Hawk: Module LIP makes Lattice Signatures Fast, Compact and Simple

Léo Ducas[1,2], Eamonn W. Postlethwaite[1], Ludo N. Pulles[1], and Wessel van Woerden[1]

[1] CWI, Cryptology Group, Amsterdam, the Netherlands
[2] Mathematical Institute, Leiden University, Leiden, The Netherlands

Abstract. We propose the signature scheme Hawk, a concrete instantiation of proposals to use the Lattice Isomorphism Problem (LIP) as a foundation for cryptography that focuses on simplicity. This simplicity stems from LIP, which allows the use of lattices such as $\mathbb{Z}^n$, leading to signature algorithms with no floats, no rejection sampling, and compact precomputed distributions. Such design features are desirable for constrained devices, and when computing signatures inside FHE or MPC. The most significant change from recent LIP proposals is the use of module lattices, reusing algorithms and ideas from NTRUSign and Falcon. Its simplicity makes Hawk competitive. We provide cryptanalysis with experimental evidence for the design of Hawk and implement two parameter sets, Hawk-512 and Hawk-1024. Signing using Hawk-512 and Hawk-1024 is four times faster than Falcon on `x86` architectures, produces signatures that are about 15% more compact, and is slightly more secure against forgeries by lattice reduction attacks. When floating-points are unavailable, Hawk signs 15 times faster than Falcon.
We provide a worst case to average case reduction for module LIP. For certain parametrisations of Hawk this applies to secret key recovery and we reduce signature forgery in the random oracle model to a new problem called the one more short vector problem.
Keywords: Post-Quantum Cryptography, Signatures, Module Lattice Isomorphism Problem, Concrete Design, Quadratic Forms.

1 Introduction

Background. Currently the most efficient lattice based signature scheme, and more generally, one of the most efficient post-quantum signature schemes, is Falcon [32]. Like its predecessor NTRUSign it has a hash-then-sign design, but fixes the issue of signature transcript leakage [27] via Discrete Gaussian Sampling (DGS) [19].

Since its introduction much progress has been made into making DGS more efficient [12,11,18], in particular by exploiting ideal or module structures [28,14] such as those of NTRU lattices. Nonetheless, DGS remains particularly difficult to implement securely and efficiently, especially on constrained devices, and even more so when side-channel attacks are a concern. In particular, DGS involves

high precision floating-point linear algebra and the evaluation of transcendental functions. A decade of research has not provided an entirely satisfactory solution to such issues.

Recently an idea emerged: use a simple lattice, maybe as simple as $\mathbb{Z}^n$ [15,7]. More precisely, use a hidden rotation of it. The idea is to base security on the problem of finding isometries between lattices, i.e. the Lattice Isomorphism Problem (LIP). While this is not only motivation for LIP based cryptography, it was noted in [15] that this avoids the difficult DGS step above: sampling from the $\mathbb{Z}^n$ lattice is much easier.

This work. The work [15], introducing the LIP based cryptography framework, mostly focused on theoretical and asymptotic results. In our work we give a concrete instantiation of their approach, based on simple module lattices, to see if it is practical and competitive. An attractive choice would be to consider the most structured option, namely modules of rank one (ideal lattices) over number fields, however this restricted version of LIP is known to be solvable in classical polynomial time [20,24].

Instead we work with rank two modules, for which the LIP problem has already received some cryptanalytic attention [33]. It was quickly noted that NTRUSign signatures [22] were leaking the Gram matrix of the secret key; recovering the secret key from this Gram matrix is precisely LIP. While the NTRUSign scheme was ultimately broken, it was only by exploiting a stronger form of leakage, not by solving LIP. In conclusion this module LIP problem is plausibly hard and is clear and simple to state, and therefore appears as a legitimate basis for cryptography.

We consider the ring $R = \mathbb{Z}[X]/(X^n + 1)$ for n a power of two, that is the ring of integers for some power of two cyclotomic field. This ring is naturally viewed as an orthogonal lattice. We must then generate a basis of R^2 following some distribution, which we achieve by mimicking NTRUSign key generation and setting the modulus $q = 1$. This allows us to make use of efficient techniques from the literature [22,29,32]. Following the ideas presented in [15] we are able to show that sampling our keys in this manner gives a worst case to average case reduction for module LIP. However, this reduction is limited to a large choice of the parameter that determines the sampling of the public key. In Hawk we make more aggressive choices based on heuristic and experimental cryptanalysis.

The original design of [15] hashed a message to $\{0, \frac{1}{q}, \ldots, \frac{q-1}{q}\}^{2n}$ for some $q = \mathsf{poly}(n)$. Another optimisation we propose is to hash the message to a smaller target space $\{0, \frac{1}{2}\}^{2n}$ to further simplify Gaussian sampling. For this variant we provide a reduction in the programmable random oracle model to a new problem: one more (approximate) SVP. This reduction also requires a specific choice of parameters, and again Hawk makes more aggressive choices. This problem is similar to the recently introduced one more inhomogenous short integer solution problem [1] used to design blind signature schemes from lattices.

We also propose efficient encodings for the public key and signatures of our scheme. Decoding the keys is cheap and recovering redundant parts is done ef-

	[32] FALCON 512	This work HAWK 512	Gain ($\frac{\text{FALCON}}{\text{HAWK}}$)	[32] FALCON 1024	This work HAWK 1024	Gain ($\frac{\text{FALCON}}{\text{HAWK}}$)
AVX2 KGen	7.95 ms	4.25 ms	×1.87	23.60 ms	17.88 ms	×1.32
Reference KGen	19.32 ms	13.14 ms	×1.47	54.65 ms	41.39 ms	×1.32
AVX2 Sign	193 µs	50 µs	× 3.9	382 µs	99 µs	× 3.9
Reference Sign	2449 µs	168 µs	×14.6	5273 µs	343 µs	×15.4
AVX2 Vf	50 µs	19 µs	×2.63	99 µs	46 µs	×2.15
Reference Vf	53 µs	178 µs	×0.30	105 µs	392 µs	×0.27
Secret key (bytes)	1281	1153	×1.11	2305	2561	×0.90
Public key (bytes)	897	1006 ± 6	×0.89	1793	2329 ± 11	×0.77
Signature (bytes)	652 ± 3	542 ± 4	×1.21	1261 ± 4	1195 ± 6	×1.05

Table 1: Performance of FALCON and HAWK for $n = 512, 1024$ on an Intel® Core™ i5-4590 @3.30GHz processor with TurboBoost disabled. HAWK was compiled with `-O2` and FALCON with `-O3`. The Sign timings correspond to batch usage; "Gain" is more favourable for HAWK in unbatched usage, see Section 5.4.

ficiently with a few number theoretic or fast Fourier transforms. Moreover, we significantly compress the signature by dropping half of it, which is effectively computationally free. Decompressing a signature uses Babai's round-off algorithm [4]. This decompression uses public data during verification, so it is not a target for side-channel or statistical attacks and does not require masking. Its use of rounding also allows us to avoid the need for high precision floats.

Performance and comparison. Following FALCON, we propose a reference implementation and an AVX2 optimised implementation. The reference implementation makes no use of floating-points (though it emulates them during key generation), whereas the AVX2 version uses floating-points.

On AVX2 CPUs, HAWK-512 outperforms FALCON-512 by a factor of about 2 for key generation and verification and a factor of 4 for signing. The situation is similar for HAWK-1024. Without floats, HAWK signs 15 times faster than FALCON, because HAWK uses number theoretic transforms in signing while FALCON emulates floating-points. The verification contains a fast decompression that uses fixed-point arithmetic but uses two number theoretic transforms making it slightly slower than FALCON. Because the numbers are smaller in HAWK's secret key, key generation is faster with HAWK.

Regarding compactness, HAWK-512 signatures are about 110 bytes shorter, but public keys about 110 bytes larger, than FALCON-512; this puts HAWK-512 on par for certificate chain applications, and should be advantageous for other applications. Additionally, secret keys are 128 bytes smaller. In HAWK-1024 we save a little on signatures, but our keys are larger.

We also note that Hawk resists forgery attacks a little better than Falcon. This is a direct result of being able to use the secret key to efficiently sample slightly smaller signatures in $\mathbb{Z}^{2n}$ than is possible in an NTRU lattice.

The recent variant of Falcon named Mitaka [17] also aims to make the signing procedure simpler and free from floating-point arithmetic. They achieve this with some loss in the signing quality compared to Falcon which makes signature forgeries somewhat easier, but their floating-point implementation signs twice as fast. In contrast, by using $\mathbb{Z}^{2n}$ we obtain an even simpler sampler while simultaneously improving the signing quality, efficiency and signature size.

Simplicity as a circuit. We claim that our signature scheme is simpler as a circuit than Falcon and therefore expect the performance gap to be larger on constrained architectures. In fact, we hope that Hawk or a variant of Hawk may be simple enough to be implemented within a Fully Homomorphic Encryption scheme for applications such as blind or threshold signatures [2]. It might also be easier to mask against side-channel attacks, similarly to how the lack of floating-points in the sampler simplifies the masking of Mitaka$_{\mathbb{Z}}$ [17, Sec. 7.3].

Implementation and source code. Our constant-time C implementation and auxiliary scripts are open source.[1] Included is a SageMath implementation of Hawk.

Roadmap. Section 2 introduces some preliminaries. Section 3 introduces the signature scheme Hawk. Section 4 details our concrete cryptanalytic model for Hawk. Section 5 details the parameters for Hawk, its estimated security, and explains implementation and performance details. Section 6 provides a worst case to average case reduction for smLIP, the search module LIP problem that underlies our key generation design. Throughout references to appendices can be found in the full version of this report [16], where we provide more information on formal reductions and our implementation.

1.1 Acknowledgments

The authors thank Nick Genise, Shane Gibbons, Thomas Prest, Noah Stephens-Davidowitz and the anonymous reviewers for helpful discussions and useful feedback. W. van Woerden was supported by the ERC-ADG-ALGSTRONGCRYPTO project (no. 740972). The research of L. Ducas and E.W. Postlethwaite was supported by the European Union's H2020 Programme under PROMETHEUS project (grant 780701). L. Ducas and L.N. Pulles were supported by the ERC-StG-ARTICULATE project (no. 947821).

2 Preliminaries

We use bold lowercase letters $\mathbf{v}$ to denote column vectors. Bold uppercase letters $\mathbf{B}$ represent matrices, and $\mathbf{B}^\mathsf{T}$ is the transpose. For a real matrix $\mathbf{B}$ let $\tilde{\mathbf{B}}$ denote

[1] `https://github.com/ludopulles/hawk-aux`

the related Gram–Schmidt matrix. Let $[n] = \{1, \ldots, n\}$ for $n \in \mathbb{Z}_{\geq 1}$. Let log without subscript denote the natural logarithm.

Lattices and quadratic forms. A full rank, n dimensional lattice Λ is a discrete subgroup of $\mathbb{R}^n$ and is given by a basis $\mathbf{B} \in \mathbb{R}^{n \times n}$ of $\mathbb{R}$-linearly independent column vectors. A lattice defined by $\mathbf{B}$ is $\Lambda(\mathbf{B}) = \{\mathbf{B} \cdot \mathbf{x} \colon \mathbf{x} \in \mathbb{Z}^n\}$. Denote by $\lambda_i(\Lambda)$ the i^{th} minima of Λ. This is the smallest radius of a centred and closed ball such that its intersection with Λ contains i linearly independent vectors. Two bases $\mathbf{B}, \mathbf{B}'$ generate the same lattice if there exists a unimodular matrix $\mathbf{U} \in \mathrm{GL}_n(\mathbb{Z})$ such that $\mathbf{B} \cdot \mathbf{U} = \mathbf{B}'$. Two lattices Λ, Λ' are *isomorphic* if there exists an orthonormal transformation $\mathbf{O} \in O_n(\mathbb{R})$ such that $\mathbf{O} \cdot \Lambda = \Lambda'$. Recovering this transformation is the Lattice Isomorphism Problem (LIP).

Definition 1 (Lattice Isomorphism Problem). *Given two isomorphic lattices Λ, Λ', find $\mathbf{O} \in O_n(\mathbb{R})$ such that $\mathbf{O} \cdot \Lambda = \{\mathbf{O} \cdot \mathbf{v} \colon \mathbf{v} \in \Lambda\} = \Lambda'$.*

If Λ, Λ' are generated by $\mathbf{B}, \mathbf{B}'$ respectively, then they are isomorphic if there exists an orthonormal transformation $\mathbf{O} \in O_n(\mathbb{R})$ and a unimodular matrix $\mathbf{U} \in \mathrm{GL}_n(\mathbb{Z})$ such that $\mathbf{O} \cdot \mathbf{B} \cdot \mathbf{U} = \mathbf{B}'$. We can remove the real valued orthonormal transformation by moving to quadratic forms. A quadratic form is a positive definite real symmetric matrix $\mathbf{Q} \in \mathcal{S}_n^{>0}(\mathbb{R})$. For any lattice basis $\mathbf{B}$ the Gram matrix $\mathbf{B}^\mathsf{T}\mathbf{B}$, consisting of all pairwise inner products, is a quadratic form. Conversely, given a quadratic form $\mathbf{Q}$, Cholesky decomposition finds a basis $\mathbf{B}_\mathbf{Q}$ such that $\mathbf{B}_\mathbf{Q}^\mathsf{T} \cdot \mathbf{B}_\mathbf{Q} = \mathbf{Q}$ and $\mathbf{B}_\mathbf{Q}$ is an upper triangular matrix. Two quadratic forms $\mathbf{Q}, \mathbf{Q}' \in \mathcal{S}_n^{>0}(\mathbb{R})$ are *equivalent* if there exists a unimodular $\mathbf{U} \in \mathrm{GL}_n(\mathbb{Z})$ such that $\mathbf{U}^\mathsf{T} \cdot \mathbf{Q} \cdot \mathbf{U} = \mathbf{Q}'$. We have that two lattices are isomorphic if and only if their Gram matrices are equivalent; this allows us to restate LIP.

Definition 2 (LIP, restated). *Given two equivalent forms $\mathbf{Q}, \mathbf{Q}'$, find $\mathbf{U} \in \mathrm{GL}_n(\mathbb{Z})$ such that $\mathbf{U}^\mathsf{T} \cdot \mathbf{Q} \cdot \mathbf{U} = \mathbf{Q}'$.*

The inner product with respect to $\mathbf{Q} \in \mathcal{S}_n^{>0}(\mathbb{R})$ is defined as $\langle \cdot, \cdot \rangle_\mathbf{Q} : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$, $(\mathbf{x}, \mathbf{y}) \mapsto \mathbf{x}^\mathsf{T} \cdot \mathbf{Q} \cdot \mathbf{y}$. The norm with respect to $\mathbf{Q} \in \mathcal{S}_n^{>0}(\mathbb{R})$ is defined as $\|\mathbf{x}\|_\mathbf{Q} = \sqrt{\langle \mathbf{x}, \mathbf{x} \rangle_\mathbf{Q}}$. Note that for a basis $\mathbf{B}$ and vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ we have

$$\langle \mathbf{B}\mathbf{x}, \mathbf{B}\mathbf{y} \rangle = \mathbf{x}^\mathsf{T}\mathbf{B}^\mathsf{T}\mathbf{B}\mathbf{y} = \langle \mathbf{x}, \mathbf{y} \rangle_{\mathbf{B}^\mathsf{T}\mathbf{B}} ,$$

and thus the geometry of $\Lambda(\mathbf{B})$ is fully described by $\mathbf{Q} = \mathbf{B}^\mathsf{T}\mathbf{B}$. Moving from lattices to quadratic forms can be viewed as forgetting about the specific embedding of the lattice in $\mathbb{R}^n$, while maintaining all geometric information. Throughout the paper we will talk about lattices and quadratic forms interchangeably.

Discrete Gaussian sampling and smoothing. Given a parameter $\sigma \in \mathbb{R}_{>0}$, we define the Gaussian mass $\rho_\sigma \colon \mathbb{R}^n \to \mathbb{R}$, $\mathbf{x} \mapsto \exp\left(-\|\mathbf{x}\|^2 / 2\sigma^2\right)$. For any $\mathbf{c} \in \mathbb{R}^n$ we denote the discrete Gaussian distribution on $\Lambda + \mathbf{c}$ with parameter σ by $D_{\Lambda+\mathbf{c},\sigma}$ which assigns the probability $\rho_\sigma(\mathbf{x}) / \sum_{\mathbf{y} \in \Lambda+\mathbf{c}} \rho_\sigma(\mathbf{y})$ to a point $\mathbf{x} \in \Lambda+\mathbf{c}$, and zero otherwise. We also define a Gaussian mass with respect to $\mathbf{Q} \in \mathcal{S}_n^{>0}(\mathbb{R})$

as $\rho_{\mathbf{Q},\sigma}\colon \mathbb{R}^n \to \mathbb{R}, \mathbf{x} \mapsto \exp\left(-\|\mathbf{x}\|_{\mathbf{Q}}^2/2\sigma^2\right)$. For any $\mathbf{c} \in \mathbb{R}^n$ we denote the discrete Gaussian distribution on $\mathbb{Z}^n + \mathbf{c}$ with respect to $\mathbf{Q}$ and parameter σ by $D_{\mathbf{Q},\mathbb{Z}^n+\mathbf{c},\sigma}$, which assigns a probability $\rho_{\mathbf{Q},\sigma}(\mathbf{x}) / \sum_{\mathbf{y}\in\mathbb{Z}^n+\mathbf{c}} \rho_{\mathbf{Q},\sigma}(\mathbf{y})$ to a point $\mathbf{x} \in \mathbb{Z}^n + \mathbf{c}$, and zero otherwise. If $\mathbf{c} \in \mathbb{Z}^n$ we write $D_{\mathbf{Q},\sigma}$. If $\mathbf{Q} = \mathbf{B}^\mathsf{T} \cdot \mathbf{B}$, note that $D_{\mathbf{Q},\mathbb{Z}^n+\mathbf{c},\sigma}(\mathbf{x}) = \mathbf{B}^{-1} \cdot D_{\Lambda(\mathbf{B})+\mathbf{B}\cdot\mathbf{c},\sigma}(\mathbf{B}\cdot\mathbf{x})$, as $\rho_{\mathbf{Q},\sigma}(\mathbf{x}) = \rho_\sigma(\mathbf{B}\cdot\mathbf{x})$. When σ is large enough compared to the maximum length of a Gram–Schmidt basis vector, we can efficiently sample a discrete Gaussian.

Lemma 1 ([8, Lem. 2.3], adapted). *There is a PPT algorithm that on input a quadratic form $\mathbf{Q} \in \mathcal{S}_n^{>0}(\mathbb{R})$, $\mathbf{c} \in \mathbb{R}^n$ and parameter $\sigma \geq \left\|\tilde{\mathbf{B}}_{\mathbf{Q}}\right\| \cdot (1/\pi) \cdot \sqrt{\log(2n+4)/2}$ outputs a sample according to $D_{\mathbf{Q},\mathbb{Z}^n+\mathbf{c},\sigma}$.*

A discrete Gaussian has a similar tail bound to a continuous Gaussian.

Lemma 2 ([6, Lem. 1.5(ii)]). *For any lattice $\Lambda \subset \mathbb{R}^n$, point $\mathbf{c} \in \mathbb{R}^n$ and $\tau \geq 1$, we have*

$$\Pr_{\mathbf{x}\sim D_{\Lambda+\mathbf{c},\sigma}}\left[\|\mathbf{x}\| > \tau\sigma\sqrt{n}\right] \leq 2\frac{\rho_\sigma(\Lambda)}{\rho_\sigma(\Lambda+\mathbf{c})} \cdot \tau^n e^{-\frac{n}{2}(\tau^2-1)}.$$

Definition 3. *Let $\hat{\Lambda}$ denote the dual of Λ. For $\varepsilon > 0$ we define the* smoothing parameter $\eta_\varepsilon(\Lambda)$ *as the smallest $\sigma \in \mathbb{R}_{>0}$ such that $\rho_{1/(2\pi\sigma)}\left(\hat{\Lambda} \setminus \{\mathbf{0}\}\right) \leq \varepsilon$.*

Note that η_ε is usually defined with respect to a width $s = \sqrt{2\pi}\sigma$. Here its value is a factor $\sqrt{2\pi}$ smaller than usual. If $\sigma \geq \eta_\varepsilon(\Lambda)$ then $D_{\Lambda+\mathbf{c},\sigma}$ exhibits several useful properties. For example, σ is close to the standard deviation of $D_{\Lambda+\mathbf{c},\sigma}$, with the closeness parametrised by ε, see [26, Lem. 4.3], and cosets have similar weights. We may say σ is 'above smoothing' to refer to $\sigma \geq \eta_\varepsilon(\Lambda)$ for some implicit appropriate ε.

Lemma 3 ([26, Proof of Lem. 4.4]). *For any lattice $\Lambda \subset \mathbb{R}^n$, point $\mathbf{c} \in \mathbb{R}^n$, and $\varepsilon \in (0,1)$, $\sigma \geq \eta_\varepsilon(\Lambda)$, we have*

$$(1-\varepsilon) \cdot \frac{\left(\sqrt{2\pi}\cdot\sigma\right)^n}{\det(\Lambda)} \leq \rho_\sigma(\Lambda+\mathbf{c}) = \rho_{\mathbf{Q},\sigma}(\mathbb{Z}^n+\mathbf{c}') \leq (1+\varepsilon) \cdot \frac{\left(\sqrt{2\pi}\cdot\sigma\right)^n}{\det(\Lambda)},$$

where $\mathbf{Q} = \mathbf{B}^\mathsf{T}\mathbf{B}$ and $\mathbf{c}' = \mathbf{B}^{-1}\mathbf{c}$ for any basis $\mathbf{B}$ of Λ.

Module lattices and Hermitian forms. A number field $\mathbb{K}$ is an algebraic extension of $\mathbb{Q}$ of finite degree $n = [\mathbb{K} : \mathbb{Q}]$. We write $\mathcal{O}_\mathbb{K}$ for the ring of integers of a general number field. In this work, we consider the cyclotomic field $\mathbb{K} = \mathbb{Q}(\zeta_{2n}) = \mathbb{Q}\left(e^{-2\pi i/2n}\right) \cong \mathbb{Q}[X]/(X^n+1)$ where $n \geq 2$ is a power of two. This is a CM field and has conductor $m = 2n$. Many of the facts below are not true for general number fields. The ring of integers $R \cong \mathbb{Z}[X]/(X^n+1)$ of $\mathbb{K}$, or any ideal of it, is a rank n lattice. Indeed, consider its image under the embedding $\sigma\colon \mathbb{K} \to \mathbb{C}^n,\ x \mapsto (\sigma_1(x), \ldots, \sigma_n(x))$. Here $\sigma_1, \sigma_2, \ldots, \sigma_n$ are the n embeddings of $\mathbb{K}$ into $\mathbb{C}$, ordered

such that $\sigma_{i+n/2} = \overline{\sigma_i}$ for $i \in [n/2]$ (for $m \geq 3$ cyclotomic fields have no real embeddings). The subset $\{(x_1, \ldots, x_n) \in \mathbb{C}^n : \forall i \in [n/2], x_{i+n/2} = \overline{x}_i\} \subset \mathbb{C}^n$ is isomorphic as an inner product space to $\mathbb{R}^n$ [25, Sec. 2.1]. We implicitly use this isomorphism and write $\sigma : \mathbb{K} \to \mathbb{R}^n$. We also have the coefficient embedding $\mathsf{vec} : \mathbb{K} \to \mathbb{Q}^n$, $a_0 + a_1 X + \cdots + a_{n-1} X^{n-1} \mapsto (a_0, a_1, \ldots, a_{n-1})^\mathsf{T}$, which is an additive group isomorphism.

The algebraic norm and trace are given by $\mathrm{N}(x) = \prod_{i=1}^n \sigma_i(x)$ and $\mathrm{Tr}(x) = \sum_{i=1}^n \sigma_i(x)$ for $x \in \mathbb{K}$. Since the σ_i are ring homomorphisms the algebraic norm is multiplicative and the trace is additive. If $x \in R$ then $\mathrm{N}(x), \mathrm{Tr}(x) \in \mathbb{Z}$. The embeddings enable us to view $\mathbb{K}$ as an inner product space over $\mathbb{Q}$ by defining $\langle \cdot, \cdot \rangle : \mathbb{K} \times \mathbb{K} \to \mathbb{Q}$ as

$$\langle f, g \rangle = \frac{1}{n} \cdot \sum_{i=1}^{n} \overline{\sigma_i(f)} \cdot \sigma_i(g).$$

We renormalise by $\frac{1}{n}$ as there is an isometry, up to a scaling factor of n, from the complex embedding to the coefficient embedding, i.e. we have $\langle f, g \rangle = \langle \mathsf{vec}(f), \mathsf{vec}(g) \rangle$ with the right hand inner product over $\mathbb{R}^n$. This gives a (geometric) norm on $\mathbb{K}$ as $\|\cdot\| : \mathbb{K} \to \mathbb{Q}$, $f \mapsto \sqrt{\langle f, f \rangle}$, which agrees with the Euclidean norm of $\mathsf{vec}(f)$. As $\mathbb{K}$ is a CM field, it has an automorphism $\cdot^* : \mathbb{K} \to \mathbb{K}$ that acts as complex conjugation on its embeddings, which we call the adjoint operator. It is the unique automorphism satisfying $\sigma_i(x^*) = \overline{\sigma_i(x)}$ for all $x \in \mathbb{K}$ and $i \in [n]$. Therefore, we have $\langle f, g \rangle = \mathrm{Tr}(f^* g)/n$.

For any $\ell \in \mathbb{Z}_{\geq 1}$, we define $\mathbb{K}^\ell = \overbrace{\mathbb{K} \oplus \cdots \oplus \mathbb{K}}^{\ell \text{ times}}$ (and similarly R^ℓ, which is an R-module). Extend $\mathsf{vec} : \mathbb{K}^\ell \to \mathbb{Q}^{n\ell}$ in the natural way. We extend the inner product and norm to vectors $\mathbf{f}, \mathbf{g} \in \mathbb{K}^\ell$ by

$$\langle \mathbf{f}, \mathbf{g} \rangle = \sum_{i=1}^{\ell} \langle f_i, g_i \rangle \qquad \text{and} \qquad \|\mathbf{f}\| = \sqrt{\langle \mathbf{f}, \mathbf{f} \rangle}.$$

We write $\mathsf{rot}(f) = \big(\mathsf{vec}(f), \mathsf{vec}(Xf), \ldots, \mathsf{vec}(X^{n-1} f)\big) \in \mathbb{Q}^{n \times n}$ for $f \in \mathbb{K}$, which is a basis for the lattice $\sigma(f)$ given by the (possibly fractional) ideal (f), and extend this to matrices $\mathbf{B} \in \mathbb{K}^{k \times \ell}$ in the natural way,

$$\mathsf{rot}(\mathbf{B}) = \begin{pmatrix} \mathsf{rot}(\mathbf{B}_{11}) & \cdots & \mathsf{rot}(\mathbf{B}_{1\ell}) \\ \vdots & \ddots & \vdots \\ \mathsf{rot}(\mathbf{B}_{k1}) & \cdots & \mathsf{rot}(\mathbf{B}_{k\ell}) \end{pmatrix}.$$

We now define a module lattice. Since R is the ring of integers of a number field, it is a Dedekind domain and the notion of rank is well defined for R-modules.

Definition 4. *Let $M \subset \mathbb{K}^k$ be an R-module of rank $\ell \leq k$ and define the map $\boldsymbol{\sigma} = (\sigma, \ldots, \sigma) : \mathbb{K}^k \to \mathbb{R}^{nk}$, $(x_1, \ldots, x_k) \mapsto (\sigma(x_1), \ldots, \sigma(x_k))$. The image $\boldsymbol{\sigma}(M)$ is a rank $n\ell$ lattice in $\mathbb{R}^{nk}$ which we call a module lattice.*

We may refer to 'the module lattice M' to mean $\boldsymbol{\sigma}(M)$. If $\mathbf{B} \in \mathbb{K}^{k\times\ell}$ is a basis for an R-module M then $\mathsf{rot}(\mathbf{B}) \in \mathbb{Q}^{nk\times n\ell}$ is a basis for the module lattice $\boldsymbol{\sigma}(M)$. For $\mathbf{B} \in \mathbb{K}^{k\times\ell}$ we write $\mathbf{B}^*$ to denote the adjoint transpose, and given a vector $\mathbf{f} \in \mathbb{K}^\ell$ we write $\mathbf{f}^*$ for the adjoint transpose row vector.

Definition 5. *For $\ell \geq 1$, the set of* Hermitian forms $\mathcal{H}_\ell^{>0}(\mathbb{K})$ *consists of all $\mathbf{Q} \in \mathbb{K}^{\ell\times\ell}$ such that $\mathbf{Q}^* = \mathbf{Q}$ and $\mathrm{Tr}(\mathbf{v}^*\mathbf{Q}\mathbf{v}) > 0$ for all $\mathbf{v} \in \mathbb{K}^\ell \setminus \{\mathbf{0}\}$.*

Equivalently, $\mathbf{Q}$ is a Hermitian form whenever $\mathsf{rot}(\mathbf{Q})$ is a quadratic form. For $\mathbf{B} \in \mathbb{K}^{\ell\times\ell}$ the Gram matrix $\mathbf{B}^*\mathbf{B}$ is a Hermitian form. Similar to the general case, we define an inner product with respect to a Hermitian form $\mathbf{Q}$ as

$$\langle \mathbf{f}, \mathbf{g} \rangle_{\mathbf{Q}} = \frac{1}{n} \cdot \mathrm{Tr}(\mathbf{f}^*\mathbf{Q}\mathbf{g}) \qquad \text{and} \qquad \|\mathbf{f}\|_{\mathbf{Q}} = \sqrt{\langle \mathbf{f}, \mathbf{f} \rangle_{\mathbf{Q}}}.$$

Once again, observe that for any $\mathbf{B}$ we have $\langle \mathbf{B}\mathbf{f}, \mathbf{B}\mathbf{g} \rangle = \langle \mathbf{f}, \mathbf{g} \rangle_{\mathbf{B}^*\mathbf{B}}$ and $\|\mathbf{B}\mathbf{f}\| = \|\mathbf{f}\|_{\mathbf{B}^*\mathbf{B}}$. We use the above to define a discrete Gaussian over Hermitian forms. For some $\mathbf{Q} \in \mathcal{H}_\ell^{>0}(\mathbb{K})$ and $\mathbf{x} \in \mathbb{K}^\ell$ set $D_{\mathbf{Q},\sigma}(\mathbf{x}) = D_{\mathsf{rot}(\mathbf{Q}),\sigma}(\mathsf{vec}(\mathbf{x}))$. Due to our choice of $\mathbb{K}$ and the definition of our norm, this is equivalent to the natural definition that follows from $\rho_{\mathbf{Q},\sigma} \colon \mathbb{K}^\ell \to \mathbb{R}$, $\mathbf{x} \mapsto \exp(-\|\mathbf{x}\|_{\mathbf{Q}}^2/2\sigma^2)$. Note that the normalised trace satisfies $\langle 1, z \rangle = \mathrm{Tr}(z)/n$, which evaluates a polynomial $z = z_0 + z_1 X + \cdots + z_{n-1}X^{n-1}$ to its constant coefficient z_0.

Signature scheme. A signature scheme is a triple of PPT algorithms $\Pi = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vf})$ such that Vf is deterministic. On input 1^n, KGen outputs a public and secret key $(\mathsf{pk}, \mathsf{sk})$. We assume n can be determined from either key. On input sk and a message m from a message space that may depend on pk, Sign outputs a signature sig. On input pk, a message m and a signature sig, Vf outputs a bit $b \in \{0, 1\}$. We say sig is a valid signature on m if and only if $b = 1$.

In our practical cryptanalysis of Section 4 we discuss two types of forgery an adversary may produce, strong and weak. A strong forgery is a signature on a message for which an adversary does not know a signature, whereas a weak forgery is a signature on a message for which an adversary may know signatures. We call a signature scheme Π for which an adversary cannot produce a weak forgery strongly unforgeable, and a signature scheme for which an adversary cannot produce a strong forgery weakly unforgeable. In Appendix B of the full version [16] we consider signature security in a formal game based model.

3 Scheme

In this section we present Hawk.[2] We first give a version of Hawk that performs no compression on its signatures for simplicity, we call this uncompressed Hawk. We then introduce (compressed) Hawk and discuss how the security of Hawk directly reduces to that of the uncompressed Hawk.

[2] See `https://github.com/ludopulles/hawk-aux/blob/main/code/hawk.sage`.

3.1 Uncompressed HAWK

The uncompressed version of our signature scheme is based on the scheme presented in [15, Sec. 6], but is adapted to number rings for efficiency. The scheme uses the number ring $R = \mathbb{Z}[X]/(X^n + 1)$ with $n \geq 2$ a power of two, the ring of integers of the number field $\mathbb{Q}(\zeta_{2n})$. We use the simplest rank 2 module lattice, $R^2 \cong \mathbb{Z}^{2n}$. We implicitly move between R^2 and $\mathbb{Z}^{2n}$ via the coefficient embedding. The secret key is some basis $\mathbf{B} \in \mathrm{SL}_2(R)$ where $\mathbf{B}$ (resp. $\mathsf{rot}(\mathbf{B})$) generates R^2 (resp. $\mathbb{Z}^{2n}$). In the context of [15, Sec. 6] this matrix represents a basis transformation applied to the trivial basis $\mathbf{I}_2(\mathbb{K})$ of R^2. The public key is the Hermitian form $\mathbf{Q} = \mathbf{B}^* \cdot \mathbf{B}$ associated to the basis $\mathbf{B}$. A signature for a message m is generated by first hashing m and a salt r to a point $\mathbf{h} = (h_0, h_1)^\mathsf{T} \in \{0,1\}^{2n}$. Applying the transformation $\mathbf{B}$ to $\frac{1}{2}\mathbf{h}$ gives us a target $\frac{1}{2}\mathbf{B} \cdot \mathbf{h}$. We then sample a short element $\mathbf{x}$ in the target's coset $R^2 + \frac{1}{2}\mathbf{B} \cdot \mathbf{h}$ via discrete Gaussian samples on $\mathbb{Z}$ and $\mathbb{Z} + 1/2$. By applying the inverse transformation $\mathbf{B}^{-1}$ we compute the signature $\mathbf{s} = \frac{1}{2}\mathbf{h} \pm \mathbf{B}^{-1}\mathbf{x} \in R^2$. This is close to $\frac{1}{2}\mathbf{h}$ with respect to $\|\cdot\|_\mathbf{Q}$, and the sign is chosen to prevent weak forgeries, see Algorithm 2 and below. See Figure 1 for a visualisation when $n = 1$. Verification checks if the distance $\left\|\frac{1}{2}\mathbf{h} - \mathbf{s}\right\|_\mathbf{Q}$ between $\mathbf{s}$ and $\frac{1}{2}\mathbf{h}$ is not too large, which only requires the public key $\mathbf{Q} = \mathbf{B}^*\mathbf{B}$ and not the secret key $\mathbf{B}$. We have the following parameters:

1. σ_{pk}: controls the length of $(f, g)^\mathsf{T}$, the first basis vector of $\mathbf{B}$,
2. σ_{sec}: controls the lower bound on the acceptable length of $(f, g)^\mathsf{T}$,
3. σ_{sign}: controls the length of of a short coset vector,
4. σ_{ver}: controls the acceptable distance between signatures and halved hashes,
5. $\mathsf{saltlen}$: controls the probability of hash collisions.

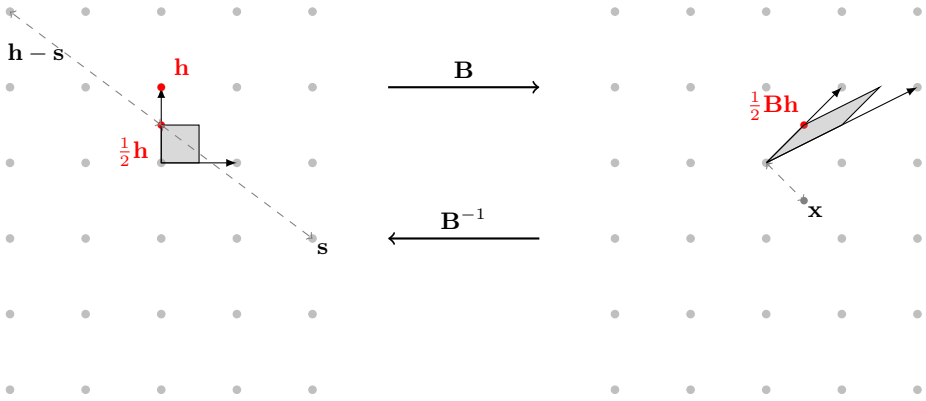


Fig. 1: Illustration of signing. First $\mathbf{h}$ is sampled (left), then $\mathbf{B}$ is applied, a short lattice point $\mathbf{x}$ is sampled from a discrete Gaussian on $\mathbb{Z}^{2n} + \frac{1}{2}\mathbf{B} \cdot \mathbf{h}$ (right). Finally $\mathbf{B}^{-1}$ applied to $\mathbf{x}$ is subtracted from $\frac{1}{2}\mathbf{h}$ to obtain a lattice point $\mathbf{s}$ close to $\mathbf{h}/2$ in $\|\cdot\|_\mathbf{Q}$. We then add $\mathbf{h} - 2\mathbf{s}$ to ensure we satisfy $\texttt{sym-break}(h_1 - 2s_1)$.

Algorithm 1 Key generation for HAWK: $\mathsf{KGen}(1^n)$

Sample $f, g \in R$ with coefficients from $D_{\mathbb{Z},\sigma_{\mathsf{pk}}}$
$q_{00} = f^*f + g^*g$
if $2 \mid \mathrm{N}(f)$ **or** $2 \mid \mathrm{N}(g)$ **or** $\|(f,g)\|^2 \leq \sigma_{\mathsf{sec}}^2 \cdot 2n$ **then**
    restart
$(F,G)^\mathsf{T} \leftarrow \mathsf{TowerSolve}l_{n,1}(f,g)$ [29, Alg. 4], **if** $\perp$, **restart**
$(F,G)^\mathsf{T} \leftarrow (F,G)^\mathsf{T} - \mathsf{ffNP}_R\left(\frac{f^*F+g^*G}{q_{00}}, \mathsf{ffLDL}_R^*(q_{00})\right) \cdot (f,g)^\mathsf{T}$ [14]
$\mathbf{B} = \begin{pmatrix} f & F \\ g & G \end{pmatrix}$.
$\mathbf{Q} = \begin{pmatrix} q_{00} & q_{01} \\ q_{10} & q_{11} \end{pmatrix} = \mathbf{B}^* \cdot \mathbf{B}$.
return $(\mathsf{pk}, \mathsf{sk}) = (\mathbf{Q}, \mathbf{B})$

Algorithm 2 Signing for HAWK: $\mathsf{Sign}_{\mathbf{B}}(\mathsf{m})$

$\mathsf{r} \leftarrow U(\{0,1\}^{\mathsf{saltlen}})$
$\mathbf{h} \leftarrow \mathsf{H}(\mathsf{m} \| \mathsf{r})$
$\mathbf{t} \leftarrow \mathbf{B} \cdot \mathbf{h} \pmod 2$
$\mathbf{x} \leftarrow D_{\mathbb{Z}^{2n}+\frac{1}{2}\mathbf{t},\sigma_{\mathsf{sign}}}$
if $\|\mathbf{x}\|^2 > \sigma_{\mathsf{ver}}^2 \cdot 2n$ **then**
    restart (optional, see Section 5.3, § Failure checks.)
$\mathbf{s} = (s_0, s_1)^\mathsf{T} = \frac{1}{2}\mathbf{h} - \mathbf{B}^{-1}\mathbf{x}$ (parse $\mathbf{x} \in R^2$ via vec^{-1}.)
if `sym-break`$(h_1 - 2s_1)$ is `False` **then**
    $\mathbf{s} \leftarrow \mathbf{h} - \mathbf{s}$
return $\mathsf{sig} = (\mathsf{r}, \mathbf{s})$

Algorithm 3 Verification for HAWK: $\mathsf{Vf}_{\mathbf{Q}}(\mathsf{m}, \mathsf{sig})$

$(\mathsf{r}, \mathbf{s}) \leftarrow \mathsf{sig}$
$\mathbf{h} \leftarrow \mathsf{H}(\mathsf{m} \| \mathsf{r})$
if $\mathbf{s} \in R^2$ **and** `sym-break`$(h_1 - 2s_1)$ is `True` **and** $\left\|\frac{\mathbf{h}}{2} - \mathbf{s}\right\|_{\mathbf{Q}}^2 \leq \sigma_{\mathsf{ver}}^2 \cdot 2n$ **then**
    return 1
else
    return 0

For uncompressed HAWK we present KGen in Algorithm 1, Sign in Algorithm 2 and Vf in Algorithm 3. The security parameter n is a power of two and we assume the internal parameters can be computed from it. We use previous work [29, Alg. 4] to generate the unimodular transformation $\mathbf{B}$ efficiently, by sampling the first basis vector $(f,g)^\mathsf{T}$, and then completing it (if possible) with a second basis vector $(F,G)^\mathsf{T}$ such that $\det \mathbf{B} = 1$. We combine this with the fast Babai reduction of [14] to obtain a shorter second basis vector $(F,G)^\mathsf{T}$. In KGen checks are performed prior to completing the basis $\mathbf{B}$. In $\mathsf{TowerSolve}l$ [29] it is necessary for $\mathrm{N}(f)$ or $\mathrm{N}(g)$ to be an odd integer. We require both to be odd to use an optimised constant-time greatest common divisor algorithm, identical

to the FALCON reference implementation. Also, we require the squared norm of $(f, g)^\mathsf{T}$ to be longer than $\sigma_{\mathsf{sec}}^2 \cdot 2n$ for our concrete cryptanalysis, see Section 4. Note that the signer has $\mathbf{B}^{-1} = \begin{pmatrix} G & -F \\ -g & f \end{pmatrix}$ since $fG - gF = \det \mathbf{B} = 1$.

In Sign and Vf we check the condition $\texttt{sym-break}(h_1 - 2s_1)$, which is required for strong unforgeability. Without it $\mathsf{sig}' = (\mathsf{r}, \mathbf{h} - \mathbf{s})$, which can be constructed from public values, is another valid signature on m if $\mathsf{sig} = (\mathsf{r}, \mathbf{s})$ is. Given $e \in R$, we define $\texttt{sym-break}(e)$ to be True if and only if $e \neq 0$ and the first non zero coefficient of $\mathsf{vec}(e)$ is positive. Checking this condition on $h_1 - 2s_1$ in Vf prevents a weak forgery attack.

Signature correctness. Assume Sign is called with message m and outputs $\mathsf{sig} = (\mathsf{r}, \mathbf{s})$. First, note $\mathbf{B}^{-1}\mathbf{x} \in R^2 + \frac{1}{2}\mathbf{h}$ and $\frac{1}{2}\mathbf{h} \pm \frac{1}{2}\mathbf{h} \in R^2$, so $\mathbf{s} = \frac{1}{2}\mathbf{h} \pm \mathbf{B}^{-1}\mathbf{x} \in R^2$. Second, suppose $\texttt{sym-break}(h_1 - 2s_1)$ is not satisfied during verification. By lines 8 and 9 of Algorithm 2, this means $\texttt{sym-break}(h_1 - 2s_1)$ and $\texttt{sym-break}(2s_1 - h_1)$ are both False, therefore $h_1 = 2s_1$. Since $\mathbf{h} \in \{0, 1\}^{2n}$, this implies $h_1 = 0$, i.e. we have found a preimage of $(h_0, 0)^\mathsf{T}$ for H. By choosing a preimage resistant H or modelling it as a random oracle, this happens with $\mathsf{negl}(n)$ probability. We allow this failure probability to simplify (compressed) HAWK.

The signing algorithm terminates only if the condition on line 5 is False. Therefore $\|\mathbf{x}\|^2 \leq \sigma_{\mathsf{ver}}^2 \cdot 2n$. Thus during verification $\left\|\frac{\mathbf{h}}{2} - \mathbf{s}\right\|_{\mathbf{Q}}^2 = \left\|\mathbf{B}\left(\frac{\mathbf{h}}{2} - \mathbf{s}\right)\right\|^2 = \|\pm\mathbf{x}\|^2 \leq \sigma_{\mathsf{ver}}^2 \cdot 2n$, with $-\mathbf{x}$ given by line 9 of Sign.

Storing pk *and* sk. We now consider how to efficiently store pk and sk, that is,

$$\mathbf{Q} = \begin{pmatrix} q_{00} & q_{01} \\ q_{10} & q_{11} \end{pmatrix} = \mathbf{B}^* \cdot \mathbf{B} \quad \text{and} \quad \mathbf{B} = \begin{pmatrix} f & F \\ g & G \end{pmatrix}$$

respectively. For $\mathbf{B}$ it is sufficient to only store f and g, but this requires the computationally expensive recovery of F and G in Sign. We note that computing F, G is the most expensive part of KGen. Instead, one stores f, g and F since G can be recovered efficiently from $fG - gF = 1$. The coefficients of f, g and F are small so we use a simple encoding with constant-time decoding for them.

For $\mathbf{Q}$ by construction we have $q_{10} = q_{01}^*$ so one may simply drop q_{10}. Moreover, since $\det(\mathbf{B}) = 1$ we have $q_{00}q_{11} - q_{01}q_{10} = \det \mathbf{Q} = \det(\mathbf{B}^*) \det(\mathbf{B}) = 1$, therefore q_{11} can be dropped and reconstructed as $q_{11} = \frac{1 + q_{01}^* \cdot q_{01}}{q_{00}}$. In addition, q_{00} is self-adjoint and therefore only the first half of its coefficients need to be encoded. More details are given in Section 5.2.

3.2 (Compressed) HAWK

HAWK is obtained by dropping s_0 from a signature $\mathbf{s} = (s_0, s_1)^\mathsf{T}$ in Sign and then reconstructing it in Vf using public values $\mathbf{Q}$ and $\mathbf{h}$. There is a probability that s_0 is not correctly recovered, but it is kept small by rejecting 'bad' key pairs in KGen. In Vf, s_0 is recovered by finding a value that makes $\frac{1}{2}\mathbf{h} - (s_0, s_1)^\mathsf{T}$ short

with respect to $\|\cdot\|_{\mathbf{Q}}$. Two ways to reconstruct s_0 are Babai's round-off algorithm and Babai's Nearest Plane algorithm [4]. Given that we work with respect to the norm induced by $\mathbf{Q}$, we must adapt one of these algorithms to quadratic forms. Because of its simplicity and good performance, we use round-off for Hawk. Specifically, we use the following to reconstruct s_0.

$$s_0' = \left\lceil \frac{h_0}{2} + \frac{q_{01}}{q_{00}} \left(\frac{h_1}{2} - s_1 \right) \right\rfloor, \tag{1}$$

where the rounding is coefficientwise and $\lceil x \rfloor = z$ for $x \in \left(z - \frac{1}{2}, z + \frac{1}{2}\right]$ and $z \in \mathbb{Z}$. Hence Vf is adapted to read a signature $\mathsf{sig} = (\mathsf{r}, s_1)$ and reconstruct s_0' using (1), before setting $\mathbf{s} = (s_0', s_1)^{\mathsf{T}}$. Observe that $s_0' = s_0$ if and only if

$$\left\lceil \frac{q_{00}\left(\frac{h_0}{2} - s_0\right) + q_{01}\left(\frac{h_1}{2} - s_1\right)}{q_{00}} \right\rfloor = 0.$$

The fraction inside the rounding function can be rewritten using $(q_{00}, q_{01}) = (f^*, g^*) \cdot \mathbf{B}$ and $\mathbf{B} \cdot \left(\frac{\mathbf{h}}{2} - \mathbf{s}\right) = (x_0, x_1)^{\mathsf{T}}$ as $\frac{f^* x_0 + g^* x_1}{f^* f + g^* g}$. Thus, we certainly recover the correct s_0 from $\mathbf{Q}$, $\mathbf{h}$ and s_1 if

$$\frac{f^* x_0 + g^* x_1}{f^* f + g^* g} \in \left(-\frac{1}{2}, \frac{1}{2}\right)^n. \tag{2}$$

Intuitively, when (f, g) is sampled such that the Euclidean norm of $(f^*/q_{00}, g^*/q_{00})$ is sufficiently small, this recovery works almost always. Note

$$\left\| \left(\frac{f^*}{q_{00}}, \frac{g^*}{q_{00}} \right) \right\|^2 = \frac{1}{n} \mathrm{Tr}\left(\frac{f^* f + g^* g}{q_{00}^2} \right) = \frac{1}{n} \mathrm{Tr}\left(q_{00}^{-1}\right) = \langle 1, q_{00}^{-1} \rangle. \tag{3}$$

Hence we choose a bound ν_{dec} such that decompression works almost always for keys satisfying $\langle 1, q_{00}^{-1} \rangle < \nu_{\mathsf{dec}}$. We provide a computation and value for ν_{dec} in Section 5.1. In summary, Algorithms 1, 2 and 3 are changed as follows for Hawk.

1. In KGen, restart if $\langle 1, q_{00}^{-1} \rangle \geq \nu_{\mathsf{dec}}$.
2. In Sign, restart if $\mathbf{x} = (x_0, x_1)^{\mathsf{T}}$ does not satisfy (2).
3. In Sign, return a signature as (r, s_1) instead of $(\mathsf{r}, \mathbf{s}) = (\mathsf{r}, (s_0, s_1)^{\mathsf{T}})$.
4. In Vf, given a signature (r, s_1) reconstruct s_0' with (1) and set $\mathbf{s} = (s_0', s_1)^{\mathsf{T}}$.

Given the above, a reconstructed signature is correct. In practice we choose ν_{dec} such that (2) is also satisfied except with small probability and forego item 2. in the list, see Section 5.3.

Security relation to the uncompressed variant Note that an adversary that can create a forgery (strong or weak) against Hawk can also create a forgery (strong or weak) against uncompressed Hawk. Indeed, if $\mathsf{sig} = (\mathsf{r}, s_1)$ is a forgery against Hawk then this implies $\mathsf{sig} = (\mathsf{r}, (s_0', s_1)^{\mathsf{T}})$ is a forgery against uncompressed Hawk. Only public quantities are required to recover s_0'. Therefore, throughout we consider the security of uncompressed Hawk. In Appendix B of the full version [16] we further reduce the forgery security of uncompressed Hawk to an assumption called the one more short vector problem, or omSVP.

4 Cryptanalysis

In this section we provide a concrete cryptanalysis of HAWK. Whereas the formal security arguments we make in Section 6 and Appendix B of the full version [16] increase our confidence in the design of HAWK, our results here aid us in choosing parameter sets that are efficient. Throughout we consider uncompressed HAWK. We consider recovering the secret key from public information and forging a new signature given at most $q_s = 2^{64}$ signatures. We report the various parameters, probabilities and blocksizes output by our cryptanalysis in Table 2.

We express the constraints on various quantities in our scheme in terms of Gaussian parameters $\sigma_\bullet$, even if they are not quantities sampled from a distribution. This allows us to present necessary relationships between these quantities as in Figure 2. In particular for $\mathbf{x} \leftarrow D_{\mathbb{Z}^d,\sigma}$, with σ above smoothing, $\mathbb{E}[\|\mathbf{x}\|] \approx \sigma\sqrt{d}$ [26, Sec. 4]. For example, the verification of signatures is determined by a distance, say ℓ. Instead of referring to ℓ, we use the shorthand $\sigma_{\mathsf{ver}} = \ell/\sqrt{d}$.

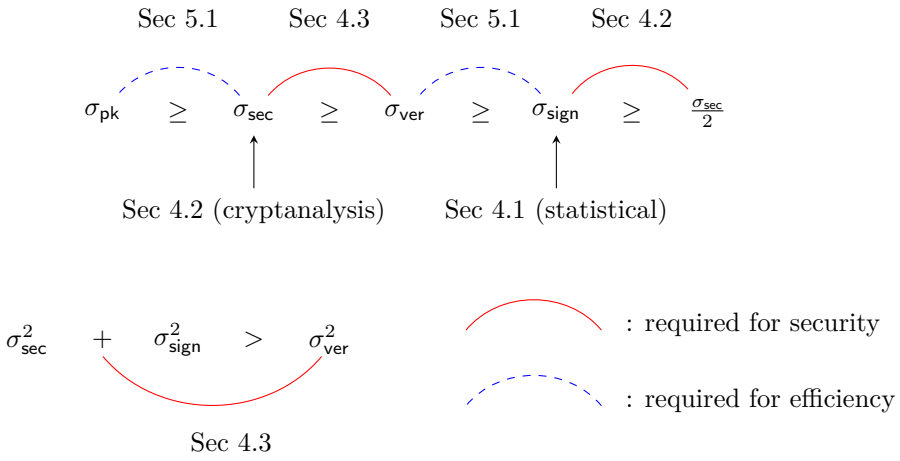


Fig. 2: A summary of the necessary relationships between the various $\sigma_\bullet$.

We stress that the relations of Figure 2 are necessary, under our experimental analysis, conditions for security – any selection must also satisfy the concrete cryptanalysis below. As a short introduction, σ_{sign} is our fundamental parameter, and we select it first. It ensures that our scheme does not suffer from learning attacks [27,13] if an adversary is given access to signature transcripts. We then choose σ_{pk} which controls key generation in Algorithm 1. It must be large enough that recovering the secret key is hard, and also that the cost of computing a sufficiently good basis to aid with signature forgeries is hard. To this end we heuristically estimate and verify experimentally σ_{sec}, a parameter that represents the shortest a public basis can be before one recovers the secret key. Finally, σ_{pk}

and σ_{ver} are chosen to ensure various rejection steps, in key generation and signing respectively, do not occur too frequently, see Section 5.1. The condition $\sigma_{\mathsf{ver}}^2 < \sigma_{\mathsf{sec}}^2 + \sigma_{\mathsf{sign}}^2$ encodes the requirement for a good basis to not help with signature forgeries.

4.1 Choosing σ_{sign}

We choose σ_{sign} large enough to avoid signatures leaking information about a secret key. Following [32, Sec. 2.6], for security parameter λ in the face of an adversary allowed q_s signatures, it is enough to set

$$\sigma_{\mathsf{sign}} \geq \frac{1}{\pi} \cdot \sqrt{\frac{\log(4n(1+1/\varepsilon))}{2}} \geq \eta_\varepsilon(\mathbb{Z}^{2n}),$$

for $\varepsilon = 1/\sqrt{q_s \cdot \lambda}$ to lose a small constant number of bits of security. We note that since we sample from $\mathbb{Z}^{2n}$ we may use the orthogonal basis $\mathbf{I}_{2n}(\mathbb{Z})$, and thus the above inequality is also sufficient for efficient sampling via Lemma 1. We ensure that, following the analysis of Falcon [32, Sec. 3.9.3], our probability distribution tables have a Rényi divergence at order 513 from their ideal distributions of less than $1 + 2^{-79}$.[3]

4.2 Key Recovery

In Hawk, the problem of recovering the secret key $\mathbf{B} \in \mathrm{SL}_2(\mathcal{O}_{\mathbb{K}})$ from the public key $\mathbf{Q} = \mathbf{B}^* \cdot \mathbf{B}$ is a (module) Lattice Isomorphism Problem. For the lattice $R^2 \cong \mathbb{Z}^{2n}$ it is equivalent to finding a $\mathbf{U} \in \mathrm{SL}_2(\mathcal{O}_{\mathbb{K}})$ such that $\mathbf{U}^* \cdot \mathbf{Q} \cdot \mathbf{U} = \mathbf{I}_2(\mathbb{K})$, i.e. reducing (any lattice basis corresponding to) $\mathbf{Q}$ to an orthonormal basis. As mentioned in [15], all known algorithms to solve LIP for modules of rank at least two require finding at least one shortest vector. Therefore we assume that the best key recovery attack requires one to find a single shortest vector.

Unusual-SVP. The shortest vectors in R^2 have length 1, which is a factor of order $\Theta(\sqrt{n})$ shorter than predicted by the Gaussian heuristic. Recovering such 'unusually' short vectors is easier than generic shortest vectors, and can be achieved by running the BKZ lattice reduction algorithm with blocksize β much lower than the full dimension $2n$. Given that for current cryptanalysis there are no significant speed-ups for solving the structured variant of this unusual-SVP, we treat the problem by considering the unstructured version (i.e. as the form $\mathsf{rot}(\mathbf{Q})$ or some rotation of $\mathbb{Z}^{2n}$). The problem of finding an unusually short vector has received much cryptanalytic attention. This has lead to accurate estimates for the required BKZ blocksize, see [3] for a survey. As an estimate, given that our lattice has unit volume and we search for a vector of unit length, we require a blocksize β such that

$$\sqrt{\beta/d} \approx \delta_\beta^{2\beta-d-1}, \tag{4}$$

[3] See `https://github.com/ludopulles/hawk-aux/blob/main/code/generate_C_tables.sage`.

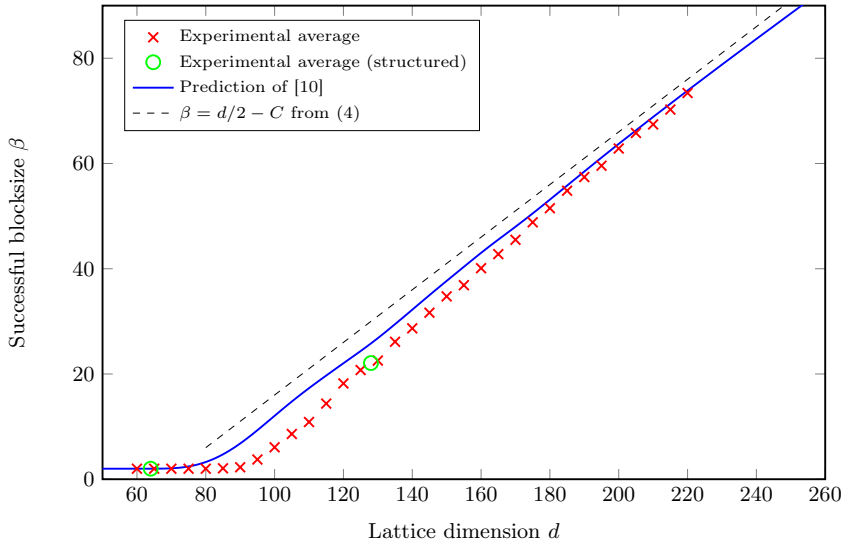
where $\delta_\beta \approx (\beta/2\pi e)^{1/2(\beta-1)}$. Asymptotically this is satisfied for some $\beta \in d/2+o(d)$. Concrete estimates also simulate the Gram–Schmidt profile, use probabilistic models for the lengths of projected vectors and account for the presence of multiple shortest vectors [9,5,10,30].

In Figure 3 we plot the estimate given by (4) where the $o(d)$ term is concretised to some constant, the estimate given by the leaky-LWE-estimator [10], which applies the concrete improvements mentioned above, and experimental data. These experiments apply the BKZ2.0 algorithm with lattice point enumeration as implemented in [34] to the public form $\mathbf{Q}$, reporting the BKZ block size required to find a shortest vector. For dimensions which are not powers of two, the experimental data uses a form sampled by [15, Alg. 1], the unstructured generation procedure upon which our key generation is based. For some small power of two dimensions we generate bases via Algorithm 1. We see that below approximately dimension 80 instances can be solved with LLL reduction, and that afterwards the required blocksize approximately increments by one when the dimension increases by two, as (4) would suggest. We also see that above approximately blocksize 70 the model of [10] appears especially accurate. We therefore use this model to determine β_{key} in Table 2. We use a simple progressive strategy where the blocksize increments by one after each tour, which we expect to require a blocksize perhaps two or three larger than a more optimal progressive strategy.

Decreasing σ_{pk}. For the experiments of Figure 3 we took a large σ_{pk} as an attempt to find a ground truth. We would like to minimise σ_{pk} to minimise the size of our keys and the complexity of computing with them, but without significantly reducing security. To this end we perform a similar experiment where we fix a set of dimensions and reduce forms of these dimensions using various $\sigma_{\mathsf{pk}} < 20$. The results of these experiments are presented in Figure 4. For σ_{pk} below a certain threshold instances can be solved by LLL, then as σ_{pk} increases past this threshold the instances become harder, before reaching an empirical "maximum hardness" (at least with respect to these experiments) where further increases in σ_{pk} appear to give no extra security.

When running BKZ one encounters shorter vectors as β grows. In a random lattice of unit volume one expects to encounter vectors of length $\delta_\beta^{d-1} \in \Theta(d)$ when $\beta = d/2 + o(d)$, but for $\mathbb{Z}^d$ this is also the moment that a vector of length 1 is found. In fact, the model of [10] predicts that we suddenly jump from finding vectors of length $\ell_0 = \Theta(d)$ to finding a shortest vector of length 1. This threshold behaviour was observed and discussed in [7, Sec. 6.2]. In our notation the authors observe the threshold effect once vectors of length approximately $\sqrt{d}/2$ are discovered. Our model and experiments suggest the threshold length is $\Theta(d)$ but with a constant smaller than 1. We verified this behaviour for $\mathbb{Z}^d$ experimentally. In Figure 5 we plot $\sigma_{\mathsf{sec}} = \ell_0/\sqrt{d}$ where ℓ_0 is the length of the shortest basis vector after the penultimate tour concludes.

We see that for large enough dimensions the behaviour matches the unusual-SVP predictions, and we obtain $\sigma_{\mathsf{sec}} = \Theta(\sqrt{d})$. For Table 2 we take σ_{sec} as the output from the prediction of [10]. We assume the value σ_{sec} represents a lower



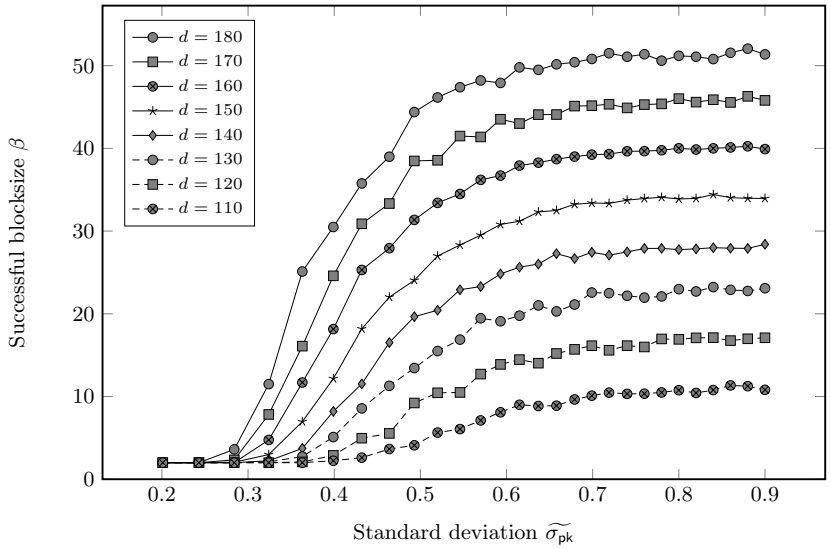
We ran progressive BKZ (one tour per blocksize) over $\mathbb{Z}^d$ using an input form generated with $\sigma_{\mathsf{pk}} = 20$ and report the average successful β that recovered a length one vector over 40 instances. We used the BKZ simulator and probabilistic model of [10], accounting for the d target solutions. See `BKZ_simulator.sage` and `exp_varying_n.sage` at `https://github.com/ludopulles/hawk-aux`.

Fig. 3: Blocksize required to recover a shortest vector via lattice reduction as a function of dimension d.

bound for σ_{pk} such that our public forms exhibit maximum hardness. In practice we take $\sigma_{\mathsf{pk}} > \sigma_{\mathsf{sec}}$ and reject keys where the length of $(f, g)^{\mathsf{T}}$ is shorter than ℓ_0. If we allow shorter $(f, g)^{\mathsf{T}}$ then the public key may give information to an adversary that she would not have unless she had already recovered the secret key.

We note that the prediction of [10] in Figure 5 is inaccurate for $d \leq 180$, similarly (but more noticeably) to Figure 3. One can improve the accuracy of estimates for these dimensions by using the geometric series assumption, and performing several tours so that basis profiles match it, for small blocksizes (say up to $\beta = 20$). Since our estimates converge in the range of feasible experiments, we choose simplicity instead.

Note that even if the statistical arguments of Section 4.1 allow it, we cannot take $\sigma_{\mathsf{sign}} < \sigma_{\mathsf{sec}}/2$. Indeed, $2 \cdot (\frac{1}{2}\mathbf{h} - \mathbf{s}) \in \mathbb{Z}^{2n}$ and if $\sigma_{\mathsf{sign}} < \sigma_{\mathsf{sec}}/2$ then $\left\|2 \cdot (\frac{1}{2}\mathbf{h} - \mathbf{s})\right\|_{\mathbf{Q}} = 2\|\mathbf{x}\| \approx 2\sigma_{\mathsf{sign}}\sqrt{d} < \sigma_{\mathsf{sec}}\sqrt{d}$. Therefore, doubling a public quantity given by a signature may describe a shorter lattice vector than those seen just before secret key recovery.



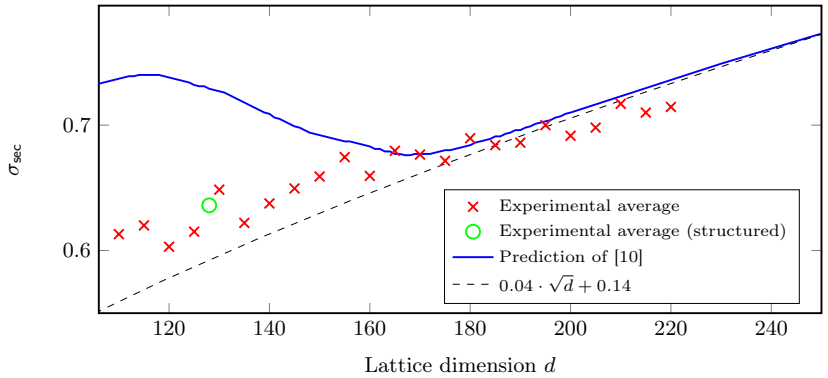
We ran progressive BKZ (one tour per blocksize) over $\mathbb{Z}^d$ using an input form generated with various σ_{pk} and report the average successful β that recovered a length one vector over 80 instances. Note that the range of σ_{pk} includes values below smoothing, for which the actual standard deviation $\widetilde{\sigma_{\mathsf{pk}}}$ can be significantly lower than the Gaussian parameter σ_{pk}. See `https://github.com/ludopulles/hawk-aux/blob/main/code/exp_varying_sigma.sage`.

Fig. 4: Blocksize required to recover a shortest vector via lattice reduction as a function of the standard deviation $\widetilde{\sigma_{\mathsf{pk}}}$.

4.3 Signature Forgery

Strong Forgery. We consider the general problem of forging a signature for some unsigned message. Specifically, given a target $\frac{1}{2}\mathbf{h}$ for some $\mathbf{h} \in \{0,1\}^{2n}$, return an $\mathbf{s} \in \mathbb{Z}^{2n}$ such that $\left\|\frac{1}{2}\mathbf{h} - \mathbf{s}\right\|_{\mathbf{Q}} \leq \sigma_{\mathsf{ver}}\sqrt{d}$. We use the heuristic that solving such an approximate CVP instance is at least as hard as solving an approximate SVP instance with the same approximation factor over the same lattice. we determine the expected blocksize β using the BKZ simulator [10] such that our first basis vector has norm less than $\sigma_{\mathsf{ver}}\sqrt{d}$, and report it as β_{forge} in Table 2. Note that since we use the BKZ simulator of [10] to estimate both β_{key} and β_{forge}, if $\sigma_{\mathsf{ver}} = \sigma_{\mathsf{sec}}$ then $\beta_{\mathrm{key}} = \beta_{\mathrm{forge}}$. Our approach mandates that $\sigma_{\mathsf{ver}} \leq \sigma_{\mathsf{sec}}$. We make this design decision because in our model it means an adversary should not be able to produce a strong forgery unless the secret key is recovered. Indeed, when $\sigma_{\mathsf{ver}} \leq \sigma_{\mathsf{sec}}$ our assumption on approximate CVP says forging a signature is as hard as finding a vector as short as those found just before key recovery.

Weak Forgery. We consider a weak forgery attack consisting of adding a short lattice vector to an existing signature for some message, and hoping that it



We ran progressive BKZ (one tour per blocksize) over $\mathbb{Z}^d$ using an input form generated with $\sigma_{\mathsf{pk}} = 20$ and report the average σ_{sec} determined by the shortest basis vector in the penultimate BKZ tour over 40 instances. See `exp_varying_n.sage` and `predict_varying_n.sage` at `https://github.com/ludopulles/hawk-aux`.

Fig. 5: Shortest basis vector length before the recovery of a length one vector, given in terms of σ_{sec}.

remains a valid signature for the same message. This vector might come from the public key, or from lattice reduction effort on it. Its length is assumed to be at least $\ell_0 = \sigma_{\mathsf{sec}} \cdot \sqrt{d}$, see Section 4.2. We give arbitrarily many such length ℓ_0 vectors to the adversary for free. We estimate the probability the attack succeeds, i.e. that $\|\mathbf{x} + \mathbf{v}\|^2 \leq d \cdot \sigma_{\mathsf{ver}}^2$ for $\mathbf{x} \leftarrow D_{\mathbb{Z}^{2n}, \sigma_{\mathsf{sign}}}$ and $\mathbf{v}$ of length ℓ_0.[4] If $\mathbf{x}$ were sampled from a spherical continuous Gaussian, then considering any such $\mathbf{v}$ would give the same distribution of squared lengths. We examine the distribution of $\|\mathbf{x} + \mathbf{v}\|^2$ for two "extremal" choices of $\mathbf{v}$; the first has all its weight in one coordinate, $\mathbf{v} = (\lfloor \ell_0 \rfloor, 0, \ldots, 0)$, and the second is as balanced as possible, e.g. $\mathbf{v} = (1, \ldots, 1, 2, \ldots, 2)$ for $\|\mathbf{v}\| = \lfloor \ell_0 \rfloor$. Note that the distribution of $\|\mathbf{x} + \mathbf{v}\|^2$ is invariant under signed permutations of $\mathbf{v}$. We report our estimate for the success probability of this attack as Pr[weak forgery] in Table 2.

This attack implies the requirement $\sigma_{\mathsf{ver}}^2 < \sigma_{\mathsf{sec}}^2 + \sigma_{\mathsf{sign}}^2$. Even if a vector from a reduced public key is orthogonal to a given signature, then if $\sigma_{\mathsf{ver}}^2 \geq \sigma_{\mathsf{sec}}^2 + \sigma_{\mathsf{sign}}^2$ adding it will likely be sufficient for a weak forgery.

Comparison with Falcon. Falcon uses a different cryptanalytic model to determine the blocksizes reported in Table 2. Our model makes use of recent improvements [10] and enjoys the experimental evidence above. The unusually short vectors in $\mathbb{Z}^d$ are a factor about 1.17 shorter than the NTRU lattice of Falcon, after appropriate renormalisation, and thus key recovery for Hawk will be slightly easier than Falcon in either model. On the other hand, our verification bound σ_{ver} is a factor about 1.15 (Hawk-512) to 1.06 (Hawk-1024)

[4] See `fail_and_forge_probability` at `https://github.com/ludopulles/hawk-aux/blob/main/code/find_params.sage`.

shorter than FALCON after renormalisation, and thus obtaining strong forgeries is slightly harder than in FALCON in either model. In both HAWK-512 and FALCON-512 key recovery is harder than signature forgery, and thus hardening the latter, as we do, gives a slightly more secure scheme overall. For HAWK-1024 and FALCON-1024 key recovery is easier than signature forgery, and in the FALCON model we obtain a slightly less secure scheme overall. See Appendix D of the full version [16] for more detail on FALCON's security methodology, and a comparison to HAWK under it. We also argue there that part of the key recovery methodology of FALCON is overconservative.

5 Parameters and Performance

In Table 2 we list parameters and the output of our concrete cryptanalysis for HAWK.[5] Section 5.1 explains how these parameters were chosen. We explain the encoding used for public keys and signatures, and the simple encoding used for secret keys, in Section 5.2. In Section 5.3 we explain the design choices made in our constant-time implementation of HAWK, written in C. Finally, Section 5.4 contains the details behind Table 1. More details can be found in Appendix C of the full version [16].

5.1 Parameter Selection

In HAWK we set $\mathsf{saltlen} = \lambda + \log_2 q_s$, where $q_s = 2^{64}$ is the limit on the signature transcript size. The probability of a hash collision is then less than $q_s \cdot 2^{-\lambda}$ [32, Section 2.2.2]. Allowing $\mathsf{saltlen}$ to depend on λ implies one must know λ before computing $\mathsf{H}(\mathsf{m}\|\mathsf{r})$, which is commonly the case. For simplicity FALCON choose a fixed salt length of 320 bits. This is not optimal for $\lambda = 128$. Here HAWK-512 saves 16 bytes on signatures, though this saving is also available to FALCON-512.

The value of σ_{pk} for HAWK-512 listed in Table 2 is such that the probability of $\|(f,g)\|^2 > \sigma_{\mathsf{sec}}^2 \cdot 2n$ is greater than 99.5% for f, g both with odd algebraic norm and sampled as in KGen. For HAWK-1024, the probability that $\|(f,g)\|^2 > \sigma_{\mathsf{sec}}^2 \cdot 2n$ holds for similar f, g is greater than 80%.

There are two failures that may occur during signing in HAWK. Firstly, $\|\mathbf{x}\|^2$ may be too large. Secondly, (2) may be violated, i.e. decompressing $\mathsf{sig} = (\mathsf{r}, s_1)$ may return $s_0' \neq s_0$, the original first component of the signature. We choose parameters σ_{ver} and ν_{dec} to make such failures unlikely.

For HAWK-512, $\mathbf{x}$ is too large with a probability of around 2^{-22}, determined by convolving the necessary distributions together.[6] To obtain a strict upper bound on this probability one can use a looser tail bound analysis via Lemma 2 and Lemma 3, with $\varepsilon = 1/\sqrt{q_s \lambda}$ and $\tau = \sigma_{\mathsf{ver}}/\sigma_{\mathsf{sign}}$, which gives 2^{-17}. Similarly

[5] See `https://github.com/ludopulles/hawk-aux/blob/main/code/find_params.sage`.

[6] See `fail_and_forge_probabilities` at `https://github.com/ludopulles/hawk-aux/blob/main/code/find_params.sage`.

for Hawk-1024, $\mathbf{x}$ is too large with a probability of around 2^{-128} and the tail bound gives a probability of at most 2^{-121}.

Given a fixed secret key $\mathsf{sk} = \mathbf{B}$, we provide a heuristic upper bound on the probability of decompression (1) giving $s_0' \neq s_0$, which is upper bounded by the probability that (2) does not hold. This also upper bounds the probability a compressed signature is correct although $s_0 \neq s_0'$. Heuristically, we assume that x_0, x_1 are independently sampled from a normal distribution on $\mathbb{R}^n$ with mean 0 and standard deviation σ_{sign}. Following Section 3.2 the decompression succeeds if $\frac{f^* x_0 + g^* x_1}{q_{00}} \in (-\frac{1}{2}, \frac{1}{2})^n$. Since $\mathbf{B}$ is fixed, each coefficient is normally distributed with mean 0 and variance $\|(f^*/q_{00}, g^*/q_{00})\|^2 \cdot \sigma_{\mathsf{sign}}^2 = \langle 1, q_{00}^{-1} \rangle \cdot \sigma_{\mathsf{sign}}^2$, using (3). Hence the probability that one of the n coefficients is not in the interval $(-\frac{1}{2}, \frac{1}{2})$ is $\mathrm{erfc}\left(\frac{1}{2} / \left(\sqrt{2 \cdot \langle 1, 1/q_{00} \rangle} \cdot \sigma_{\mathsf{sign}}\right)\right)$, where erfc is the complementary error function. By a union bound, the probability that decompression fails is heuristically bounded from above by $n \cdot \mathrm{erfc}\left(1/(\sqrt{8 \cdot \langle 1, 1/q_{00} \rangle} \cdot \sigma_{\mathsf{sign}})\right)$. By rejecting keys for which $\langle 1, q_{00}^{-1} \rangle \geq \nu_{\mathsf{dec}}$, decompression fails for any $\mathbf{B}$ heuristically with probability at most $n \cdot \mathrm{erfc}\big(1/\left(\sqrt{8\nu_{\mathsf{dec}}} \cdot \sigma_{\mathsf{sign}}\right)\big)$.

Taking $\nu_{\mathsf{dec}} = 1/1000$ in Hawk-512 this upper bound is 2^{-105}. This condition on (f, g) in KGen fails in about 9% of cases. We empirically determined this by sampling f and g with odd algebraic norm 100,000 times. Combining this with the small probability that $\|(f, g)\|$ is too small, one can efficiently sample (f, g) until all requirements, before $\mathsf{TowerSolve}$ is invoked, are met.

In Hawk-1024 we take $\nu_{\mathsf{dec}} = 1/3000$ and decompression fails on a signature with probability less than 2^{-315} for a key satisfying $\langle 1, q_{00}^{-1} \rangle < \nu_{\mathsf{dec}}$. This condition fails in about 0.9% of the cases during sampling of (f, g) inside KGen.

Parametrisations for formal reductions. The parameters above are determined by concrete cryptanalysis and do not follow our formal reductions. In Section 6 we give a worst case to average case reduction for smLIP, and in Appendix A of the full version [16] show how it applies to the KGen of Hawk. For this reduction to be efficient σ_{pk} must grow exponentially in n. We discuss this in final paragraph of Section 6. In Appendix B of the full version [16] we reduce the strong signature forgery security of Hawk's design to omSVP. To ensure the parametrisation of the omSVP problem we reduce to is not easy we require $\sigma_{\mathsf{ver}} < \sqrt{2}\sigma_{\mathsf{sign}}$ and $2\sigma_{\mathsf{sign}} < \sigma_{\mathsf{pk}}$. The existence of plausibly hard parametrisations of omSVP encourages us that there is no inherent flaw in our design. We take σ_{pk} smaller than this requirement and discuss this further in Appendix B of the full version [16].

5.2 Encoding

In Hawk both the secret key and $\mathbf{x}$ in Sign are sampled from a discrete Gaussian. As a consequence, the coefficients of the public key and the signature roughly follow a normal distribution. Therefore, it is beneficial to use the Golomb–Rice coding [21]. This encoding is used for the signatures in Falcon [32].

	HAWK-256	HAWK-512	HAWK-1024
Targeted security	Challenge	NIST-1	NIST-5
Dimension $d = 2n$	512	1024	2048
Bit security λ	64	128	256
Transcript size limit q_s	2^{32}	2^{64}	2^{64}
Signature Std. dev. σ_{sign}	1.010	1.278	1.299
Verif. Std. dev. σ_{ver}	1.040	1.425	1.572
Key Recov. Std. dev. σ_{sec}	1.042	1.425	1.974
Key Gen. Std. dev. σ_{pk}	1.1	1.5	2
Salt Length (bits)	112	192	320
$\log_2(\Pr[\text{sign fail}])$	-2	-22	-128
Key Recov. (BKZ) β_{key}	211	452	940
Strong Forgery (BKZ) β_{forge}	211	452	1009
$\log_2(\Pr[\text{weak forgery}])$	-83	-143	-683

Table 2: Parameter sets and their estimated security are given. The dimension, bit security and transcript size are used to determine σ_{sign}. Other standard deviations are determined in Section 4. We then estimate the probability that a signature fails for being too long. The estimated required blocksizes β for BKZ reduction to achieve key recovery and signature forgery are then given. Finally, we give the estimated probability of finding a weak forgery via the attack in Section 4.3.

For the coding, we use an altered absolute value $|\cdot|' \colon \mathbb{Z} \to [0, \infty)$, $x \mapsto x$ for $x \geq 0$ and $x \mapsto -x - 1$ for $x < 0$. The map that sends x to its sign and $|x|'$ gives a bijection $\mathbb{Z} \to \{0, 1\} \times \mathbb{Z}_{\geq 0}$. Given a quantity that is sampled from a discrete Gaussian distribution with (an above smoothing) parameter σ, take an integer k close to $\log_2(\sigma)$. To encode a value $x \in \mathbb{Z}$, first output the sign of x and the lowest k bits of $|x|'$ in binary. Then output $\lfloor |x|'/2^k \rfloor$ in unary, i.e. $\lfloor |x|'/2^k \rfloor$ zeros followed by a one. Note that FALCON uses $|\cdot|$ where we use $|\cdot|'$, but their decoding fails when negative zero is encountered to ensure unique encodings [32, Section 3.11.2]. An advantage of this altered absolute value is that it sometimes saves one bit and is easy to implement: $|\mathtt{x}|'$ is the XOR of `x` and `-(x >> 15)` when `x` has 16 bits.

We use the Golomb–Rice coding on pk and s_1. In particular, for pk the coefficients of q_{01} and coefficients 1 up to and including $n/2 - 1$ of q_{00} are encoded, with $k = 9$ and $k = 5$ respectively for HAWK-512 and $k = 10$ and $k = 6$ for HAWK-1024. The constant coefficient of q_{00} is output with 16 bits as its size is much larger than the other coefficients. The second half of q_{00} can be deduced from its self-adjointedness. For s_1 we use $k = 8$ and $k = 9$ in our implementation for HAWK-512 and HAWK-1024 respectively.

For the secret key, we note the sampler in our implementation of HAWK-512 generates coefficients for f and g with an absolute value at most $13 < 2^4$, we encode these with 5 bits, one of which is the sign. For the remaining polynomial

F of the secret key, we encode with one byte per coefficient (recall G can be reconstructed). We use this simple encoding and decoding for our secret key as it is constant-time. HAWK-1024 requires 6 bits for coefficients of f and g since the sampler generates values of absolute value at most $18 < 2^5$.

5.3 Implementation Details

We implemented HAWK-512 and HAWK-1024 in the C programming language, together with an AVX2 optimised implementation. Due to the many algorithmic similarities between HAWK and FALCON, we were able to reuse a significant portion of the public implementation of FALCON. The code for key generation and signing is isochronous; all is constant-time except the encoding of the public quantities pk and sig. Verification is trivially isochronous as it only uses public information.

Babai reduction. In KGen we perform another reduction step to make $(F, G)^\mathsf{T}$ smaller than the output of $\mathsf{TowerSolve}$. $\mathsf{TowerSolve}$ returns an element $(F, G)^\mathsf{T}$ whose projection onto the module lattice $M = (f, g)^\mathsf{T} \cdot R$, i.e. the rank n real lattice with basis $\mathbf{C} = \mathsf{rot}((f, g)^\mathsf{T})$, lies in the fundamental parallelepiped defined by $\mathbf{C}$. If this projection is uniformly distributed here, the expectation of its squared norm is $\frac{n}{12} \cdot 2n\sigma_{\mathsf{pk}}^2$. However, line 6 in Alg. 1 implies the projection of $(F, G)^\mathsf{T}$ onto M lies in the fundamental domain generated by the Gram–Schmidt orthogonalisation of the rotations of $(f, g)^\mathsf{T}$ in bit reversed order. By [31, Lemma 6.9], the i^{th} vector has an expected norm of $\sqrt{\frac{2n+1-i}{2n}} \cdot \|(f, g)\|$ for $i \in [n]$. Therefore, the expected squared norm of a point sampled uniformly from this fundamental domain will be

$$\frac{1}{12} \cdot \frac{(3n+1)n/2}{2n} \cdot 2n\sigma_{\mathsf{pk}}^2 = \frac{3n+1}{48} \cdot 2n\sigma_{\mathsf{pk}}^2 \leq \frac{n}{12} \cdot 2n\sigma_{\mathsf{pk}}^2.$$

We observe that the squared norm of (F, G) is reduced by a factor of $4/3$ by line 6 of Algorithm 1. This shrinks the HAWK-512 public key size from 1027 bytes on average to 1006.

Sampling and pseudorandomness. We use the SHAKE256 extendable output function to seed a pseudorandom number generator (PRNG) based on CHACHA20 during sampling in KGen and Sign. As the Gaussian parameters used in the scheme are fixed, DGS can be performed efficiently with precomputed probability tables and this PRNG. KGen uses a sampler that requires 64 bits of randomness. For sampling in Sign we implement DGS for $\mathbb{Z} + \frac{1}{2}t$ with $t \in \{0, 1\}$ that is constant-time over input t and that uses 80 bits of randomness, sufficient for Section 4.1. This sampler uses a reverse cumulative distribution table scaled by a factor of 2^{78} similar to [32, Section 3.9.3]. Almost half of the time of Sign is spent on sampling.

Fast polynomial arithmetic. We want all computations to be performed in time $O(n \log n)$. Addition of two polynomials in $R = \mathbb{Z}[X]/(X^n + 1)$ is in $O(n)$, but naïve multiplication in R requires $O(n^2)$ integer multiplications. There are two ways to achieve $O(n \log n)$ via the specific structure of the used number ring. First, one can use the fast Fourier transform (FFT) to perform a multiplication in $O(n \log n)$. Alternatively, one can perform multiplications with the number theoretic transform NTT, which works with a (sufficiently large) prime modulus $p \equiv 1 \pmod{2n}$ and an element $\omega \in \mathbb{F}_p^\times$ of order $2n$. Then, as $\mathbb{F}_p^\times$ is a cyclic group of order $p - 1$ the NTT computes $f(\omega^i) \in R$ for all $i \in (\mathbb{Z}/2n\mathbb{Z})^\times$ in time $O(n \log n)$. When polynomials are transformed with the FFT or NTT, multiplication is coefficientwise. We only use NTT in the reference implementation. Multiplications in R could also be implemented with Karatsuba or Toom–Cook style multiplication. For certain applications such as masking this may have comparable performance for the given parameters. We leave the trade-off between masking techniques and alternative multiplication methods as future work.

For the NTT, multiplying two polynomials $a(X), b(X)$ requires p to be twice larger than the absolute value of any coefficient of $a(X)$, $b(X)$ or $a(X) \cdot b(X)$. This allows one to recover the correct result in $\mathbb{Z}$ via the inverse transformation. In HAWK-512 and HAWK-1024, $p = 12289$ is sufficient for signing. In the reference implementation we use $p = 2^{16} + 1$ since this Fermat prime allows a faster multiplication procedure than using Montgomery reduction with $p = 12289$. Signing using $p = 2^{16} + 1$ is 17% faster than using $p = 12289$. If one wants to reduce memory usage from 15kB to 8kB for HAWK-512, one can safely use the prime $p = 18433$ such that values fit in the 16 bits instead of 32.

By demanding coefficients of $\mathbf{s}$ and $\mathbf{Q}$ are within 6 standard deviations of their means, we can bound the integer $\|\mathbf{h} - 2\mathbf{s}\|_{\mathbf{Q}}$ by a product of two 31 bit primes. Hence, in the reference implementation of Vf, we can compute the norm of a signature by computing it with the NTT modulo these two primes.

The FFT in our implementation uses double precision floating-point numbers (`double`). When a processor has a floating-point unit and AVX2 support, this FFT is much faster than the NTT, but also requires more RAM.

Divisions and signature decompression. During decoding we require a polynomial division in $\mathbb{K}$ to recover $G = (1 + gF)/f$ and $q_{11} = (1 + q_{10}q_{01})/q_{00}$. Since these exact divisions have output in R, they can be computed with either the FFT or NTT, by performing a division coefficientwise in the transformed domain. In KGen, it should be checked that all NTT coefficients of f and q_{00} are nonzero.

The signature decompression requires a division with rounding to the closest integral point in R, which can be done efficiently with the FFT. One can do this with fixed-point arithmetic: it is highly unlikely that the numerical error yields an incorrect rounding. Especially, when we require that the quantity in (2) has to be in $(-0.49, 0.49)^n$, an absolute error of 0.01 is tolerated.

Failure checks. By default we catch invalid signatures before they are issued. The first failure check is line 5 of Algorithm 2. A decompression may also output an incorrect $s_0' \neq s_0$. To catch this we could check with FFT if (2) holds, and restart

	Falcon-512	Hawk-512	$\left(\frac{\text{Falcon}}{\text{Hawk}}\right)$	Falcon-1024	Hawk-1024	$\left(\frac{\text{Falcon}}{\text{Hawk}}\right)$
AVX2 Sign	320 µs	58 µs	×5.5	656 µs	114 µs	×5.8
Ref. Sign	5427 µs	168 µs	×32	11868 µs	343 µs	×35

Table 3: Performance of dynamic signing for Falcon and Hawk on same PC with same compilation flags as in Table 1.

if not. We remove this decompression check, because it is extremely unlikely that it restarts over the lifetime of a key (see Section 5.1).

Omitting the first check might be necessary when implementing Hawk as a circuit inside an FHE scheme, where `while` loops are impractical. This comes at the cost of a rare but non negligible probability (see Section 5.1) of an invalid signature, which might be mitigated by reparametrising the scheme.

In contrast to Falcon, we sample a new salt whenever Sign restarts as we do not see how reusing the same target can be made compatible with the security argument of [19]. Reusing the salt may lead to a statistical leak for Falcon, though it may be hard to exploit as failures in signing are rare. Nevertheless, we choose to be cautious when it comes to statistical leaks.

5.4 Performance

We report on the performance of our implementation of Hawk-512 and compare it to that of Falcon-512 in Table 1. Hawk was compiled with the `gcc` compiler (version 12.1.0) and compilation flag `-O2` (and `-mavx2` for AVX2), as `-O3` actually made the performance worse. The code for Falcon was taken from the 'Extra' folder in the Round 3 submission package `https://falcon-sign.info/falcon-round3.zip`, and was compiled with the same `gcc` but had compilation flags `-O3 -march=native`.

Memory usage. The reference implementation of Hawk-512 uses 24kB, 15kB and 18kB of RAM for KGen, Sign and Vf respectively, versus 16kB, 40kB and 4kB for Falcon-512 respectively. Hawk requires more RAM for KGen compared to Falcon to execute line 6 of Algorithm 1. RAM usage of Falcon's KeyGen is more than reported on `https://falcon-sign.info` as we took the RAM usage of the API functions, which takes sizes of decoded keys into account. The AVX2 optimised implementation of Hawk requires 27kB and 24kB for Sign and Vf respectively, prioritising speed over memory usage. For Hawk-1024 and Falcon-1024 memory usage roughly doubles.

Batched vs. dynamic signing. For consistency with the Falcon report [32], Table 1 reports signing speeds for batched usage, that is, after some precomputation expanding the secret key (called `expand_seckey`). If one needs to start from the secret key without expanding it, the performance of Falcon is significantly affected while the precomputation is much lighter for Hawk. The timings for dynamic signing are given in Table 3.

6 Module LIP Self Reduction

In this section we give a worst to average case reduction for the search module LIP problem, which underlies secret key recovery in HAWK. The average case distribution we give does not match Algorithm 1 exactly, as it does not include some conditions which may cause a restart. We also replace TowerSolveI, which 'completes' f and g into a basis with determinant one via F and G, with HermiteSolve. HermiteSolve fails if and only if it is impossible to complete a particular f, g. We show that the distribution of public keys output by Algorithm 1 after the above changes enjoys a worst to average case reduction. In Appendix A of the full version [16] we discuss the unaltered public key distribution of HAWK and show that the reduction is still applicable to HAWK.

Throughout we are concerned with asymptotic security, in contrast to the main body of the paper where we find efficient parameters that are supported by concrete cryptanalysis. In particular, the choice of σ_{pk} required to make the reduction efficient is larger than is chosen in our parametrisations for HAWK.

6.1 Module Lattice Isomorphism Problem

Here we introduce a generalisation of the Lattice Isomorphism Problem (LIP) to module lattices. Given the Hermitian inner product, the correct generalisation of orthonormal transformations for the module version is to that of unitary matrices. To avoid confusion with $\mathbf{U}$, which is often used for $\mathbf{U} \in \mathrm{GL}_\ell(\mathcal{O}_\mathbb{K})$ in lattice based cryptography, we will use $\mathbf{O} \in \mathrm{U}_\ell(\mathbb{K}_\mathbb{R}) = \{\mathbf{O} \in \mathbb{K}_\mathbb{R}^{\ell\times\ell} \colon \mathbf{O}^* \cdot \mathbf{O} = \mathbf{I}_\ell(\mathbb{K})\}$ for unitary matrices.

Definition 6 (Module Lattice Isomorphism Problem). *Given two $\mathcal{O}_\mathbb{K}$-modules $M, M' \subset \mathbb{K}^\ell$ find $\mathbf{O} \in \mathrm{U}_\ell(\mathbb{K}_\mathbb{R})$ such that $\mathbf{O}\cdot M = \{\mathbf{O}\cdot m \colon m \in M\} = M'$.*

Moving to Hermitian forms, the natural translation becomes equivalence under the action of $\mathrm{GL}_\ell(\mathcal{O}_\mathbb{K})$. However, for simplicity we restrict ourselves to equivalence under the action of $\mathrm{SL}_\ell(\mathcal{O}_\mathbb{K})$, and we denote the equivalence class by $[\mathbf{Q}]_{\mathrm{sl}} = \{\mathbf{U}^* \cdot \mathbf{Q} \cdot \mathbf{U} \colon \mathbf{U} \in \mathrm{SL}_2(\mathcal{O}_\mathbb{K})\}$. Throughout we implicitly restrict to $\mathbb{K}$ that are CM fields, e.g. all cyclotomic fields. Using (generalisations of) the Gentry–Szydlo algorithm [20,24,23], solving LIP under both actions is equivalent for such fields. We can now define the worst case module LIP variant. Note that our worst case and average case problems are *within* a particular class.

Definition 7 (Worst case smLIP). *Given $\mathbb{K}$ and $\mathbf{Q} \in \mathcal{H}_\ell^{>0}(\mathbb{K})$ an instance of $\mathsf{wc{-}smLIP}^{\mathbf{Q}}_{\mathbb{K},\ell}$, the worst case search module Lattice Isomorphism Problem, is given by $\mathbf{Q}$ and any $\mathbf{Q}' \in [\mathbf{Q}]_{\mathrm{sl}}$. A solution is $\mathbf{U} \in \mathrm{SL}_\ell(\mathcal{O}_\mathbb{K})$ such that $\mathbf{Q}' = \mathbf{U}^*\mathbf{Q}\mathbf{U}$.*

We now define an average case version of smLIP relevant to HAWK. It is less general than the worst case version in that we implicitly fix $\ell = 2$ and consider only power of two cyclotomics with conductor $m = 2^\kappa = 2n$ for $\mathbb{K}$. We define our average case distribution for any class $[\mathbf{Q}]_{\mathrm{sl}}$, but note that the average

case distribution over $[\mathbf{I}_2(\mathbb{K})]_{\mathrm{sl}}$ relates to our key generation in Algorithm 1. Finally, we define our average case distribution $\mathsf{AC}_\sigma([\mathbf{Q}]_{\mathrm{sl}}, \mathbb{K})$ algorithmically, see Algorithm 6. This algorithm takes as input a particular form $\mathbf{Q}$ and a parameter σ that controls an internal discrete Gaussian sampling procedure, and outputs a sample from $[\mathbf{Q}]_{\mathrm{sl}}$. One can think of $\sigma = \sigma_{\mathsf{pk}}$ in the case of HAWK.

A subroutine of Algorithm 6 must 'complete' a vector $(f, g)^\mathsf{T} \in \mathcal{O}_\mathbb{K}^2$, if possible, into a basis $\mathbf{Y} \in \mathcal{O}_\mathbb{K}^{2\times 2}$ with second column $(F, G)^\mathsf{T}$ and determinant one. To perform this operation we define a subroutine called HermiteSolve. This is an algorithm that outputs $\perp$ if and only if the particular vector cannot be completed, and otherwise outputs such a completion.

We also define a procedure Reduce with respect to the form $\mathbf{Q}$. This is a simpler, but less efficient, version of ffNP used in Algorithm 1. It serves two purposes; from a theoretical perspective it ensures that the distribution is well defined, i.e. that the distribution of the output form is independent of the input form being used to sample, and from a practical perspective it ensures the second column of the completed basis is relatively short.

Algorithm 4 $\mathsf{HermiteSolve}(\mathbb{K}, f, g)$: completing f, g if possible.

Require: Conductor $m = 2^\kappa$ cyclotomic $\mathbb{K}$, $f, g \in \mathcal{O}_\mathbb{K}$
Ensure: Completion $F, G \in \mathcal{O}_\mathbb{K}$ such that $\det(\mathbf{Y}) = 1$ if it exists, else $\perp$
1: Let $\mathbf{X} = (\mathsf{rot}(f)\ \mathsf{rot}(g))$
2: Find $\mathbf{U} \in \mathrm{GL}_{2n}(\mathbb{Z})$ such that $\mathbf{X} \cdot \mathbf{U}$ is in Hermite Normal Form
3: **if** $\mathbf{X} \cdot \mathbf{U} \neq (\mathbf{I}_n(\mathbb{Z})\ \mathbf{0})$ **then return** $\perp$
4: Let $(\mathsf{vec}(G)\ -\mathsf{vec}(F))^\mathsf{T}$ be the first column of $\mathbf{U}$ **return** F, G

Algorithm 4 uses the Hermite Normal Form over the integers. If there exist F, G such that $fG - gF = 1$ in $\mathcal{O}_\mathbb{K}$, i.e. such that $\det(\mathbf{Y}) = 1$, then the ideal $(f, g) = \mathcal{O}_\mathbb{K}$, and this is equivalent to the Hermite Normal Form of $(\mathsf{rot}(f)\ \mathsf{rot}(g))$ being $(\mathbf{I}_n(\mathbb{Z})\ \mathbf{0}) \in \mathbb{Z}^{n\times 2n}$. One can then check that setting F, G as in Algorithm 4 satisfies $fG - gF = 1$. Given F, G we use Algorithm 5 to find a short, and canonical with respect to $\mathbf{Q}$, pair $F_\mathbf{Q}, G_\mathbf{Q}$ that also satisfy $fG_\mathbf{Q} - gF_\mathbf{Q} = 1$. Note that in Algorithm 5 we require a rounding function that partitions $\mathbb{R}$, i.e. $\lceil \cdot \rfloor \colon \mathbb{R} \to \mathbb{Z}$ that rounds $a + 1/2$ to $a + 1$ so the preimage of integer a is $[a - 1/2, a + 1/2)$. This rounding is applied coefficientwise.

Algorithm 5 $\mathsf{Reduce}(\mathbf{Q}, f, g, F, G)$: reduction of F, G by $\mathbf{Q}$ and (f, g).

Require: Conductor $m = 2^\kappa$ cyclotomic $\mathbb{K}$, $f, g, F, G \in \mathcal{O}_\mathbb{K}$, $\mathbf{Q} \in \mathcal{H}_2^{>0}(\mathbb{K})$
Ensure: Canonical $F_\mathbf{Q}, G_\mathbf{Q}$ reduced with respect to $\mathbf{Q}$ and (f, g)
1: Let $\mathbf{x} = (f, g)^\mathsf{T}$, $\mathbf{y} = (F, G)^\mathsf{T}$, and $\nu = \left\lceil \frac{\mathbf{x}^* \cdot \mathbf{Q} \cdot \mathbf{y}}{\mathbf{x}^* \cdot \mathbf{Q} \cdot \mathbf{x}} \right\rfloor \in \mathcal{O}_\mathbb{K}$
2: Let $F_\mathbf{Q} = F - \nu f$, $G_\mathbf{Q} = G - \nu g$ **return** $F_\mathbf{Q}, G_\mathbf{Q}$

Algorithm 6 $\mathtt{ac}_\sigma(\mathbf{Q}, \mathbb{K})$: sampling from $\mathsf{AC}_\sigma([\mathbf{Q}]_{\mathrm{sl}}, \mathbb{K})$.

Require: Conductor $m = 2^\kappa$ cyclotomic $\mathbb{K}$, $\mathbf{Q} \in \mathcal{H}_2^{>0}(\mathbb{K})$
Ensure: $\mathbf{R} \in [\mathbf{Q}]_{\mathrm{sl}}$ and $\mathbf{Y} \in \mathrm{SL}_2(\mathcal{O}_\mathbb{K})$ such that $\mathbf{R} = \mathbf{Y}^* \cdot \mathbf{Q} \cdot \mathbf{Y}$
1: Parse $\mathbf{y}_1 \leftarrow D_{\mathbf{Q},\sigma}$ as $\mathbf{y}_1 = (f, g)^\mathsf{T} \in \mathcal{O}_\mathbb{K}^2$
2: **if** $\mathsf{HermiteSolve}(\mathbb{K}, f, g)$ returns $\perp$ **then**
3: **restart**
4: **else** $F, G \leftarrow \mathsf{HermiteSolve}(\mathbb{K}, f, g)$
5: Let $\mathbf{y}_2 = (F_\mathbf{Q}\ G_\mathbf{Q})^\mathsf{T}$ for $F_\mathbf{Q}, G_\mathbf{Q} \leftarrow \mathsf{Reduce}(\mathbf{Q}, f, g, F, G)$
6: Let $\mathbf{Y} = (\mathbf{y}_1\ \mathbf{y}_2)$ and $\mathbf{R} = \mathbf{Y}^* \cdot \mathbf{Q} \cdot \mathbf{Y}$ **return** $(\mathbf{R}, \mathbf{Y})$

The following lemma ensures that a sample $\mathbf{R} \in [\mathbf{Q}]_{\mathrm{sl}}$ output by Algorithm 6 only depends on the class $[\mathbf{Q}]_{\mathrm{sl}}$, and not on the input representative $\mathbf{Q}$. As a result the distribution $\mathsf{AC}_\sigma([\mathbf{Q}]_{\mathrm{sl}}, \mathbb{K})$ is well defined. We can then define our average case module LIP.

Lemma 4. *For any power of two cyclotomic* $\mathbb{K}$, $\mathbf{Q} \in \mathcal{H}_2^{>0}(\mathbb{K})$, $\mathbf{Q}' \in [\mathbf{Q}]_{\mathrm{sl}}$ *and* $\sigma > 0$ *the distributions of* $(\mathbf{R}, \cdot) \leftarrow \mathtt{ac}_\sigma(\mathbf{Q}, \mathbb{K})$ *and* $(\mathbf{R}', \cdot) \leftarrow \mathtt{ac}_\sigma(\mathbf{Q}', \mathbb{K})$ *are equal.*

Proof. For $\mathbf{Q}' \in [\mathbf{Q}]_{\mathrm{sl}}$ there exists a $\mathbf{U} \in \mathrm{SL}_2(\mathcal{O}_\mathbb{K})$ such that $\mathbf{Q}' = \mathbf{U}^* \cdot \mathbf{Q} \cdot \mathbf{U}$. It is sufficient to show that for any $\mathbf{Y}$ created during $\mathtt{ac}_\sigma(\mathbf{Q}, \mathbb{K})$, $\mathbf{Y}' = \mathbf{U}^{-1} \cdot \mathbf{Y}$ is created within $\mathtt{ac}_\sigma(\mathbf{Q}', \mathbb{K})$ with the same probability. Having shown this, since $\mathbf{Y}^* \cdot \mathbf{Q} \cdot \mathbf{Y} = (\mathbf{Y}')^* \cdot \mathbf{Q}' \cdot \mathbf{Y}'$, the two distributions are equal. Firstly, letting $\mathbf{y}_1' = \mathbf{U}^{-1} \cdot \mathbf{y}_1$ we see that $\rho_{\mathbf{Q}',\sigma}(\mathbf{y}_1') = \rho_{\mathbf{Q},\sigma}(\mathbf{y}_1)$. Given that the normalisation constant for a given σ will be equal over all forms in $[\mathbf{Q}]$, the probability of sampling $\mathbf{y}_1' \leftarrow D_{\mathbf{Q}',\sigma}$ is equal to the probability of sampling $\mathbf{y}_1 \leftarrow D_{\mathbf{Q},\sigma}$.

We must now show that, after completing $\mathbf{y}_1'$ to $\mathbf{y}_2'$ via Algorithm 4 and then reducing it with respect to $\mathbf{y}_1'$ and $\mathbf{Q}'$ using Algorithm 5, we have $\mathbf{y}_2' = \mathbf{U}^{-1} \cdot \mathbf{y}_2$. This is precisely the statement that $\mathbf{Y}' = \mathbf{U}^{-1} \cdot \mathbf{Y}$. Note that Algorithm 4 finds a solution if one exists. Since $\mathbf{U}^{-1} \cdot \mathbf{y}_2$ is such a solution, Algorithm 4 succeeds.

We parse $\mathbf{y}_1' = (f'\ g')^\mathsf{T}$ and $\mathbf{y}_2' = (F'\ G')^\mathsf{T}$ so that $f'G' - g'F' = 1$. For fixed (f', g'), any $\tilde{F}, \tilde{G}$ such that $f'\tilde{G} - g'\tilde{F} = 1$ are of the form $\tilde{F} = F' + \lambda \cdot f'$, $\tilde{G} = G' + \lambda \cdot g'$ for some $\lambda \in \mathcal{O}_\mathbb{K}$. For example, one has

$$f' \cdot (G' - \tilde{G}) = g' \cdot (F' - \tilde{F}) \Rightarrow G' - \tilde{G} = g' \cdot \Big(G' \cdot (F' - \tilde{F}) - F' \cdot (G' - \tilde{G})\Big)$$
$$\Rightarrow \lambda = -\Big(G' \cdot (F' - \tilde{F}) - F' \cdot (G' - \tilde{G})\Big),$$

with the same λ for the $F' - \tilde{F}$ case. If we let $\tilde{\mathbf{y}} = \mathbf{U}^{-1} \cdot \mathbf{y}_2$ it is therefore enough to show that $\tilde{\mathbf{y}}$ is the unique reduced completion of $\mathbf{y}_1'$ into $\mathbf{Y}' \in \mathrm{SL}_2(\mathcal{O}_\mathbb{K})$, i.e. $\mathbf{y}_2'$. We have

$$\left\lceil \frac{\mathbf{y}_1'^* \cdot \mathbf{Q}' \cdot \tilde{\mathbf{y}}}{\mathbf{y}_1'^* \cdot \mathbf{Q}' \cdot \mathbf{y}_1'} \right\rfloor = \left\lceil \frac{(\mathbf{U}^{-1} \cdot \mathbf{y}_1)^* \cdot \mathbf{Q}' \cdot (\mathbf{U}^{-1} \cdot \mathbf{y}_2)}{(\mathbf{U}^{-1} \cdot \mathbf{y}_1)^* \cdot \mathbf{Q}' \cdot (\mathbf{U}^{-1} \cdot \mathbf{y}_1)} \right\rfloor = \left\lceil \frac{\mathbf{y}_1^* \cdot \mathbf{Q} \cdot \mathbf{y}_2}{\mathbf{y}_1^* \cdot \mathbf{Q} \cdot \mathbf{y}_1} \right\rfloor = 0,$$

since $\mathbf{y}_2$ is reduced with respect to $\mathbf{y}_1$ and $\mathbf{Q}$ by the construction of $\mathbf{Y}$ in $\mathtt{ac}_s(\mathbf{Q}, \mathbb{K})$. Therefore $\tilde{\mathbf{y}}$ is reduced with respect to $\mathbf{y}_1'$ and $\mathbf{Q}'$.

Definition 8 (Average case smLIP). *Given some power of two cyclotomic* $\mathbb{K}$ *and* $\mathbf{Q} \in \mathcal{H}_2^{>0}(\mathbb{K})$ *an instance of* $\mathsf{ac{-}smLIP}^{\mathbf{Q}}_{\mathbb{K},\sigma}$*, the average case search module Lattice Isomorphism Problem, is given by* $\mathbf{Q}$ *and an element* $\mathbf{Q}' \in [\mathbf{Q}]_{\mathrm{sl}}$ *sampled from* $\mathsf{AC}_\sigma([\mathbf{Q}]_{\mathrm{sl}}, \mathbb{K})$*. A solution is* $\mathbf{U} \in \mathrm{SL}_2(\mathcal{O}_\mathbb{K})$ *such that* $\mathbf{Q}' = \mathbf{U}^* \cdot \mathbf{Q} \cdot \mathbf{U}$*.*

We expect the problem to become harder as the parameter σ increases. In fact, following [15], if σ is large enough we have equivalence with the corresponding worst case problem, assuming that HermiteSolve does not fail too often.

Lemma 5 (Worst case to average case). *Given a machine that can solve* $\mathsf{ac{-}smLIP}^{\mathbf{Q}}_{\mathbb{K},\sigma}$ *in time* T *with probability* $\varepsilon > 0$*, for* $\sigma \geq 2^{\Theta(n)} \cdot \lambda_{2n}([\mathsf{rot}(\mathbf{Q})])$*, one may solve* $\mathsf{wc{-}smLIP}^{\mathbf{Q}}_{\mathbb{K},\ell}$ *in expected time* $T + \mathbb{E}_{samples} \cdot \mathrm{poly}(n, \log \sigma)$ *with probability* ε*. Here* $\mathbb{E}_{samples}(n, \sigma) \geq 1$ *is the expected number of times* $(f\ g)^\mathsf{T}$ *is (re)sampled in Algorithm 6.*

Proof. As input we receive $\mathbf{Q} \in \mathcal{H}_2^{>0}(\mathbb{K})$ and some $\mathbf{Q}' \in [\mathbf{Q}]_{\mathrm{sl}}$. By first LLL reducing $\mathbf{Q}$ (by considering $\mathsf{rot}(\mathbf{Q}) \in \mathbb{Q}^{2n \times 2n}$) we can sample efficiently from $D_{\mathbf{Q},\sigma}$ via Lemma 1, and thus we can sample $(\mathbf{Q}'', \mathbf{U}'') \leftarrow \mathsf{ac}_\sigma(\mathbf{Q}, \mathbb{K})$ in time $\mathbb{E}_{samples} \cdot \mathrm{poly}(n, \sigma)$ by Algorithm 6. The sample $\mathbf{Q}''$ is distributed as $\mathsf{AC}_\sigma([\mathbf{Q}]_{\mathrm{sl}}, \mathbb{K})$, and we have $\mathbf{Q}'' = \mathbf{U}''^* \mathbf{Q} \mathbf{U}''$. We now have an average case instance between $\mathbf{Q}'$ and $\mathbf{Q}''$, which the machine can solve in time T with probability ε. On success we obtain some $\mathbf{U}'$ such that $\mathbf{Q}'' = \mathbf{U}'^* \mathbf{Q}' \mathbf{U}'$, and $\mathbf{U} = \mathbf{U}''(\mathbf{U}')^{-1}$ gives a solution to the worst case instance, i.e. $\mathbf{Q}' = \mathbf{U}^* \mathbf{Q} \mathbf{U}$.

Following the same argument as [15, Lem 3.10] we may reduce σ in the reduction at the expense of an additive loss from the cost of stronger lattice reduction. In Appendix A of the full version [16] we give a heuristic explanation and matching experimental evidence for why HermiteSolve fails with probability around $\frac{1}{4}$, which gives $\mathbb{E}_{samples} \approx \frac{4}{3}$, and thus the reduction above is in fact efficient.

References

1. Agrawal, S., Kirshanova, E., Stehle, D., Yadav, A.: Practical, round-optimal lattice-based blind signatures. Cryptology ePrint Archive, Paper 2021/1565 (2021), https://eprint.iacr.org/2021/1565
2. Agrawal, S., Stehle, D., Yadav, A.: Round-optimal lattice-based threshold signatures, revisited. Cryptology ePrint Archive, Paper 2022/634 (2022). https://doi.org/10.4230/LIPIcs.ICALP.2022.41, https://eprint.iacr.org/2022/634
3. Albrecht, M.R., Ducas, L.: Lattice Attacks on NTRU and LWE: A History of Refinements, pp. 15–40. London Mathematical Society Lecture Note Series, Cambridge University Press (2021). https://doi.org/10.1017/9781108854207.004
4. Babai, L.: On Lovász' lattice reduction and the nearest lattice point problem. Combinatorica **6**(1), 1–13 (1986)
5. Bai, S., Stehlé, D., Wen, W.: Measuring, simulating and exploiting the head concavity phenomenon in BKZ. In: Peyrin, T., Galbraith, S. (eds.) ASIACRYPT 2018, Part I. LNCS, vol. 11272, pp. 369–404. Springer, Heidelberg (Dec 2018). https://doi.org/10.1007/978-3-030-03326-2_13

6. Banaszczyk, W.: New bounds in some transference theorems in the geometry of numbers. Mathematische Annalen **296**(1), 625–635 (1993)
7. Bennett, H., Ganju, A., Peetathawatchai, P., Stephens-Davidowitz, N.: Just how hard are rotations of $\mathbb{Z}^n$? algorithms and cryptography with the simplest lattice. Cryptology ePrint Archive, Report 2021/1548 (2021), https://eprint.iacr.org/2021/1548
8. Brakerski, Z., Langlois, A., Peikert, C., Regev, O., Stehlé, D.: Classical hardness of learning with errors. In: Boneh, D., Roughgarden, T., Feigenbaum, J. (eds.) 45th ACM STOC. pp. 575–584. ACM Press (Jun 2013). https://doi.org/10.1145/2488608.2488680
9. Chen, Y., Nguyen, P.Q.: BKZ 2.0: Better lattice security estimates. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 1–20. Springer, Heidelberg (Dec 2011). https://doi.org/10.1007/978-3-642-25385-0_1
10. Dachman-Soled, D., Ducas, L., Gong, H., Rossi, M.: LWE with side information: Attacks and concrete security estimation. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020, Part II. LNCS, vol. 12171, pp. 329–358. Springer, Heidelberg (Aug 2020). https://doi.org/10.1007/978-3-030-56880-1_12
11. Ducas, L., Durmus, A., Lepoint, T., Lyubashevsky, V.: Lattice signatures and bimodal Gaussians. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part I. LNCS, vol. 8042, pp. 40–56. Springer, Heidelberg (Aug 2013). https://doi.org/10.1007/978-3-642-40041-4_3
12. Ducas, L., Nguyen, P.Q.: Faster Gaussian lattice sampling using lazy floating-point arithmetic. In: Wang, X., Sako, K. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 415–432. Springer, Heidelberg (Dec 2012). https://doi.org/10.1007/978-3-642-34961-4_26
13. Ducas, L., Nguyen, P.Q.: Learning a zonotope and more: Cryptanalysis of NTRUSign countermeasures. In: Wang, X., Sako, K. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 433–450. Springer, Heidelberg (Dec 2012). https://doi.org/10.1007/978-3-642-34961-4_27
14. Ducas, L., Prest, T.: Fast Fourier orthogonalization. In: Proceedings of the ACM on International Symposium on Symbolic and Algebraic Computation. pp. 191–198. ISSAC '16, Association for Computing Machinery, New York, NY, USA (2016). https://doi.org/10.1145/2930889.2930923
15. Ducas, L., van Woerden, W.P.J.: On the lattice isomorphism problem, quadratic forms, remarkable lattices, and cryptography. In: Dunkelman, O., Dziembowski, S. (eds.) EUROCRYPT 2022, Part III. LNCS, vol. 13277, pp. 643–673. Springer, Heidelberg (May / Jun 2022). https://doi.org/10.1007/978-3-031-07082-2_23
16. Ducas, L., Postlethwaite, E.W., Pulles, L.N., van Woerden, W.: Hawk: Module LIP makes lattice signatures fast, compact and simple. Cryptology ePrint Archive, Paper 2022/1155 (2022), https://eprint.iacr.org/2022/1155
17. Espitau, T., Fouque, P.A., Gérard, F., Rossi, M., Takahashi, A., Tibouchi, M., Wallet, A., Yu, Y.: Mitaka: A simpler, parallelizable, maskable variant of falcon. In: Dunkelman, O., Dziembowski, S. (eds.) EUROCRYPT 2022, Part III. LNCS, vol. 13277, pp. 222–253. Springer, Heidelberg (May / Jun 2022). https://doi.org/10.1007/978-3-031-07082-2_9
18. Genise, N., Micciancio, D.: Faster Gaussian sampling for trapdoor lattices with arbitrary modulus. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018, Part I. LNCS, vol. 10820, pp. 174–203. Springer, Heidelberg (Apr / May 2018). https://doi.org/10.1007/978-3-319-78381-9_7

19. Gentry, C., Peikert, C., Vaikuntanathan, V.: Trapdoors for hard lattices and new cryptographic constructions. In: Ladner, R.E., Dwork, C. (eds.) 40th ACM STOC. pp. 197–206. ACM Press (May 2008). https://doi.org/10.1145/1374376.1374407
20. Gentry, C., Szydlo, M.: Cryptanalysis of the revised NTRU signature scheme. In: Knudsen, L.R. (ed.) EUROCRYPT 2002. LNCS, vol. 2332, pp. 299–320. Springer, Heidelberg (Apr / May 2002). https://doi.org/10.1007/3-540-46035-7_20
21. Golomb, S.: Run-length encodings (corresp.). IEEE Transactions on Information Theory **12**(3), 399–401 (1966). https://doi.org/10.1109/TIT.1966.1053907
22. Hoffstein, J., Howgrave-Graham, N., Pipher, J., Silverman, J.H., Whyte, W.: NTRUSIGN: Digital signatures using the NTRU lattice. In: Joye, M. (ed.) CT-RSA 2003. LNCS, vol. 2612, pp. 122–140. Springer, Heidelberg (Apr 2003). https://doi.org/10.1007/3-540-36563-X_9
23. Kirchner, P.: Algorithms on ideal over complex multiplication order. Cryptology ePrint Archive, Report 2016/220 (2016), https://eprint.iacr.org/2016/220
24. Lenstra Jr., H.W., Silverberg, A.: Lattices with symmetry. Journal of Cryptology **30**(3), 760–804 (Jul 2017). https://doi.org/10.1007/s00145-016-9235-7
25. Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 1–23. Springer, Heidelberg (May / Jun 2010). https://doi.org/10.1007/978-3-642-13190-5_1
26. Micciancio, D., Regev, O.: Worst-case to average-case reductions based on Gaussian measures. SIAM Journal on Computing **37**(1), 267–302 (2007). https://doi.org/10.1137/S0097539705447360
27. Nguyen, P.Q., Regev, O.: Learning a parallelepiped: Cryptanalysis of GGH and NTRU signatures. Journal of Cryptology **22**(2), 139–160 (Apr 2009). https://doi.org/10.1007/s00145-008-9031-0
28. Peikert, C.: An efficient and parallel Gaussian sampler for lattices. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 80–97. Springer, Heidelberg (Aug 2010). https://doi.org/10.1007/978-3-642-14623-7_5
29. Pornin, T., Prest, T.: More efficient algorithms for the NTRU key generation using the field norm. In: Lin, D., Sako, K. (eds.) PKC 2019, Part II. LNCS, vol. 11443, pp. 504–533. Springer, Heidelberg (Apr 2019). https://doi.org/10.1007/978-3-030-17259-6_17
30. Postlethwaite, E.W., Virdia, F.: On the success probability of solving unique SVP via BKZ. In: Garay, J. (ed.) PKC 2021, Part I. LNCS, vol. 12710, pp. 68–98. Springer, Heidelberg (May 2021). https://doi.org/10.1007/978-3-030-75245-3_4
31. Prest, T.: Gaussian sampling in lattice-based cryptography. Ph.D. thesis, Ecole normale supérieure-ENS PARIS (2015)
32. Prest, T., Fouque, P.A., Hoffstein, J., Kirchner, P., Lyubashevsky, V., Pornin, T., Ricosset, T., Seiler, G., Whyte, W., Zhang, Z.: FALCON. Tech. rep., National Institute of Standards and Technology (2020), available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions
33. Szydlo, M.: Hypercubic lattice reduction and analysis of GGH and NTRU signatures. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 433–448. Springer, Heidelberg (May 2003). https://doi.org/10.1007/3-540-39200-9_27
34. development team, T.F.: fpylll, a Python wraper for the fplll lattice reduction library, Version: 0.5.6 (2021), https://github.com/fplll/fpylll, available at https://github.com/fplll/fpylll