Fast Authentication from Aggregate Signatures with Improved Security

Muslum Ozgur Ozmen[1], Rouzbeh Behnia[1], and Attila A. Yavuz[2]

[1] Oregon State University {ozmenmu,behniar}@oregonstate.edu
[2] University of South Florida attilaayavuz@usf.edu

Abstract. An attempt to derive signer-efficient digital signatures from aggregate signatures was made in a signature scheme referred to as Structure-free Compact Rapid Authentication (SCRA) (IEEE TIFS 2017). In this paper, we first mount a practical universal forgery attack against the NTRU instantiation of SCRA by observing only 8161 signatures. Second, we propose a new signature scheme (`FAAS`), which transforms any single-signer aggregate signature scheme into a signer-efficient scheme. We show two efficient instantiations of `FAAS`, namely, `FAAS-NTRU` and `FAAS-RSA`, both of which achieve high computational efficiency. Our experiments confirmed that `FAAS` schemes achieve up to 100× faster signature generation compared to their underlying schemes. Moreover, `FAAS` schemes eliminate some of the costly operations such as Gaussian sampling, rejection sampling, and exponentiation at the signature generation that are shown to be susceptible to side-channel attacks. This enables `FAAS` schemes to enhance the security and efficiency of their underlying schemes. Finally, we prove that `FAAS` schemes are secure (in random oracle model), and open-source both our attack and `FAAS` implementations for public testing purposes.

Keywords: Authentication, Digital signatures, Universal forgery, NTRU-based signatures

1 Introduction

Efficient authentication is critical for applications that need to generate a large throughput of authenticated data in a short amount of time. For instance, in smart grids [23,41], vehicular [1,24] and commercial drone networks [37,42], a large number of messages should be authenticated and transmitted to ensure reliable service and safe operation. While conventional digital signatures (e.g., RSA [35], ECDSA [5]) are deemed as an ideal mean to provide authentication, they might not offer the computational efficiency required by such applications. It is essential to propose fast digital signature schemes that can meet with the stringent requirements of such applications.

Achieving this computational efficiency becomes even harder when considering security in the post-quantum era [2,28]. While efficient and easy-to-implement

Work done in part while Attila A. Yavuz was at Oregon State University

one-time/multiple-time signatures exist (e.g., [26]), the case for polynomially-unbounded signatures seems to be more difficult. The proposal of such schemes is also necessary to support post-quantum key encapsulation schemes [11]. In this direction, the Department of Energy (DOE) and the Department of Homeland Security (DHS) have shown increased interest in proposals on post-quantum secure authentication schemes [3] for smart grids. To ensure a smooth and timely transition, National Institute of Standards and Technology (NIST) has started the initial rounds of accepting proposals for PQ secure constructions [4].

Aggregate digital signature schemes allow multiple signatures to be aggregated into a single one [10]. They are used to achieve efficient signature generation as shown in [43] with Rapid Authentication (RA) scheme. In RA, the signer precomputes a set of individual signatures in the key generation algorithm, and aggregates a subset of them to efficiently compute signatures. However, RA requires messages to be in a predefined (fixed-length) format.

It is very desirable to develop fast digital signature schemes that can avoid both the storage/re-generation of one-time signatures and the need of a predefined message format. Such schemes can potentially support the legacy systems (aggregate RSA-based signatures [31]) and be secure against quantum computers, with the advent of post-quantum aggregate signature schemes [25]. One recent attempt to address these issues was proposed in *Structure-free Compact and Rapid Authentication (SCRA)* [44]. In this paper, we show that, it is a challenging and yet feasible task to create such fast signatures by first mounting an attack to SCRA [44], and then constructing a new generic scheme that can address the aforementioned limitations.

1.1 Our Contributions

Our contributions are two-fold: (i) We identify a weakness in SCRA which leads to a universal forgery attack on its lattice-based instantiations. (ii) We then present a new scheme called *Fast Authentication from Aggregate Signatures* (`FAAS`) which achieves significant performance gains on the signer's side along with an improved security in terms of side-channel resiliency.

Attack on Lattice-Based Instantiation of SCRA [44]: We identified a flaw in generic SCRA where each signature leaks the aggregation of a subset of the private key, along with their corresponding indexes. In the lattice-based instantiation of SCRA, we show how the adversary can form a set of linear equations, and forge signatures on any message only after observing 8161 signatures. We have fully implemented our attack and forged signatures in a few milliseconds after a one-time 2.5-hour learning phase.

Fast Authentication from Aggregate Signatures (`FAAS`): We present our new signature scheme `FAAS` that can be instantiated from any aggregate signature scheme. We prove the security of `FAAS` in the random oracle model (ROM) under the hardness of breaking the underlying aggregate signature scheme. We propose two efficient instantiations of `FAAS`: (i) An instantiation with *pqNTRUsign* [25] called `FAAS-NTRU` (ii) and an instantiation with *Condensed-RSA* [31] called `FAAS-RSA`. The desired properties of `FAAS` are as follows:

Table 1: Experimental performance comparison and analysis.

Schemes	Sign	Verify	Delay	Sig Size	SK Size	PK Size	Online Phase		
							Gauss	Exp	RS
RSA [35]	8.08	0.05	8.13	372	768	386	×	✓	×
ECDSA [5]	0.73	0.93	1.66	64	32	32	×	✓	×
Ed25519 [9]	0.13	0.33	0.46	64	32	32	×	✓	×
SPHINCS [8]	13.46	0.37	13.83	41000	1088	1056	×	×	×
pqNTRUsign [25]	14.52	0.30	14.82	576	1024	1024	✓	×	✓
FAAS-RSA	**0.19**	0.06	**0.25**	768	197408	1024	×	×	×
FAAS-NTRU	**0.49**	0.71	**1.20**	3072	525328	1024	×	×	×
FAAS-NTRU′	**0.14**	0.71	**0.85**	3072	1049600	1024	×	×	×

FAAS schemes do not require any Gaussian sampling (Gauss), exponentiation (Exp) or rejection sampling (RS) at its online calculations. Therefore, they have an improved side-channel resiliency and better performance that is further explained in §5.1.
All sizes are in *Bytes (B)*. All times are in *milliseconds (ms)*. The results are obtained on a laptop equipped with an Intel i7 6th generation CPU operating at 2.6GHz. All parameter sizes are selected to provide $\kappa = 128$-bit security level (except for SPHINCS [8], that provides $\kappa = 256$-bit security). A detailed performance analysis and comparison are given in §6.

i) Improved Side-Channel Resiliency: The signature generation of FAAS only relies on signature aggregation, and therefore improves the side-channel resiliency of its base schemes (i.e., [25,31]). For instance, FAAS-NTRU does not require any Gaussian sampling algorithm (in its signature generation) which is known to be susceptible to a number of side-channel attacks (e.g., [18,22]). Besides, FAAS-NTRU offers a fixed-time sign algorithm (as opposed to its underlying scheme [25]) since it does not require any rejection sampling. Moreover, FAAS instantiations offer deterministic signing and therefore immune to side-channel attacks targeting weak pseudorandom number generators (PRNGs). Lastly, FAAS-RSA is not susceptible to the attacks targeting the square-and-multiply method on traditional RSA signatures [20,34].

ii) High Computational Efficiency: We instantiate FAAS with verification-efficient digital signature schemes to complement the benefits of the improved signature generation, and therefore achieving a low delay[3] for FAAS instantiations. For instance, FAAS-RSA and FAAS-NTRU′ improve the delay of their base schemes by 32× and 17×, respectively.

iii) Generic Design: FAAS can be instantiated with any aggregate signature. For example, several lattice-based post-quantum signature schemes (which require Gaussian and/or rejection sampling) have been proposed to the NIST post-quantum competition (e.g., Dilithium [16]). FAAS can be used to enhance the security (from side-channel perspective) and performance of these schemes provided that they offer a secure signature aggregation capability.

Limitations: FAAS has an increased private key size due to the storage of precomputed signatures. FAAS-RSA and FAAS-NTRU require 193 KB and 511 KB private keys, respectively, with $\kappa = 128$-bit security (see Table 1). This increased private key size; however, translates into 30× and 42× faster signing with an improved side-channel resiliency, for FAAS-RSA and FAAS-NTRU, respectively.

[3]Delay is defined as the aggregated time required to compute and verify a signature.

2 Preliminaries

Notation. $|a|$ denotes the bit length of variable a. $\mathcal{M}$ denotes the message space. $a \xleftarrow{\$} S$ denotes that a is selected from set S at random. In $x||y$, $||$ denotes the concatenation of bit strings of x and y. We represent vectors as bold letters (i.e., $\mathbf{a}$), and matrices are defined by bold capital letters (i.e., $\mathbf{A}$), while scalars are represented as non-bold letters (i.e., a). $||\mathbf{a}||_2$ and $||\mathbf{a}||_\infty$ denote the Euclidean norm and infinity norm of vector $\mathbf{a}$, respectively. We define hash functions $H_0 : \{0,1\}^* \rightarrow \{0,1\}^{l_0}$, $H_1 : \{0,1\}^* \rightarrow \{0,1\}^{l_1}$ and $H_2 : \{0,1\}^* \rightarrow \{0,1\}^{l_2}$ for some integers l_0, l_1 and l_2, to be defined in §6.2. $\mathcal{A}^{\mathcal{O}_1,\ldots,\mathcal{O}_i}$ denotes that algorithm $\mathcal{A}$ is provided with access to oracles $\mathcal{O}_1, \ldots, \mathcal{O}_i$.

Definition 1. *A digital signature* $\mathtt{SGN} = (\mathtt{Kg},\mathtt{Sig},\mathtt{Ver})$ *is defined as follows.*

- $(sk, PK) \leftarrow \mathtt{SGN.Kg}(1^\kappa)$: Given the security parameter κ, it outputs the public/private key pair (sk, PK).
- $\sigma \leftarrow \mathtt{SGN.Sig}(m, sk)$: Given a message m and sk, it outputs the signature σ.
- $\{0,1\} \leftarrow \mathtt{SGN.Ver}(m, \sigma, PK)$: Given a message-signature pair (m, σ), and PK, it outputs $b \in \{0,1\}$.

We say that SGN *is correct if* $1 \leftarrow \mathtt{SGN.Ver}(m, \mathtt{SGN.Sig}(m, sk), PK)$.

Definition 2. *Existential Unforgeability under Chosen Message Attack (EU-CMA) [27] experiment* $Expt^{EU-CMA}_{\mathtt{SGN},\mathcal{A}}$ *against an adversary* $\mathcal{A}$ *is as follows.*

$L_m \leftarrow \emptyset$
$(sk, PK) \leftarrow \mathtt{SGN.Kg}(1^\kappa)$
$(m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathtt{Sign}_{sk}(\cdot)}(PK)$

$\mathtt{Sign}_{sk}(m_i)$
  $\sigma_i \leftarrow \mathtt{SGN.Sig}(m_i, sk)$
  $L_m \leftarrow L_m \cup m_i$

We say $\mathcal{A}$ *wins in time* t*, and after making* q_S *signatures and* q_h *queries to random oracles (*H_1, H_2*, and* H_3*), if* $((\mathtt{SGN.Ver}(m^*, \sigma^*, PK) \wedge (m \cap L_m = \emptyset))$. The advantage of $\mathcal{A}$ is defined as $Adv^{EU\text{-}CMA}_{\mathtt{SGN},\mathcal{A}}(t, q_S, q_h) = \Pr[Exp^{EU\text{-}CMA}_{\mathtt{SGN},\mathcal{A}} = 1]$.

Definition 3. *A single-signer aggregate signature* $\mathtt{ASig} = (\mathtt{Kg},\mathtt{Sig},\mathtt{Agg},\mathtt{Ver})$ *is defined as follows.*

- $(sk, PK) \leftarrow \mathtt{Asig.Kg}(1^\kappa)$: *Given the security parameter* κ *as the input, it returns a private/public key pair* (sk, PK).
- $\sigma \leftarrow \mathtt{Asig.Sig}(m, sk)$: *Given a message* $m \in \{0,1\}^*$ *and* sk *as the input, it returns a signature* γ *of the message under* sk.
- $s \leftarrow \mathtt{Asig.Agg}(\sigma_1, \ldots, \sigma_L)$: *Given a set of signatures* $(\sigma_1, \ldots, \sigma_L)$ *as the input, it returns a single-compact signature* s.
- $\{0,1\} \leftarrow \mathtt{Asig.Ver}(\overrightarrow{m}, s, PK)$: *Given messages* $\overrightarrow{m} = (m_1, \ldots, m_L)$, s *and* PK *as the input, it returns a bit:* 1 *means* valid *and* 0 *means* invalid.

Definition 4. *Agg function, that is used to aggregate multiple messages to a single message, is defined as follows.*

- $m \leftarrow \mathtt{Agg}(m_1, \ldots, m_L)$: *Given a set of messages* $(m_1, \ldots, m_L)$ *as the input,* Agg *function returns a single message* m *as the output.*

Algorithm 1 pqNTRUsign Signature Generation [25]

$(\mathbf{v}') \leftarrow$ `pqNTRUsign.Sig`$(m, sk', \mathbf{h})$:

1: Compute $(\mathbf{u_p}, \mathbf{v_p}) = H_N(m||\mathbf{h})$ and sample $\mathbf{r} \leftarrow \chi_{\tilde{\sigma}}^N$ and $b \xleftarrow{\$} \{0,1\}$
2: Compute $\mathbf{u_1} \leftarrow \mathbf{pr} + \mathbf{u_p}$, $\mathbf{v_1} \leftarrow \mathbf{u_1 h} \mod q$ and $\mathbf{a} \leftarrow (\mathbf{v_p} - \mathbf{v_1})/\mathbf{g} \mod p$
3: **if** $||\mathbf{af}||_2 > \nu_s$ or $||\mathbf{ag}||_\infty > \nu_t$ **then go to** Step 1
4: $\mathbf{v}' = \mathbf{v}_1 + (-1)^b \mathbf{ag}$
5: **if** $||\mathbf{v}'||_\infty > q/2 - B_t$ **then go to** Step 1
6: Calculate $\mathbf{b} = (\mathbf{r} + (-1)^b \mathbf{af})$ with probability $1/\big(M_s \exp\big(-\frac{||\mathbf{af}||}{2\tilde{\sigma}^2}\big)\cosh\big(\frac{\langle \mathbf{b}, \mathbf{af}\rangle}{\tilde{\sigma}^2}\big)\big)$, else **go to** Step 1
7: Return $\mathbf{v}'$

`Agg` function is also a part of the `Asig.Ver` algorithm that allows the batch verification of multiple messages. This function can be instantiated as modular multiplication in RSA [31] or vector addition in pqNTRUsign [25].

2.1 Lattice-Based Tools

We work over a polynomial ring $\mathcal{R}_q = \mathbb{Z}_q[x]/(x^N+1)$ for a prime q and a positive integer N [25]. For `FAAS-NTRU`, we model a hash function $H_N : \{0,1\}^* \rightarrow \mathbb{Z}_q^N$. This enables generating random elements $\mathbf{m_p} = (\mathbf{u_p}, \mathbf{v_p})$ with $\mathbf{u_p} \in \mathbb{Z}_p^{N_1}$ and $\mathbf{v_p} \in \mathbb{Z}_p^{N_2}$ for a prime p and $N = N_1 + N_2$.

NTRU Lattice: Following the work in [25], we work over a NTRU lattice as an $\mathcal{R}_q$ module of rank 2. We let $f(x), g(x), h(x) \in \mathcal{R}_q$ where $f(x)$ and $g(x)$ have small coefficients and $h(x) = p^{-1}g(x)f^{-1}(x)$.

The NTRU lattice associated with $\mathbf{h}$ is defined as $\mathcal{L} = \{(\hat{\mathbf{u}}, \hat{\mathbf{v}}) \in \mathcal{R}_q^2 : \hat{\mathbf{u}}\mathbf{h} = \hat{\mathbf{v}}\}$. A vector in NTRU lattice can be written as $\mathbf{v} = \langle \hat{\mathbf{s}}, \hat{\mathbf{t}} \rangle$ where $\hat{\mathbf{s}}, \hat{\mathbf{t}} \in \mathcal{R}_q$, following [25], we refer to $\hat{\mathbf{s}}$ as the s-side and $\hat{\mathbf{t}}$ as the t-side of the vector.

Definition 5. *An N-dimensional Gaussian function* $\rho_{\tilde{\sigma},c} : \mathbb{R} \rightarrow (0,1]$) *is defined as* $\rho_{\tilde{\sigma},c}(x) \triangleq \exp(-\frac{\|x-c\|^2}{2\tilde{\sigma}^2})$. *Given a lattice* $\Lambda \subset \mathbb{R}^n$, *the discrete Gaussian distribution over* Λ *is* $D_{\Lambda,s,c}(\mathbf{x}) = \frac{\rho_{\tilde{\sigma},c}(\mathbf{x})}{\rho_{\tilde{\sigma},c}(\Lambda)}$ *for all* $\mathbf{x} \in \Lambda$.

Hoffstein et al. uses a Bimodal Gaussian distribution $\chi_{\tilde{\sigma}}^N$ [15] with standard deviation $\tilde{\sigma}$ to sample an N-dimension random vector $\mathbf{r}$. Hoffstein et al. also uses rejection sampling to ensure that the signature components do not leak any information about the private keys by checking if its norm is in $(-\frac{q}{2} + \nu_y, \frac{q}{2} - \nu_y)$ for some public parameter ν where $y \in \{s\text{-side}, t\text{-side}\}$. This is done as in the Step 3 of Algorithm 1. We also note that $\tilde{\sigma}$ in Algorithm 1 is a Gaussian distribution parameter which ensures a bound on the value of the sampled vector's coordinates. In Step 6, $b \leftarrow \{0,1\}$ is a random bit related to bimodal Gaussian distribution [15]. M_s as defined in [25], is the repetition rate which determines the rate of rejection.

3 On the Security of SCRA-NTRU

We first recall SCRA signature scheme and also present its lattice-based instantiation. We then describe the idea behind our attack, followed by its detailed description and implementation.

Algorithm 2 SCRA - pqNTRUsign Instantiation

$(sk, PK) \leftarrow$ `SCRA-NTRU.Kg`(1^{κ}):

1: Generate secrets $\mathbf{f}, \mathbf{g} \in \mathcal{R}_q$ such that $h(x) = p^{-1}g(x)f^{-1}(x)$
2: If $\mathbf{f}$ and $\mathbf{g}$ are not invertible mod q, go to Step 1
3: $sk' \leftarrow (\mathbf{f}, \mathbf{g})$, $PK' \leftarrow \mathbf{h}$ and $P \xleftarrow{\$} \{0,1\}^{l_0}$ and
4: Select integers (b, L) such that $b \cdot L = l_0$
5: $\tilde{m}_{i,j} \leftarrow (i||j||P)$, $\boldsymbol{\gamma}_{i,j} \leftarrow$ `pqNTRUsign.Sig`$(\tilde{m}_{i,j}, sk', PK')$, $i = 1, \ldots, L$ and $j = 0, \ldots, 2^b - 1$
6: $sk \leftarrow (sk', \boldsymbol{\Gamma})$ and $PK \leftarrow (PK', P)$, where $\boldsymbol{\Gamma} \leftarrow \{\boldsymbol{\gamma}_{i,j}\}_{i=1,j=0}^{L,2^b-1}$

$\sigma \leftarrow$ `SCRA-NTRU.Sig`(m, sk):

1: $(M_1^*, \ldots M_L^*) \leftarrow H_0(m||r)$ where $r \xleftarrow{\$} \{0,1\}^{\kappa}$ and $M_i^* \in [0, 2^b - 1]$, $i = 1, \ldots, L$.
2: $\mathbf{s} \leftarrow \sum_{i=1}^{L} \boldsymbol{\gamma}_{i,M_i^*}$ and $\boldsymbol{\sigma} \leftarrow (r, \mathbf{s})$

$\{0,1\} \leftarrow$ `SCRA-NTRU.Ver`(m, σ, PK):

1: $(M_1^*, \ldots M_L^*) \leftarrow H_0(m||r)$
2: $\hat{\mathbf{u}} = \boldsymbol{s}\boldsymbol{h}^{-1} \mod q$
3: **if** $||\hat{\mathbf{u}}||_{\infty} > \sqrt{(k+L)}\tau p\tilde{\sigma}$ **then** return 0
4: $(\mathbf{u_{p_i}}, \mathbf{v_{p_i}}) \leftarrow H_N(i||M_i^*||P||\mathbf{h})$ where $i = 1, \ldots, L$
5: **if** $(\hat{\mathbf{u}}, \mathbf{s}) = \sum_{i=1}^{L}(\mathbf{u_{p_i}}, \mathbf{v_{p_i}})$ **then** return 1, **else** return 0

3.1 SCRA Signature Scheme

SCRA was proposed as a generic scheme that transforms a single-signer aggregate signature scheme into a fast signature scheme. Before we highlight the weakness that leads to our attack, we briefly recall the generic SCRA signature scheme below. For a detailed description, we refer the interested reader to [44].

Key Generation: A set of signatures are precomputed for each b-bit L fields of the hash output, where $l_0 = b \cdot L$, and l_0 is the bit length of the hash output. These signatures are stored in a precomputed table containing $2^b \cdot L$ signatures. This table and the public key of the underlying aggregate signature are the private and public keys, respectively.

Signature Generation: The message is hashed with a randomness, and for each of L fields of the hash output, their corresponding precomputed signatures (retrieved from the private key of the signer) are aggregated. The randomness is sent along with the aggregated signature to enable signature verification.

Signature Verification: Individual indexes are recovered and their batch verification is performed under the signer's public key.

SCRA Lattice-Based Instantiations: SCRA was instantiated with lattice-based aggregate signatures in [17]. Recently, Hoffstein et al. [25] proposed an NTRU-based signature scheme called pqNTRUsign which offers *provably secure* single-signer secure aggregation [25]. Therefore, we instantiate SCRA with the scheme in [25]. We present this instantiation in Algorithm 2 with a reference to the signature generation of pqNTRUsign in Algorithm 1. We remark that given the similarity of pqNTRUsign with the schemes used in the instantiations

of SCRA (e.g., [17]), our attack can be directly applied to the lattice-based instantiations originally proposed in SCRA.

3.2 Our Attack

In generic SCRA, the signature generation algorithm releases an aggregation of a subset of the precomputed signature components without any masking. Furthermore, to enable signature verification, the message fields (i.e., indexes), dictating the selected precomputed components, are publicly released. These leakages can be observed in all instantiations of SCRA and they permit an adversary $\mathcal{A}$ to learn which private key components are aggregated to form the signature for a given message. In this paper, we leverage such leakages to mount a universal forgery attack on the lattice-based instantiations of SCRA.

Attack Algorithm: Since one can compute the indexes used to form the signature, each signature leaks the aggregation of L private keys. For lattice-based instantiations, these are vector additions of individual private key components.

In our first attempt, our goal was to observe enough signatures to perform a key recovery attack. Since there are $2^b \cdot L$ private key components, we have $2^b \cdot L$ variables in our linear equations, and $\mathcal{A}$ needs the observe the same number of equations/signatures. However, to solve this equation system, each equation $\mathcal{A}$ observes must be *linearly independent*. However, due to the selection of private keys in signature generation (i.e., one private key component from each L fields), the adversary can only observe $(2^b - 1) \cdot L + 1$ linearly independent equations. While one can use methods such as least mean squares to estimate the private key components, it is not possible to fully recover them with this many equations. However, we observed that $(2^b - 1) \cdot L + 1$ linearly independent equations are enough for $\mathcal{A}$ to generate signatures on any message (i.e., universal forgery). The details of this attack are given in Algorithm 3, and further explained below.

The function $\mathbf{Y} \leftarrow \texttt{echelon}(\mathbf{X})$ computes the row echelon form of matrix $\mathbf{X}$. Following the definition of *EU-CMA* in Definition 2, our attack takes place in two phases, namely the learning phase and the forgery phase.

In the attack, $\mathcal{A}$ first observes enough signatures to construct the linear equations. In Step 3, $\mathcal{A}$ parses signatures $(\mathbf{s}_i, r_i) \leftarrow \sigma_i$ and extracts $\mathbf{s}_i$'s to form the matrix $\mathbf{B}$. This matrix represents *the solutions of each linear equation, since they are derived by vector addition in* `SCRA.Sig`. In Step 4, $\mathcal{A}$ derives all the indexes from the messages as in the signature verification. Using these indexes, $\mathcal{A}$ forms a matrix $\mathbf{A}$, that represents which private key components are aggregated to derive the signature $\mathbf{s_i}$. $\mathcal{A}$ then concatenates these two matrices (as in Step 6) and calculates the *row echelon form* of the new matrix. This enables the adversary to easily form the linear combinations of these vectors to generate new signatures. Therefore, after this point, $\mathcal{A}$ selects *any message* that was not queried before, get the indexes with a simple hash function, and forms a new row from the row echelon matrix, based on the indexes of the target message. Since this matrix includes the signature components observed ($\mathbf{C} \leftarrow \mathbf{A}|\mathbf{B}$), the right side of this vector gives the signature for the selected message.

Algorithm 3 Attack on SCRA Lattice-Based Instantiation

Setup:

$(sk, PK) \leftarrow \texttt{SCRA-NTRU.Kg}(1^{\kappa})$ and $L_m \leftarrow \emptyset$

Learning:

Query $\sigma_i \leftarrow \texttt{SCRA-NTRU.Sig}(m_i, sk)$ and $L_m = L_m \cup m_i$ for $i = 1, \ldots, (2^b - 1) \cdot L + 1$

Parse $(\mathbf{s}_i, r_i) \leftarrow \sigma_i$ and form $\mathbf{B} = [\mathbf{B}|\mathbf{s_i}]$ for $i = 1, \ldots, (2^b - 1) \cdot L + 1$

$L_i = (M^*_{1,i}, \ldots, M^*_{L,i}) \leftarrow H_0(m||r_i)$ for $i = 1, \ldots, (2^b - 1) \cdot L + 1$

Set $\mathbf{A}$ s.t. $A[i, j] = 1$ if $j \in L_i$, for $i = 1, \ldots, (2^b - 1) \cdot L + 1$, $j = 1, \ldots, 2^b \cdot L$, otherwise, $A[i, j] = 0$.

$\mathbf{C} \leftarrow [\mathbf{A}|\mathbf{B}]$ and $\mathbf{C}' \leftarrow \texttt{echelon}(\mathbf{C})$

Forgery:

$m' \xleftarrow{\$} \mathcal{M}$ where $m' \notin L_m$

$r' \xleftarrow{\$} \{0, 1\}^{\kappa}$, and $L = (M^*_1, \ldots M^*_L) \leftarrow H_0(m'||r')$.

Linearly combine rows of $\mathbf{C}'$ to generate a row, such that for $i = 1 \ldots 2^b \cdot L$, $\mathbf{a}[i] = 1$ if $i \in L$ and 0 otherwise.

The right side of the new row $\mathbf{a}$ gives a valid signature over m'.

Attack implementation: We have fully implemented our attack[4] and forged a signature over the message "*May the force be with you*". We used C for the hash operations and computing SCRA signatures and Matlab for the matrix operations. Specifically, we used the predefined `rref` function in Matlab to generate the row echelon form of the matrix and then used this matrix to forge signatures. With the suggested parameters of SCRA ($b = 8, L = 32$), `rref` function of Matlab took around 2.5 hours (executed only once). After that, each forgery only took a few milliseconds. Therefore, we were able to forge signatures on any message, by observing only 8161 signatures. Note that, although SCRA can be instantiated with different (b, L) parameters, since the storage overhead is $2^b \cdot L$, increasing parameters to make our attack impractical would also make the signature scheme impractical for the signer.

4 The Proposed Scheme

Main Idea: Following the works in [43,44], we capitalize on the observations that signature aggregation of some signature schemes is significantly faster than their signature generation. `FAAS` differs from the previous constructions in the way that messages and randomness are encoded and computed: (i) `FAAS` has significantly shorter private keys since we only rely on sampling L-out-of-2^b different combinations (as in [12]) rather than encoding the message as L b-bit structures as in SCRA. (ii) Most importantly, `FAAS` masks the aggregation of private key components (i.e., individual signatures) via an aggregate one-time masking technique (elaborated below) to address the vulnerability identified in SCRA [44] (see §3 for our attack).

Aggregate One-time Masking of Signatures: The security flaw in SCRA stems from disclosing the aggregation of private key components. To efficiently

[4] www.github.com/ozgurozmen/SCRA-NTRU_ATTACK

Algorithm 4 Generic `FAAS` Scheme

$(sk, PK) \leftarrow$ `FAAS.Kg`(1^κ):

1: Select integers (k, t) such that $k \cdot |t| = l_2$ and (b, L) such that $b \cdot L = l_0$
2: $(sk', PK') \leftarrow$ `ASig.Kg`(1^κ) and $z \leftarrow \{0,1\}^\kappa$
3: $u_i \leftarrow H_1(i||z)$ and $\beta_i \leftarrow$ `ASig.Sig`(u_i, sk') for $i = 0, \ldots, t-1$
4: Set precomputed signature table $B \leftarrow \{\beta_i\}_{i=0}^{t-1}$
5: $\gamma_i \leftarrow$ `ASig.Sig`(i, sk') for $i = 0, \ldots, 2^b - 1$
6: Set precomputed signature table $\Gamma \leftarrow \{\gamma_i\}_{i=0}^{2^b-1}$
7: $sk \leftarrow (z, B, \Gamma)$ and $PK \leftarrow PK'$

$\sigma \leftarrow$ `FAAS.Sig`(m, sk):

1: $(j_1, \ldots, j_k) \leftarrow H_2(m||z)$, where each $\{j_i\}_{i=1}^k$ is interpreted as a $|t|$-bit integer.
2: $u_{j_i} \leftarrow H_1(j_i||z)$ for $i = 1, \ldots, k$
3: $u \leftarrow Agg(u_{j_1}, \ldots, u_{j_k})$
4: $\sigma_U \leftarrow$ `ASig.Agg`$(\beta_{j_1}, \ldots, \beta_{j_k})$
5: $(j_1^*, \ldots, j_L^*) \leftarrow H_0(m||u)$, where each $\{j_i\}_{i=1}^L$ is interpreted as a b-bit integer.
6: $s \leftarrow$ `ASig.Agg`$(\sigma_U, \gamma_{j_1^*}, \ldots, \gamma_{j_L^*})$ and set $\sigma \leftarrow (u, s)$

$\{0,1\} \leftarrow$ `FAAS.Ver`(m, σ, PK):

1: $(j_1^*, \ldots, j_L^*) \leftarrow H_0(m||u)$
2: $\{0,1\} \leftarrow$ `ASig.Ver`$(\langle u, j_1^*, \ldots, j_L^* \rangle, s, PK')$

overcome this, we (deterministically) generate *random* message components u_i and their corresponding signatures β_i in the key generation phase (Step 3 in `FAAS.Kg` of Algorithm 4) and then aggregate a subset of them to generate a random message-signature pair (u, σ_U) as in Step 3-4 of `FAAS.Sig` (Algorithm 4). We then use this one-time randomness σ_U to hide the aggregation of private keys at Step 6 of `FAAS.Sig`. Although the aggregated message u is released with the signature, computing the individual message components or the selected indexes is as hard as breaking the underlying signature scheme.

4.1 Generic FAAS

Generic `FAAS` is presented in Algorithm 4, and is further elaborated as follows.
Key Generation: In Step 1-2, first, parameters (k, t) and (b, L) are generated. We then create the private/public key pair of the underlying aggregate signature and a random seed z, which are used to generate two precomputed signature tables: (i) In Step 3-4, we deterministically derive t random numbers u_i with a keyed hash and compute their corresponding individual signatures β_i to be stored in table B. (ii) In Step 5-6, we generate 2^b signatures, from which L of them will be selected to encode the message in signature generation (`FAAS.Sig` Step 5-6). Finally, the tables (B, Γ) and z constitute `FAAS` private key, while the public key of the underlying aggregate signature scheme is used as `FAAS` public key.
Signature Generation: In Step 1-2, we derive *the secret indexes* $(j_1, \ldots, j_k)$ from the message m and compute their corresponding random numbers $(u_{j_1}, \ldots, u_{j_k})$ via a keyed hash. In Step 3-4, we set u as the aggregation of the random

Algorithm 5 `FAAS` pqNTRUsign instantiation

$(sk, PK) \leftarrow$ `FAAS-NTRU.Kg`(1^κ):

1: Generate secrets $\mathbf{f}, \mathbf{g} \in \mathcal{R}_q$ such that $h(x) = p^{-1}g(x)f^{-1}(x)$
2: If $\mathbf{f}$ and $\mathbf{g}$ are not invertible mod q, go to Step 1
3: $sk' \leftarrow (\mathbf{f}, \mathbf{g})$, $PK' \leftarrow \mathbf{h}$ and $z \leftarrow \{0,1\}^\kappa$
4: Select integers (k, t, b, L) as in generic `FAAS.Kg` Step 1
5: $u_i \leftarrow H_1(i||z)$, $\boldsymbol{\beta_i} \leftarrow$ `pqNTRUsign.Sig`(u_i, sk', PK'), where $|u_i| = \kappa$ for $i = 0, \ldots, t-1$
6: Set precomputed signature table $\mathbf{B} \leftarrow \{\boldsymbol{\beta}_i\}_{i=0}^{t-1}$
7: $\boldsymbol{\gamma}_i \leftarrow$ `pqNTRUsign.Sig`(i, sk'), for $i = 0, \ldots, 2^b - 1$
8: Set precomputed signature table $\boldsymbol{\Gamma} \leftarrow \{\boldsymbol{\gamma}_i\}_{i=0}^{2^b-1}$
9: $sk \leftarrow (z, \mathbf{B}, \boldsymbol{\Gamma})$ and $PK \leftarrow PK'$

$\sigma \leftarrow$ `FAAS-NTRU.Sig`(m, sk):

1: $(j_1, \ldots, j_k) \leftarrow H_2(m||z)$, where each $\{j_i\}_{i=1}^k$ is interpreted as a $|t|$-bit integer.
2: $u_{j_i} \leftarrow H_1(j_i||z)$ and $(\mathbf{u_{p_{j_i}}}, \mathbf{v_{p_{j_i}}}) \leftarrow H_N(u_{j_i}||h)$ where $i = 1, \ldots, k$
3: $(\mathbf{u_p}, \mathbf{v_p}) \leftarrow \sum_{i=1}^k (\mathbf{u_{p_{j_i}}}, \mathbf{v_{p_{j_i}}})$, and $\boldsymbol{\sigma}_U \leftarrow \sum_{i=1}^k \beta_i$
4: $(j_1^*, \ldots, j_L^*) \leftarrow H_0(m||\mathbf{u_p}||\mathbf{v_p})$, where each $\{j_i\}_{i=1}^L$ is interpreted as a b-bit integer.
5: $\mathbf{s} \leftarrow \boldsymbol{\sigma}_U + \sum_{i=1}^L \boldsymbol{\gamma}_{j_i^*}$ and $\boldsymbol{\sigma} \leftarrow (\mathbf{u_p}, \mathbf{v_p}, \mathbf{s})$

$\{0,1\} \leftarrow$ `FAAS-NTRU.Ver`(m, σ, PK):

1: $(j_1^*, \ldots, j_L^*) \leftarrow H_0(m||u)$
2: $\hat{\mathbf{u}} = \boldsymbol{s}\boldsymbol{h}^{-1} \mod \hat{q}$
3: **if** $||\hat{\mathbf{u}}||_\infty > \sqrt{(k+L)}\tau p \tilde{\sigma}$ **then** return 0
4: $(\mathbf{u_{p_i}}, \mathbf{v_{p_i}}) \leftarrow H_N(j_i^*||\mathbf{h})$ where $i = 1, \ldots, L$
5: **if** $(\hat{\mathbf{u}}, \mathbf{s}) = (\mathbf{u_p}, \mathbf{v_p}) + \sum_{i=1}^L (\mathbf{u_{p_i}}, \mathbf{v_{p_i}})$ **then** return 1, **else** return 0

$(u_{j_1}, \ldots, u_{j_k})$ and aggregate their corresponding signatures $(\beta_{j_1}, \ldots, \beta_{j_k})$ fetched from table B, as σ_U. In Step 5-6, we first encode the message and u to get indexes $(j_1^*, \ldots, j_L^*)$, and then mask the aggregation of $(\gamma_{j_1^*}, \ldots, \gamma_{j_L^*})$ with σ_U as $s \leftarrow$ `ASig.Agg`$(\sigma_U, \gamma_{j_1^*}, \ldots, \gamma_{j_1^*})$. We set `FAAS` signature as $\sigma = (u, s)$.
Signature Verification: This algorithm checks if the aggregated random number u and indexes $(j_1^*, \ldots, j_L^*) \leftarrow H_0(m||u)$ are verified with s under PK'.

Remark 1. It is essential to keep individual random messages and their indexes as secrets. We do this with an aggregation function $u \leftarrow Agg(u_{j_1^*}, \ldots, u_{j_k^*})$ as in Step 4 for random message components. The aggregation function $Agg(.)$ is instantiated as modular multiplication and vector addition in Condensed RSA (C-RSA) [31] and pqNTRUSign [25], respectively. We derive indexes $(j_1, \ldots, j_k)$, which select random numbers to be aggregated, via the private key z. Therefore, unlike public indexes $(j_1^*, \ldots, j_L^*)$ that are used to encode the message, secret indexes $(j_1, \ldots, j_k)$ are only known to the signer.

4.2 FAAS Instantiations

`FAAS` can be instantiated with any single-signer aggregate signature scheme. We propose two efficient instantiations of `FAAS` as below.

Algorithm 6 Instantiation of FAAS with *C-RSA*

$(sk, PK) \leftarrow$ FAAS-C-RSA.Kg(1^κ):

1: Generate two large primes (p, q) and $n \leftarrow p \cdot q$. Compute (e, d) such that $e \cdot d \equiv 1 \bmod \phi(n)$, where $\phi(n) \leftarrow (p-1)(q-1)$
2: $sk' \leftarrow (n, d)$, $PK' \leftarrow (n, e)$ and $z \leftarrow \{0,1\}^\kappa$
3: Select integers (k, t, b, L) as in generic FAAS.Kg Step 1
4: $u_i \leftarrow H_1(i||z)$ and $\beta_i \leftarrow {u_i}^d \bmod n$, where $|u_i| = |n|$ for $i = 0, \ldots, t-1$
5: Set precomputed signature table $B \leftarrow \{\beta_i\}_{i=0}^{t-1}$
6: $\gamma_i \leftarrow H_F(i)^d \bmod n$, for $i = 0, \ldots, 2^b - 1$, where $H_F : \{0,1\}^* \rightarrow Z_n^*$
7: Set precomputed signature table $\Gamma \leftarrow \{\gamma_i\}_{i=0}^{2^b-1}$
8: $sk \leftarrow (z, B, \Gamma)$ and $PK \leftarrow PK'$

$\sigma \leftarrow$ FAAS-C-RSA.Sig(m, sk):

1: $(j_1, \ldots, j_k) \leftarrow H_2(m||z)$, where each $\{j_i\}_{i=1}^k$ is interpreted as a $|t|$-bit integer.
2: $u_{j_i} \leftarrow H_1(j_i||z)$ for $i = 1, \ldots, k$
3: $u \leftarrow \prod_{i=1}^k u_{j_i} \bmod n$ and $\sigma_U \leftarrow \prod_{i=1}^k \beta_{j_i} \bmod n$
4: $(j_1^*, \ldots, j_L^*) \leftarrow H_0(m||u)$, where each $\{j_i\}_{i=1}^L$ is interpreted as a b-bit integer.
5: $s \leftarrow \sigma_U \cdot \prod_{i=1}^L \gamma_{j_i^*} \bmod n$ and set $\sigma \leftarrow (u, s)$

$\{0,1\} \leftarrow$ FAAS-C-RSA.Ver(m, σ, PK):

1: $(j_1^*, \ldots, j_L^*) \leftarrow H_0(m||u)$
2: **if** $s^e = u \cdot \prod_{i=1}^L H_F(j_i^*) \bmod n$ **then** return 1, **else** return 0

Lattice-Based Instantiation (FAAS-NTRU): Lattice-based signature schemes provide a viable post-quantum security promise [16]. Among the identified lattice-based signature schemes with secure aggregation [17,25], pqNTRUsign [25] offers fast verification with a slow signature generation that requires Gaussian sampling. Thus, FAAS-NTRU improves the security and signing efficiency of pqNTRUsign by eliminating the Gaussian sampling and rejection sampling from the online signature generation phase. The detailed description of FAAS-NTRU is presented in Algorithm 5, that refers to pqNTRUsign signature generation algorithm defined in Algorithm 1, to refrain from repetitions in the algorithm description. Notice that expensive calculations such as Gaussian sampling and polynomial multiplication are done in the key generation algorithm (once and offline). At the signing phase, only polynomial additions are performed.

RSA-Based Instantiation (FAAS-RSA): We instantiate FAAS with Condensed RSA (C-RSA) [31], which is secure under the RSA assumption in the ROM [7]. The signature generation of C-RSA requires an exponentiation over a large modulus, whereas its verification only requires an exponentiation over a small modulus (e.g., 65537). Therefore, FAAS-RSA, given in Algorithm 6, gains significant improvements over C-RSA in terms of signature generation.

5 Security Analysis

Theorem 1. $Adv^{EU\text{-}CMA}_{FAAS,\mathcal{A}}(t, q_S, q_h) \leq Adv^{A\text{-}EU\text{-}CMA}_{Asig,\mathcal{B}}(t', q'_S, q'_h)$ *where* $t' = O(t) + 2q_S(t_{RNG} + t_{Sig} + t_{Agg})$, $q'_S \geq 2q_S$ *and* $q_H = q'_H$.

Proof. Please refer to Appendix B

5.1 Security and Performance of Online Operations

Side-channel attacks pose a serious threat to cryptographic implementations. Some critical operations that are prone to side-channel attacks are given below.
Gaussian Sampling: Lattice-based cryptography offers efficient solutions with post-quantum security promise. However, most of the efficient lattice-based signature schemes require a (high precision) sampling from a distribution, mostly a Gaussian, which not only degrades their performance on the signer's side but also is highly prone to side-channel attacks. For instance, BLISS [15], as one of the most efficient instances of such schemes, has been targeted with a number of side-channel attacks [18,22]. Secure implementation approaches might mitigate some of these side-channel attacks; however, they are deemed to be a highly challenging and error-prone task [16].
Rejection Sampling: This operation is required in lattice-based signatures to ensure signatures do not leak information about the private key. The number of rejections can significantly decrease the performance of a scheme. For instance, in pqNTRUsign [25], the probability that the signature lies in a desired range is only 6%. Aside from the performance burdens, an efficient variant of this algorithm is shown to be prone to side-channel attacks [18,22]. These attacks showed the vulnerability of the Bernoulli-based algorithm for rejection sampling in BLISS signature.
Exponentiation and PRNG: There has been many attacks on the efficient exponentiations and elliptic curve scalar multiplications [13,20,29]. While countermeasures were proposed in [36], similar to the blinding technique, they incur performance sacrifice. The security of PRNGs is highly dependent on the hardware. Although one can find secure PRNGs in well-developed CPUs, the PRNG implementations in low-end IoT processors are prone to attacks. Therefore, it is a desirable property for a signature scheme to be deterministic (i.e., do not require any fresh randomness in signature generation phase) [9].

`FAAS` instantiations do not require any of the above operations in their signing algorithm. Therefore, we believe `FAAS` instantiations can offer improved side-channel resiliency and easy implementation as compared to some of their underlying schemes.

6 Performance Evaluation and Comparison

We first give the analytical costs of `FAAS` instantiations and their counterparts in terms of computational overhead and key/signature sizes. We then outline our experimental setup, parameters and provide a detailed experimental comparison.

6.1 Analytical Performance Analysis

Key Generation: Key generation of `FAAS` instantiations require the computation of two tables, and therefore, it is more expensive than their base schemes. Specifically, to generate the two tables, $2^b + t$ signatures of the underlying aggregate signature schemes should be computed.

Signature Generation: FAAS signature generation requires signature aggregations $(k+L)\cdot$ ASig.Agg$(\cdot)$, message aggregations $k\cdot$ Agg$(\cdot)$ and hash calls $k\cdot H(\cdot)$.

pqNTRUsign [25] requires Gaussian sampling and polynomial multiplication to generate a signature. This is reduced to polynomial additions and mapping functions H_N in FAAS-NTRU. We present a variant of FAAS-NTRU (referred to as FAAS-NTRU$'$) to improve the efficiency of signature generation with the cost of an increased private key size. Our implementation (see §6.2) showed that the mapping function in FAAS-NTRU takes a significant time. FAAS-NTRU$'$ stores the results of the mappings as the private key to eliminate this overhead. Aside from the Gaussian sampling, pqNTRUsign also requires a rejection sampling to ensure that signatures do not leak information about the private key distribution. Due to rejection sampling, the signature generation of this scheme does not have a constant time, whereas FAAS instantiations do not require any rejection sampling during signature generation.

FAAS-RSA signing only requires a few hash calls and modular multiplications over n, while RSA takes an exponentiation with a large exponent d.

Signature Verification and Delay: FAAS instantiations add a slight overhead to the verification of their base schemes, which is equal to L message aggregations. However, since message aggregation is efficient, this only incurs a slight overhead, especially considering the overall gain in terms of the total delay due to the highly improved signature generation.

Storage and Transmission Overhead: FAAS requires two tables to be stored at the signer's side, with the size of $(2^b+t+1)\cdot|\sigma_i|+\kappa$ where $\sigma_i \leftarrow$ ASig.Sig$(\cdot)$. Moreover, in addition to their base scheme, FAAS requires an aggregated randomness which makes the total signature size $|s|+|u|$. Note that, in FAAS-NTRU, the signature size changes from $|s|$ bits to $|v'|$ bits, since the 't-side' of the vector should be transmitted for aggregate verification in pqNTRUsign [25]. Therefore, the signature size of FAAS-NTRU increases slightly more. The public key size of FAAS instantiations is the same with that of their base signature schemes.

6.2 Performance Evaluation

Experimental Setup: We implemented FAAS instantiations on a laptop equipped with Intel i7 Skylake 2.6GHz processor and 12 GB RAM. Our operating system was Ubuntu 16.04 with gcc version 5.4.0.

Software Libraries and Implementation: We developed FAAS instantiations[5] in C. We implemented FAAS-RSA with GMP due to its optimized modular arithmetic operations [21]. We used the open-source pqNTRUsign implementation available in NTRU open-source project [25] to develop FAAS-NTRU. We used Blake2 as our hash function (as in SPHINCS [8]), due to its high efficiency [6].

We ran the open-source implementations of our state-of-the-art counterparts in our experimental setup, to draw a fair comparison. We benchmarked the ECDSA in MIRACL library [39] and RSA in GMP library [21]. We benchmarked Ed25519 and SPHINCS using their Supercop implementations. Lastly, we used the open-source implementation of pqNTRUsign [25].

[5] www.github.com/ozgurozmen/FAAS

Parameters We selected parameters to achieve $\kappa = 128$-bit security.
PQ Secure Schemes: We used the suggested parameters providing $\kappa \approx 128$-bit security for pqNTRUsign [25]. More specifically, $\tilde{\sigma} = 107$, $N = 512$, and $q = 2^{16} + 1$ and $d = 77$ to achieve $\kappa = 128$. For SPHINCS, we refer the reader for the suggested parameters to [8].
Traditional Schemes: We selected $|n| = 3072$ bit, $|e| = 17$ bit and $|d| \approx 3072$ bit for RSA-based schemes. We chose $|p'| = |q'| = 256$ bit for ECC-based schemes.
FAAS Parameters: FAAS parameters are selected as $(b, L) = (8, 32)$ and $(k, t) = (32, 256)$ for $l_0 = l_2 = 256$. The security of these parameters depend on how many different combinations one can derive with k-out-of-t precomputed components, that is $\binom{t}{k} = \binom{2^b}{L}$. With current parameters, there are 2^{141} different combinations that can be created. Another important aspect is to keep the indexes secret. As discussed in §4, this ensures that presented attack cannot be applied to FAAS. Since we are concatenating a secret (z) in the hash call (H_1), the indexes will remain as secret. On the other hand, one can attack H_0 and try to obtain an m^* such that $H_0(m||u)$ corresponds to the same indexes as $H_0(m^*||u)$. However, since u is a random value derived based on secret indexes, the attacker must conduct a *target collision attack* to find such m^*. Since, any permutation of the indexes would correspond to a collision on H_0, there are $k!$ different possible index permutations. Thus, the probability to find such an m^* is $\frac{L!}{2^{2^b}}$. With the current parameter selection, the probability for this is 2^{-138}. Since the underlying signature schemes' parameters are selected to provide $\kappa = 128$-bit security, all in all, FAAS instantiations offer $\kappa = 128$-bit security.

Experimental Comparison: Table 1 shows numerical evaluation and comparison of FAAS instantiations and their counterparts.

FAAS instantiations offer notably faster signing over their base schemes with a slightly slower verification. (i) FAAS-NTRU and FAAS-NTRU′ improve pqNTRUsign [25] signature generation by 29.67× and 105.29×, respectively. (ii) For FAAS-RSA, signature generation is over 40× faster than traditional RSA.

However, FAAS instantiations require storing a private key up to 1 MB (Table 1). With their improved side-channel resiliency and fast signature generation, FAAS instantiations can be preferred for delay-aware applications where the signer can tolerate storing up to 1MB of private key. We observed that the signing cost of FAAS-NTRU was dominated by the mapping functions, which map messages to vectors. We also noticed that these vectors can be stored as a private key component, instead of being deterministically generated during signing. This resulted in a trade-off between the signing time and private key size, where signing speeds up 3.55× with a 2× increased private key size (Table 1).

Recall that, SCRA [44] does not use a masking strategy, and therefore, leaks its private key (as shown in §3). Since FAAS uses an efficient and constant-size aggregate masking strategy, its signature generation requires only twice as much signature aggregations and k message aggregations compared to insecure SCRA. This results in an approximately three times slower signature generation. The signature verification times of the both schemes are highly similar. Moreover, since FAAS relies on an efficient message encoding (see §4), the private key of FAAS is $L\times$ smaller than that of SCRA. In practice, since L is selected as 32,

this results in a significant improvement in terms of private key size. Therefore, FAAS addresses the flaws of SCRA with a small computation overhead and a more compact private key size.

7 Related Work

Online/offline signatures [19,33,38,43] offer fast signing since they precompute tokens for each message to be signed in the offline phase. In the online phase, these precomputed tokens are used to provide efficient signature generation. However, such methods incur linear storage with respect to the number of messages to be signed. Moreover, as tokens are depleted, they should be renewed that might introduce further overhead. Therefore, we believe they may not be practical for real-time networks that require continuous signature generation.

There are many schemes that leverage signature aggregation to ensure authentication and integrity in outsourced databases (e.g., [31,32,40]). In such applications, the signatures of a small set of messages with well-defined indexes (e.g., signatures belonging to some row elements in a database table) are aggregated to obtain compact signatures for the response of database queries [32]. Despite their merits, potential security issues that stem from the homomorphic properties of these signatures were pointed out [30,32]. Specifically, it has been shown that since aggregate signatures are mutable, one can create "new signatures" on data items that have not been explicitly queried by combining previously obtained aggregate signatures. To prevent this, immutable signatures (e.g., [30,32]) have been developed, which generally rely on one-time masking and/or sentinel signatures. Recently, signature schemes that depend on secure aggregation (e.g. RA [43] and SCRA [44]) have been proposed. However, as discussed, RA [43] is an online/offline signature with a dependency on predefined structures in messages. In this paper, we showed that an adversary can forge signatures on any message in SCRA by observing a small number of signatures.

8 Conclusion

We first presented an attack to SCRA signature scheme that can forge signatures over any message by observing only 8161 signatures. We fully implemented our attack and forged signatures in only a few milliseconds after a one-time 2.5-hour preparation phase. We then proposed a new generic signature scheme (i.e., FAAS) that can transform any secure single-signer aggregate signature into a signer efficient signature scheme. We proposed two instantiations of FAAS called FAAS-RSA and FAAS-NTRU that can offer up to 42× and 105× faster signature generation as compared to their base signature schemes, respectively. Moreover, FAAS instantiations do not require some operations that are vulnerable to side-channel attacks, and therefore, they provide an improved side-channel resiliency, where FAAS-NTRU also provides a post-quantum promise.

Acknowledgments. We would like to thank Zhenfei Zhang and the anonymous reviewers for their insightful comments and suggestions. This work is supported by the Department of Energy Award DE-OE0000780 and NSF Award #1652389.

References

1. Ieee guide for wireless access in vehicular environments (wave) - architecture. IEEE Std 1609.0-2013 pp. 1–78 (March 2014)
2. D-Wave Systems Previews 2000-Qubit Quantum System (2016), https://www.dwavesys.com/press-releases/d-wave-systems-previews-2000-qubit-quantum-system
3. The cyber resilient energy delivery consortium (credc) (2018), https://cred-c.org/
4. Post-quantum cryptography standardization conference (2018), https://csrc.nist.gov/Projects/Post-Quantum-Cryptography
5. American Bankers Association: ANSI X9.62-1998: Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA) (1999)
6. Aumasson, J.P., Henzen, L., Meier, W., Phan, R.C.W.: Sha-3 proposal blake. Submission to NIST (Round 3) (2010), http://131002.net/blake/blake.pdf
7. Bellare, M., Garay, J., Rabin, T.: Fast batch verification for modular exponentiation and digital signatures. In: Advances in Cryptology EUROCRYPT'98, Lecture Notes in Computer Science, vol. 1403, pp. 236–250. Springer Berlin Heidelberg (1998)
8. Bernstein, D.J., Hopwood, D., Hülsing, A., Lange, T., Niederhagen, R., Papachristodoulou, L., Schneider, M., Schwabe, P., Wilcox-O'Hearn, Z.: SPHINCS: Practical stateless hash-based signatures. In: Advances in Cryptology – EUROCRYPT 2015: 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 368–397. Springer Berlin Heidelberg (2015)
9. Bernstein, D., Duif, N., Lange, T., Schwabe, P., Yang, B.Y.: High-speed high-security signatures. Journal of Cryptographic Engineering 2(2), 77–89 (2012)
10. Boneh, D., Gentry, C., Lynn, B., Shacham, H.: Aggregate and verifiably encrypted signatures from bilinear maps. In: Biham, E. (ed.) Advances in Cryptology — EUROCRYPT 2003. pp. 416–432. Springer Berlin Heidelberg (2003)
11. Bos, J., Costello, C., Ducas, L., Mironov, I., Naehrig, M., Nikolaenko, V., Raghunathan, A., Stebila, D.: Frodo: Take off the ring! practical, quantum-secure key exchange from lwe. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. pp. 1006–1018. CCS '16, ACM, New York, NY, USA (2016), http://doi.acm.org/10.1145/2976749.2978425
12. Bos, J.N.E., Chaum, D.: Provably unforgeable signatures. In: Brickell, E.F. (ed.) Advances in Cryptology — CRYPTO' 92. pp. 1–14. Springer Berlin Heidelberg (1993)
13. Brier, É., Joye, M.: Weierstraß elliptic curves and side-channel attacks. In: Naccache, D., Paillier, P. (eds.) Public Key Cryptography. pp. 335–345. Springer Berlin Heidelberg, Berlin, Heidelberg (2002)
14. Coron, J., Naccache, D.: Boneh et al.'s k-element aggregate extraction assumption is equivalent to the diffie-hellman assumption. In: Proceedings of the 9th International Conference on the Theory and Application of Cryptology (ASIACRYPT 03'). pp. 392–397 (2003)
15. Ducas, L., Durmus, A., Lepoint, T., Lyubashevsky, V.: Lattice signatures and bimodal gaussians. In: Canetti, R., Garay, J.A. (eds.) Advances in Cryptology – CRYPTO 2013: 33rd Annual Cryptology Conference. Proceedings, Part I. pp. 40–56. Springer Berlin Heidelberg (2013)
16. Ducas, L., Lepoint, T., Lyubashevsky, V., Schwabe, P., Seiler, G., Stehle, D.: Crystals – dilithium: Digital signatures from module lattices. Cryptology ePrint Archive, Report 2017/633 (2017), https://eprint.iacr.org/2017/633

17. El Bansarkhani, R., Buchmann, J.: Towards lattice based aggregate signatures. In: Progress in Cryptology AFRICACRYPT 2014. Lecture Notes in Computer Science, vol. 8469, pp. 336–355. Springer International Publishing (2014)
18. Espitau, T., Fouque, P., Gérard, B., Tibouchi, M.: Side-channel attacks on BLISS lattice-based signatures: Exploiting branch tracing against strongswan and electromagnetic emanations in microcontrollers. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017. pp. 1857–1874 (2017)
19. Even, S., Goldreich, O., Micali, S.: Online/offline digital signatures. In: Proceedings on Advances in Cryptology (CRYPTO '89). pp. 263–275. Springer-Verlag (1989)
20. Genkin, D., Valenta, L., Yarom, Y.: May the fourth be with you: A microarchitectural side channel attack on several real-world applications of curve25519. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. pp. 845–858. CCS '17, ACM, New York, NY, USA (2017), http://doi.acm.org/10.1145/3133956.3134029
21. Granlund, T.: GNU multiple precision arithmetic library 6.1.2. https://gmplib.org/
22. Groot Bruinderink, L., Hülsing, A., Lange, T., Yarom, Y.: Flush, gauss, and reload – a cache attack on the bliss lattice-based signature scheme. In: Gierlichs, B., Poschmann, A.Y. (eds.) Cryptographic Hardware and Embedded Systems – CHES 2016. pp. 323–345. Springer Berlin Heidelberg (2016)
23. Gungor, V.C., Sahin, D., Kocak, T., Ergut, S., Buccella, C., Cecati, C., Hancke, G.P.: Smart grid technologies: Communication technologies and standards. IEEE Transactions on Industrial Informatics 7(4), 529–539 (2011)
24. Harding, J., Powell, G., Yoon, R., Fikentscher, J., Doyle, C., Sade, D., Lukuc, M., Simons, J., Wang, J.: Vehicle-to-Vehicle Communications: Readiness of V2V Technology for Application. U.S. Department of Transportation National Highway Traffic Safety Administration (NHTSA) (August 2014)
25. Hoffstein, J., Pipher, J., Whyte, W., Zhang, Z.: A signature scheme from learning with truncation. Cryptology ePrint Archive, Report 2017/995 (2017), https://eprint.iacr.org/2017/995
26. Kalach, K., Safavi-Naini, R.: An efficient post-quantum one-time signature scheme. In: Dunkelman, O., Keliher, L. (eds.) Selected Areas in Cryptography – SAC 2015. pp. 331–351. Springer International Publishing, Cham (2016)
27. Katz, J., Lindell, Y.: Introduction to Modern Cryptography (Chapman & Hall/Crc Cryptography and Network Security Series). Chapman & Hall/CRC (2007)
28. Kelly, J.: A preview of bristlecone, google?s new quantum processor (2018), https://ai.googleblog.com/2018/03/a-preview-of-bristlecone-googles-new.html
29. Kocher, P.C.: Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In: Koblitz, N. (ed.) Advances in Cryptology — CRYPTO '96. pp. 104–113. Springer Berlin Heidelberg, Berlin, Heidelberg (1996)
30. Ma, D., Tsudik, G.: A new approach to secure logging. ACM Transaction on Storage (TOS) 5(1), 1–21 (2009)
31. Mykletun, E., Narasimha, M., Tsudik, G.: Signature bouquets: Immutability for aggregated/condensed signatures. In: Proceedings of the 9th European Symposium on Research in Computer Security (ESORICS '04). pp. 160–176. Springer-Verlag (2004)
32. Mykletun, E., Tsudik, G.: Aggregation queries in the database-as-a-service model. In: Proceedings of the 20th IFIP WG 11.3 working conference on Data and Applications Security. pp. 89–103. DBSEC'06, Springer-Verlag (2006)

33. Naccache, D., M'Raïhi, D., Vaudenay, S., Raphaeli, D.: Can D.S.A. be improved? Complexity trade-offs with the digital signature standard. In: Proceedings of the 13th International Conference on the Theory and Application of Cryptographic Techniques (EUROCRYPT '94). pp. 77–85 (1994)
34. Nguyen, P.Q., Shparlinski, I.E.: The insecurity of the elliptic curve digital signature algorithm with partially known nonces. Designs, Codes and Cryptography 30(2), 201–217 (2003)
35. Rivest, R., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM 21(2), 120–126 (1978)
36. Schindler, W.: Exclusive exponent blinding may not suffice to prevent timing attacks on rsa. In: Güneysu, T., Handschuh, H. (eds.) Cryptographic Hardware and Embedded Systems – CHES 2015. pp. 229–247. Springer Berlin Heidelberg (2015)
37. Seo, S.H., Won, J., Bertino, E., Kang, Y., Choi, D.: A security framework for a drone delivery service. In: Proceedings of the 2Nd Workshop on Micro Aerial Vehicle Networks, Systems, and Applications for Civilian Use. pp. 29–34. DroNet '16, ACM (2016)
38. Shamir, A., Tauman, Y.: Improved online/offline signature schemes. In: Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology. pp. 355–367. CRYPTO '01, Springer-Verlag, London, UK (2001)
39. Shamus: Multiprecision integer and rational arithmetic c/c++ library (MIRACL). https://github.com/miracl/MIRACL, Last Accessed on 30/01/2018.
40. Song, W., Wang, B., Wang, Q., Peng, Z., Lou, W.: Tell me the truth: Practically public authentication for outsourced databases with multi-user modification. Information Sciences 387, 221 – 237 (2017)
41. Tesfay, T., Boudec, J.Y.L.: Experimental comparison of multicast authentication for wide area monitoring systems. IEEE Transactions on Smart Grid PP(99) (2017)
42. Won, J., Seo, S.H., Bertino, E.: A secure communication protocol for drones and smart objects. In: Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security. pp. 249–260. ASIA CCS '15, ACM (2015)
43. Yavuz, A.A.: An efficient real-time broadcast authentication scheme for command and control messages. IEEE Transactions on Information Forensics and Security 9(10), 1733–1742 (2014)
44. Yavuz, A.A., Mudgerikar, A., Singla, A., Papapanagiotou, I., Bertino, E.: Real-time digital signatures for time-critical networks. IEEE Transactions on Information Forensics and Security 12(11), 2627–2639 (2017)

Appendix A Security Definitions

Definition 6. *Aggregate Existential Unforgeability under Chosen Message Attack (A-EU-CMA) for a single user aggregate signature is as follows.*

$Exp^{A\text{-}EU\text{-}CMA}_{\texttt{Asig},\mathcal{A}}(1^\kappa)$:

$L_m \leftarrow \emptyset$
$(sk, PK) \leftarrow \texttt{Asig.Kg}(1^\kappa)$
$(\overrightarrow{m}^*, \sigma^*) \leftarrow \mathcal{A}^{\texttt{SigA}_{sk}(\cdot), RO(\cdot)}(PK)$

$\texttt{SigA}_{sk}(\overrightarrow{m})$
  $\gamma_i \leftarrow \texttt{Asig.Sig}(m_i, sk)$ for $i = 1, \ldots, j$
  $\sigma \leftarrow \texttt{Asig.Agg}(\gamma_1, \ldots, \gamma_j)$
  $L_m \leftarrow L_m \cup \overrightarrow{m}$

We say $\mathcal{A}$ wins in time t, and after q_S and q_h queries if $((\texttt{Asig.Ver}(\overrightarrow{m}^*, \sigma^*, PK) \wedge (\overrightarrow{m}^* \cap L_m = \emptyset))$. *The A-EU-CMA advantage of* $\mathcal{A}$ is defined as $Adv^{A\text{-}EU\text{-}CMA}_{\texttt{Asig},\mathcal{A}}(t, q_S, q_h) = \Pr[Exp^{A\text{-}EU\text{-}CMA}_{\texttt{Asig},\mathcal{A}} = 1]$.

FAAS requires that the underlying aggregate signature achieves *k-element Aggregate Extraction (AE)* property [10,14], which is defined in the following.

Definition 7. *For a given aggregate signature* $s \leftarrow \texttt{SigA}_{sk}(\overrightarrow{m})$ *computed on k individual data items* $\overrightarrow{m} = (m_1, \ldots, m_k)$*, it is difficult to extract the individual signatures* $(\gamma_1, \ldots, \gamma_k)$ *of* $(m_1, \ldots, m_k)$ *provided that only s is known to the extractor.*

Initially, Boneh et al. [10] assumed that it is a hard problem to extract individual BLS signatures given an aggregate BLS signature, which was then proven to hold in [14] under the Computational Diffie-Hellmann assumption. We note that C-RSA [31] and pqNTRUsign [25], which are used in FAAS instantiations, achieve this property (see Appendix B for a discussion on pqNTRUsign).

Appendix B Proof of Theorem 5.1

Theorem 5.1 $Adv^{EU\text{-}CMA}_{\texttt{FAAS},\mathcal{A}}(t, q_S, q_h)$ *is bounded as follows.*

$$Adv^{EU\text{-}CMA}_{\texttt{FAAS},\mathcal{A}}(t, q_S, q_h) \leq Adv^{A\text{-}EU\text{-}CMA}_{\texttt{Asig},\mathcal{B}}(t', q'_S, q'_h)$$

where $t' = O(t) + 2q_S(t_{\texttt{RNG}} + t_{\texttt{Sig}} + t_{\texttt{Agg}})$, $q'_S \geq 2q_S$ *and* $q_H = q'_H$.

Proof. We prove that if there exists an adversary $\mathcal{A}$ that can break the *EU-CMA* security of FAAS signatures as in Definition 2 in time t, and after making q_S and q_H signature generation and hash queries, respectively, then one can use $\mathcal{A}$ to build an algorithm $\mathcal{B}$ that can break the *A-EU-CMA* security of the underlying aggregate signature scheme ASig signatures as in Definition 6 in time t', and after making q'_S and q'_H signature generation and hash queries, respectively.
Setup: $\mathcal{B}$ is initiated with the public key of the underlying aggregate signature scheme PK' where $(sk', PK') \leftarrow \texttt{Asig.Kg}(1^\kappa)$

– $\mathcal{B}$ setups the *H-Sim* function to handle $\mathcal{A}$'s $RO(\cdot)$ queries. That is, the cryptographic hash function H is modeled as a random oracle via *H-Sim* as follows.

- $h \leftarrow H\text{-}Sim(x, L_H, i)$: If $x \in L_H$ then $H\text{-}Sim$ returns the corresponding value $h \leftarrow L_H(x)$. Otherwise, it returns $h \xleftarrow{\$} \{0,1\}^{l_i}$ where $i \in \{0,1,2\}$ as the answer, inserts (x, h) into L_H, respectively.

– $\mathcal{B}$ keeps tables (L_H, L_m) to keep track of random oracle and signature queries.
– $\mathcal{B}$ picks $z \xleftarrow{\$} \{0,1\}^{\kappa}$, sets $PK \leftarrow PK'$ and passes PK' to $\mathcal{A}$ as the public key of `FAAS` signature scheme.

Execute $\mathcal{A}^{\texttt{FAAS.Sig}(\cdot), RO(\cdot)}$: $\mathcal{B}$ replies $\mathcal{A}$'s signature and hash queries as follows.

– To reply a signature queries on m_i for $i \in (1, \ldots, q_S)$ $\mathcal{B}$ works as follows:
  1. $\mathcal{B}$ computes $(j_1, \ldots, j_k) \leftarrow H\text{-}Sim(m_i||z, L_H, 2)$ where each $\{j_i\}_{i=1}^{k}$ is interpreted as a $|t|$-bit integer, then computes $(u_{j_1} \ldots u_{j_k}) \leftarrow H\text{-}Sim(j_i||z, L_H, 1)$ and $u_i \leftarrow Agg(u_1, \ldots, u_k)$.
  2. $\mathcal{B}$ queries $\sigma_{u_i} \leftarrow \texttt{SigA}_{sk'}(u_1, \ldots, u_k)$ and stores (u_i, σ_{U_i}) in L_m.
  3. $\mathcal{B}$ computes $(j_1^*, \ldots, j_k^*) \leftarrow H\text{-}Sim(m_i||u_i, L_H, 0)$ where each $\{j_i^*\}_{i=1}^{L}$ interpreted as a $|b|$-bit integer.
  4. $\mathcal{B}$ queries $\sigma_{m_i} \leftarrow \texttt{Asig.A}_{sk'}(j_1^*, \ldots, j_L^*)$ and $s \leftarrow \texttt{Asig.Agg}(\sigma_{u_i}, \sigma_{m_i})$ and stores (m_i, σ_{m_i}) in L_m.
  5. $\mathcal{B}$ returns $\sigma_i = (u_i, s_i)$.
– $RO(\cdot)$ Queries: $\mathcal{B}$ replies to $\mathcal{A}$'s queries on H_0, H_1 and H_2 hash functions on input x by initiating the $H\text{-}Sim(x, L_H, i)$ oracle where $i \in \{0,1,2\}$ as defined in §2.

Forgery of $\mathcal{A}$: After at most q_S signature queries, $\mathcal{A}$ outputs a forgery $\langle m^*, (u^*, s^*)\rangle$ under PK. As in Definition 6, $\mathcal{A}$ wins if $(\texttt{FAAS.Ver}(m^*, \sigma^* = (u^*, s^*), PK) \wedge (m^* \cap L_m = \emptyset))$. Otherwise, $\mathcal{A}$ loses in the experiment. If $\mathcal{A}$ loses in the experiment, $\mathcal{B}$ also loses and outputs 0.
Forgery of $\mathcal{B}$: Given $\mathcal{A}$'s successful forgery $\langle m^*, (u^*, s^*)\rangle$, $\mathcal{B}$ works as follows.

1. First computes $(\tilde{j}_1^*, \ldots, \tilde{j}_k^*) \leftarrow H\text{-}Sim(m^*||z, L_H, 2)$ and $(u^*_{\tilde{j}_1^*}, \ldots, u^*_{\tilde{j}_k^*}) \leftarrow H\text{-}Sim(\tilde{j}_i^*||z, L_H, 1)$ and checks if $u^* \stackrel{?}{=} Agg(u^*_{\tilde{j}_1^*}, \ldots, u^*_{\tilde{j}_k^*})$ holds, it outputs 0 (this event can happen with a negligible probability since it is equivalent to breaking the hash function).
2. $\mathcal{B}$ then computes $(j_1^*, \ldots, j_L^*) \leftarrow H\text{-}Sim(m^*||u^*, L_H, 0)$ where each $\{j_i^*\}_{i=1}^{L}$ is interpreted as a $|b|$-bit integer.
3. $\mathcal{B}$ queries $\sigma_m \leftarrow \texttt{SigA}_{sk'}(j_1^*, \ldots, j_L^*)$, computes $\tilde{s}^* \leftarrow \texttt{ASig.Agg}(s^*, \texttt{Inv}(\sigma_m))$ (where `Inv` is the inverting function based on the mathematical structure of the underlying aggregate signature scheme).
4. $\mathcal{B}$ outputs $(u^*, \tilde{s}^*)$ as a successful forgery of the underlying aggregate signature scheme `ASig`.

Success Probability Analysis: We analyze the events that are needed for $\mathcal{B}$ to win the $A\text{-}EU\text{-}CMA$ experiment as defined in Definition 6 for `ASig` as follows.

– $\overline{Abort1}$: $\mathcal{B}$ does not fail in answering any of $\mathcal{A}$'s queries.
– $Forge$: $\mathcal{A}$ wins the $EU\text{-}CMA$ experiment for `FAAS`.

– $\overline{Abort2}$: $\mathcal{B}$ does not abort during the forgery of $\mathcal{B}$.
– Win: $\mathcal{B}$ wins the $A\text{-}EU\text{-}CMA$ experiment of $\texttt{Asig}$

$\mathcal{B}$ wins if all the events happens and therefore, the probability $Adv^{A\text{-}EU\text{-}CMA}_{\texttt{Asig},\mathcal{B}}(t', q'_S, q'_h)$ decomposes as:

$$\Pr[Win] = \Pr[\overline{Abort1}] \cdot \Pr[Forge|\overline{Abort1}] \cdot \Pr[\overline{Abort2}|\overline{Abort1} \wedge Forge]$$

1. $\overline{Abort1}$: $\mathcal{B}$ responds to each of $\mathcal{A}$'s sign queries by querying the $\texttt{SigA}_{sk'}(\cdot)$ as defined in Definition 6 twice. Therefore, $\mathcal{B}$ only aborts if it cannot receive a valid signature from the $\texttt{SigA}_{sk'}(\cdot)$ that only happens with a negligible probability, and therefore, $\Pr[\overline{Abort1}] = 1$.
2. $Forge$: $\mathcal{B}$ only aborts if adversary $\mathcal{A}$ aborts and since the simulation transcript is indistinguishable from that of the actual scheme (based on the discussion in the indistinguishability analysis), the probability that $\mathcal{B}$ does not abort and $\mathcal{A}$ wins the $EU\text{-}CMA$ experiment is $\Pr[Forge|\overline{Abort1}] = Adv^{EU\text{-}CMA}_{\texttt{FAAS},\mathcal{A}}(t, q_S, q_h)$.
3. $\overline{Abort2}$: As highlighted in the Step 1 in Forgery of $\mathcal{B}$, the probability that $(u^*, \tilde{s}^*)$ is not a valid message-signature pair is negligible. Moreover, the probability that $u^* \cap L_m \rightarrow u^*$ happens is only $\frac{1}{2^\kappa}$, since it requires breaking the underlying hash function. Therefore, after the successful forgery by $\mathcal{A}$, $\mathcal{B}$'s forgery will also be valid and non-trivial with an overwhelming probability and it can be concluded that $\Pr[\overline{Abort2}|\overline{Abort1} \wedge Forge] \approx 1$.
4. Win: $\mathcal{B}$ wins the $A\text{-}EU\text{-}CMA$ experiment of $\texttt{Asig}$ with probability denoted as $\Pr[Win] = Adv^{A\text{-}EU\text{-}CMA}_{\texttt{Asig},\mathcal{B}}(t', q'_S, q'_h)$. This event only happens when all the above events happen. This implies that the $EU\text{-}CMA$ advantage of $\texttt{FAAS}$ adversary is bounded by the $A\text{-}EU\text{-}CMA$ advantage of the underlying $\texttt{Asig}$ adversary.

Execution Time Analysis: $\mathcal{B}$'s running time is that of $\mathcal{A}$'s plus all the the time it takes to respond $\mathcal{A}$'s queries. Each $\mathcal{A}$'s query requires two random number generator calls (which its cost is denoted by $t_{\texttt{RNG}}$) and two $\texttt{SigA}_{sk'}(\cdot)$ calls. Each $\texttt{SigA}_{sk'}(\cdot)$ call corresponds to a $\texttt{ASig.Sig}(\cdot)$ and a $\texttt{ASig.Agg}(\cdot)$ calls, which cost $t_{\texttt{Sig}}$ and $t_{\texttt{Agg}}$, respectively. Therefore, $\mathcal{B}$'s running time is estimated as $t' = O(t) + 2q_S(t_{\texttt{RNG}} + t_{\texttt{Sig}} + t_{\texttt{Agg}})$.
Transcript Indistinguishability: $\mathcal{A}$'s view of the actual scheme is the public key $\overline{PK}$, the signatures $(\sigma_1, \ldots, \sigma_j)$ for $j \leq q_S$ and the output of the hash functions. $PK = PK'$ is an output of $\texttt{ASig.Kg}(1^\kappa)$, which is identical to the actual scheme. For the signatures $\{\sigma_j = (u_j, s_j)\}_{j=1}^{q_S}$, one can see that the distribution of u is identical to the actual scheme since it is the output of the $Agg(\cdot)$ algorithm on the values $\{u_i\}_{i=1}^{k}$ which have the same distribution as in the actual scheme, (due to the $RO(\cdot)$ calls). Moreover, the distribution of $\{s_i\}_{i=1}^{q_S}$ in the simulation is identical to those in the actual scheme, since they are all the output of the same $\texttt{ASig.Agg}(\cdot)$. Lastly, the output of the random oracles in the proof is simulated with the same domain for H_0, H_1 and H_2 as in the actual schemes.

Discussion: We note that for the instantiations in §4, the underlying scheme, C-RSA [31], is proven to be *k-element* secure as in Definition 7. As for the

FAAS-NTRU instantiation [25], because of the probabilistic nature of signature generation due to the sampling step $r \leftarrow \chi_{\tilde{\sigma}}^{N}$, the aggregation of each $\mathbf{v} = \mathbf{v_1} + (-1)^b\mathbf{ag}$ where $\mathbf{u_1} = \mathbf{pr} + \mathbf{u_p}$, and $\mathbf{v_1} = \mathbf{u_1 h} \mod q$ leads to the aggregation of $\mathbf{r}$ in the aggregate signature. The aggregated randomness contributes to the hardness of the signature extraction problem since to do so, one needs to first take out the aggregated randomness from the signature.

We also note that since we are only aggregating 64 signatures, which is much less than even the theoretical bound mentioned in [25], FAAS-NTRU is immune to attack on batch signatures proposed in [25].