Sequential Half-Aggregation of Lattice-Based Signatures

Katharina Boudgoust[1] and Akira Takahashi[2]

[1] Aarhus University, Denmark
katharina.boudgoust@cs.au.dk
[2] Aarhus University, Denmark
takahashi@cs.au.dk

December 29, 2022

Abstract. Sequential aggregate signature (SAS) schemes (Boneh et al., Eurocrypt'04) allow a group of signers to sequentially combine signatures on distinct messages in a compressed manner. The present paper constructs the first Fiat-Shamir with Aborts (Lyubashevsky, Eurocrypt'12) based SAS with signature size smaller than naive concatenation and compares it with existing lattice-based SAS which follow the GPV-paradigm (Gentry et al., STOC'06).

Table of Contents

1 Introduction

Aggregate signature (AS) schemes, introduced by [BGLS03], allow N signers to individually produce signatures $\sigma_1, \ldots, \sigma_N$ on distinct messages $m_1, \ldots, m_N$, and later combine them into a single, compact signature σ_{AS}. Such σ_{AS} can be verified with respect to the participants' verification keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_N$. Classical applications of aggregate signatures include certificate chains: in a public key infrastructure (PKI) one has to include their certificate in every sent message, which itself comes from a chain of certificates issued by different authorities. Since the naive concatenation of single-user signatures significantly adds to the certificate chain (e.g., [BGLS03] reports 15% of a typical X.509 certificate length is occupied by the signature), it is paramount to replace them with a compact, aggregated signature to save bandwidth. In the literature, essentially two different paradigms of fully compact aggregate signatures have been proposed: (1) dedicated constructions based on bilinear pairings [BGLS03, BNN07], and (2) generic solutions exploiting iO [HKW15] or non-interactive arguments [DGKV22, WW22, ACL$^+$22], where a signature aggregator produces a succinct proof of knowledge of N valid signatures.

There also exists a slightly restricted primitive called *sequential aggregate signatures* (SAS) [LMRS04]. In this setting, signing and aggregation are carried out altogether: signer i associated with pk_i receives from signer $i-1$ *aggregate so-far* σ_{i-1} with a key-message list $L_{i-1} = (\mathsf{pk}_1, m_1, \ldots, \mathsf{pk}_{i-1}, m_{i-1})$, adds a signature on the message m_i of their own choice to produce σ_i, and then passes along σ_i and $L_i = L_{i-1}||(\mathsf{pk}_i, m_i)$ to the next signer $i+1$. Unlike general aggregate signatures, SAS require round-robin communication among signers, which however fits well in typical application scenarios such as a certificate chain. A plethora of work proposed highly efficient, constant-size SAS using pairings [LOS$^+$06, BGOY07, BNN07, FLS12] or assuming the existence of trapdoor permutations (TDP) [LMRS04, Nev08, BGR12, GOR18].

Half-Aggregation of Fiat-Shamir Signatures. Perhaps unsurprisingly, not many aggregation methods tailored to *Fiat-Shamir signatures* [FS87] such as Schnorr [Sch91] are known.[3] Recall that Fiat-Shamir signatures are typically constructed from three-round Σ-protocols [Cra96]: the signer invokes the underlying Σ-protocol prover to generate the first-round *commit* value u, samples random *challenge* c by hashing u together with the message m to be signed, creates *response* z, and outputs $\sigma = (c, z)$ as a signature. The verifier then reconstructs u from (pk, c, z) through certain algebraic operations and checks the recomputed hash against c. Equivalently, the signer can set $\sigma = (u, z)$ and the verifier recomputes the hash c, while checking if a certain relation between c and (pk, u, z) holds. The difficulty of aggregating Fiat-Shamir mainly lies in the challenge hash function: since its typical instantiation such as SHA-256 has no algebraic structure, it does not blend well with nice homomorphic properties of the underlying Σ-protocol transcript. This is why the existing approaches (e.g., [BN06, DEF$^+$19, NRS21]) require (at least) two rounds of interaction so that all signers can first agree on a combined u that leads to the same challenge c, from which they can compute shares of z.

To avoid interaction, recent papers proposed *half-aggregation* of Schnorr/EdDSA [CGKN21, Kas22, CZ22]. These are middle ground solutions where only the u or the z component gets aggregated, and the other part consists of a concatenation of N partial signatures. Although it is asymptotically no better than the trivial concatenation of N signatures, reducing the signature size by a constant factor has meaningful implications in practice, e.g., in certain cases the entire certificate chain of size $O(N)$ needs to be transmitted anyway.

Another possible approach would be adapting one of the aforementioned generic solutions and having an aggregator node to prove the knowledge of N tuples of the form (u, c, z) satisfying the verification conditions descried as a circuit. However, the prover's complexity likely hinges on mixture of algebraic operations and non-algebraic hash computation in verification.

Aggregate Signatures from Lattices. Given that NIST has announced in their post-quantum cryptography standardization project two signature finalists—Falcon [PFH$^+$20] and $\mathsf{Dilithium}$ [LDK$^+$20]—based on (structured) lattice assumptions, a natural question is whether tailor-made aggregate signatures can be instantiated using lattices (instead of generic solutions such as [DGKV22, ACL$^+$22]). Both finalists represent the two major design principles to build lattice-based signatures: $\mathsf{Dilithium}$ follows Lyubashevsky's Fiat-Shamir with Aborts (FSwA) paradigm [Lyu09, Lyu12] and Falcon is a GPV-type signature using preimage sampleable trapdoor functions [GPV08].

There are a limited number of proposals within the FSwA paradigm. Boneh and Kim [BK20] presented a lattice-based instantiation of [BN06] but it requires three rounds of interactions. Boudgoust and Roux-

[3] It is well known that *interactive* multi-signatures can be generically converted to interactive aggregate signatures by asking all participants to sign a concatenation of N messages and public keys [BN06, DEF$^+$19, NRS21]. However, this requires the signers to agree on all N messages and who they co-sign with in advance, and does not fit in the typical use cases of aggregate signatures such as a certificate chain.

Langlois [BR21] are the first to securely instantiate non-interactive half-aggregation of FSwA assuming the hardness of (the module versions of) LWE and SIS. From a high level perspective, they adapt the half-aggregation of Schnorr [CGKN21] to the lattice-setting. Whereas in Schnorr, it doesn't really matter whether we output $\sigma = (u, z)$ or $\sigma = (c, z)$, it makes a big difference in the lattice setting. The signature size significantly decreases in the second case. During the half-aggregation of [BR21], only the z-parts are aggregated, but all the u-parts are transmitted. Note that it's not sufficient to transmit all the c-parts, as we can't recover the different commitments anymore from an aggregated response. However, we need every single commitment in order to verify an aggregate signature. In consequence, the provably secure version of their construction outputs a signature $\sigma_{\mathsf{AS}} = (u_1, \ldots, u_N, z)$ which is *always* larger than the naive concatenation of N signatures $\sigma_{\mathsf{con}} = (c_1, z_1, \ldots, c_N, z_N)$. The MMSAT scheme [DHSS20] is a candidate half-aggregate signature scheme based on a non-standard lattice problem, called the Partial Fourier Recovery problem. However, it turned out that the security proof is flawed and even simple forgery attacks exist [BR21]. Regarding sequential aggregation, the only known lattice-based solutions we are aware of follow the GPV-paradigm [EB14, WW19]. Given all this, we are motivated to ask the following question in this paper:

> Can we construct a non-interactive half-aggregate FSwA signature scheme (1) with a signature size smaller than the naive concatenation, and (2) without invoking expensive generic solutions?

1.1 Our Contributions

- We present a sequential aggregate signature based on the Fiat-Shamir with Aborts framework. The aggregation paradigm closely follows recent Schnorr-based SAS due to Chen and Zhao [CZ22]. As elaborated before, the main obstacle in previous works is that without interaction it is difficult to aggregate the commitments $u_1, \ldots, u_N$ that are responsible for the large aggregate signature size in [BR21]. If, however, we place ourselves in the *sequential* aggregate model, we can aggregate over the u-parts by letting the parties sign one after each other. A sequential aggregate signature of our construction now is of the form $\sigma_{\mathsf{SAS}} = (u, z_1, \ldots, z_N)$. Once the size of $(c_1, \ldots, c_N)$ is larger than the size of u, our SAS produces signatures that are indeed smaller than the trivial concatenation σ_{con}. Unfortunately, when looking at the ratio between σ_{SAS} and σ_{con}, it's the $(z_1, \ldots, z_N)$-part (that both have in common) that makes up for most of the signature size and hence the compression rate is close to 1. Although our concrete parameter estimates indicate the output signature is only $\sim 1\%$ smaller than the naive concatenation, we believe ours to be an important step towards better understanding the possibilities and limits of lattice-based aggregate signatures.
- The security of our scheme tightly reduces to the existential unforgeability of the standard single-user FSwA scheme instantiated with structured lattices. We prove security in the so-called *full history* setting of SAS. Further, we also discuss its security in a new model that has been introduced in [CZ22], which we call the *partial-signature history-free* security model. Although our construction closely follows the one of [CZ22], our security proof is more involved because of subtleties that arise in the lattice setting. We have to consider several bad events that might happen and bound their probability.
- We also compare our scheme with the two existing lattice-based SAS [EB14, WW19] following the GPV-paradigm. As in the lattice setting we only have so-called preimage sampleable trapdoor functions (and no TDP), they can't achieve constant-size SAS neither. The upshot is that neither of them saves more than 4% of signature size if a fair comparison is made against the naive concatenation and taking recent advances [ETWY22] into account.
- As a separate contribution, we point out insecurities of two existing aggregate signature schemes explicitly instantiated with NIST finalists: (1) Falcon-based SAS of [WW19] does not guarantee the claimed security property due to the existence of a forgery attack, and (2) $\mathsf{Dilithium}$-based interactive multi-signature of [FH20] (which can be generically turned into an aggregate signature using [BN06]'s trick) leaks part of the secret key due to the misuse of Bai-Galbraith's $\mathsf{HighBits}$ optimization trick [BG14]. Given the existence of attacks and our parameter estimates, we then conclude that concretely efficient aggregation of the lattice-based NIST finalists is still an unexplored area and highlight it as an interesting direction for future work.

1.2 Other Related Work

Imposing a sequential way of signing is not the only way how to restrict the model of aggregate signatures. Other works look for instance at a synchronous model [GR06, AGH10], where signatures are aggregated together if they have been issued at the same time interval. One recent result studies lattice-based aggregate signatures in the synchronous model [FSZ22].

As already mentioned before, a related concept are multi-signatures [MOR01, Bol03], where we allow the parties to interact with each other. There are several recent results on lattice-based multi-signatures [DOTT21, BTT22] and we refer to the references herein.

2 Preliminaries

Notations. For any positive integer N, we denote by $[N]$ the set $\{1, \ldots, N\}$. For a finite set S, we denote its cardinality by $|S|$ and the uniform distribution over S by $U(S)$. We simply write $s \xleftarrow{\$} S$ to indicate sampling s from $U(S)$. For a probability distribution $\mathscr{D}$, we write $s \leftarrow \mathscr{D}$ to indicate sampling s from $\mathscr{D}$; for a randomized (resp. deterministic) algorithm $\mathcal{A}$ we write $s \leftarrow \mathcal{A}$ (resp. $s := \mathcal{A}$) to indicate assigning an output from $\mathcal{A}$ to s. Throughout, the security parameter is denoted by λ. The abbreviation PPT stands for probabilistic polynomial-time.

Following [LDK+20] $r' = r \mod^+ \alpha$ denotes the unique integer $r' \in [0, \alpha)$ such that $r' \equiv r \bmod \alpha$. For an even (resp. odd) positive integer α, $r' = r \mod^{\pm} \alpha$ denotes the unique integer $r' \in (-\alpha/2, \alpha/2]$ (resp. $r' \in [-(\alpha-1)/2, (\alpha-1)/2]$) such that $r' \equiv r \bmod \alpha$.

Throughout the paper, we work over the ring $R =: \mathbb{Z}[X]/\langle X^n + 1\rangle$, where n is a power of 2. For any ring element $r \in R$, we define $\|r\|_2, \|r\|_1$ and $\|r\|_\infty$ to be the respective norms of its coefficient vector. For some prime q, we define $R_q := R/(qR)$ and for some positive integer γ, we set $S_\gamma := \{r \in R \colon \|r\|_\infty \leq \gamma\}$.

2.1 Probability and Regularity

The Rényi divergence (RD) defines a measure of distribution closeness. We follow [BLR+18] and set the RD as the exponential of the classical definition.

Definition 2.1 (Rényi Divergence). *Let P and Q be two discrete probability distributions such that $\mathsf{Supp}(P) \subseteq \mathsf{Supp}(Q)$. The* Rényi divergence *(of order 2) is defined by*

$$\mathrm{RD}_2(P \| Q) = \sum_{x \in \mathsf{Supp}(P)} \frac{P(x)^2}{Q(x)}.$$

The RD fulfills the following properties, as proved in [vEH14].

Lemma 2.2. *Let P, Q be two discrete probability distributions with $\mathsf{Supp}(P) \subseteq \mathsf{Supp}(Q)$.*

Data Processing Inequality: $\mathrm{RD}_2(P^g \| Q^g) \leq \mathrm{RD}_2(P \| Q)$ *for any function g, where P^g (resp. Q^g) denotes the distribution of $g(y)$ induced by sampling $y \leftarrow P$ (resp. $y \leftarrow Q$),*

Probability Preservation: *Let $E \subset \mathsf{Supp}(Q)$ be an event, then*

$$P(E) \leq \sqrt{Q(E) \cdot \mathrm{RD}_2(P \| Q)}.$$

In Section 3, we need the following regularity result.

Lemma 2.3 ([BJRW22, Lem. 2.7] Simplified). *Let $k, \ell, q, \eta \in \mathbb{N}$ such that q is prime. Further, let $R = \mathbb{Z}[X]/\langle X^n + 1\rangle$, where n is a power of 2. Then,*

$$\mathrm{RD}_2((\mathbf{A}, \mathbf{A}\mathbf{y}') \| (\mathbf{A}, \mathbf{v})) \leq \left(1 + \frac{q^k}{(2\gamma+1)^\ell}\right)^n,$$

where $\mathbf{A} \xleftarrow{\$} R_q^{k \times \ell}$, $\mathbf{y}' \xleftarrow{\$} S_\gamma^\ell = \mathscr{D}^\ell$ and $\mathbf{v} \xleftarrow{\$} R_q^k$.

In order to obtain a constant Rényi divergence, we require

$$\ell \geq k \cdot \frac{\log_2 q}{\log_2(2\gamma+1)} + O\left(\frac{\log_2 q}{\log_2(2\gamma+1)}\right). \tag{1}$$

Remark 2.4. Alternatively, if one prefers the discrete Gaussian distribution for $\mathscr{D}$, one can use the regularity result [LPR13, Cor. 7.5]. It comes with the advantage that it holds for any $q \geq 2$, not necessarily prime, and that the parameter ℓ can be arbitrarily close to k, in particular $\ell = k$ is a possible choice, which is common in practice. However, the Gaussian distribution is with respect to the canonical embedding, which is not implementation friendly and requires to switch between canonical and coefficient embedding throughout the scheme. Further, the resulting Gaussian width is much larger than parameters used in practice.

2.2 Module Lattice Problems

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

Definition 2.5 (M-LWE)**.** *Let $k, \ell, \eta \in \mathbb{N}$. The* Module Learning With Errors problem $\mathsf{M\text{-}LWE}_{k,\ell,\eta}$ *is defined as follows. Given $\mathbf{A} \xleftarrow{\$} R_q^{k\times\ell}$ and $\mathbf{t} \in R_q^k$. Decide whether $\mathbf{t} \xleftarrow{\$} R_q^k$ or if $\mathbf{t} = [\mathbf{A}|\mathbf{I}_k] \cdot \mathbf{s}$, where $\mathbf{s} \xleftarrow{\$} S_\eta^{\ell+k}$.*

The M-LWE assumption states that no PPT algorithm can distinguish between 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 2.6 (M-SIS)**.** *Let $k, \ell, b \in \mathbb{N}$. The* Module Short Integer Solution problem $\mathsf{M\text{-}SIS}_{k,\ell,b}$ *is as follows. Given a uniformly random matrix $\mathbf{A} \xleftarrow{\$} R_q^{k\times\ell}$. Find a non-zero vector $\mathbf{s} \in R_q^{k+\ell}$ such that $\|\mathbf{s}\|_2 \leq b$ and $[\mathbf{A}|\mathbf{I}_k] \cdot \mathbf{s} = \mathbf{0} \in R_q^k$.*

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.

2.3 Fiat-Shamir with Aborts Signatures

In this paper, we build a sequential aggregate signature FSwA-SAS starting from a well-studied signature scheme FSwA-S = (Setup, Gen, Sign, Ver) whose definition we recall in Algorithm 1. It follows the so-called *Fiat-Shamir with Aborts* paradigm [Lyu09, Lyu12] and can be seen as the module variant of [GLP12] or the 'vanilla' flavor of Dilithium.

Modification. A difference to the standard design is that instead of outputting $\sigma = (c, \mathbf{z})$, we output $\sigma = (\mathbf{u}, \mathbf{z})$. For a single signature, both cases are equivalent, as $\mathbf{u}$ defines c via the hash function H (and the public key $\mathbf{t}$ and the message m) and c defines $\mathbf{u}$ via the equation $\mathbf{u} = \bar{\mathbf{A}}\mathbf{z} - c \cdot \mathbf{t}$. However, this is not the case for a (sequential) aggregate signature scheme and we thus need to transmit the information $\mathbf{u}$.

Distribution $\mathscr{D}$. During the signing algorithm Sign, the FSwA-S scheme uses a distribution $\mathscr{D}$ to sample a vector of ring elements of short norm over R. In the literature, mainly two different ways of instantiating $\mathscr{D}$ are studied. The first uses discrete Gaussian distributions (as for instance in [Lyu12]) and the second uses the uniform distribution over a bounded set, i.e., $\mathscr{D} = U(S_\gamma)$ for some $\gamma \ll q$ (as for instance in [GLP12]). The concrete instantiation of $\mathscr{D}$ then influences the choice of the rejection algorithm RejSamp during signing and of the bound B during verification. In this paper, we focus on the latter as this is the choice commonly used in practice, as for instance in Dilithium. In this case, the algorithm RejSamp outputs $\perp$ if $\|\mathbf{z}\|_\infty > \gamma - \kappa \cdot \eta$, else it outputs $\mathbf{z}$. Accordingly, the bound is set to $B = \gamma - \kappa \cdot \eta$.

Security. Overall, the UF-CMA security of the scheme FSwA-S as specified in Algorithm 1 is based on the hardness of M-LWE and M-SIS. For the reason of space limits, we don't go into details here, but refer the interested reader to the original security proofs in [Lyu12, GLP12] in the random oracle model.

2.4 Sequential Aggregate Signatures

Sequential aggregate signatures (SAS) were first introduced by Lysyanskaya et al. [LMRS04]. We recall now the syntax of a (full-history) SAS scheme, together with the definitions of correctness and security following the notations of Gentry et al. [GOR18]. See also Appendix **??** for a discussion of alternative definitions of SAS schemes and corresponding security notions.

Definition 2.7 (SAS)**.** *A* sequential aggregate signature *scheme* (SAS) *for a message space M consists of a tuple of* PPT *algorithms* SAS = (Setup, Gen, SeqSign, SeqVerify) *defined as follows:*

Setup(1^λ) $\to$ pp*: On input the security parameter λ, the setup algorithm outputs the public parameters* pp*.*

Gen(pp) $\to$ (sk, pk)*: On input the public parameters* pp*, the key generation algorithm outputs a pair of secret key* sk *and public key* pk*.*

Algorithm 1: FSwA-S

The challenge space is $\mathsf{Ch} := \{c \in R : \|c\|_\infty = 1 \wedge \|c\|_1 = \kappa\}$ and the message space is $M = \{0,1\}^l$. The random oracle is $\mathsf{H} : \{0,1\}^* \to \mathsf{Ch}$.

$\mathsf{Setup}(1^\lambda)$

$\mathbf{A} \xleftarrow{\$} R_q^{k\times\ell}$
$\bar{\mathbf{A}} := [\mathbf{A}|\mathbf{I}_k]$
return $\bar{\mathbf{A}}$

$\mathsf{Gen}(\bar{\mathbf{A}})$

$\mathbf{s} \xleftarrow{\$} S_\eta^{\ell+k}$
$\mathbf{t} := \bar{\mathbf{A}}\mathbf{s} \bmod q$
$\mathsf{sk} := \mathbf{s}$
$\mathsf{pk} := \mathbf{t}$
return $(\mathsf{sk}, \mathsf{pk})$

$\mathsf{Ver}(\mathsf{pk}, \sigma, m)$

$(\mathbf{u}, \mathbf{z}) := \sigma$
$\mathbf{t} := \mathsf{pk}$
$c := \mathsf{H}(\mathbf{u}, \mathbf{t}, m)$
if $\|\mathbf{z}\|_\infty \leq B \wedge \bar{\mathbf{A}}\mathbf{z} = c \cdot \mathbf{t} + \mathbf{u}$ **then**
  return 1
else
  return 0

$\mathsf{Sign}(\mathsf{sk}, m)$

$\mathbf{s} := \mathsf{sk}$
$\mathbf{t} := \bar{\mathbf{A}}\mathbf{s} \bmod q$
$\mathbf{z} := \perp$
while $\mathbf{z} := \perp$ **do**
  $\mathbf{y} \xleftarrow{\$} \mathscr{D}^{\ell+k}$
  $\mathbf{u} := \bar{\mathbf{A}}\mathbf{y} \bmod q$
  $c := \mathsf{H}(\mathbf{u}, \mathbf{t}, m)$
  $\mathbf{z} := c \cdot \mathbf{s} + \mathbf{y}$
  $\mathbf{z} := \mathsf{RejSamp}(\mathbf{z}, c \cdot \mathbf{s})$
$\sigma := (\mathbf{u}, \mathbf{z})$
return σ

$\mathsf{SeqSign}(\mathsf{sk}_i, m_i, L_{i-1}, \sigma_{i-1}) \to \sigma_i$: *On input a secret key* sk_i*, a message* $m_i \in M$*, a list* L_{i-1} *with* $L_{i-1} := (\mathsf{pk}_1, m_1)||\dots||(\mathsf{pk}_{i-1}, m_{i-1})$*, and a so-far signature* σ_{i-1}*, the sequential signing algorithm outputs a new so-far signature* σ_i*.*

$\mathsf{SeqVerify}(L_N, \sigma_N) \to \{0,1\}$: *On input a list* L_N *of* N *message-public-key pairs and a sequential aggregate signature* σ_N*, the sequential verification algorithm either outputs* 1 *(accept) or* 0 *(reject).*

For convenience, given a list $L_j = (\mathsf{pk}_1, m_1)||\dots||(\mathsf{pk}_j, m_j)$, we denote by L_i its ith prefix $L_i := (\mathsf{pk}_1, m_1)||\dots||(\mathsf{pk}_i, m_i)$ for $1 \leq i < j$.

Definition 2.8 (Correctness). *Let* $\mathsf{SAS} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{SeqSign}, \mathsf{SeqVerify})$ *be a sequential aggregate signature scheme for a message space* M*. It is called* correct *if for all* $\lambda, N \in \mathbb{N}$ *it yields*

$$\Pr\left[\mathsf{SeqVerify}(L_N, \sigma_N) = 1\right] = 1 - \mathsf{negl}(\lambda),$$

where $m_i \in M$*,* $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$*,* $(\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathsf{Gen}(\mathsf{pp})$*,* $L_i = (\mathsf{pk}_1, m_1)||\dots||(\mathsf{pk}_i, m_i)$ *and* $\sigma_i \leftarrow \mathsf{SeqSign}(\mathsf{sk}_i, m_i, L_{i-1}, \sigma_{i-1})$ *for all* $i \in [N]$*. Let* $L_0 = \varnothing$ *and* $\sigma_0 = (\mathbf{0}, \mathbf{0})$*.*

Definition 2.9 (FH-UF-CMA Security). *A* SAS *scheme satisfies* <u>f</u>ull <u>h</u>istory <u>unf</u>orgeabilty against <u>c</u>hosen <u>m</u>essage <u>a</u>ttacks*, if for all* PPT *adversaries* $\mathcal{A}$*,*

$$\mathsf{Adv}_{\mathsf{SAS}}^{\mathsf{FH\text{-}UF\text{-}CMA}}(\mathcal{A}) := \Pr\left[\mathsf{FH\text{-}UF\text{-}CMA}_{\mathsf{SAS}}(\mathcal{A}, \lambda) = 1\right] = \mathsf{negl}(\lambda),$$

where the $\mathsf{FH\text{-}UF\text{-}CMA}_{\mathsf{SAS}}$ *game is described in Game 1.*

3 Sequential Half-Aggregation of FSwA Signatures

3.1 Definition and Correctness of the Scheme

Our scheme is described in Algorithm 2. The overall structure closely follows the one by Chen and Zhao [CZ22]. We remark that, for the sake of security proof, the key generation algorithm slightly differs

Game 1: $\mathsf{FH\text{-}UF\text{-}CMA}_{\mathsf{SAS}}(\mathcal{A}, \lambda)$

$\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$
$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(\mathsf{pp})$
$\mathcal{Q} := \varnothing$
$(L_N^*, \sigma_N^*) \leftarrow \mathcal{A}^{\mathsf{OSeqSign}}(\mathsf{pp}, \mathsf{pk})$
if $\mathsf{SeqVerify}(L_N^*, \sigma_N^*) \ \wedge \ \exists i^* \in [N] \colon (\mathsf{pk}_{i^*} = \mathsf{pk} \ \wedge \ (m_{i^*}, L_{i^*-1}) \notin \mathcal{Q})$ **then**
    return 1
else
    return 0

$\mathsf{OSeqSign}(m_i, \sigma_{i-1}, L_{i-1})$
$\sigma_i \leftarrow \mathsf{SeqSign}(\mathsf{sk}, m_i, L_{i-1}, \sigma_{i-1})$
$\mathcal{Q} := \mathcal{Q} \cup \{(m_i, L_{i-1})\}$
return σ_i

from the original one in Algorithm 1. It keeps regenerating a key pair until the public key $\mathbf{t}$ contains at least one invertible element. This terminates relatively quickly in practice. Let p_{inv} be the probability that $\mathbf{t} = \bar{\mathbf{A}}\mathbf{s}$ has at least one invertible coefficient over R_q, where $\mathbf{s}$ is uniformly sampled from $S_\eta^{\ell+k}$. Then the expected running time of Gen is $1/p_{\mathsf{inv}}$. One can experimentally find p_{inv} for each parameter set.

Remark 3.1. Whereas it seems to be hard to give unconditionally provable lower bounds for p_{inv} (at least in our parameter setting), it is possible to bound it assuming the hardness of M-LWE (which is also used in the security proof of FSwA-S). Let p_{R_q} denote the probability that an element of R_q sampled uniformly at random is invertible. There exist exact formulas to express this number, depending on the splitting behavior of the ideal generated by q in the ring R. For the fully splitting case, i.e., $q = 1 \bmod 2n$, it yields $p_{R_q} = (1 - 1/q)^n$, see for instance [LPR13, Claim 2.25]. Assuming the hardness of M-LWE, it yields $p_{\mathsf{inv}} = p_{R_q} + \mathsf{negl}(\lambda)$. If not, an adversary against M-LWE could simply test a given instance for invertibility.

Lemma 3.2 (Correctness). *The scheme* $\mathsf{FSwA\text{-}SAS} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{SeqSign}, \mathsf{SeqVerify})$ *as specified in Algorithm 2 is correct.*

Proof. We first verify that in the case of $N > 1$ the computed signature $\sigma_{N-1} = (\tilde{\mathbf{u}}_N, \mathbf{z}_1, \ldots, \mathbf{z}_N)$ is the correct previous so-far signature. As $\mathbf{z}_N$ has been correctly computed, it yields $\bar{\mathbf{A}} \cdot \mathbf{z}_N - \mathbf{t}_N \cdot c_N = \mathbf{u}_N$. Hence, $\tilde{\mathbf{u}}_{N-1}$ can be recovered via $\tilde{\mathbf{u}}_N - \mathbf{u}_N$ and thus $\sigma_{N-1} = (\tilde{\mathbf{u}}_{N-1}, \mathbf{z}_1, \ldots, \mathbf{z}_{N-1})$ is the correct previous so-far signature. Now, let's consider $N = 1$. It yields $\|\mathbf{z}_1\|_\infty \leq B$ and $\bar{\mathbf{A}} \cdot \mathbf{z}_1 = \mathbf{t}_1 \cdot c_1 + \tilde{\mathbf{u}}_1$ because of the linearity of matrix-vector multiplication over R_q. ◻

3.2 Security Proof

We focus on proving the FH-UF-CMA security (as in Definition 2.9) of Algorithm 2. We discuss in Appendix ?? alternative security notions for SAS and show that our scheme can also be proven secure with respect to them.

Theorem 3.3 (FH-UF-CMA security). *Let $k, \ell, n, q, \eta, \gamma, l \in \mathbb{N}$ such that n is a power of 2, q is prime and Equation 1 is fulfilled. Let p_{inv} be the probability that $\bar{\mathbf{A}}\mathbf{s}$ has at least one invertible coefficient over R_q, where $\mathbf{s}$ is uniformly sampled from $S_\eta^{\ell+k}$ and $\bar{\mathbf{A}} = [\mathbf{A}|\mathbf{I}_k]$ with $\mathbf{A}$ is uniformly sampled from $R_q^{k\times\ell}$, respectively. If the signature scheme* FSwA-S *with message space $M = \{0,1\}^l$, as described in Algorithm 1, is* UF-CMA *secure, then is the sequential aggregate signature* FSwA-SAS*, as described in Algorithm 2,* FH-UF-CMA *secure. Concretely, for any adversary $\mathcal{A}$ against* FH-UF-CMA *security that makes at most Q_h queries to the random oracle* H*, Q_s queries to the* OSeqSign *oracle and outputs a forgery with a history of length N, there exists an adversary $\mathcal{B}$ against* UF-CMA *security such that*

$$\frac{\mathbf{Adv}^{\mathsf{UF\text{-}CMA}}_{\mathsf{FSwA\text{-}S}}(\mathcal{B})}{p_{\mathsf{inv}}} + O\left(\frac{Q_s(Q_h + Q_s)}{q^{nk/2}}\right) + \frac{(Q_h + Q_s + 1)^2}{|\mathsf{Ch}|} + \frac{Q_s(2Q_h + 1)}{2^l} \geq \mathbf{Adv}^{\mathsf{FH\text{-}UF\text{-}CMA}}_{\mathsf{FSwA\text{-}SAS}}(\mathcal{A}),$$

and $\mathrm{Time}(\mathcal{B}) = \mathrm{Time}(\mathcal{A}) + O((N + Q_h)k\ell t_{\mathsf{pmul}})$*, where t_{pmul} is the time of polynomial multiplication in R_q.*

Algorithm 2: FSwA-SAS

The challenge space is $\mathsf{Ch} := \left\{c \in R : \|c\|_\infty = 1 \ \wedge \ \|c\|_1 = \kappa\right\}$ and the message space is $M' = \{0,1\}^l$. The random oracle is $\mathsf{H} : \{0,1\}^* \to \mathsf{Ch}$. The starting point is $i = 1$. Let $L_0 = \varnothing$ and $\sigma_0 = (\mathbf{0}, \mathbf{0})$. Setup is as in Algorithm 1.

$\mathsf{Gen}(\bar{\mathbf{A}})$

- $\mathbf{t} := \mathbf{0}$
- **while** $\mathbf{t}$ has no invertible coefficient **do**
  - $\mathbf{s} \xleftarrow{\$} S_\eta^{\ell+k}$
  - $\mathbf{t} := \bar{\mathbf{A}}\mathbf{s} \bmod q$
- $\mathsf{sk} := \mathbf{s}$
- $\mathsf{pk} := \mathbf{t}$
- **return** $(\mathsf{sk}, \mathsf{pk})$

$\mathsf{SeqSign}(\mathsf{sk}_i, m_i, L_{i-1}, \sigma_{i-1})$

- $(\tilde{\mathbf{u}}_{i-1}, \mathbf{z}_1, \ldots, \mathbf{z}_{i-1}) := \sigma_{i-1}$
- $\mathbf{s}_i := \mathsf{sk}_i$
- $\mathbf{t}_i := \bar{\mathbf{A}}\mathbf{s}_i \bmod q$
- $L_i := L_{i-1}||(\mathbf{t}_i, m_i)$
- $\mathbf{z}_i := \perp$
- **while** $\mathbf{z}_i := \perp$ **do**
  - $\mathbf{y}_i \leftarrow \mathscr{D}^{\ell+k}$
  - $\mathbf{u}_i := \bar{\mathbf{A}}\mathbf{y}_i \bmod q$
  - $\tilde{\mathbf{u}}_i := \tilde{\mathbf{u}}_{i-1} + \mathbf{u}_i \bmod q$
  - $c_i := \mathsf{H}(\tilde{\mathbf{u}}_i, L_i, \mathbf{z}_{i-1})$
  - $\mathbf{z}_i := c_i \cdot \mathbf{s}_i + \mathbf{y}_i$
  - $\mathbf{z}_i := \mathsf{RejSamp}(\mathbf{z}_i, c_i \cdot \mathbf{s}_i)$
- $\sigma_i := (\tilde{\mathbf{u}}_i, \mathbf{z}_1, \ldots, \mathbf{z}_i)$
- **return** σ_i

$\mathsf{SeqVerify}(L_N, \sigma_N)$

- $(\mathbf{t}_1, m_1)|| \ldots ||(\mathbf{t}_N, m_N) := L_N$
- $(\tilde{\mathbf{u}}_N, \mathbf{z}_1, \ldots, \mathbf{z}_N) := \sigma_N$
- $\mathbf{z}_0 := 1$
- **if** $\exists i$ such that $\mathbf{t}_i$ has no invertible element **then**
  - **return** 0
- **for** $i = N, \ldots, 1$ **do**
  - **if** $\|\mathbf{z}_i\|_2 > B$ **then**
    - **return** 0
  - $L_i := (\mathbf{t}_1, m_1)|| \ldots ||(\mathbf{t}_i, m_i)$
  - $c_i := \mathsf{H}(\tilde{\mathbf{u}}_i, L_i, \mathbf{z}_{i-1})$
  - $\mathbf{u}_i := \bar{\mathbf{A}}\mathbf{z}_i - c_i\mathbf{t}_i \bmod q$
  - $\tilde{\mathbf{u}}_{i-1} := \tilde{\mathbf{u}}_i - \mathbf{u}_i \bmod q$
- **if** $\tilde{\mathbf{u}}_1 = \mathbf{u}_1$ **then return** 1

Proof. We first sketch the high level ideas of the reduction $\mathcal{B}$. The complete description of $\mathcal{B}$ is found in Alg. 3. The random oracle and the signing oracle in the FH-UF-CMA game (resp. UF-CMA game) are denoted by H and $\mathsf{OSeqSign}$ (resp. H' and OSign). On receiving the public parameter $\mathbf{A}$ and the challenge public key $\mathbf{t}^*$, $\mathcal{B}$ checks that $\mathbf{t}^* \in R_q^k$ contains at least one invertible element. If so, $\mathcal{B}$ forwards $(\mathbf{A}, \mathbf{t}^*)$ to $\mathcal{A}$.

$\mathsf{OSeqSign}$ replies to queries by asking OSign for a signature on uniformly chosen m and programs H such that it outputs c returned by the outer random oracle H'. Here we cannot just forward m_i to OSign, because it might be that a forgery submitted by $\mathcal{A}$ later reuses the same m_i. Then submitting a forgery w.r.t. m_i is not valid in the UF-CMA game, causing $\mathcal{B}$ to lose.

In more detail, starting from the original FH-UF-CMA game, we construct several hybrid games towards the one used by the final reduction $\mathcal{B}$. We denote by $\Pr[\mathrm{G}_i(\mathcal{A})]$ the probability that $\mathrm{G}_i(\mathcal{A})$ halts with output 1.

G_0 This game is identical to the FH-UF-CMA game. At the beginning, the game initializes an empty key-value look-up table HT. Upon receiving a query to the random oracle H with input X, it returns $\mathrm{HT}[X]$ if the table entry is non-empty; otherwise, it samples uniform $c \in \mathsf{Ch}$, sets $\mathrm{HT}[X] := c$, and returns c. It holds that $\Pr[\mathrm{G}_0(\mathcal{A})] = \mathbf{Adv}^{\mathsf{FH\text{-}UF\text{-}CMA}}_{\mathsf{FSwA\text{-}SAS}}(\mathcal{A})$.

G_1 This game is identical to G_0, except that $\mathsf{OSeqSign}$ samples uniform $c_i \in \mathsf{Ch}$ instead of calling $c_i = \mathsf{H}(\tilde{\mathbf{u}}_i, L_i, \mathbf{z}_{i-1})$ after $\tilde{\mathbf{u}}_i$ is computed, and that it programs the RO table $\mathrm{HT}[\tilde{\mathbf{u}}_i, L_i, \mathbf{z}_{i-1}] := c_i$ as soon as the rejection sampling step succeeds; if $\mathrm{HT}[\tilde{\mathbf{u}}_i, L_i, \mathbf{z}_{i-1}]$ is already set, the game aborts by setting $\mathsf{bad}_{\mathsf{ucol}} = \mathsf{true}$. It holds that $|\Pr[\mathrm{G}_0(\mathcal{A})] - \Pr[\mathrm{G}_1(\mathcal{A})]| \leq \Pr[\mathsf{bad}_{\mathsf{ucol}}]$.

G_2 This game is identical to G_1, except that responses to random oracle queries $\mathsf{H}(\tilde{\mathbf{u}}_i, L_i, \mathbf{z}_{i-1})$ are simulated as follows. Initialize an empty key-value look-up table ZT. If $i = 1$ or there exists some $X := (\tilde{\mathbf{u}}_{i-1}, L_{i-1}, \mathbf{z}_{i-2})$ such that $\mathrm{ZT}[X] = \bar{\mathbf{A}}\mathbf{z}_{i-1} \bmod q$, then extract $\mathbf{u}_i := \tilde{\mathbf{u}}_i - \tilde{\mathbf{u}}_{i-1}$, sample uniform $c_i \in \mathsf{Ch}$, and set $\mathrm{ZT}[\tilde{\mathbf{u}}_i, L_i, \mathbf{z}_{i-1}] := \mathbf{u}_i + c_i\mathbf{t}_i$. If there already exists some entry $X' := (\tilde{\mathbf{u}}_{i-1}, L_{i-1}, \mathbf{z}_{i-2})$ such that $\mathrm{ZT}[X'] = \mathbf{u}_i + c_i\mathbf{t}_i$, the game aborts by setting $\mathsf{bad}_{\mathsf{zcol}} = \mathsf{true}$. It holds that $|\Pr[\mathrm{G}_1(\mathcal{A})] - \Pr[\mathrm{G}_2(\mathcal{A})]| \leq \Pr[\mathsf{bad}_{\mathsf{zcol}}]$.

G_3 This game is identical to G_2, except that $\mathsf{OSeqSign}$ and H proceed as follows. The game initializes an empty set $\mathcal{M}$ and key-value look-up table MT. Whenever $\mathsf{OSeqSign}$ receives a query, it internally samples a uniform message $m \in M$ and adds m to $\mathcal{M}$. Whenever H receives a query with input $(\tilde{\mathbf{u}}_i, L_i, \mathbf{z}_{i-1})$ and manages to extract $\mathbf{u}_i$ as above, it samples a uniform message $m \in M$ and aborts by setting $\mathsf{bad}_{\mathsf{mcol}} = \mathsf{true}$ if $m \in \mathcal{M}$. Else, it sets $\mathrm{MT}[\tilde{\mathbf{u}}_i, L_i, \mathbf{z}_{i-1}] = m$. It holds that $|\Pr[\mathrm{G}_2(\mathcal{A})] - \Pr[\mathrm{G}_3(\mathcal{A})]| \leq \Pr[\mathsf{bad}_{\mathsf{mcol}}]$.

G_4 This game is identical to G_3, except that it performs the following checks against the ZT entries after the adversary outputs a valid signature-history pair $(L_N, (\tilde{\mathbf{u}}_N, \mathbf{z}_1, \ldots, \mathbf{z}_N))$ as follows. Let $\tilde{\mathbf{u}}_{N-1}, \ldots, \tilde{\mathbf{u}}_1$ be as derived during the execution of $\mathsf{SeqVerify}$. If for some $i \in [N]$ the entry $\mathrm{ZT}[\tilde{\mathbf{u}}_i, L_i, \mathbf{z}_{i-1}]$ is undefined, the game halts by setting $\mathsf{bad}_{\mathsf{ord}} = \mathsf{true}$. It holds that $|\Pr[\mathrm{G}_3(\mathcal{A})] - \Pr[\mathrm{G}_4(\mathcal{A})]| \leq \Pr[\mathsf{bad}_{\mathsf{ord}}]$.

$\mathcal{B}$ Given an adversary $\mathcal{A}$ winning G_4, the reduction $\mathcal{B}$ described in Alg. 3 is obtained as follows. Upon receiving a query to $\mathsf{OSeqSign}$, $\mathcal{B}$ makes a query to OSign of the UF-CMA game with a uniform message $m \in M$, receives $\mathbf{u}_i$ and $\mathbf{z}_i$, and programs HT using challenge c_i output by the outer random oracle $\mathsf{H}'(\mathbf{u}_i, \mathbf{t}^*, m)$. Moreover, H obtains fresh challenge c_i for $\mathbf{t}_i = \mathbf{t}^*$ by querying the outer random oracle $\mathsf{H}'(\mathbf{u}_i, \mathbf{t}^*, m)$ if it succeeds in extracting $\mathbf{u}_i = \tilde{\mathbf{u}}_i - \tilde{\mathbf{u}}_{i-1}$. Since $\mathcal{A}$ is guaranteed to receive an invertible challenge public key in $\mathcal{B}$, the view of $\mathcal{A}$ is identical to that of G_4.

We now show that, as long as none of the bad events happen, $\mathcal{B}$ is guaranteed to output a message-signature pair $(m, (\mathbf{u}_{i^*}, \mathbf{z}_{i^*}))$ that gets accepted in the UF-CMA game, i.e., $\|\mathbf{z}_{i^*}\|_\infty \leq B$ and $\mathbf{u}_{i^*} = \bar{\mathbf{A}}\mathbf{z}_{i^*} - c\mathbf{t}^* \bmod q$ where $c = \mathsf{H}'(\mathbf{u}_{i^*}, \mathbf{t}^*, m)$. The former condition is immediate from the verification condition of $\mathsf{SeqVerify}$. To argue the latter, notice that we have $c = \mathsf{H}'(\mathbf{u}_{i^*}, \mathbf{t}^*, m) = \mathrm{HT}[\tilde{\mathbf{u}}_{i^*}, L_{i^*}, \mathbf{z}_{i^*-1}] = c_{i^*}$ as long as the RO entries $\mathrm{HT}[\tilde{\mathbf{u}}_1, L_1, \mathbf{z}_0], \ldots, \mathrm{HT}[\tilde{\mathbf{u}}_N, L_N, \mathbf{z}_{N-1}]$ have been set in the right order and thus $\mathbf{u}_{i^*} = \tilde{\mathbf{u}}_{i^*} - \tilde{\mathbf{u}}_{i^*-1}$ is extracted during the invocation of $\mathsf{H}(\tilde{\mathbf{u}}_{i^*}, L_{i^*}, \mathbf{z}_{i^*-1})$. The following lemma indeed assures that such queries have been made in the right order as long as $\mathsf{bad}_{\mathsf{zcol}} = \mathsf{bad}_{\mathsf{ord}} = \mathsf{false}$.

Lemma 3.4. *Let $\sigma_N = (\tilde{\mathbf{u}}_N, \mathbf{z}_1, \ldots, \mathbf{z}_N)$ and $L_N = (\mathbf{t}_1, m_1)||\ldots||(\mathbf{t}_N, m_N)$ a valid signature-history pair that $\mathcal{B}$ received from $\mathcal{A}$. Let $\tilde{\mathbf{u}}_1, \ldots, \tilde{\mathbf{u}}_{N-1}$ be as derived in* $\mathsf{SeqVerify}$ *run by $\mathcal{B}$. Suppose $\mathsf{bad}_{\mathsf{zcol}} = \mathsf{false}$. Then for $i \in [N-1]$, the random oracle entry $\mathrm{HT}[\tilde{\mathbf{u}}_{i+1}, L_{i+1}, \mathbf{z}_i]$ had been set after $\mathrm{HT}[\tilde{\mathbf{u}}_i, L_i, \mathbf{z}_{i-1}]$ was set if and only if $\mathsf{bad}_{\mathsf{ord}} = \mathsf{false}$.*

Proof. "Only if" We first argue that if the oracle entries $\mathrm{HT}[\tilde{\mathbf{u}}_1, L_1, \mathbf{z}_0], \ldots, \mathrm{HT}[\tilde{\mathbf{u}}_N, L_N, \mathbf{z}_{N-1}]$ have been set in this order and $\mathsf{bad}_{\mathsf{zcol}} = \mathsf{false}$, the corresponding ZT entries are all non-empty and thus $\mathsf{bad}_{\mathsf{ord}}$ must be false. Suppose this statement holds for $1 \le i \le j$, i.e., $\mathrm{ZT}[\tilde{\mathbf{u}}_1, L_1, \mathbf{z}_0], \ldots, \mathrm{ZT}[\tilde{\mathbf{u}}_j, L_j, \mathbf{z}_{j-1}]$ are non-empty. Due to the verification condition it must be that $\tilde{\mathbf{u}}_{j-1} = \tilde{\mathbf{u}}_j - \bar{\mathbf{A}}\mathbf{z}_j + c_j\mathbf{t}_j$ and thus $\bar{\mathbf{A}}\mathbf{z}_j = \tilde{\mathbf{u}}_j - \tilde{\mathbf{u}}_{j-1} + c_j + \mathbf{t}_j$. Because we assumed that $\mathrm{HT}[\tilde{\mathbf{u}}_{j-1}, L_{j-1}, \mathbf{z}_{j-2}]$ and $\mathrm{HT}[\tilde{\mathbf{u}}_j, L_j, \mathbf{z}_{j-1}]$ are set in this order, the invocation of $\mathsf{H}(\tilde{\mathbf{u}}_j, L_j, \mathbf{z}_{j-1})$ must have extracted $\mathbf{u}_j = \tilde{\mathbf{u}}_j - \tilde{\mathbf{u}}_{j-1}$ and have set $\mathrm{ZT}[\tilde{\mathbf{u}}_j, L_j, \mathbf{z}_{j-1}] = \mathbf{u}_j + c_j\mathbf{t}_j$. Note that, since $\mathsf{bad}_{\mathsf{zcol}} = \mathsf{false}$, there is no other entry in ZT that records the same value as $\mathbf{u}_j + c_j\mathbf{t}_j$. Thus, when $(\tilde{\mathbf{u}}_{j+1}, L_{j+1}, \mathbf{z}_j)$ is queried, H can uniquely find a tuple $(\tilde{\mathbf{u}}_j, L_j, \mathbf{z}_{j-1})$ such that $\mathrm{ZT}[\tilde{\mathbf{u}}_j, L_j, \mathbf{z}_{j-1}] = \bar{\mathbf{A}}\mathbf{z}_j$ and then set $\mathrm{ZT}[\tilde{\mathbf{u}}_{j+1}, L_{j+1}, \mathbf{z}_j] = \tilde{\mathbf{u}}_{j+1} - \tilde{\mathbf{u}}_j + c_{j+1}\mathbf{t}_{j+1}$. It is easy to see that the base case $j = 1$ is true: when $(\tilde{\mathbf{u}}_1, L_1, \mathbf{z}_0)$ is queried, the invocation of H always sets $\mathrm{ZT}[\tilde{\mathbf{u}}_1, L_1, \mathbf{z}_0] = \mathbf{u}_1 + c_1\mathbf{t}_1$ where $\mathbf{u}_1 = \tilde{\mathbf{u}}_1$.

"If" We give a proof by induction. As an induction hypothesis, we assume that for $i = 1, \ldots, j-1$ the random oracle entry $\mathrm{HT}[\tilde{\mathbf{u}}_{i+1}, L_{i+1}, \mathbf{z}_i]$ had been set after $\mathrm{HT}[\tilde{\mathbf{u}}_i, L_i, \mathbf{z}_{i-1}]$ was set whenever $\mathsf{bad}_{\mathsf{zcol}} = \mathsf{bad}_{\mathsf{ord}} = \mathsf{false}$. Now suppose, for a contradiction, that $\mathrm{HT}[\tilde{\mathbf{u}}_{j+1}, L_{j+1}, \mathbf{z}_j]$ was set *before* $\mathrm{HT}[\tilde{\mathbf{u}}_j, L_j, \mathbf{z}_{j-1}]$ while $\mathsf{bad}_{\mathsf{zcol}} = \mathsf{bad}_{\mathsf{ord}} = \mathsf{false}$. Because $\mathsf{bad}_{\mathsf{ord}} = \mathsf{false}$, the entry $\mathrm{ZT}[\tilde{\mathbf{u}}_i, L_i, \mathbf{z}_{i-1}]$ is non-empty for all $i \in [N]$. When H is queried with input $(\tilde{\mathbf{u}}_{j+1}, L_{j+1}, \mathbf{z}_j)$, since the corresponding entry in ZT is non-empty, it must be that there exists some $X' := (\tilde{\mathbf{u}}'_j, L_j, \mathbf{z}'_{j-1}) \neq (\tilde{\mathbf{u}}_j, L_j, \mathbf{z}_{j-1})$ such that $c'_j := \mathrm{HT}[X']$ and $\mathrm{ZT}[X'] = \bar{\mathbf{A}}\mathbf{z}_j$ are already set. This implies that X' has been queried to H before and that $\bar{\mathbf{A}}\mathbf{z}_j = \mathbf{u}'_j + c'_j\mathbf{t}_j \bmod q$, where $\mathbf{u}'_j$ is some value extracted inside $\mathsf{H}(X')$. On the other hand, due to the verification condition it also holds that $\bar{\mathbf{A}}\mathbf{z}_j = \mathbf{u}_j + c_j\mathbf{t}_j \bmod q$, where $c_j = \mathrm{HT}[\tilde{\mathbf{u}}_j, L_j, \mathbf{z}_{j-1}]$ and $\mathbf{u}_j = \tilde{\mathbf{u}}_j - \tilde{\mathbf{u}}_{j-1}$. Here, $\mathbf{u}_j$ is the value extracted when $(\mathbf{u}_j, L_j, \mathbf{z}_{j-1})$ is queried to H for the first time, because due to the induction hypothesis $\mathrm{HT}[\tilde{\mathbf{u}}_{j-1}, L_{j-1}, \mathbf{z}_{j-2}]$ had been already set at this point. However, this implies that $\mathsf{bad}_{\mathsf{zcol}}$ is set when $(\tilde{\mathbf{u}}_j, L_j, \mathbf{z}_{j-1})$ is queried to H, contradicting the assumption that $\mathsf{bad}_{\mathsf{zcol}} = \mathsf{false}$.

Let us prove the base case $j = 2$ in a similar manner. Suppose $\mathrm{HT}[\tilde{\mathbf{u}}_2, L_2, \mathbf{z}_1]$ was set *before* $\mathrm{HT}[\tilde{\mathbf{u}}_1, L_1, \mathbf{z}_0]$ while $\mathsf{bad}_{\mathsf{zcol}} = \mathsf{bad}_{\mathsf{ord}} = \mathsf{false}$. When H is queried with input $(\tilde{\mathbf{u}}_2, L_2, \mathbf{z}_1)$, since the corresponding entry in ZT is non-empty, it must be that there exists some $X' := (\tilde{\mathbf{u}}'_1, L_1, \mathbf{z}_0) \neq (\tilde{\mathbf{u}}_1, L_1, \mathbf{z}_0)$ such that $c'_1 := \mathrm{HT}[X']$ and $\mathrm{ZT}[X'] = \bar{\mathbf{A}}\mathbf{z}_1$ are already set. This implies that X' has been queried to H before and that $\bar{\mathbf{A}}\mathbf{z}_1 = \tilde{\mathbf{u}}'_1 + c'_1\mathbf{t}_1 \bmod q$. On the other hand, due to the verification condition it also holds that $\bar{\mathbf{A}}\mathbf{z}_1 = \tilde{\mathbf{u}}_1 + c_1\mathbf{t}_1 \bmod q$, where $c_1 = \mathrm{HT}[\tilde{\mathbf{u}}_1, L_1, \mathbf{z}_0]$. However, this implies that $\mathsf{bad}_{\mathsf{zcol}}$ is set when $(\tilde{\mathbf{u}}_1, L_1, \mathbf{z}_0)$ is queried to H, contradicting the assumption that $\mathsf{bad}_{\mathsf{zcol}} = \mathsf{false}$. $\square$

All in all, unless $\mathcal{B}$ sets $\mathsf{bad}_{\mathsf{inv}} = \mathsf{true}$, $\mathcal{B}$ wins the $\mathsf{UF\text{-}CMA}$ game if and only if G_4 outputs 1. In other words,

$$\begin{aligned}\mathbf{Adv}^{\mathsf{UF\text{-}CMA}}_{\mathsf{FSwA\text{-}S}}(\mathcal{B}) &= (1 - \Pr[\mathsf{bad}_{\mathsf{inv}}]) \cdot \Pr[\mathrm{G}_4] \\ &\ge p_{\mathsf{inv}} \cdot \left(\mathbf{Adv}^{\mathsf{FH\text{-}UF\text{-}CMA}}_{\mathsf{FSwA\text{-}SAS}}(\mathcal{A}) - \Pr[\mathsf{bad}_{\mathsf{ucol}}] - \Pr[\mathsf{bad}_{\mathsf{zcol}}] - \Pr[\mathsf{bad}_{\mathsf{mcol}}] - \Pr[\mathsf{bad}_{\mathsf{ord}}]\right).\end{aligned}$$

The running time of $\mathcal{B}$ is at most the running time of $\mathcal{A}$ plus the time it takes for running verification operations and handling random oracle queries. The former takes $O(Nk\ell t_{\mathsf{pmul}})$ because each iteration of the for-loop involves matrix-vector multiplication $\bar{\mathbf{A}}\mathbf{z}_i$ (ignoring the run-time for polynomial addition as it's much smaller than multiplication). The latter takes $O(Q_h k\ell t_{\mathsf{pmul}})$ because $\mathcal{B}$ carries out matrix-vector multiplication $\bar{\mathbf{A}}\mathbf{z}_{i-1}$ for each query to the RO H.

In the following, we provide a concrete bound for each bad event.

Bounding $\Pr[\mathsf{bad}_{\mathsf{mcol}}]$ Assuming that the adversary makes at most Q_s queries to $\mathsf{OSeqSign}$, there are at most Q_s distinct values in $\mathcal{M}$ and Q_h distinct values in MT, respectively. The $\mathsf{bad}_{\mathsf{mcol}}$ flag is potentially set due to two different causes: (1) H internally samples m that is already recorded in $\mathcal{M}$ and thus the corresponding entry is not stored in MT, or (2) $\mathsf{OSeqSign}$ internally samples m that is already recorded in MT and thus the corresponding entry gets removed. The probability that a randomly sampled m inside H collides with one of the values in $\mathcal{M}$ is at most $Q_s/|M| = Q_s/2^l$. Since at most Q_h queries to H are made by $\mathcal{A}$, the probability that case (1) occurs during such queries is at most $Q_s \cdot Q_h/2^l$. If for some $i \in I$ a tuple $(\tilde{\mathbf{u}}_i, L_i, \mathbf{z}_{i-1})$ is queried to H during the invocation of $\mathsf{SeqVerify}$ for the first time, in order to cause $\mathsf{bad}_{\mathsf{mcol}} = \mathsf{true}$, *for all such* i independently sampled m must be in $\mathcal{M}$. Thus, the probability that case (1) occurs during such queries is at most $Q_s/2^l$. The probability that a randomly sample m inside $\mathsf{OSeqSign}$ collides with one of the values in MT is at most $Q_h/2^l$. Since at most Q_s queries to $\mathsf{OSeqSign}$ are made, the probability that case (2) occurs is at most $Q_s \cdot Q_h/2^l$. Overall, we get $\Pr[\mathsf{bad}_{\mathsf{mcol}}] \le Q_s \cdot (2Q_h + 1)/2^l$.

Bounding $\Pr[\mathsf{bad}_{\mathsf{ucol}}]$ Since there are at most $Q_h + Q_s$ values in HT and $\mathbf{u}$ is generated by the signing algorithm Sign from $\mathsf{FSwA\text{-}S}$, for each query to $\mathsf{OSeqSign}$ the probability that the flag $\mathsf{bad}_{\mathsf{ucol}}$ is set is at most

$$\max_{\mathbf{u}} \Pr\left[\mathbf{u} = \bar{\mathbf{A}}\mathbf{y} \bmod q \;:\; \mathbf{y} \xleftarrow{\$} \mathscr{D}^{\ell+k}\right]. \tag{2}$$

Algorithm 3: Reduction to UF-CMA security of FSwA-S

The random oracle in the UF-CMA game is denoted by H'. The sign oracle in the UF-CMA game is denoted by OSign. Let $\tilde{\mathbf{u}}_0 = 0$. Without loss of generality, $\mathcal{A}$ queries H with input public keys $\mathbf{t}_1, \ldots, \mathbf{t}_i$ all of which contain at least one invertible element, because otherwise such keys will be rejected by the verification algorithm anyway. All flags are initially set to false.

$\mathcal{B}^{\mathsf{OSign},\mathsf{H}'}(\bar{\mathbf{A}}, \mathbf{t}^*)$

- $\mathcal{Q} := \varnothing$; $\mathcal{M} := \varnothing$
- **if** $\mathbf{t}^*$ has no invertible element **then**
  - $\mathsf{bad}_{\mathsf{inv}} := \mathsf{true}$
- $(\sigma_N, L_N) \leftarrow \mathcal{A}^{\mathsf{OSeqSign},\mathsf{H}}(\bar{\mathbf{A}}, \mathbf{t}^*)$
- $(\mathbf{t}_1, m_1)||\ldots||(\mathbf{t}_N, m_N) := L_N$
- $(\tilde{\mathbf{u}}_N, \mathbf{z}_1 \ldots, \mathbf{z}_N) := \sigma_N$
- $I := \{i \in [N] \,:\, \mathbf{t}_i = \mathbf{t}^* \wedge (m_i, L_{i-1}) \notin \mathcal{Q}\}$
- **if** $\mathsf{SeqVerify}(\sigma_N, L_N) = 1 \wedge |I| \neq 0$ **then**
  - Derive $\tilde{\mathbf{u}}_{N-1}, \ldots, \tilde{\mathbf{u}}_1$ as in $\mathsf{SeqVerify}$
  - **if** $\exists i \in [N]$ such that $\mathrm{ZT}[\tilde{\mathbf{u}}_i, L_i, \mathbf{z}_{i-1}] = \perp$ **then**
    - $\mathsf{bad}_{\mathsf{ord}} := \mathsf{true}$
  - **if** $\exists i^* \in I$ such that $m := \mathrm{MT}[\tilde{\mathbf{u}}_{i^*}, L_{i^*}, \mathbf{z}_{i^*-1}] \neq \perp$ **then**
    - $\mathbf{u}_{i^*} := \tilde{\mathbf{u}}_{i^*} - \tilde{\mathbf{u}}_{i^*-1}$
    - **return** $(m, \mathbf{u}_{i^*}, \mathbf{z}_{i^*})$
  - **else**
    - $\mathsf{bad}_{\mathsf{mcol}} := \mathsf{true}$

$\mathsf{OSeqSign}(m_i, L_{i-1}, \sigma_{i-1})$

- $\mathcal{Q} := \mathcal{Q} \cup \{(m_i, L_{i-1})\}$
- $(\tilde{\mathbf{u}}_{i-1}, \mathbf{z}_1, \ldots, \mathbf{z}_{i-1}) := \sigma_{i-1}$
- $m \xleftarrow{\$} M$
- **if** $\exists X$ such that $\mathrm{MT}[X] = m$ **then**
  - $\mathrm{MT}[X] := \perp$
- $\mathcal{M} := \mathcal{M} \cup \{m\}$
- $(\mathbf{u}, \mathbf{z}) \leftarrow \mathsf{OSign}(m)$
- $c := \mathsf{H}'(\mathbf{u}, m, \mathbf{t}^*)$
- $L_i := L_{i-1}||(\mathbf{t}^*, m_i)$
- $\tilde{\mathbf{u}}_i := \tilde{\mathbf{u}}_{i-1} + \mathbf{u} \bmod q$
- $\mathbf{z}_i := \mathbf{z}$
- **if** $\mathrm{HT}[\tilde{\mathbf{u}}_i, L_i, \mathbf{z}_{i-1}] \neq \perp$ **then**
  - $\mathsf{bad}_{\mathsf{ucol}} := \mathsf{true}$
- **else**
  - $\mathrm{HT}[\tilde{\mathbf{u}}_i, L_i, \mathbf{z}_{i-1}] := c$
- $\sigma_i := (\tilde{\mathbf{u}}_i, \mathbf{z}_1, \ldots, \mathbf{z}_i)$
- **return** σ_i

$\mathsf{H}(\tilde{\mathbf{u}}_i, L_i, \mathbf{z}_{i-1})$

- **if** $\mathrm{HT}[\tilde{\mathbf{u}}_i, L_i, \mathbf{z}_{i-1}] \neq \perp$ **then**
  - **return** $\mathrm{HT}[\tilde{\mathbf{u}}_i, L_i, \mathbf{z}_{i-1}]$
- $(\mathbf{t}_1, m_1)||\ldots||(\mathbf{t}_i, m_i) := L_i$
- **if** $i = 1$ or $\exists X := (\tilde{\mathbf{u}}_{i-1}, L_{i-1}, \mathbf{z}_{i-2})$ such that $\mathrm{HT}[X] \neq \perp \wedge \mathrm{ZT}[X] = \bar{\mathbf{A}}\mathbf{z}_{i-1} \bmod q$ **then**
  - $\mathbf{u}_i := \tilde{\mathbf{u}}_i - \tilde{\mathbf{u}}_{i-1} \bmod q$
  - **if** $\mathbf{t}_i = \mathbf{t}^*$ **then**
    - $m \xleftarrow{\$} M$
    - $c_i := \mathsf{H}'(\mathbf{u}_i, m, \mathbf{t}^*)$
    - **if** $m \notin \mathcal{M}$ **then**
      - $\mathrm{MT}[\tilde{\mathbf{u}}_i, L_i, \mathbf{z}_{i-1}] := m$
  - **else**
    - $c_i \xleftarrow{\$} \mathsf{Ch}$
  - **if** $\exists X' := (\tilde{\mathbf{u}}'_i, L_i, \mathbf{z}'_{i-1})$ such that $\mathrm{ZT}[X'] = \mathbf{u}_i + c_i\mathbf{t}_i \bmod q$ **then**
    - $\mathsf{bad}_{\mathsf{zcol}} := \mathsf{true}$
  - $\mathrm{ZT}[\tilde{\mathbf{u}}_i, L_i, \mathbf{z}_{i-1}] := \mathbf{u}_i + c_i\mathbf{t}_i \bmod q$
- **else**
  - $c_i \xleftarrow{\$} \mathsf{Ch}$
- $\mathrm{HT}[\tilde{\mathbf{u}}_i, L_i, \mathbf{z}_{i-1}] := c_i$
- **return** c_i

Equation 2 can be upper bounded by Lemma 2.2, using the probability preservation property of the Rényi divergence. It yields

$$\max_{\mathbf{u}} \Pr\left[\mathbf{u} = \bar{\mathbf{A}}\mathbf{y} \bmod q \ : \ \mathbf{y} \xleftarrow{\$} \mathscr{D}^{\ell+k}\right]$$
$$\leq \sqrt{\max_{\mathbf{u}} \Pr\left[\mathbf{u} = \mathbf{v} \bmod q \ : \ \mathbf{v} \xleftarrow{\$} R_q^k\right]} \cdot \mathrm{RD}_2((\bar{\mathbf{A}}, \bar{\mathbf{A}}\mathbf{y}) \| (\bar{\mathbf{A}}, \mathbf{v})).$$

The probability in the second inequality is given by $\sqrt{1/|R_q^k|} = q^{-nk/2}$. By Lem. 2.2, using the data processing inequality, it holds $\mathrm{RD}_2((\bar{\mathbf{A}}, \bar{\mathbf{A}}\mathbf{y}) \| (\bar{\mathbf{A}}, \mathbf{v})) \leq \mathrm{RD}_2((\mathbf{A}, \mathbf{A}\mathbf{y}') \| (\mathbf{A}, \mathbf{v}))$, where $\mathbf{y}' \xleftarrow{\$} \mathscr{D}^{\ell}$. By Lem. 2.3, the latter is bounded above by a constant if Eq. 1 is fulfilled. Since $\mathsf{OSeqSign}$ receives at most Q_s queries, overall, we obtain $\Pr[\mathsf{bad}_{\mathsf{ucol}}] \leq O(Q_s(Q_h + Q_s)/q^{nk/2})$.

Bounding $\Pr[\mathsf{bad}_{\mathsf{zcol}}]$ Fix an existing entry in ZT of the form $\tilde{\mathbf{u}}_i' + c_i'\mathbf{t}_i \bmod q$. Then the probability that $\mathbf{u}_i + c_i\mathbf{t}_i \bmod q$ hits such an entry is

$$\Pr_{c_i \xleftarrow{\$} \mathsf{Ch}} \left[\mathbf{u}_i + c_i\mathbf{t}_i = \tilde{\mathbf{u}}_i' + c_i'\mathbf{t}_i \bmod q\right] = \Pr_{c_i \xleftarrow{\$} \mathsf{Ch}} \left[c_i\mathbf{t}_i = \tilde{\mathbf{u}}_i' + c_i'\mathbf{t}_i - \mathbf{u}_i \bmod q\right].$$

Since at least one coefficient of $\mathbf{t}_i$ is invertible, the above probability is bounded by $1/|\mathsf{Ch}|$. Let Q_i be the number of entries in HT indexed by a tuple containing a history L of size i. Then we have $Q_h + Q_s = \sum_{i=1}^{N} Q_i$. Because H receives at most $Q_i + 1$ queries for each (where "+1" comes from the fact that an additional query is made from inside $\mathsf{SeqVerify}$), by the union bound, we have that $\Pr[\mathsf{bad}_{\mathsf{zcol}}] \leq \sum_{i=1}^{N} Q_i(Q_i + 1)/(2|\mathsf{Ch}|)$.

Bounding $\Pr[\mathsf{bad}_{\mathsf{ord}}]$ Due to Lemma 3.4, "$\mathsf{bad}_{\mathsf{ord}} = \mathsf{true}$ while $\mathsf{bad}_{\mathsf{zcol}} = \mathsf{false}$" implies that there exists some $i \in [N-1]$, such that the entry $\mathrm{HT}[\tilde{\mathbf{u}}_{i+1}, L_{i+1}, \mathbf{z}_i]$ was set before $\mathrm{HT}[\tilde{\mathbf{u}}_i, L_i, \mathbf{z}_{i-1}]$, where $(\tilde{\mathbf{u}}_N, (\mathbf{z}_1, \ldots, \mathbf{z}_N))$ and L_N are signature-history pair output by $\mathcal{A}$ at the end of the game. We argue that this event occurs with negligible probability if the verification condition is satisfied. The event potentially occurs in two ways: (1) for some $i \in [N-1]$, $\mathrm{HT}[\tilde{\mathbf{u}}_i, L_i, \mathbf{z}_{i-1}]$ is set for the first time *during the invocation of* $\mathsf{SeqVerify}$, and (2) for some $i \in [N-1]$, $\mathrm{HT}[\tilde{\mathbf{u}}_i, L_i, \mathbf{z}_{i-1}]$ was set for the first time *before the invocation of* $\mathsf{SeqVerify}$*, but after* $\mathrm{HT}[\tilde{\mathbf{u}}_{i+1}, L_{i+1}, \mathbf{z}_i]$ *was set.*

To bound case (1), it is sufficient to prove the following statement inductively: if none of $\mathrm{HT}[\tilde{\mathbf{u}}_1, L_1, \mathbf{z}_0], \ldots, \mathrm{HT}[\tilde{\mathbf{u}}_{i-1}, L_{i-1}$ have been set for the first time during the invocation of $\mathsf{SeqVerify}$, the probability that $\mathrm{HT}[\tilde{\mathbf{u}}_i, L_i, \mathbf{z}_{i-1}]$ is set for the first time while running $\mathsf{SeqVerify}$ *and* that the verification condition is satisfied, is at most $Q_{i-1}/|\mathsf{Ch}|$, where Q_i's are defined as above and let $Q_0 = 1$ for convenience. Since $\tilde{\mathbf{u}}_{i-1}, \tilde{\mathbf{u}}_i, \mathbf{z}_i, \mathbf{t}_i$ have been already fixed at the moment when $(\tilde{\mathbf{u}}_i, L_i, \mathbf{z}_{i-1})$ is queried to H inside $\mathsf{SeqVerify}$, the probability that the signature gets accepted is

$$\Pr_{c_i \xleftarrow{\$} \mathsf{Ch}} \left[\tilde{\mathbf{u}}_{i-1} = \tilde{\mathbf{u}}_i - (\bar{\mathbf{A}}\mathbf{z}_i - c_i\mathbf{t}_i) \bmod q\right],$$

which is at most $1/|\mathsf{Ch}|$. Because there are at most Q_{i-1} entries HT indexed by a tuple containing L_i and thus at most Q_{i-1} different values for $\tilde{\mathbf{u}}_{i-1}$ exist, by the union bound, the probability that case (1) happens is at most $Q_{i-1}/|\mathsf{Ch}|$. The base case is true: if $i = 1$, since $\tilde{\mathbf{u}}_1, \mathbf{z}_1, \mathbf{t}_1$ have been already fixed at the moment when $(\tilde{\mathbf{u}}_1, L_1, \mathbf{z}_0)$ is queried to H inside $\mathsf{SeqVerify}$, the probability that the signature gets accepted is

$$\Pr_{c_1 \xleftarrow{\$} \mathsf{Ch}} \left[\tilde{\mathbf{u}}_1 = \bar{\mathbf{A}}\mathbf{z}_1 - c_1\mathbf{t}_1 \bmod q\right],$$

which is at most $1/|\mathsf{Ch}|$.

To bound case (2), it is sufficient to prove the following statement inductively: if $\mathrm{HT}[\tilde{\mathbf{u}}_1, L_1, \mathbf{z}_0], \ldots, \mathrm{HT}[\tilde{\mathbf{u}}_i, L_i, \mathbf{z}_{i-1}]$ have been set in this order before the invocation of $\mathsf{SeqVerify}$, the probability that $\mathrm{HT}[\tilde{\mathbf{u}}_{i+1}, L_{i+1}, \mathbf{z}_i]$ was set before $\mathrm{HT}[\tilde{\mathbf{u}}_i, L_i, \mathbf{z}_{i-1}]$ *and* that the verification condition is satisfied, is at most $Q_i Q_{i+1}/|\mathsf{Ch}|$, where Q_i's are defined as above. By the definition of the verification procedure, it holds that $\tilde{\mathbf{u}}_{i-1} = \tilde{\mathbf{u}}_i - (\bar{\mathbf{A}}\mathbf{z}_i - c_i\mathbf{t}_i) \bmod q$. However, because both $\mathrm{HT}[\tilde{\mathbf{u}}_{i-1}, L_{i-1}, \mathbf{z}_{i-2}]$ and $\mathrm{HT}[\tilde{\mathbf{u}}_{i+1}, L_{i+1}, \mathbf{z}_i]$ have been already set *before* $c_i = \mathrm{HT}[\tilde{\mathbf{u}}_i, L_i, \mathbf{z}_{i-1}]$ is sampled, for fixed $\tilde{\mathbf{u}}_i, \tilde{\mathbf{u}}_{i-1}, \mathbf{z}_i, \mathbf{t}_i, \bar{\mathbf{A}}$, the probability that fresh c_i meets the verification condition is

$$\Pr_{c_i \xleftarrow{\$} \mathsf{Ch}} \left[\tilde{\mathbf{u}}_{i-1} = \tilde{\mathbf{u}}_i - (\bar{\mathbf{A}}\mathbf{z}_i - c_i\mathbf{t}_i) \bmod q\right],$$

which is at most $1/|\mathsf{Ch}|$. Because there are at most Q_{i+1} different existing entries of $\mathrm{HT}[\tilde{\mathbf{u}}_{i+1}, L_{i+1}, \mathbf{z}_i]$ and fresh c_i is sampled at most Q_i times, by the union bound, we obtain the overall upper bound $Q_i Q_{i+1}/|\mathsf{Ch}|$. The base case $i = 1$ is clearly true: if the tuple $(\tilde{\mathbf{u}}_1, L_1, \mathbf{z}_0)$ is queried after $\mathrm{HT}[\tilde{\mathbf{u}}_2, L_2, \mathbf{z}_1]$ has been set, for the verification condition to be met fresh c_1 must satisfy $c_1\mathbf{t}_1 = \tilde{\mathbf{u}}_1 - \bar{\mathbf{A}}\mathbf{z}_1 \bmod q$ for fixed $\tilde{\mathbf{u}}_1, \mathbf{z}_1, \mathbf{t}_1$. The overall probability is thus bounded by $Q_1 Q_2/|\mathsf{Ch}|$ using the same argument as above. All in all, we have that $\Pr[\mathsf{bad}_{\mathsf{ord}}] \leq (1 + \sum_{i=1}^{N-1}(Q_i + Q_i Q_{i+1}))/|\mathsf{Ch}|$. Note that

$$\begin{aligned}\Pr[\mathsf{bad}_{\mathsf{zcol}}] + \Pr[\mathsf{bad}_{\mathsf{ord}}] &\leq \sum_{i=1}^{N} Q_i(Q_i+1)/(2|\mathsf{Ch}|) + \left(1 + \sum_{i=1}^{N-1}(Q_i + Q_i Q_{i+1})\right)/|\mathsf{Ch}| \\ &< \left(\sum_{i=1}^{N} Q_i + 1\right)^2/|\mathsf{Ch}| = (Q_h + Q_s + 1)^2/|\mathsf{Ch}|.\end{aligned}$$

Putting all the bounds above together, we obtain the concrete bound in the theorem statement. □

4 Performance Estimates and Comparison

4.1 Performance Estimates

In the following, we provide some concrete sample parameters and performance estimates for the FSwA-SAS from Section 3. We provide a formula for the compression rate τ and a lower bound for N, from which on our SAS signature σ_N is smaller than the trivial solution of concatenating N independent single signatures σ_{con}. The compression rate is defined as $\tau(N) = \frac{\mathsf{len}(\sigma_N)}{\mathsf{len}(\sigma_{\mathsf{con}})}$, where $\mathsf{len}(\cdot)$ denotes the bit size of an element.

A FSwA-SAS signature after N steps is given by $\sigma_N = (\tilde{\mathbf{u}}_N, \mathbf{z}_1, \ldots, \mathbf{z}_N)$ and the concatenation of N single FSwA-S signatures by $\sigma_{\mathsf{con}} = (c_1, \ldots, c_N, \mathbf{z}_1, \ldots, \mathbf{z}_N)$. Here, we have applied the standard trick to shorten FSwA-S signatures by replacing the commitment $\mathbf{u}$ by the challenge c. Thus, its compression rate is

$$\tau(N) = \frac{\mathsf{len}(\mathbf{u}) + N \cdot \mathsf{len}(\mathbf{z})}{N \cdot \mathsf{len}(c) + N \cdot \mathsf{len}(\mathbf{z})} = \frac{kn\lceil\log_2 q\rceil + N(k+\ell)n\lceil\log_2 B\rceil}{Nn + N(k+\ell)n\lceil\log_2 B\rceil} \tag{3}$$

$$= 1 - \frac{1}{1 + (k+\ell)\lceil\log_2 B\rceil} + \frac{k\lceil\log_2 q\rceil}{N(k+\ell)\lceil\log_2 B\rceil}, \tag{4}$$

where $\mathbf{u} \in R_q^k$, $\mathbf{z} \in S_B^{\ell+k}$ and $c \in \mathsf{Ch} := \left\{c \in R : \|c\|_\infty = 1 \ \wedge\ \|c\|_1 = \kappa\right\}$. An element $c \in \mathsf{Ch}$ can be represented by n bits [LDK+20, Sec. 5.3].

The SAS signature starts to be smaller than the concatenation as soon as $\mathsf{len}(\mathbf{u}) < N \cdot \mathsf{len}(c)$, hence, the tipping point is $N_0 > \frac{kn\lceil\log_2 q\rceil}{n} = k\lceil\log_2 q\rceil$.

In Table 1, we provide the number N_0 and some τ values for the different security parameters of Dilithium. We can clearly see that in Equation 4 the compression rate asymptotically goes towards $1 - 1/(1+(k+\ell)\lceil\log_2 B\rceil)$ and for example for the Level 2 parameters of Dilithium this is exactly the rate 0.9927 that we observe at $N = 1{,}000{,}000$.

Table 1: Tipping point N_0 and some τ values of our FSwA-SAS (Alg. 2) for the three different parameter sets of Dilithium.

Parameter	Level 2	Level 3	Level 5
q	8380417	8380417	8380417
n	256	256	256
(k, ℓ)	$(4, 4)$	$(6, 5)$	$(8, 7)$
$B = \gamma - \kappa \cdot \eta$	130994	524092	524168
N_0	92	138	184
$\tau(200)$	0.9961	0.9985	0.9997
$\tau(250)$	0.9954	0.9979	0.9991
$\tau(500)$	0.9941	0.9966	0.9978
$\tau(1000)$	0.9934	0.9959	0.9971
$\tau(1{,}000{,}000)$	0.9927	0.9952	0.9965

Unlike other proposals to aggregate lattice-based signatures (either interactive [DOTT21, BTT22] or non-interactive [BR21]), the modulus q doesn't need to be increased in our construction. This is due to the fact that we aggregate over the $\mathbf{u}$-parts of the signature (which are uniform modulo q), and not over the $\mathbf{z}$-parts (which are small and hence the size of their sum increases).

4.2 Comparison With SAS Using Trapdoors

In this section, we compare our lattice-based SAS scheme with existing proposals of lattice-based SAS schemes [EB14, WW19]. As summarized in the introduction, they can be seen as sequential aggregate versions of GPV-signatures. In the following, we take Falcon as a concrete instantiation for such a signature.

As for the $\mathsf{FSwA\text{-}S}$ signature, the size of a single GPV-signature can be significantly reduced by applying a small trick. More precisely, a Falcon signature of a message m is defined as $\sigma = (s_1, s_2, r)$, where $(s_1, s_2) \in R \times R$ is a pair of short polynomials such that $s_1 h + s_2 = \mathsf{H}(m, r)$, where H is a random oracle, r is some randomness salt and $h \in R_q$ defines the public basis of the underlying NTRU lattice. Here, R is again the ring $\mathbb{Z}[X]/\langle X^n + 1\rangle$ for n a power of 2 and q some prime integer. As s_2 is determined by m and s_1 (given the public key h and the salt r), one can omit s_2 in the signature and only set $\sigma = (s_1, r)$. Intuitively, this (roughly) halves the signature size.

Unfortunately, this trick can't be used in the (sequential) aggregate signature setting. Thus, when assessing the compactness of an aggregate signature, one has to compare it with the trivial concatenation of all single signatures, where each is only composed of the second polynomial. This fair comparison has been done in [EB14], but not in [WW19].

Recently, Espitau et al. [ETWY22] used exactly this trick to make Falcon signatures even shorter. By using elliptical instead of spherical Gaussians, the norm of s_1 can be made smaller. At the same time, the norm of s_2 gets larger, accordingly. Again, this trick does not apply to (sequential) aggregate signatures, as the total size of (s_1, s_2) stays the same.

In the existing SAS schemes that aggregate GPV-style signatures, the main bottleneck is that in the lattice setting there are no known trapdoor permutations. To circumvent this, they replace the trapdoor permutations from the RSA setting by so-called preimage sampleable trapdoor functions [GPV08]. However, those functions have different domain Do and range Ra spaces. In the case of Falcon, the domain is given by R_q, i.e., any element $x \in \mathsf{Ra}$ is of bit length $\mathsf{len}(x) = n\lceil\log_2 q\rceil$. The range, however, is given by pairs of polynomials of degree less than n with coefficients that come from a discrete Gaussian distribution. Naively, one could apply the Gaussian tail bound to argue that the coefficient's absolute values are bounded by some parameter β, and hence any element $y \in \mathsf{Do}$ can be represented by a bit string of length $\mathsf{len}(y) = 2n\lceil\log_2 \beta\rceil$.[4] The specifications of Falcon [PFH+20, Sec. 3.11.2] propose a more intelligent representation of elements in the domain by using the Huffman encoding. Note that in both cases it yields $\mathsf{len}(\mathsf{Ra}) > \mathsf{len}(\mathsf{Do})$. As the output of one preimage sampleable function serves as the input for the next preimage sampleable function (of the same domain as before), existing constructions [EB14] pack as many bits of the so-far signature as they can into a vector that serves as the new input. The remaining bits ($b := \mathsf{len}(\mathsf{Ra}) - \mathsf{len}(\mathsf{Do})$) are stored in some vector α and appended (at every step) to the so-far signature and appear at the end in the final sequential aggregate signature. Clearly, they can't achieve a constant-size aggregate signature.

For concreteness, take the sample parameters of $\mathsf{Falcon\text{-}512}$, i.e., $q = 12289$ and $n = 512$. It yields $\mathsf{len}(\mathsf{Do}) = n\lceil\log_2 q\rceil = 7168$ and $\mathsf{len}(\mathsf{Ra}) = 2 \cdot 5000$, and hence $b = 2832$.[5] The final sequential aggregate signature after N steps is given by $\sigma_N = (s_{N,1}, s_{N,2}, \alpha_1, \ldots, \alpha_{N-1}, r_1, \ldots, r_N)$, where $(s_{N,1}, s_{N,2}) \in \mathsf{Ra}$, $\mathsf{len}(\alpha_i) = b$ and $\mathsf{len}(r_i) = 328$ for $i \in [N]$. On the other side, the concatenation of N single Falcon signatures is given by $\sigma_{\mathsf{con}} = (s_{1,1}, \ldots, s_{N,1}, r_1, \ldots, r_N)$, where we applied the 'omit the second polynomial' trick. Thus, its compression rate is given by

$$\begin{aligned}\tau(N) &= \frac{\mathsf{len}(s_1) + \mathsf{len}(s_2) + (N-1)\cdot\mathsf{len}(\alpha) + N\cdot\mathsf{len}(r)}{N\cdot\mathsf{len}(s_1) + N\cdot\mathsf{len}(r)} \\ &= \frac{2\cdot 5000 + (N-1)b + N\cdot 328}{N\cdot 5000 + N\cdot 328} \\ &= 1 - \frac{5000 - b}{5000 + 328} + \frac{10000 - b}{N(5000 + 328)},\end{aligned}$$

where $(s_1, s_2) \in \mathsf{Ra}$, α is the carry-over information and r the salt.

[4] This analysis has been done by [EB14].

[5] We compute $\mathsf{len}(\mathsf{Ra})$ as $2 \cdot (8 \cdot \mathsf{sbytelen} - 328)$ with $\mathsf{sbytelen}$ taken from [PFH+20, Table 3.3].

We provide the number N_0 and some τ values for the two different security parameters of Falcon in Table 2. From the equations above, we can clearly see that the compression rate asymptotically goes towards $1 - (5000 - b)/5328$, which is exactly the rate 0.5931 that we observe at $N = 1,000,000$.

In the following, we explain how the recent results of Espitau et al. [ETWY22] extremely leverage the benefit of GPV-style SAS. Overall, they significantly reduce the size of the trivial concatenation by replacing spherical Gaussians by elliptical Gaussians. The main idea is that there are now two different lengths, $\mathsf{len}(s_1)$ and $\mathsf{len}(s_2)$, where the first holds for s_1 and the latter for s_2 for every pair $(s_1, s_2) \in \mathsf{Ra}$. Whereas before both s_1 and s_2 followed a Gaussian distribution of width σ, they now introduce a distortion factor γ and set $\sigma_1 = \sigma/\gamma$ and $\sigma_2 = \sigma\gamma$. One can see that the total size of (s_1, s_2) is preserved as it yields $2\log_2 \sigma = \log_2 \sigma_1 + \log_2 \sigma_2$. If one takes $\gamma = 8$ (as suggested by Espitau et al. [ETWY22, Table 1]), one can see that $\mathsf{len}(s_1) = 2952$, by again using the formulas of the Falcon specifications.

Table 2: Tipping point N_0 and some τ values for SAS based on Falcon-512, for spherical and elliptical Gaussians (distortion factor $\gamma = 8$).

Parameter	Falcon-512 (spherical)	Falcon-512 (elliptical)
q	12289	12289
n	512	512
$\mathsf{len}(\mathsf{Do})$	7168	7168
$\mathsf{len}(s_1)$	5000	2952
$\mathsf{len}(s_2)$	5000	7048
b	2832	2832
$\mathsf{len}(r)$	328	328
N_0	4	60
$\tau(150)$	0.6021	0.9780
$\tau(200)$	0.5998	0.9743
$\tau(250)$	0.5985	0.9722
$\tau(500)$	0.5958	0.9678
$\tau(1000)$	0.5944	0.9656
$\tau(1,000,000)$	0.5931	0.9634

The compression rate in the elliptical Gaussian case is

$$\begin{aligned}
\tau(N) &= \frac{\mathsf{len}(s_1) + \mathsf{len}(s_2) + (N-1)\cdot\mathsf{len}(\alpha) + N\cdot\mathsf{len}(r)}{N\cdot\mathsf{len}(s_1) + N\cdot\mathsf{len}(r)} \\
&= \frac{2\cdot 5000 + (N-1)b + N\cdot 328}{N\cdot 2952 + N\cdot 328} \\
&= 1 - \frac{2952 - b}{2952 + 328} + \frac{10000 - b}{N(2952+328)},
\end{aligned}$$

where $(s_1, s_2) \in \mathsf{Ra}$, α is the carry-over information and r the salt.

Here, we can clearly see that the compression rate asymptotically goes towards $1-(2952-b)/(2952+328)$, which is exactly the rate 0.9634 that we observe at $N = 1,000,000$.

5 Attacks on Existing Schemes

5.1 Attack on [WW19]

In the following, we identify an insecurity of the history-free sequential aggregate signature from Wang and Wu [WW19], published in the proceedings of the PROVSEC conference from 2019. More precisely, Lemma 5.1 gives an attack that breaks its security in the history-free setting, whose definition we recall in Section ??. From a high level perspective, the signing procedure is not randomized (enough) to prevent standard attacks in the history-free setting, which were already pointed out by Brogle et al. [BGR12, App. A] and Gentry et al. [GOR18, Sec. 4.3].

Recall that their construction focuses on lattice signatures that follow the GPV-paradigm. For simplicity, we adapt in the rest of the section the syntax of Falcon, as in Section 5 of [WW19].

Let us (again) briefly recap how Falcon works. As before, we are working over the ring $R_q = \mathbb{Z}_q[X]/\langle X^n + 1\rangle$ for some power-of-two integer n and some prime modulus q. The key generation algorithm

Algorithm 4: History-Free SAS' [WW19]

The message space is $M = \{0,1\}^l$. The two random oracles are $\mathsf{H}_1, \mathsf{H}_2 \colon \{0,1\}^* \to \{0,1\}^\lambda$ and the ideal cipher is $\pi \colon \{0,1\}^* \times R_q \to R_q$ with inverse $\pi^{-1} \colon \{0,1\}^* \times R_q \to R_q$. Let $\sigma_0 = ((0,0),0)$.

$\mathsf{Gen}(1^\lambda)$

1: $(h, T_h) \leftarrow \mathsf{TrapGen}(1^\lambda)$
2: $\mathsf{pk} := h$
3: $\mathsf{sk} := (h, T_h)$
4: **return** $(\mathsf{pk}, \mathsf{sk})$

$\mathsf{SeqSign}(\mathsf{sk}_i, m_i, \sigma_{i-1})$

1: $(x_{i-1}, \alpha_{i-1}) := \sigma_{i-1}$
2: $(h_i, T_{h_i}) := \mathsf{sk}_i$
3: $K_i := h_i || \mathsf{H}_1(m_i) || \mathsf{H}_2(\alpha_{i-1})$
4: $(s_{i-1}, s'_{i-1}) := x_{i-1}$
5: $\alpha_i := \alpha_{i-1} || s'_{i-1}$
6: $y_i := \pi^{-1}(K_i, s_{i-1}) \in R_q$
7: $x_i \leftarrow \mathsf{SamplePre}(T_{h_i}, y_i) \in \mathsf{Do}$
8: **return** $\sigma_i := (x_i, \alpha_i)$

$\mathsf{SeqVerify}(L_N, \sigma_N)$

1: $\{(h_1, m1), \ldots, (h_N, m_N)\} := L_N$
2: $(x_N, \alpha_N) := \sigma_N$
3: $(s'_1, \ldots, s'_N) := \alpha_N$
4: **for** $i = N, \ldots, 1$ **do**
5: $\quad \alpha_i = (s'_1, \ldots, s'_i)$
6: $\quad K_i = h_i || \mathsf{H}_1(m_i) || \mathsf{H}_2(\alpha_i)$
7: $\quad s_{i-1} = \pi(K_i, f_{h_i}(x_i))$
8: $\quad x_{i-1} = (s_{i-1}, s'_{i-1})$
9: $\quad$ **if** $x_{i-1} \notin \mathsf{Do}$ **then**
10: $\quad\quad$ **return** 0
11: **if** $x_0 = (0,0)$ **then**
12: $\quad$ **return** 1
13: **else**
14: $\quad$ **return** 0

invokes a function $\mathsf{TrapGen}$ which outputs a ring element $h \in R_q$[6], together with an associated trapdoor T_h. This trapdoor is needed to invert the function $f_h \colon \mathsf{Do} \subset R_q \times R_q \to R_q = \mathsf{Ra}$, where $f_h(s, s') = hs + s'$, with the help of a pre-image sampleable function $\mathsf{SamplePre}(T_h, \cdot)$. Without specifying the domain Do precisely, we remark that it only contains pairs of *short* ring elements. The trapdoor defines the secret key, whereas the element h defines the public key. In order to sign a message m, a random oracle $\mathsf{H} \colon \{0,1\}^* \to R_q$ is invoked on m which outputs a ring element in R_q. Then, the function $\mathsf{SamplePre}$ is used to compute $(s, s') \in \mathsf{Do}$ such that $f_h(s, s') = hs + s' = \mathsf{H}(m)$. The signature is defined as $x = (s, s')$. In order to verify a signature $x = (s, s')$ for a message m, one simply checks if $(s, s') \in \mathsf{Do}$ and if the equation $hs + s' = \mathsf{H}(m)$ holds in R_q.

The main idea of the history-free sequential aggregate signature SAS' by Wang and Wu [WW19] is to adapt the framework for trapdoor-permutation-based sequential aggregate signatures by Gentry et al. [GOR18] to the lattice setting. As in [GOR18], the scheme is making use of an ideal cipher. Additionally, and in contrast to [GOR18], the scheme in [WW19] also uses two random oracles. As the domain $\mathsf{Do} \subset R_q \times R_q$ is larger than the range $\mathsf{Ra} = R_q$, we don't have trapdoor permutations in the case of lattice signatures, but only pre-image sampleable functions [GPV08]. This is why a so-far signature σ_{i-1} has to be split into a first part (denoted by x_{i-1}) that contains the output of a previous call on $\mathsf{SamplePre}$, and a second part (denoted by α_{i-1}) which stores the information of the previous signatures that didn't fit into the sequential signing process. This part grows linearly in the number of signed messages.

We summarize (a simplified version of) the SAS' scheme of [WW19] in Algorithm 4[7] and present the attack in Lemma 5.1. The key idea of the attack is that an adversary can *predict* the one-time key K_i for a message m_i and public key h_i.

Lemma 5.1. *The history-free* SAS' *described in Algorithm 4 is not* $\mathsf{HF\text{-}UF\text{-}CMA}$ *secure.*

Proof. Let $\mathcal{A}$ be a PPT adversary. Their goal is to generate an aggregate signature σ^* for a list L^* claiming that signer i signed message m_i (where the public key h_i of signer i is the challenge public key pk given to $\mathcal{A}$) without having queried the signing oracle OSign on input m_i.

1 The attacker computes $\sigma_{i-1} = (x_{i-1}, \alpha_{i-1})$ for arbitrary and self-chosen key pairs and messages, defining L_i. Let $(s_{i-1}, s'_{i-1}) := x_{i-1}$.

2 They choose some $m_i \neq \widetilde{m_i}$ and let $K_i := h_i || \mathsf{H}_1(m_i) || \mathsf{H}_2(\alpha_{i-1})$ and $\widetilde{K_i} := h_i || \mathsf{H}_1(\widetilde{m_i}) || \mathsf{H}_2(\alpha_{i-1})$.

---

[6] The ring element $h \in R_q$ is computationally close to uniform assuming the hardness of decision NTRU.

[7] We slightly modified their scheme by adding Step 9-10 in the $\mathsf{SeqVerify}$ algorithm in order to check if the so-far signature x_{i-1} recovered at each step lies indeed in the domain D. Else, it would be very easy for the adversary to come up with forgeries.

3 They compute $\widetilde{s_{i-1}} := \pi(\widetilde{K_i}, \pi^{-1}(K_i, s_{i-1}))$.

4 Let $\widetilde{x_{i-1}} := (\widetilde{s_{i-1}}, s'_{i-1})$. They query OSign with input $\widetilde{\sigma_{i-1}} := (\widetilde{x_{i-1}}, \alpha_{i-1})$ and $\widetilde{m_i}$.

The oracle responds with $\widetilde{\sigma_i} = (x_i, \alpha_i)$, such that (1) $\alpha_i = \alpha_{i-1}||s'_{i-1}$ and (2) $\pi(\widetilde{K_i}, f_{h_i}(x_i)) = \widetilde{s_{i-1}}$. The adversary outputs $\sigma^* := \widetilde{\sigma_i}$ and $L^* := L_i \cup \{(h_i, m_i)\}$. Recall from Step 3 of the attack that $\pi^{-1}(\widetilde{K_i}, \widetilde{s_{i-1}}) = \pi^{-1}(K_i, s_{i-1})$. Thus, this is a valid forgery as $\pi(K_i, f_{h_i}(x_i)) = s_{i-1}$. $\square$

Remark 5.2. An easy fix against this attack, at the expense of larger sequential aggregate signatures, is to make the key for the ideal cipher *unpredictable* for the adversary. One possible strategy is to freshly sample a truly random string $r \xleftarrow{\$} \{0,1\}^\lambda$ and append it to the public key h and the message m to obtain the ideal cipher key $K := h||m||r$. Besides, this makes the use of the random oracles H_1 and H_2 superfluous. However, the randomness r has to be carried over through out the sequential signing process, increasing the size of the final signature by $\lambda \cdot N$ bits, where N is the number of involved signatures. This strategy has already been formalized in the second construction of Gentry and al. [GOR18, Sec. 4.2]. They also propose a deterministic solution by using tag-based trapdoor permutations [GOR18, Sec. 4.3]. It would be interesting to study whether this approach can be adapted to pre-image sampleable functions.

5.2 Attack on [FH20]

In this section, we describe how to mount a (partial) secret-key recovery attack against the Dilithium-based multi-signature by Fukumitsu and Hasegawa [FH20], published in the proceedings of the PROVSEC conference from 2020. In order to be successful, the adversary only needs one valid and honestly generated signatures. The attack exploits the fact that every party of the multi-signature (including the adversary) obtains the full information of the first signature part $\mathbf{u}$, which enables them to compute $c \cdot \mathbf{s}_2$, where $(\mathbf{s}_1, \mathbf{s}_2)$ is the secret key corresponding to the provided challenge public key. As it is easier to see the vulnerability in the single signature setting, we first describe an insecure variant of FSwA-S (specified in Algorithm 5) and then show how the vulnerability is carried over to the multi-signature from [FH20].

The main difference between the original (and secure) FSwA-S (Algorithm 1) and the modified (and insecure) version (Algorithm 5) is that we apply a trick due to Bai and Galbraith [BG14], which enables to compress the size of the signature. This technique is also used in Dilithium. The key idea is to compute $\mathbf{u} = \mathbf{A}\mathbf{y}$ (instead of $\mathbf{u} = \mathbf{A}\mathbf{y}_1 + \mathbf{y}_2$ as before) and to only use the high order bits $\mathsf{HighBits}(\mathbf{u})$ of $\mathbf{u}$ to derive the challenge c. Subsequently, in the verification algorithm only the high order bits of $\mathbf{u}$ and $\mathbf{A}\mathbf{z} - c\mathbf{t}$ are compared to each other. This modification reduces the dimension of $\mathbf{z}$ from $\ell + k$ to ℓ, where $\mathbf{A} \in R_q^{k \times \ell}$.

In contrast to Dilithium and [BG14], our version contains the full commitment $\mathbf{u}$, and not the challenge c. Transmitting the challenge instead of the commitment is a well-known technique in the lattice setting to further shrink the size of the signature. As we see in the following, it is also crucial for security once the trick of by Bai and Galbraith [BG14] has been applied.

Lemma 5.3. *Let* $\mathbf{A} \leftarrow \mathsf{Setup}(1^\lambda)$ *and* $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Gen}(\mathbf{A})$ *with* $\mathsf{sk} = (\mathbf{s}_1, \mathbf{s}_2)$ *as in Alg. 5. Given* $(\mathbf{u}, \mathbf{z}) = \sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, m)$ *for the messages* $m \in M$ *together with the public key* $\mathsf{pk} = \mathbf{t}$*, a* PPT *adversary* $\mathcal{A}$ *can recover* $\mathbf{s}_2$*.*

Proof. The adversary re-constructs $c = \mathsf{H}(\mathsf{HighBits}(\mathbf{u}), \mathbf{t}, m)$ and computes

$$\mathbf{b} = \mathbf{A}\mathbf{z} - \mathbf{u} - c\mathbf{t} = \mathbf{A}\mathbf{y} + c\mathbf{A}\mathbf{s}_1 - \mathbf{A}\mathbf{y} - c\mathbf{A}\mathbf{s}_1 - c\mathbf{s}_2 = -c\mathbf{s}_2.$$

Note that the last equation gives $\mathbf{b} = -c\mathbf{s}_2 \bmod q$, but as there is no wrapping around modulo q (as both c and $\mathbf{s}_2$ are short elements) the equation also holds in R. The adversary now embeds the elements in the field $K = \mathbb{Q}[x]/(x^d + 1)$ and uses the fact that every non-zero element is invertible in K. Note that $0 \notin \mathsf{Ch}$. Hence, they multiply the result by c^{-1} (the K-inverse of c) and recover $-\mathbf{s}_2$. $\square$

We can now easily move to the multi-signature in [FH20]. We don't re-state the full protocol of their scheme here, but simply refer to Figure 5 in [FH20]. For simplicity, we use our notations in the following and ignore the optimization to reduce the public key size in [FH20, Fig. 5]. The main issue is that every signer broadcasts untruncated $\mathbf{u}$ in the clear during the interactive signing process. We now explain how an adversary can during the common signing procedure recover half of the secret key of the single honest signer.

Algorithm 5: Insecure FSwA-S

The challenge space is $\mathsf{Ch} := \{c \in R : \|c\|_\infty = 1 \wedge \|c\|_1 = \kappa\}$ and the message space is $M = \{0,1\}^l$. The random oracle is $\mathsf{H} : \{0,1\}^* \to \mathsf{Ch}$. Let $\mathscr{D}, B$ and $\mathsf{RejSamp}$ be as in Sec. 2.3.

$\mathsf{Setup}(1^\lambda)$

$\mathbf{A} \xleftarrow{\$} R_q^{k\times\ell}$
return $\mathbf{A}$

$\mathsf{Gen}(\mathbf{A})$

$(\mathbf{s}_1, \mathbf{s}_2) \xleftarrow{\$} S_\eta^\ell \times S_\eta^k$
$\mathbf{t} := \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2 \bmod q$
$\mathsf{sk} := (\mathbf{s}_1, \mathbf{s}_2)$
$\mathsf{pk} := \mathbf{t}$
return $(\mathsf{sk}, \mathsf{pk})$

$\mathsf{Ver}(\mathsf{pk}, \sigma, m)$

$(\mathbf{u}, \mathbf{z}) := \sigma$
$\mathbf{t} := \mathsf{pk}$
$c := \mathsf{H}(\mathsf{HighBits}(\mathbf{u}), \mathbf{t}, m)$
if $\|\mathbf{z}\|_\infty \leq B \wedge \mathsf{HighBits}(\mathbf{Az} - c \cdot \mathbf{t}) = \mathsf{HighBits}(\mathbf{u})$ **then**
    return 1
else
    return 0

$\mathsf{Sign}(\mathsf{sk}, m)$

$(\mathbf{s}_1, \mathbf{s}_2) := \mathsf{sk}$
$\mathbf{t} := \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2 \bmod q$
$\mathbf{z} := \perp$
while $\mathbf{z} := \perp$ **do**
    $\mathbf{y} \xleftarrow{\$} \mathscr{D}^\ell$
    $\mathbf{u} := \mathbf{Ay} \bmod q$
    $c := \mathsf{H}(\mathsf{HighBits}(\mathbf{u}), \mathbf{t}, m)$
    $\mathbf{z} := c \cdot \mathbf{s}_1 + \mathbf{y}$
    $\mathbf{z} := \mathsf{RejSamp}(\mathbf{z}, c \cdot \mathbf{s})$
$\sigma := (\mathbf{u}, \mathbf{z})$
return σ

Lemma 5.4. *Let* $\mathsf{MS} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$ *be the multi-signature as defined in [FH20, Fig. 5]. Further, let* $\mathcal{A}$ *be a* PPT *adversary who is controlling all-but-one parties. We denote by* $(\mathsf{pk}^*, \mathsf{sk}^*)$ *the key pair of the honest signer, where* $\mathsf{sk}^* = (\mathbf{s}_1^*, \mathbf{s}_2^*)$*. After one successful multi-signature signing process,* $\mathcal{A}$ *can recover* $\mathbf{s}_2^*$*.*

Proof. Without loss of generality, let party 1 be the honest signer and party 2 to party N be the ones controlled by $\mathcal{A}$. Let $\mathbf{t}_i$ denote the public key of party $i \in [N]$. During the signing process, every party computes $\mathbf{u}_i := \mathbf{A}\mathbf{y}_i$ (denoted by $\mathbf{w}_v$ in the original protocol). At the second stage, they broadcasts $\mathbf{u}_i$ to all the co-signers. In particular, $\mathcal{A}$ receives $\mathbf{u}_1$ and computes $\mathbf{u} := \sum_{i\in[N]} \mathbf{u}_i$ (denoted by $\mathbf{w}$ in the original protocol). The multi-signature contains $\mathbf{z} := \sum_{i\in[N]} \mathbf{z}_i$, where $\mathbf{z}_i = \mathbf{y}_i + c_i\mathbf{s}_{i,1}$. The adversary re-constructs all challenges $c_i = \mathsf{H}(\mathsf{HighBits}(\mathbf{u}), \mathbf{t}_i, m)$ and computes

$$\mathbf{b} = \mathbf{Az} - \mathbf{u} - \sum_{i\in[N]} c_i\mathbf{t}_i = \sum_{i\in[N]} \mathbf{A}\mathbf{y}_i + c_i\mathbf{A}\mathbf{s}_{i,1} - \mathbf{A}\mathbf{y}_i - c_i\mathbf{A}\mathbf{s}_{i,1} - c_i\mathbf{s}_{i,2} = -\sum_{i\in[N]} c_i\mathbf{s}_{i,2}.$$

As they know $\mathbf{s}_{i,2}$ for all $1 < i \leq N$, they can compute $\mathbf{b} + \sum_{i=2}^N c_i\mathbf{s}_{i,2}$ and recover $c_1\mathbf{s}_{1,2}$. With the same reasoning as in the attack above, they can easily recover $\mathbf{s}_{1,2} = \mathbf{s}_2^*$. ◻

Remark 5.5. In the simple signature setting (Algorithm 5) it is easy to fix this attack by only outputting $\sigma := (\mathsf{HighBits}(\mathbf{u}), \mathbf{z})$. However, in the multi-signature setting, this fix doesn't seem to apply in a trivial manner. The problem is that the function $\mathsf{HighBits}$ is not linear, and in general $\mathsf{HighBits}(\mathbf{u}_1) + \mathsf{HighBits}(\mathbf{u}_2) \neq \mathsf{HighBits}(\mathbf{u}_1 + \mathbf{u}_2)$.

6 Conclusion

In this paper, we proposed a sequential aggregate signature based on the FSwA framework and showed that it can indeed save bandwidth compared to the naive concatenation. It exploits the fact that aggregation of the u-part is much more critical than the z-part in the lattice-based FSwA setting. Admittedly, the benefit of our construction appears still limited in practice according to the concrete parameter analysis of Section 4. It would make an interesting future direction to design a new aggregation paradigm tailored to FSwA exploiting small c_i's in the non-aggregated part.

Another natural follow-up question would be how to adapt our scheme to half-aggregate Dilithium signatures incorporating all the bit-truncation optimizations. In terms of correctness, we observe no obvious issue because our scheme can be securely modified by aggregating the u-part with XOR instead of summation $\mod q$ (as already observed by [CZ22] in the Schnorr setting), and thus the fact that HighBits destroys homomorphism is not a major obstacle, unlike the issue with [FH20] we pointed out in Section 5.2. However, one must carefully adapt the probability that bad events happen in the reduction, and thus we leave detailed analysis for future work.

We also observe that the situation with Falcon-based SAS is not satisfactory either given that the recent trick of [ETWY22] significantly saves the size of naive concatenation and one of the existing instantiations turned out to be insecure. Since aggregation of hash-then-signatures essentially amounts to batch-proving the knowledge of short preimages of the function $f_h : (s, s') \mapsto hs + s'$ and requires no proof of correct hash evaluation, we conjecture that generic methods proposed by [DGKV22, ACL+22] will likely lead to concretely efficient instantiations in this setting.

Acknowledgment

This research was partly funded by the Danish Independent Research Council under project number 0165-00107B (C3PO). We thank our anonymous referees for their thorough proof reading and constructive feedback.

References

ACL+22. M. R. Albrecht, V. Cini, R. W. F. Lai, G. Malavolta, and S. A. K. Thyagarajan. Lattice-based snarks: Publicly verifiable, preprocessing, and recursively composable. *IACR Cryptol. ePrint Arch.*, p. 941, 2022. 3, 20

AGH10. J. H. Ahn, M. Green, and S. Hohenberger. Synchronized aggregate signatures: new definitions, constructions and applications. In *ACM CCS 2010*, pp. 473–484. ACM Press, 2010. 4

BG14. S. Bai and S. D. Galbraith. An improved compression technique for signatures based on learning with errors. In *CT-RSA 2014*, vol. 8366 of *LNCS*, pp. 28–47. Springer, Heidelberg, 2014. 4, 18

BGLS03. D. Boneh, C. Gentry, B. Lynn, and H. Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In *EUROCRYPT 2003*, vol. 2656 of *LNCS*, pp. 416–432. Springer, Heidelberg, 2003. 3

BGOY07. A. Boldyreva, C. Gentry, A. O'Neill, and D. H. Yum. Ordered multisignatures and identity-based sequential aggregate signatures, with applications to secure routing. In *ACM CCS 2007*, pp. 276–285. ACM Press, 2007. 3

BGR12. K. Brogle, S. Goldberg, and L. Reyzin. Sequential aggregate signatures with lazy verification from trapdoor permutations - (extended abstract). In *ASIACRYPT 2012*, vol. 7658 of *LNCS*, pp. 644–662. Springer, Heidelberg, 2012. 3, 16

BJRW20. K. Boudgoust, C. Jeudy, A. Roux-Langlois, and W. Wen. Towards classical hardness of module-LWE: The linear rank case. In *ASIACRYPT 2020, Part II*, vol. 12492 of *LNCS*, pp. 289–317. Springer, Heidelberg, 2020. 6

BJRW22. K. Boudgoust, C. Jeudy, A. Roux-Langlois, and W. Wen. On the hardness of module learning with errors with short distributions. *IACR Cryptol. ePrint Arch.*, p. 472, 2022. 5

BK20. D. Boneh and S. Kim. One-time and interactive aggregate signatures from lattices. *preprint*, 2020. 3

BLR+18. S. Bai, T. Lepoint, A. Roux-Langlois, A. Sakzad, D. Stehlé, and R. Steinfeld. Improved security proofs in lattice-based cryptography: Using the Rényi divergence rather than the statistical distance. *Journal of Cryptology*, 31(2):610–640, 2018. 5

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. 3, 4

BNN07. M. Bellare, C. Namprempre, and G. Neven. Unrestricted aggregate signatures. In *ICALP 2007*, vol. 4596 of *LNCS*, pp. 411–422. Springer, Heidelberg, 2007. 3

Bol03. A. Boldyreva. Threshold signatures, multisignatures and blind signatures based on the gap-Diffie-Hellman-group signature scheme. In *PKC 2003*, vol. 2567 of *LNCS*, pp. 31–46. Springer, Heidelberg, 2003. 5

BR21. K. Boudgoust and A. Roux-Langlois. Compressed linear aggregate signatures based on module lattices. *IACR Cryptol. ePrint Arch.*, p. 263, 2021. 4, 15

BTT22. C. Boschini, A. Takahashi, and M. Tibouchi. Musig-l: Lattice-based multi-signature with single-round online phase. *IACR Cryptol. ePrint Arch.*, p. 1036, 2022. 5, 15

CGKN21. K. Chalkias, F. Garillot, Y. Kondi, and V. Nikolaenko. Non-interactive half-aggregation of EdDSA and variants of Schnorr signatures. In *CT-RSA 2021*, vol. 12704 of *LNCS*, pp. 577–608. Springer, Heidelberg, 2021. 3, 4

Cra96. R. Cramer. *Modular Design of Secure yet Practical Cryptographic Protocols.* PhD thesis, CWI, Amsterdam, 1996. `https://ir.cwi.nl/pub/21438`. 3

CZ22. Y. Chen and Y. Zhao. Half-aggregation of schnorr signatures with tight reductions. *IACR Cryptol. ePrint Arch.*, p. 222, 2022. 3, 4, 7, 20

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. 3

DGKV22. L. Devadas, R. Goyal, Y. Kalai, and V. Vaikuntanathan. Rate-1 non-interactive arguments for batch-np and applications. Cryptology ePrint Archive, Paper 2022/1236, 2022. `https://eprint.iacr.org/2022/1236`. 3, 20

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`. 4

DOTT21. I. Damgård, C. Orlandi, A. Takahashi, and M. Tibouchi. Two-round n-out-of-n and multi-signatures and trapdoor commitment from lattices. In *Public-Key Cryptography - PKC 2021 - 24th IACR International Conference on Practice and Theory of Public Key Cryptography, Virtual Event, May 10-13, 2021, Proceedings, Part I*, vol. 12710 of *Lecture Notes in Computer Science*, pp. 99–130. Springer, 2021. 5, 15

EB14. R. El Bansarkhani and J. Buchmann. Towards lattice based aggregate signatures. In *AFRICACRYPT 14*, vol. 8469 of *LNCS*, pp. 336–355. Springer, Heidelberg, 2014. 4, 15

ETWY22. T. Espitau, M. Tibouchi, A. Wallet, and Y. Yu. Shorter hash-and-sign lattice-based signatures. *IACR Cryptol. ePrint Arch.*, p. 785, 2022. 4, 15, 16, 20

FH20. M. Fukumitsu and S. Hasegawa. A lattice-based provably secure multisignature scheme in quantum random oracle model. In *ProvSec 2020*, vol. 12505 of *LNCS*, pp. 45–64. Springer, Heidelberg, 2020. 2, 4, 18, 19, 20

FLS12. M. Fischlin, A. Lehmann, and D. Schröder. History-free sequential aggregate signatures. In *SCN 12*, vol. 7485 of *LNCS*, pp. 113–130. Springer, Heidelberg, 2012. 3

FS87. A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO'86*, vol. 263 of *LNCS*, pp. 186–194. Springer, Heidelberg, 1987. 3

FSZ22. N. Fleischhacker, M. Simkin, and Z. Zhang. Squirrel: Efficient synchronized multi-signatures from lattices. *IACR Cryptol. ePrint Arch.*, p. 694, 2022. 5

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. 6

GOR18. C. Gentry, A. O'Neill, and L. Reyzin. A unified framework for trapdoor-permutation-based sequential aggregate signatures. In *PKC 2018, Part II*, vol. 10770 of *LNCS*, pp. 34–57. Springer, Heidelberg, 2018. 3, 6, 16, 17, 18

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. 3, 15, 17

GR06. C. Gentry and Z. Ramzan. Identity-based aggregate signatures. In *PKC 2006*, vol. 3958 of *LNCS*, pp. 257–273. Springer, Heidelberg, 2006. 4

HKW15. S. Hohenberger, V. Koppula, and B. Waters. Universal signature aggregators. In *EUROCRYPT 2015, Part II*, vol. 9057 of *LNCS*, pp. 3–34. Springer, Heidelberg, 2015. 3

Kas22. Y. Kondi and abhi shelat. Improved straight-line extraction in the random oracle model with applications to signature aggregation. Cryptology ePrint Archive, Paper 2022/393, 2022. `https://eprint.iacr.org/2022/393`. 3

LDK+20. V. Lyubashevsky, L. Ducas, E. Kiltz, T. Lepoint, P. Schwabe, G. Seiler, D. Stehlé, and S. Bai. CRYSTALS-DILITHIUM. Technical report, National Institute of Standards and Technology, 2020. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions`. 3, 5, 14

LMRS04. A. Lysyanskaya, S. Micali, L. Reyzin, and H. Shacham. Sequential aggregate signatures from trapdoor permutations. In *EUROCRYPT 2004*, vol. 3027 of *LNCS*, pp. 74–90. Springer, Heidelberg, 2004. 3, 6

LOS+06. S. Lu, R. Ostrovsky, A. Sahai, H. Shacham, and B. Waters. Sequential aggregate signatures and multisignatures without random oracles. In *EUROCRYPT 2006*, vol. 4004 of *LNCS*, pp. 465–485. Springer, Heidelberg, 2006. 3

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. 5, 8

LS15. A. Langlois and D. Stehlé. Worst-case to average-case reductions for module lattices. *Des. Codes Cryptogr.*, 75(3):565–599, 2015. 6

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. 3, 6

Lyu12. V. Lyubashevsky. Lattice signatures without trapdoors. In *EUROCRYPT 2012*, vol. 7237 of *LNCS*, pp. 738–755. Springer, Heidelberg, 2012. 3, 6

MOR01. S. Micali, K. Ohta, and L. Reyzin. Accountable-subgroup multisignatures: Extended abstract. In *ACM CCS 2001*, pp. 245–254. ACM Press, 2001. 5

Nev08. G. Neven. Efficient sequential aggregate signed data. In *EUROCRYPT 2008*, vol. 4965 of *LNCS*, pp. 52–69. Springer, Heidelberg, 2008. 3

NRS21. J. Nick, T. Ruffing, and Y. Seurin. Musig2: Simple two-round schnorr multi-signatures. In *Advances in Cryptology - CRYPTO 2021 - 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16-20, 2021, Proceedings, Part I*, vol. 12825 of *Lecture Notes in Computer Science*, pp. 189–221. Springer, 2021. 3

PFH+20. T. Prest, P.-A. Fouque, J. Hoffstein, P. Kirchner, V. Lyubashevsky, T. Pornin, T. Ricosset, G. Seiler, W. Whyte, and Z. Zhang. FALCON. Technical report, National Institute of Standards and Technology, 2020. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions. 3, 15

Sch91. C.-P. Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174, 1991. 3

vEH14. T. van Erven and P. Harremoës. Rényi divergence and kullback-leibler divergence. *IEEE Trans. Inf. Theory*, 60(7):3797–3820, 2014. 5

WW19. Z. Wang and Q. Wu. A practical lattice-based sequential aggregate signature. In *ProvSec 2019*, vol. 11821 of *LNCS*, pp. 94–109. Springer, Heidelberg, 2019. 2, 4, 15, 16, 17

WW22. B. Waters and D. J. Wu. Batch arguments for np and more from standard bilinear group assumptions. Cryptology ePrint Archive, Paper 2022/336, 2022. https://eprint.iacr.org/2022/336. 3