SOLMAE[⋆]

Algorithm Specifications

Kwangjo Kim[1], Mehdi Tibouchi[2], Alexandre Wallet[4],
Thomas Espitau[2], Akira Takahashi[3], Yang Yu[5], and Sylvain Guilley[6]

[1] International Research Institute for Cyber Security(IRCS)/KAIST
kkj@kaist.ac.kr
[2] NTT Social Informatics Laboratories, Japan
mehdi.tibouchi.br,thomas.espitau.ax}@hco.ntt.co.jp
[3] Aarhus University, Denmark
takahashi.akira.58s@gmail.com
[4] Inria, France
alexandre.wallet@inria.fr
[5] Tsinghua University, China
yang.yu0986@gmail.com
[6] Secure-IC, France
sylvain.guilley@secure-ic.com

Abstract. This document specifies the SOLMAE signature scheme submitted to the Korean Post-Quantum Competition. SOLMAE is a lattice-based signature scheme following the hash-and-sign paradigm (in the style of Gentry–Peikert–Vaikuntanathan signatures), and instantiated over NTRU lattices. In that sense, it is closely related to, and a successor of, several earlier schemes including Ducas–Lyubashevsky–Prest (DLP), FALCON and MITAKA. More precisely, SOLMAE offers the "best of both worlds" between FALCON and MITAKA.

FALCON has the advantage of providing short public keys and signatures (offering essentially the best bandwidth trade-off among post-quantum constructions) as well as high security levels; however, it is plagued by a contrived signing algorithm that makes it very difficult to implement correctly, not very fast for signing and hard to parallelize; it also has very little flexibility in terms of parameter settings. In contrast, MITAKA is much simpler to implement, twice as fast in equal dimension, straightforward to parallelize and fully versatile in terms of parameters; however, it has lower security than FALCON in equal dimension, has an even more contrived key generation algorithm that tends to be quite slow, and has somewhat larger keys and signatures at equivalent security levels.

SOLMAE solves the conundrum of choosing between those two schemes by offering all the advantages of both. It uses the same simple, fast, parallelizable signing algorithm as MITAKA, with flexible parameters. However, by leveraging a novel key generation algorithm that is much faster and achieves higher security, SOLMAE achieves the same high security and short key and signature sizes as FALCON. It is also compatible with recently introduced ellipsoidal lattice Gaussian sampling techniques to further reduce signature sizes. This makes SOLMAE the state-of-the-art in terms of constructing efficient lattice-based signatures over structured lattices. Some further challenges are left in the conclusion.

Keywords: Signature schemes · Lattice-based cryptography · Hash-and-sign paradigm · Module lattices · Lattice Gaussian sampling

[⋆] This work is submitted to 'Korean Post-Quantum Cryptography Competition' (www.kpqc.or.kr).

Table of Contents

1 Introduction

Designing cryptographically-strong primitives such as digital signatures or key encapsulation mechanisms, *etc.* are really a big challenge and could not be accomplished in a short time by one expert. A group of smart designers must understand all the known attacks so far from the theoretical and implementation points of view and anticipate the feasible attacks in the near future. Our team consisting of top-level cryptographers around the world has started to suggest long-term quantum-secure digital signature against quantum attack based on NTRU lattices, well-understood by the cryptographic community since their introduction around two decades ago.

SOLMAE is a lattice-based signature scheme inspired by several pioneering works and stands for quantum-**S**ecure alg**O**rithm for **L**ong-term **M**essage **A**uthentication and **E**ncryption. At its core, it is based on the hash-then-sign signature paradigm proposed by Gentry, Peikert and Vaikuntanathan [GPV08]. To be efficiently instantiated, this framework needs a class of lattices enjoying efficiently computable *trapdoor bases* for the signing procedure. Ducas, Lyubashevsky and Prest [DLP14] showed that NTRU lattices are such a nice class of lattices. Signing then amounts to sampling short Gaussian vectors in a public NTRU lattice; a quasi-linear (time and memory) but complex procedure called Fast Fourier Sampling was described by Ducas and Prest [DP16], culminating in Falcon, one of the recent winner of the NIST call for post-quantum standards. This competitor achieves the most compact signatures, and the most compact verification key and signatures combined sizes, and boasts verification times on par with elliptic curves signatures.

As its name suggests, SOLMAE is inspired from Falcon's design. Some of the new theoretical foundations of our scheme were laid out in the presentation of Mitaka [EFG+22]. At a high-level, it removes the inherent technicality of the sampling procedure, and most of its induced complexity from an implementation standpoint, for *free*, that is with no loss of efficiency. The simplicity of our design translates into faster operations while preserving signatures and verification keys sizes, on top of allowing for additional features absent from Falcon, such as enjoying cheaper masking, and being parallelizable. By using the novel compression techniques and tools of [ETWY22], we can also obtain smaller signatures and verification keys than those already achieved by Falcon. The impact on the security of these techniques needs to be carefully evaluated; while [ETWY22] has paved the way for this analysis, we will implant these ingredients in future versions of SOLMAE. To sum-up, our scheme achieves *better performances* for the *same security and advantages* as Falcon.

Two publications related with SOLMAE are:

- MITAKA paper presented at Eurocrypt2022 [EFG+22] and
- Compression paper presented at Crypto2022 [ETWY22].

1.1 Design rationale

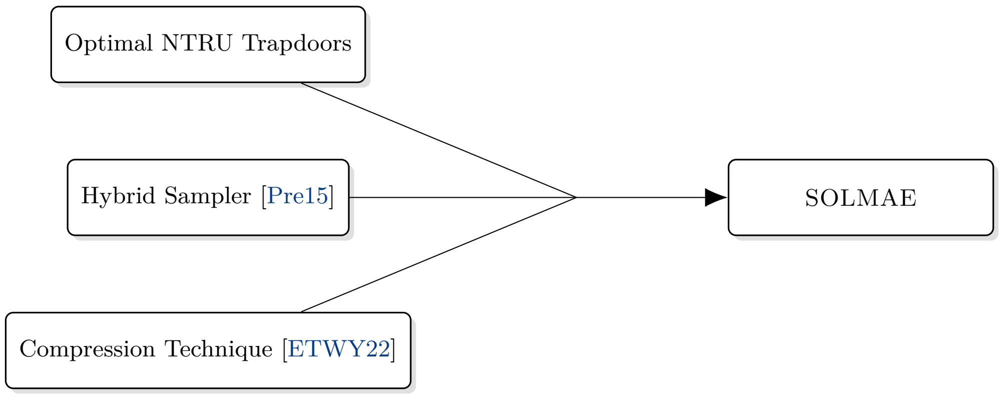

Fig. 1: Overview of SOLMAE

Overall, SOLMAE is summarized in Figure 1.

More details about all the object mentioned in this section can be found later in the documentation. Here we focus on the big lines behind our scheme's principles, keeping details at a minimum. While its predecessor FALCON could be summed-up as "an efficient instantiation of the GPV framework", SOLMAE takes it one step further. The ingredients behind the boxes in Figure 1 are:

- the **Hybrid sampler** is a faster, simpler, parallelizable and maskable Gaussian sampler to generate signatures;
- an **optimally tuned key generation algorithm**, enhancing the security of our new sampler to that of FALCON's level[7];
- **dedicated compression techniques** to reduce bandwith consumption even further, at no cost on the security according to our analyses — as mentioned, these will be added in future versions of the scheme.

As mentioned previously, part of the compression techniques are generic and without impact on the security level. On the other hand, other techniques require to tweak the key-generation and signing procedures. Their addition to SOLMAE's design will appear in future versions.

The rest of the section enters more in depth about the history behind these improvements, and the reason of their existence. For a more concrete description of the objects, the readers is deferred to Section 3.

A quick overview of hash-then-sign over lattices Almost all hard cryptographic problems from lattices involve either computing short vectors or decoding a target to a close lattice point, from an arbitrarily bad description of the lattice. Hash-then-sign over lattices is no exception, as it can be described as follows:

- a message M is hashed as a vector $m = H(M)$ in the ambient space of a *public* lattice $\mathcal{L}$;
- After computing a point $v \in \mathcal{L}$ *quite close* to $H(M)$, a signature is $s = H(M) - v$;
- a pair (M, s) is valid if $H(M) - s$ belongs to $\mathcal{L}$ and s is short enough.

On the one hand, only the signer should be able to *efficiently* compute v close enough to an arbitrary target. This is a decoding problem that can be solved when a basis of *short* vectors is known. On the other hand, anyone wanting to check the validity of a signature should be able to verify lattice membership. The scheme therefore relies on two main ingredients:

1. the ability to generate pairs $(\mathbf{A}, \mathbf{B})$ of bases for a given lattice $\mathcal{L}$, where $\mathbf{B}$ remains secret and is composed of short vectors;
2. an efficient procedure exploiting $\mathbf{B}$ to compute signatures.

It is common to call the secret basis $\mathbf{B}$ a *trapdoor*, and in this documentation we will call $(\mathbf{A}, \mathbf{B})$ a trapdoor pair.

Lessons learned from the first instantiation NTRUSIGN [HHP+03] was the first hash-then-sign signature scheme relying on lattice problems, and historically the second use of the so-called NTRU lattices [HPS98]. For small polynomials $f, g \in \mathbb{Z}[X]/(X^n - 1)$, that is, with coefficients of small magnitude, let $h = g/f \bmod q$. One can represent h by its matrix of multiplication $[h]$ and the NTRU lattice is then

$$\mathcal{L}_{\text{NTRU}} := \{(u, v) \in \mathbb{Z}^{2d} : [h]u - v = 0 \bmod q\}.$$

To check lattice membership, it is enough to know h and q. In particular, by their definition, all vectors $(x^i f, x^i g)$ are short, and in the lattice. The authors of [HHP+03] gave an algorithm to complete them into a full basis of $\mathcal{L}_{\text{NTRU}}$ (see also Section 3.2). To sign a message, the idea was to rely on Babai's round-off algorithm : take the coordinate of a message in the basis $\mathbf{B}$, and round them to their nearest integers to obtain a close-by lattice point. While signing was efficient, it was soon realized that this approach was flawed beyond repair: each signature was leaking information about $\mathbf{B}$. After enough observations, an attacker could use statistical learning to recover $\mathbf{B}$ itself [NR06, DN12]. The scheme was thus abandoned.

[7] This corresponds to NIST-I and NIST-V requirements.

The Gentry-Peikert-Vaikuntanathan paradigm For the purpose of this section, we will only focus on the necessary ingredients of the GPV framework [GPV08]. More details can be found in Section 3. The key observation is that one can get a non-leaking signature algorithm by replacing the round-off by *lattice Gaussian sampling*. Such a procedure would output random vectors in $\mathcal{L}_{\text{NTRU}}$ with a Gaussian-like distribution independent of the lattice basis, thwarting statistical attacks to recover $\mathbf{B}$. The authors of [GPV08] also recalled Klein's algorithm to sample lattice Gaussians in quadratic time, but the *practical* efficiency of the whole design was not really addressed. This result was undeniably a huge step forward for hash-then-sign over lattices, however trapdoors were now not only asked to be made of short vectors, they would also need that their *Gram-Schmidt orthogonalization* was made of vectors as short as possible. This was indeed necessary to ensure that Klein's algorithm would output Gaussians with small variance; without such a property, it would be easier for an adversary to forge valid signatures.

NTRU strikes again: the trapdoors of Ducas, Lyubashevsky and Prest In [DLP14], NTRU-like lattices were again the center of attention. Switching from $\mathbb{Z}[X]/(X^n - 1)$ to a ring of cyclotomic integers $R = \mathbb{Z}[X]/(X^{2^n} + 1)$, the authors observed that the algebraic structure underlying these lattices gave strong and useful geometric constraints. Generally, the largest Gram-Schmidt vectors of an NTRU lattices would correspond to (f, g) and a completion (F, G) of the secret basis. It is hopeless to expect *any* (f, g) to lead to a trapdoor basis useful for signing, but one could hope to find a good pair reasonably fast by some kind of random walk among the set of potential keys. Indeed, backed-up by extensive experimental confirmations, the authors gave a carefully tuned key generation algorithm relying on Gaussian cyclotomic integers, and while their focus was a more advanced cryptographic functionality, this was arguably the birthstone of FALCON.

Sampling: an interplay between trapdoor quality and security level Klein's algorithm actually suffers from its quadratic complexity in practice. Because of the algorithm's design, it also unfriendly to parallelization. Ducas and Prest soon realized that these limitations could be avoided thanks to the algebraic structure of the underlying cyclotomic ring $\mathbb{Z}[X]/(X^{2^n} + 1)$. They described a recursive *quasi-linear* approach [DP16] to Klein's algorithm exploiting the *tower* structure of the ring, in the spirit of the Fast Fourier Transform algorithm — which gave its name to their new sampler. At the price of an intricate implementation, the resulting Gaussian sampler achieves impressive performances, close to signing time of the other NIST winning signature, DILITHIUM.

A natural question is to wonder whether such complications can be avoided. There are, after all, other approaches to lattice Gaussian sampling such as Peikert's "randomized Round-off" [Pei10] and Ducas-Prest's *Hybrid sampler*. Both are way simpler than the Fast Fourier Sampler and even more efficient, but the limitations of these algorithms boil down to the metric driving their *quality*, that is, the variance they can achieve — recall that the smaller variance is, the better the security is. Each sampling algorithm relies on a different metric tied to its geometric design, or in other words, the notion of "good trapdoor" differs from a sampling algorithm to another. It was already identified by Prest [Pre15] that Klein's approach would always be the better choice for security, with the hybrid being a not-so-close second and Peikert's arriving even further. More precisely, Prest's experiments suggested that finding useful trapdoors for these two other approaches was quite less likely to happen; in other words, the sparsity of good trapdoors made them expensive to find. From these observations, FALCON's team made the understandable choice of a better security in general. This choice unfortunately sacrificed not only simplicity, but also side-channel resilience.

SOLMAE takes off SOLMAE's design relies on the much simpler *Hybrid sampler* [Pre15], that can fully exploit the algebraic structure underlying NTRU lattices. As stated above, a simple reuse of FALCON's key-generation algorithm would however not be enough to go with. In [EFG$^+$22], a refined key generation algorithm put back the hybrid sampler into light, allowing to find better trapdoors in time comparable to that of FALCON, while keeping mild security losses. We can in fact go a step further: we designed a new, tailored key-generation algorithm, allowing not only to compute hybrid sampler trapdoors more efficiently, but even drastically improving over their quality from [EFG$^+$22]. These ingredients are then assembled together with the new advances on the cryptanalysis of the underlying algorithmic problems [ETWY22] to optimize parameter sets and minimize the bandwith consumption.

1.2 Advantages and limitations

SOLMAE enjoys the following advantages.

- **Compactness:** The signature size, or the combined verification key plus signature size, are comparable to that of Falcon's, which was the lightest in bandwith consumption among the winning signatures in NIST's competition. They can be further reduced by the addition of the compression techniques of [ETWY22].
- **Simplicity and efficiency:** The hybrid sampler is tailored to exploit the algebraic structures of NTRU lattices, involves only straightforward, elementary operations between polynomials, and is practically more efficient than the FFO sampler.
- **Side-channel resilience:** Masking SOLMAE can be done with standard and well-understood counter-measures, at cheaper overhead than Falcon.

On the other hand, our scheme presents some drawbacks:

- **Reliance on floating point arithmetic:** Just like its ancestor Falcon, our scheme relies importantly on the Fourier representation of polynomials, that is, representing them by evaluation at complex roots of unity, prompting the use for floating points arithmetic.
- **Algebraically structured security assumptions:** ntru lattices enjoys strong symmetries stemming from their algebraic structure, meaning that the underlying hardness assumption corresponds to a subclass of problems potentially easier than for plain, regular lattices. We stress that up to current knowledge, no significant improvement on the cryptanalysis or the asymptotic complexity are known for these problems (which is not a guarantee that none will ever be found).
- **No formal security proof:** Signature schemes in the GPV framework [GPV08] were proven resistant against forgery (sEF-CMA in qROM [BDF+11]) in a regime of parameters that differs noticeably from SOLMAE's (or Falcon, for that matter). Even if one relies on "the NTRU assumption" (allowing to consider that the public key is uniformly random), the parameters of the scheme do not follow the regime of the formal proof. This is a common discrepancy between concrete instantiations and theoretical schemes.

2 Preliminaries

Vectors are in bold lower case, and considered in column. Matrices are in bold upper case. When we say a matrix is a basis of a space, we mean the column vectors of the matrix are the basis. The ℓ_2-norm of a vector $\mathbf{x} = (x_1, \ldots, x_d)$ is $\|\mathbf{x}\| = (\sum_i |x_i|^2)^{1/2}$ and its ℓ_∞-norm is $\|\mathbf{x}\|_\infty = \max_i |x_i|$.

Lattices A lattice is a discrete subgroup of $\mathbb{R}^n$. Equivalently, it is the set of *integer* linear combinations obtained from a basis $\mathbf{B}$ of $\mathbb{R}^n$. The volume of a lattice is $\det \mathbf{B}$ for any of its basis.

Cyclotomic powers-of-two rings For $d = 2^n$, we let $K = \mathbb{Q}[X]/(X^d + 1)$ be the d-th cyclotomic field. We will work often in the subring $R = \mathbb{Z}[X]/(X^d + 1)$, and sometime in the overring $K_\mathbb{R} = \mathbb{R}[X]/(X^d + 1)$. Let $\zeta_j = \exp(i(2j-1)\pi/d)$ for $1 \le j \le d$ be the d-th primitive roots of 1, and for all $f \in K_\mathbb{R}$, let $\varphi_i(f) = f(\zeta_i)$. A polynomial in $K_\mathbb{R}$ can be represented in several ways. A first is the so-called coefficient embedding $f = \sum f_i X^i \mapsto \mathbf{f} = (f_0, \ldots, f_{d-1})$. Another is the canonical embedding $f \mapsto \varphi(f) = (\varphi_1(f), \ldots, \varphi_d(f))$. The map φ is also known as the Discrete Fourier Transform (DFT) and in particular, φ maps the polynomial multiplication in $K_\mathbb{R}$ to a coordinate-wise multiplication. The set of f's such that all $\varphi_i(f) \in \mathbb{R}^*_+$ is denoted by $K_\mathbb{R}^{++}$. We have $\|\varphi(f)\| = \sqrt{d}\|\mathbf{f}\|$. We let f^* be the complex conjugate of f and

$$f^* = f_0 - f_{d-1}X - \ldots - f_1 X^{d-1}, \quad \varphi(f^*) = (\overline{\varphi_i(f)})_i.$$

A third representation of cyclotomic elements is by matrices of multiplication:

$$f \mapsto [f] := \begin{bmatrix} f_0 & -f_{d-1} & \dots & -f_1 \\ f_1 & f_0 & \dots & -f_2 \\ \vdots & & \ddots & \vdots \\ f_{d-1} & f_{d-2} & \dots & f_0 \end{bmatrix}.$$

We have $[f^*] = [f]^t$.
In algorithms, we will write $(f_i)_i \in \mathbb{R}^d$ or $f \in K_\mathbb{R}$: the former highlights that the vector of coefficient is considered, while the latter implies the Fourier representation is used.

Algebraic Gram-Schmidt For $(f, g), (F, G) \in K_\mathbb{R}^{2\times 2}$, we define $\langle (f, g), (F, G)\rangle_K = f^*F + g^*G$. The Gram-Schmidt orthogonalization of $(F, G) \in K_\mathbb{R}^2$ with respect to (f, g) is

$$(\widetilde{F}, \widetilde{G}) = (F, G) - \frac{\langle (f, g), (F, G)\rangle_K}{\langle (f, g), (f, g)\rangle_K} \cdot (f, g),$$

and one checks that $\langle (f, g), (\widetilde{F}, \widetilde{G})\rangle_K = 0$.

NTRU lattices Let q be an integer, and $f \in R$ such that f is invertible modulo q (equivalently, $\det[f]$ is coprime to q). Let $h = g/f \bmod q$ and consider the NTRU module associated to h:

$$\mathcal{M}_{\text{NTRU}} = \{(u, v) \in R^2 : hu - v = 0 \bmod q\},$$

and its lattice version

$$\mathcal{L}_{\text{NTRU}} = \{(\mathbf{u}, \mathbf{v}) \in \mathbb{Z}^{2d} : [h]\mathbf{u} - \mathbf{v} = 0 \bmod q\}.$$

This lattice has volume q^d. Over R, it is generated by (f, g) and any (F, G) such that $fG - gF = q$. For such a pair $(f, g), (F, G)$, this means that $\mathcal{L}_{\text{NTRU}}$ has a basis of the form

$$\mathbf{B}_{f,g} = \begin{bmatrix} [f] & [F] \\ [g] & [G] \end{bmatrix}.$$

One checks that $([h], -\mathrm{Id}_d) \cdot \mathbf{B}_{f,g} = 0 \bmod q$, so the verification key is h. More details are given in Section 3.2. The NTRU-search problem is : given $h = g/f \mod q$, find any $(f' = x^i f, g' = x^i g)$. In its decision variant, one must distinguish $h = g/f \bmod q$ from a uniformly random $h \in R_q := \mathbb{Z}[X]/(q, X^d + 1) = (\mathbb{Z}/q\mathbb{Z})[X]/(X^d + 1)$. These problems are assumed to be intractable for large d.

Quality of an NTRU basis The secret basis will not be any pair, as it also needs to enable the sampling of short Gaussian vectors in $\mathcal{L}_{\text{NTRU}}$ by hybrid sampling. The *quality* of an NTRU basis $\mathbf{B}_{f,g}$ quantifies this:

$$\mathcal{Q}(f, g) = \max_{1\le i\le d/2} \max\left(\frac{|\varphi_i(f)|^2 + |\varphi_i(g)|^2}{q}, \frac{q}{|\varphi_i(f)|^2 + |\varphi_i(g)|^2}\right)^{1/2}.$$

Note that only half of the embeddings are needed, since the remaining ones correspond to the complex conjugates. It is one of the main parameter driving the security of the scheme. It cannot be lower than 1, but the closer it is to 1, the harder forgery attacks are.

Gaussian distributions The Gaussian function centered at $\mathbf{c} \in \mathbb{R}^d$ and (positive definite) covariance Σ is $\rho_{\mathbf{c},\Sigma}(\mathbf{x}) = \exp(-\frac{1}{2}(\mathbf{x} - \mathbf{c})^t \Sigma^{-1}(\mathbf{x} - \mathbf{c}))$. The normal distribution $\mathcal{N}_{\mathbf{c},\Sigma}$ centered at $\mathbf{c}$ and covariance Σ has density proportional to $\rho_{\mathbf{c},\Sigma}$. When we write $x \leftarrow \mathcal{N}_\Sigma^{K_\mathbb{R}}$, we mean that the corresponding d dimensional vector $\frac{1}{\sqrt{d}}(\Re\varphi_1(x), \Im\varphi_1(x) \dots, \Re\varphi_{d/2}(x), \Im\varphi_{d/2}(x))$ has distribution $\mathcal{N}_\Sigma$, where $\Re z, \Im z$ are the real and imaginary part of the complex z. For a lattice $\mathcal{L} \subset \mathbb{R}^d$, the discrete Gaussian distribution over $\mathcal{L}$ with parameters $\mathbf{c} \in \mathbb{R}^d$ and Σ is defined for all $\mathbf{x} \in \mathcal{L}$ as

$$D_{\mathcal{L},\mathbf{c},\Sigma}(\mathbf{x}) = \frac{\rho_{\mathbf{c},\Sigma}(\mathbf{x})}{\rho_\Sigma(\mathcal{L} - \mathbf{c})}.$$

We omit the center if it is 0. When Σ is a scalar matrix $s^2\mathbf{I}$, we note $\mathcal{N}_s$ or $D_{\mathcal{L},\mathbf{t},s}$.

3 Specifications

We first detail the principles of the GPV framework. This highlights the necessary ingredients for an efficient signature scheme. The next sections are devoted to the scheme's description, and how we instantiate the triple `KeyGen`, `Sign`, `Verif`. The concrete values for the many parameters to be introduced can be found at the end of the section in Table 1.

3.1 High-level view of the SOLMAE's signature scheme

As in any signature scheme, we need to instantiate three algorithms `KeyGen`, `Sign` and `Verif`. Moreover, the GPV framework adds the following requirements:

– a class of lattices where `KeyGen` computes trapdoor pairs;
– `Sign` uses a `Sample` procedure to generate random vectors in these lattices, with a distribution not leaking information about the trapdoor.

In the GPV framework, `KeyGen` outputs trapdoor pair $(\mathbf{A}, \mathbf{B})$ for a lattice $\mathcal{L}$ with $\mathbf{AB} = 0 \bmod q$, for a public integer q. Note that $\mathcal{L}$ is spanned by $\mathbf{B}$. The signing-verification routine is then:

1. `Sign`$(\mathbf{B}, M)$ first uses a cryptographic hash function `H` to get a vector $\mathbf{m} = \mathtt{H}(M)$ in the ambient space $\mathbb{R}^m$, and computes a preimage $\mathbf{c} = \mathbf{A}^{-1}\mathbf{m}$. Then `Sample`$(\mathbf{B}, \mathbf{c})$ finds a random vector $\mathbf{v} \in \mathcal{L}$ close to $\mathbf{c}$. Thanks to the knowledge of $\mathbf{B}$, the vector $\mathbf{s} = \mathbf{c} - \mathbf{v}$ is indeed short.
2. `Verif`$(\mathbf{A}, M, \mathbf{s}, \gamma)$ computes $\mathbf{m} = \mathtt{H}(M)$, then $\mathbf{c}' = \mathbf{As} \bmod q$. If $\mathbf{s}$ is a valid signature, it can be written $\mathbf{s} = \mathbf{c} - \mathbf{v}$, and then $\mathbf{c}' = \mathbf{Ac} = \mathbf{m} \bmod q$ because $\mathbf{v} \in \mathcal{L}$, so `Verif` first checks this property. But a valid signature is also short, so `Verif` additionally checks that $\|\mathbf{s}\|$ does not exceed a (public, fixed) bound γ. When these two checks are satisfied, the signature is accepted, else it is rejected.

Of course, the resulting scheme must be *efficient* in practice: consume the *least possible* bandwith while ensuring *practically fast* signing and verification timings. This implies that:

– `KeyGen` should give *compact* trapdoor pairs (that is, small bit representation);
– `Sample` is fast, and outputs short vectors; the shortness of the output depends on the *quality* of the trapdoor.
– `Sign` outputs the smallest (in bitsize) possible signature from these random vectors.

Lastly, note that intuitively, the shorter the output of `Sample` is, the harder it is for an attacker to forge a valid signature, so that only with the knowledge of a very good trapdoor pair should signatures be short vectors. Technicalities start when one realizes that the notion of "good" trapdoor highly depends on how `Sample` is instantiated. Optimizing this aspect is the main guiding principle of our design, and in GPV's instantiation in general.

3.2 Design of `KeyGen`

For the class of NTRU lattices, a trapdoor pairs is $(h, \mathbf{B}_{f,g})$, and Prest & Pornin showed that a completion (F, G) can be computed in $O(d \log d)$ time from f, g. In practice their implementation is as efficient as can be for this technical procedure: it is called `NtruSolve` in FALCON. Their algorithm only depends on the underlying ring and has now a stable version for $\mathbb{Z}[X]/(X^d + 1)$, where $d = 2^n$. We therefore reuse it in our design.

An important concern here is that not all pair $(f, g), (F, G)$ gives good trapdoor pairs for `Sample`. Schemes such as FALCON and MITAKA solve this technicality essentially by sieving among all possible bases to find the ones that reach an acceptable quality for the `Sample` procedure. This technique is costly, and many tricks were used to achieve an acceptable `KeyGen`. We *totally bypass* this sieving routine by redesigning completely how good quality bases can be found — see the next section for details. This improves the running time of `KeyGen`, and also increases the security offered by `Sample`. In any case, note that `NtruSolve`'s running time largely dominates the overall time for `KeyGen`: this is not avoidable as the basis completion algorithm require to work with quite large integers and relatively high-precision floating-point arithmetic.

At the end of the procedure, the secret key contains not only the secret basis, but also the necessary data for `Sign` and `Sample`. This additional information can be represented by elements in $K_{\mathbb{R}}$, and is computed during or at the end of `NtruSolve`. All-in-all, `KeyGen` outputs:

$$\begin{aligned}\mathsf{sk} &= (\mathbf{b}_1 = (f, g), \mathbf{b}_2 = (F, G), \widetilde{\mathbf{b}}_2 = (\widetilde{F}, \widetilde{G}), \Sigma_1, \Sigma_2, \beta_1, \beta_2),\\ \mathsf{pk} &= (h, q, \sigma_{\mathtt{sig}}, \eta),\end{aligned}$$

where we recall that $h = g/f \bmod q$. These parameters are described more thoroughly in Section 3.3 and 3.4. A list of their practical value is given in Table 1. Informally, they correspond to the following:

- $(f, g), (F, G)$ is a good basis of the lattice $\mathcal{L}_{\text{NTRU}}$ associated to h, with quality $\mathcal{Q}(f, g) = \alpha$, and $\widetilde{\mathbf{b}}_2$ is the Gram-Schmidt orthogonalization of (F, G) with respect to (f, g);
- $\sigma_{\mathtt{sig}}, \eta$ are respectively the standard deviation for signature vectors, and a tight upper bound on the "smoothing parameter of $\mathbb{Z}^d$";
- $\Sigma_1, \Sigma_2 \in K_{\mathbb{R}}$ represent covariance matrices for two intermediate Gaussian samplings in `Sample`;
- the vectors $\beta_1, \beta_2 \in K_{\mathbb{R}}^2$ represent the orthogonal projections from $K_{\mathbb{R}}^2$ onto $K_{\mathbb{R}} \cdot \mathbf{b}_1$ and $K_{\mathbb{R}} \cdot \widetilde{\mathbf{b}}_2$ respectively. In other words, they act as "getCoordinates" for vectors in $K_{\mathbb{R}}^2$. They are used by `Sample`, and are precomputed for efficiency.

Specifications of `KeyGen`: Algorithm 1 computes the necessary data for signature sampling, then outputs the key pair. Note that `NtruSolve` could also compute the sampling data and the public key, but for clarity, the pseudo-code gives these tasks to `KeyGen`. Figure 2 sketches the key generation procedure.

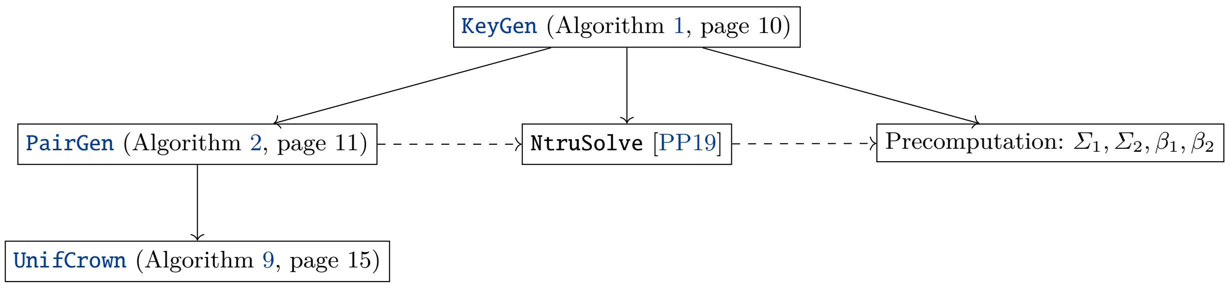
KeyGen (Algorithm 1, page 10)
PairGen (Algorithm 2, page 11)
NtruSolve [PP19]
Precomputation: $\Sigma_1, \Sigma_2, \beta_1, \beta_2$
UnifCrown (Algorithm 9, page 15)

Fig. 2: Flowchart of `KeyGen`.

The two subroutines `PairGen` (Algorithm 2) and `NtruSolve` are described below.

1. The `PairGen` algorithm generates d complex numbers $(x_j e^{i\theta_j})_{j \le d/2}, (y_j e^{i\theta_j})_{j \le d/2}$ to act as the FFT representations of two *real* polynomial $f^{\mathbb{R}}, g^{\mathbb{R}}$ in $K_{\mathbb{R}}$. The magnitude of these complex numbers are sampled in a planar annulus whose small and big radii are set to match a target $\mathcal{Q}(f, g)$ with `UnifCrown` (see also Section 3.6). It then finds close elements $f, g \in R$ by round-off, unless maybe the rounding-error was too large. When the procedure ends, it outputs a pair (f, g) such that $\mathcal{Q}(f, g) = \alpha$, where α depends on the security level.
2. `NtruSolve` is exactly Prest & Pornin's algorithm and implementation [PP19]. It takes as input $(f, g) \in R^2$ and a modulus q, and outputs $(F, G) \in R^2$ such that $(f, g), (F, G)$ is a basis of $\mathcal{L}_{\text{NTRU}}$ associated to $h = g/f \bmod q$. It does so by solving the Bézout-like equation $fG - gF = q$ in R using recursively the tower of subfields for optimal efficiency.

Specifications of `PairGen`: This algorithm is a new addition to schemes such as Mitaka. The ntru module associated to a public key h is morally a 2-dimensional object. When q is the modulus of the lattice and $(f, g), (F, G)$ is a basis, we have

$$\det \mathcal{M}_{\text{NTRU}} = (ff^* + gg^*)(\widetilde{F}\widetilde{F}^* + \widetilde{G}\widetilde{G}^*) = q^2 \in R,$$

Algorithm 1: KeyGen

```
Input: A modulus $q$, a target quality parameter $1 < \alpha$, parameters $\sigma_{\mathtt{sig}}, \eta > 0$
Output: A basis $((f, g), (F, G)) \in R^2$ of an NTRU lattice $\mathcal{L}_{\text{NTRU}}$ with $\mathcal{Q}(f, g) = \alpha$;
// Secret basis computation:
repeat
    $\mathbf{b}_1 := (f, g) \leftarrow$ PairGen$(q, \alpha, R_-, R_+)$;
until $f$ is invertible modulo $q$;
$\mathbf{b}_2 := (F, G) \leftarrow$ NtruSolve$(q, f, g)$;

// Public key data computation:
$h \leftarrow g/f \bmod q$;
$\gamma \leftarrow 1.1 \cdot \sigma_{\mathtt{sig}} \cdot \sqrt{2d}$;                    /* tolerance for signature length */

// Sampling data computation, in Fourier domain:
$\beta_1 \leftarrow \frac{1}{\langle \mathbf{b}_1, \mathbf{b}_1 \rangle_K} \cdot \mathbf{b}_1$;
$\Sigma_1 \leftarrow \sqrt{\frac{\sigma_{\mathtt{sig}}}{\langle \mathbf{b}_1, \mathbf{b}_1 \rangle_K} - \eta^2}$;
$\widetilde{\mathbf{b}}_2 := (\widetilde{F}, \widetilde{G}) \leftarrow \mathbf{b}_2 - \langle \beta_1, \mathbf{b}_2 \rangle \cdot \mathbf{b}_1$;
$\beta_2 \leftarrow \frac{1}{\langle \widetilde{\mathbf{b}}_2, \widetilde{\mathbf{b}}_2 \rangle_K} \cdot \widetilde{\mathbf{b}}_2$;
$\Sigma_2 \leftarrow \sqrt{\frac{\sigma_{\mathtt{sig}}}{\langle \widetilde{\mathbf{b}}_2, \widetilde{\mathbf{b}}_2 \rangle_K} - \eta^2}$;

sk $\leftarrow (\mathbf{b}_1, \mathbf{b}_2, \widetilde{\mathbf{b}}_2, \Sigma_1, \Sigma_2, \beta_1, \beta_2)$;
pk $\leftarrow (q, h, \sigma_{\mathtt{sig}}, \eta, \gamma)$;
return sk, pk;
```

the same way the area of a rectangle is the product of its length and width. This means that all the information about $\widetilde{F}\widetilde{F}^* + \widetilde{G}\widetilde{G}^*$ is determined by (f, g) and q, and this explains why *only* (f, g) is needed to know the quality of the basis found by `NtruSolve`. Recall from Section 2 that the quality is

$$\alpha := \mathcal{Q}(f, g) = \max_{1 \leq i \leq d/2} \max \left(\frac{|\varphi_i(f)|^2 + |\varphi_i(g)|^2}{q}, \frac{q}{|\varphi_i(f)|^2 + |\varphi_i(g)|^2} \right)^{1/2}.$$

Equivalently, a basis has quality α when for all $i \leq d/2$, we have

$$\frac{q}{\alpha^2} \leq |\varphi_i(f)|^2 + |\varphi_i(g)|^2 \leq \alpha^2 q. \tag{1}$$

As this is determined by the evaluations of the polynomials f, g, or equivalently, by their FFT representations, it is natural to sample directly these polynomials via their complex evaluations. This is handled by a subroutine `UnifCrown`, described in Section 3.6, Algorithm 9.

A hiccup is that inverting the `FFT` leads a priori only to real polynomials $f^{\mathbb{R}}, g^{\mathbb{R}} \in K_{\mathbb{R}}$. We thus have to round the coordinates of $f^{\mathbb{R}}, g^{\mathbb{R}}$ to their nearest integers which incurs a small error. We handle this error by sampling $f^{\mathbb{R}}, g^{\mathbb{R}}$ in FFT-format in a smaller annulus (or crown) with radii

$$R_- = \left(\frac{1}{\alpha} + \delta\right)\sqrt{q} \quad \text{and} \quad R_+ = (\alpha - \delta)\sqrt{q},$$

where δ is a small correction parameter to ensure that the rounding will stay in the correct crown $A(\sqrt{q}/\alpha, \alpha\sqrt{q})$ with large probability. In practice we take $\delta_{512} = 0.065$ and $\delta_{1024} = 0.3$, values that are set so that ≈ 80 tries in average for $d = 512$, resp. ≈ 5 tries in average for $d = 1024$, of the decoded polynomials (f, g) satisfy Equation 1.

Let us explain the determination of these values. The decoding errors can be accurately modelled as continuous 2-dimensional Gaussian vectors. This means that in average, the decoding makes an error proportional to the standard deviation of these Gaussians, which are heuristically identically and independently distributed. Thanks to the good concentration properties of Gaussians, we can fine-tune the tolerance with standard calculations, that match with experimental observations. This leads to `PairGen` below, where the target quality is by default $\alpha_{512} = 1.17$ and $\alpha_{1024} = 1.64$.

Algorithm 2: `PairGen`

```
Input: A modulus $q$, a target quality parameter $1 < \alpha$, two radii parameters $0 < R_- < R_+$
Output: A pair $(f, g)$ with $\mathcal{Q}(f, g) = \alpha$
for $i = 1$ to $d/2$ do
    $x_i, y_i \leftarrow$ UnifCrown$(R_-, R_+)$ ;                /* see Algorithm 9 */
    $\theta_x, \theta_y \leftarrow \mathcal{U}(0, 1)$;
    $\varphi_{f,i} \leftarrow |x_i| \cdot e^{2i\pi\theta_x}$;
    $\varphi_{g,i} \leftarrow |y_i| \cdot e^{2i\pi\theta_y}$;
end

$(f^{\mathbb{R}}, g^{\mathbb{R}}) \leftarrow \left(\text{FFT}^{-1}((\varphi_{f,i})_{i\le d/2}), \text{FFT}^{-1}((\varphi_{g,i})_{i\le d/2})\right)$;
$(\mathbf{f}, \mathbf{g}) \leftarrow (\lfloor f_i^{\mathbb{R}} \rceil)_{i\le d/2}, (\lfloor g_i^{\mathbb{R}} \rceil)_{i\le d/2}$;

$(\varphi(f), \varphi(g)) \leftarrow (\text{FFT}(\mathbf{f}), \text{FFT}(\mathbf{g}))$;
for $i = 1$ to $d/2$ do
    if $\alpha^2/q > |\varphi_i(f)|^2 + |\varphi(g)|^2$ or $\alpha^2 q < |\varphi_i(f)|^2 + |\varphi(g)|^2$ then
        restart;
    end
end
return $(\mathbf{f}, \mathbf{g})$;
```

3.3 Design of `Sign`:

Recall that NTRU lattices live in $\mathbb{R}^{2d}$. Their structure also helps to simplify the preimage computation. Indeed, the signer only needs to compute $\mathbf{m} = \mathtt{H}(M) \in \mathbb{R}^d$, as then $\mathbf{c} = (0, \mathbf{m})$ is a valid preimage: the corresponding polynomials satisfy $(h, 1) \cdot c = m$.

Another interesting feature is that only the *first half* of the signature $(\mathbf{s}_1, \mathbf{s}_2) \in \mathcal{L}_{\text{NTRU}}$ needs to be sent along the message, as long as h is available to the verifier. This comes from the identity $hs_1 = s_2 \bmod q$ defining these lattices, as we will see in the `Verif` algorithm description. [8]

Because of their nature as Gaussian integer vectors, signatures can be encoded to reduce the size of their bit-representation. The standard deviation of `Sample` is large enough so that the $\lfloor \log \sqrt{q} \rfloor$ least significant bits of one coordinate are essentially random. In FALCON, the most significant bits (MSB's) of each coordinate are then sent through an unary (or Huffman) encoding together with their corresponding tails, which already save a nice portion of bandwith. This generic approach is summed-up as the `Compress` and `Decompress` functions in Section 3.6.

Following [ETWY22], we can go a step further by noticing that instead of encoding each sets of MSB's separately, we can *batch*-encode them as a whole using for example Asymmetric Numeral Systems (ANS). Our signatures then enjoy an additional $7-12\%$ compression factor. This will be implemented in future version of SOLMAE.

In the description above, the scheme cannot output two different signatures for a message. This well-known concern of the GPV framework can be addressed in several ways, for example making a stateful scheme or by hash randomization. Like FALCON, we chose the latter solution for efficiency purpose. In practice, `Sign` adds a random "salt" $r \in \{0,1\}^k$, where k is large enough that an unfortunate collision of messages is unlikely to happen, that is, it hashes $(r||M)$ instead of M — our analysis in this regard is identical to FALCON. A signature is then $\mathtt{sig} = (r, \mathtt{Compress}(s_1))$.

Figure 3 sketches the signing procedure.

Specifications of `Sign`: The signing algorithm `Sign` handles the hash and the Fourier conversions, leaving the important task of generating the signature vectors to `Sample`. The hash function is instantiated as explained in Section 3.6. It also generates the salt $r \in \{0,1\}^k$; the length of the salt is obtained by standard conversions from a conservative target bit-security and the maximum number of queries $q_s = 2^{64}$ as $k = 320 = 256 + \log q_s$.

The parameter η is selected as a tight upper bound on *the smoothing parameter* $\eta_\epsilon(\mathbb{Z}^d)$, for some $\epsilon > 0$. Informally, this parameter quantifies the amount of noise to smooth out the discreteness

[8] The same identity can also be used to check the validity of signatures only with a hash of the public key h, requiring this time send both $\mathbf{s}_1$ and $\mathbf{s}_2$, but we will not consider this setting in this documentation.

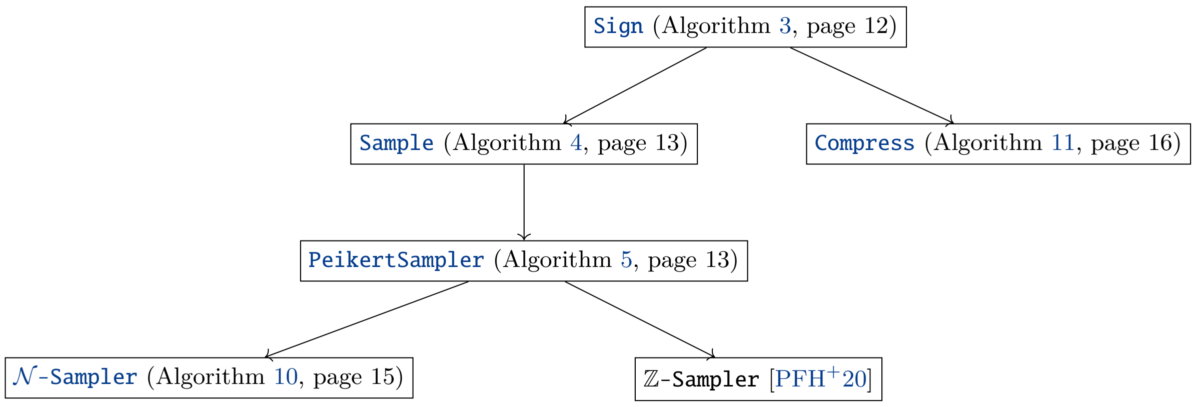
Sign (Algorithm 3, page 12)
Sample (Algorithm 4, page 13)
Compress (Algorithm 11, page 16)
PeikertSampler (Algorithm 5, page 13)
$\mathcal{N}$-Sampler (Algorithm 10, page 15)
$\mathbb{Z}$-Sampler [PFH+20]

Fig. 3: Flowchart of Sign.

of lattice Gaussian vectors. The value of ϵ is deduced following [Pre17, EFG+22] and nowadays standard Renyi divergence arguments. The corresponding value for η is then

$$\epsilon = 2^{-41} \quad \text{and} \quad \eta = \frac{1}{\pi}\sqrt{\frac{1}{2} \cdot \log\left(2d\left(1 + \frac{1}{\epsilon}\right)\right)}.$$

Concretely, we have $\eta_{512} \approx 1.338$ and $\eta_{1024} \approx 1.351$. The output is a Gaussian vector $\mathbf{v} \in \mathcal{L}_{\text{NTRU}}$ centered at $\mathbf{c} = (0, \mathtt{H}(r||M))$; the parameter $\sigma_{\mathtt{sig}}$ determines the expected distance from $\mathbf{v}$ to its center, and therefore drives the hardness of forgery. Its value is determined from [Pre15, EFG+22]:

$$\sigma_{\mathtt{sig}} = \eta \cdot \mathcal{Q}(f, g) \cdot \sqrt{q}.$$

Algorithm 3: Sign

```
Input: A message $M \in \{0,1\}^*$, a tuple $\mathbf{sk} = ((f,g),(F,G),(\widetilde{F},\widetilde{G}),\sigma_{\mathtt{sig}},\Sigma_1,\Sigma_2,\eta)$, a
        rejection parameter $\gamma > 0$.
Output: A pair $(r, \mathtt{Compress}(\mathbf{s}_1))$ with $r \in \{0,1\}^{320}$ and $\|(\mathbf{s}_1,\mathbf{s}_2)\| \le \gamma$.
$r \leftarrow \mathcal{U}(\{0,1\}^{320})$;
$\mathbf{c} \leftarrow (0, \mathtt{H}(r||M))$;
$\hat{\mathbf{c}} \leftarrow \mathsf{FFT}(\mathbf{c})$;
repeat
    $(\hat{s}_1, \hat{s}_2) \leftarrow \hat{\mathbf{c}} - \mathtt{Sample}(\hat{\mathbf{c}}, \mathbf{sk})$;
    // $(\mathbf{s}_1, \mathbf{s}_2) \leftarrow D_{\mathcal{L}_{\text{NTRU}},\mathbf{c},\sigma_{\mathtt{sig}}}$
until $\|(\hat{s}_1, \hat{s}_2)\|^2 \le \gamma^2$;
$s_1 \leftarrow \mathsf{FFT}^{-1}(\hat{s}_1)$;
$s \leftarrow \mathtt{Compress}(s_1)$;
return $(r, s)$;
```

The sampling of the signature vector in Algorithm 3 is a cascade of different sampling algorithms, Sample, PeikertSampler, and at the deepest level, $\mathbb{Z}$-Sampler. Their specifications follow.

3.4 Design of Sample:

Our second main change from Falcon is to rely on the *hybrid sampler* for the Sample procedure. To make sense of this change, we give the briefest possible outline of Lattice Gaussian sampling — as the sampling of signature ultimately is the whole core of the entire "fast-GPV" design, this is anyway a required paragraph.

Such algorithms are usually seen as *randomized decoding procedure.* Babai described two well-known lattice decoding algorithms: the round-off and the nearest-plane. Staying informal, the first just round coordinates of the target in the lattice basis to their nearest integer, while the second is more fine-grained and round the *Gram-Schmidt* coordinates of the target iteratively, correcting the deviation from the original lattice basis with each decoding step. By randomizing the rounding using Gaussian integers, one obtains either Peikert's sampler [Pei10] or Klein's sampler [Kle00, GPV08]. The former is more efficient over NTRU lattices, but suffer from a lack of good trapdoors. In other words, signatures are longer with this approach. The latter compensates its lack of efficiency by large sets of very good trapdoors, and therefore almost optimal signature lengths. As we mentioned, if one accepts a delicate and practically unmaskable implementation involving tree data structures, this drawback can be mitigated.

The hybrid sampler stands in-between, both in efficiency and signature lengths: the Gram-Schmidt orthogonalization and randomized decoding are both done *at the ring level*, so that the algorithm can be seen as a combination of Klein and Peikert's approach. In particular, only two decoding steps are done for NTRU lattices, compared to $2d$ for the FFO sampler, and the randomization is handled by Peikert's algorithm in the ring, which runs in quasi-linear time: this explains its overall better efficiency. On its first analysis [Pre15], it seemed that good trapdoor for this sampler were costlier to find. Thanks to our new `KeyGen` algorithm, this is no more a concern, and we are now able to enjoy all the pros that the hybrid sampler brings versus the FFO sampler: it is way simpler to implement, it is more efficient, it is more friendly to parallelization. Moreover, its masking can be done with standard, well-understood techniques [EFG+22] and online-offline approaches, since the tree structure is now entirely avoided.

Specifications of `Sample`: `Sample` is an instantiation of the hybrid sampler [Pre15, EFG+22], and can be seen as 2-step randomized decoding, with randomization happening *at the ring level*, or morally, in a d-dimensional space. In Algorithm 4 below, note that *all operations are done in Fourier domain.*

Algorithm 4: `Sample`

```
Input: A target $\mathbf{c} = (0, \mathbf{c}') \in K_{\mathbb{R}}^2$, a tuple
        $\mathsf{sk} = (\mathbf{b}_1 = (f, g), \mathbf{b}_2 = (F, G), \widetilde{\mathbf{b}_2} = (\widetilde{F}, \widetilde{G}), \sigma_{\mathtt{sig}}, \Sigma_1, \Sigma_2, \beta_1, \beta_2)$.
Output: A vector $\mathbf{v} \in \mathcal{L}_{\text{NTRU}}$ with distribution statistically close to $D_{\mathcal{L}_{\text{NTRU}}, \mathbf{c}, \sigma_{\mathtt{sig}}}$.
$\mathbf{t} \leftarrow \mathbf{c}, \mathbf{v} \leftarrow \mathbf{0}$;
for $i = 2$ to $1$ do
    $t_i \leftarrow \langle \beta_i, \mathbf{t} \rangle_K$;
    $z_i \leftarrow$ PeikertSampler$(t_i, \Sigma_i, \eta)$ ;          /* $z_i \leftarrow D_{R, t_i, \frac{\sigma_{\mathtt{sig}}}{\langle \widetilde{\mathbf{b}_i}, \widetilde{\mathbf{b}_i} \rangle}}$ */
    $\mathbf{t} \leftarrow \mathbf{t} - z_i \mathbf{b}_i, \mathbf{v} \leftarrow \mathbf{v} + z_i \mathbf{b}_i$;
end
return $\mathbf{v}$;
```

The randomization is done by a call to `PeikertSampler` (see Algorithm 5), which outputs elements in R with a Gaussian distribution of suitable covariance matrices. In Fourier domain, these covariance matrices are diagonal, and can then be represented by a vector of complex numbers. From Section 3.2, we view $\sigma_{\mathtt{sig}}$ as a constant in $K_{\mathbb{R}}$, and the intermediate standard deviation parameters are

$$\Sigma_i = \sqrt{\frac{\sigma_{\mathtt{sig}}}{\langle \widetilde{\mathbf{b}_i}, \widetilde{\mathbf{b}_i} \rangle} - \eta^2} \in K_{\mathbb{R}}.$$

By choice of $\sigma_{\mathtt{sig}}$, they correspond in Fourier domain to positive definite matrices, actually diagonal with entries the embeddings of Σ_i. The square roots are then easily computed coordinate-wise.

Specifications of `PeikertSampler`: Again, input, output as well as arithmetic operations are all done in Fourier representation. The principle of `PeikertSampler` is a perturbation-based sampling. This translates concretely as a *continuous, elliptic* Gaussian sampling which is easy to handle in Fourier domain with enough precision, combined with a *spherical, discrete* Gaussian sampling of width η, showcased in Algorithm 5. The latter is handled by d calls to $\mathbb{Z}$-`Sampler`.

Algorithm 5: PeikertSampler

```
Input: A target $t \in K_{\mathbb{R}}$, parameters $\Sigma, \eta \in K_{\mathbb{R}}^{++}$.
Output: A vector $\mathbf{v} \in R$ with distribution statistically close to $D_{R,t,\sigma}$, where $\sigma = \sqrt{\Sigma^2 + \eta^2}$.
$p \leftarrow \Sigma \cdot \mathcal{N}_1^{K_{\mathbb{R}}}$ ;          /* $p \leftarrow \mathcal{N}_{\Sigma^2}^{K_{\mathbb{R}}}$, done with $\mathcal{N}$-Sampler (Algorithm 10) */
$(p_1, \ldots, p_d) \leftarrow$ FFT$^{-1}(p)$ ;                                        /* $(p_i)_i \in \mathbb{R}^d$ */
$(t_1, \ldots, t_d) \leftarrow$ FFT$^{-1}(t)$ ;                                        /* $(t_i)_i \in \mathbb{Z}^d$ */
for $i = 1$ to $d$ do
    $x_i \leftarrow \mathbb{Z}$-Sampler$(t_i - p_i, \eta)$;
end
return FFT$(x_1, \ldots, x_d)$;
```

By definition of the parameters, the calls to Algorithm 5 in Algorithm 4 output two ring elements with respective covariance $\sigma_{\texttt{sig}}/\langle \widetilde{\mathbf{b}}_i, \widetilde{\mathbf{b}}_i \rangle_K$, in Fourier representation.

Specifications of $\mathbb{Z}$-Sampler: This step is surprisingly delicate. We reuse the ingenious method of Falcon, and refer to their documentation [PFH+20] for the details about the parameters. Below, we give only an informal description of the necessary steps based on [ZSS20, HPRR20].

The first concern is the necessity to sample around an arbitrary center $c \in \mathbb{R}$. A technique is to first identify it to its fractional part $\{c\} \in [0, 1)$, and to then use a rejection approach to allow a sampling *centered around* 0 instead. This implies computations of the rejection probability, which involve the exponential function and must be done efficiently at a high enough floating-point precision. Next, the sampling uses a Cumulative Distribution Table (CDT) approach (also known as an inversion-based sampler). Space is saved by using the CDT of a *half* Gaussian distribution combined to a Bernoulli sampler to obtain the sign of the output.

3.5 Design of Verif:

The last step of the scheme is thankfully simpler to describe. Upon receiving a signature (r, s) and message M, the verifier decompresses s to a polynomial s_1 and $\mathbf{c} = (0, \mathtt{H}(r||M))$, then wants to recover the full signature vector $\mathbf{v} = (s_1, s_2)$. If $\mathbf{v}$ is a valid signature, the verification identity is $(h, -1) \cdot (\mathbf{c} - \mathbf{v}) = -\mathtt{H}(r||M) - hs_1 + s_2 \bmod q = 0$, or equivalently the verifier can compute

$$s_2 = \mathtt{H}(r||M) + hs_1 \bmod q.$$

This is computed in the ring R_q, and can be done very efficiently for a good choice of modulus q using the Number Theoretic Transform (NTT). We currently follow the standard choice (like in Falcon) of $q = 12289$, as the multiplication in NTT format amounts to d integer multiplications in $\mathbb{Z}/q\mathbb{Z}$. In future versions, we will consider different trade-offs between verification efficiency and public-key size. The last step is to check that $\|(\mathbf{s}_1, \mathbf{s}_2)\|^2 \leq \gamma^2$: the signature is only accepted in this case.

The rejection bound γ comes from the expected length of vectors outputted by Sample. Since they are morally Gaussian, they concentrate around their standard deviation; a "slack" parameter $\tau = 1.042$ is tuned to ensure that 90% of the vectors generated by Sample will get through the loop:

$$\gamma = \tau \cdot \sigma_{\texttt{sig}} \cdot \sqrt{2d}.$$

3.6 Miscellanous

In this section we define several supporting algorithms invoked by KeyGen, Sign, and Verif.

Algorithm 6: Verif

```
Input: A signature $(r, s)$ on $M$, a public key pk $= h$, a bound $\gamma$.
Output: Accept or reject.

$s_1 \leftarrow$ Decompress$(s)$;
$c \leftarrow \mathtt{H}(r||M)$;
$s_2 \leftarrow c + hs_1 \bmod q$;
if $\|(\mathbf{s}_1, \mathbf{s}_2)\|^2 > \gamma^2$ then
  | return Reject.
end
return Accept.
```

Specifications of FFT and FFT^{-1}: Algorithm 7 and Algorithm 8 respectively show the computation of FFT and its inverse over $K_\mathbb{R}$.

Algorithm 7: FFT

```
Input: $f \in K_\mathbb{R} = \mathbb{R}[X]/(X^d + 1)$.
Output: the FFT representation of $f$

$\zeta = \exp(i\pi/d)$;
$\varphi(f) \leftarrow (f(\zeta^1), f(\zeta^3), \cdots, f(\zeta^{2d-1}))$;
return $\varphi(f)$
```

Algorithm 8: FFT^{-1}

```
Input: $\mathbf{c} = (c_0, \cdots, c_{d-1}) \in \mathbb{C}^d$ such that $c_{d-1-i} = \overline{c_i}$.
Output: $f \in K_\mathbb{R}$ such that $\mathbf{c} = \varphi(f)$

$\zeta = \exp(i\pi/d)$;
$\mathbf{V} = \left(\overline{\zeta^{jk}}\right)_{j \in \mathbb{Z}_d, k \in \mathbb{Z}_{2d}^*}$;
$f \leftarrow \frac{1}{d} \cdot \mathbf{Vc}$;
return $f$
```

Specifications of UnifCrown: Algorithm 9 below outputs a uniformly random element in a fixed planar annulus $A(R_-, R_+) = \{(x, y) \in \mathbb{R}^2 \ : \ R_-^2 \leq x^2 + y^2 \leq R_+^2\}$, from uniformly random numbers in $(0, 1)$ — that is, uniform in the set of floating point numbers with given mantissa and exponent inside $(0, 1)$.

Algorithm 9: UnifCrown

```
Input: Parameters $0 < R_- < R_+$.
Output: A point $(x, y)$ with uniform distribution in $A(R_-, R_+)$

$u_\rho, u_x, u_y \leftarrow \mathcal{U}(0, 1)$;
$\rho \leftarrow \sqrt{R_-^2 + u_\rho(R_+^2 - R_-^2)}$;
$x \leftarrow \rho \cdot \cos(2\pi u_x)$;
$y \leftarrow \rho \cdot \sin(2\pi u_y)$;
return $(x, y)$
```

Specifications of $\mathcal{N}$-Sampler: In Algorithm 5, the first step is the sampling of a continuous Gaussian perturbation with some elliptic covariance E. If Σ is a matrix such that $\Sigma^t \Sigma = E$, then $\mathcal{N}_E = \Sigma \cdot \mathcal{N}_1$. As mentioned previously, in FFT domain the target covariance matrix is diagonal with positive entries: we have $\Sigma = \sqrt{E}$, where the square root is taken entry-wise on the diagonal.

Hence this step of `PeikertSampler` boils down to the well-known sampling of a normal variate. We carry out this step by Box-Müller's approach, which we recall below for cross-referencing purposes.

Algorithm 10: $\mathcal{N}$-`Sampler`

```
Input: The degree $d$ of $R$.
Output: Two variables $x, y$ with distribution $\mathcal{N}_d$
$u_\rho, u_\theta \leftarrow \mathcal{U}(0,1)$;
$\rho \leftarrow \sqrt{-2d \ln u_\rho}$;
$x \leftarrow \rho \cdot \cos(2\pi u_\theta)$;
$y \leftarrow \rho \cdot \sin(2\pi u_\theta)$;
return $(x, y)$
```

Specifications of `Compress` and `Decompress`: We reuse the same method as in FALCONto encode and decode a Gaussian vector. For completeness, we describe compression and decompression functions in Algorithm 11 and Algorithm 12 respectively. Note that $slen = 8 \cdot |sgn| - 320$ by default where $|sgn|$ denotes the signature size in bytes. Again, we can use improved encoding technique suggested in [ETWY22] to further reduce the signature size and this will be implemented in future version of SOLMAE.

Algorithm 11: `Compress`

```
Input: A polynomial $s = \sum_{i=0}^{d-1} s_i X^i \in R = \mathbb{Z}[X]/(X^d + 1)$ and an integer $slen$.
Output: A compressed representation of $str$ of $s$ of bitsize $slen$, or $\perp$
$str \leftarrow \{\}$;
for $i = 0$ to $d - 1$ do
    $str \leftarrow (str \,||\, b)$ where $b = 1$ if $s_i < 0$, $b = 0$ otherwise;
    $str \leftarrow (str \,||\, b_6 b_5 \cdots b_0)$ where $b_j = (|s_i| \gg j)\&0x1$;
    $k \leftarrow |s_i| \gg 7$;
    $str \leftarrow (str \,||\, 0^k 1)$
end
if $|str| > slen$ then
    $str \leftarrow \perp$;
end
else
    $str \leftarrow (str \,||\, 0^{slen - |str|})$
end
return $str$
```

3.7 List of parameters

For clarity, we sum-up all the concrete values for the parameters described in this section in Table 1.

4 Preliminary performance analysis

4.1 Description of platform

Our implementation has been tested on various x86–64 platforms, and consistently outperforms FALCON in signing and verification in equal dimension, while key generation is slightly slower. Timings below have been collected on a single core of a Ryzen Threadripper Pro 5975WX @ 3.60 GHz workstation with hyperthreading and frequency scaling disabled.

Algorithm 12: `Decompress`

```
Input:  A bitstring $str$ of bitsize $slen$
Output: A polynomial $s = \sum_{i=0}^{d-1} s_i X^i \in R = \mathbb{Z}[X]/(X^d + 1)$ or $\perp$

if $|str| \neq slen$ then
    return $\perp$;
end
for $i = 0$ to $d - 1$ do
    $s'_i \leftarrow \sum_{j=0}^{6} 2^{6-j} str[1 + j]$;
    $k \leftarrow 0$;
    while $str[8 + k] = 0$ do
        $k \leftarrow k + 1$
    end
    $s_i \leftarrow (-1)^{str[0]} \cdot (s'_i + 2^7 k)$;
    if $s_i = 0$ and $str[0] = 1$ then
        return $\perp$
    end
    $str \leftarrow str[9 + k : ]$
end
if $|str| \neq 0^{|str|}$ then
    return $\perp$;
end
return $s = \sum_{i=0}^{d-1} s_i X^i$
```

Table 1: List of parameters for SOLMAE

	SOLMAE–512	SOLMAE–1024
ring degree d	512	1024
dimension $2d$	1024	2048
modulus q	12289	12289
salt length k	320	320
smoothing η	1.338	1.351
smoothness ϵ	2^{-41}	2^{-41}
quality α	1.17	1.64
correction δ	0.065	0.3
lower radius R_-	101.95	100.85
upper radius R_+	122.49	148.54
signature width $\sigma_{\mathtt{sig}}$	173.54	245.62
slack τ	1.04	1.04
rejection bound γ^2	33870790	134150669

4.2 Performance of our reference implementation

Our current implementation displays the performance below. Falcon performance numbers are also provided using the `speed` tool included in the official code archive; the tool does not include cycle counts, so they are omitted in Table 2.

Observe that our new `KeyGen` is fairly close to Falcon's in terms of speed, and as mentioned in Section 3, this is obtained while *increasing* the security level of SOLMAE compared to that of Mitaka. For signing, SOLMAE consistently outperforms Falcon by a factor of about 2, and for verification, SOLMAE is also about 50% faster.

Table 2: Performance comparison between SOLMAE and Falcon.

		SOLMAE–512	SOLMAE–1024	Falcon–512	Falcon–1024
`KeyGen` time	Mcycles	27	65	—	—
	time (ms)	7.5	18	5.0	15
`pk` size	Bytes	896	1792	896	1792
`Sign` time	kcycles	387	775	—	—
	time (μs)	108	216	220	441
`sgn` size	kBytes	666	1375	666	1280
`Verif` time	kcycles	40	84	—	—
	time (μs)	11	23	18	36

Under the low-speed computing environment, Intel(R) Core(TM) i7-8550U CPU@1.80GHz 8.00GB RAM, we have executed the performance check of our reference implementation without compression/decompression for SOLMAE–512 and SOLMAE–1024 whose C-src codes are attached in our submission package to KpqC competition.

For this test, the input messages are chosen 1,024 byte randomly per 10,000 times with each count using different key pairs. The average clock cycle and time (μs) during `KeyGen`, `Sign` and `Verif` using SOLMAE–512 and SOLMAE–1024 are shown in Table 3.

5 Security

5.1 Model for lattice reduction

In all of the following, we follow the so-called *Geometric series assumption* (GSA), asserting that a reduced basis sees its Gram-Schmidt vectors' norm decrease with geometric decay. More formally, it can be instantiated as follows for self-dual BKZ (DBKZ) reduction algorithm of Micciancio and

Table 3: Average performance per each step of SOLMAE–512 and SOLMAE–1024

	SOLMAE–512		SOLMAE–1024	
`KeyGen`	26,336,721.9	13,231.4	56,381,295.8	28,301.3
`Sign`	499,836.9	244.2	975,022.5	491.9
`Verif`	35,427.8	15.0	69,530.2	35.6

Walter [MW17]: an output basis $(\mathbf{b}_1, \ldots, \mathbf{b}_n)$ yielded by DBKZ algorithm with block size β on a lattice $\mathcal{L}$ of rank n satisfies

$$\|\mathbf{b}_i^*\| = \delta_\beta^{d-2(i-1)} \det(\mathcal{L})^{\frac{1}{n}}, \quad \text{where} \quad \delta_\beta = \left(\frac{(\pi\beta)^{\frac{1}{\beta}} \cdot \beta}{2\pi e}\right)^{\frac{1}{2(\beta-1)}},$$

for $\mathbf{b}_i^*$ being the i-th Gram Schmidt vector of the basis.

5.2 Key recovery attack

The key recovery consists in finding the private secret key (i.e. $f, g \in \mathcal{R}^2$) from the sole data of the public elements q and h. The most powerful attacks are up-to-our-knowledge realized through lattice reduction. It consists in constructing the algebraic lattice over $\mathcal{R}$ spanned by the vectors $(q, 0)$ and $(h, 1)$ (i.e. the public basis of the NTRU key) and retrieve the lattice vector $\mathbf{s} = (\mathbf{g}, \mathbf{f})$ among all possible lattice vectors of norm bounded by $\|\mathbf{s}\| = \sqrt{2d}\sigma$ (or a functionally equivalent vector, for instance $(\mu g, \mu f)$ for any unit μ of the number field).

We make use of the so-called *projection trick* to avoid enumerating over all the sphere of radius $\sqrt{2d}\sigma$ (which contains around $\left(\frac{2d\sigma^2}{q}\right)^d$ vectors under the Gaussian heuristic). More precisely we proceed as follows. Set β to be the block size parameter of the DBKZ algorithm and start by reducing the public basis with this latter algorithm. Call $[\mathbf{b}_1, \ldots, \mathbf{b}_{2d}]$ the resulting vectors. Then if we can recover the *projection* of the secret key onto $\mathcal{P}$, the orthogonal space to $\mathrm{Span}(\mathbf{b}_1, \ldots, \mathbf{b}_{2d-\beta-1})$, then we can retrieve in polynomial time the full key by *Babai nearest plane* algorithm to lift it to a lattice vector of the desired norm. Hence it suffices to be able find the projection of the secret key among the shortest vector of the lattice generated by the last β vectors projected onto $\mathcal{P}$. Classically, sieving on this projected lattice will recover all vectors of norm smaller than $\sqrt{\frac{4}{3}}\ell$, where ℓ is the norm of the $2d-\beta$-th Gram-Schmidt vector $\mathbf{b}_{2d-\beta}^*$ of the reduced basis. Under the GSA, we have:

$$\ell = \sqrt{q}\delta_\beta^{-2d+2\beta+2} \approx \left(\frac{\beta}{2\pi e}\right)^{1-\frac{d}{\beta}}.$$

Moreover, considering that $\mathbf{s}$ behaves as a random vector of norm $\sqrt{2d}\sigma$, and using the GSA to bound the norm of the Gram-Schmidt vectors $[\mathbf{b}_1^*, \ldots, \mathbf{b}_{2d-\beta}^*]$, that the norm of its projection over $\mathcal{P}$ is roughly

$$\sqrt{\frac{\beta}{2d}}\|\mathbf{s}\| = \beta^{\frac{1}{2}}\sigma.$$

Hence, we will retrieve the projection among the sieved vectors if $\beta^{\frac{1}{2}}\sigma \leq \sqrt{\frac{4}{3}}\ell$, that is if the following condition is fulfilled:

$$\sigma^2 \leq \frac{4q}{3\beta}\delta_\beta^{4(\beta+1-d)} \tag{2}$$

5.3 Signature forgery by reduction to Approx-CVP.

As a Hash-and-Sign paradigm signature, forging a signature stems to feeding a lattice point $\mathbf{v}$ at a bounded distance from a random space point $\mathbf{x}$. This Approx-CVP problem can be solved using the so-called *Nearest-Cospace* framework developed in [EK20]. Under the Geometric Series assumption, Theorem 3.3 of [EK20] states that under the condition: $\|\mathbf{x} - \mathbf{v}\| \leq \left(\delta_\beta^{2d} q^{\frac{1}{2}}\right)$, the decoding can be done in time Poly(d) calls to a CVP oracle in dimension β.

As mentioned in [CPS+20] a standard optimization of this attack consists only considering the lattice spanned by a subset of the vectors of the public basis and perform the decoding within this sublattice. The only interesting subset seems to consists in forgetting the $k \leq n$ first vectors. The dimension is of course reduced by k, at the cost of working with a lattice with covolume $q^{\frac{k}{2(2d-k)}}$ bigger. Henceforth the global condition of decoding becomes the (slightly more general) inequality $\|\mathbf{x} - \mathbf{v}\| \leq \min_{k \leq d}\left(\delta_\beta^{2d-k} q^{\frac{d}{2d-k}}\right)$ As such, we need to enforce the condtion:

$$\gamma \geq \min_{k \leq d}\left(\delta_\beta^{2d-k} q^{\frac{d}{2d-k}}\right) \tag{3}$$

5.4 On the other attacks on SOLMAE

In this section, we list the other possible type of attacks on the signature, which are nonetheless irrelevant for the set of parameters we are using.

Algebraic attacks As remarked in the design of NTRU-based schemes (such as for instance Falcon or ModFalcon signatures), there exists a rich algebraic structure in the modules over the convolution ring $\mathcal{R}$ used in Solmae. However, there is no known way to improve all the algorithms previously mentioned with respect to their general lattice equivalent by more than polynomial factors (see for instance the speedup on lattice reduction of [KEF20]).

Overstretched NTRU-type As observed in [KF17], when the modulus q is significantly larger than the magnitudes of the NTRU secret key coefficients, the attack on the key based on lattice reduction recovers the secret key better than the results presented above. This so-called "overstretched NTRU" parameters occurs when $q > (2d)^{2.83}$ for binary secrets, implying that, as it is the case for Falcon and other NTRU based NIST candidates, that even *very* significant improvements of this attack would still be irrelevant for the security of the scheme.

Hybrid attacks Odlyzko's meet on the middle attack, or more recently the hybrid attack of Howgrave-Graham [How07] which combines a meet-in-the-middle algorithm with a key recovery by lattice reduction were used effectively against NTRU, mainly due to its design using sparse polynomials. As it is not the case (secrets are dense elements in the ring $\mathcal{R}$), their impact is not sufficient to be a problem on the parameter selection of SOLMAE.

5.5 Concrete security

In order to assess the concrete security of our signature scheme, we proceed using the usual cryptanalytic methodology of estimating the complexity of the best attacks against *key recovery attacks* on the one hand, and *signature forgery* on the other.

The analyses translate into concrete bit-security estimates following the methodology of NewHope [ADPS16], sometimes called "core-SVP methodology". In this model [BDGL16, Laa16], the bit complexity of lattice sieving (which is asymptotically the best SVP oracle) is taken as $\lfloor 0.292\beta \rfloor$ in the classical setting and $\lfloor 0.265\beta \rfloor$ in the quantum setting in dimension β.

The resulting security in terms of the sampling quality α is given in Fig. 4 in dimensions 512 and 1024.

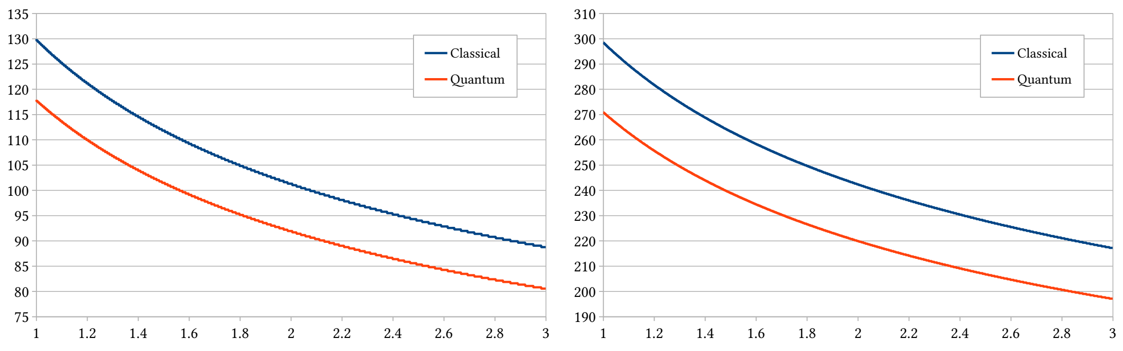

Fig. 4: Security (classical and quantum) against forgery as a function of the quality $1 \leq \alpha \leq 3$ of the lattice sampler (left: dimension 512 and right: dimension 1024).

Table 4: Security level for SOLMAE
(C is classical security, Q is quantum security.)

	SOLMAE−512	SOLMAE−1024
bit security (C/Q)	127/115	256/232
NIST equivalent	NIST-I	NIST-V

5.6 Side-Channel Resilience

As is the case with Falcon, the difficulty of realizing constant-time implementation lies in the use of floating-point arithmetic. A potential approach to address the problem would be to rely on the work of Pornin [Por19], which has successfully presented constant-time implementation of floating-point arithmetic by either benefiting from dedicated floating-point hardware when available, or otherwise by emulating floating-point with only integer operations. At the core of `PeikertSampler` is Gaussian sampling over the integers ($\mathbb{Z}$-`Sampler`). We recommend implementing this step by following the isochronous sampler of Howe et al. [HPRR20] Their version of $\mathbb{Z}$-`Sampler` essentially invokes a base Gaussian sampler that samples an element with the fixed half Gaussian distribution (which can be made constant-time by naively going through all entries in the CDT) and then rejects that element with certain probability in such a way that the rejection rate leaks no information about the secret. We defer the details to Section 4 of [HPRR20].

6 Summary or Conclusion

This document summarizes the SOLMAE signature scheme submitted to the Korean Post-Quantum Comptetition. SOLMAE is a lattice-based signature scheme following the hash-and-sign paradigm (in the style of Gentry–Peikert–Vaikuntanathan signatures), and instantiated over NTRU lattices. In that sense, it is closely related to, and a successor of, several earlier schemes including Ducas–Lyubashevsky–Prest (DLP), Falcon and Mitaka. More precisely, SOLMAE offers the “best of both worlds” between Falcon and Mitaka.

Falcon has the advantage of providing short public keys and signatures (offering essentially the best bandwidth trade-off among post-quantum constructions) as well as high security levels; however, it is plagued by a contrived signing algorithm that makes it very difficult to implement correctly, not very fast for signing and hard to parallelize; it also has very little flexibility in terms of parameter settings. In contrast, Mitaka is much simpler to implement, twice as fast in equal dimension, straightforward to parallelize and fully versatile in terms of parameters; however, it has lower security than Falcon in equal dimension, has an even more contrived key generation algorithm that tends to be quite slow, and has somewhat larger keys and signatures at equivalent security levels.

We can conclude that SOLMAE solves the conundrum of choosing between those two schemes by offering all the advantages of both. It uses the same simple, fast, parallelizable signing algorithm as Mitaka, with flexible parameters. However, by leveraging a novel key generation algorithm that is much faster and achieves higher security, SOLMAE achieves the same high security and short key and signature sizes as Falcon. It is also compatible with recently introduced ellipsoidal lattice Gaussian sampling techniques to further reduce signature sizes. This makes SOLMAE the state-of-the-art in terms of constructing efficient lattice-based signatures over structured lattices.

Some challenges are left to do next:

– Implementation of intermediate NIST security level from II to IV
– Implementation of compression and decompression to reduce the size of signature
– Optimized Implementation on various platform
– Backup documents to understand the underlying theory of SOLMAE, *etc.*

References

ADPS16. E. Alkim, L. Ducas, T. Pöppelmann, and P. Schwabe. Post-quantum key exchange - A new hope. In *USENIX Security 2016*, pp. 327–343. USENIX Association, 2016. 20

BDF$^+$11. D. Boneh, Ö. Dagdelen, M. Fischlin, A. Lehmann, C. Schaffner, and M. Zhandry. Random oracles in a quantum world. In *ASIACRYPT 2011*, vol. 7073 of *LNCS*, pp. 41–69. Springer, Heidelberg, 2011. 6

BDGL16. A. Becker, L. Ducas, N. Gama, and T. Laarhoven. New directions in nearest neighbor searching with applications to lattice sieving. In *27th SODA*, pp. 10–24. ACM-SIAM, 2016. 20

CPS$^+$20. C. Chuengsatiansup, T. Prest, D. Stehlé, A. Wallet, and K. Xagawa. ModFalcon: Compact signatures based on module-NTRU lattices. In *ASIACCS 20*, pp. 853–866. ACM Press, 2020. 19

DLP14. L. Ducas, V. Lyubashevsky, and T. Prest. Efficient identity-based encryption over NTRU lattices. In *ASIACRYPT 2014, Part II*, vol. 8874 of *LNCS*, pp. 22–41. Springer, Heidelberg, 2014. 3, 5

DN12. L. Ducas and P. Q. Nguyen. Faster Gaussian lattice sampling using lazy floating-point arithmetic. In *ASIACRYPT 2012*, vol. 7658 of *LNCS*, pp. 415–432. Springer, Heidelberg, 2012. 4

DP16. L. Ducas and T. Prest. Fast fourier orthogonalization. In *Proceedings of the ACM on International Symposium on Symbolic and Algebraic Computation, ISSAC 2016, Waterloo, ON, Canada, July 19-22, 2016*, pp. 191–198. ACM, 2016. 3, 5

EFG$^+$22. 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*, vol. 13277 of *LNCS*, pp. 222–253. Springer, Heidelberg, 2022. 3, 5, 12, 13

EK20. T. Espitau and P. Kirchner. The nearest-colattice algorithm: Time-approximation tradeoff for approx-cvp. *Open Book Series*, 4(1):251–266, 2020. 19

ETWY22. T. Espitau, M. Tibouchi, A. Wallet, and Y. Yu. Shorter hash-and-sign lattice-based signatures. *IACR Cryptol. ePrint Arch., to appear at CRYPTO 2022*, p. 785, 2022. 3, 6, 11, 16

GPV08. C. Gentry, C. Peikert, and V. Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *40th ACM STOC*, pp. 197–206. ACM Press, 2008. 3, 5, 6, 13

HHP$^+$03. J. Hoffstein, N. Howgrave-Graham, J. Pipher, J. H. Silverman, and W. Whyte. NTRUSIGN: digital signatures using the NTRU lattice. In *Topics in Cryptology - CT-RSA 2003, San Francisco, CA, USA, April 13-17, 2003*, vol. 2612, pp. 122–140. Springer, 2003. 4

How07. N. Howgrave-Graham. A hybrid lattice-reduction and meet-in-the-middle attack against NTRU. In *CRYPTO 2007*, vol. 4622 of *LNCS*, pp. 150–169. Springer, Heidelberg, 2007. 20

HPRR20. J. Howe, T. Prest, T. Ricosset, and M. Rossi. Isochronous gaussian sampling: From inception to implementation. In *Post-Quantum Cryptography - 11th International Conference, PQCrypto 2020*, pp. 53–71. Springer, Heidelberg, 2020. 14, 21

HPS98. J. Hoffstein, J. Pipher, and J. H. Silverman. NTRU: A ring-based public key cryptosystem. In *Algorithmic Number Theory, Third International Symposium, ANTS-III, Portland, Oregon, USA, June 21-25, 1998*, vol. 1423 of *Lecture Notes in Computer Science*, pp. 267–288. Springer, 1998. 4

KEF20. P. Kirchner, T. Espitau, and P.-A. Fouque. Fast reduction of algebraic lattices over cyclotomic fields. In *CRYPTO 2020, Part II*, vol. 12171 of *LNCS*, pp. 155–185. Springer, Heidelberg, 2020. 20

KF17. P. Kirchner and P.-A. Fouque. Revisiting lattice attacks on overstretched NTRU parameters. In *EUROCRYPT 2017, Part I*, vol. 10210 of *LNCS*, pp. 3–26. Springer, Heidelberg, 2017. 20

Kle00. P. N. Klein. Finding the closest lattice vector when it's unusually close. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms, January 9-11, 2000, San Francisco, CA, USA*, pp. 937–941. ACM/SIAM, 2000. 13

Laa16. T. Laarhoven. *Search problems in cryptography: from fingerprinting to lattice sieving*. PhD thesis, Mathematics and Computer Science, 2016. Proefschrift. 20

MW17. D. Micciancio and M. Walter. Gaussian sampling over the integers: Efficient, generic, constant-time. In *CRYPTO 2017, Part II*, vol. 10402 of *LNCS*, pp. 455–485. Springer, Heidelberg, 2017. 19

NR06. P. Q. Nguyen and O. Regev. Learning a parallelepiped: Cryptanalysis of GGH and NTRU signatures. In *EUROCRYPT 2006*, vol. 4004 of *LNCS*, pp. 271–288. Springer, Heidelberg, 2006. 4

Pei10. C. Peikert. An efficient and parallel Gaussian sampler for lattices. In *CRYPTO 2010*, vol. 6223 of *LNCS*, pp. 80–97. Springer, Heidelberg, 2010. 5, 13

PFH$^+$20. T. Prest, P.-A. Fouque, J. Hoffstein, P. Kirchner, V. Lyubashevsky, T. Pornin, T. Ricosset, G. Seiler, W. Whyte, and Z. Zhang. FALCON. Technical report, National Institute of Standards and Technology, 2020. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions`. 12, 14

Por19. T. Pornin. New efficient, constant-time implementations of falcon. Cryptology ePrint Archive, Paper 2019/893, 2019. https://eprint.iacr.org/2019/893. 21

PP19. T. Pornin and T. Prest. More efficient algorithms for the NTRU key generation using the field norm. In *PKC 2019, Part II*, vol. 11443 of *LNCS*, pp. 504–533. Springer, Heidelberg, 2019. 9

Pre15. T. Prest. *Gaussian Sampling in Lattice-Based Cryptography*. PhD thesis, École Normale Supérieure, Paris, France, 2015. 3, 5, 12, 13

Pre17. T. Prest. Sharper bounds in lattice-based cryptography using the Rényi divergence. In *ASIACRYPT 2017, Part I*, vol. 10624 of *LNCS*, pp. 347–374. Springer, Heidelberg, 2017. 12

ZSS20. R. K. Zhao, R. Steinfeld, and A. Sakzad. Facct: Fast, compact, and constant-time discrete gaussian sampler over integers. *IEEE Transactions on Computers*, 69(1):126–137, 2020. 14