1}^{N-1} z_j$.

We claim that the forgery passes verification. As all $z_j \sim D_s^{k+\ell}$ and $y \sim D_s^{k+\ell}$, the sum $z \sim D_s^{N \cdot k+\ell}$. Thus, with overwhelming probability the norm of $z$ is bounded above by $\sqrt{N} \cdot B$. Further, it yields $T(\hat{u}) = \sum_j T(u_j)$, as the $T(u_j)$ are computed by honestly compressing the corresponding $u_j$. It remains to show that $[A|I_k] \cdot z = \sum_{j=1}^{N} (t_j \cdot c_j) + \hat{u}$.

\[
[A|I_k] \cdot z = [A|I_k] \cdot y_N + \sum_{j=1}^{N-1} [A|I_k] \cdot z_j \\
= u_N + [A|I_k] \cdot z_{N-1} + t_{N-1} \cdot c_{N-1} + \sum_{j=1}^{N-2} t_j \cdot c_j + u_j \\
= u_N + u_{N-1} + t_N \cdot c_N + t_{N-1} \cdot c_{N-1} + \sum_{j=1}^{N-2} t_j \cdot c_j + u_j \\
= \sum_{j=1}^{N} t_j \cdot c_j + \hat{u},
\]

where we used that $[A|I_k] \cdot y_{N-1} = u_{N-1} + t_N \cdot c_N$. In this attack the adversary exploits the fact that they can define a rogue-commitment with respect to the challenge $c_N$. In our security model, this is prevented as the adversary has to fix the signatures they are going to use in the forgery before receiving the challenge key $t_N$.

Note that Zhao proposed a modified protocol to prevent this type of attacks at the expense of introducing a new (non-standard) problem. This modified protocol was later attacked by Hanaoka et al. [HOS+20]. We leave it as an open problem to provide an aggregate signature scheme which is provably secure in the chosen-key model from Boneh et al. [BGLS03].

Acknowledgments

This work was supported by the European Union PROMETHEUS project (Horizon 2020 Research and Innovation Program, grant 780701). It has also received a French government support managed by the National Research Agency in the
"Investing for the Future" program, under the national project RISQ P141580-266001 / DOS0044216. Katharina Boudgoust is funded by the Direction Générale de l’Armement (Pôle de Recherche CYBER). We warmly thank Akira Takahashi for making us aware of the attacks against Schnorr-based aggregate signatures in the chosen-key model and discussions with him and Claudio Orlandi. We also would like to thank Olivier Sanders for an interesting exchange regarding the security proof.

References


