Squirrels

Square Unstructured Integer Euclidean Lattice Signature

Thomas Espitau, Guilhem Niot, Chao Sun, Mehdi Tibouchi

Contents

1 Introduction

1.1 Context and motivation

In the rapidly advancing field of post-quantum cryptography, significant progress has been achieved over the past decade. The recognition of its criticality has extended beyond the academic community to the general sphere, leading to a growing awareness of the need for robust cryptographic solutions. As a result, a diverse range of designs has emerged, reaching a level of maturity that enables their practical deployment.

A notable testament to the progress made in this field is the recent standardization efforts undertaken by the National Institute of Standards and Technology (NIST). In recent years, NIST standardized the stateful hash-based signature schemes XMSS and LMS [10], and selected 3 lattice-based schemes (the Kyber KEM and the signatures Dilithium and Falcon [41, 16]) and 1 stateless hash-based signature (SPHINCS [8]) for upcoming standardization, further demonstrating the increasing significance and practicality of post-quantum cryptographic primitives.

However, given the significant challenges associated with the costly and time-consuming deployment of entirely new cryptographic systems, coupled with the need to ensure the longevity of these systems over decades, there are application domains where adopting a *conservative* approach to selecting a post-quantum candidate scheme is preferable. In light of the unpredictable trajectory of quantum computing technology and quantum cryptanalysis in years to come, practitioners with valuable data requiring long-term confidentiality and authenticity guarantees may want to prioritize security and simplicity over premature optimization. In that perspective, the question of which class of hard problem to rely on is critical.

1.1.1 Structured vs. unstructured lattices

The security of public-key cryptosystems relies on the assumption that certain computational problems are difficult to solve. In lattice-based cryptography, two

related important problems are the Learning with Errors (LWE) and the Short Integer Solution (SIS) problem. The former involves solving a noisy, random linear system over the ring $\mathbb{Z}/q\mathbb{Z}$, whereas the latter asks to find a short solution to a linear system, again over $\mathbb{Z}/q\mathbb{Z}$. Both problems can also be interpreted as approximate close vector problems (i.e., the problems of decoding errors of a certain size) in random q-ary lattices.

Variants of these problems include algebraically structured versions such as *Ring-LWE* [33] and *Module-LWE* [30], as well as problems associated with the so-called NTRU lattices. These variants correspond to decoding problems in lattices over rings of algebraic integers (endowing the lattices themselves with additional algebraic structure). Cryptosystems based on those structured assumptions and structured lattices generally offer greater efficiency, since the corresponding lattices admit more compact representations, and typically benefit from the faster arithmetic of the underlying rings. In principle, however, the additional structure could also, introduce vulnerabilities that do not exist in the unstructured setting.

The current state of the art indicates that the recommended parameterizations for algebraically structured lattice problems do not appear to exhibit specific weaknesses when compared to plain lattice versions. However, the (quantum) complexity of certain related problems on specific types of algebraic lattices is lower than their counterparts on general lattices (see, e.g., [11, 12]). Whether new cryptanalytic techniques may in the future improve and extend those stronger attacks that exist in the structured setting to the point of meaningfully reducing the security of cryptosystems based on NTRU or structured variants of LWE and SIS remain to be seen. Uncertainty in this matter does however make unstructured lattices appear as a clearly more conservative choice.

1.2 The choice of unstructured lattices

Given the uncertain long-term prospects of algebraically structured lattices and the need for post-quantum standards to remain secure, our proposal is based on plain lattice problems, i.e., without additional algebraic structure. We choose to employ conservative parameterizations for enhanced security. While this choice incurs some efficiency trade-offs compared to algebraic variants, we believe it ensures robustness against potential weaknesses in the future.

Furthermore, the use of plain lattices presents minimal restrictions on parameter choices, making it possible to reach specific security levels in a fine-grained manner.

2 Design rationale

2.1 The Gentry–Peikert–Vaikuntanathan Framework

2.1.1 Provably secure lattice signatures

In 2008, Gentry, Peikert, and Vaikuntanathan [25] introduced a provably secure lattice-based signature scheme under the SIS assumption. The framework relies on the existence of a trapdoor basis, which serves as the secret key and exhibits excellent properties for Gaussian sampling to generate a lattice point close to the hash of the message. The verification process involves checking whether the signature belongs to the lattice using linear algebra techniques and ensuring that the distance between the lattice point and the message hash is sufficiently small. Over time, this robust framework has undergone enhancements to offer highly compact and efficient signature schemes, most notably resulting in the NIST-standardized Falcon [23] and variants such as Mitaka [19]. Before further ado, let us give a bit more details on the GPV framework, starting with its ancestor, the Goldreich-Goldwasser-Halevi signature.

2.1.2 From GGH to GPV

The GGH signature scheme [26] operates by mapping a message m to a random point $\mathbf{c}$ in the Euclidean space and then sampling a lattice point $\mathbf{s}$ that is in close proximity to $\mathbf{c}$ using a "good" secret basis that is both short and nearly orthogonal. Verification involves checking whether $\mathbf{s}$ is a lattice point close to $\mathbf{c}$ using a "bad" public basis that is independent of the secret basis. However, this construction leaks the shape of the secret basis.

In contrast, the GPV signature scheme represents an improvement over the GGH signature scheme by randomizing the choice of the close lattice point. It generates a signature that is statistically close to a discrete Gaussian distribution centered around the hashed message. This statistical property ensures that the output distribution becomes statistically independent of the secret basis, thereby providing the security of the signature.

The GPV signature can be described as follows:

- The public key is a full-rank matrix $\mathbf{A} \in \mathbb{Z}_q^{m \times n} (m < n)$.
- The secret key is a matrix $\mathbf{B} \in \mathbb{Z}_q^{n \times n}$ with short entries such that $\mathbf{B} \cdot \mathbf{A}^T = 0$.

- Given a message m', we first hash it to $H(m')$ where H is a hash function defined as $\{0,1\}^* \to \mathbb{Z}_q^n$. A signature is a short $\mathbf{s} \in \mathbb{Z}_q^n$ such that $\mathbf{s} \cdot \mathbf{A}^T = H(m')$. It is straightforward to check the validity of $\mathbf{v}$ by verifying that $\mathbf{s}$ is indeed short and $\mathbf{s} \cdot \mathbf{A}^T = H(m')$.

- To generate a signature, first a preimage $\mathbf{c}_0 \in \mathbb{Z}_q^n$ is computed such that $\mathbf{c}_0 \cdot \mathbf{A}^T = H(m')$, then $\mathbf{B}$ is used to find a vector $\mathbf{c}$ in the lattice spanned by $\mathbf{B}$ that is close to $\mathbf{c}_0$. The difference $\mathbf{s} = \mathbf{c_0} - \mathbf{c}$ is a valid signature, because $\mathbf{s} \cdot \mathbf{A}^T = \mathbf{c_0} \cdot \mathbf{A}^T - \mathbf{c} \cdot \mathbf{A}^T = H(m')$.

2.1.3 Beyond GPV

The GPV signature, originally designed for SIS-like lattices, presents a flexible and adaptable framework that can be extended to various lattice types while accommodating varying underlying assumptions. The core concept of the GPV signature simply involves the existence of a randomized sampler to generate a signature that closely approximates a discrete Gaussian distribution. This adaptability allows the GPV framework to serve as a versatile tool in the domain of lattice-based cryptography. Notably, Falcon [23] implemented GPV over NTRU lattices, whereas we propose to maintain an unstructured approach relying on plain lattices. To ensure efficient verification, akin to the Falcon verification equation, we focus on a broad class of plain lattices known as co-cyclic lattices.

2.2 On co-cyclic lattices

The structure of the quotient group $\mathbb{Z}^n/\mathcal{L}$, where $\mathcal{L}$ is an integer lattice, holds significant importance in the study of lattices. It provides insights into the average complexity of lattice problems, as highlighted in works such as [3] and its generalization [24]. In particular, when this quotient group is cyclic, meaning it is spanned by a single element, we refer to the lattice as *co-cyclic*. Notably, co-cyclic lattices exhibit a natural density of approximately 85% [37]. Moreover, Paz and Schnorr demonstrated in [39] that the worst-case hardness problems, such as SVP (Shortest Vector Problem) and CVP (Closest Vector Problem), on co-cyclic lattices are as challenging as those on unconstrained lattices.

Another equivalent characterization, as described in [39], involves the existence of a vector $\mathbf{w}$ such that $\mathcal{L} = \{\mathbf{x} \mid \langle \mathbf{x}, \mathbf{w} \rangle \bmod d = 0\}$. In other words, the lattice $\mathcal{L}$ can be defined as the set of all vectors $\mathbf{x}$ for which the inner product of $\mathbf{x}$ and $\mathbf{w}$ modulo d yields a specific value. This characterization also relates to the

row Hermite Normal Form (HNF) of the lattice, which takes the following form:

$$\begin{bmatrix} \mathbf{I}_{n-1} & \mathbf{v}_{\text{check}}^T \\ \mathbf{0} & \Delta \end{bmatrix} \tag{1}$$

where $\Delta = \det(\mathcal{L})$.

This characterization is very interesting from a practical point of view as it allows us to perform efficient *lattice membership check*. Computing the row HNF allows to find such a vector $\mathbf{w}$: $\mathbf{w} = (\mathbf{v}_{\text{check}}, -1)$. Indeed, given a point $\mathbf{c} \in \mathbb{Z}^n$ and the HNF of a co-cyclic lattice $\mathcal{L}$:

$$\begin{aligned} \mathbf{c} = (c_1, ..., c_n) \in \mathcal{L} &\iff \exists \mathbf{Y} = (y_1, ..., y_n) \mid \mathbf{Y} \cdot \text{HNF}(\mathcal{L}) = \mathbf{c} \\ &\iff \exists \mathbf{Y} \mid c_1 = y_1, c_2 = y_2, ..., c_{n-1} = y_{n-1}, \\ &\qquad\qquad c_n = \sum_{1 \leqslant i \leqslant n-1} y_i \cdot v_{\text{check},i} + y_n \cdot \Delta \\ &\iff c_n = \sum_{1 \leqslant i \leqslant n-1} c_i \cdot v_{\text{check},i} \bmod \Delta \end{aligned} \tag{2}$$

Hence, in the context of GPV-type signatures, we can achieve efficient verification when the underlying lattice is co-cyclic. To fully capitalize on this observation and create a practical signature scheme, our focus lies in the construction of reliable trapdoors specifically designed for co-cyclic lattices. By successfully addressing this aspect, we can develop an effective and secure signature scheme that harnesses the advantages of co-cyclic lattices.

2.3 Security assumptions

2.3.1 SIS-hash in the co-cyclic case.

As previously mentioned, the GPV framework is a versatile framework that can be instantiated to different classes of lattices. The underlying hardness assumptions however vary with this choice. In our case, SQUIRRELS relies on co-cyclic lattices instead of the uniform SIS-like lattices presented in [25] or NTRU lattices [23, 19].

The GPV framework constructs a signature scheme from a preimage samplable function f that is supposed to be collision resistant. We recall the formal definition of this class of function in Appendix A.1. The prototype of such function is the SIS hash $\mathbf{e} \mapsto \mathbf{A} \cdot \mathbf{e} \bmod q$, where $\mathbf{A}$ is the public matrix of the scheme, and $\mathbf{e}$ follows a Gaussian distribution of standard deviation $s \geqslant \eta_\varepsilon(\mathcal{L})$, the smoothing parameter of this lattice.

This can be adapted seamlessly to SQUIRRELS by taking:

$$f\colon D_n \to \mathbb{Z}_\Delta \\ \mathbf{x} \mapsto \mathbf{x}\cdot\mathbf{A}^T \bmod \Delta$$

with $D_n = \{\mathbf{e} \in \mathbb{Z}^n \mid ||\mathbf{e}|| \leqslant \beta\}$, the input distribution of $\mathbf{e}$ is $D_{\mathbb{Z}^n,\beta/\sqrt{n}}$ and $\mathbf{A} = (\mathbf{v}_{\text{check}}, -1)$ the public key of SQUIRRELS. Lemma 5.2 of [25] adapts directly to prove that when $\mathbf{e}$ follows $D_{\mathbb{Z}^n,\beta/\sqrt{n}}$ with $\beta/\sqrt{n} \geqslant \eta_\varepsilon(\mathcal{L})$, $f(\mathbf{e})$ is statistically close to $\mathcal{U}(\mathbb{Z}_\Delta)$. To get possible instantiation, we need to estimate the smoothing parameter of the lattice $\mathcal{L}$. It appears that for the class of co-cyclic we will relate to, it will be a constant factor in the smoothing of $\mathbb{Z}^n$ with overwhelming probability, as we discuss in Section 2.3.3.

Then, we define the (sampleable) inversion function $f^{-1}(u)$ for $u \in \mathbb{Z}_\Delta$ as follows: we chose via linear algebra a $\mathbf{t} \in \mathbb{Z}_\Delta^n$ such that $\mathbf{t}\cdot\mathbf{A}^T = u$, then sample $\mathbf{v}$ from $D_{\mathcal{L},u,-\mathbf{t}}$ using the Klein sampler with the secret basis, and output $\mathbf{e} = \mathbf{t} + \mathbf{v}$.

Now the proof of theorem 5.9 of [25] translates in asserting that f forms a preimage samplable function under a variant of ISIS, the Group-SIS $\text{GISIS}_{\mathbb{Z}_\Delta,\beta}$ with cyclic group $\mathbb{Z}_\Delta$, for a uniform syndrome $u \in \mathbb{Z}_\Delta$. It is collision-resistant under the hardness of the corresponding $\text{GSIS}_{n,\mathbb{Z}_\Delta,2\cdot\beta}$ problem.

2.3.2 Generalized SIS and hardness assumptions.

The problems mentioned above are formally defined as the following generalization of the (I)SIS problems when the structure of the quotient $\mathbb{Z}^n/\mathcal{L}$ is prescribed:

- **$\text{GSIS}_{n,\mathbb{Z}_\Delta,\beta}$:** Given a random lattice $\mathcal{L}$ of dimension n such that $\mathbb{Z}^n/\mathcal{L} = \mathbb{Z}_\Delta$, find a vector $\mathbf{x}$ such that $||\mathbf{x}|| \leqslant \beta$ and $\mathbf{x}\cdot\mathbf{A}^T = 0 \bmod \Delta$.
- **$\text{GISIS}_{n,\mathbb{Z}_\Delta,\beta}$:** Given a random lattice $\mathcal{L}$ of dimension n such that $\mathbb{Z}^n/\mathcal{L} = \mathbb{Z}_\Delta$ and a uniform $u \in \mathbb{Z}_\Delta$, find a vector $\mathbf{x}$ such that $||\mathbf{x}|| \leqslant \beta$ and $\mathbf{x}\cdot\mathbf{A}^T = u \bmod \Delta$.

The security of GSIS for co-cyclic lattices with determinant Δ is supported by an average to worst-case reduction in any integer lattice in [24]. The high density of co-cyclic lattices among the moduli space of all integer lattices (without the determinant constraint) also allows reducing the average GISIS on co-cyclic lattices to average ISIS on all integer lattices. This gives us confidence that our two hardness assumptions are as hard as on any full-rank integer lattices.

2.3.3 Regularity of the keygen output.

To fully follow the security proof of GLP, we hence only need to assume that the lattices sampled by our key generation algorithm "behave as if" they were sampled randomly from the family above. More precisely, we make the following assumption:

- **SQR-PR$_\lambda$:** The public matrix $\mathbf{A} = (\mathbf{v}_{\text{check}}, -1)$ output by **KeyGen**(1^λ) (Algorithm 1) is computationally indistinguishable from a uniformly random element of $\mathbb{Z}_\Delta^{n-1} \times \{-1\}$

This assumption is very natural from the construction and is the exact non-structured analog of the NTRU assumption (the NTRU assumption over the ring $\mathbb{Z}$ would coincide exactly with our assumption as the normal form of the NTRU lattice would be exactly the Hermite form of the basis). Under this assumption, it is straightforward to prove that the smoothing parameter of the lattices used in SQUIRRELSis as announced: a constant factor off from the smoothing of the cubic lattice $\mathbb{Z}^n$ (see Appendix A.2.3 for proof of this claim).

2.4 The SQUIRRELS family

We present the SQUIRRELS family, a collection of lattice-based digital signature schemes for each of the five NIST security level requirements. These schemes adopt a hash-and-sign structure construction and rely on unstructured co-cyclic lattices as their foundation. At the core of these schemes is an integral matrix of dimensions $n \times n$, serving as a trapdoor sampling basis for the lattice, which we will require to be co-cyclic. This matrix consists of a set of n short and relatively orthogonal vectors, with constraints on their so-called Gram-Schmidt norm to ensure good sampling properties. On the other hand, a public basis, expressed in the Hermite Normal Form (HNF), is made available to enable the membership test for this lattice. Unlike NTRU lattices, the secret basis employed in SQUIRRELS cannot be easily compressed due to the absence of strong geometric properties. It is worth noting that this is a drawback inherent to any plain lattice scheme. Indeed, we are sampling our keys from a distribution that is computationally indistinguishable from the distribution of maximal entropy for the dimension and size involved.

The sampler used to produce signatures is the so-called *Klein sampler* [29], already used in the original GPV proposal [25].

2.4.1 SQUIRRELS secret keys

Even though signatures generation still relies on the Klein sampler, the generation of the trapdoor differs significantly from earlier work, including the original GPV, or even from the more recent Falcon and Mitaka signatures. We start with the observation that the quality of signatures generated by the Klein sampler [29, 25] depends on the maximal norm of the Gram-Schmidt vectors of the trapdoor basis. It follows that we would like to generate trapdoors with such Gram-Schmidt norm as small as possible. However, as we fix the determinant both to address the fact that the problem is essentially scale-invariant and for efficiency purposes, we rather want trapdoor basis vectors to have their Gram-Schmidt norms to differ as little as possible from the geometric mean imposed by the determinant. To construct such a basis, we depart from the usual approach of sampling a trapdoor with random Gaussian vectors and testing its quality afterward, as in [25, 23]. Instead, our *key pair generation* sequentially samples secret vectors from regions of the space that are carefully crafted to provide a well-controlled Gram-Schmidt norm. These vectors should ideally be short and close to orthogonal for the best possible sampling quality, but not *too* short and orthogonal so as not to jeopardize security with respect to key recovery attacks. Hence, at each step the algorithm samples a vector of controlled norm close to the orthogonal subspace of the vector subspace spanned by the previous vectors. It finally selects the last vector in such a way that the resulting matrix matches the prescribed determinant.

2.4.2 Public key derivation

To derive an efficient public key, we require additionally that the sampled lattice has to be co-cyclic, i.e., we can find a vector $\mathbf{w} = (\mathbf{v}_{\text{check}}, -1)$ such that $\mathbf{c} \in \mathcal{L} \iff \langle \mathbf{c}, \mathbf{w} \rangle = 0 \bmod \Delta$ as seen in section 2.2. We choose a square-free determinant, as described in appendix B. This minor check actually automatically enforces the co-cyclicity of the lattice, so in practice we do not need to reject lattices because they are not co-cyclic.

The vector $\mathbf{v}_{\text{check}}$ can be computed from the row HNF of the secret basis. The HNF can be computed in polynomial time from any lattice and thus gives a basis of the lattice that is "as bad as can be". The public key of SQUIRRELS is chosen to be the vector $\mathbf{v}_{\text{check}}$.

2.4.3 Signature sampling

As explained previously, *signature generation* consists in first hashing the message to sign, along with a random nonce, into a vector $\mathbf{h}$, whose coefficients are uniformly mapped to integers in the 0 to $q-1$ range. Then, the signer uses his knowledge of the secret lattice basis to produce a vector $\mathbf{c}$ belonging to the lattice that is close to $\mathbf{h}$. The signature s properly is $\mathbf{c}-\mathbf{h}$. Sampling a vector in the lattice (very) close to an arbitrary point is in general a hard problem, but here we rely on the fact that the secret basis is a basis of the lattice composed of short vectors. Klein's Gaussian sampler [29, 25] is used to efficiently sample this Gaussian vector.

2.4.4 Fast verification

The *verification* procedure first recomputes the hash of the message $\mathbf{h}$, and the lattice point $\mathbf{c}=\mathbf{s}+\mathbf{h}$. It verifies that $\mathbf{s}$ is a short vector, and that $\mathbf{c}$ actually belongs to the lattice. Efficient membership checks are possible using the properties of co-cyclic lattices from section 2.2. We simply need to verify the equation 2 using $\mathbf{v}_{\text{check}}$. Fixing the determinant of the lattice to a fixed product of primes allows to work modulo small primes, thanks to the Chinese remainder theorem, in the verification procedure, instead of modulo Δ, and makes the verification procedure more efficient.

3 Advantages and limitations

3.1 Advantages

Confidence in unstructured lattices. One concern about Falcon is its use of NTRU lattices which might be vulnerable to specialized and more powerful attacks than the generic attacks on lattices. Our scheme samples lattices with no strong geometric property and bases its security on generic lattice problems. These problems have been studied for decades, notably with average-to-worst-case reductions, which give strong confidence in their security.

Compact signatures. Despite leveraging an unstructured problem, our scheme still manages to generate remarkably compact signatures. The byte-size of our signatures falls within the range of Falcon and Dilithium, both of which are renowned for producing concise signatures in the post-quantum setting.

Efficient signature generation and verification. SQUIRRELS is also very competitive in terms of signature generation and verification efficiency. On a personal laptop, it is capable of generating several dozens to hundreds of signatures per second, while also verifying thousands of signatures within a single second.

Simple signature verification. The signature verification process is remarkably straightforward, primarily consisting of a hash computation and the verification of a *single linear equation*. Its simplicity streamlines the verification procedure without compromising the security of the scheme.

3.2 Limitations

Slow key generation. One aspect that warrants consideration is the relatively slow key generation procedure employed in our scheme. This process involves computationally expensive operations on high-dimensional matrices, such as determinant calculations and HNF (Hermite Normal Form) computations. Consequently depending on the target hardware and security level desired, generating a single keypair can take anywhere several dozen seconds. The acceptability of this duration depends on the specific application at hand. However, it may prove to be prohibitive if the application necessitates a high frequency of rotation of signature keys. This is a direct consequence of the choice of using unstructured lattices: every non-trivial linear algebra operation is at the very least quadratic in the dimension of the lattice.

Large public keys. Due to the absence of structure in the lattices we sample, it is not possible to compress the public key in a manner similar to Falcon. As a result, our public keys are larger by a factor of $\mathcal{O}(n)$, weighing several hundred kilobytes to a few megabytes. This increase in size should be taken into consideration, particularly when storage or transmission constraints are significant factors in the system's design. Once again, this can not be improved when dealing with unstructured lattices, as the entropy of the matrix representing the keys is essentially maximal for their size and dimensions.

Floating-point arithmetic. It is important to note that our signature scheme utilizes floating-point arithmetic during both key generation and signature generation procedures. While this choice contributes to the scheme's effectiveness,

it may pose a significant limitation when implementing the scheme on very constrained devices with limited computational capabilities or limited support for floating-point operations. Careful consideration must be given to the feasibility and the practicality of implementing our scheme in such environments. We stress that verification, on the other hand, does not rely on floating point arithmetic, so the more common use case where signatures only need to be *verified* on constrained devices (e.g., bootloader signing) is supported without issue.

4 Security considerations

To assess the concrete security of the SQUIRRELS 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. We first give a quick discussion on the modelization of practical lattice reduction algorithms.

4.1 Heuristic modelization of lattice reduction, GSA and beyond

4.1.1 On the core-SVP model

To accurately assess the hardness of the underlying problems and ensure security in terms of bits, it is necessary to model the behavior of a practical oracle that approximates the Shortest Vector Problem (SVP). Our problems involve finding relatively short vectors in various lattices. For this purpose, we will employ the well-known (self-dual) Block Korkine-Zolotarev (BKZ) algorithm. The BKZ algorithm, with a block size denoted as B, may require a polynomial number of calls to an SVP oracle in dimension B, with a heuristic estimation of the number of calls being essentially linear. To account for potential future improvements in this reduction technique, we will only consider the cost of a single call to the SVP oracle. This conservative estimation is referred to as "core-SVP hardness." This cautious approach is motivated by the fact that there are methods to amortize the cost of SVP calls within BKZ, particularly when sieving is employed as the SVP oracle. Sieving is getting the de facto standard for larger cryptographic block sizes (we for instance refer to [4] for more details on the practical challenges raised by the use of sieving within lattice reduction).

4.1.2 Modelization of the output of reduced bases.

In all of the following and to ease the presentation, 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 Walter [36]: an output basis $(\mathbf{b}_1, \ldots, \mathbf{b}_n)$ yielded by DBKZ algorithm with block size B on a lattice $\mathcal{L}$ of rank n satisfies

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

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

To obtain a more accurate estimation when computing actual figures, it is beneficial to enhance this analysis by employing the probabilistic simulation proposed in [14]. This simulation provides a more precise determination of the Block Korkine-Zolotarev (BKZ) block size B required for a successful attack, surpassing the coarse estimation based on the Geometric series assumption (GSA). By incorporating this probabilistic simulation, we can consider the widely recognized "quadratic tail" phenomenon of reduced bases [44], thereby improving the precision of our calculations.

4.1.3 From lattice reduction blocksize to bitsec estimates.

This analysis translates into concrete bit-security estimates following the methodology of NEWHOPE [5] (so-called "core-SVP methodology"). In this model, the bit complexity of lattice sieving (which is asymptotically the best SVP oracle) is taken as $\lfloor 0.292B \rfloor$ in the classical [7] setting and $\lfloor 0.257B \rfloor$ in the quantum setting[9] in dimension B.

As the whole methodology is restated, we now turn to the fine-grained security of key recovery and then forgery.

4.2 Key Recovery attack

The key recovery attack aims at finding (at least) one of the short vectors of the secret basis, from the knowledge of the public key. A direct approach to key recovery is to do lattice reduction on a public basis, aiming at finding a relatively short vector in the spanned lattice: such attacks are addressed in Section 4.2.1. However, when the key becomes sparse ternary because of its length, combinatorics

and more importantly *hybrid* attacks (combining lattice reduction and meet-in-the-middle approach) can be considered as a potential threat. This effect is quite similar to the caveats raised in [21] on distorted hash-and-sign signatures.

4.2.1 Basic projection attack

This technique, initially described in the Falcon specification [23] and subsequently utilized in Mitaka [19] , operates by examining the lattice formed by the public basis, which is encoded in the public key as the Hermite Normal Form of the lattice in our scheme. It then finds vectors of the secret basis by listing all possible lattice vectors of norm less than $g_{\min}$. The attack avoids listing all lattice vectors in that sphere by restricting the search space to a projection.

More precisely, we fix B the block size of the DBKZ algorithm [36], and we first reduce the public basis using DBKZ to obtain a reduced basis $[\mathbf{b}_1, \ldots, \mathbf{b}_n]$. Then, we consider the lattice projected on $P = \text{Span}(\mathbf{b}_1, \ldots, \mathbf{b}_{n-B-1})^{\perp}$. If we can find a projection of a secret vector in P, we can efficiently lift it to a vector of the target norm using *Babai Nearest Plane* algorithm [6]. Running a classical sieve (see [17] for instance) on P will list all vectors of norm smaller than $\sqrt{\frac{4}{3}}\ell$, where ℓ if the norm of the $n - B$-th Gram-Schmidt vector of the reduced basis. Under the GSA assumption, we have:

$$\ell = \text{covol}(\mathcal{L})^{\frac{1}{n}} \cdot \delta_{2B+2-n} \approx \text{covol}(\mathcal{L})^{\frac{1}{n}} \cdot \left(\frac{B}{2\pi e}\right)^{1-\frac{n}{2B}}$$

Assuming that secret vectors behave as random vectors of norm $g_{\min}$, their projection on P is roughly:

$$\|\pi_P(\mathbf{b}_i)\| = \sqrt{\frac{B}{n}} \cdot g_{\min}$$

Thus, we will retrieve the projection among the sieved vectors if $\sqrt{\frac{B}{n}} \cdot g_{\min} \leqslant \sqrt{\frac{4}{3}}\ell$, that is if the following condition is fulfilled:

$$g_{\min} \leqslant \sqrt{\frac{4n}{3B}} \cdot \text{covol}(\mathcal{L})^{\frac{1}{n}} \left(\frac{B}{2\pi e}\right)^{1-\frac{n}{2B}}. \tag{3}$$

Remark.

- *This approach is similar to the one used in the security evaluation of [5] , but we use all the vectors given by the last step of sieving, resulting in a slightly stronger attack and as such more conservative parameters choices.*

- *(On the size of the enumeration window.) In the previous description we only considered the space $\mathcal{P}$, orthogonal to $\mathrm{span}(b_1, \ldots, b_{2d-B-1})$. It is natural to want to extend its dimension and choose the optimal one. It appears that for the specific parameters of our work, this optimization would only result in a difference of less than a single bit of security. Besides, on the one hand, by using the exact block size beta we can extract the vectors we need to sieve for free from the preliminary run of DBKZ, avoiding the need for an additional sieving pass. On the other hand, using a larger dimension for the additional sieving pass adds a non-negligible cost. Note that this is a consequence of the Core-SVP methodology which ignores the polynomial overhead cost of (D)BKZ.*

4.3 Hybridizing the attack for sparse secrets

We now show that we can improve this baseline when one of short vectors (in practice, the fist one) is in fact sparse. Indeed, if the sparsity level – the number of zeros in the vector – is high, then with a reasonable probability we can correctly guess the positions of some zeros of the vector. With such a good guess of positions, say all indices $\mathcal{I} \subset \{1, \ldots, n\}$, then we only need to restrict our search for the secret keys out outside this guess, that is to say, we intersect the public lattice $\mathcal{L}$ with $\mathbb{Z}^{\overline{\mathcal{I}}}$. (where $\overline{\mathcal{I}}$ refers to the complement of the set $\mathcal{I}$ in the overset $\{1, \cdots, n\}$). In this lattice, we can apply readily the methodology of Section 4.2.1 to retrieve the intersected secret and as such the secret itself. This new lattice has dimension $n - |\mathcal{I}|$ and its covolume is likely to be the same as the one of $\mathcal{L}$. (see infra for a discussion of this phenomenon). As a result, the normalized covolume of the intersection lattice is bigger than previously, and its dimension of course smaller. As such, this final lattice reduction part is now easier and thus faster. Hence, there exists a trade-off between the probability of right guessing (the more zeroes to guess, the harder it becomes to correctly guess their positions) and the time required by the lattice reduction. We now turn to the estimation of the cost of this hybrid attack.

4.3.1 Good guess probability estimation.

Before further ado, we need an estimate of the probability of making such successful guesses of the zero coefficients. Set the sparsity level of the first secret vector to be $0 < \kappa < n$ and that $|\mathcal{I}| = g$. Then, over the randomness of the secret, the probability of getting a correct guess for a given secret vector v_i is equal to the probability of I being a subset of the set of zeros of v_i, i.e. is $\binom{\kappa}{g} / \binom{n}{g}$.

4.3.2 Volume of intersection.

We now need to estimate the volume of the remaining part of the lattice, that is to say of an intersection $\mathbb{Z}^{\overline{\mathcal{I}}} \cap \mathcal{L}$. Remark that when denoting by $\pi_{\mathcal{I}}$ the orthogonal projection onto $(\mathbb{Z}^{\overline{\mathcal{I}}})^{\perp} = \mathbb{Z}^{\mathcal{I}}$:

$$\mathrm{covol}\left(\mathbb{Z}^{\overline{\mathcal{I}}} \cap \mathcal{L}\right)^{-1} = \mathrm{covol}\left(\left(\mathbb{Z}^{\overline{\mathcal{I}}} \cap \mathcal{L}\right)^{\vee}\right) = \mathrm{covol}\left(\pi_{\mathcal{I}} \cdot \mathcal{L}^{\vee}\right) = \frac{\mathrm{covol}\left(\mathcal{L}^{\vee}\right)}{\mathrm{covol}\left(\mathcal{L}^{\vee} \cap \mathbb{Z}^{\mathcal{I}}\right)}$$

Hence, we now need to evaluate the volume of the dual $\mathcal{L}^{\vee} \cap \mathbb{Z}^{\mathcal{I}}$. By definition of primitivity, if the set of vectors defined by $\mathbb{Z}^{\mathcal{I}}$ is primitive in $\mathcal{L}^{\vee}$, this intersection is exactly $\mathbb{Z}^{\mathcal{I}}$, meaning that its volume is exactly 1. Remark that this condition is realized if and only if there exists an integer matrix $X \in \mathbb{Z}^{g \times n}$ with the gcd of its principal minors equal to 1 and such that $X \cdot B^* = [e_i]_{i \in \mathcal{I}}$, for the dual basis B^* associated to a basis B. Hence, we have $X = B^T[\mathcal{I}]$. This is equivalent to requiring $B^T[\mathcal{I}]$ to be primitive in $\mathbb{Z}^n$. With probability essentially 1, the intersection has volume 1 for $\mathbf{card}(\mathcal{I}) \leqslant \frac{2n}{3}$ (see for instance [22]). As such, the normalized volume of the intersection can be considered to be $\mathrm{covol}(\mathcal{L})^{\frac{1}{n-g}}$, which is slightly larger than the original volume.

4.3.3 Putting it all together

The attack rationale can then be summarized as follows:

1. While a secret vector is not discovered do
   (a) Randomly guess the positions $\mathcal{I} \subset \{1, \ldots, n\}$ of g zeros
   (b) Compute the lattice $\mathcal{L}' = \mathcal{L} \cap \mathbb{Z}^{\bar{\mathcal{I}}}$
   (c) Reduce $\mathcal{L}'$ using BKZ-B algorithm.
   (d) Enumerate all vectors of length smaller than $\sqrt{\frac{4}{3}}\|\tilde{\mathbf{b}}_{n-B}\|$ where $\tilde{\mathbf{b}}_i$ is the $n-B$-th Gram Schmidt vector of the basis obtained at step (c).
   (e) For each such $\mathbf{v}$, lift it as a vector of $\mathcal{L}'$ using Babai's nearest plane algorithm and check if it is shorter than g_{max} and return it.

4.3.4 On sparsity.

We can evaluate the sparsity level of secret vectors by modeling them as following a discrete Gaussian centered in zero, and of standard deviation $\sigma_{\text{model}} = g_{\text{min}}/\sqrt{n}$ - which ensures an average norm g_{min} in the continuous setting.

Given $\Phi(x) = \frac{1+\text{erf}(x/\sqrt{2})}{2}$, the cumulative distribution function of the normal distribution, each coefficient is zero with probability $P(\mathrm{v}_i = 0) = \Phi(0.5/\sigma_{\text{model}}) - \Phi(-0.5/\sigma_{\text{model}})$.

Then, we evaluate the number of zero coefficients in one secret vector:

$$\kappa = n \cdot P(\mathrm{v}_0 = 0) = n \cdot (\Phi(0.5/\sigma_{\text{model}}) - \Phi(-0.5/\sigma_{\text{model}}))$$

With the promise that the parameter B is large enough so that the condition of equation 3 is non vacuous, the expected whole complexity of the attack is, therefore:

$$\mathcal{C}_{BKZ_B}(n-g) \times \frac{\binom{n}{g}}{\binom{\kappa}{g}}$$

Finding the optimal parameters g and B is done by an exhaustive search to minimize the previous quantity while still ensuring correctness. It appears in practice between this hybridized technique and the standard attack is less than a few bits, meaning that the vectors are not too sparse to deteriorate meaningfully the security of the scheme.

Remark. *Remark that we supposed here that we analyzed the attack using the most pessimistic-case possible, i.e., when the sparsity is as large as possible. As this level is likely to be smaller in practice, the hybrid attack might be worse than the naive attack. Despite this, we have made a conservative decision to incorporate the hybrid attack in our parameter selection methodology to maintain a high level of security, even in the event of advancements in lattice reduction techniques.*

4.4 Signature forgery by BDD reduction.

As a Hash-and-Sign paradigm signature, forging a signature stems from feeding a lattice point $\mathbf{v}$ at a bounded distance from a random space point $\mathbf{x}$ (in practice which is actually $(H(r\|m))$. This bounded distance decoding (BDD) problem can be solved using the so-called *Nearest-Cospace* framework developed in [20]. Under the Geometric Series assumption, Theorem 3.3 of [20] states that under the condition: $\|\mathbf{x}-\mathbf{v}\| \leqslant \left(\delta_B^n \,\text{covol}(\mathcal{L})^{\frac{1}{n}}\right)$, the decoding can be done in time $\text{Poly}(n)$ calls to a CVP oracle in dimension B.

Remark. *On the contrary to Falcon and Mitaka, we manage to reduce the security gap between forgery and key recovery to only a few bits (even less than 1 for SQUIRRELS-128), thanks to the flexibility of the choice of parameters.*

4.4.1 Additional "BUFF" Security Properties

In [13] Cremers et al. discuss three additional security properties that go beyond the security requirement of existentially unforgeable digital signatures with respect to an adaptive chosen message attack (EUF-CMA). These properties are not necessarily implied by EUF-CMA or by each other. However, following the analysis of [13] we can easily prove that SQUIRRELS provides the *message-bound signatures* (MBS) property, as Falcon does. In order to ensure malicious strong universal exclusive ownership (M-S-UEO) and no re-signing without message (NR) one can readily apply the so-called *BUFF transform* to retrieve these properties at the cost of a very slight increase in the signature size (around 5%).

5 Specification

5.1 Notations and useful definitions

General notations. We use the following notations throughout this document:

- Vectors are denoted with bold lower-case letters (e.g. $\mathbf{a}, \mathbf{v}$), matrices are denoted in bold upper-case letters (e.g. $\mathbf{A}, \mathbf{M}$). For a set D, the set of n-dimensional vectors with coordinates in D is denoted D^n, the set of matrices of n rows, m columns with coordinates in D is denoted $D^{n \times m}$.
- Given an n-dimension vector $\mathbf{v}$, its i-th coordinate is written v_i for $1 \leqslant i \leqslant n$.
- Given an n-by-m matrix $\mathbf{A}$, its (i, j)-th coordinate (the coordinate in its i-th row, j-th column) is written $\mathrm{A}_{i,j}$ for $1 \leqslant i \leqslant n, 1 \leqslant j \leqslant m$.
- The matrix multiplication is denoted $\mathbf{A} \cdot \mathbf{B}$. It is extended to n-dimensional vectors by interpreting them as a 1-by-n matrix.
- The transpose of the matrix $\mathbf{A}$ is denoted $\mathbf{A}^T$. Its orthogonal space is denoted $\mathbf{A}^\perp = \{\mathbf{x} \mid \forall \mathbf{y}, \langle \mathbf{x}, \mathbf{y} \cdot \mathbf{A} \rangle = 0\}$
- The ring of integers is denoted $\mathbb{Z}$. For a positive integer q, we denote the quotient ring of integers modulo q as $\mathbb{Z}_q = \mathbb{Z}/q\mathbb{Z}$.
- For a finite set S, we denote the uniform distribution over S as $\mathcal{U}(S)$.
- The floor of a real number a, i.e., the largest integer less than or equal to a, is denoted by $\lfloor a \rfloor$.

- For a real vector $\mathbf{v} \in \mathbb{R}^n$, its euclidean norm (i.e. ℓ_2) is denoted $||\mathbf{v}||$.
- For an integer a, and a positive integer p, we denote the reduction of a modulo p by $a \bmod p \in [0, p-1]$.
- For two n-dimensional vectors $\mathbf{a}, \mathbf{b}$ over a common ring, their inner product is denoted by $\langle \mathbf{a}, \mathbf{b} \rangle = \sum_{i=1}^{n} \mathrm{a}_i \cdot \mathrm{b}_i$.
- A lattice over the ring of integers is denoted as $\mathcal{L}$. A basis of the lattice is represented by the matrix $\mathbf{B} \in \mathbb{R}^{m \times n}$, where each row of $\mathbf{B}$ corresponds to one vector of the basis. Then, $\mathcal{L} = \{\mathbf{y} \cdot \mathbf{B} \mid \mathbf{y} \in \mathbb{Z}^n\}$.
- Given a lattice $\mathcal{L}$, and some basis $\mathbf{B}$, its covolume is denoted $\mathrm{covol}(\mathcal{L}) = \mathrm{covol}(\mathbf{B}) = \mathrm{Volume}(\mathbb{R}^n/\mathcal{L}) = \det(\mathbf{B})$.
- The dual of a lattice is denoted $\mathcal{L}^\vee = \{\mathbf{c} \in \mathrm{Span}(\mathcal{L}) \mid \forall \mathbf{x} \in \mathcal{L}, \langle \mathbf{c}, \mathbf{x} \rangle \in \mathbb{Z}\}$
- The discrete Gaussian distribution over set S, with standard deviation σ is denoted $D_{S,\sigma}$.

The Gram-Schmidt orthogonalization: Given a matrix $\mathbf{B}$, write

$$\mathbf{B} = \mathbf{L} \cdot \tilde{\mathbf{B}},$$

where $\mathbf{L}$ represents a lower triangular matrix with 1's on the diagonal, and the rows $\tilde{\mathbf{b}}_i$ of $\tilde{\mathbf{B}}$ satisfy $\langle \tilde{\mathbf{b}}_i, \tilde{\mathbf{b}}_j \rangle = 0$ for $i \neq j$. This particular decomposition, known as the Gram-Schmidt orthogonalization (or GSO), is unique when $\mathbf{B}$ is full-rank.

The Gram-Schmidt norm of $\mathbf{B}$ can be defined as follows:

$$||\mathbf{B}||_{\mathrm{GS}} = \max_{\tilde{\mathbf{b}}_i \in \tilde{\mathbf{B}}} ||\tilde{\mathbf{b}}_i||$$

Here, $||\tilde{\mathbf{b}}_i||$ represents the norm of each row $\tilde{\mathbf{b}}_i$ in $\tilde{\mathbf{B}}$. The Gram-Schmidt norm provides a measure of the magnitude of the rows after the orthogonalization process, which will be useful as being the right measure of quality for the Klein sampler (see infra.).

LLL reduction: For $\mathbf{B} = [\mathbf{b}_1 | \cdots | \mathbf{b}_n]$ a matrix, the definition of an LLL-reduced basis is as follows: define $\mu_{i,j} = \frac{\langle \mathbf{b}_i, \tilde{\mathbf{b}}_j \rangle}{\langle \tilde{\mathbf{b}}_j \cdot \tilde{\mathbf{b}}_j \rangle}$ for any $1 \leq j < i \leq n$. We say that $\mathbf{B}$ is δ LLL-reduced for a parameter $\delta \in (\frac{1}{4}, 1)$ if the following holds:

- Size-reduced: for $1 \leq j < i \leq n$: $|\mu_{i,j}| \leq \frac{1}{2}$.

- Lovász condition: for $k = 2, \cdots, n$, $\delta||\tilde{\mathbf{b}}_{k-1}||^2 \leq |\tilde{\mathbf{b}}_k||^2 + \mu_{k,k-1}^2|\tilde{\mathbf{b}}_{k-1}||^2$.

The celebrated LLL [31] algorithm outputs an LLL-reduced basis. In practice, the FPLLL [15] library is used, which is an implementation of the lazy floating-point quadratic variant (called L^2) of LLL [38].

Remark. *Although this form is not unique and its usage in the key generation could make it differ in behavior depending on the reduction algorithm used, we consider that the usage of fplll is quite generalized and offers a reliable way of LLL-reducing matrices in a deterministic way. If preferred or required by the usecase, the use of LLL in the key generation could be totally removed at the cost of a slightly slower keygen (see infra.).*

Hermite Normal Form (HNF): Any m-by-n matrix $\mathbf{M}$ can be factored as $\mathbf{U} \cdot \mathbf{H}$ with $\mathbf{U}$ being an m-by-m unimodular matrix, and $\mathbf{H}$ is an m-by-n matrix verifying:

- $\mathbf{H}$ is upper triangular with positive coefficients, and rows of zeros are below any non-zero row.
- The first non-zero entry in row i, called the pivot and noted $e_i = \mathrm{H}_{i,j_i}$ is strictly to the right of the non-zero entry of the previous row: $j_i > j_{i-1}$.
- The pivot of each row is strictly larger than all entries above it in the same column: $\forall 1 \leqslant i' < i, \mathrm{H}_{i',j_i} < \mathrm{H}_{i,j_i}$.

This definition uniquely defines the HNF of any matrix. Any algorithm computing it can thus be used in our procedure.

Our implementation uses Pernet-Stein [40] which we observed to be the fastest for large matrices, but other algorithms exist notably using lattice basis reduction [27].

5.2 Public parameters

SQUIRRELS uses public parameters in its algorithms:

1. n the dimension of the lattices sampled.
2. Bounds $g_{\text{min}} < g_{\text{max}}$ on the Gram-Schmidt norms accepted during the key generation after rounding the vectors.

3. Bounds $g_{0,\text{min}} < g_{0,\text{max}}$ on the norms of the vectors sampled in key generation, before rounding.
4. Bound e_δ controls the distance to the target determinant of the sampled basis at each step of the key generation.
5. A target determinant $\Delta = \prod_{p \in P_\Delta} p$. With P_Δ a set of primes in $[2^{30}, 2^{31}]$. We also define $l_{\text{det}} = \frac{\log(\Delta)}{n}$, which corresponds to the target Gram-Schmidt norm.
6. A bound $q \in \mathbb{N}^*$ on coefficients of hashed messages during signature generation. Messages are hashed in $[0, q-1]^n$.
7. A real bound $\lfloor \beta^2 \rfloor > 0$ on the square norm of signatures.
8. Standard deviations σ and $\sigma_{\text{min}} < \sigma_{\text{max}}$ used in Klein's sampler.
9. Integers sig_{size} and sig_{rate} used to compress the signatures.

5.3 Keys

5.3.1 Private Key

The private key in SQUIRRELS is a matrix of size $n \times n$. Each row is a vector of $\mathbf{B}$.

It is a "good" basis of the underlying lattice in the sense that it verifies the following—geometric—assumptions:

1. $\forall \tilde{\mathbf{b}}_i \in \tilde{\mathbf{B}}$, we have $g_{\text{min}} \leqslant ||\tilde{\mathbf{b}}_i|| \leqslant g_{\text{max}}$.
2. $\det(\mathbf{B}) = \Delta$.
3. $\mathbf{B}$ is co-cyclic, i.e. its HNF is of the form 1.

The private key may be recomputed dynamically from seed, but it is an expensive process so we expect that at least the last vector of $\mathbf{B}$, which is computed using extensive arithmetic, will be stored alongside the seed.

5.3.2 Public Key

The Public Key is a representation of $\mathbf{v}_{\text{check}}$ modulo every prime $p \in P_\Delta$:

$$\text{pk} = (\text{v}_{\text{check},i} \bmod p)_{1 \leqslant i \leqslant n-1, p \in P_\Delta}$$

Note that we don't need to store the last coordinate of $\mathbf{v}_{\text{check}}$ in PK as it is the determinant which is a fixed parameter.

5.4 Key pair generation

Keypair generation splits in three main steps:

1. First, we generate the first $n-1$ vectors of the secret basis.
2. Then, we compute the last secret basis vector so that the basis has the target determinant.
3. Finally we compute the row HNF of the secret basis and derive the public key from it.

During the whole process, we must carefully control the norm of the Gram-Schmidt vectors, and the expected distance to the target determinant so that the norm of the last vector is also bounded.

This procedure is described in algorithm 1.

Algorithm 1 Keygen()

Ensure: A secret key sk, a public key pk

```
while True do
    B ← GenVectors(n − 1)          ▷ Sample the n − 1 first vectors of the basis
    v_last ← ComputeLastVector(B)  ▷ v_last such that det(B ∪ {v_last}) = Δ
    if v_last = ⊥ then
        continue
    end if
    sk ← B ∪ {v_last}
    pk ← ComputePK(sk)             ▷ Verify the lattice co-cyclicity, and derive pk
    if pk = ⊥ then
        continue                   ▷ The sampled lattice is not co-cyclic
    end if
    return sk, pk
end while
```

5.4.1 Generation of the first vectors

The first step generates $n-1$ vectors, with Gram-Schmidt norms between g_{min} and g_{max}. Additionally, to bound the norm of the last Gram-Schmidt vector, as we have $||\tilde{\mathbf{b}}_n|| = \frac{\Delta}{\prod_{1 \leqslant i \leqslant n-1} ||\tilde{\mathbf{b}}_i||}$, we control the value $\delta := \sum_{1 \leqslant j < i} \log(||\tilde{\mathbf{b}}_j||) -$

$\frac{\log(\Delta)}{n} \cdot (i-1)$ at each step i so that it remains small in absolute value.

This procedure works sequentially and at each step, it samples a candidate vector $\mathbf{v} = \mathbf{v_B} + \mathbf{v_{B^\perp}}$ verifying:

- $\mathbf{v_{B^\perp}}$ is uniformly distributed among the vectors of $\mathbf{B}^\perp$ with a norm bounded between b_{low} and b_{up}.
- $\mathbf{v_B}$ follows a Gaussian distribution in $\mathbf{B}$ of standard deviation $\frac{g_{\text{max}}}{\sqrt{n}}$.

The vector $\mathbf{v}$ is then rounded to an integral vector by rounding each of its coordinates. This rounding introduces an error on the norm of the resulting Gram-Schmidt vector, but parameters are chosen so that when sampling in $[g_{0,min}, g_{0,\max}]$, the Gram-Schmidt vector after rounding will have a norm in $[g_{\text{min}}, g_{\text{max}}]$ with probability at least 90%. Thus, by default, we take $b_{\text{low}} = g_{0,\text{min}}$ and $b_{\text{up}} = g_{0,\text{max}}$ and reject vectors if their Gram-Schmidt norm is not in $[g_{\text{min}}, g_{\text{max}}]$ after rounding.

As noted before, we also control the distance to the target determinant at each step. At step i, we define the drift $\delta = \left(\sum_{1 \leqslant j < i} \log(||\tilde{\mathbf{b}}_j||)\right) - l_{\text{det}} \cdot (i-1)$. If $\delta > e_\delta$, then we update $b_{\text{up}} = \frac{b_{\text{up}} + 3 \cdot b_{\text{low}}}{4}$. If $\delta < -e_\delta$, then we update $b_{\text{low}} = \frac{3 \cdot b_{\text{up}} + b_{\text{low}}}{4}$.

This leads to algorithm 2.

5.4.2 Computation of the last secret vector

Once we have the $n-1$ first vectors, we determine a vector $\mathbf{v}_{\text{last}}$ such that the determinant of the extended basis is Δ and with a Gram-Schmidt norm in $[g_{\text{min}}, g_{\text{max}}]$.

We recall that given a matrix $\mathbf{B} \in \mathbf{R}^{n \times n}$, we can expand its determinant on its last row using minors:

$$\det(\mathbf{B}) = \sum_{1 \leqslant i \leqslant n} (-1)^{i+1} \mathrm{B}_{n,i} \cdot \text{Minor}_{n,i}(\mathbf{B})$$

where $\text{Minor}_{n,i}(\mathbf{B})$ is the determinant of the matrix $\mathbf{B}$ where we removed line n and column i.

If we can find a set of $\text{Minor}_{n,i}$ which are co-prime, using Euclid's extended algorithm we can find c_i such that $\sum_{1 \leqslant i \leqslant n} c_i \cdot \text{Minor}_{n,i} = 1$. Multiplying by Δ gives us a relation $\sum_{1 \leqslant i \leqslant n} c'_i \cdot \text{Minor}_{n,i} = \Delta$. We can then simply take $\mathbf{v}_{\text{last}} = ((-1)^{i+1} c'_i)_{1 \leqslant i \leqslant n}$.

Remark. *For efficiency, we compute only the last 4 minors: $(\text{Minor}_{n,n+1-i})_{1 \leqslant i \leqslant 4}$. We evaluated that there are co-prime with probability close to 42%, so we can restart*

Algorithm 2 GenVectors(k)

Require: A number k of vectors to generate

Ensure: k vectors, with Gram-Schmidt norms between g_{min} and g_{max}

$\mathbf{B} \leftarrow []$ ▷ Contains the sampled basis

$\tilde{\mathbf{B}} \leftarrow []$ ▷ Contains the Gram-Schmidt orthogonalization of $\mathbf{B}$

for $i = 1, ..., k$ **do**

  while True **do**

    $\delta \leftarrow \left(\sum_{1 \leqslant j < i} \log(||\tilde{\mathbf{b}}_j||)\right) - l_{\text{det}} \cdot (i-1)$ ▷ Compute the drift

    $b_{\text{up}} \leftarrow g_{0,\text{max}}$

    $b_{\text{low}} \leftarrow g_{0,\text{min}}$

    if $\delta > e_\delta$ **then**

      $b_{\text{up}} \leftarrow \frac{b_{\text{up}} + 3 \cdot b_{\text{low}}}{4}$

    end if

    if $\delta < -e_\delta$ **then**

      $b_{\text{low}} \leftarrow \frac{3 \cdot b_{\text{up}} + b_{\text{low}}}{4}$

    end if

    $\mathbf{c} \leftarrow \text{SampleOrthogonal}(\mathbf{B}, b_{\text{low}}, b_{\text{up}})$

    $\mathbf{v} \leftarrow \text{Round}(\mathbf{c})$ ▷ Round each coordinate (round-to-nearest-even)

    $\tilde{\mathbf{v}} \leftarrow \text{GramSchmidt}(\mathbf{B}, \mathbf{v})$

    if $g_{\text{min}} \leqslant ||\tilde{\mathbf{v}}|| \leqslant g_{\text{max}}$ **then**

      $\mathbf{B} \leftarrow \mathbf{B} \cup \{\mathbf{v}\}$

      $\tilde{\mathbf{B}} \leftarrow \tilde{\mathbf{B}} \cup \{\tilde{\mathbf{v}}\}$

      break

    end if

  end while

end for

return $\mathbf{B}$

Algorithm 3 SampleOrthogonal($\mathbf{B}, b_{\text{low}}, b_{\text{up}}$)

Require: $\mathbf{B}$ a set of ℓ independent vectors, $b_{\text{low}} < b_{\text{up}}$ two bounds

Ensure: A vector $\mathbf{v} = \mathbf{v_B} + \mathbf{v_{B^\perp}}$ such that $\mathbf{v_B}$ is a Gaussian vector with mean 0 and standard deviation $\frac{g_{\max}}{\sqrt{n}}$, and $\mathbf{v_{B^\perp}} \in \mathbf{B}^\perp$ uniform verifying $b_{\text{low}} \leqslant ||\mathbf{v_{B^\perp}}|| \leqslant b_{\text{up}}$

$\mathbf{v} \leftarrow \text{SampleNormal}(g_{\max}/\sqrt{n}, n)$ ▷ Sample a Gaussian vector

$\mathbf{v_{B^\perp}} \leftarrow \text{GramSchmidt}(\mathbf{B}, \mathbf{v})$ ▷ First, sample a direction in $\mathbf{B}^\perp$

$\mathbf{v_B} \leftarrow \mathbf{v} - \mathbf{v_{B^\perp}}$

$x \leftarrow$ a random double uniform in $[0, 1]$

$r \leftarrow b_{\text{low}} \cdot \left(x \cdot \left(\left(\frac{b_{\text{up}}}{b_{\text{low}}} \right)^{n-\ell} - 1 \right) + 1 \right)^{1/(n-\ell)}$ ▷ Sample a target norm

return $\mathbf{v_B} + \frac{\mathbf{v_{B^\perp}}}{||\mathbf{v_{B^\perp}}||} \cdot r$

the key generation in case they are not without affecting the scheme security. In practice, the speedup gained by only computing 4 minors instead of $n-1$ is non-negligible, even with the restarts (about 2 on average).

We also want the vector $\mathbf{v}_{\text{last}}$ to have small coefficients for efficiency so we need to reduce it:

1. we reduce the c'_i using the algorithm ComputeReducedXGCD from [34] so that their absolute value is lower than $\max((|\text{Minor}_{n,n+1-i}|/2)_{1 \leqslant i \leqslant 4}, \Delta)$.
2. we reduce the c'_i further by putting the following matrix into an LLL-reduced form (see definition in paragraph 5.1) and retrieving the row with the last coefficient equal to Δ after reduction:

$$\begin{bmatrix} c'_1 & c'_2 & c'_3 & c'_4 & \Delta \\ m_1/\gcd(m_1,m_2) & -m_2/\gcd(m_1,m_2) & 0 & 0 & 0 \\ m_1/\gcd(m_1,m_3) & 0 & -m_3/\gcd(m_1,m_3) & 0 & 0 \\ m_1/\gcd(m_1,m_4) & 0 & 0 & -m_4/\gcd(m_1,m_4) & 0 \\ 0 & m_2/\gcd(m_2,m_3) & -m_3/\gcd(m_2,m_3) & 0 & 0 \\ 0 & m_2/\gcd(m_2,m_4) & 0 & -m_4/\gcd(m_2,m_4) & 0 \\ 0 & 0 & m_3/\gcd(m_3,m_4) & -m_4/\gcd(m_3,m_4) & 0 \end{bmatrix} \tag{4}$$

Remark. *As stated previously, to put a matrix into LLL-reduced form, in practice the library fplll [15] is used. The use of LLL in this step of the key generation is not strictly necessary to sample lattices of the desired shape, but it speeds up the computation of the last vector.*

3. we apply Babai Nearest Plane algorithm [6] to $(0, ..., 0, c'_4, -c'_3, c'_2, -c'_1)$ to reduce the part of the vector in $\mathbf{B}$. The part of the vector in $\mathbf{B}^\perp$ is guaranteed to be small by the control of the drift δ in the GenVectors algorithm.

The pseudo-code of this procedure is in Algorithm 4.

Algorithm 4 ComputeLastVector($\mathbf{B}$)

Ensure: vector $\mathbf{v}_{\text{last}}$, with Gram-Schmidt norm between $g_{\min}$ and $g_{\max}$, and such that $\mathbf{B} \cup \{\mathbf{v}_{\text{last}}\}$ has determinant Δ.

$m \leftarrow []$
for $k = 1, ..., 4$ **do**
  $m \leftarrow m \cup \{\det((\mathrm{B}_{i,j})_{i \in [1,n-1], j \in [1,n] \setminus \{n+1-k\}})\}$ ▷ Computes $\text{Minor}_{n,k}(\mathbf{B})$
end for
if $\gcd(m) \neq 1$ **then**
  return $\perp$ ▷ We won't be able to find a combination of the minors equal to Δ
end if
$c' \leftarrow$ ComputeReducedXGCD(m, Δ) ▷ Algorithm from [34]

$M \leftarrow$ matrix given in equation 4
$\mathbf{c}'' \leftarrow \text{LLL}(M)[-1]$ ▷ Reduced coefficients will be the last row of the LLL reduced matrix

$\mathbf{v}_{\text{last}} \leftarrow (0, ..., 0, \mathrm{c}''_4, -\mathrm{c}''_3, \mathrm{c}''_2, -\mathrm{c}''_1)$
return Reduce$(\mathbf{v}_{\text{last}})$ ▷ Reduce $\mathbf{v}_{\text{last}}$ with Babai Nearest Plane

5.4.3 Public Key derivation

This procedure computes the row HNF of the complete secret basis $\mathbf{B}$, verifies that it is of the form 1, i.e. that the lattice is co-cyclic, and finally extracts the public

Algorithm 5 Reduce$(\mathbf{v}, \mathbf{B})$

Require: $\mathbf{v}$ a vector, $\mathbf{B} = \begin{pmatrix} \mathbf{b}_1 \\ \vdots \\ \mathbf{b}_n \end{pmatrix}$ a basis

Ensure: a vector reduced using the basis $\mathbf{B}$ and with coefficients fitting on 32 bits signed integers, or $\perp$

$m \leftarrow 0$
$m_{\text{prev}} \leftarrow -1$
while $m \neq m_{\text{prev}}$ **do**
  $m_{\text{prev}} \leftarrow m$
  $m \leftarrow \text{MaxBits}(\mathbf{v})$ ▷ Number of bits to represent the values $|v_i|$
  $\mathbf{a} \leftarrow (0, 0, ...)$ ▷ Approximation of $\mathbf{v}$ fitting on doubles
  for $i = 1, ..., n$ **do**
    $t, e \leftarrow \text{DoubleRepr}(\text{v}_i)$ ▷ Double $t \in [0, 1]$ such that $\text{v}_i \approx t \cdot 2^e$
    $\text{a}_i \leftarrow \mathbf{t} \cdot 2^{\min(53,m)-(m-e)}$
  end for
  for $i = n - 1, ..., 1$ **do**
    $c \leftarrow \left\lfloor \frac{\langle \tilde{\mathbf{b}}_i, \mathbf{a} \rangle}{\langle \tilde{\mathbf{b}}_i, \tilde{\mathbf{b}}_i \rangle} \right\rceil$ ▷ Rounding with round-to-nearest-even rule
    $\mathbf{v} \leftarrow \mathbf{v} - c \cdot \mathbf{b}_i \cdot 2^{m-\min(53,m)}$
  end for
end while
if $m < 31$ **then**
  return $\mathbf{v}$
else
  return $\perp$
end if

Algorithm 6 ComputeReducedXGCD(m, Δ)

Require: vector of minors $\mathbf{m} = (m_1, m_2, m_3, m_4)$ such that $\gcd(\mathbf{m}) = 1$

Ensure: coefficients $\mathbf{c} = (c_1, ..., c_4)$ such that $|c_i| \leqslant max(|m_i|/2, \Delta)$ and $\sum_{i=1}^{4} c_i \cdot m_i = \Delta$

$k_1, k_2, g_1 = \text{xgcd}(m_1, m_2)$

$k_3, k_4, g_2 = \text{xgcd}(m_3, m_4)$

$x_1, x_2, g = \text{xgcd}(g_1, g_2)$ ▷ $x_1 \cdot g_1 + x_2 \cdot g_2 = g$, and $g = 1$ by input constraint

$x_1 \leftarrow x_1 \cdot \Delta$ ▷ We multiply the last equation by Δ

$x_2 \leftarrow x_2 \cdot \Delta$

$x_1, x_2 \leftarrow x_1 + g_2 \cdot \lfloor \frac{x_2}{g_1} \rceil, x_2 - g_1 \cdot \lfloor \frac{x_2}{g_1} \rceil$

$k_1 \leftarrow k_1 \cdot x_1$

$k_2 \leftarrow k_2 \cdot x_1$

$c_1, c_2 \leftarrow k_1 + m_2 \cdot \lfloor \frac{k_2}{m_1} \rceil, k_2 - m_1 \cdot \lfloor \frac{k_2}{m_1} \rceil$

$k_3 \leftarrow k_3 \cdot x_2$

$k_4 \leftarrow k_4 \cdot x_2$

$c_3, c_4 \leftarrow k_3 + m_4 \cdot \lfloor \frac{k_4}{m_3} \rceil, k_4 - m_3 \cdot \lfloor \frac{k_4}{m_3} \rceil$

return $\mathbf{c} = (c_1, c_2, c_3, c_4)$

key values. We use Pernet-Stein [40] algorithm to efficiently compute a row HNF, but any algorithm can be used as the row HNF is unique.

This is done in ComputePK.

Remark. *ComputePK verifies that the lattice is co-cyclic for explicitness although this is not required as we enforce a square-free determinant when fixing public parameters, as described in the appendix B, which automatically makes the lattice co-cyclic.*

Algorithm 7 ComputePK($\mathbf{B}$)

Require: $\mathbf{B}$ a matrix of size $n \times n$, with determinant Δ

Ensure: verify that $\mathbf{B}$ corresponds to a co-cyclic lattice, and returns the corresponding public key pk

$\mathbf{A} \leftarrow \text{RowHNF}(\mathbf{B})$ ▷ In practice we use Pernet-Stein algorithm from [40]

Verify that $\mathbf{A}$ is of the form $\begin{bmatrix} \mathbf{I}_{n-1} & \mathbf{v}_{\text{check}}^T \\ \mathbf{0} & \Delta \end{bmatrix}$

if not of the above form **then**

  return $\perp$ ▷ $\mathbf{B}$ does not correspond to a co-cyclic lattice

end if

return $(\text{v}_{\text{check},i} \bmod p)_{1 \leqslant i \leqslant n, p \in P_\Delta}$

5.5 Hashing

A hash-and-sign signature scheme hashes the message before signing or verifying it. In SQUIRRELS, we need to derive a vector in $\mathbb{Z}^n$ from the message. In our procedure, we use an approved extendable-output hash function (XOF), as specified in FIPS 202 [18]. It needs to have a security level at least equal to the one targeted by our signature scheme, we use the same for all levels: SHAKE-256.

- SHAKE-256-INIT() initializes a SHAKE-256 hashing context.
- SHAKE-256-Inject(ctx, m) injects the bytes m in the hashing context ctx.
- SHAKE-256-Extract(ctx, b) extract b bits of pseudo-randomness from the hashing context ctx.

HashToPoint (algorithm 8) hashes a message into a vector of $[0, q-1]^{n-1} \times \{0\}$. It is defined for powers of two $q = 2^\ell \leqslant 2^{16}$. It extracts pseudo-random

integers from a SHAKE-256 context, interpreted in big-endian convention, and takes their modulo q to obtain pseudo randomness in $[0, q-1]$.

In the original GPV framework, this hash $\mathbf{h}$ is used to draw a syndrome from $\mathbb{Z}/\mathcal{L}$ and the public matrix $\mathbf{A}$ is then used to find a target vector $\mathbf{c}$ such that $\mathbf{A} \cdot \mathbf{c} = \mathbf{h}$. In this scheme, we directly use $\mathbf{h}$ as the target vector, and take as syndrome $\mathbf{u} = \mathbf{A} \cdot \mathbf{h}$ as inverting is too costly with the lattices we sample. We can however show that the distribution $(\mathbf{A}, \mathcal{U}([0, \Delta]))$ is indistinguishable from $(\mathbf{A}, \left(\mathcal{U}([0, q-1]^{n-1} \times \{0\})\right) \cdot \mathbf{A}^T)$.

We have $\mathbf{A} = \begin{pmatrix} \mathbf{v}_{\text{check}}^T \\ -1 \end{pmatrix}$. The underlying assumption of our scheme, **SQR-PR** (see Section 2.3.3) asserts that $\mathbf{v}_{\text{check}}$ is indistinguishable from a random vector in $\mathbb{Z}_\Delta$.

We can see that the function

$$\begin{aligned} h \colon \left(\mathbb{Z}_\Delta^{n-1} \times \{-1\}\right) \times \mathcal{X} &\longrightarrow \mathbb{Z}_\Delta \\ (\mathbf{A}, \mathbf{x}) &\longmapsto \mathbf{x} \cdot \mathbf{A}^T \bmod \Delta \end{aligned}$$

with $\mathcal{X} = [0, q-1]^{n-1} \times \{0\}$, is a 2-universal hash function.

Indeed, given $\mathbf{x}, \mathbf{x}'$ and i_0 such that $x_{i_0} \neq x'_{i_0}$, and assuming q is lower than all the prime factors of Δ, we have:

$$\begin{aligned} P_{\text{uniform } \mathbf{A}}(\mathbf{x} \cdot \mathbf{A}^T = \mathbf{x}' \cdot \mathbf{A}^T) &= P((\mathbf{x} - \mathbf{x}') \cdot \mathbf{A}^T = 0 \bmod \Delta) \\ &= P((\mathrm{x}_{i_0} - \mathrm{x}'_{i_0}) \cdot \mathrm{A}_{i_0} = -\sum_{i \neq i_0} (\mathrm{x}_i - \mathrm{x}'_i) \cdot \mathrm{A}_i \bmod \Delta) \\ &= \frac{1}{\Delta} \end{aligned}$$

indeed, as $(\mathrm{x}_{i_0} - \mathrm{x}'_{i_0})$ is invertible $\bmod \Delta$, $(\mathrm{x}_{i_0} - \mathrm{x}'_{i_0}) \cdot \mathrm{A}_{i_0}$ is uniform.

Then, we can apply the leftover hash lemma [28], the guessing probability of $\mathcal{X}$ being $\frac{1}{q^{n-1}}$, we obtain that the distinguishing advantage between $(\mathbf{A}, \mathcal{U}([0, \Delta]))$ and $(\mathbf{A}, \left(\mathcal{U}([0, q-1]^{n-1} \times \{0\})\right) \cdot \mathbf{A}^T)$ is bounded by:

$$\frac{1}{q^{n-1}} \cdot |\Delta|$$

which is lower than 2^{-6000} for all the instanciations of our scheme.

As our scheme public matrix, A is assumed indistinguishable from $\mathcal{U}(\mathbb{Z}_\Delta^{n-1} \times \{-1\})$, and up to Q_s application of the above indistinguishability result ensures that this change from the original GPV framework does not affect the security of our scheme.

Algorithm 8 HashToPoint(m, q, n)

Require: A message m, size of hashing space $q \leqslant 2^{16}$ (assumed to be a power of two), dimension n

Ensure: A vector $\mathbf{v}$ in $[0, q-1]^{n-1} \times \{0\}$.

ctx $\leftarrow$ SHAKE-256-INIT()
SHAKE-256-Inject(ctx, m)
for $i = 1, ..., n-1$ **do**

$t \leftarrow$ SHAKE-256-Extract(ctx, 16)
▷ Sample a 2-byte number, interpreted with big-endian convention
$\mathrm{v}_i \leftarrow t \bmod q$ ▷ Uniform in $[0, q-1]$ as q is a power of two
end for
$\mathrm{v}_n \leftarrow 0$
return $\mathbf{v}$

5.6 Signature generation

To sign a message m, SQUIRRELS first hashes m with a salt r to a point $\mathbf{h}$ in $[0, q-1]^{n-1} \times \{0\}$ using HashToPoint.

Then, we use Klein's trapdoor sampler to find a point in the lattice close to $\mathbf{h}$. The Sign procedure is described in algorithm 14.

Algorithm 9 KleinSampling$(\mathrm{sk}, \mathbf{t}, \sigma)$

Require: A private key sk, a target vector $\mathbf{t} \in \mathcal{Z}^n$

Ensure: A lattice vector close to $\mathbf{t}$, with distribution $D_{\mathcal{L},\sigma,\mathbf{t}}$

$\mathbf{t}_n \leftarrow \mathbf{t}$
$\mathbf{v}_n \leftarrow 0$
for $i = n, ..., 1$ **do**
$d_i \leftarrow \langle \mathbf{t}_i, \tilde{\mathbf{b}}_i \rangle / ||\tilde{\mathbf{b}}_i||^2$
$\sigma_i \leftarrow \sigma / ||\tilde{\mathbf{b}}_i||$
$z_i \leftarrow$ SamplerZ(d_i, σ_i) ▷ Gaussian sampling in $\mathbb{Z}$ with mean d_i, std σ_i
$\mathbf{t}_{i-1} \leftarrow \mathbf{t}_i - z_i \cdot \mathbf{b}_i$
$\mathbf{v}_{i-1} \leftarrow \mathbf{v}_i + z_i \cdot \mathbf{b}_i$
end for
return v_0

5.7 Sampler Over the Integers

This part is essentially the same as Falcon [41], but for completeness, we include the materials.

Let $1 \leq \sigma_{\min} \leq \sigma_{\max}$. The objective here is to demonstrate the secure sampling of Gaussian samplers $z \sim D_{\mathbb{Z},\mu,\sigma}$ for any $\sigma \in [\sigma_{\min}, \sigma_{\max}]$ and $\mu \in \mathbb{R}$. To achieve this, we utilize the SamplerZ algorithm (Algorithm 10), which incorporates the BaseSampler (Algorithm 11) and BerExp (Algorithm 12) algorithms. In our notation, we denote bitwise right-shift and AND operations as $\gg$ and $\&$, respectively. Additionally, we introduce the notations S and UniformBits:

$$\text{For any logical proposition P, } S(P) = 1 \text{ if P is true, } S(P) = 0 \text{ otherwise.} \quad (5)$$

$$\forall k \in \mathbb{Z}^+, \text{UniformBits(k) samples } z \text{ uniformly in } \{0, 1, \cdots, 2^k - 1\} \quad (6)$$

BaseSampler. Let pdt be as in Table 5.7. Our first procedure is BaseSampler (algorithm 11). It samples an integer $z_0 \in \mathbb{Z}^+$ according to the distribution χ of support $\{0, \cdots, 18\}$ uniquely defined as:

$$\forall i \in \{0, \cdots, 18\}, \quad \chi(i) = 2^{-72} \cdot \text{pdt}[i] \quad (7)$$

The distribution χ is extremely close to the "half-Gaussian" $D_{\mathbb{Z}^+,\sigma_{\max}}$ in the sense that $R_{513}(\chi || D_{\mathbb{Z}^+,\sigma_{\max}}) \leq 1 + 2^{-78}$, where R_* is the Rényi divergence. For completeness, table 5.7 provides the value of:

- the (scaled) probability distribution table pdt[i];
- the (scaled) cumulative distribution table $\text{cdt[i]} = \sum_{\text{j}\leq\text{i}} \text{pdt[j]}$;
- the (scaled) reverse cumulative distribution table $\text{RCDT[i]} = \sum_{\text{j}>\text{i}} \text{pdt[j]} = 2^{72} - \text{cdt[i]}$.

i	pdt [i]	cdt$[i]$	RCDT [i]
0	1 697 680 241 746 640 300 030	1 697 680 241 746 640 300 030	3 024 686 241 123 004 913 666
1	1 459 943 456 642 912 959 616	3 157 623 698 389 553 259 646	1 564 742 784 480 091 954 050
2	928 488 355 018 011 056 515	4 086 112 053 407 564 316 161	636 254 429 462 080 897 535
3	436 693 944 817 054 414 619	4 522 805 998 224 618 730 780	199 560 484 645 026 482 916
4	151 893 140 790 369 201 013	4 674 699 139 014 987 931 793	47 667 343 854 657 281 903
5	39 071 441 848 292 237 840	4 713 770 580 863 280 169 633	8 595 902 006 365 044 063
6	7 432 604 049 020 375 675	4 721 203 184 912 300 545 308	1 163 297 957 344 668 388
7	1 045 641 569 992 574 730	4 722 248 826 482 293 120 038	117 656 387 352 093 658
8	108 788 995 549 429 682	4 722 357 615 477 842 549 720	8 867 391 802 663 976
9	8 370 422 445 201 343	4 722 365 985 900 287 751 063	496 969 357 462 633
10	476 288 472 308 334	4 722 366 462 188 760 059 397	20 680 885 154 299
11	20 042 553 305 308	4 722 366 482 231 313 364 705	638 331 848 991
12	623 729 532 807	4 722 366 482 855 042 897 512	14 602 316 184
13	14 354 889 437	4 722 366 482 869 397 786 949	247 426 747
14	244 322 621	4 722 366 482 869 642 109 570	3 104 126
15	3 075 302	4 722 366 482 869 645 184 872	28 824
16	28 626	4 722 366 482 869 645 213 498	198
17	197	4 722 366 482 869 645 213 695	1
18	1	4 722 366 482 869 645 213 696	0

Algorithm 10 SamplerZ

Require: A mean value μ, standard deviation σ such that $\sigma \in [\sigma_{\min}, \sigma_{\max}]$

Ensure: An integer $z \in \mathbb{Z}$ sampled from a distribution very close to $D_{\mathbb{Z},\mu,\sigma}$.

```
r ← μ − ⌊μ⌋
ccs ← σ_min/σ
while True do
    z_0 ← BaseSampler()
    b ← UniformBits(8) & 0x1
    z ← b + (2 · b − 1)z_0
    x ← (z−r)^2/(2σ^2) − z_0^2/(2σ_max^2)
    if BerExp(x, ccs) = 1 then
        return z + ⌊μ⌋
    end if
end while
```

BerExp and ApproxExp. BerExp (algorithm 12) and its subroutine ApproxExp (algorithm 13) serve to peform rejection sampling.

Algorithm 11 BaseSampler

Require:

Ensure: An integer $z_0 \in \{0, 1, \cdots, 18\}$ such that $z \sim \chi$

$u \leftarrow \text{UniformBits}(72)$
$z_0 \leftarrow 0$
for $i = 0, 1, \cdots, 17$ **do**
  $z_0 \leftarrow z_0 + S(u < \text{RCDT}[i])$
end for
return z_0

Algorithm 12 BerExp(x,ccs)

Require: Floating point values $x, ccs \geq 0$

Ensure: A single bit, equal to 1 with probability $\approx ccs \cdot exp(-x)$.

$s \leftarrow \lfloor x/\ln(2) \rfloor$
$r \leftarrow x - s \cdot \ln(2)$
$s \leftarrow \min(s, 63)$
$z \leftarrow (2 \cdot \text{ApproxExp}(r, ccs) - 1) \gg s$
$i \leftarrow 64$
while True **do**
  $i \leftarrow i - 8$
  $w \leftarrow \text{UniformBits}(8) - ((z \gg i)\ \&\ 0x\mathbf{FF})$
  if $w \neq 0$ or $i \leq 0$ **then** break
  end if
end while
return $S(w < 0)$

Algorithm 13 ApproxExp(x,ccs)

Require: Floating point values $x \in [0, \ln(2)]$ and $ccs \in [0, 1]$

Ensure: An integral approximation of $2^{63} \cdot ccs \cdot \exp(-x)$

$C = [0\text{x}00000004741183\text{A}3, 0\text{x}00000036548\text{CFC}06, 0\text{x}0000024\text{FDCBF}140\text{A}, 0\text{x}0000171\text{D}939\text{DE}045, 0\text{x}0000\text{D}00\text{CF}58\text{F}6\text{F}84, 0\text{x}000680681\text{CF}796\text{E}3, 0\text{x}002\text{D}82\text{D}8305\text{B}0\text{FEA}, 0\text{x}011111110\text{E}066\text{FD}0, 0\text{x}0555555555070\text{F}00, 0\text{x}155555555581\text{FF}00, 0\text{x}400000000002\text{B}400, 0\text{x}7\text{FFFFFFFFFFF}4800, 0\text{x}8000000000000000]$

$y \leftarrow C[0]$

$z \leftarrow \lfloor 2^{63} \cdot x \rfloor$

for $u = 1, \cdots, 12$ **do**

$y \leftarrow C[u] - (z \cdot y) \gg 63$

end for

$z \leftarrow \lfloor 2^{63} \cdot ccs \rfloor$

$y \leftarrow (z \cdot y) \gg 63$

return y

Algorithm 14 Sign(m, sk, $\lfloor \beta^2 \rfloor$)

Require: A message m, a secret key sk, a bound $\lfloor \beta^2 \rfloor$

Ensure: A signature (r, s) of m

$r \leftarrow \{0, 1\}^{320}$ uniformly

$\mathbf{h} \leftarrow$ HashToPoint$(m||r, q, n)$

while true **do**

$\mathbf{c} \leftarrow$ KleinSampling(sk, $\mathbf{h}, \sigma$)

$\mathbf{sig} \leftarrow \mathbf{c} - \mathbf{h}$

if $||\mathbf{sig}||^2 > \lceil \beta^2 \rceil$ **then**

continue ▷ If signature produced is not short enough

end if

$s \leftarrow$ Compress($\mathbf{sig}, \text{sig}_{\text{size}} - 41, \text{sig}_{\text{rate}}$) ▷ 40 bytes for salt r, 1 byte for header

if $s \neq \perp$ **then**

return (r, s)

end if

end while

5.8 Signature verification

To verify a signature, we first recompute the corresponding hashed point $\mathbf{h}$ from the input message m and salt r using HashToPoint. Then we verify that the signature s has a Euclidean norm smaller than $\lfloor \beta \rfloor$ and that $\mathbf{s} + \mathbf{h}$ is indeed part of the lattice using the public key.

To verify the lattice membership of $\mathbf{s} + \mathbf{h}$, we use the equation 2, and leverage the fact that Δ is a product of distinct fixed primes with the Chinese Remainder Theorem to only do computations modulo each $p \in P_\Delta$:

$$\begin{aligned} \mathbf{c} \in \mathcal{L} &\iff \mathrm{c}_n = \sum_{i=1}^{n-1} \mathrm{c}_i \cdot \mathrm{v}_{\mathsf{check},i} \bmod \Delta \\ &\iff \forall p \in P_\Delta, \mathrm{c}_n = \sum_{i=1}^{n-1} \mathrm{c}_i \cdot \mathrm{v}_{\mathsf{check},i} \bmod p \end{aligned}$$

This results in the Algorithm Verify.

5.9 Encoding formats

5.9.1 Bits and bytes

A *byte* is a sequence of 8 *bits*. Bits in a byte are ordered from left to right, and we associate the value $\sum_{i=1}^{8} 2^{8-i} \cdot b_i$ to the byte $(b_1, ..., b_8)$.

5.9.2 Integers and doubles

Signed little-endian 32-bit integer: We represent 32-bit signed integers in little-endian representation and the first bit of the last byte represents the sign, i.e. the memory representation in bytes (b_0, b_1, b_2, b_3) encodes the integer $2^{24} \cdot (b_3 \& \overline{01111111}^2) + 2^{16} \cdot b_2 + 2^8 \cdot b_1 + b_0 - 2^{31} \cdot (b_3 >> 7)$.

Double: A double represents an approximation of a real value using 64 bits of memory. In this specification, we consider that doubles are encoded following IEEE-754 specification [2].

5.9.3 Compressed Gaussian vectors

The signature procedure of SQUIRRELS samples vectors which coefficients are distributed around 0 following a Gaussian distribution of standard deviation $\sigma \leqslant$

Algorithm 15 Verify$(m, (r, s), \text{pk}, \lfloor \beta^2 \rfloor)$

Require: A message m, a signature (r, s), a public key $\text{pk} = (\text{v}_{\text{check},i} \bmod p)_{1 \leqslant i \leqslant n-1, p \in P_\Delta}$, and a bound $\lfloor \beta^2 \rfloor$

Ensure: Accept or reject

```
h ← HashToPoint(m||r, q, n)
sig ← Decompress(s, sig_size − 41, sig_rate)   ▷ 40 bytes for salt r, 1 byte for header
if sig = ⊥ then
    return "reject"
end if
c ← s + h
result ← "accept"
for p ∈ P_Δ do
    sum ← 0
    for 1 ⩽ i ⩽ n − 1 do
        sum ← sum + c_i · (v_check,i mod p)
    end for
    if sum − c_n ≠ 0 mod p then
        result ← "reject"
    end if
end for
if ||sig||^2 > ⌊β^2⌋ then
    return "reject"
else
    return result
end if
```

$\tau_{\text{sig}} \cdot \eta_\varepsilon(\mathbb{Z}^n) \cdot g_{\max} \lesssim 1.5 \cdot g_{\max}$. Due to the shape of a Gaussian, coefficients are quite concentrated in a small interval, and having large coefficients happen rarely.

Based on this observation, it is interesting to have short representations for small coefficients and larger representations for the less likely big coefficients. The compression of signature vectors works as follows:

1. Each coefficient s_i is compressed into str_i as follows:
   (a) The first bit is the sign of s_i
   (b) Then, the next sig_{rate} bits are the sig_{rate} least significant bits of $|s_i|$. We preserve their order, i.e. from the most to the least significant.
   (c) The remaining bits of $|s_i|$ are encoded in unary, i.e. if $\lceil s_i/2^{\text{sig}_{\text{rate}}} \rceil = k$, the encoding is 0^k1.
2. The full signature is the concatenation $\text{str}_1||...||\text{str}_n$, padded with zeros to have slen bytes.

This compression algorithm is very similar to the one used in Falcon [23], except that they fix $\text{sig}_{\text{rate}} = 7$ for all the security levels.

The algorithm is specified in Compress. The corresponding decompression procedure is described in Decompress.

Algorithm 16 Compress($\mathbf{s}$, slen, rate)

Require: A vector $\mathbf{s} = (s_1, ..., s_n)$, a bit-length slen, a compression rate
Ensure: A compressed representation str of $\mathbf{s}$, or $\perp$
  for $i = 1, ..., n$ **do**
    str $\leftarrow$ str$||b$, where $b = 1$ if $s_i < 0$, else $b = 0$
    str $\leftarrow$ str$||b_{rate-1}...b_0$, where $b_j = (|s_i| >> j)$ & 1
    $k \leftarrow s_i >>$ rate
    str $\leftarrow$ str$||0^k1$
  end for
  if $|\text{str}| > 8 \cdot \text{slen}$ **then**
    return $\perp$
  end if
  return str$||0^{8 \cdot \text{slen} - |\text{str}|}$

Algorithm 17 Decompress(str, slen, rate)

Require: A bit-string $\text{str} = \text{str}[0]...\text{str}[|\text{str}|-1]$, a bit-length $slen$, a compression rate.

Ensure: A vector $\mathbf{s} = (s_1, ..., s_n)$, or $\perp$

```
if |str| ≠ slen then
    return ⊥                                   ▷ We enforce a fixed bit-length
end if
for i = 1, ..., n do
    s'_i ← Σ_{j=0}^{rate−1} 2^{rate−1−j} · str[1 + j]   ▷ We recover the lowest bits of |s_i|
    k ← 0                                      ▷ We recover the highest bits of |s_i|
    while str[rate + 1 + k] = 0 do
        k ← k + 1
    end while
    s_i ← (−1)^{str[0]} · (s'_i + 2^{rate} k)
    if s_i = 0 and str[0] = 1 then
        return ⊥                               ▷ Ensures unique encoding if s_i = 0
    end if
    str ← str[rate + 2 + k, ..., |str| − 1]    ▷ Removes bits that were encoding s_i
end for
if str ≠ 0^{|str|} then
    return ⊥
end if
return s = (s_1, ..., s_n)
```

5.9.4 Signatures

Signatures are composed of two strings r (salt) and s (compressed Gaussian vector). We define an encoding that includes both.

The first byte of the encoding is a header following the format:

$$0\,0\,1\,0\,0\,n\,n\,n$$

where the bits $n\ n\ n$ encodes the target NIST security level (1 to 5).

5.9.5 Private Keys

The private key encoding contains $\mathbf{B}$ and $\tilde{\mathbf{B}}$. $\tilde{\mathbf{B}}$ could be recomputed in the signature procedure, but that would significantly degrade its performance.

Coefficients of $\mathbf{B}$ are encoded as signed little-endian 32-bit integers. Coefficients of $\tilde{\mathbf{B}}$ are encoded as doubles, as defined in IEEE 754 [2].

The encoding algorithm is described in EncodeSK.

Algorithm 18 EncodeSK(sk)

Require: A secret basis $\mathbf{B}$, and its Gram Schmidt orthogonalization $\tilde{\mathbf{B}}$. $B_{i,j}$ is the j-th coefficient of the i-th vector of the basis.
Ensure: A bit-string representation of sk

```
s ← {}
for i = 1, ..., n do
    for j = 1, ..., n do
        s ← s||B_{i,j}        ▷ Concatenate 32-bit signed little-endian representation
    end for
end for
for i = 1, ..., n do
    for j = 1, ..., n do
        s ← s||B̃_{i,j}        ▷ Concatenate double representation from IEEE-754
    end for
end for
return s
```

5.9.6 Public Keys

The public key encoding contains each coordinates of $\mathbf{v}_{\text{check}}$ modulo $p \in P_\Delta$. We use signed little-endian 32-bit integers for this. The p in P_Δ are in the range

$[2^{30}, 2^{31}]$ so that the coefficients modulo p_i always fit in a signed 32-bit integer.

The public encoding procedure is described in EncodePK.

Algorithm 19 EncodePK(pk)

Require: The vector $\mathbf{v}_{\text{check}}$
Ensure: A binary representation of pk
  $s \leftarrow \{\}$
  for $p \in P_\Delta$ **do** ▷ Order follows the one from Appendix B
    for $i = 1, ..., n-1$ **do**
      $s \leftarrow s || (\text{v}_{\text{check},i} \bmod p)$ ▷ 32-bit signed little-endian representation
    end for
  end for
  return s

5.10 Recommended Parameters

Our scheme has a large number of public parameters, described in section 5.2, that can be tuned to precisely achieve a given security level. For each security level defined by NIST, we explored a grid of these parameters and minimized the final signature size. The selection process is reproducible and can be found in *Supporting_Documentation/additional/params.py*.

5.10.1 Interplay between parameters

We describe below relations and constraints on our parameters to ensure the correctness and security of our scheme.

Number of queries Q_s, targeted security level λ. The first parameters are the maximal number of signing queries Q_s, and the targeted security level λ. According to [1], we take $Q_s = 2^{64}$, and:

$$\begin{aligned} \lambda &= 128 \quad \text{for NIST level I and II} \\ \lambda &= 192 \quad \text{for NIST level III and IV} \\ \lambda &= 256 \quad \text{for NIST level V} \end{aligned}$$

Bounds on sampled norms $gs_{0,min}$ and $gs_{0,max}$, and lattice determinant Δ. We wish to have the bounds $g_{0,\text{min}}$ and $g_{0,\text{max}}$ as close to each other as the greater the lower bound, the greater the key recovery security, and as the lower the upper bound, the lower the length of signatures. We are however constrained

by the correction of the determinant drift: we need to ensure we can sample vectors with a norm smaller or larger than $\sqrt[n]{\Delta}$ as desired.

To correct the drift δ in the GenVectors, we sample vectors with a norm lower than $\frac{3 \cdot g_{0,\min} + g_{0,\max}}{4}$ to reduce the drift, or higher than $\frac{g_{0,\min} + 3 \cdot g_{0,\max}}{4}$ to increase the drift. For this correction to be effective, we enforce that after sampling two-thirds of the vectors the parameters verify:

$$\frac{3 \cdot g_{0,\min} + g_{0,\max}}{4} + e_{round} < \sqrt[n]{\Delta} < \frac{g_{0,\min} + 3 \cdot g_{0,\max}}{4} - 0.5$$

where e_{round} is an upper bound on the rounding error after sampling two third of the bases.
We chose as a determinant a product of large primes verifying the above inequalities.

Bounds on accepted Gram-Schmidt norms $g_{\min}$ and $g_{\max}$. These bounds extend the interval $[g_{0,\min}, g_{0,\max}]$ to take into account the rounding error and accept rounded vectors with high probability. Knowing these bounds in advance allows us to have a more precise security analysis of our scheme.

Standard deviation σ of the signatures. Signatures follow a discrete Gaussian distribution and are sampled using Klein's sampler. To ensure we lose at most $O(1)$ bits of security by using this sampler instead of a perfect distribution, it suffices to take $\varepsilon \leqslant 1/\sqrt{Q_s \cdot \lambda}$, and:

$$\begin{aligned} \sigma &= \frac{1}{\pi} \cdot \sqrt{\frac{\log(2n(1+1/\varepsilon))}{2}} \cdot g_{\max} \\ &\geqslant \eta_\varepsilon(\mathbb{Z}^n) \cdot ||\mathbf{B}||_{\text{GS}} \end{aligned}$$

Maximal norm β of the signatures. Signatures have an expected norm of $\sqrt{n} \cdot \sigma$. We reject too large signatures by using a tail-cut τ_{sig} i.e. we reject signatures of norm larger than $\beta = \tau_{\text{sig}} \cdot \sqrt{n} \cdot \sigma$.

We take $\tau_{\text{sig}} = 1.1$. Lemma 4.4 of [32] ensures that rejection happens only with a small probability.

Signature byte-length sig_{size} and sig_{rate}. Given a sig_{rate}, we evaluate the corresponding sig_{size} as the average compressed signature size of vectors of norms β. We choose the $\text{sig}_{\text{rate}} \in \{4, 5, 6, 7\}$ giving the smallest sig_{size}.

5.10.2 Concrete parameters

	SQUIRRELS-I	SQUIRRELS-II	SQUIRRELS-III
Target NIST Level	I	II	III
Lattice dimension n	1034	1164	1556
Size of hash space q	4096		
Lower bound $g_{0,\min}$	27.9	29	33.8
Upper bound $g_{0,\max}$	30.1	31	36.2
Lower bound $g_{\min}$	27.898036819196015	28.998036819196017	33.798036819196014
Upper bound $g_{\max}$	31.491273142076107	32.52421298740167	37.94718416834481
Bound e_δ	0.01		
l_{det} and Δ	See Appendix B		
Standard deviation σ	40.24667610603854	41.64307184026483	48.955191460637074
$\sigma_{\min}$	1.2780263257208286	1.2803713915043962	1.2900875923614654
$\sigma_{\max}$	1.8205		
Max signature square norm $\lfloor \beta^2 \rfloor$	2026590	2442439	4512242
Signature rate sig_{rate}	4	5	5
Key-recovery:			
BKZ block-size B	433	491	666
Core-SVP hardness (C)	126	143	194
Core-SVP hardness (Q)	114	130	176
Hybrid Key-recovery:			
Core-SVP hardness (C)	124	141	192
Core-SVP hardness (Q)	112	128	174
Forgery:			
BKZ block-size B	431	499	709
Core-SVP hardness (C)	125	145	207
Core-SVP hardness (Q)	114	132	187
Public key byte-length	681 780	874 576	1 629 640
Signature byte-length	1 019	1 147	1 554

		SQUIRRELS-IV	SQUIRRELS-V
Target NIST Level		IV	V
Lattice dimension n		1718	2056
Size of hash space q		4096	
Lower bound $g_{0,\min}$		27.8	30.7
Upper bound $g_{0,\max}$		30.2	33.3
Lower bound $g_{\min}$		27.798036819196014	30.698036819196012
Upper bound $g_{\max}$		32.47328023599738	35.78439165303195
Bound e_δ		0.01	
l_{det} and Δ		See Appendix B	
Standard deviation σ		41.956477667696724	46.460893820222594
$\sigma_{\min}$		1.2920307823164412	1.2983563971328835
$\sigma_{\max}$		1.8205	
Max signature square norm $\lfloor \beta^2 \rfloor$		3659372	5370115
Signature rate sig_{rate}		5	5
Key-recovery:	BKZ block-size B	736	887
	Core-SVP hardness (C)	214	259
	Core-SVP hardness (Q)	195	235
Hybrid Key-recovery:	Core-SVP hardness (C)	211	256
	Core-SVP hardness (Q)	192	232
Forgery:	BKZ block-size B	784	968
	Core-SVP hardness (C)	228	282
	Core-SVP hardness (Q)	207	256
Public key byte-length		1 888 700	2 786 580
Signature byte-length		1 676	2 025

6 Performance

6.1 Description of the Reference implementation

The submission package includes a reference implementation written exclusively in portable C, supporting all five security levels and adapted using a compilation flag.

These implementations use dynamically linked libraries, used only in the key generation for big integer and matrix computations:

- GMP [43]
- Flint [42]
- fplll [15]

All the computations in the key generation requiring big integers or matrix manipulations are performed using Flint structures, this includes notably computations of matrix determinant and HNF. The reference implementation also includes an efficient function to compute several minors at a time inspired by the DoubleDet algorithm from [40].

In the Verify procedure, we note that the variable sum always fits on 64-bits signed integers:

- $\mathrm{v}_{\mathrm{check},i} \bmod p$ is in $[0, 2^{31})$
- $\mathrm{c}_i = \mathrm{s}_i + \mathrm{h}_i \in [-4096, 8192)$ as $|\mathrm{s}_i| \leqslant \beta < 4096$ and $\mathrm{h}_i \in [0, q)$ (in case $|\mathrm{s}_i| > \beta$ the result is rejected later during norm checking in the procedure).
- there are $n < 4096 = 2^{12}$ summations of products $\mathrm{c}_i \cdot (\mathrm{v}_{\mathrm{check},i} \bmod p)$

So we need no more than $31 + 13 + 12 = 57$ bits to store the variable sum, plus one bit for the sign. For efficiency, we thus directly compute it on a 64-bit signed integer and reduce it modulo reduction p only once.

Due to their large or varying size, many structures of our implementations go on the heap.

6.2 Evaluation on the NIST x64 Reference Target

In this section, we summarize our performance evaluation on a 2018 laptop Lenovo Y530, equiped with Intel Core i5-8300H (8 CPU threads at 2.3GHz), 32 GB of physical RAM and running Manjaro 22.1.

The benchmark program is compiled with GCC version 12.2.1 with flags `-O3 -march=native -Ofast` and calls the NIST API `crypto_sign_keypair()`, `crypto_sign()`, and `crypto_sign_open()`. We then ran it on one CPU thread. Execution time was measured with `clock` POSIX calls, number of cycles was measured with `rdtsc` instruction.

	keygen		sign		vrfy	
	seconds	RAM (kB)	cycles/sign	sign/s	cycles/vrfy	vrfy/s
SQUIRRELS-I	74	189 776	4 230 444	545.1	201 737	11429.8
SQUIRRELS-II	120	236 356	5 305 045	434.8	244 573	9427.3
SQUIRRELS-III	306	410 152	8 977 892	257.2	430 609	5375.5
SQUIRRELS-IV	425	433 164	10 693 130	215.6	481 334	4789.2
SQUIRRELS-V	1175	626 824	41 512 206	55.6	1 048 477	2207.4

6.3 Evaluation on x64 AMD

In this section, we evaluate the performance on a 2022 Lenovo Thinkpad P14s equiped with a Ryzen Pro 7 5850U (16CPU threads at 3GHz), boost disabled, and running Manjaro 22.1.

The benchmark program was compiled with GCC version 12.2.1 with flags `-O3 -Ofast -march=native` and ran on one CPU thread.

	keygen (s)	sign		vrfy	
		cycles/sign	sign/s	cycles/vrfy	vrfy/s
SQUIRRELS-I	34	3 164 772	601.6	145 351	13099.4
SQUIRRELS-II	52	3 732 387	509.0	159 887	11871.9
SQUIRRELS-III	127	7 139 278	266.2	287 974	6594.4
SQUIRRELS-IV	179	9 097 631	208.7	329 066	5765.5
SQUIRRELS-V	351	10 670 614	177.9	481 938	3937.5

References

[1] Call for Proposals - Post-Quantum Cryptography: Digital Signature Schemes — CSRC — CSRC — csrc.nist.gov. https://csrc.nist.gov/projects/pqc-dig-sig/standardization/call-for-proposals.

[2] IEEE standard for floating-point arithmetic. Standard IEEE Std 754-2008, IEEE Computer Society, New York, NY, USA, August 2008.

[3] M. Ajtai. Generating hard instances of lattice problems (extended abstract). In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of*

Computing, New York, NY, USA, 1996. Association for Computing Machinery.

[4] Martin R. Albrecht, Léo Ducas, Gottfried Herold, Elena Kirshanova, Eamonn W. Postlethwaite, and Marc Stevens. The general sieve kernel and new records in lattice reduction. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part II*, volume 11477 of *LNCS*, pages 717–746. Springer, Heidelberg, May 2019.

[5] Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. Post-quantum key exchange - A new hope. In Thorsten Holz and Stefan Savage, editors, *USENIX Security 2016*, pages 327–343. USENIX Association, August 2016.

[6] László Babai. On lovász' lattice reduction and the nearest lattice point problem. *Comb.*, 6(1):1–13, 1986.

[7] Anja Becker, Léo Ducas, Nicolas Gama, and Thijs Laarhoven. New directions in nearest neighbor searching with applications to lattice sieving. In Robert Krauthgamer, editor, *27th SODA*, pages 10–24. ACM-SIAM, January 2016.

[8] Daniel J. Bernstein, Christoph Dobraunig, Maria Eichlseder, Scott R. Fluhrer, Stefan-Lukas Gazdag, Andreas Hülsing, Panos Kampanakis, Stefan Kölbl, Tanja Lange, Martin M. Lauridsen, Florian Mendel, Ruben Niederhagen, Christian Rechberger, Joost Rijneveld, and Peter Schwabe. SPHINCS+ submission to the NIST post-quantum project. 2017.

[9] André Chailloux and Johanna Loyer. Lattice sieving via quantum random walks. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part IV*, volume 13093 of *LNCS*, pages 63–91. Springer, Heidelberg, December 2021.

[10] David A. Cooper, Daniel C. Apon, Quynh H. Dang, Michael S. Davidson, Morris J. Dworkin, and Carl A. Miller. Recommendation for Stateful Hash-Based Signature Schemes. NIST Special Publication SP 800-208, October 2020.

[11] Ronald Cramer, Léo Ducas, Chris Peikert, and Oded Regev. Recovering short generators of principal ideals in cyclotomic rings. In *Proceedings, Part II, of the 35th Annual International Conference on Advances in Cryptology — EUROCRYPT 2016 - Volume 9666*, Berlin, Heidelberg, 2016. Springer-Verlag.

[12] Ronald Cramer, Léo Ducas, and Benjamin Wesolowski. Short stickelberger class relations and application to ideal-svp. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology – EUROCRYPT 2017*, 2017.

[13] Cas Cremers, Samed Düzlü, Rune Fiedler, Marc Fischlin, and Christian Janson. Buffing signature schemes beyond unforgeability and the case of post-quantum signatures. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 1696–1714, 2021.

[14] Dana Dachman-Soled, Léo Ducas, Huijing Gong, and Mélissa Rossi. LWE with side information: Attacks and concrete security estimation. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part II*, volume 12171 of *LNCS*, pages 329–358. Springer, Heidelberg, August 2020.

[15] The FPLLL development team. fplll, a lattice reduction library, Version: 5.4.4. Available at https://github.com/fplll/fplll, 2023.

[16] Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. Crystals-dilithium algorithm specifications and supporting documentation. 2017.

[17] Léo Ducas. Shortest vector from lattice sieving: a few dimensions for free. Cryptology ePrint Archive, Paper 2017/999, 2017. https://eprint.iacr.org/2017/999.

[18] Morris Dworkin. Sha-3 standard: Permutation-based hash and extendable-output functions, 2015-08-04 2015.

[19] Thomas Espitau, Pierre-Alain Fouque, François Gérard, Mélissa Rossi, Akira Takahashi, Mehdi Tibouchi, Alexandre Wallet, and Yang Yu. Mitaka: A simpler, parallelizable, maskable variant of falcon. In *Advances in Cryptology – EUROCRYPT 2022: 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Trondheim, Norway, May 30 – June 3, 2022, Proceedings, Part III*, page 222–253, Berlin, Heidelberg, 2022. Springer-Verlag.

[20] Thomas Espitau and Paul Kirchner. The nearest-colattice algorithm. Cryptology ePrint Archive, Report 2020/694, 2020. https://eprint.iacr.org/2020/694.

[21] Thomas Espitau, Mehdi Tibouchi, Alexandre Wallet, and Yang Yu. Shorter hash-and-sign lattice-based signatures. Cryptology ePrint Archive, Paper 2022/785, 2022. https://eprint.iacr.org/2022/785.

[22] Felix Fontein and Pawel Wocjan. On the probability of generating a lattice. *Journal of Symbolic Computation*, 64:3–15, 2014. Mathematical and computer algebra techniques in cryptology.

[23] Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Prest, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. Falcon: Fast-fourier lattice-based compact signatures over ntru. 2019.

[24] Nicolas Gama, Malika Izabachène, Phong Q. Nguyen, and Xiang Xie. Structural lattice reduction: Generalized worst-case to average-case reductions and homomorphic cryptosystems. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016*, 2016.

[25] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 197–206, 2008.

[26] Oded Goldreich, Shafi Goldwasser, and Shai Halevi. Public-key cryptosystems from lattice reduction problems. In Burton S. Kaliski Jr., editor, *CRYPTO'97*, volume 1294 of *LNCS*, pages 112–131. Springer, Heidelberg, August 1997.

[27] George Havas, Bohdan S Majewski, and Keith R Matthews. Extended gcd and hermite normal form algorithms via lattice basis reduction. *Experimental Mathematics*, 7(2):125–136, 1998.

[28] Corrigan-Gibb Henry, David J. Wu, and Kim Sam. CS 355, lecture 10: Inhomogeneous SIS and the LWE problem - Stanford University, 2018.

[29] Philip Klein. Finding the closest lattice vector when it's unusually close. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '00, page 937–941, USA, 2000. Society for Industrial and Applied Mathematics.

[30] Adeline Langlois and Damien Stehle. Worst-case to average-case reductions for module lattices. Cryptology ePrint Archive, Paper 2012/090, 2012. https://eprint.iacr.org/2012/090.

[31] Arjen K Lenstra, Hendrik Willem Lenstra, and László Lovász. Factoring polynomials with rational coefficients. *Mathematische annalen*, 261(ARTICLE):515–534, 1982.

[32] Vadim Lyubashevsky. Lattice signatures without trapdoors. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, pages 738–755, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[33] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, 2010.

[34] Bohdan S. Majewski and George Havas. The complexity of greatest common divisor computations. In Leonard M. Adleman and Ming-Deh Huang, editors, *Algorithmic Number Theory*, pages 184–193, Berlin, Heidelberg, 1994. Springer Berlin Heidelberg.

[35] Daniele Micciancio and Oded Regev. Worst-case to average-case reductions based on Gaussian measures. In *45th FOCS*, pages 372–381. IEEE Computer Society Press, October 2004.

[36] Daniele Micciancio and Michael Walter. Practical, predictable lattice basis reduction. In Marc Fischlin and Jean-Sébastien Coron, editors, *LNCS*, volume 9665, pages 820–849, Vienna, Austria, 2016. Springer, Heidelberg, Germany.

[37] Phong Q. Nguyen and Igor E. Shparlinski. Counting Co-Cyclic Lattices. *SIAM Journal on Discrete Mathematics*, 30(3):1358–1370, July 2016.

[38] Phong Q Nguyen and Damien Stehlé. An lll algorithm with quadratic complexity. *SIAM Journal on Computing*, 39(3):874–903, 2009.

[39] Azaria Paz and Claus-Peter Schnorr. Approximating integer lattices by lattices with cyclic factor groups. In *International Colloquium on Automata, Languages and Programming*, 1987.

[40] Clément Pernet and William Stein. Fast computation of hermite normal forms of random integer matrices. *Journal of Number Theory*, 130(7):1675–1683, 2010.

[41] Thomas Prest, Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Ricosset, Gregor Seiler, William

Whyte, and Zhenfei Zhang. Falcon. *Post-Quantum Cryptography Project of NIST*, 2020.

[42] The FLINT team. *FLINT: Fast Library for Number Theory*, 2023. Version 2.9.0, https://flintlib.org.

[43] The GMP team. *GNU Multiple Precision Arithmetic Library*, 2023. Version 6.2.1, https://gmplib.org/.

[44] Yang Yu and Léo Ducas. Second order statistical behavior of LLL and BKZ. In Carlisle Adams and Jan Camenisch, editors, *SAC 2017*, volume 10719 of *LNCS*, pages 3–22. Springer, Heidelberg, August 2017.

A Additional notions

A.1 Preimage samplable function (PSF)

We start by recalling the definition of *collision-resistant preimage sampleable (trapdoor) functions*, which are given by a tuple of probabilistic polynomial-time algorithms (**Trapgen**, **SampleDom**, **SamplePre**). A collection of one-way preimage sampleable functions (PSFs) satisfies the following:

1. *Generating a function with trapdoor:* **Trapgen**(1^n) outputs (a, t), where a is the description of an efficiently-computable function $f_a : D_n \to R_n$ (for some efficiently-recognizable domain D_n and range R_n depending on n), and t is some trapdoor information for f_a.
   For the remaining properties, fix some $(a, t) \leftarrow$ **Trapgen**(1^n).

2. Domain sampling with uniform output: **SampleDom**(1^n) samples an x from some (possibly non-uniform) distribution over D_n, for which the distribution of $f_a(x)$ is uniform over R_n.

3. Preimage sampling with trapdoor: for every $y \in R_n$, **SamplePre**(t, y) samples from the conditional distribution of $x \leftarrow$ **SampleDom**(1^n), given $f_a(x) = y$.

4. One-wayness without trapdoor: for any probabilistic polynomial-time algorithm A, the probability that $A(n, a, y) \in f_a^{-1}(y) \subseteq D$ is negligible, where the probability is taken over the choice of a, the target value $y \leftarrow R_n$ chosen uniformly at random, and A's random coins.

5. Preimage min-entropy: for every $y \in R_n$, the conditional min-entropy of $x \leftarrow \textbf{SampleDom}(1^n)$ given $f_a(x) = y$ is at least $\omega(\log n)$.

6. Collision resistance without trapdoor: for any probabilistic polynomial-time algorithm A, the probability that $A(n, a)$ outputs distinct $x, x' \in D_n$ such that $f_a(x) = f_a(x')$ is negligible, where the probability is taken over the choice of a and A's random coins.

A.2 Lattices and related notions

A.2.1 Lattice and their invariants

A (real) *lattice* $\mathcal{L}$ is a finitely generated free $\mathbb{Z}$-module, endowed with a Euclidean norm $\|.\|$ on the real vector space $\mathcal{L}_\mathbb{R} := \mathcal{L} \otimes_\mathbb{Z} \mathbb{R}$. By definition, there exists a finite family $(\mathbf{b}_1, \ldots, \mathbf{b}_n) \in \mathcal{L}^n$ of linearly independent elements such that $\mathcal{L} = \bigoplus_{i=1}^n \mathbf{b}_i\mathbb{Z}$, and we write $\mathcal{L} = \mathcal{L}(\mathbf{B})$, with the matrix $\mathbf{B} = [\mathbf{b}_1, \ldots, \mathbf{b}_n]$. It is called a *basis* of $\mathcal{L}$. Every basis has the same number of elements $\mathrm{rk}(\mathcal{L})$, called the *rank* of the lattice. We let $\lambda_1(\mathcal{L})$ be the Euclidean norm of a shortest non-zero vector in $\mathcal{L}$. The (co)volume is $\det \mathcal{L} = \sqrt{\det \mathbf{B}^T\mathbf{B}}$, for any basis $\mathbf{B}$ of $\mathcal{L}$.

In this work, when dealing with lattices embedded in $\mathbb{R}^n$, we only consider the standard Euclidean norm, corresponding to the canonical inner product $\langle , \rangle$, but we stress that most of our algorithms are agnostic to the choice of the norm. The dual of a lattice $\mathcal{L}$ is the lattice $\mathcal{L}^\vee = \{\mathbf{x} \in \mathcal{L}_\mathbb{R} \mid \langle \mathbf{x}, \mathbf{v} \rangle \in \mathbb{Z}, \forall\, \mathbf{v} \in \mathcal{L}\}$, and we always endow it with the same norm as $\mathcal{L}$. If $\mathcal{L}$ is a full-rank lattice of basis $\mathbf{B}$, then $\mathbf{B}^{-T}$ is a basis of $\mathcal{L}^\vee$; if it is not full rank, $\mathbf{B}(\mathbf{B}^T\mathbf{B})^{-1}$ is a basis of $\mathcal{L}^\vee$.

A.2.2 Discrete Gaussian distribution over a lattice

Let Σ be a positive definite matrix. We define $\rho_\Sigma(\mathbf{x}) = \exp(-\pi\mathbf{x}^T\Sigma^{-1}\mathbf{x})$ as the Gaussian kernel of covariance Σ. Equivalently, we could call it the standard Gaussian mass for the norm induced by Σ^{-1}. In that case, one sees that a Gaussian function is always *isotropic*, i.e., its value only depends on the designated norm of its input. When $\Sigma = s^2\mathbf{I}_n$, the subscript Σ is shortened[1] in s^2 and s is called the *width*.

Let now $\Lambda \subset \mathbb{R}^m$ of rank $n \leqslant m$. The discrete Gaussian distribution over $\mathcal{L}$

[1] Most of the folklore literature uses s or $\sqrt{\Sigma}$, that is, an analog of standard deviation instead of the covariance.

with center $\mathbf{c} \in \mathcal{L}_{\mathbb{R}}$ and covariance $\Sigma \in \mathbb{R}^{m \times m}$ is defined by the density

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

When $\mathbf{c} = \mathbf{0}$, we omit the script $\mathbf{c}$.

A.2.3 Smoothing parameter.

For a lattice $\mathcal{L}$ and real parameter $\varepsilon > 0$, the *smoothing parameter* $\eta_{\varepsilon}(\mathcal{L})$ is the smallest $s > 0$ such that $\rho_{\frac{1}{s^2}}(\mathcal{L}^{\vee}) \leqslant 1 + \varepsilon$.

Proposition A.1. *Let $\varepsilon > 0$, Δ be a fixed integer bigger than 2 and $n > 1$. Let u a uniform vector of $\mathbb{Z}_{\Delta}^{n-1}$. Then with overwhelming probability in n, the smoothing parameter of the Δ-orthogonal to u, that is to say $\mathcal{L} = \{v \in \mathbb{Z}^n | \langle u, x\rangle = 0 \mod \Delta\}$ is a $\Theta(\eta_{\varepsilon}(\mathbb{Z}^n))$.*

Proof. (sketch) Using lemma 3.3 of [35], we know that $\eta_{\varepsilon}(\mathcal{L}) \leqslant \sqrt{\frac{\ln(2n(1+1/\varepsilon))}{\pi}} \lambda_1^{\infty}(\mathcal{L}^{\vee})$. Remark that the dual of $\mathcal{L}$ can be described as $u\Delta^{-1}\mathbb{Z} + \mathbb{Z}^n$, so that its shortest vector is $\frac{u}{\Delta}$. By definition of u, its largest coefficient follows the order statistics of order $n-1$ for the uniform distribution on $\{0, \ldots, \frac{\Delta-1}{\Delta}\}$. As such the probability that its infinity norm is lower than $\frac{1}{2}$ is bounded by the probability of each coefficients to be lower than $\frac{1}{2}$, which is a $\Theta(2^{-n})$. Hence, with probability at least $1 - 2^{-n}$, we have $\lambda_q^{\infty}(\mathcal{L}^{\vee}) \geqslant \frac{1}{2}$. □

B Fixed determinants

To avoid dealing with big integers in the verification procedure, we fix the determinant to a product of large prime integers. We choose them all distinct so that we can do the computations modulo each of this prime, and conclude on a value modulo the determinant due to Chinese Remainder Theorem.

We want the n-th root of the determinant to be in the interval given in the parapraph 5.10.1. We target the middle of this interval, that we note t. We sample products of large primes (with $p \in [2^{30}, 2^{31}]$ using a pseudo-random generator until $|\log(\Delta) - n \cdot \log(t)| < 0.3$.

We obtain the following determinants for each instance of our scheme:

SQUIRRELS-I: ($l_{\text{det}} = 3.383648132136603$)

[1082849167, 1083150197, 1096975571, 1116936539, 1117258837, 1120904671, 1131206801, 1131626053, 1140174487, 1141771781, 1145176871, 1179753499, 1186833079, 1194969773, 1197479849, 1205496491,

1235961239, 1238028503, 1244100863, 1251340723, 1257590617, 1257859357, 1279650991, 1282937839, 1297120687, 1309168739, 1315178947, 1316478539, 1323389993, 1334183941, 1336785727, 1340128807, 1340972107, 1341934127, 1359972973, 1361739607, 1366698379, 1375400783, 1376077739, 1376535361, 1390372631, 1393772297, 1423430357, 1425027137, 1427978467, 1429520359, 1430069009, 1433253103, 1434713299, 1437348049, 1438256957, 1441700537, 1465612283, 1466591447, 1479434207, 1493508659, 1496250131, 1499976899, 1522112533, 1532593187, 1539070391, 1541156371, 1542752803, 1546166197, 1547595683, 1554265021, 1560616399, 1570489429, 1570535201, 1575840923, 1576535707, 1583174881, 1584936173, 1590119071, 1594809691, 1595421929, 1608753659, 1615299551, 1617331757, 1620356761, 1624998763, 1629175039, 1638175603, 1648544069, 1655236447, 1655627299, 1661300761, 1662943273, 1663542823, 1665766643, 1667748001, 1667854973, 1670638121, 1674345791, 1680892093, 1683937427, 1687878809, 1711056827, 1715166247, 1721655589, 1728028147, 1738698383, 1741433987, 1765244093, 1790315321, 1811237629, 1816543997, 1818795521, 1833440711, 1841104289, 1844828147, 1852874651, 1860470987, 1869343913, 1870736327, 1877054999, 1882045987, 1886014657, 1890532213, 1894340417, 1899233621, 1900376393, 1924522573, 1930931809, 1938795811, 1947253261, 1949541851, 1961026967, 1963399481, 1968432881, 1970806261, 1971228491, 1971726683, 1977648139, 1978001489, 1978102057, 2002754837, 2003826413, 2004535817, 2007684103, 2009441053, 2020874759, 2024420599, 2029924843, 2035508857, 2039996743, 2040125897, 2044757833, 2058941611, 2067378259, 2068584179, 2069139211, 2072445043, 2087925011, 2106678449, 2114727497, 2116937629, 2122307311, 2124313783, 2124842983, 2127157261, 2129072927, 2130204781, 2135681707, 2146505059]

SQUIRRELS-II: ($l_{\text{det}} = 3.416887402501479$)

[1076678539, 1084546769, 1084837181, 1086295073, 1092141923, 1104651083, 1108968319, 1113832081, 1118557801, 1121935567, 1123602631, 1134350887, 1140609749, 1162171067, 1162562743, 1167887597, 1172204497, 1178242309, 1185427673, 1193032891, 1196551507, 1201802411, 1213374641, 1216908071, 1222278229, 1223973593, 1225421831, 1227166877, 1251629963, 1261329779, 1262321033, 1263078491, 1269964117, 1280979787, 1281981023, 1293707647, 1294628449, 1294755233, 1295067931, 1296605083, 1299460157, 1311763199, 1312605757, 1320914611, 1322401939, 1322682497, 1324808267, 1329503041, 1334423917, 1345468127, 1346906917, 1348031159, 1369826537, 1371017147, 1387171927, 1396166789, 1401650267, 1405585813, 1406704157, 1408396469, 1408794487, 1408913749, 1410139897, 1411093129, 1412444431, 1415220857, 1419442049, 1419596621, 1421212907, 1428177481, 1428362227, 1431228647, 1431352651, 1432535851, 1432894069, 1435965397, 1436512919, 1439346011, 1446484433, 1454226437, 1457672131, 1458030943, 1462575761, 1480380457, 1487783119, 1491911591, 1493910157, 1509658963, 1510554847, 1510794781, 1512845617, 1513662061, 1519617637, 1524018073, 1536262291, 1540689133, 1546212739, 1560783541, 1563662509, 1578177739, 1585623521, 1586164351, 1590374843, 1596116563, 1601452367, 1610228339, 1617348101, 1617707653, 1620897449, 1625886173, 1634241751, 1639486643, 1639510189, 1645174943, 1653853207, 1659092359, 1667207741, 1677651673, 1678549967, 1685427451, 1689670861, 1703608141, 1706711221, 1714307117, 1725552847, 1726343063, 1732590767, 1733346379, 1733775509, 1734261623, 1739255387, 1748827921, 1752631249, 1759332103, 1774685183, 1782373013, 1786341307, 1796363893, 1800744679, 1801284869, 1806551653, 1810351957, 1818710801, 1818752491, 1821839419, 1826277571, 1827478361, 1830260513, 1845082147, 1853781539, 1858775777, 1859870611, 1859905319, 1865554283, 1869117487, 1873005287, 1882891937, 1889956973, 1908354209, 1908840499, 1910425481, 1916628761, 1926720977, 1934667173, 1938453533, 1950861337, 1956914731, 1957911493, 1958767717, 1978488521, 1981098463, 1992801821, 1993703639, 2028477119, 2030110793, 2038586387, 2047173979, 2048726557, 2051540167, 2070252491, 2070739291, 2073201373, 2080343479, 2101530833,

2106335447, 2113114147, 2119980839, 2120508163]

SQUIRRELS-III: ($l_{\text{det}} = 3.57120554551052$)

[1074596759, 1091834951, 1100703061, 1105274353, 1106015381, 1108184081, 1126934713, 1134403783, 1143221021, 1148115503, 1148518513, 1151474713, 1163181377, 1166575027, 1166935283, 1168538419, 1171498739, 1172200669, 1180646473, 1184049523, 1190909429, 1196719031, 1199389531, 1201145089, 1205086453, 1207882969, 1209898489, 1211058379, 1212788573, 1217939389, 1219434883, 1221910523, 1235014331, 1235692169, 1239227887, 1248927599, 1255745597, 1264517491, 1268799599, 1270011653, 1280402969, 1281777089, 1288420403, 1292617727, 1293746687, 1305635951, 1307897659, 1309011833, 1312550677, 1313900789, 1314276967, 1316086901, 1318521593, 1320914051, 1322373263, 1337780011, 1341501437, 1344937049, 1348393357, 1350184499, 1353038131, 1358013721, 1364512481, 1367160163, 1368041371, 1372089041, 1372118309, 1374219013, 1383734323, 1388463971, 1391440979, 1393251403, 1408008299, 1422490291, 1426883771, 1433871371, 1436563537, 1439193377, 1441794521, 1444481567, 1445171059, 1446358567, 1448022959, 1448809051, 1449299087, 1452967687, 1456543169, 1457926697, 1459637479, 1468643129, 1470005707, 1481576959, 1484280823, 1495641761, 1496060681, 1496075003, 1508157641, 1514639717, 1520733607, 1524933229, 1531809023, 1537643929, 1539369103, 1541158291, 1544947081, 1547071951, 1564365637, 1564944737, 1567973483, 1574737211, 1586939059, 1589311897, 1590398143, 1593804631, 1601310397, 1603032251, 1617138287, 1618488841, 1621381469, 1621882931, 1629171493, 1633134973, 1633200677, 1634318111, 1639608731, 1640093179, 1640711669, 1645296943, 1645444091, 1653888827, 1659622103, 1663988323, 1673223901, 1674627749, 1676123489, 1677063151, 1678134041, 1678356727, 1679218501, 1683423073, 1683814063, 1685068141, 1686528251, 1691353129, 1696929347, 1698073807, 1700051933, 1725150001, 1735814831, 1738830179, 1739801123, 1747775671, 1748499679, 1751045563, 1759811737, 1762231637, 1769727647, 1775175133, 1778468339, 1780745149, 1780759369, 1786096369, 1798069909, 1801503659, 1808902549, 1813514581, 1813669943, 1819960421, 1822669727, 1823786033, 1829302633, 1830852941, 1835534089, 1842700781, 1844642479, 1848026053, 1867069559, 1870562923, 1873784657, 1875766117, 1876371401, 1878427741, 1883724883, 1888956583, 1894782623, 1897180877, 1906119013, 1908457183, 1916774747, 1926761219, 1934779403, 1943435651, 1949029079, 1956237727, 1958327537, 1959198077, 1959881953, 1961914921, 1970396321, 1972303591, 1976731717, 1979309887, 1996420567, 2000668093, 2002974047, 2006242639, 2011788341, 2013196181, 2022544919, 2025963701, 2030934413, 2030992903, 2036491181, 2039065277, 2046210407, 2047668347, 2051108441, 2051676727, 2051987089, 2053193231, 2053592773, 2053930919, 2057284211, 2057360449, 2059444649, 2059832629, 2060405869, 2066526367, 2067955787, 2069396789, 2069655121, 2070275299, 2070924913, 2078941321, 2086903297, 2090536759, 2092830853, 2093038901, 2093382001, 2095813367, 2102626171, 2105908549, 2106702277, 2107305253, 2108978981, 2111270911, 2114557111, 2115059657, 2119853083, 2121777149, 2122711603, 2128599853, 2131236049, 2133880013, 2134333007, 2139119491, 2142288223, 2143696103, 2145728093, 2146519457, 2147055941, 2147284747]

SQUIRRELS-IV: ($l_{\text{det}} = 3.3897981925509293$)

[1078128517, 1087938437, 1090337467, 1090513979, 1091425859, 1092503603, 1095144587, 1107900851, 1110016591, 1110642887, 1122302351, 1127275309, 1132910113, 1135469273, 1136195329, 1140855239, 1141177369, 1145823167, 1146575531, 1149497417, 1160456861, 1164337637, 1168668121, 1169817533, 1170444721, 1174864739, 1180399021, 1180660939, 1184367923, 1184584243, 1189360631, 1191431977, 1192522571, 1193888383, 1197332429, 1208416939, 1212739097, 1214924273, 1219677577, 1219876751,

1220365637, 1226832799, 1235989037, 1237475749, 1239139961, 1241593757, 1243270663, 1247686871, 1250564207, 1253033333, 1254736369, 1255839301, 1258373671, 1259826877, 1260207259, 1263499849, 1266189571, 1267786171, 1270903213, 1271808623, 1272112433, 1280845619, 1281029467, 1290057667, 1291973863, 1292529577, 1297207573, 1300663963, 1311011159, 1324300933, 1324571467, 1328413477, 1331069107, 1333324673, 1348581193, 1355931107, 1363177931, 1364689933, 1364731579, 1366534501, 1369704871, 1374986311, 1390047881, 1397745289, 1405358231, 1415614987, 1426150631, 1427918147, 1433107909, 1434026623, 1437251657, 1443325153, 1443645871, 1452121483, 1454226259, 1456850627, 1458208249, 1463464927, 1463721697, 1465444699, 1466857993, 1471471319, 1478753777, 1480703797, 1483022687, 1497401677, 1499277979, 1500655097, 1502946961, 1508467333, 1516988947, 1521113149, 1523969899, 1526986969, 1532203843, 1533736657, 1540331759, 1542119407, 1545295139, 1551422689, 1552219399, 1554194753, 1556250103, 1556800717, 1564686713, 1565142373, 1565449531, 1566239329, 1571910007, 1574180891, 1574651797, 1580832697, 1582983881, 1583568803, 1588386169, 1590210257, 1592121527, 1595609293, 1597006577, 1607088817, 1608736951, 1610939137, 1611782773, 1616188727, 1618531913, 1619044787, 1628338427, 1633719179, 1640523629, 1641140531, 1642854659, 1669024289, 1671610201, 1674517211, 1682673329, 1684356467, 1692536159, 1703732941, 1715083841, 1715233939, 1715484893, 1715728153, 1717163813, 1717762009, 1721422837, 1723605613, 1727761039, 1734851249, 1743265061, 1749964693, 1750559753, 1752629167, 1756643443, 1758887681, 1761766553, 1762432757, 1764363803, 1765364753, 1765394317, 1769223721, 1773403537, 1773743201, 1779497981, 1782074909, 1784610643, 1790522827, 1791396799, 1794384407, 1796270263, 1797290977, 1800104893, 1800850781, 1802269709, 1802871401, 1807131659, 1813932091, 1815514669, 1819126951, 1822731641, 1835500231, 1838425133, 1839467407, 1839976067, 1850846197, 1863733043, 1864079263, 1865083037, 1867220323, 1868447239, 1872404917, 1886507383, 1889335183, 1889942629, 1891068847, 1898711951, 1899734917, 1899745759, 1912028023, 1914345403, 1924024513, 1925781227, 1925894609, 1926577171, 1929001693, 1930906343, 1934170753, 1948142323, 1961266777, 1963775113, 1980191351, 1980558467, 1986438577, 1989887377, 1990377929, 1990949803, 1993328191, 1993643459, 1994664907, 1996817057, 1997560127, 2003456993, 2007738833, 2007957151, 2014400177, 2016242453, 2018251343, 2026439999, 2027622907, 2030570629, 2031734863, 2037536477, 2038909349, 2039262751, 2042692129, 2047046921, 2056855531, 2063010281, 2075425151, 2076387011, 2085558599, 2088901769, 2095740667, 2097847471, 2108397601, 2118952867, 2120156893, 2121090691, 2121264349, 2121604451, 2122129649, 2122883351, 2127818509, 2130503807, 2141045869, 2146083341]

SQUIRRELS-V: ($l_{\text{det}} = 3.488127515563797$)

[1079946877, 1081459969, 1092099691, 1095194153, 1095388979, 1102018369, 1102670357, 1106171071, 1106603033, 1109797763, 1110190127, 1112195629, 1113097829, 1117791617, 1118803067, 1119350753, 1122806029, 1130613167, 1133572201, 1142596321, 1143466171, 1156172387, 1156910219, 1159990549, 1161256639, 1162157627, 1162958789, 1171821367, 1172023807, 1173828881, 1175043679, 1176798443, 1181376611, 1185841883, 1186036853, 1189200413, 1189749863, 1189917413, 1191835247, 1192150537, 1194403943, 1195722067, 1196490089, 1197993887, 1198262393, 1198984357, 1199801657, 1201331921, 1203885373, 1204174259, 1210655729, 1216157291, 1216248743, 1216339261, 1222802047, 1222937477, 1223927437, 1225056611, 1227112111, 1231307191, 1231675933, 1231953647, 1232614853, 1235165171, 1237179467, 1240107343, 1241353571, 1242003407, 1247729837, 1248453431, 1249776089, 1250768719, 1251659009, 1252273777, 1255671097, 1255776217, 1261126333, 1267445713, 1267667263, 1269451861, 1270931143, 1271242571, 1272231509, 1280165371, 1286334977, 1287773897, 1290437957, 1296433267, 1296608809, 1302411421, 1311762761, 1312444451, 1314076573, 1314441347, 1316576153, 1328405269,

1331057401, 1331246099, 1332208651, 1338410299, 1338852971, 1341903943, 1341978481, 1346399029, 1349114329, 1349659277, 1350234661, 1351168141, 1351289087, 1351643791, 1352338577, 1353006133, 1354440947, 1357660673, 1363484567, 1366760539, 1381130087, 1388743303, 1394356267, 1398527693, 1399272691, 1400258677, 1409526047, 1411868963, 1412221667, 1412746243, 1416477529, 1420218809, 1430971079, 1442480257, 1444779143, 1446331097, 1446475001, 1448979503, 1449840043, 1452189463, 1456357219, 1459279517, 1461154421, 1463609249, 1465439869, 1469776589, 1474113479, 1481057267, 1481284279, 1484311343, 1490150339, 1492643227, 1493978251, 1494905011, 1496850317, 1498077169, 1500375011, 1503507277, 1509081869, 1514442217, 1515555893, 1520520427, 1522779373, 1525135361, 1527767743, 1528632421, 1530062273, 1532783683, 1533809677, 1538155343, 1542866951, 1543515583, 1545122899, 1548658387, 1554761179, 1554794621, 1554994537, 1566175697, 1566789661, 1569172991, 1569685693, 1573761347, 1576945613, 1578707113, 1584043421, 1584330073, 1590422657, 1595323337, 1598555417, 1600783369, 1603580141, 1605032537, 1606450217, 1607404223, 1609925579, 1610775773, 1610971343, 1611068497, 1618767151, 1620010477, 1620400843, 1623489619, 1636482493, 1636547669, 1640346613, 1645225339, 1649818189, 1652533061, 1664213669, 1665305539, 1669871557, 1684106701, 1693338277, 1697967377, 1700734579, 1711114931, 1712690611, 1714900639, 1718697293, 1723369541, 1730685937, 1734000661, 1735591733, 1739451457, 1744830697, 1746433357, 1747672259, 1751903281, 1755161459, 1756186123, 1757298113, 1763350261, 1769925629, 1771024201, 1773746467, 1778535653, 1778988839, 1781556613, 1784142247, 1785764683, 1793195533, 1797406469, 1799850137, 1804965527, 1805239699, 1805761271, 1805967451, 1812214903, 1812879293, 1817027659, 1821029543, 1822023319, 1822911913, 1826240389, 1827570599, 1828465159, 1831249529, 1843157971, 1844910731, 1846903997, 1851213433, 1851618599, 1854042109, 1854104519, 1856135597, 1857104971, 1857744613, 1858130927, 1858397963, 1859225891, 1862936443, 1867235779, 1868373011, 1868708287, 1872303547, 1873720889, 1883080279, 1886257369, 1887742231, 1888501891, 1890661657, 1901870039, 1903420201, 1905187967, 1915314377, 1918278731, 1919263933, 1924364699, 1925579221, 1925754071, 1930960027, 1932735419, 1933320229, 1936304291, 1947131887, 1949404757, 1950668197, 1951497893, 1951539323, 1952573453, 1955534821, 1958831491, 1964086759, 1971674329, 1976925143, 1980128929, 1992070477, 1997525029, 2001255479, 2002916957, 2003114923, 2005999217, 2012604947, 2015267599, 2015581691, 2018635609, 2019884627, 2021742241, 2022692407, 2025227719, 2030897851, 2031634909, 2033313097, 2035746233, 2035842773, 2036979887, 2049007073, 2055246779, 2064774073, 2065106947, 2067772019, 2076616753, 2083591271, 2090308141, 2092051667, 2093442763, 2107489127, 2109605053, 2119371857, 2120126543, 2120160541, 2125289717, 2129853763]