HAETAE: Shorter Lattice-Based Fiat-Shamir Signatures

Jung Hee Cheon
Seoul National University & CryptoLab Inc.
jhcheon@snu.ac.kr

Hyeongmin Choe
Seoul National University
sixtail528@snu.ac.kr

Julien Devevey
École Normale Supérieure de Lyon
julien.devevey@ens-lyon.fr

Tim Güneysu
Ruhr University Bochum & DFKI
tim.gueneysu@rub.de

Dongyeon Hong
CryptoLab Inc.
decenthong93@cryptolab.co.kr

Markus Krausz
Ruhr University Bochum
markus.krausz@rub.de

Georg Land
Ruhr University Bochum
georg.land@rub.de

Marc Möller
Ruhr University Bochum
marc.moeller@rub.de

Damien Stehlé
CryptoLab Inc.
damien.stehle@cryptolab.co.kr

MinJune Yi
Seoul National University
yiminjune@snu.ac.kr

***Abstract*—We present HAETAE (Hyperball bimodAl modulE rejecTion signAture schemE), a new lattice-based signature scheme, which we submitted to the Korean Post-Quantum Cryptography Competition for standardization. Like the NIST-selected Dilithium signature scheme, HAETAE is based on the Fiat-Shamir with Aborts paradigm, but our design choices target an improved complexity/compactness compromise that is highly relevant for many space-limited application scenarios. We primarily focus on reducing signature and verification key sizes so that signatures fit into one TCP or UDP datagram while preserving a high level of security against a variety of attacks. As a result, our scheme has signature and verification key sizes up to 40% and 25% smaller, respectively, compared than Dilithium. Moreover, we describe how to efficiently protect HAETAE against implementation attacks such as side-channel analysis, making it an attractive candidate for use in IoT and other embedded systems.**

***Index Terms*—Signature, Fiat-Shamir, Lattice-based Cryptography, Bimodal Distribution**

1. Introduction

The rise of quantum computing has brought up – among others – the necessity of new, post-quantum digital signature schemes. In the standardization process of post-quantum cryptography by the American National Institute of Standards and Technology (NIST), the lattice-based schemes Falcon [1] and Dilithium [2] have already been announced as future standards. The critical challenge in developing lattice-based digital signatures lies in finding a balance between security and practicality: while developing secure schemes against a wide range of attacks is essential, it is also vital to ensure they are practical for real-world applications. This challenge becomes even more critical with the increasing prevalence of embedded devices and the Internet of Things (IoT). Both technologies have become ubiquitous, from home appliances to medical devices connected to the internet.

In particular, this leads to two practical requirements:

1) The verification key and signature sizes must be as small as possible since both are frequently transmitted. Specifically, it is helpful if the signature is small enough to be sent in only one UDP or TCP datagram, as this minimizes the need for packet fragmentation. The importance of the signature and verification key sizes for communication protocols has been highlighted already in multiple evaluations [3, 4, 5]. Paquin et al. [4] observe for TLS, that fragmentation over many packets has a significant performance impact for network links with non-ideal packet loss rates. Benchmarking DNSSEC [5] revealed, that the smaller signatures of Falcon lead to faster resolution times in comparison to Dilithium in most scenarios, although the signature computation and verification is much faster with Dilithium compared to Falcon.
2) The secret-dependent operations such as key generation and message signing must be easy to protect against implementation attacks. This is essential in embedded use cases like the IoT, where attackers have physical access and can measure power consumption or electromagnetic emanation [6], additionally to the timing behaviour [7], which is also exploitable from remote.

In this context, Falcon fulfills the first requirement very well, but efforts for making it satisfy the second requirement, namely Mitaka [8], were recently broken [9]. Dilithium, on the other hand, focuses on being easy to implement and protect against side-channel attacks. However, this comes at the sacrifice of larger signatures and verification keys, which for example do not allow a signature to fit in one UDP datagram. We summarize this discussion in Table 1 and compare the two with HAETAE.

Scheme	Lvl.	Sig.	vk	Const.-T.	Maskable
Falcon-512	1	666B	897B	✓ [10]	✗ [9]
Dilithium-2	2	2,420B	1,312B	✓ [2]	✓ [11]
HAETAE-120	2	1,463B	992B	✓	✓

TABLE 1: NIST security level, signature size, verification key size, and implementation security, with respect to constant-time and masking of selected signature schemes

Contribution. We present HAETAE[1], a lattice-based digital signature scheme that improves over Dilithium by up to 40 % smaller signature and key sizes while being similarly easy to protect against physical attacks. Its quantum security is based on the hardness of the module versions of the lattice problems LWE and SIS [12, 13], in the Quantum Random Oracle Model (QROM). The scheme design follows the "Fiat-Shamir with Aborts" paradigm [14, 15], which relies on rejection sampling: rejection sampling is used to transform a signature trial whose distribution depends on sensitive information, into a signature whose distribution can be publicly simulated.

HAETAE is in part inspired from Dilithium, a post-quantum "Fiat-Shamir with Aborts" signature scheme, notably concerning the use of the module LWE and SIS assumptions. HAETAE differs from Dilithium in two major aspects: (i) we use a bimodal distribution for the rejection sampling, like in the BLISS signature scheme [16], instead of a "unimodal" distribution like Dilithium, (ii) we sample from and reject to hyperball uniform distributions, instead of discrete hypercube uniform distributions. The design departs from BLISS, as HAETAE relies on module lattice problems while BLISS relies either on assumptions on unstructured lattices or the NTRU assumption [17]: this leads us to introduce a new key generation algorithm. A further difference is that BLISS involves on discrete Gaussian distributions whereas HAETAE considers hyperball uniform distributions as suggested in [18]. This choice allows for simple rejection sampling without transcendental function computation and tail-cutting, while retaining the signature compactness enabled by Gaussian distributions.

HAETAE benefits from several novel improvements in the key generation algorithm. We introduce a new rejection procedure in the key generation algorithm to minimize the magnitude of the secret key when multiplied by the challenge. This facilitates rejection sampling in the signing algorithm and leads to smaller signatures. The key generation rejection is also designed to be efficient and simple to implement. It significantly improves over a procedure with a similar objective in the key generation of BLISS. Furthermore, we introduce to the bimodal setting a verification key truncation with the same objective as Dilithium's. A direct adaptation would lead to large bounds for the verification algorithm and degraded security. Instead, we compensate for the verification key truncation by correcting the signing key accordingly. It increases the magnitude of the signing key, but by a much smaller amount than the naive approach.

1. The haetae is a mythical Korean lion-like creature with the innate ability to distinguish right from wrong.

Param. set	Lvl.	Sig.	vk	KeyGen	Sign	Verify
H-120/D-2	2	60%	76%	408%	548%	106%
H-180/D-3	3	71%	75%	383%	484%	123%
H-260/D-5	5	63%	80%	181%	363%	94%
F-512/H-120	1/2	46%	90%	3,885%	277%	27%
F-1024/H-260	5	44%	86%	9,110%	423%	25%

TABLE 2: Relative comparison between HAETAE (H), Dilithium (D) and Falcon (F). The percentages are the ratio of their sizes and the execution times. The execution time is measured as the median cycle counts among 1000 executions, obtained on one core of an Intel Core i7-10700k, with TurboBoost and hyperthreading disabled.

For the signing algorithm, we adapt Dilithium's signature compression so that it is compatible with our module lattices key generation algorithm, by taking into account the residues modulo 2. Further, we apply the signature encoding technique from [19] to hyperball uniform distributions. The main novelty in the signing algorithm is a detailed description of a fixed-point arithmetic algorithm for sampling uniformly in a hyperball, which was left open in [18]. The discretization leads to numerical errors: we bound them and bound their effect on the scheme security.

Implementation and performance. We propose three parameter sets with NIST security levels 2, 3 and 5. Each parameter set of HAETAE has 20-25% smaller verification key size and 30-40% shorter signatures than its counterpart in Dilithium. Based on our C implementation of HAETAE, the verification process is as fast as Dilithium's, while the resulting key generation and signing algorithm are up to six times slower than Dilithium's. However, we emphasize that our implementation of HAETAE is yet unoptimized but already portable and constant-time. In addition, up to 80% of the signing time is consumed by the hyperball sampling. Thus, any improvement to this sampling would contribute greatly to the efficiency of HAETAE, an independent speedup to further optimizations. Nonetheless, on our benchmark signing with HAETAE-120 is still 2-3 times faster than with Falcon-512 and signing with HAETAE-260 is 3-4 times faster than with Falcon-1024's, with emulated floating-point operations. We summarize the comparison results in Table 2.

Moreover, we observe that masking HAETAE against physical attacks is nearly as easy as masking Dilithium, based on the similarity of the scheme design and the use of fixed-point arithmetic. One of the conceptual differences between HAETAE and Falcon (and their variants) regarding physical attacks is that HAETAE only needs Gaussian samples for secret-independent centers and standard deviations.

Finally, we note that like other Fiat-Shamir signatures, such as Schnorr signatures [20], the randomized signing of HAETAE can take advantage of pre-computations. By sampling from the hyperball and pre-computing the message-independent components offline, the online signing phase of HAETAE is cut by factor five.

Our code is available on the team HAETAE website: https://kpqc.cryptolab.co.kr/.

2. Preliminaries

Before introducing the core concepts in Section 3 and the HAETAE scheme in Section 4, we start by defining notations used throughout this paper and recapitulate relevant fundamental work.

2.1. Notations

Matrices are denoted in bold font and upper case letters (e.g., $\mathbf{A}$), while vectors are denoted in bold font and lowercase letters (e.g., $\mathbf{y}$ or $\mathbf{z}_1$). The i-th component of a vector is denoted with subscript i (e.g., y_i for the i-th component of $\mathbf{y}$).

Every vector is a column vector. We denote concatenation between vectors by putting the rows below as $(\mathbf{u}, \mathbf{v})$ and the columns on the right as $(\mathbf{u}|\mathbf{v})$. We naturally extend the latter notation to concatenations between matrices and vectors (e.g., $(\mathbf{A}|\mathbf{b})$ or $(\mathbf{A}|\mathbf{B})$).

We define a polynomial ring $\mathcal{R} = \mathbb{Z}[x]/(x^n+1)$ where n is a power of 2 integer and for any positive integer q the quotient ring $\mathcal{R}_q = \mathbb{Z}[x]/(q, x^n+1) = \mathbb{Z}_q[x]/(x^n+1)$. We abuse notations and identify $\mathcal{R}_2$ with the set of elements in $\mathcal{R}$ with binary coefficients. We also define a polynomial ring over real numbers $\mathcal{R}_{\mathbb{R}} = \mathbb{R}[x]/(x^n+1)$. For an integer η, we let the set of polynomials of degree less than n with coefficients in $[-\eta, \eta] \cap \mathbb{Z}$ be denoted by S_η. Given $\mathbf{y} = (\sum_{0\le i<n} y_i\ x^i, \cdots, \sum_{0\le i<n} y_{nk-n+i}\ x^i)^\top \in \mathcal{R}^k$ (or $\mathcal{R}_{\mathbb{R}}^k$), we define its ℓ_2-norm as the ℓ_2-norm of the corresponding "flattened" vector $\|\mathbf{y}\|_2 = \|(y_0, \cdots, y_{nk-1})^\top\|_2$.

Let $\mathcal{B}_{\mathcal{R},m}(r, \mathbf{c}) = \{\mathbf{x} \in \mathcal{R}_{\mathbb{R}}^m | \|\mathbf{x}-\mathbf{c}\|_2 \le r\}$ denote the continuous hyperball with center $\mathbf{c} \in \mathcal{R}^m$ and radius $r > 0$ in dimension $m > 0$. When $\mathbf{c} = \mathbf{0}$, we omit the center. Let $\mathcal{B}_{(1/N)\mathcal{R},m}(r, \mathbf{c}) = (1/N)\mathcal{R}^m \cap \mathcal{B}_{\mathcal{R},m}(r, \mathbf{c})$ denote the discretized hyperball with radius $r > 0$ and center $\mathbf{c} \in \mathcal{R}^m$ in dimension $m > 0$ with respect to a positive integer N. When $\mathbf{c} = \mathbf{0}$, we omit it. Given a measurable set $X \subseteq \mathcal{R}^m$ of finite volume, we let $U(X)$ denote the continuous uniform distribution over X. It admits $\mathbf{x} \mapsto \chi_X(\mathbf{x})/\mathsf{Vol}(X)$ as a probability density, where χ_X is the indicator function of X and $\mathsf{Vol}(X)$ is the volume of the set X. For the normal distribution over $\mathbb{R}$ centered at μ with standard deviation σ, we use the notation $\mathcal{N}(\mu, \sigma)$.

For a positive integer α, we define $r \bmod^{\pm} \alpha$ as the unique integer r' in the range $[-\alpha/2, \alpha/2)$ satisfying the relation $r = r' \bmod \alpha$. We also define $r \bmod^{+} \alpha$ as the unique integer r' in the range $[0, \alpha)$ satisfying $r = r' \bmod \alpha$. We denote the least significant bit of an integer r with $\mathsf{LSB}(r)$. We naturally extend this to integer polynomials and vectors of integer polynomials, by applying it component-wise.

2.2. Signatures

We briefly recall the formalism of digital signatures.

Definition 1 (Digital Signature)**.** *A signature scheme is a tuple of* PPT *algorithms* (KeyGen, Sign, Verify) *with the following specifications:*

- $\mathsf{KeyGen} : 1^\lambda \to (\mathsf{vk}, \mathsf{sk})$ *outputs a verification key* vk *and a signing key* sk*;*
- $\mathsf{Sign} : (\mathsf{sk}, \mu) \to \sigma$ *takes as inputs a signing key* sk *and a message* μ *and outputs a signature* σ*;*
- $\mathsf{Verify} : (\mathsf{vk}, \mu, \sigma) \to b \in \{0,1\}$ *is a deterministic algorithm that takes as inputs a verification key* vk*, a message* μ*, and a signature* σ *and outputs a bit* $b \in \{0,1\}$*.*

Let $\gamma > 0$*. We say that it is* γ*-correct if for any pair* $(\mathsf{vk}, \mathsf{sk})$ *in the range of* KeyGen *and* μ*,*

$$\Pr[\mathsf{Verify}(\mathsf{vk}, \mu, \mathsf{Sign}(\mathsf{sk}, \mu)) = 1] \ge \gamma,$$

where the probability is taken over the random coins of the signing algorithm. We say that it is correct in the (Q)ROM if the above holds when the probability is also taken over the randomness of the random oracle modeling the hash function used in the scheme.

We also give two security notions, namely the existential unforgeability under chosen message attacks, and under no-message attacks.

Definition 2 (Security)**.** *Let* $T, \delta \ge 0$*. A signature scheme* $\mathsf{sig} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ *is said to be* (T, δ)-UF-CMA *secure in the QROM if for any quantum adversary* $\mathcal{A}$ *with runtime* $\le T$ *given (classical) access to the signing oracle and (quantum) access to a random oracle* H*, it holds that*

$$\Pr_{(\mathsf{vk},\mathsf{sk})}[\mathsf{Verify}(\mathsf{vk}, \mu^*, \sigma^*) = 1 | (\mu^*, \sigma^*) \leftarrow \mathcal{A}^{H, \mathsf{Sign}}(\mathsf{vk})] \le \delta,$$

where the randomness is taken over the random coins of $\mathcal{A}$ *and* $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$*. The adversary should also not have issued a sign query for* μ^**. The above probability of forging a signature is called the advantage of* $\mathcal{A}$ *and denoted by* $\mathsf{Adv}_{\mathsf{sig}}^{\mathsf{UF\text{-}CMA}}(\mathcal{A})$*. If* $\mathcal{A}$ *does not output anything, then it automatically fails.*

Existential unforgeability against no-message attack, denoted by UF-NMA *is defined similarly except that the adversary is not allowed to query any signature per message.*

2.3. Lattice assumptions

We first recall the well-known lattice assumptions MLWE and MSIS on algebraic lattices.

Definition 3 (Decision-$\mathsf{MLWE}_{n,q,k,\ell,\eta}$)**.** *For positive integers* q, k, ℓ, η *and the dimension* n *of* $\mathcal{R}$*, we say that the advantage of an adversary* $\mathcal{A}$ *solving the decision-*$\mathsf{MLWE}_{n,q,k,\ell,\eta}$ *problem is*

$$\mathsf{Adv}_{n,q,k,\ell,\eta}^{\mathsf{MLWE}}(\mathcal{A}) = \left| \begin{array}{l} \Pr\left[b = 1 \mid \mathbf{A} \leftarrow \mathcal{R}_q^{k\times\ell}; \mathbf{b} \leftarrow \mathcal{R}_q^k; b \leftarrow \mathcal{A}(\mathbf{A}, \mathbf{b})\right] \\ \quad - \Pr\left[b = 1 \;\middle|\; \begin{array}{c} \mathbf{A} \leftarrow \mathcal{R}_q^{k\times\ell}; (\mathbf{s}_1, \mathbf{s}_2) \leftarrow S_\eta^\ell \times S_\eta^k; \\ b \leftarrow \mathcal{A}(\mathbf{A}, \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2) \end{array}\right] \end{array} \right|.$$

Definition 4 (Search-$\mathsf{MSIS}_{n,q,k,\ell,\beta}$)**.** *For positive integers* q, k, ℓ*, a positive real number* β *and the dimension* n *of* $\mathcal{R}$*,*

we say that the advantage of an adversary $\mathcal{A}$ solving the search-$\mathsf{MSIS}_{n,q,k,\ell,\beta}$ *problem is*

$$\mathsf{Adv}^{\mathsf{MSIS}}_{n,q,k,\ell,\beta}(\mathcal{A}) = \Pr\left[\begin{array}{c} 0 < \|\mathbf{y}\|_2 < \beta \wedge \\ (\mathbf{A}|\ \mathbf{Id}_k)\cdot\mathbf{y} = 0 \mod q \end{array} \middle|\ \mathbf{A} \leftarrow \mathcal{R}_q^{k\times\ell}; \mathbf{y} \leftarrow \mathcal{A}(\mathbf{A})\right].$$

We finally introduce a variant of the SelfTargetMSIS problem introduced in Dilithium [2], which corresponds to our setting.

Definition 5 ($\mathsf{BimodalSelfTargetMSIS}_{H,n,q,k,\ell,\beta}$)**.** *Let $H : \{0,1\}^* \times \mathcal{M} \to \mathcal{R}_2$ be a cryptographic hash function. Let $q, k, \ell > 0$, $\beta \geq 0$ and the dimension n of $\mathcal{R}$. An adversary $\mathcal{A}$ solving the search-*$\mathsf{BimodalSelfTargetMSIS}_{H,n,q,k,\ell,\beta}$ *problem with respect to $\mathbf{j} \in \mathcal{R}_2^k \setminus \{0\}$ has advantage*

$$\mathsf{Adv}^{\mathsf{BimodalSelfTargetMSIS}}_{H,n,q,k,\ell,\beta}(\mathcal{A}) = \Pr\left[\begin{array}{c} 0 < \|\mathbf{y}\|_2 < \beta \wedge \\ H(\mathbf{Ay} - qc\mathbf{j} \mod 2q, M) = c \\ \hline (\mathbf{A}_0|\mathbf{b}) \leftarrow \mathcal{R}_q^{k\times\ell}; \\ \mathbf{A} = (2\mathbf{b} + q\mathbf{j}|\ 2\mathbf{A}_0|\ 2\mathbf{Id}_k) \mod 2q; \\ (\mathbf{y}, c, M) \leftarrow \mathcal{A}^{|H(\cdot)\rangle}(\mathbf{A}) \end{array}\right].$$

In the ROM (resp. QROM), the adversary is given classical (resp. quantum) access to H.

The following classical reduction from MSIS to BimodalSelfTargetMSIS is very similar to the reduction from MSIS to SelfTargetMSIS introduced in [2] and is similarly non-tight. Moreover, since the reduction relies on the forking lemma; it cannot be directly extended to a quantum reduction in the QROM.

Theorem 1 (Classical Reduction from MSIS to BimodalSelfTargetMSIS)**.** *Let $q > 0$ be an odd modulus, $H : \{0,1\}^* \times \mathcal{M} \to \mathcal{R}_2$ be a cryptographic hash function modeled as a random oracle and that every polynomial-time classical algorithm has a negligible advantage against* $\mathsf{MSIS}_{n,q,k,\ell,\beta}$*. Then every polynomial-time classical algorithm has negligible advantage against* $\mathsf{BimodalSelfTargetMSIS}_{n,q,k,\ell,\beta/2}$*.*

Proof sketch. Consider a $\mathsf{BimodalSelfTargetMSIS}_{n,q,k,\ell,\beta/2}$ classical algorithm $\mathcal{A}$ that is polynomial-time and has classical access to H. If $\mathcal{A}^{H(\cdot)}(\mathbf{A})$ makes Q hash queries $H(\mathbf{w}_i, M_i)$ for $i = 1, \cdots, Q$ and outputs a solution $(\mathbf{y}, c, M_j)$ for some $j \in [Q]$, then we can construct an adversary $\mathcal{A}'$ for $\mathsf{MSIS}_{n,q,k,\ell,\beta}$ as follows.

The adversary $\mathcal{A}'$ can first rewind $\mathcal{A}$ to the point at which the j-th query was made and reprogram the hash as $H(\mathbf{w}_j, M_j) = c' (\neq c)$. Then, with probability approximately $1/Q$, algorithm $\mathcal{A}$ will produce another solution $(\mathbf{y}', c', M_j)$. We then have

$$\begin{cases} \mathbf{Ay} - qc\mathbf{j} = \mathbf{z}_j = \mathbf{Ay}' - qc'\mathbf{j} \mod 2q, \\ \|\mathbf{y}\|_2, \|\mathbf{y}'\|_2 < \beta/2. \end{cases}$$

As q is odd, we have $\mathbf{A}(\mathbf{y} - \mathbf{y}') = (c - c')\mathbf{j} \mod 2$. The fact that $c' \neq c$ implies that the latter is non-zero modulo 2, and hence so is $\mathbf{y} - \mathbf{y}'$ over the integers. As it also satisfies $(\mathbf{b}|\ \mathbf{A}_0|\ \mathbf{Id}_k) \cdot (\mathbf{y} - \mathbf{y}') = 0 \mod q$ and $\|\mathbf{y} - \mathbf{y}'\| < \beta$, it provides a $\mathsf{MSIS}_{n,q,k,\ell,\beta}$ solution for the matrix $(\mathbf{b}|\ \mathbf{A}_0|\ \mathbf{Id}_k)$, where the submatrix $(-\mathbf{b}|\ \mathbf{A}_0) \in \mathcal{R}_q^{k\times\ell}$ is uniform. □

2.4. Bimodal hyperball rejection sampling

Recently, Devevey et al. [18] conducted a study of rejection sampling in the context of lattice-based Fiat-Shamir with Aborts signatures. They observe that (continuous) uniform distributions over hyperballs can be used to obtain compact signatures, with a relatively simple rejection procedure. HAETAE uses (discretized) uniform distributions over hyperballs, in the bimodal context. The proof of the following lemma is available in Appendix A.

Lemma 1 (Bimodal Hyperball Rejection Sampling)**.** *Let n be the degree of $\mathcal{R}$, $c > 1$, $r, t, m > 0$, and $r' \geq \sqrt{r^2 + t^2}$. Define $M = 2(r'/r)^{mn}$ and set*

$$N \geq \frac{1}{c^{1/(mn)} - 1} \frac{\sqrt{mn}}{2} \left(\frac{c^{1/(mn)}}{r} + \frac{1}{r'}\right).$$

Let $\mathbf{v} \in \mathcal{R}^m \cap \mathcal{B}_{(1/N)\mathcal{R},m}(t)$. Let $p : \mathbb{R}^m \to \{0, 1/2, 1\}$ be defined as follows

$$p(\mathbf{z}) = \begin{cases} 0 & \text{if } \|\mathbf{z}\| \geq r, \\ 1/2 & \text{else if } \|\mathbf{z} - \mathbf{v}\| < r' \wedge \|\mathbf{z} + \mathbf{v}\| < r', \\ 1 & \text{otherwise.} \end{cases}$$

Then there exists $M' \leq cM$ such that the output distributions of the two algorithms from Figure 2 are identical.

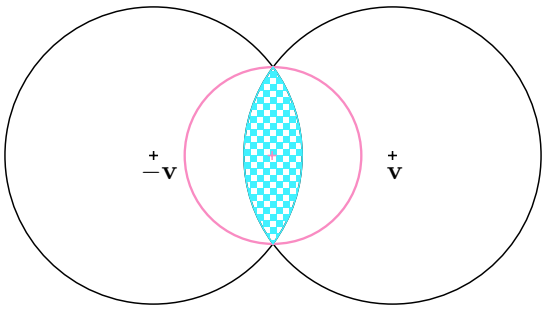


Figure 1: The HAETAE eyes

Figure 1 illustrates (the continuous version) of the rejection sampling that we consider. The black circles have radii equal to r' and the pink circle has radius r. We sample a vector $\mathbf{z}$ uniformly inside one of the black circles (with probability $1/2$ for each) and keep $\mathbf{z}$ with $p(\mathbf{z}) = 1/2$ if $\mathbf{z}$ lies in the blue zone, with probability $p(\mathbf{z}) = 1$ if it lies inside the pink circle but not in the blue zone, and with probability $p(\mathbf{z}) = 0$ everywhere else.

Figure 2 illustrates bimodal hyperball rejection sampling algorithm ($\mathcal{A}$) and its simulator ($\mathcal{B}$). As we do not know the exact value of M', we cannot use algorithm $\mathcal{B}$ as a signature simulator in the security proof of HAETAE. Note that in

$\mathcal{A}(\mathbf{v})$:	$\mathcal{B}$:
1: $\mathbf{y} \leftarrow U(\mathcal{B}_{(1/N)\mathcal{R},m}(r'))$	1: $\mathbf{z} \leftarrow U(\mathcal{B}_{(1/N)\mathcal{R},m}(r))$
2: $\mathbf{b} \leftarrow U(\{0,1\})$	2: **return** $\mathbf{z}$ with probability $1/M'$, else $\perp$
3: $\mathbf{z} \leftarrow \mathbf{y} + (-1)^b\mathbf{v}$	
4: **return** $\mathbf{z}$ with probability $p(\mathbf{z})$, else $\perp$	

Figure 2: Bimodal hyperball rejection sampling

the security proofs of lattice-based Fiat-Shamir with Aborts signatures, it is required to have an efficient simulator that simulates all iterations of the signature algorithm. Hence, simply replacing $\mathcal{B}$ with a version that always output $\mathbf{z}$ does not suffice. Our proposal is to use $\mathcal{A}(\mathbf{0})$ as an efficient simulator: as $\mathbf{0}$ has norm at most t for any $t > 0$, algorithm $\mathcal{A}(\mathbf{0})$ has statistical distance 0 with $\mathcal{B}$ and thus with $\mathcal{A}(\mathbf{v})$ for any $\mathbf{v}$ with norm $\leq t$.

2.5. Sampling from the continuous hyperball-uniform

In order to sample in practice from discrete hyperball uniform, we rely on the following result.

1: $y_i \leftarrow \mathcal{N}(0,1)$ for $i = 0, \cdots, nk+1$
2: $L \leftarrow \|(y_0, \cdots, y_{nk+1})^\top\|_2$
3: $\mathbf{y} \leftarrow r''/L \cdot (\sum_{i=0}^{n-1} y_i\ x^i, \cdots, \sum_{i=nk-n}^{nk-1} y_i\ x^i)^\top$
4: **return** $\mathbf{y}$ $\triangleright$ $\mathbf{y} \in \mathcal{R}_{\mathbb{R}}^k$

Figure 3: Hyperball uniform sampling

Lemma 2 ([21])**.** *The distribution of the output of the algorithm in Figure 3 is $U(\mathcal{B}_{\mathcal{R},k}(r''))$.*

Using Lemma 2, we can conclude that if we use the algorithms in Figures 1 to 3 and if we can sample from a normal distribution correctly, then the resulting distribution of $\mathbf{z}$ is indeed the uniform sample from the discretized hyperball.

We delay to Section 3.2 the analysis of a version which turns discrete Gaussian samples to a discretized version of the hyperball-uniform distribution.

2.6. High and low bits

A HAETAE signature is principally a vector $\mathbf{z}$, whose lower part is replaced with a (smaller) hint. HAETAE makes use of high and low bits decomposition when compressing a signature as well as when computing a hint. Furthermore, we use the decomposition for key truncation. For these ends, we use different decompositions.

We use the following base method of decomposing an element in high and low bits. We first recall the Euclidean division with a centered remainder.

Lemma 3. *Let $a \geq 0$ and $b > 0$. It holds that*

$$a = \left\lfloor \frac{a + b/2}{b} \right\rfloor \cdot b + (a \bmod^{\pm} b),$$

and this writing as $a = bq + r$ with $r \in [-b/2, b/2)$ is unique.

We define our decomposition for compressing the upper part of the signature.

Definition 6 (High and low bits)**.** *Let $r \in \mathbb{Z}$ and α be a power of two. Define $r_1 = \lfloor (r + \alpha/2)/\alpha \rfloor$ and $r_0 = r \bmod^{\pm} \alpha$. Finally, define the tuple:*

$$(\mathsf{LowBits}(r,\alpha), \mathsf{HighBits}(r,\alpha)) \ = \ (r_0, r_1).$$

We extend these definitions to vectors by applying them component-wise. We state that this decomposition lets us recover the original element and bound the components of the decomposition in Lemma 4. The proof is available in Appendix A.

Lemma 4. *Let α be a power of two. Let $q > 2$ be a prime with $\alpha | 2(q-1)$ and $r \in \mathbb{Z}$. Then it holds that*

$$\begin{aligned} &r = \alpha \cdot \mathsf{HighBits}(r,\alpha) + \mathsf{LowBits}(r,\alpha), \\ &\mathsf{LowBits}(r,\alpha) \in [-\alpha/2, \alpha/2), \\ &r \in [0, 2q-1] \implies \mathsf{HighBits}(r,\alpha) \in [0, (2q-1)/\alpha]\,. \end{aligned}$$

We define $\mathsf{HighBits}^{z1}(r) = \mathsf{HighBits}(r, 256)$, $\mathsf{LowBits}^{z1}(r) = \mathsf{LowBits}(r, 256)$ and $\mathsf{HighBits}^{\mathsf{vk}}(r) = \mathsf{HighBits}(r,d)$, $\mathsf{LowBits}^{\mathsf{vk}}(r) = \mathsf{LowBits}(r,d)$.

2.6.1. High and low bits for h. In order to produce the hint that we send instead of the lower part of $\mathbf{z}$, we could use the previous bit decomposition. However, as noted in [2, Appendix B] in a preliminary version, a slight modification allows to further reduce the entropy of the hint.

The idea is to pack the high bits in the range $[0, 2(q-1)/\alpha_{\mathsf{h}})$. This is possible if we use the range $[-\alpha_{\mathsf{h}}/2 - 2, 0)$ to represent the integers that are close to $2q-1$.

Definition 7 (High and low bits for h)**.** *Let $r \in \mathbb{Z}$. Let q be a prime and $\alpha_{\mathsf{h}} | 2(q-1)$ be a power of two. Let $m = 2(q-1)/\alpha_{\mathsf{h}}$, $r_1 = \mathsf{HighBits}(r \bmod^{+} 2q, \alpha_{\mathsf{h}})$, and $r_0 = \mathsf{LowBits}(r \bmod^{+} 2q, \alpha_{\mathsf{h}})$. If $r_1 = m$, let $(r_0', r_1') = (r_0 - 2, 0)$. Else, $(r_0', r_1') = (r_0, r_1)$. We define:*

$$(\mathsf{LowBits}^{\mathsf{h}}(r), \mathsf{HighBits}^{\mathsf{h}}(r)) \ = \ (r_0', r_1').$$

As before, we extend these definitions to vectors by applying them component-wise. We state that this decomposition lets us recover the original element and bound the decomposition components.

Lemma 5. *Let $r \in \mathbb{Z}$. Let q be a prime, $\alpha_{\mathsf{h}} | 2(q-1)$ be a power of two and define $m = 2(q-1)/\alpha_{\mathsf{h}}$. It holds that*

$$\begin{aligned} &r = \alpha_{\mathsf{h}} \cdot \mathsf{HighBits}^{\mathsf{h}}(r) + \mathsf{LowBits}^{\mathsf{h}}(r) \mod 2q, \\ &\mathsf{LowBits}^{\mathsf{h}}(r) \in [-\alpha/2 - 2, \alpha/2), \\ &\mathsf{HighBits}^{\mathsf{h}}(r) \in [0, m-1]\,. \end{aligned}$$

The proof of Lemma 5 is available in Appendix A.

SampleInBall(ρ, τ):
1: Initialize $\mathbf{c} = c_0 c_1 \dots c_{255} = 00 \dots 0$
2: **for** $i = 256 - \tau$ to 255 **do**
3: $\quad j \hookleftarrow \{0, \dots, i\}$
4: $\quad c_i = c_j$
5: $\quad c_j = 1$
6: Return $\mathbf{c}$

Figure 4: Challenge sampling algorithm

2.7. Challenge sampling

Our challenges are polynomials $c \in \mathcal{R}$ with binary coefficients and exactly τ of them are nonzero. The challenge space has size $\binom{n}{\tau}$. To sample such challenges we rely on the SampleInBall algorithm from Dilithium, which we recall in Fig. 4.

For HAETAE 260, however, we require 255 bits of entropy for the challenge, which cannot be reached with the current challenge sizes. To achieve it, we replace the challenge sampling with the following. Given a 256-bits hash $w_0 \dots w_{255}$ with Hamming weight w, do the following. If $w < 128$, return $\sum_{i=0}^{255} w_i x^i$. If $w = 128$, return $\sum_{i=0}^{255} w_i \otimes w_0 x^i$. Otherwise, return $\sum_{i=0}^{255} w_i \otimes 1 x^i$. Exactly half of all binary polynomials are reachable this way, which means that the challenge set has size 2^{255} as desired.

3. Building blocks

While our scheme is reminiscent of Dilithium, the bimodal setting hinders the adaptation of some of the building blocks. In this section, we describe the parts that are specific to HAETAE. First, the key generation algorithm departs from known key generation algorithms for BLISS, as we work in the module setting. Second, we study the precision needed for our fixed-point discrete hyperball sampler. Finally, we describe the signature encoding, based on range asymmetric numeral system (rANS).

3.1. Key generation

The bimodal rejection sampling relies on having a key pair $(\mathbf{A}, \mathbf{s}) \in \mathcal{R}_p^{k \times (k+\ell)} \times \mathcal{R}_p^{k+\ell}$ such that $\mathbf{As} = -\mathbf{As} \bmod p$. To generate such a pair, following [16], we choose $p = 2q$ and aim at $\mathbf{As} = q\mathbf{j} \bmod 2q$ for $\mathbf{j} = (1, 0, \dots, 0)^\top$.

3.1.1. Key generation and encoding. To build a key pair, we start from an MLWE sample $\mathbf{b} - \mathbf{a} = \mathbf{A}_0 \mathbf{s}_0 + \mathbf{e}_0 \bmod q$, where $\mathbf{A}_0 \hookleftarrow U(\mathcal{R}_q^{k \times (\ell-1)})$, $\mathbf{a} \hookleftarrow U(\mathcal{R}_q^k)$ and $(\mathbf{s}_0, \mathbf{e}_0) \hookleftarrow U(S_\eta^{\ell-1} \times S_\eta^k)$. For any $\mathbf{b} = \mathbf{b}_1 + \mathbf{b}_0$, we define $\mathbf{A} = (2(\mathbf{a} - \mathbf{b}_1) + q\mathbf{j} | 2\mathbf{A}_0 | 2\mathbf{I}_k)$ as well as $\mathbf{s}^\top = (1 | \mathbf{s}_0^\top | (\mathbf{e}_0 - \mathbf{b}_0)^\top)$. One sees that $\mathbf{As} = q\mathbf{j} \bmod 2q$. In practice, the verification key is then comprised of $\mathbf{b}_1$ and the seed that allows generating $\mathbf{A}_0$ and $\mathbf{a}$. The secret key is the seed used to generate $\mathbf{s}$ and $(\mathbf{A}_0, \mathbf{a})$.

It remains to choose the decomposition of $\mathbf{b}$, that we see as an nk-dimensional vector with coordinates in $[0, q-1]$. We choose $\mathbf{b}_0$ with coordinates in $\{-1, 0, 1\}$ such that if a coordinate of $\mathbf{b}$ is odd, then it is rounded to the nearest multiple of 4. We can then write $\mathbf{b} = \mathbf{b}_0 + 2\mathbf{b}_1$, where $\mathbf{b}_1$ is encoded using $\lceil \log_2(q) - 1 \rceil$ bits per coordinate. This is computed coordinate-wise with $\mathbf{b}_0 = (-1)^{\lfloor \mathbf{b}/2 \rfloor \bmod 2} \mathbf{b} \bmod 2$, i.e. one less bit than $\mathbf{b}$. In all of the following, we let $(\mathsf{LowBits}^{\mathsf{vk}}(\mathbf{b}), \mathsf{HighBits}^{\mathsf{vk}}(\mathbf{b}))$ denote $(\mathbf{b}_0, \mathbf{b}_1)$. When $\mathbf{b}$ is uniform, we notice that the coordinates of $\mathbf{b}_0$ roughly follow a (centered) binomial law with parameters $(2, 1/2)$, which experimentally leads to smaller choices for β, which we discuss and introduce now.

3.1.2. Rejection sampling on the key. A critical step of our scheme is bounding $\|\mathbf{s}c\|_2$, where $\mathbf{s}$ is generated as before and $c \in \mathcal{R}$ is a polynomial with coefficients in $\{0, 1\}$ and has less than or equal to τ nonzero coefficients. The lower this bound is, the smaller the signature is, which in turn leads to harder forging. In the key generation algorithm, we apply the following rejection condition for some heuristic value β:

$$\tau \cdot \sum_{i=1}^{m} \max_j{}^{i\text{-th}} \|\mathbf{s}(\omega_j)\|_2^2 + r \cdot \max_j{}^{(m+1)\text{-th}} \|\mathbf{s}(\omega_j)\|_2^2 \leq \frac{n\beta^2}{\tau},$$

where $m = \lfloor n/\tau \rfloor$ and $r = n \bmod \tau$. We argue that the left hand side is a bound on $\frac{n}{\tau} \cdot \|\mathbf{s}c\|_2^2$ and that this condition leads to asserting $\|\mathbf{s}c\|_2 \leq \beta$.

Lemma 6. *For a binary challenge $c \in \{0, 1\}^n$ with hamming weight τ and a secret $\mathbf{s} \in S_\eta^{k+\ell}$, $n\|c\mathbf{s}\|_2^2$ is bounded by*

$$\tau^2 \cdot \sum_{i=1}^{m} \max_j{}^{i\text{-th}} \|\mathbf{s}(\omega_j)\|_2^2 + r \cdot \tau \cdot \max_j{}^{(m+1)\text{-th}} \|\mathbf{s}(\omega_j)\|_2^2,$$

where $m = \lfloor n/\tau \rfloor$ and $r = n \bmod \tau$.

Proof. We first rewrite $\|\mathbf{s}c\|^2$ as:

$$\|\mathbf{s}c\|_2^2 = \frac{\sum_i |c(\omega_j)|^2 \cdot \|\mathbf{s}(\omega_j)\|_2^2}{n},$$

where $\mathbf{s}(\omega_j) = (\mathbf{s}_1(\omega_j), \cdots, \mathbf{s}_{k+\ell}(\omega_j))$, and ω_j's are the primitive $2n$-th roots of unity. For $n = m \cdot \tau + r$, let $m = \lfloor n/\tau \rfloor$ and $r = n \bmod \tau$. Since $\sum_{j=1}^n |c(\omega_j)|^2 = n\tau$ and

$$|c(\omega_j)|^2 = |\ \omega_{j,1} + \dots + \omega_{j,\tau}|^2 \leq \tau^2,$$

we can bound $\sum_{j=1}^n |c(\omega_j)|^2 \cdot \|\mathbf{s}(\omega_j)\|_2^2$ by rearrangement: let $m = \lfloor n/\tau \rfloor$ be the maximum number of $|c(\omega_j)|^2$'s that can be τ^2. By sorting $\|\mathbf{s}(\omega_j)\|_2$ in a decreasing order,

$$\|\mathbf{s}(\omega_{\sigma(1)})\|_2 \geq \|\mathbf{s}(\omega_{\sigma(2)})\|_2 \geq \dots \geq \|\mathbf{s}(\omega_{\sigma(n)})\|_2,$$

where σ is a permutation for the indices, we have

$$\sum_{j=1}^{n} |c(\omega_j)|^2 \cdot \|\mathbf{s}(\omega_j)\|_2^2 \leq \sum_{j=1}^{m} |c(\omega_{\sigma(j)})|^2 \cdot \|\mathbf{s}(\omega_{\sigma(j)})\|_2^2 + \sum_{j=m+1}^{n} |c(\omega_{\sigma(j)})|^2 \cdot \|\mathbf{s}(\omega_{\sigma(m+1)})\|_2^2.$$

Then it reaches the maximum when the m largest $\|\mathbf{s}(\omega_j)\|_2^2$'s are multiplied with the m number of τ^2's. That is,

$$\sum_{j=1}^{n} |c(\omega_j)|^2 \cdot \|\mathbf{s}(\omega_j)\|_2^2 \leq \sum_{j=1}^{m} \tau^2 \cdot \|\mathbf{s}(\omega_{\sigma(j)})\|_2^2 + \Big(\sum_{j=1}^{n} |c(\omega_j)|^2 - m\tau^2\Big) \cdot \|\mathbf{s}(\omega_{\sigma(j)})\|_2^2 = \tau^2 \cdot \sum_{j=1}^{m} \|\mathbf{s}(\omega_{\sigma(j)})\|_2^2 + r \cdot \tau \cdot \|\mathbf{s}(\omega_{\sigma(j)})\|_2^2.$$

□

3.2. Sampling in a discrete hyperball

Our approach in sampling is to avoid the use of floating point arithmetic for two reasons: First, many microarchitectures do not provide floating-point units and even if so, the execution time of floating-point instructions may be data-dependent and thus unsuitable [22] for a constant-time implementation. Floating-point computation would also prohibit a masked implementation, that is protected against power side-channel attacks, because known masking techniques are only applicable to integers. And second, the required precision is higher than achievable even in IEEE double. In order to do so, we replace the continuous Gaussian sampler from Lemma 2 and instead use discrete Gaussian distributions, as we know that they approximate well continuous Gaussian distribution for large standard deviation.

3.2.1. Discrete Gaussian sampling. As we will lose precision when computing the inverse square root of a Gaussian sample, we require a Gaussian sampler with high fix-point precision. This is achieved by sampling over $\mathbb{Z}$ with a large standard deviation and then scaling the resulting sample to our convenience. We use [23, Algorithm 12] to sample from a discrete Gaussian distribution with $\sigma = 2^{76}, k = 2^{72}$. In essence, we start by sampling a discrete Gaussian x with $\sigma = 16$ using a cumulative distribution table and a uniform $y \in \{0, \ldots, 2^{72} - 1\}$ and set the Gaussian sample candidate as $x2^{72} + y$. Subsequently, this candidate is accepted with probability $\exp(-y(y + x2^{73})/2^{153})$. The resulting value is (implicitly) scaled by the factor 2^{-76} to obtain a continous sample from the standard normal distribution.

Approximating the exponential For this, we need to approximate the exponential function e^{-x} by a polynomial $f(x)$ on the closed interval $[c - \frac{w}{2}, c + \frac{w}{2}]$, with center c and width w. We obtain $f(x)$ by truncating the expansion of e^{-x} into a series of Chebyshev polynomials of the first kind $T_n(x)$ with linearly transformed input, as this is known to minimize the absolute approximation error for a given polynomial order. We find:

$$e^{-x} = -e^{-c} + 2e^{-c} \sum_{n=0}^{\infty} (-1)^n I_n\left(\tfrac{w}{2}\right) T_n\left(\tfrac{x-c}{w/2}\right)$$
$$x \in [c - \tfrac{w}{2}, c + \tfrac{w}{2}]$$

where $I_n(z)$ are modified Bessel functions of the first kind, which rapidly converge to zero for growing n. We recall $\|T_n(x)\| \leq 1$ for $\|x\| \leq 1$. For intervals $[0, w]$ with not too large widths we find $2e^{-c} I_{m+1}(\frac{w}{2})$ to be a useful estimate of the maximum absolute error, when truncating the series at order $m > 1$. This relation allows us to directly choose m according to the interval to cover and the maximum permissible error. The number of fraction bits is chosen to match this error. The numerical evaluation is performed in fixed-point arithmetic using the Horner scheme and multiplying with shifts to retain significant bits. When shifting right, we round half up, which retains about one additional bit of accuracy when compared to truncation.

Barthe et al. [24] introduced the GALACTICS toolbox to derive suitable polynomials approximating e^{-x}. They numerically evaluate and modify trial polynomials, minimizing the *relative* error, until an acceptable level is reached. The polynomials are evaluated using a Horner scheme, similar to this work, but rely on truncation. When comparing to polynomials derived using the GALACTICS toolbox, our approximation has a slightly smaller absolute error for intervals of interest in this work, while maintaing the same polynomial order and constant time properties. This holds true even when introducing rounding to the GALACTICS evalution of polynomials. Moreover, our approach is somewhat less heuristic than the GALACTICS method.

Approximating the inverse square root In order to turn the vector of standard normal distributed variates into a hyperball sample candidate, we must compute its norm. For this, we accumulate all squared samples and approximate its inverse square root. The approximation result is then multiplied by the constant $r' + \sqrt{nm}/(2N)$, which yields the scaling factor that is multiplied to each Gaussian sample. For the inverse square root, we deploy Newton's method, which is a well-known technique for that purpose. However, Newton's method requires a starting approximation that is, with each iteration, turned into a better approximation. Fortunately, we know that the sum of $nm + 2$ independent squared standard normal variables follows a χ^2 distribution with expected value $nm+2$. Hence, the starting approximation can be fixed and precomputed as $1/\sqrt{nm + 2}$. The number of iterations for a targeted precision can be determined experimentally. Therefore, we performed the approximation for the first input values that have negligible probabilities either for the cumulative distribution function of $\chi^2(nm+2)$ or its survival function, and checked how many iterations are required to still reach reasonable precision.

3.2.2. Discretizing the output. Once we obtain an "hyperball" sample, we choose to round it. Then, if the resulting sample lies too close to the border of the hyperball, we reject it. This ensures that for any possible sample, they have the same amount of pre-rounding predecessors. This also decreases the precision but the output is now discrete in a hyperball with a somewhat-smaller radius. We simply increase the starting radius to compensate.

$\mathbf{y} \leftarrow U(\mathcal{B}_{(1/N)\mathcal{R},m}(r'))$:

1: $\mathbf{y} \leftarrow U(\mathcal{B}_{\mathcal{R},m}(Nr' + \sqrt{mn}/2))$
2: **if** $\|\lfloor \mathbf{y} \rceil\|_2 \leq Nr'$ **then**
3: return $\lfloor \mathbf{y} \rceil / N$
4: **else**, restart

Figure 5: Discrete hyperball uniform sampling

We study in the following lemma the rejection probability of this step.

Lemma 7. *Let n be the degree of $\mathcal{R}$, $M_0 \geq 1$, $r', m > 0$ and set*

$$N \geq \frac{\sqrt{mn}}{2r'} \cdot \frac{M_0^{1/(mn)} + 1}{M_0^{1/(mn)} - 1}.$$

At each iteration, the algorithm from Figure 5 succeeds with probability $\geq 1/M_0$. Moreover, the distribution of the output is $U(\mathcal{B}_{(1/N)\mathcal{R},m}(r'))$.

The proof of this lemma can be found in Appendix A.

3.3. Signature encoding via range asymmetric numeral system

To encode a signature, we will split some of its components into low and high bits. If we correctly choose the number of low bits, they will be distributed almost uniformly. The high bits on the other hand, will then follow a distribution with a very small variance and can be considerably compressed with a suitable encoding. While Huffman coding would be applied on each coordinate at a time, an arithmetic coding encodes the entire coordinates in a single number. In contrast to Huffman coding, arithmetic coding gets close to entropy also for alphabets, where the probabilities of the symbols are not powers of two. We recall a recent type of entropy coding, named range Asymmetric Numeral systems (rANS) [25], that encodes the state in a natural number and thus allows faster implementations. As a stream variant, rANS can be implemented with finite precision integer arithmetic by using renormalization.

Definition 8 (Range Asymmetric Numeral System (rANS) Coding)**.** *Let $n > 0$ and $S \subseteq [0, 2^n - 1]$. Let $g : [0, 2^n - 1] \to \mathbb{Z} \cap (0, 2^n]$ such that $\sum_{x \in S} g(x) \leq 2^n$ and $g(x) = 0$ for all $x \notin S$. We define the following:*

- $\mathsf{CDF} : S \to \mathbb{Z}$*, defined as* $\mathsf{CDF}(s) = \sum_{y=0}^{s-1} g(y)$.
- $\mathsf{symbol} : \mathbb{Z} \to S$*, where* $\mathsf{symbol}(y)$ *is defined as* $s \in S$ *satisfying* $\mathsf{CDF}(s) \leq y < \mathsf{CDF}(s+1)$.
- $C : \mathbb{Z} \times S \to \mathbb{Z}$*, defined as*

$$C(x, s) = \left\lfloor \frac{x}{g(s)} \right\rfloor \cdot 2^n + (x \bmod^+ g(s)) + \mathsf{CDF}(s).$$

Then, we define the rANS encoding/decoding for the set S and frequency $g/2^n$ as in Figure 6.

Lemma 8 (Adapted from [25])**.** *The rANS coding is correct, and the size of the rANS code is asymptotically equal to Shannon entropy of the symbols. That is, for any choice of $\mathbf{s} = (s_1, \cdots, s_m) \in S^m$, $\mathsf{Decode}(\mathsf{Encode}(\mathbf{s})) = \mathbf{s}$. Moreover, for any positive x and any probability distribution p over S, it holds that*

$$\sum_{s \in S} p(s) \log(C(x, s)) \leq \log(x) + \sum_{s \in S} p(s) \log\left(\frac{g(s)}{2^n}\right) + \frac{2^n}{x}.$$

Finally, the cost of encoding the first symbol is $\leq n$, i.e., for any $x \in S$, we have $\log(C(0, s)) \leq n$.

$\mathsf{Encode}((s_1, \cdots, s_m) \in S^m)$:

1: $x_0 = 0$
2: **for** $i = 0, \cdots, m-1$ **do**
3: $x_{i+1} = C(x_i, s_{i+1})$
4: **return** x_m

$\mathsf{Decode}(x \in \mathbb{Z})$:

1: $y_0 = x$
2: $i = 0$
3: **while** $y_i > 0$ **do**
4: $t_{i+1} = \mathsf{symbol}(y_i \bmod^+ 2^n)$
5: $y_{i+1} = \lfloor y_i / 2^n \rfloor \cdot g(t_{i+1}) + (y_i \bmod^+ 2^n) - \mathsf{CDF}(t_{i+1})$
6: i++
7: $m = i - 1$
8: **return** $(t_m, \cdots, t_1) \in S^m$

Figure 6: rANS encoding and decoding procedures

We determine the frequency of the symbols experimentally, by executing the signature computation and collecting several million samples. Finally, we apply some rounding strategy to compute g such that the average overcost per coordinate caused by this rounding is almost negligible.

4. The HAETAE signature scheme

In this section, we describe three different versions of HAETAE. As a warm-up, we give an uncompressed, untruncated version of HAETAE, implementing the Fiat-Shamir with aborts paradigm in the bimodal hyperball-uniform setting. We then give the full description of optimized and deterministic HAETAE as we implemented it. Finally, we discuss the parts of the signing algorithm which can be pre-computed.

4.1. Uncompressed description

As a first approach, we give a high-level, uncompressed, description of our signature scheme in Figure 7. In all of the following sections, we let $\mathbf{j} = (1, 0, \ldots, 0) \in \mathcal{R}^k$, as well as k, ℓ be two dimensions, $N > 0$ the fix-point precision and $\tau > 0$ the challenge min-entropy parameter. The parameters B, B', and B'' refer to the radii of hyperballs.

Let q be an odd prime and $\alpha_{\mathsf{h}}|2(q-1)$ is a power of two. We define the key rejection function:

$$f : \mathbf{s} \mapsto \tau \cdot \sum_{i=1}^{m} \max_{j}^{i\text{-th}} \|\mathbf{s}(\omega_j)\|_2^2 + r \cdot \max_{j}^{(m+1)\text{-th}} \|\mathbf{s}(\omega_j)\|_2^2,$$

and the parameter β is the maximum allowed value for $\frac{\tau}{n}\sqrt{f(\mathbf{s})}$, which ensures that $\|\mathbf{s}c\|_2 \leq \beta$ for all $c \in \mathcal{R}_2$ satisfying $\|c\| \leq \sqrt{\tau}$. The key generation algorithm is a simplified version from Section 3.1, which removes the verification key compression, for conceptual simplicity.

$\underline{\mathsf{KeyGen}(1^\lambda)}$:

1: $(\mathbf{A}_{\mathsf{gen}}) \leftarrow \mathcal{R}_q^{k\times(\ell-1)}$ and $(\mathbf{s}_{\mathsf{gen}}, \mathbf{e}_{\mathsf{gen}}) \leftarrow S_\eta^{\ell-1} \times S_\eta^k$
2: $\mathbf{b} = \mathbf{A}_{\mathsf{gen}} \cdot \mathbf{s}_{\mathsf{gen}} + \mathbf{e}_{\mathsf{gen}} \in \mathcal{R}_q^k$
3: $\mathbf{A} = (-2\mathbf{b} + q\mathbf{j}|\ 2\mathbf{A}_{\mathsf{gen}}|\ 2\mathbf{Id}_k) \bmod 2q$
4: $\mathbf{s} = (1, \mathbf{s}_{\mathsf{gen}}, \mathbf{e}_{\mathsf{gen}})$
5: **if** $f(\mathbf{s}) > n\beta^2/\tau$ **then** restart
6: **return** $\mathsf{sk} = (\mathbf{A}, \mathbf{s})$, $\mathsf{vk} = \mathbf{A}$

$\underline{\mathsf{Sign}(\mathsf{sk}, M)}$:

1: $\mathbf{y} \leftarrow U(\mathcal{B}_{(1/N)\mathcal{R},(k+\ell)}(B))$
2: $\mathbf{w} \leftarrow \mathbf{A}\lfloor\mathbf{y}\rceil$
3: $c = H(\mathbf{w}, M) \in \mathcal{R}_2$
4: $\mathbf{z} = (\mathbf{z}_1, \mathbf{z}_2) = \mathbf{y} + (-1)^b c \cdot \mathbf{s}$ for $b \leftarrow U(\{0,1\})$
5: **if** $\|\mathbf{z}\|_2 \geq B'$ **then** restart
6: **else if** $\|2\mathbf{z} - \mathbf{y}\|_2 < B$ **then** restart with prob. $1/2$
7: **return** $\sigma = (\lfloor\mathbf{z}\rceil, c)$

$\underline{\mathsf{Verify}(\mathsf{vk}, M, \sigma = (\mathbf{z}, c))}$:

1: $\tilde{\mathbf{w}} = \mathbf{A}\mathbf{z} - qc\mathbf{j} \bmod 2q$
2: **return** $(c = H(\tilde{\mathbf{w}}, M)) \wedge \left(\|\mathbf{z}\| < B + \frac{\sqrt{n(k+\ell)}}{2}\right)$

Figure 7: Uncompressed description of HAETAE

4.2. Specification of HAETAE

We now give the full description of the signature scheme HAETAE in Figure 8 with the following building blocks:

- Hash function H_{gen} for generating the seeds and hashing the messages,
- Hash function H for signing, returning a seed ρ for sampling a challenge,
- Extendable output function expandA for deriving $\mathbf{A}_{\mathsf{gen}}$ from $\mathsf{seed}_{\mathbf{A}}$,
- Extendable output function expandS for deriving $(\mathbf{s}_{\mathsf{gen}}, \mathbf{e}_{\mathsf{gen}}) \in S_\eta^{\ell-1} \times S_\eta^k$ from $\mathsf{seed}_{\mathsf{sk}}$ and $\mathsf{counter}_{\mathsf{sk}}$,
- Extendable output function expandYbb for deriving $\mathbf{y}$, b and b' from seed_{ybb} and counter,

The above building blocks can be implemented with symmetric primitives.

Note that at Step 3 of the Verify algorithm, the division by 2 is well-defined as the operand is even and defined modulo $2q$.

$\underline{\mathsf{KeyGen}(1^\lambda)}$:

1: $\mathsf{seed} \leftarrow \{0,1\}^\rho$
2: $(\mathsf{seed}_{\mathbf{A}}, \mathsf{seed}_{\mathsf{sk}}, K) = H_{\mathsf{gen}}(\mathsf{seed})$
3: $(\mathbf{a}|\ \mathbf{A}_{\mathsf{gen}}) \coloneqq \mathsf{expandA}(\mathsf{seed}_{\mathbf{A}}) \in \mathcal{R}_q^{k\times\ell}$
4: $\mathsf{counter}_{\mathsf{sk}} = 0$
5: $(\mathbf{s}_{\mathsf{gen}}, \mathbf{e}_{\mathsf{gen}}) \coloneqq \mathsf{expandS}(\mathsf{seed}_{\mathsf{sk}}, \mathsf{counter}_{\mathsf{sk}})$
6: $\mathbf{b} = \mathbf{a} + \mathbf{A}_{\mathsf{gen}} \cdot \mathbf{s}_{\mathsf{gen}} + \mathbf{e}_{\mathsf{gen}} \in \mathcal{R}_q^k$
7: $(\mathbf{b}_0, \mathbf{b}_1) = (\mathsf{LowBits}^{\mathsf{vk}}(\mathbf{b}), \mathsf{HighBits}^{\mathsf{vk}}(\mathbf{b}))$
8: $\mathbf{A} = (2(\mathbf{a} - 2\mathbf{b}_1) + q\mathbf{j}|\ 2\mathbf{A}_{\mathsf{gen}}|\ 2\mathbf{Id}_k) \bmod 2q$
9: $\mathbf{s} = (1, \mathbf{s}_{\mathsf{gen}}, \mathbf{e}_{\mathsf{gen}} - \mathbf{b}_0)$
10: **if** $f(\mathbf{s}) > n\beta^2/\tau$ **then** $\mathsf{counter}_{\mathsf{sk}}$++ and **Go to 5**
11: **return** $\mathsf{sk} = \mathbf{s}$, $\mathsf{vk} = (\mathsf{seed}_{\mathbf{A}}, \mathbf{b}_1)$

$\underline{\mathsf{Sign}(\mathsf{sk}, M)}$:

1: $\mu = H_{\mathsf{gen}}(\mathsf{seed}_{\mathbf{A}}, \mathbf{b}_1, M)$
2: $\mathsf{seed}_{ybb} = H_{\mathsf{gen}}(K, \mu)$
3: $\mathsf{counter} = 0$
4: $(\mathbf{y}, b, b') \coloneqq \mathsf{expandYbb}(\mathsf{seed}_{ybb}, \mathsf{counter})$
5: $\mathbf{w} \leftarrow \mathbf{A}\lfloor\mathbf{y}\rceil$
6: $\rho = H(\mathsf{HighBits}^{\mathsf{h}}(\mathbf{w}), \mathsf{LSB}(\lfloor y_0 \rceil), \mu)$
7: $c = \mathsf{SampleInBall}(\rho, \tau)$
8: $\mathbf{z} = (\mathbf{z}_1, \mathbf{z}_2) = \mathbf{y} + (-1)^b c \cdot \mathbf{s}$
9: $\mathbf{h} = \mathsf{HighBits}^{\mathsf{h}}(\mathbf{w}) - \mathsf{HighBits}^{\mathsf{h}}(\mathbf{w} - 2\lfloor\mathbf{z}_2\rceil) \bmod^+ \frac{2(q-1)}{\alpha_{\mathsf{h}}}$
10: **if** $\|\mathbf{z}\|_2 \geq B'$ **then**
11: counter++ and **Go to 4**
12: **else if** $\|2\mathbf{z} - \mathbf{y}\|_2 < B \ \wedge\ b' = 0$ **then**
13: counter++ and **Go to 4**
14: **else**
15: $x = \mathsf{Encode}(\mathsf{HighBits}^{z1}(\lfloor\mathbf{z}_1\rceil))$
16: $\mathbf{v} = \mathsf{LowBits}^{z1}(\lfloor\mathbf{z}_1\rceil)$
17: **return** $\sigma = (x, \mathbf{v}, \mathsf{Encode}(\mathbf{h}), c)$

$\underline{\mathsf{Verify}(\mathsf{vk}, M, \sigma = (x, \mathbf{v}, h, c))}$:

1: $\tilde{\mathbf{z}}_1 = \mathsf{Decode}(x) \cdot a + \mathbf{v}$ and $\tilde{\mathbf{h}} = \mathsf{Decode}(h)$
2: $(\mathbf{a}|\ \mathbf{A}_{\mathsf{gen}}) = \mathsf{expandA}(\mathsf{seed}_{\mathbf{A}})$
3: $\mathbf{A}_1 = (2(\mathbf{a} - 2\mathbf{b}_1) + q\mathbf{j}|\ 2\mathbf{A}_{\mathsf{gen}}) \bmod 2q$
4: $\mathbf{w}_1 = \tilde{\mathbf{h}} + \mathsf{HighBits}^{\mathsf{h}}(\mathbf{A}_1\tilde{\mathbf{z}}_1 - qc\mathbf{j}) \bmod^+ \frac{2(q-1)}{\alpha_{\mathsf{h}}}$
5: $w' = \mathsf{LSB}(\tilde{z}_0 - c)$
6: $\tilde{\mathbf{z}}_2 = (\mathbf{w}_1 \cdot \alpha_{\mathsf{h}} + w'\mathbf{j} - (\mathbf{A}_1\tilde{\mathbf{z}}_1 - qc\mathbf{j}))/2 \bmod^\pm q$
7: $\tilde{\mathbf{z}} = (\tilde{\mathbf{z}}_1, \tilde{\mathbf{z}}_2)$
8: $\tilde{\mu} = H_{\mathsf{gen}}(\mathsf{seed}_{\mathbf{A}}, \mathbf{b}_1, M)$
9: $\tilde{\rho} = H(\mathbf{w}_1, w', \tilde{\mu})$
10: **return** $(c = \mathsf{SampleInBall}(\tilde{\rho}, \tau)) \wedge (\|\tilde{\mathbf{z}}\| < B'')$

Figure 8: Full description of deterministic HAETAE

Lemma 9. *We borrow the notations from Figure 8. If we run* $\mathsf{Verify}(\mathsf{vk}, M, \sigma)$ *on the signature* σ *returned by* $\mathsf{Sign}(\mathsf{sk}, M)$ *for an arbitrary message* M *and an arbitrary key-pair* $(\mathsf{sk}, \mathsf{vk})$ *returned by* $\mathsf{KeyGen}(1^\lambda)$*, then the following relations hold:*

1) $\mathbf{w}_1 = \mathsf{HighBits}^{\mathsf{h}}(\mathbf{w})$,
2) $w'\mathbf{j} = \mathsf{LSB}(\lfloor y_0 \rceil) \cdot \mathbf{j} = \mathsf{LSB}(\mathbf{w}) = \mathsf{LSB}(\mathbf{w} - 2\lfloor\mathbf{z}_2\rceil)$.
3) $2\lfloor\mathbf{z}_2\rceil - 2\tilde{\mathbf{z}}_2 = \mathsf{LowBits}^{\mathsf{h}}(\mathbf{w}) - \mathsf{LSB}(\mathbf{w})$ *assuming it holds*

that $B' + \alpha_{\mathsf{h}}/4 + 1 \leq B'' < q/2$*,*

Proof. Let $m = 2(q-1)/\alpha_{\mathsf{h}}$. Let us prove the first statement. By definition of $\mathbf{h}$, it holds that $\mathbf{w}_1 = \mathsf{HighBits}^{\mathsf{h}}(\mathbf{w}) \mod m$. However, the latter part of the equality already lies in $[0, m-1]$ by Lemma 5. The first part lies in the same range as we reduce $\mod^+ m$. Hence, the equality stands over $\mathbb{Z}$ too.

We move on to the second statement. By considering only the first component of $\mathbf{z} = \mathbf{y} + (-1)^b c \cdot \mathbf{s}$, we obtain, modulo 2:

$$\tilde{z}_0 = \lfloor z_0 \rceil = \lfloor y_0 \rceil + (-1)^b c = \lfloor y_0 \rceil + c.$$

This yields the result. Moreover, considering everywhere a 2 appears in the definition of $\mathbf{A}$, we obtain that

$$\mathbf{w} = \mathbf{A}_1 \lfloor \mathbf{z}_1 \rceil - qc\mathbf{j} = (\lfloor z_0 \rceil - c)\mathbf{j} \bmod 2.$$

For the last statement, let us use the two preceding results. In particular, we note the identity

$$\mathbf{w}_1 \cdot \alpha_{\mathsf{h}} + w'\mathbf{j} = \mathbf{w} - \mathsf{LowBits}^{\mathsf{h}}(\mathbf{w}) + \mathsf{LSB}(\mathbf{w}).$$

We note that the last two elements have same parity, as the former one has the same parity as $\mathsf{LowBits}(\mathbf{w}, \alpha_{\mathsf{h}})$. By Lemma 5 their sum has infinite norm $\leq \alpha_{\mathsf{h}}/2 + 2$. Hence from its definition, it holds that

$$2\tilde{\mathbf{z}}_2 = 2\lfloor \mathbf{z}_2 \rceil - \mathsf{LowBits}^{\mathsf{h}}(\mathbf{w}) + \mathsf{LSB}(\mathbf{w}) \bmod^{\pm} 2q.$$

Finally, this identity holds over the integers as the right-hand side has infinite norm at most $2B' + \alpha_{\mathsf{h}}/2 + 2 < q$. □

Theorem 2 (Completeness)**.** *Assume that* $B'' = B' + \sqrt{n(k+\ell)}/2 + \sqrt{nk} \cdot (\alpha_{\mathsf{h}}/4 + 1) < q/2$*. Then the signature schemes of Figures 8 is complete, i.e., for every message* M *and every key-pair* $(\mathsf{sk}, \mathsf{vk})$ *returned by* $\mathsf{KeyGen}(1^\lambda)$*, we have:*

$$\mathsf{Verify}(\mathsf{vk}, M, \mathsf{Sign}(\mathsf{sk}, M)) = 1.$$

Proof. We use the notations of the algorithms. The first and second equations from Lemma 9 state that $\rho = \tilde{\rho}$ and thus

$$c = \mathsf{SampleInBall}(\tilde{\rho}, \tau).$$

On the other hand, we use the last equation from the same lemma to bound the size of $\tilde{\mathbf{z}}$. We have:

$$\begin{aligned}
\|\tilde{\mathbf{z}}\| &\leq \|\mathbf{z}\| + \|\mathbf{z} - \lfloor \mathbf{z} \rceil\| + \|\lfloor \mathbf{z} \rceil - \tilde{\mathbf{z}}\| \\
&\leq B' + \sqrt{n(k+\ell)} \cdot \|\mathbf{z} - \lfloor \mathbf{z} \rceil\|_\infty + \|\lfloor \mathbf{z}_2 \rceil - \tilde{\mathbf{z}}_2\| \\
&\leq B' + \frac{\sqrt{n(k+\ell)}}{2} + \sqrt{nk} \cdot \|\mathsf{LowBits}^{\mathsf{h}}(\mathbf{w})\|_\infty \\
&\leq B' + \frac{\sqrt{n(k+\ell)}}{2} + \sqrt{nk} \cdot \left(\frac{\alpha_{\mathsf{h}}}{4} + 1\right).
\end{aligned}$$

The definition of B'' implies that the scheme is correct. □

$\mathsf{Sim}(\mathbf{A}, c)$:
1: $\mathbf{y} \leftarrow U(\mathcal{B}_{(1/N)\mathcal{R},m}(r'))$
2: $\mathbf{w} \leftarrow (\mathsf{HighBits}^{\mathsf{h}}(\mathbf{A}\lfloor \mathbf{y} \rceil), \mathsf{LSB}(y_0))$
3: $\mathbf{b} \leftarrow U(\{0,1\})$
4: $\mathbf{z} \leftarrow \mathbf{y} + (-1)^b \mathbf{v}$
5: $\mathbf{u} \leftarrow U(\mathcal{R}_q^k)$
6: $u_0 \leftarrow U(\mathcal{R}_2)$
7: $\tilde{\mathbf{w}} \leftarrow (\mathsf{HighBits}^{\mathsf{h}}(2\mathbf{u} + q\mathbf{j}u_0), u_0)$
8: **return** $(\mathbf{w}, c, \mathbf{z})$ with probability $p(\mathbf{z})$, **else** $(\tilde{\mathbf{w}}, c, \perp)$

Figure 9: HAETAE simulator

4.3. Security proof

When proving security in the Fiat-Shamir with aborts setting in the QROM, on typically relies on the generic reduction from [26]. However, as pointed out in [27] and [28], this analysis is flawed. Both works give adaptations to Fiat-Shamir with aborts of the analysis from [29] of Fiat-Shamir (without aborts). Moreover, the reduction from [26] assumes an arbitrary bound on the number of restarts, which is not the case here. This restriction is waived in both [27] and [28].

Theorem 3. *Let* $B' \geq (k+\ell)n/(2e\sqrt{\pi})q^{k/(k+\ell)}$*. Then HAETAE as described in Figure 8 is* UF-CMA *secure in the QROM.*

Proof sketch. The proof relies on the analysis of [27], which reduces UF-CMA security to UF-NMA security, where an adversary is not allowed to make signing queries. This analysis requires that the commitment min-entropy is high and the underlying Σ-protocol is Honest-Verifier Zero-Knowledge (HVZK). The latter is proved by providing a simulator for non-aborting transcripts and proving that the distribution of $\lfloor \mathbf{y} \rceil$ has sufficiently large min-entropy.

Commitment min-entropy. We first claim that the underlying Σ-protocol has large commitment min-entropy. Indeed, $\mathsf{LSB}(y_0)$ is part of the initial commitment, and has min-entropy n.

HVZK. Next, we show that the underlying Σ-protocol satisfies the HVKZ property. To do so, we follow the strategy from [27, Section 4], which studies the simulation of non-aborting transcripts and applies the leftover hash lemma when simulating aborting transcripts. We propose the following simulator in Figure 9. On input a challenge c, it runs $\mathcal{A}(0)$, and if it fails, it samples a uniform commitment and no answer.

(i) Simulating non-aborting transcripts. When a sample is accepted, Lemma 1 states that the simulator follows exactly the same distribution as the real aglorithm.

(ii) High min-entropy for source distribution. When a sample is aborted, the distribution of $\lfloor \mathbf{y} \rceil$ has sufficiently high min-entropy to apply [27, Lemma 4] (we can first consider not applying high and least significant bits and adding them later for free thanks to the data processing inequality of the statistical distance). Indeed, the distribution

has min-entropy $\log((B'\sqrt{\pi})^{n(k+\ell)}/(n(k+\ell)/2)!)$ as n is even. Setting $B' \geq (k+\ell)n/(2e\sqrt{\pi})q^{k/(k+\ell)} \cdot 2\log(1/\epsilon)$ is then enoguh to adapt [27, Theorem 1] and show that the output of the simulator is whithin statistical distance ϵ to the distribution of a real transcript.

These two properties allow us to apply [27, Theorem 4] to reduce the SUF-CMA security to UF-NMA security.

Proving UF-NMA security. Finally, we note that the UF-NMA security game is exactly the problem defined in Definition 5, up to replacing the verification key by an uniform matrix (still in HNF form), which is done under the MLWE assumption. □

4.4. HAETAE with pre-computation

We observe that in the randomized signing process of HAETAE, many operations do not depend on the message M, and some do not even on the signing key. This enables efficient "offline" procedures, i.e., precomputations that speed up the "online" phase.

Specifically, there are two levels of offline signing that can be applied to randomized HAETAE:

1) **Generic.** If neither the message M nor the signing key is yet unchosen in advance, it is still possible to perform hyperball sampling. This removes the most time-consuming operation from the online phase.
2) **Designated signing key.** Here, only the message M is unknown during offline signing, while the signing key is fixed. This allows us to perform even more pre-computations by using only the verifiction key, as shown in Figure 10. Most notably, there is no longer a matrix-vector multiplication in the online phase.

We showcase the offline and online parts of the (randomized) version of HAETAE in Figure 10.

$\underline{\mathsf{Sign}^{\mathsf{offline}}(\mathsf{vk})}$:
1: $(\mathbf{a}|\ \mathbf{A}_{\mathsf{gen}}) = \mathsf{expandA}(\mathsf{seed}_{\mathbf{A}})$
2: $\mathbf{A}_1 = (2(\mathbf{a} - 2\mathbf{b}_1) + q\mathbf{j}|\ 2\mathbf{A}_{\mathsf{gen}}) \bmod 2q$
3: $\mathsf{List} = ()$
4: **for** iter in $[L]$ **do**
5: $\quad \mathbf{y} \leftarrow U(\mathcal{B}_{(1/N)\mathcal{R},(k+\ell)}(B))$
6: $\quad \mathbf{w} = \mathbf{A}\lfloor \mathbf{y} \rceil$
7: $\quad \mathbf{w}_1 = \mathsf{HighBits}^{\mathsf{h}}(\mathbf{w})$
8: $\quad \mathsf{List.append}(\mathbf{y}, \mathbf{w}, \mathbf{w}_1, \mathsf{LSB}(\lfloor y_0 \rceil))$
9: **return** List

$\underline{\mathsf{Sign}^{\mathsf{online}}(\mathsf{sk}, \mathsf{List}, M)}$:
1: $\mu = H_{\mathsf{gen}}(\mathsf{seed}_{\mathbf{A}}, \mathbf{b}_1, M)$
2: $\mathsf{tuple} = (\mathbf{y}, \mathbf{w}, \mathsf{tuple}_3, \mathsf{tuple}_4) \leftarrow \mathsf{List}$
3: $\mathsf{List.delete}(\mathsf{tuple})$
4: $c = \mathsf{SampleInBall}(H(\mathsf{tuple}_3, \mathsf{tuple}_4, \mu), \tau)$
5: $(b, b') \leftarrow \{0,1\}^2$
6: $\mathbf{z} = (\mathbf{z}_1, \mathbf{z}_2) = \mathbf{y} + (-1)^b c \cdot \mathbf{s}$
7: $\mathbf{h} = \mathsf{tuple}_3 - \mathsf{HighBits}^{\mathsf{h}}(\mathbf{w} - 2\lfloor \mathbf{z}_2 \rceil) \bmod^{+} \frac{2(q-1)}{\alpha_{\mathsf{h}}}$
8: **if** $\|\mathbf{z}\|_2 \geq B'$ **then Go to 2**
9: **else if** $\|2\mathbf{z} - \mathbf{y}\|_2 < B \ \wedge \ b' = 0$ **then Go to 2**
10: **else**
11: $\quad x = \mathsf{Encode}(\mathsf{HighBits}^{z1}(\lfloor \mathbf{z}_1 \rceil))$
12: $\quad \mathbf{v} = \mathsf{LowBits}^{z1}(\lfloor \mathbf{z}_1 \rceil)$
13: $\quad$ **return** $\sigma = (x, \mathbf{v}, \mathsf{Encode}(\mathbf{h}), c)$

Figure 10: Randomized, on/off-line signing. Note that app is the function that appends a tuple to the list.

5. Implementation

In this section, we give the implementation details of HAETAE. The constant-time reference implementation and the supporting scripts can be found on the team HAETAE website: https://kpqc.cryptolab.co.kr.

5.1. Parameter sets and signature sizes

We propose three different parameter sets with varying security levels, where we prioritize low signature and verification key sizes over faster execution time. The parameter choices are versatile, adaptable and allow size vs. speed trade-offs at consistent security levels. For example at cost of larger signatures, a smaller repetition rate M is possible and thus a faster signing process. This versatility is a notable advantage over Falcon and Mitaka.

Like in Dilithium, our modulus q is constant over the parameter sets and allows an optimized NTT implementation shared for all sets. With only 16-bit in size, our modulus also allows storing coefficients memory-efficiently without compression.

The rANS encoded values h and high bits of $\mathbf{z}_1$ lead to a varying signature size. In our current implementation we opted for a fixed signature size as reported in Table 4. We evaluated the distribution empirically and determined a threshold that requires a rejection in less than 0.1% of the cases. A field of two bytes indicates the length of the encoded values, the padding can be done with arbitrary data.

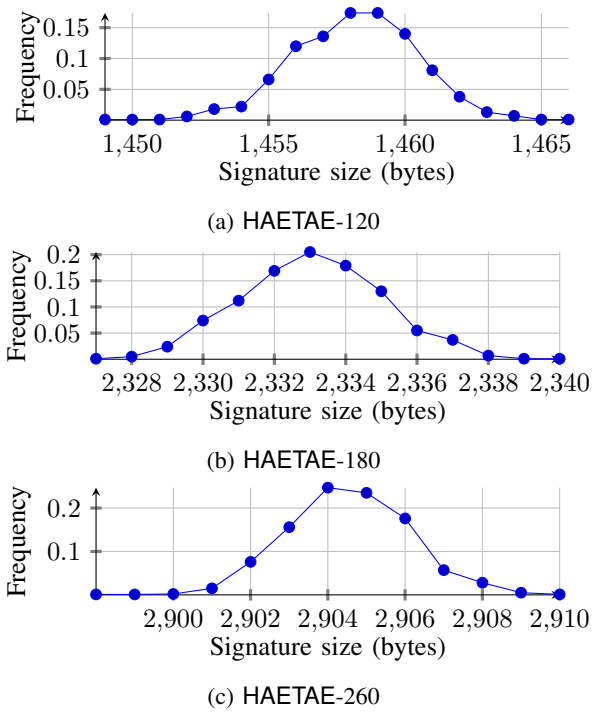
A dynamic signature size would allow an individual implementation to reject and recompute signatures until a desired size threshold is reached and still be compatible with implementations without this rejection. Due to the small variance in the distribution of the signature size, however, this would result in a distinct performance overhead, if the threshold is more than a few bytes below to the average size. Figure 11 displays the signature size distribution of 1000 executions.

In Table 4 we compare the signature and key sizes of HAETAE, Dilithium, and Falcon. The verification keys in HAETAE are 20% (HAETAE-260) to 25% (HAETAE-120 and HAETAE-180) smaller, than their counterparts in Dilithium. The advantage of the hyperball sampling manifests itself in the signature sizes, HAETAE has 30% to 40% smaller signatures than Dilithium. Less relevant are the

Security	120	180	260
q	64513	64513	64513
M	6.0	5.0	6.0
Key Rate	0.1	0.25	0.1
β	354.82	500.88	623.72
B	9388.97	17773.21	22343.66
B'	9382.26	17766.15	22334.95
B''	12320.79	21365.10	24441.49
(k, ℓ)	(2,4)	(3,6)	(4,7)
η	1	1	1
τ	58	80	128
α_{h}	512	512	256
d	1	1	0
Forgery			
BKZ block-size b	409 (333)	617 (512)	878 (735)
Classical hardness	119 (97)	180 (149)	256 (214)
Quantum hardness	105 (85)	158 (131)	225 (188)
Key-Recovery			
BKZ block-size b	428	810	988
Classical hardness	125	236	288
Quantum hardness	109	208	253

TABLE 3: HAETAE parameters sets. Hardness is measured with the Core-SVP methodology.

secret key sizes, that are almost half the size in HAETAE compared to Dilithium. A direct comparison to Falcon for the same claimed security level is only possible for the highest parameter set, Falcon-1024 has a signature of less than half the size compared to HAETAE-260, and its verification key is about 14% smaller.



(a) HAETAE-120

(b) HAETAE-180

(c) HAETAE-260

Figure 11: Signature size distribution over 1000 executions.

Scheme	Lvl.	vk	Signature	Sum	Secret key
HAETAE-120	2	992	1,463	2,455	1,376
HAETAE-180	3	1,472	2,337	3,809	2,080
HAETAE-260	5	2,080	2,908	4,988	2,720
Dilithium-2	2	1,312	2,420	3,732	2,528
Dilithium-3	3	1,952	3,293	5,245	4,000
Dilithium-5	5	2,592	4,595	7,187	4,864
Falcon-512	1	897	666	1,563	1,281
Falcon-1024	5	1,792	1,280	3,072	2,305

TABLE 4: NIST security level, signature and key sizes (bytes) of HAETAE, Dilithium, and Falcon.

5.2. Performance

We developed a unoptimized, portable and constant-time implementation in C for HAETAE and report median and average cycle counts of one thousand executions for each parameter set in Table 5. Due to the key and signature rejection steps, the median and average values for key generation and signing respectively differ clearly, whereas the two values are much closer for the verification.

For a fair comparison, we also performed measurements on the same system with identical settings of the reference implementation of Dilithium[2] and the implementation with emulated floating-point operations, and thus also fully portable, of Falcon[3], as given in Table 5. The performance of the signature verification for HAETAE is very close to Dilithium throughout the parameter sets. HAETAE-180 verification is about 23% slower than its' counter-part, HAETAE-260 on the other hand, is even 6% faster than the respective Dilithium parameter set. For key generation and signature computation, our current implementation of HAETAE is clearly slower than Dilithium. We measure a slowdown of factors two to five. In comparison to Falcon, however, HAETAE reports 30-90 times faster key generation and 2-4 times faster signing speed. For the verification, Falcon outperforms both Dilithium and HAETAE by a factor of four.

A closer look at the key generation reveals, that the complex Fast Fourier Transformation, that is required for the rejection step, is with 53% by far the most expensive operation and a sensible target for optimized implementations.

Profiling the signature computation reveals, that the slowdown compared to Dilithium is mainly caused by the sampling from a hyperball, where about 80% of the computation time is spent. The hyperball sampling itself is dominated by the generation of randomness, which we derive from the extendable output function SHAKE256 [30], which is also used in the Dilithium implementation. Almost 60% of the signature computation time is spend in SHAKE256. We expect an optimized software implementation e.g. with parallel hashing to be much more efficient and closer to, but not faster than Dilithium.

Based on the profiling and benchmarking of subcom-

2. https://github.com/pq-crystals/dilithium/tree/master/ref
3. https://falcon-sign.info/falcon-round3.zip

Scheme		KeyGen	Sign	Verify
HAETAE-120	*med*	1,384,274	6,253,166	387,594
	ave	1,832,973	8,903,852	388,377
HAETAE-180	*med*	2,333,614	9,472,724	718,010
	ave	3,464,004	11,763,246	719,400
HAETAE-260	*med*	1,693,776	8,989,980	913,378
	ave	2,129,737	12,459,046	914,336
Dilithium-2	*med*	339,334	1,140,794	367,264
	ave	339,569	1,446,174	367,399
Dilithium-3	*med*	609,696	1,955,296	585,536
	ave	610,114	2,359,859	585,755
Dilithium-5	*med*	935,830	2,473,582	975,802
	ave	936,202	2,904,138	976,350
Falcon-512	*med*	53,778,476	17,332,716	103,056
	ave	60,301,272	17,335,484	103,184
Falcon-1024	*med*	154,298,384	38,014,050	224,378
	ave	178,516,059	38,009,559	224,840

TABLE 5: Median and average cycle counts of 1000 executions for HAETAE, Dilithium, and Falcon. Cycle counts were obtained on one core of an Intel Core i7-10700k, with TurboBoost and hyperthreading disabled.

ponents, we estimate the performance of a randomized HAETAE implementation with pre-computation. The generic version, that is independent of the key, would already achieve a speedup of a factor five for its online signing, because the expensive hyperball sampling can be done offline. For the pre-computation variant with a designated signing key, additionally a lot of matrix-vector multiplications and therefore most of the transformations from and to the Number-Theoretic Transform (NTT) domain, can be precomputed. We estimate about 12% of the full deterministic signing running time, for the online signing in this case.

While the smallest parameter set HAETAE-120 yields the fastest implementation, our parameter selection leads to the unusual situation, that the most secure HAETAE-260 is very close or even faster in key generation and signing than HAETAE-180. HAETAE-180 is nevertheless a viable option, due to the smaller signature and key sizes compared to HAETAE-260.

Our rANS encoding is based on an implementation by Fabian Giesen [31].

5.3. Security against physical attacks

Implementation security is a crucial aspect of making cryptosystems feasible in real-world applications. A significant advantage of HAETAE is that it can be protected against power side-channel attacks efficiently and with reasonable overhead. In this context, we emphasize the similarity of HAETAE to Dilithium. Hence, past works analyzing concrete attacks [32, 33], but also countermeasures [11, 34], mainly apply to HAETAE as well.

While a fully protected implementation is out of scope for this paper, we briefly sketch its feasibility. During signing, the most critical operation is multiplying the (public) challenge polynomial c with $\mathbf{s}$ and subsequently adding the result to $\mathbf{y}$. Since this operation may leak information about the secret key statistically over many executions, implementers must protect it accordingly. As countermeasures against these so-called Differential Power Analysis (DPA) attacks, masking has been proven effective.

This operation is straightforward to mask at arbitrary order by splitting the secret key polynomials into multiple additive shares in $\mathcal{R}_q$. A masked implementation then stores the NTT of each share of $\mathbf{s}$ and multiplies them to c, obtaining a shared $c\mathbf{s}$. Following this, the inverse NTT is applied share-wise. Since $\mathbf{y}$ is a polynomial vector in $(1/N)\mathcal{R}$, it is not trivially possible to add our shares of $c\mathbf{s} \in \mathcal{R}_q^{k+\ell}$.

On the other hand, $\mathbf{y}$ is not a secret-key-dependent value. Therefore, it is not required to be protected against DPA but only against the much stronger attacker model of an Simple Power Analysis (SPA). In fact, coefficient-wise shuffling of the addition is sufficient at this point (cf. [34]). This might involve a masking conversion from $\mathbb{Z}_q$ to $\mathbb{Z}_{2^{32}}$, but no multiplication of masked fix-point values, which would be costly. Subsequently, the computation of $2\mathbf{z} - \mathbf{y}$ and the bound checks can be shuffled without applying costly masking.

The same idea applies to the whole hyperball sampling procedure. Since the order of the Gaussian samples is, in principle, irrelevant, they can be generated in random order. This is particularly an advantage for randomized HAETAE.

It is noteworthy that hashing the challenge seed is only required to be protected against SPA as well. Since the input order into the hash function cannot be randomized, the preceding values must still be protected by masking. Therefore, we propose to perform a shuffled point-wise multiplication of $\mathbf{A}$ and $\mathbf{y}$, directly followed by freshly masking the resulting coefficients. Then, a share-wise inverse NTT and a masking conversion to the Boolean domain will be performed, which enables a secure HighBits operation. For the LSBs of y_0, generating a fresh Boolean masking during the shuffled generation of the hyperball sample's coefficients is sufficient.

Comparison to Falcon and Mitaka. While there is no known method to efficiently mask Falcon, Mitaka [8] was designed to be easy to protect against implementation attacks, while still having the advantage of similarly small signatures as Falcon. For Mitaka, the crux regarding side-channel security is sampling Gaussian-distributed values. Together with Mitaka, an efficient, masked algorithm for discrete Gaussian sampling was presented. However, Prest broke its security proof recently [9]. In this respect, HAETAE has the strong advantage, that Gaussian sampling only needs to be secured against the much stronger SPA attacker model, which allows for simpler countermeasures, while Mitaka's side-channel security will always depend on a masked sampler.

5.4. Hardware implementations

Hashing and generation of randomness are the most time-consuming operations of HAETAE. Therefore, we assume that hardware implementations will bring significant speedup and can be competitive to Dilithium, particularly through

efficient Keccak cores. Furthermore, hardware implementations will benefit significantly from applying the offline approach. Naturally, a module generating hyperball samples can be instantiated and run parallel to the online phase, thus, hiding its latency behind the online phase. Moreover, high-speed applications could adopt the offline approach with designated signing key, including the multiplication of $\mathbf{A}$ and $\mathbf{y}$, to further reduce the latency of the online phase.

6. Conclusion

With HAETAE, we close an important gap between the two state-of-the-art digital signature schemes Dilithium and Falcon. Novel contributions in key generation and rejection sampling allow us to reach smaller signature and verification key sizes, while still allowing physical side-channel protected implementations for IoT use-cases.

Acknowledgments. Part of this work was done while Damien Stehlé was in École Normale Supérieure de Lyon and Institut Universitaire de France, and MinJune Yi was in CryptoLab Inc. Julien Devevey and Damien Stehlé were supported by the AMIRAL ANR grant (ANR-21-ASTR-0016), the PEPR quantique France 2030 programme (ANR-22-PETQ-0008) and the PEPR Cyber France 2030 programme (ANR-22-PECY-0003). Tim Güneysu, Markus Krausz, Georg Land and Marc Möller have been supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy - EXC 2092 CASA - 390781972, and by the German Federal Ministry of Education and Research BMBF through the projects QuantumRISC (16KIS1038) and PQC4Med (16KIS1044).

References

[1] P.-A. Fouque, J. Hoffstein, P. Kirchner, V. Lyubashevsky, T. Pornin, T. Prest, T. Ricosset, G. Seiler, W. Whyte, Z. Zhang *et al.*, "Falcon: Fast-fourier lattice-based compact signatures over ntru," *Submission to the NIST's post-quantum cryptography standardization process*, vol. 36, no. 5, 2018.

[2] L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, and D. Stehlé, "CRYSTALS-Dilithium: A lattice-based digital signature scheme," *IACR TCHES*, vol. 2018, no. 1, pp. 238–268, 2018, https://tches.iacr.org/index.php/TCHES/article/view/839.

[3] B. Westerbaan, "Sizing Up Post-Quantum Signatures," Cloudflare, Tech. Rep., 2021. [Online]. Available: https://blog.cloudflare.com/sizing-up-post-quantum-signatures/

[4] C. Paquin, D. Stebila, and G. Tamvada, "Benchmarking post-quantum cryptography in tls," in *Post-Quantum Cryptography: 11th International Conference, PQCrypto 2020, Paris, France, April 15–17, 2020, Proceedings 11*. Springer, 2020, pp. 72–91.

[5] J. Goertzen and D. Stebila, "Post-quantum signatures in dnssec via request-based fragmentation," *arXiv preprint arXiv:2211.14196*, 2022.

[6] E. Karabulut and A. Aysu, "Falcon down: Breaking falcon post-quantum signature scheme through side-channel attacks," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*, 2021, pp. 691–696.

[7] W. Schindler, "A timing attack against rsa with the chinese remainder theorem," in *Cryptographic Hardware and Embedded Systems—CHES 2000: Second International Workshop Worcester, MA, USA, August 17–18, 2000 Proceedings 2*. Springer, 2000, pp. 109–124.

[8] T. Espitau, P.-A. Fouque, F. Gérard, M. Rossi, A. Takahashi, M. Tibouchi, A. Wallet, and Y. Yu, "Mitaka: A simpler, parallelizable, maskable variant of falcon," in *EUROCRYPT 2022, Part III*, ser. LNCS, O. Dunkelman and S. Dziembowski, Eds., vol. 13277. Springer, Heidelberg, May / Jun. 2022, pp. 222–253.

[9] T. Prest, "A key-recovery attack against mitaka in the t-probing model," Cryptology ePrint Archive, Report 2023/157, 2023, https://eprint.iacr.org/2023/157.

[10] T. Pornin, "New efficient, constant-time implementations of falcon," Cryptology ePrint Archive, Paper 2019/893, 2019. [Online]. Available: https://eprint.iacr.org/2019/893

[11] V. Migliore, B. Gérard, M. Tibouchi, and P.-A. Fouque, "Masking Dilithium - efficient implementation and side-channel evaluation," in *ACNS 19*, ser. LNCS, R. H. Deng, V. Gauthier-Umaña, M. Ochoa, and M. Yung, Eds., vol. 11464. Springer, Heidelberg, Jun. 2019, pp. 344–362.

[12] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(Leveled) fully homomorphic encryption without bootstrapping," in *ITCS 2012*, S. Goldwasser, Ed. ACM, Jan. 2012, pp. 309–325.

[13] A. Langlois and D. Stehlé, "Worst-case to average-case reductions for module lattices," *Des. Codes Cryptogr.*, vol. 75, no. 3, pp. 565–599, 2015.

[14] V. Lyubashevsky, "Fiat-Shamir with aborts: Applications to lattice and factoring-based signatures," in *ASIACRYPT 2009*, ser. LNCS, M. Matsui, Ed., vol. 5912. Springer, Heidelberg, Dec. 2009, pp. 598–616.

[15] ——, "Lattice signatures without trapdoors," in *EUROCRYPT 2012*, ser. LNCS, D. Pointcheval and T. Johansson, Eds., vol. 7237. Springer, Heidelberg, Apr. 2012, pp. 738–755.

[16] L. Ducas, A. Durmus, T. Lepoint, and V. Lyubashevsky, "Lattice signatures and bimodal Gaussians," in *CRYPTO 2013, Part I*, ser. LNCS, R. Canetti and J. A. Garay, Eds., vol. 8042. Springer, Heidelberg, Aug. 2013, pp. 40–56.

[17] J. Hoffstein, J. Pipher, and J. H. Silverman, "NTRU: A ring-based public key cryptosystem," in *ANTS-III*, ser. LNCS, J. Buhler, Ed., vol. 1423. Springer, 1998, pp. 267–288.

[18] J. Devevey, O. Fawzi, A. Passelègue, and D. Stehlé, "On rejection sampling in lyubashevsky's signature scheme," in *ASIACRYPT 2022, Part IV*, ser. LNCS, S. Agrawal and D. Lin, Eds., vol. 13794. Springer, Heidelberg, Dec. 2022, pp. 34–64.

[19] T. Espitau, M. Tibouchi, A. Wallet, and Y. Yu, "Shorter hash-and-sign lattice-based signatures," in *CRYPTO 2022, Part II*, ser. LNCS, Y. Dodis and T. Shrimpton, Eds., vol. 13508. Springer, Heidelberg, Aug. 2022, pp. 245–275.

[20] C.-P. Schnorr, "Efficient identification and signatures for smart cards," in *CRYPTO'89*, ser. LNCS, G. Brassard, Ed., vol. 435. Springer, Heidelberg, Aug. 1990, pp. 239–252.

[21] A. R. Voelker, J. Gosmann, and T. C. Stewart, "Efficiently sampling vectors and coordinates from the n-sphere and n-ball," *Centre for Theoretical Neuroscience-Technical Report*, 01 2017.

[22] M. Andrysco, D. Kohlbrenner, K. Mowery, R. Jhala, S. Lerner, and H. Shacham, "On subnormal floating point and abnormal timing," in *2015 IEEE Symposium on Security and Privacy*. IEEE, 2015, pp. 623–639.

[23] M. Rossi, "Extended security of lattice-based cryptography," Ph.D. dissertation, 2020. [Online]. Available: https://www.di.ens.fr/~mrossi/docs/thesis.pdf

[24] G. Barthe, S. Belaïd, T. Espitau, P.-A. Fouque, M. Rossi, and M. Tibouchi, "GALACTICS: Gaussian sampling for lattice-based constant- time implementation of cryptographic signatures, revisited," in *ACM CCS 2019*, L. Cavallaro, J. Kinder, X. Wang, and J. Katz, Eds. ACM Press, Nov. 2019, pp. 2147–2164.

[25] J. Duda, "Asymmetric numeral systems: entropy coding combining speed of huffman coding with compression rate of arithmetic coding," 2013, arXiv preprint, available at https://arxiv.org/abs/1311.2540.

[26] E. Kiltz, V. Lyubashevsky, and C. Schaffner, "A concrete treatment of Fiat-Shamir signatures in the quantum random-oracle model," in *EUROCRYPT 2018, Part III*, ser. LNCS, J. B. Nielsen and V. Rijmen, Eds., vol. 10822. Springer, Heidelberg, Apr. / May 2018, pp. 552–586.

[27] J. Devevey, P. Fallahpour, A. Passelègue, and D. Stehlé, "A detailed analysis of Fiat-Shamir with aborts," Cryptology ePrint Archive, Paper 2023/245, 2023, https://eprint.iacr.org/2023/245. [Online]. Available: https://eprint.iacr.org/2023/245

[28] M. Barbosa, G. Barthe, C. Doczkal, J. Don, S. Fehr, B. Grégoire, Y. Huang, A. Hülsing, Y. Lee, and X. Wu, "Fixing and mechanizing the security proof of Fiat-Shamir with aborts and Dilithium," Cryptology ePrint Archive, Paper 2023/246, 2023, https://eprint.iacr.org/2023/246. [Online]. Available: https://eprint.iacr.org/2023/246

[29] A. B. Grilo, K. Hövelmanns, A. Hülsing, and C. Majenz, "Tight adaptive reprogramming in the QROM," in *ASIACRYPT 2021, Part I*, ser. LNCS, M. Tibouchi and H. Wang, Eds., vol. 13090. Springer, Heidelberg, Dec. 2021, pp. 637–667.

[30] M. J. Dworkin, "Sha-3 standard: Permutation-based hash and extendable-output functions," 2015.

[31] F. Giesen, "Interleaved entropy coders," *arXiv preprint arXiv:1402.3392*, 2014.

[32] L. G. Bruinderink and P. Pessl, "Differential fault attacks on deterministic lattice signatures," *IACR TCHES*, vol. 2018, no. 3, pp. 21–43, 2018, https://tches.iacr.org/index.php/TCHES/article/view/7267.

[33] S. Marzougui, V. Ulitzsch, M. Tibouchi, and J.-P. Seifert, "Profiling side-channel attacks on Dilithium: A small bit-fiddling leak breaks it all," Cryptology ePrint Archive, Report 2022/106, 2022, https://eprint.iacr.org/2022/106.

[34] M. Azouaoui, O. Bronchain, G. Cassiers, C. Hoffmann, Y. Kuzovkova, J. Renes, M. Schönauer, T. Schneider, F.-X. Standaert, and C. van Vredendaal, "Leveling Dilithium against leakage: Revisited sensitivity analysis and improved implementations," Cryptology ePrint Archive, Report 2022/1406, 2022, https://eprint.iacr.org/2022/1406.

Appendix A. Missing Proofs

A.1. Useful Lemma

We will rely on the following claim.

Lemma 10. *Let n be the degree of $\mathcal{R}$. Let $m, N, r > 0$ and $\mathbf{v} \in \mathcal{R}^m$. Then the following statements hold:*

1) $|(1/N)\mathcal{R}^m \cap \mathcal{B}_{\mathcal{R},m}(r)| = |\mathcal{R}^m \cap \mathcal{B}_{\mathcal{R},m}(Nr)|$,
2) $|\mathcal{R}^m \cap \mathcal{B}_{\mathcal{R},m}(r, \mathbf{v})| = |\mathcal{R}^m \cap \mathcal{B}_{\mathcal{R},m}(r)|$,
3) $\mathsf{Vol}(\mathcal{B}_{\mathcal{R},m}(r - \frac{\sqrt{mn}}{2})) \leq |\mathcal{R}^m \cap \mathcal{B}_{\mathcal{R},m}(r)| \leq \mathsf{Vol}(\mathcal{B}_{\mathcal{R},m}(r + \frac{\sqrt{mn}}{2}))$.

Proof. For the first statement, note that we only scaled $(1/N)\mathcal{R}^m$ and $\mathcal{B}_{\mathcal{R},m}(r)$ by a factor N. For the second statement, note that the translation $\mathbf{x} \mapsto \mathbf{x} - \mathbf{v}$ maps $\mathcal{R}^m$ to $\mathcal{R}^m$.

We now prove the third statement. For $x \in \mathcal{R}^m$, we define $T_{\mathbf{x}}$ as the hypercube of $\mathcal{R}_{\mathbb{R}}^m$ centered in $\mathbf{x}$ with side-length 1. Observe that the $T_{\mathbf{x}}$'s tile the whole space when $\mathbf{x}$ ranges over $\mathcal{R}^m$ (the way bounderies are handled does not matter for the proof). Also, each of those tiles has volume 1. As any element in $T_{\mathbf{x}}$ is at Euclidean distance at most $\sqrt{mn}/2$ from $\mathbf{x}$, the following inclusions hold:

$$\mathcal{B}_{\mathcal{R},m}\Big(r - \frac{\sqrt{mn}}{2}\Big) \subseteq \bigcup_{\mathbf{x} \in \mathcal{R}^m \cap \mathcal{B}_{\mathcal{R},m}(r)} T_{\mathbf{x}} \subseteq \mathcal{B}_{\mathcal{R},m}\Big(r + \frac{\sqrt{mn}}{2}\Big).$$

Taking the volumes gives the result. □

A.2. Proof of Lemma 1

Proof. Figure 2 is the bimodal rejection sampling algorithm applied to the source distribution $U((1/N)\mathcal{R}^m \cap \mathcal{B}_{\mathcal{R},m}(r'))$ and target distribution $U((1/N)\mathcal{R}^m \cap \mathcal{B}_{\mathcal{R},m}(r))$ (see, e.g., [18]). It then suffices that the support of the bimodal shift of the source distribution by $\mathbf{v}$ contains the support of the target distribution. It is implied by $r' \geq \sqrt{r^2 + t^2}$.

We now consider the number of expected iterations, i.e., the maximum ratio between the two distributions. To guide the intuition, note that if we were to use continuous distributions, the acceptance probability $1/M'$ would be bounded by $1/M$. In our case, the acceptance probability can be bounded as follows (using Lemma 10):

$$\begin{aligned}\frac{1}{M'} &= \frac{|(1/N)\mathcal{R}^m \cap \mathcal{B}_{\mathcal{R},m}(r)|}{2|(1/N)\mathcal{R}^m \cap \mathcal{B}_{\mathcal{R},m}(r')|} = \frac{|\mathcal{R}^m \cap \mathcal{B}_{\mathcal{R},m}(Nr)|}{2|\mathcal{R}^m \cap \mathcal{B}_{\mathcal{R},m}(Nr')|} \\ &\geq \frac{\mathsf{Vol}(\mathcal{B}_{\mathcal{R},m}(Nr - \sqrt{mn}/2))}{2\mathsf{Vol}(\mathcal{B}_{\mathcal{R},m}(Nr' + \sqrt{mn}/2))} \\ &= \frac{1}{2}\left(\frac{Nr - \sqrt{mn}/2}{Nr' + \sqrt{mn}/2}\right)^{mn}.\end{aligned}$$

It now suffices to bound the latter term from below by $1/(cM) = 1/(2c(r'/r)^{mn})$. This inequality is equivalent to:

$$c \geq \frac{1}{2} \cdot \left(\frac{r}{r - \sqrt{mn}/(2N)}\right)^{mn} \cdot \left(\frac{r' + \sqrt{mn}/(2N)}{r'}\right)^{mn},$$

and to:

$$N \geq \frac{1}{c^{1/(mn)} - 1} \cdot \frac{\sqrt{mn}}{2}\left(\frac{c^{1/(mn)}}{r} + \frac{1}{r'}\right),$$

which allows to complete the proof. □

A.3. Proof of Lemma 4

Proof. By Lemma 3, there exists a unique representation

$$r = \lfloor (r + \alpha/2)/\alpha \rfloor \, \alpha + (r \bmod^{\pm} \alpha).$$

By identifying $\mathsf{HighBits}(r, \alpha)$ and $\mathsf{LowBits}(r, \alpha)$ in the above equation, we obtain the first result.

By definition of $\bmod^{\pm} \alpha$, we have that $r' \in [-\alpha/2, \alpha/2)$.

Finally, since $r \mapsto \lfloor (r + \alpha/2)/\alpha \rfloor$ is non-decreasing, it suffices to show that $\lfloor (2q - 1 + \alpha/2)/\alpha \rfloor \leq \lfloor (2q-1)/\alpha \rfloor$. We have $(2q - 1 + \alpha/2) \leq \lfloor (2q-1)/\alpha \rfloor \alpha + \alpha - 1$ by assumption on q. Dividing by α and taking the floor yields the result.

□

A.4. Proof of Lemma 5

Proof. Let $r \in [0, 2q-1]$. Let r_1, r_2, r'_1, and r'_2 defined as in Definition 7.

The equality $r'_1 + r'_2 \cdot \alpha_{\mathsf{h}} = r_1 + r_2 \cdot \alpha_{\mathsf{h}} \bmod 2q$ holds vacuously if $r'_1 = r_1$ and $r'_2 = r_2$.

If not, then $r'_1 = r_1 - 2$ and $r'_2 = r_2 - 2(q-1)/\alpha_{\mathsf{h}}$ and $r'_1 + r'_2\alpha_{\mathsf{h}} = r_1 + r_2\alpha_{\mathsf{h}} - 2q$. By Lemma 4, we get the first equality.

The second property stems from the second property in Lemma 4. The modifications to r_1 make r'_1 lie in the range $[-\alpha_{\mathsf{h}}/2 - 2, \alpha_{\mathsf{h}}/2)$.

The last property stems from the third property in Lemma 4 and the fact that if $r_2 = m$, then we have $r'_2 = 0$. □

A.5. Proof of Lemma 7

Proof. Let $\mathbf{y} \in \mathcal{B}_{\mathcal{R},m}(Nr' + \sqrt{mn}/2)$ and set $\mathbf{z} = \lfloor \mathbf{y} \rceil$. Note that $\mathbf{z}$ is sampled (before the rejection step) with probability

$$\frac{\mathsf{Vol}(T_{\mathbf{z}} \cap \mathcal{B}_{\mathcal{R},m}(Nr' + \sqrt{mn}/2))}{\mathsf{Vol}(\mathcal{B}_{\mathcal{R},m}(Nr'))},$$

where $T_{\mathbf{z}}$ is the hypercube of $\mathcal{R}^m_{\mathbb{R}}$ centered in $\mathbf{z}$ with side-length 1. By the triangle inequality, this probability is equal to $1/\mathsf{Vol}(\mathcal{B}_{\mathcal{R},m}(Nr' + \sqrt{mn}/2)$ when $\mathbf{z} \in \mathcal{B}_{\mathcal{R},m}(Nr')$. Hence the distribution of the output is exactly $U(\mathcal{R}^m \cap \mathcal{B}_{\mathcal{R},m}(Nr'))$, as each element is sampled with equal probability and as the algorithm almost surely terminates (its runtime follows a geometric law of parameter the rejection probability).

It remains to consider the acceptance probability.

$$\frac{\sum_{\mathbf{y} \in \mathcal{R}^m \cap \mathcal{B}_{\mathcal{R},m}(Nr')} \mathsf{Vol}(T_{\mathbf{y}} \cap \mathcal{B}_{\mathcal{R},m}(Nr' + \sqrt{mn}/2))}{\mathsf{Vol}(\mathcal{B}_{\mathcal{R},m}(Nr' + \sqrt{mn}/2))}.$$

By the triangle inequality and Lemma 10, it is

$$\frac{|\mathcal{R}^m \cap \mathcal{B}_{\mathcal{R},m}(Nr')|}{\mathsf{Vol}(\mathcal{B}_{\mathcal{R},m}(Nr' + \sqrt{mn}/2))} \geq \left(\frac{Nr' - \sqrt{mn}/2}{Nr' + \sqrt{mn}/2}\right)^{mn}.$$

Note that by our choice of N, this is $\geq 1/M_0$. □