SOLMAE: Faster and Simpler Quantum-safe Signature based on NTRU-lattices

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

[1] International Research Institute for Cyber Security(IRCS)/KAIST, Korea
kkj@kaist.ac.kr
[2] NTT Social Informatics Laboratories, Japan
mehdi.tibouchi@normalesup.org
[3] Inria, France
alexandre.wallet@inria.fr
[4] PQShield SAS, France
t.espitau@gmail.com
[5] Tsinghua University, China
yang.yu0986@gmail.com
[6] LGUPLUS, Korea
cherryk@lguplus.co.kr

April 7, 2024

Abstract. This paper introduces the SOLMAE which is faster and simpler quantum-safe signature following the traditionally hash-and-sign paradigm in the style of Gentry–Peikert–Vaikuntanathan signatures, and instantiates it over NTRU lattices. In that sense, it is closely related to, and a successor of, several earlier schemes including Ducas–Lyubashevsky–Prest (DLP), Falcon, Mitaka and ANTRAG. More precisely, SOLMAE offers the "best of three worlds" between Falcon, Mitaka and ANTRAG.
Falcon is known to have the advantage of providing short public keys and signatures (essentially offering 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 by leveraging ANTRAG. 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 and a faster `Sign` procedure. This approach 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. Further implementation challenges are left to the conclusion.

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

1 Introduction

Designing cryptographically strong primitives such as digital signatures or key encapsulation mechanisms, *etc.* is really a major challenge and cannot 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 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 attacks based on NTRU lattices, which are well-scrutinized by the cryptographic community since their introduction approximately two decades ago.

FALCON [PFH+22] was selected as one of the final quantum-safe digital signature algorithms for almost six years long NIST PQC standardization project from 2017 based on the NTRU trapdoor together with DILITHIUM and SPHINCS+ in 2023; this algorithm was designed by leveraging FFO(Fast Fourier Orthogonalization)[DP16] but it is slow and complicated to implement.

At its core, FALCON is a hash-then-sign signature paradigm proposed by Gentry, Peikert and Vaikuntanathan [GPV08] based on the lattice problem. To be efficiently instantiated, this framework needs a class of lattices with efficiently computational *trapdoor bases* for the signing procedure. Ducas, Lyubashevsky and Prest [DLP14] showed that NTRU lattices are a nice class of lattices. Thus, short Gaussian vectors are sampled 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]. FALCON achieves the most compact signatures, and the most compact verification key and signatures combined sizes, and boasts verification times on par with elliptic curve signatures.

Inspired by FALCON's design. Espitau *et al.* presented so–called MITAKA[EFG+22] to reduce some drawbacks of FALCON. 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 signature and verification key sizes, in addition to allowing for additional features absent from FALCON, such as enjoying less expensive masking, and being parallelizable. In 2023, Espitau *et al.* [ENS+23] suggested so–called ANTRAG in order to improve MITAKA without loss of security covering all NIST 5 levels of security using the degree of cyclotomic ring from 512 to 1024 over specific cyclotomic polynomials under the prime modulus but is not limited to the power of 2.

Taking all advantages of FALCON, MITAKA and ANTRAG, SOLMAE [¶] is yet another quantum–safe signature based on NTRU trapdoor and achieves *better performance* for the *same security and advantages* as FALCON which focused only on NIST I and V levels of security. More precisely, SOLMAE offers the "best of three worlds" between FALCON, MITAKA and ANTRAG.

1.1 Design rationale

Overall, SOLMAE is summarized in Fig.1.

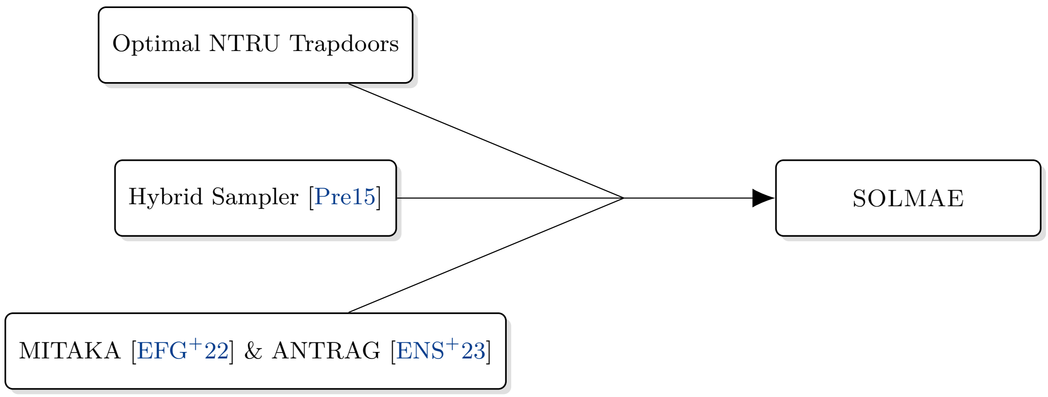


Fig. 1: Overview of SOLMAE

More details about all the objects mentioned in this section can be found later. 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 as follows:

[¶] It is an abbreviation of quantum–**S**ecure alg**O**rithm for **L**ong-term **M**essage **A**uthentication and **E**ncryption.

– An **optimally tuned key generation algorithm**, enhancing the security of our new sampler to that of FALCON's level[‖];
– The **hybrid sampler** is a faster, simpler, parallelizable and maskable Gaussian sampler to generate signatures;
– **Easy implementation** by assembling all the advantages of MITAKA and ANTRAG to make faster and simpler for practical purposes.

On the other hand, other techniques require tweaking the key generation and signing procedures.

The rest of the section provides more in–depth information about the history behind these improvements, and the reasons for their existence. For a more concrete description of the objects, the readers can refer 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 decoding problem can be solved when the 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 secret basis $\mathbf{B}$ a *trapdoor*, and in this documentation we call $(\mathbf{A}, \mathbf{B})$ a trapdoor pair.

The 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 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.

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 obtain a nonleaking 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

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

recover **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 addressed. This result was undeniably an enormous 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 their *Gram-Schmidt orthogonalization* to be 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], in which NTRU-like lattices were again the center of attention. By 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 provided strong and useful geometric constraints. Generally, the largest Gram-Schmidt vectors of an NTRU lattice 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 quickly by some kind of random walk among the set of potential keys. Indeed, backed by extensive experimental confirmations, the authors proposed a carefully tuned key generation algorithm relying on Gaussian cyclotomic integers, and while their focus was more advanced cryptographic functionality, this algorithm was arguably the birthstone of FALCON.

Sampling: the interplay between trapdoor quality and security level Klein's algorithm actually suffers from its quadratic complexity in practice. Because of the algorithm's design, parallelization is also unfriendly. 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 performance, close to the signing time of the other NIST winning signature, DILITHIUM.

A natural question is 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 methods are simpler than the Fast Fourier Sampler and are even more efficient, but the limitations of these algorithms are related to the metric driving their *quality*, that is, the variance they can achieve — recall that the smaller the variance is, the better the security is. Each sampling algorithm relies on a different metric tied to its geometric design; in other words, the notion of a"good trapdoor" differs from one 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 occur; in other words, the sparsity of good trapdoors made them expensive to find. From these observations, FALCON's team made the understandable choice of better security in general. This choice unfortunately sacrifices not only simplicity, but also side-channel resilience.

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

1.2 Advantages and limitations

SOLMAE has the following advantages.

- **Compactness:** The signature size, or the combined verification key plus signature size, is 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:** The SOLMAE can be masked with standard and well-understood counter-measures at a lower overhead than that of Falcon.

On the other hand, our scheme has several drawbacks:

- **Reliance on floating point arithmetic:** Similar to 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 of the floating points arithmetic. However we can reduce the floating point arithmetic in key generation procedures borrowed from the idea of Thomas [Por23].
- **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 to the best of our knowledge, no significant improvement on the cryptanalysis or asymptotic complexity is 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 those of 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.

1.3 Errata and their correction

Table 1 summarizes the errors and typos of SOLMAEv1 [KTE+23], and their correction in SOLMAEv2 was modified here.

Table 1: Errata in SOLMAEv1 and their correction in SOLMAEv2

No	Location in SOLMAEv1	SOLMAEv1	SOLMAEv2
1	line 5 from bottom, p7	$\frac{1}{\sqrt{d}}(\Re\varphi_1(x), \Im\varphi_1(x)...$	$\sqrt{\frac{2}{d}}(\Re\varphi_1(x), \Im\varphi_1(x)...$
2	line 13 and 16 in **Algorithm 1**, p10	σ_{sig}	σ_{sig}^2
3	line 13 in **Algorithm 2**, p11	α^2/q	q/α^2
4	line 13 in **Algorithm 2**, p11 (2 places)	$\varphi(g)$	$\varphi_i(g)$
5	line 10 in **Algorithm 3**, p12	$\|(\hat{s_1}, \hat{s_2})\|$	$\|(FFT^{-1}(\hat{s_1}), FFT^{-1}(\hat{s_2}))\|$
6	line 7 in **Algorithm 4**, p13	σ_{sig}	σ_{sig}^2
7	line 8 from bottom in Def. of Σ_i, p13	σ_{sig}	σ_{sig}^2
8	line 2, p14	σ_{sig}	σ_{sig}^2
9	line 3 in **Algorithm 9**, p15	u_x, u_y	u_θ
10	line 5 in **Algorithm 9**, p15	$x \leftarrow \rho \cdot \cos(2\pi u_x)$	$x \leftarrow \rho \cdot \cos(\frac{\pi}{2} u_\theta)$
11	line 6 in **Algorithm 9**, p15	$y \leftarrow \rho \cdot \sin(2\pi u_y)$	$y \leftarrow \rho \cdot \sin(\frac{\pi}{2} u_\theta)$
12	line 2 in **Algorithm 10**, p16	$\mathcal{N}_d$	$\mathcal{N}_{d/2}$
13	line 4 in **Algorithm 10**, p16	$\rho \leftarrow \sqrt{-2d \ln u_\rho}$	$\rho \leftarrow \sqrt{-d \ln u_\rho}$
14	unit of **sgn** size in Table 2, p18	kBytes	Bytes

2 Preliminaries

Vectors are in bold lower case, and considered to be in column. Matrices are in bold upper case. When we say that a matrix is a basis of a space, we mean that 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 often work in the subring $R = \mathbb{Z}[X]/(X^d + 1)$, and sometimes 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 root 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. The 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

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

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

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

We have $[f^*] = [f]^t$.
In algorithms, we 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 that 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 with 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 a 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 as follows: given $h = g/f \bmod q$, 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 parameters 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 $\sqrt{\frac{2}{d}}(\Re\varphi_1(x), \Im\varphi_1(x) \ldots, \Re\varphi_{d/2}(x), \Im\varphi_{d/2}(x))$ has a distribution $\mathcal{N}_\Sigma$, where $\Re z$ and $\Im z$ are the real and imaginary parts of complex z, respectively. 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 2.

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 has 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 $\mathtt{H}$ 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 as $\mathbf{s} = \mathbf{c} - \mathbf{v}$, 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. Otherwise it is rejected.

Of course, the resulting scheme must be *efficient* in practice: consume the *least possible* bandwidth 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 more difficult 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 pair 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 now has 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 pairs (f, g) and (F, G) provides 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 have been 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, the running time of NtruSolve largely dominates the overall time for KeyGen: this is not avoidable because the basis completion algorithm requires the operation of 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 Sections 3.3 and 3.4. A list of their practical value is given in Table 2. Informally, they correspond to the following:

– $(f, g), (F, G)$ is a good basis of the lattice $\mathcal{L}_{\text{NTRU}}$ associated with 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);
– $\sigma_{\mathtt{sig}}$ and η are the standard deviation for signature vectors and a tight upper bound on the smoothing parameter of $\mathbb{Z}^d$, respectively;
– $\Sigma_1, \Sigma_2 \in K_\mathbb{R}$ represent covariance matrices for two intermediate Gaussian samplings in Sample;
– the vectors β_1 and $\beta_2 \in K_\mathbb{R}^2$ 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, and 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. Fig. 2 sketches the key generation procedure.

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

1. The PairGen algorithm generates d complex numbers $(x_j e^{i\theta_j})_{j \leq d/2}$ and $(y_j e^{i\theta_j})_{j \leq d/2}$ to act as the FFT representations of two *real* polynomial $f^\mathbb{R}$ and $g^\mathbb{R}$ in $K_\mathbb{R}$, respectively. The magnitudes of these complex numbers are sampled in a planar annulus whose small and large 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 rounding, unless the rounding-error is 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 with $h = g/f \bmod q$. It does so by solving the Bézout-like equation $fG - gF = q$ in R using recursively the tower of subfields for optimal efficiency.

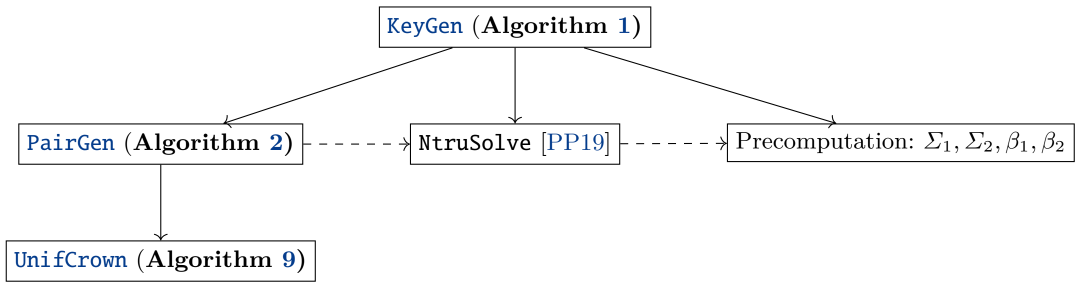


Fig. 2: Flowchart of KeyGen.

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

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;
```

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

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

Similarly, 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, which 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

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

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

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

As this is determined by the evaluations of the polynomials f and g, or equivalently, by their FFT representations, it is natural to sample these polynomials directly 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 $f^{\mathbb{R}}$ and $g^{\mathbb{R}} \in K_{\mathbb{R}}$. We thus have to round the coordinates of $f^{\mathbb{R}}$ and $g^{\mathbb{R}}$ to their nearest integers which incurs a small error. We handle this error by sampling $f^{\mathbb{R}}$ and $g^{\mathbb{R}}$ 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 remain in the correct crown $A(\sqrt{q}/\alpha, \alpha\sqrt{q})$ with a large probability. In practice we take $\delta_{512} = 0.065$ and $\delta_{1024} = 0.3$, values that are set so that ≈ 80 tries on average for $d = 512$, resp. ≈ 5 tries on average for $d = 1024$, of the decoded polynomials (f, g) satisfy Eq.(1).

Let us explain the determination of these values. The decoding errors can be accurately modeled 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. Due to the good concentration properties of Gaussians, we can fine-tune the tolerance with standard calculations, that match experimental observations. This leads to PairGen, where the target quality is by default $\alpha_{512} = 1.17$ and $\alpha_{1024} = 1.64$.

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. **

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 (MSBs) 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

** 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 here.

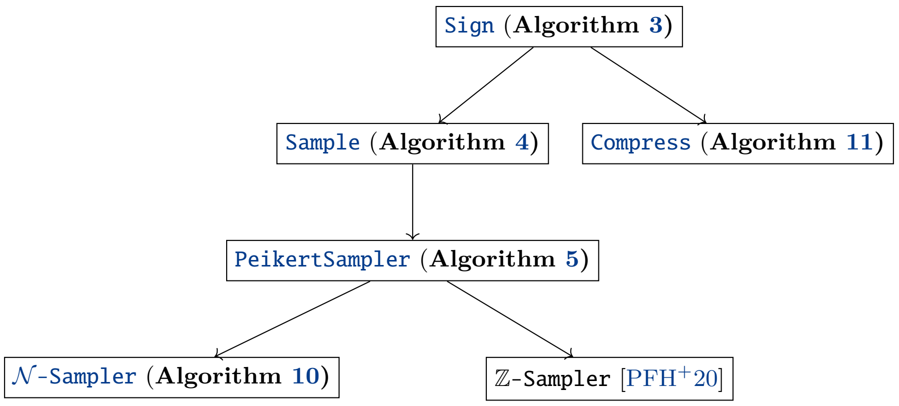
Algorithm 2: PairGen

```
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

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

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

Systems (ANS). Our signatures then enjoy an additional $7 - 12\%$ compression factor. This will be implemented in a 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 purposes. 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 occur, that is, it hashes $(r||M)$ instead of M — our analysis in this regard is identical to that of Falcon. A signature is then $\mathtt{sig} = (r, \mathtt{Compress}(s_1))$.

Fig. 3 sketches the signing procedure.



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

Fig. 3: Flowchart of `Sign`.

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 of lattice Gaussian vectors. The value of ϵ is deduced following [Pre17, EFG+22] and the current 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 the 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 $\mathsf{sk} = ((f,g),(F,G),(\widetilde{F},\widetilde{G}),\sigma_{\mathtt{sig}},\Sigma_1,\Sigma_2,\eta)$, a
       rejection parameter $\gamma > 0$.
Output: A pair $(r, \mathtt{Compress}(\mathbf{s}_1))$ with $r \in \{0,1\}^{320}$ and $\|(\mathbf{s}_1,\mathbf{s}_2)\| \leq \gamma$.
$r \leftarrow \mathcal{U}(\{0,1\}^{320})$;
$\mathbf{c} \leftarrow (0, \mathtt{H}(r||M))$;
$\hat{\mathbf{c}} \leftarrow \mathsf{FFT}(\mathbf{c})$;
repeat
    $(\hat{s}_1, \hat{s}_2) \leftarrow \hat{\mathbf{c}} - \mathtt{Sample}(\hat{\mathbf{c}}, \mathsf{sk})$;
    // $(\mathbf{s}_1, \mathbf{s}_2) \leftarrow D_{\mathcal{L}_{\text{NTRU}},\mathbf{c},\sigma_{\mathtt{sig}}}$
until $\|(\mathsf{FFT}^{-1}(\hat{s}_1), \mathsf{FFT}^{-1}(\hat{s}_2))\|^2 \leq \gamma^2$;
$s_1 \leftarrow \mathsf{FFT}^{-1}(\hat{s}_1)$;
$s \leftarrow \mathtt{Compress}(s_1)$;
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 provide 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 regarded as *randomized decoding procedures*. Babai described two well-known lattice decoding algorithms: the round-off and the nearest-plane. Intuitively speaking, the first just rounds coordinates of the target in the lattice basis to their nearest integer, while the second is more fine-grained and rounds 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 suffers from a lack of good trapdoors. In other words, signatures are longer with this approach. The latter compensates for 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 performed 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 trapdoors for this sampler were costlier to find. Owing to our new `KeyGen` algorithm, this is no longer a concern, and we are now able to enjoy all the advantages that the hybrid sampler brings versus the FFO sampler: it is simpler to implement, it is more efficient, and it is more friendly to parallelization. Moreover, its masking can be performed 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 ocurring *at the ring level*, or morally, in a d-dimensional space. In **Algorithm 4** below, note that *all operations are performed 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$(t_i, \Sigma_i, \eta)$ ;        /* $z_i \leftarrow D_{R, t_i, \frac{\sigma_{\mathtt{sig}}^2}{\langle \widetilde{\mathbf{b}_i}, \widetilde{\mathbf{b}_i} \rangle}}$ */
    $\mathbf{t} \leftarrow \mathbf{t} - z_i \mathbf{b}_i, \mathbf{v} \leftarrow \mathbf{v} + z_i \mathbf{b}_i$;
end
return $\mathbf{v}$;
```

The randomization is performed 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

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

By choice of $\sigma_{\mathtt{sig}}$, they correspond in Fourier domain to positive definite matrices which are 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 perturbation-based sampling. This translates concretely as a *continuous, elliptic* Gaussian sampling that is easy to handle in Fourier domain with enough precision, combined with a *spherical, discrete* Gaussian sampling of width η, as shown 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) */
$(p_1, \ldots, p_d) \leftarrow \mathsf{FFT}^{-1}(p)$ ;        /* $(p_i)_i \in \mathbb{R}^d$ */
$(t_1, \ldots, t_d) \leftarrow \mathsf{FFT}^{-1}(t)$ ;        /* $(t_i)_i \in \mathbb{Z}^d$ */
for $i = 1$ to $d$ do
    $x_i \leftarrow \mathbb{Z}$-Sampler$(t_i - p_i, \eta)$;
end
return $\mathsf{FFT}(x_1, \ldots, x_d)$;
```

By definition of the parameters, the calls to **Algorithm 5** in **Algorithm 4** output two ring elements with respective covariance $\sigma_{\texttt{sig}}^2/\langle \widetilde{\mathbf{b}}_i, \widetilde{\mathbf{b}}_i\rangle_K$, 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 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 performed efficiently at a sufficiently high floating-point precision. Next, the sampling uses a Cumulative Distribution Table (CDT) approach (also known as an inversion-based sampler). The space is saved by using the CDT of a *half* Gaussian distribution combined with 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 as shown in **Algorithm 6**. Upon receiving a signature (r, s) and message M, the verifier decompresses s to a polynomial s_1 and $\mathbf{c} = (0, \mathtt{H}(r||M))$ 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 performed very efficiently for a good choice of modulus q using the Number Theoretic Transform (NTT). We currently follow the standard choice (as 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 pass through the loop:

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

Algorithm 6: `Verif`

```
Input: A signature $(r, s)$ on $M$, a public key $\mathsf{pk} = h$, 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.
```

3.6 Miscellaneous

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

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

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 $A(R_-, R_+)$
$u_\rho, u_\theta \leftarrow \mathcal{U}(0, 1)$;
$\rho \leftarrow \sqrt{R_-^2 + u_\rho(R_+^2 - R_-^2)}$;
$x \leftarrow \rho \cdot \cos(\frac{\pi}{2} u_\theta)$;
$y \leftarrow \rho \cdot \sin(\frac{\pi}{2} u_\theta)$;
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 $\mathcal{N}_{d/2}$
$u_\rho, u_\theta \leftarrow \mathcal{U}(0, 1)$;
$\rho \leftarrow \sqrt{-d \ln u_\rho}$;
$x \leftarrow \rho \cdot \cos(2\pi u_\theta)$;
$y \leftarrow \rho \cdot \sin(2\pi u_\theta)$;
return $(x, y)$
```

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

```
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 $s = \sum_{i=0}^{d-1} s_i X^i$
```

3.7 Domain parameters

For the sake of clarity, we summarize all the concrete values for the domain parameters as described in Table 2. Note that smoothing η of 1.320 can be used for SOLMAE-512 and SOLMAE-1024 simultaneously for the sake of convenience. In practice, there are no any cryptographic differences.

Table 2: Domain parameters for SOLMAE family

	SOLMAE-512	SOLMAE-1024
ring degree d	512	1024
dimension $2d$	1024	2048
modulus q	12289	12289
salt length k	320	320
smoothing η	1.320	1.320
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 $\sigma_{\texttt{sig}}$	173.54	245.62
slack τ	1.04	1.04
rejection bound γ^2	33870790	134150669

4 Security

4.1 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 the self-dual BKZ (DBKZ) reduction algorithm of Micciancio and Walter [MW17]: an output basis $(\mathbf{b}_1, \ldots, \mathbf{b}_n)$ yielded by the 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.

4.2 Key recovery attack

The key recovery consists of finding the private secret key (i.e., $f, g \in \mathcal{R}^2$) from the sole data of the public elements q and h. To the best of our knowledge, the most powerful attacks are realized through lattice reduction. It consists of 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 the 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. If we can recover the *projection* of the secret key onto $\mathcal{P}$, the orthogonal space to $\text{Span}(\mathbf{b}_1, \ldots, \mathbf{b}_{2d-\beta-1})$, then we can retrieve the full key in polynomial time by *Babai nearest plane* algorithm to lift it to a lattice vector of the desired norm. Hence it suffices to be able to 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 with a 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:

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

4.3 Signature forgery by reduction to Approx-CVP

As a Hash-and-Sign paradigm signature, forging a signature stems to feed 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 GSA, Theorem 3.3 of [EK20] states that under the condition: $\|\mathbf{x}-\mathbf{v}\| \leq \left(\delta_\beta^{2d}q^{\frac{1}{2}}\right)$, decoding can be performed in time poly(d) calls to a CVP oracle in dimension β.

As mentioned in [CPS⁺20] a standard optimization of this attack consists of considering only the lattice spanned by a subset of the vectors of the public basis and decoding within this sublattice. The only interesting subset seems to consist of 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 $q^{\frac{k}{2(2d-k)}}$ larger. 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:

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

4.4 On the other attacks on SOLMAE

In this section, we list the other possible types 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" parameter occurs when $q > (2d)^{2.83}$ for binary secrets, implying that, as is the case for Falcon and other NTRU-based NIST candidates, 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 this 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.

4.5 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 the "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 the dimensions 512 and 1024.

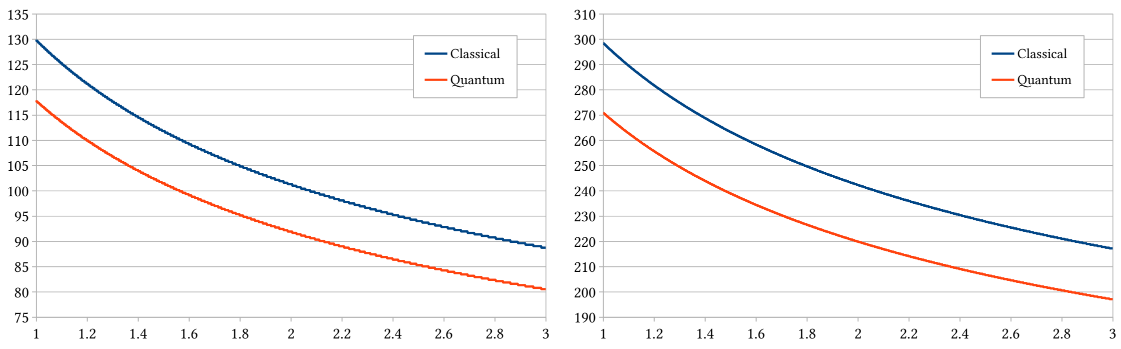


Fig. 4: Security (classical and quantum) against forgery as a function of the quality $1 \leq \alpha \leq 3$ of the lattice sampler (left: dimension 512 and right: dimension 1024).

Table 3: 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

4.6 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 this 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 a 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 a 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].

5 Performance analysis

Kim [Kim23b] verified asymptotic complexities of Falcon and SOLMAE in `KeyGen`, `Sign` and `Verif` procedures and proved that all of them take asymptotically similar $\Theta(d\log(d))$ complexity which is difficult to evaluate the exact performances of Falcon and SOLMAE from Python implementation.

Using the reference implementations of Falcon and SOLMAE which is and will be available over the github in C language, respectively, we show their specific execution times on average and make the performance comparison on the same platform.

5.1 Description of platform

The data below were collected on two cores of Intel Xeon CPU E5–2640 v3 2.60GHz, 4.00GB of RAM on the Ubuntu–20.04 server using the `CFLAGS = -Wall -Wextra -march=native -O3` compiling flag of gcc. All the tests were executed 10,000 times and the average values were taken for the fair evaluation.

5.2 Performance comparison

For this test, the input messages are chosen to be 1,024 bytes randomly per 10,000 times with each count using different key pairs. Falcon performance numbers are also provided using the `speed` tool included in the official code archive.

First, we checked and compared the performance of the Falcon with different security level of 256(educational level), 512(NIST level I) and 1024(NIST level V) in Fig. 5. The `Sign` procedure consumes more time than `KeyGen` and `Verif` procedures.

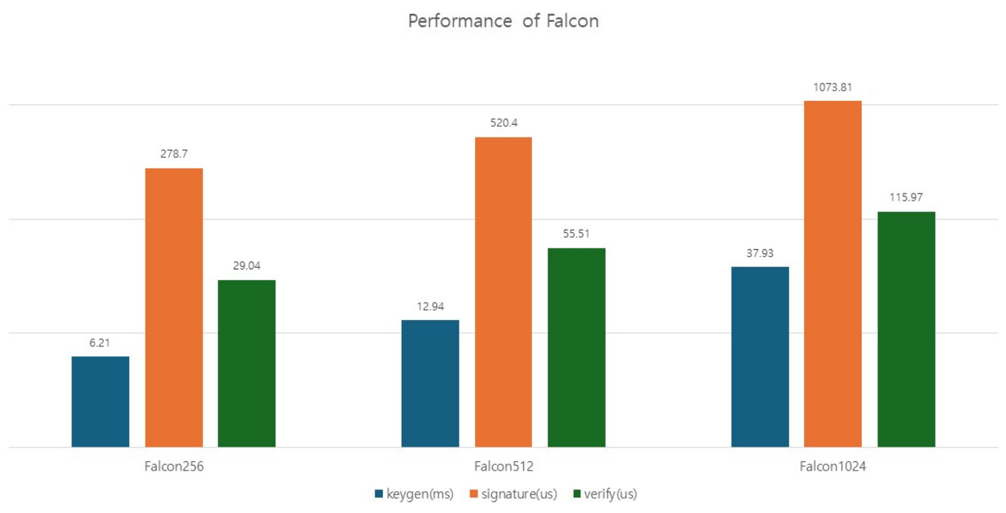


Fig. 5: Performance comparison of Falcon-{256, 512, 1024}

The average clock cycle and time (μs) during `KeyGen`, `Sign` and `Verif` procedures using SOLMAE-512 and SOLMAE-1024 are shown in Table 4.

Table 4: Performance comparison of SOLMAE and FALCON

		SOLMAE-512	SOLMAE-1024	FALCON–512	FALCON–1024
KeyGen time	Mcycles	58.77	137.76	33.65	98.65
	time (ms)	22.6	52.97	12.94	37.93
pk size	Bytes	896	1,792	896	1,792
Sign time	Kcycles	758.3	1,512.9	1,353.15	2,792.22
	time (μs)	291.63	581.82	520.4	1,073.81
sgn size	Bytes	666	1,375	666	1,280
Verif time	Kcycles	121.3	289.48	144.31	301.52
	time (μs)	46.66	111.34	55.51	115.97

Table 5: Average time (μs) of common computations for FALCON and SOLMAE families

	FFT	poly-add	pointwise-mult	FFT-mul-adj	poly-div-FFT	discrete-Gauss	Gaussian- sampling
SOLMAE-512	4.62	0.17	0.21	0.21	0.68	68.85	46.06
SOLMAE-1024	8.94	0.3	0.4	0.38	1.37	139.07	89.99

From these experiments, we found that SOLMAE without embedding compression and decompression functions consistently outperforms FALCON in and Sign and Verif procedures in equal dimension, while its KeyGen procedure is slightly slower due to its extra computation of additional data for the private key in advance. The time to execute compression and decompression functions can be ignored.

The average time in μs of internal common computations such as FFT, point-wise multiplication and Gaussian sampling, *etc.* which are commonly used for FALCON and SOLMAE families shown in Table 5.

On the other hand, in Table 6 we compare performance of SOLMAE-512 with Shake-128 and that of ECDSA P256r1 with SHA256 executed by Dreamsecurity Engineer [Kim23a] using our SOLMAE-512 C language reference implementation on their computing platform. Compared with ECDSA, the KeyGen of SOLMAE-512 is slower than that of EDCSA P256r1 by online computation [††]. However, the signing and verification of SOLMAE-512 takes about 10 times faster than those of ECDSA P256r1 currently used in TSL or SSL. This experiment shows that SOLMAE family makes no speed degradation when we apply quantum-secure SOLMAE for PKI and various embedded security applications.

Table 6: Comparison of SOLMAE and ECDSA

		SOLMAE	ECDSA
Specification		512	P256r1
Size(Bytes)	pk	1,792	65
	sgn	1,375	32
Time	KeyGen(ms)	30.21	2.53
	Sign(μs)	288.2	2,582.8
	Verif(μs)	55.6	7,744.7

[††] Note that KeyGen of SOLMAE can be operated online/offline together in parallel.

6 Implementation

We follow the same encoding and decoding formats used in Falcon for the maximum compatibility but some customized headers are used for keypairs and signature of SOLMAE.

6.1 Encoding Formats

Bits and Bytes We restate this part from Falcon specification to increase the readability of this paper. As usual, a byte is a sequence of eight bits (formally, an octet). Within a byte, bits are ordered from left to right. A byte has a numerical value, which is obtained by adding the weighted bits; the leftmost bit, also called *top bit* or *most significant*, has a weight of 128; the next bit has a weight of 64, and so on, until the rightmost bit, which has a weight of 1. Some of the encoding formats defined below use sequences of bits. When a sequence of bits is represented as bytes, the following rules apply:

– The first byte will contain the first eight bits of the sequence; the second byte will contain the next eight bits, and so on. item Within each byte, bits are ordered left-to-right in the same order as they appear in the source bit sequence.
– If the bit sequence length is not a multiple of 8, up to 7 extra padding bits are added at the end of the sequence. The extra padding bits MUST have value zero.

This handling of bits matches widely deployed standards, *e.g.,* bit ordering in the SHA-2 and SHA-3 functions, and BIT STRING values in ASN.1.

Signatures A SOLMAE signature consists of two strings r and s. These strings may conceptually be encoded separately, because the salt r must be known before beginning to hash the message itself, while the s value can be obtained or verified only after the whole message has been processed. In a format that supports streamed processing of long messages, the salt r would normally be encoded before the message, while the s value would appear after the message bytes. However, we here define an encoding that includes both r and s. The first byte is a header with the following format (bits indicated from most to least significant):

$$1cc0nnnn$$

with these conventions:

– The leftmost bit is 1, and the fourth bit from the left is 0.
– Bits cc are 01 or 10 to specify the encoding method for s. Encoding 01 uses the compression (or decompression) **Algorithm 11** (or **Algorithm 12**); encoding 10 is alternate uncompressed encoding in which each coefficient of s is encoded over a fixed number of bits. The encoding 00 is used for no encoding applied.
– Bits $nnnn$ encodes the hexadecimal value of $0x02$ for SOLMAE-512 and $0x0C$ for SOLMAE-1024 in Table 7.

Public Keys A public key in Falcon and SOLMAE is a polynomial h whose coefficients are considered modulo q. An encoded public key starts with a header byte:

$$1000nnnn$$

with these conventions:

– The four leftmost bits are 1000.
– Bits $nnnn$ encode the same value as the encoding header of signature from Table 7.

Using these conventions, the header byte of encoded public key is $0x82$ and $0x8C$ for SOLMAE-512 and SOLMAE-1024, respectively. After the header byte comes the encoding of h: each value (in the 0 to $q-1$ range) is encoded as a 14-bit sequence (since $q = 12289$, 14 bits per value are used). The encoded values are concatenated into a bit sequence of $14n$ bits, which is then represented as $\lceil 14n/8 \rceil$ bytes.

Private Keys Private keys use the following header byte:

$$1001nnnn$$

with these conventions:

- The four leftmost bits are 1, and the fourth bit from the left is 1.
- Bits $nnnn$ encode the same value as the encoding header of signature and public key from Table 7

Using these conventions, the header byte of encoded private key is $0x92$ and $0x9C$ for SOLMAE-512 and SOLMAE-1024, respectively.

Table 7: Coding value for SOLMAE-d-q

d	q	$nnnn$	Comments
512	12289	0x02	
1024	12289	0x0C	

7 Concluding Remarks

This paper summarizes the important features of SOLMAE which is a lattice-based signature scheme following the hash-and-sign paradigm (in the style of Gentry–Peikert–Vaikuntanathan signatures), and is 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, MITAKA and ANTRAG. More precisely, SOLMAE offers the "best of three worlds" between FALCON, MITAKA and ANTRAG.

We can conclude that SOLMAE solves the conundrum of choosing between three schemes by offering all their advantages. 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 and a faster `Sign` procedure. This approach is also compatible with recently introduced ellipsoidal lattice Gaussian sampling techniques to further reduce signature sizes. This makes SOLMAE state-of-the-art in terms of constructing efficient lattice-based signatures over structured lattices.

At Real World Post-Quantum Cryptography 2023(PQC2023) conference held in Tokyo, Prest [Pre23] commented in this talk that *SOLMAE uses the same simple, fast, parallelizable signing alorithm as Mitaka. By leveraging a novel key generation algorithm, SOLMAE achieves the same high security and short key and signature size as FALCON.* Moreover, based on SOLMAE specification [KTE+23], Cottaar *et el.* [CHH+23] concluded that *SOLMAE is close to FALCON, but easier to implement* in their evaluation report on KpqC submissions in terms of practical implementation.

Many practical reports on good–fit for FALCON were presented on V2V communications [TBRM22], TLS certificates [SKD00], DNSSEC [MdJvH+20, GS23], low–resource verification of FPGA [BKG22] and Cortex–M3 [GHK+21]. It is quite clear that SOLMAE can take all these practices as well due to its fundamental inheritance from FALCON.

Further implementation challenges are left to do next:

- Improving key generation procedure without floating-point arithmetic using Pornin's idea [Por23] due to non–constant time execution of FALCON using native FPU instructions with Cortex M7 Raspberry Pi 3 and others [HW22].
- Implementation of compression and decompression functions to reduce the size of keypairs and signature using FALCON's C language implementation.
- Optimized implementation on various platform, *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. 19

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. 5

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. 19

BKG22. L. Beckwith, J. Kaps, and K. Gaj. Fpqa energy consumption of post-quantum cryptography. Fourth PQC Standardization Conference, 2022. 23

CHH$^+$23. J. Cottaar, K. Hövelmanns, A. Hülsing, M. M. Tanja Lange, A. Pellegrini, A. Ravagnani, S. Schäge, M. Trimoska, and B. de Weger. Report on evaluation of kpqc candidates. Eindhoven University of Technology, OCt., 31 2023. https://groups.google.com/g/kpqc-bulletin/c/yrfBll_hr14. 23

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. 18

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. 2, 4

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. 3

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. 2, 4

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. 2, 4, 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. 18

ENS$^+$23. T. Espitau, T. T. Q. Nguyen, C. Sun, M. Tibouchi, and A. Wallet. Antrag: Annular ntru trapdoor generation. *Proc. of Asiacrypt2023, Part VII, Guangzhou, China*, pp. 3–32, 2023. 2

ETWY22. T. Espitau, M. Tibouchi, A. Wallet, and Y. Yu. Shorter hash-and-sign lattice-based signatures. *Proc. of CRYPTO 2022, Part II*, pp. 245–276, 2022. 4, 5, 10, 16

GHK$^+$21. R. Gonzalez, A. Hülsing, M. J. Kannwischer, J. Krämer, T. Lange, M. Stöttinger, E. Waitz, T. Wiggers, and B.-Y. Yang. Verifying post-quantum signatures in 8 kb of ram. In *Post-Quantum Cryptography*, pp. 215–233, Cham, 2021. Springer International Publishing. 23

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. 2, 3, 4, 5, 12

GS23. J. Goertzen and D. Stebila. Post-quantum signatures in dnssec via request-based fragmentation. In *Post-Quantum Cryptography*, pp. 535–564, Cham, 2023. Springer Nature Switzerland. 23

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. 3

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. 19

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, 19, 20

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. 3

HW22. J. Howe and B. Westerbaan. Benchmarking and analysing nist pqc lattice-based signature scheme standards on the arm cortex–m7. Fourth PQC Standardization Conference, 2022. 23

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. 18

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. 18

Kim23a. H. Kim. Personal correspondence, 2023. Provide upon request. 21

Kim23b. K. Kim. Theoretical and empirical analysis of falcon and solmae using their python implementation. *Proc. of ICISC2023, Seoul, Korea*, 2023. 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. 12

KTE$^+$23. K. Kim, M. Tibouchi, T. Espitau, A. Takashima, A. Wallet, Y. Yu, S. Guilley, and S. Kim. Solmae. KpqC Round 1 Submission, 2023. 5, 23

Laa16. T. Laarhoven. *Search problems in cryptography: from fingerprinting to lattice sieving*. PhD thesis, Mathematics and Computer Science, 2016. Proefschrift. 19

MdJvH$^+$20. M. Müller, J. de Jong, M. van Heesch, B. Overeinder, and R. van Rijswijk-Deij. Retrofitting post-quantum cryptography in internet protocols: A case study of dnssec, Oct. 2020. 23

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. 17

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. 3

Pei10. C. Peikert. An efficient and parallel Gaussian sampler for lattices. In *CRYPTO 2010*, vol. 6223 of *LNCS*, pp. 80–97. Springer, Heidelberg, 2010. 4, 12

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`. 11, 14

PFH$^+$22. 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, 2022. available at `https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022`. 2

Por19. T. Pornin. New efficient, constant-time implementations of falcon. Cryptology ePrint Archive, Paper 2019/893, 2019. `https://eprint.iacr.org/2019/893`. 19

Por23. T. Pornin. Improved key pair generation for falcon, bat and hawk. Cryptology ePrint Archive, Paper 2023/290, 2023. `https://eprint.iacr.org/2023/290`. 5, 23

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. 8, 9

Pre15. T. Prest. *Gaussian Sampling in Lattice-Based Cryptography*. PhD thesis, École Normale Supérieure, Paris, France, 2015. 2, 4, 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

Pre23. T. Prest. Lessons learned talks of the selected candidates: Falcon. Proc. of RWPQC2023, Tokyo, Mar. 26, 2023. `https://www.youtube.com/watch?v=J0QpSV2xSvM&t=17347s`. 23

SKD00. D. Sikeridis, P. Kampanakis, and M. Devetsikiotis. Post-quantum authentication in tls 1.3: A performance study. In *Proceedings of NDSS Symposium, San Diego, California, USA*. The Internet Society, Feb. 23–26, 2000. 23

TBRM22. G. Twardokus, N. Bindel, H. Rahbari, and S. McCarthy. When cryptography needs a hand: Practical post-quantum authentication for v2v communications. Cryptology ePrint Archive, Paper 2022/483, 2022. `https://eprint.iacr.org/2022/483`. 23

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