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

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

[1] CWI, Cryptology Group, Amsterdam, the Netherlands
[2] Mathematical Institute, Leiden University, Leiden, The Netherlands
{ewp,lnp}@cwi.nl

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 [46]. Like its predecessor NTRUSign it has a hash-then-sign design, but fixes the issue of signature transcript leakage [40] via Discrete Gaussian Sampling (DGS) [27].

Since its introduction much progress has been made into making DGS more efficient [18,17,26], in particular by exploiting ideal or module structures [41,20] 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 $\mathbb{Z}^n$ [21,11]. 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 [21] that this avoids the difficult DGS step above: sampling from the $\mathbb{Z}^n$ lattice is much easier.

This work. The work [21], 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 [28,33].

Instead we work with rank two modules, for which the LIP problem has already received some cryptanalytic attention [47]. It was quickly noted that NTRUSign signatures [30] 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 [30,42,46]. Following the ideas presented in [21] 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 [21] 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-

	[46] FALCON 512	This work HAWK 512	Gain ($\frac{\text{FALCON}}{\text{HAWK}}$)	[46] 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 [7]. 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 [23] 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$_{\mathbb{Z}}$ [23, 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. Appendix A examines the key generation of Hawk and its relation to the smLIP reduction. Appendix B introduces the one more SVP problem and reduces the strong forgery security of Hawk to it for particular parametrisations. Appendix C discusses implementation details in greater depth. Appendix D describes in detail the concrete security model of Falcon and compares Hawk to it in this model.

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).

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

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 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

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

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 ([12, 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 ([9, 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 *as*

$$\eta_\varepsilon(\Lambda) = \arg\min_{\sigma>0}\left[\rho_{\frac{1}{2\pi\sigma}}\left(\hat{\Lambda} \setminus \{\mathbf{0}\}\right)\right] \leq \varepsilon.$$

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 [36, 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 ([36, 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 $\left\{(x_1, \ldots, x_n) \in \mathbb{C}^n : \forall i \in [n/2], x_{i+n/2} = \overline{x}_i\right\} \subset \mathbb{C}^n$ is isomorphic as an inner product space to $\mathbb{R}^n$ [35, Sec. 2.1]. We implicitly use this isomorphism and write $\sigma\colon \mathbb{K} \to \mathbb{R}^n$. We also have the coefficient embedding $\mathsf{vec}\colon \mathbb{K} \to \mathbb{Q}^n,\ a_0 + a_1 X + \cdots + a_{n-1}X^{n-1} \mapsto (a_0, a_1, \ldots, a_{n-1})^\mathsf{T}$, 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 $\|\cdot\|\colon \mathbb{K} \to \mathbb{Q},\ f \mapsto \sqrt{\langle f, f \rangle}$, which agrees with the Euclidean norm of $\mathsf{vec}(f)$. As $\mathbb{K}$ is a CM field, it has an automorphism $\cdot^*\colon \mathbb{K} \to \mathbb{K}$ 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 $\mathsf{vec}\colon \mathbb{K}^\ell \to \mathbb{Q}^{n\ell}$ 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 $\mathsf{rot}(f) = \left(\mathsf{vec}(f), \mathsf{vec}(Xf), \ldots, \mathsf{vec}(X^{n-1}f)\right) \in \mathbb{Q}^{n \times n}$ 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 $\boldsymbol{\sigma} = (\sigma, \ldots, \sigma) \colon \mathbb{K}^k \to \mathbb{R}^{nk}$, $(x_1, \ldots, x_k) \mapsto (\sigma(x_1), \ldots, \sigma(x_k))$. 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 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.

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

We then introduce (compressed) HAWK and discuss how the security of HAWK directly reduces to that of the uncompressed HAWK.

3.1 Uncompressed HAWK

The uncompressed version of our signature scheme is based on the scheme presented in [21, 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 [21, 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.

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

1: Sample $f, g \in R$ with coefficients from $D_{\mathbb{Z},\sigma_{\mathsf{pk}}}$
2: $q_{00} = f^*f + g^*g$
3: **if** $2 \mid \mathrm{N}(f)$ **or** $2 \mid \mathrm{N}(g)$ **or** $\|(f, g)\|^2 \leq \sigma_{\mathsf{sec}}^2 \cdot 2n$ **then**
4: **restart**
5: $(F, G)^\mathsf{T} \leftarrow \mathsf{TowerSolve}_{n,1}(f, g)$ [42, Alg. 4], **if** $\perp$, **restart**
6: $(F, G)^\mathsf{T} \leftarrow (F, G)^\mathsf{T} - \mathsf{ffNP}_R\left(\frac{f^*F+g^*G}{q_{00}}, \mathsf{ffLDL}_R^*(q_{00})\right) \cdot (f, g)^\mathsf{T}$ [20]
7: $\mathbf{B} = \begin{pmatrix} f & F \\ g & G \end{pmatrix}$.
8: $\mathbf{Q} = \begin{pmatrix} q_{00} & q_{01} \\ q_{10} & q_{11} \end{pmatrix} = \mathbf{B}^* \cdot \mathbf{B}$.
9: **return** $(\mathsf{pk}, \mathsf{sk}) = (\mathbf{Q}, \mathbf{B})$

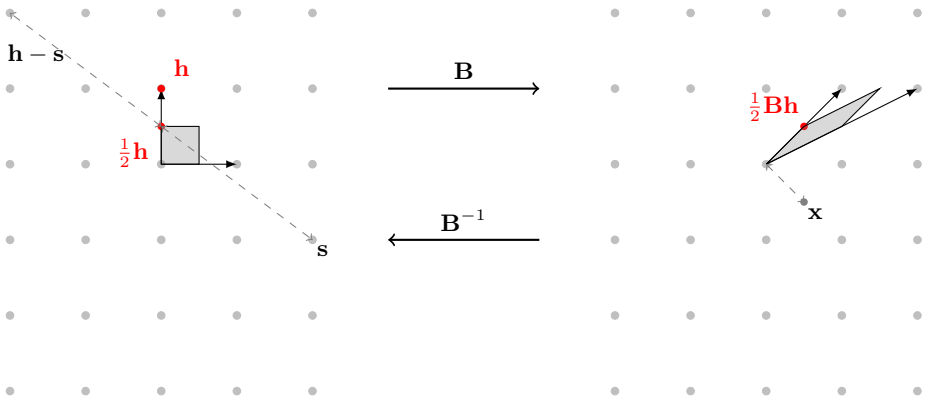


Fig. 1: Illustration of signing. First $\mathbf{h}$ is sampled (left), then $\mathbf{B}$ is applied, a short lattice point $\mathbf{x}$ is sampled from a discrete Gaussian on $\mathbb{Z}^{2n} + \frac{1}{2}\mathbf{B} \cdot \mathbf{h}$ (right). Finally $\mathbf{B}^{-1}$ applied to $\mathbf{x}$ is subtracted 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 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 $\texttt{sym-break}(h_1 - 2s_1)$ 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** $\texttt{sym-break}(h_1 - 2s_1)$ is `True` **and** $\left\|\frac{\mathbf{h}}{2} - \mathbf{s}\right\|_{\mathbf{Q}}^2 \le \sigma_{\mathsf{ver}}^2 \cdot 2n$ **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 [42, 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 [20] to obtain a shorter second basis vector $(F, G)^\mathsf{T}$. In KGen checks are performed prior to completing the basis $\mathbf{B}$. In TowerSolve! [42] 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 [7]. 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 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 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, $\mathbb{E}[\|\mathbf{x}\|] \approx \sigma\sqrt{d}$ [36, Sec. 4]. For example, the verification of signatures is determined by a distance, say ℓ. Instead of referring to ℓ, we use the shorthand $\sigma_{\mathsf{ver}} = \ell/\sqrt{d}$.

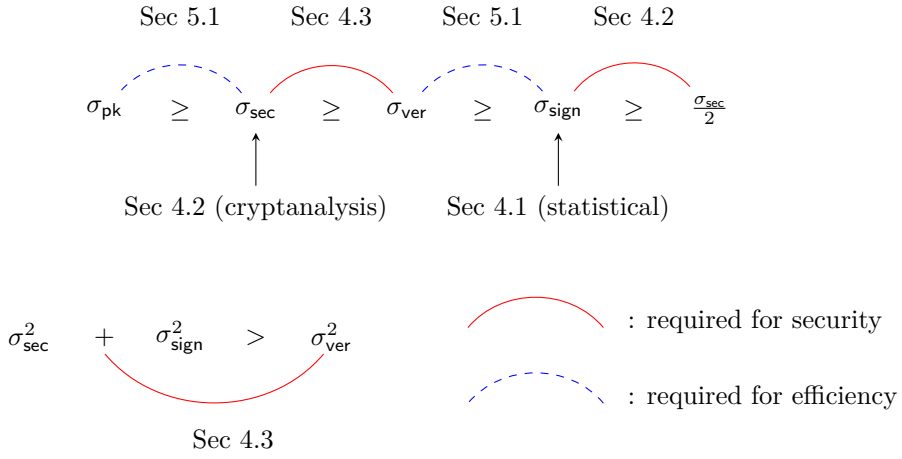


Fig. 2: A summary of the necessary relationships between the various $\sigma_\bullet$.

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 [40,19] 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 [46, 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 [46, 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 [21], 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.

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

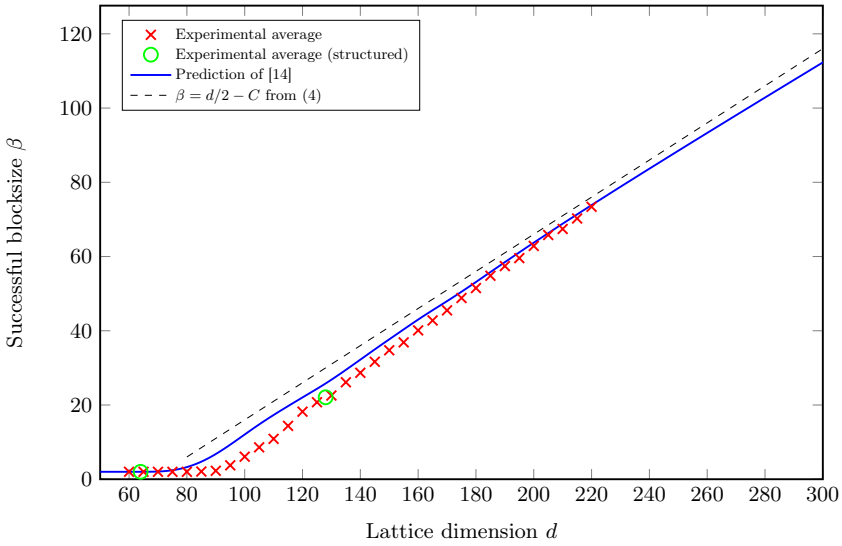
Unusual-SVP. The shortest vectors in R^2 have length 1, which is a factor of order $\Theta(\sqrt{n})$ shorter than predicted by the Gaussian heuristic. Recovering such 'unusually' short vectors is easier than generic shortest vectors, and can be achieved by running the BKZ lattice reduction algorithm 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 [4] 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

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

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 [13,8,14,43].

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 [14], which applies the concrete improvements mentioned above, and experimental data. These experiments apply the BKZ2.0 algorithm with lattice point enumeration as implemented in [48] 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 [21, 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 [14] 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.

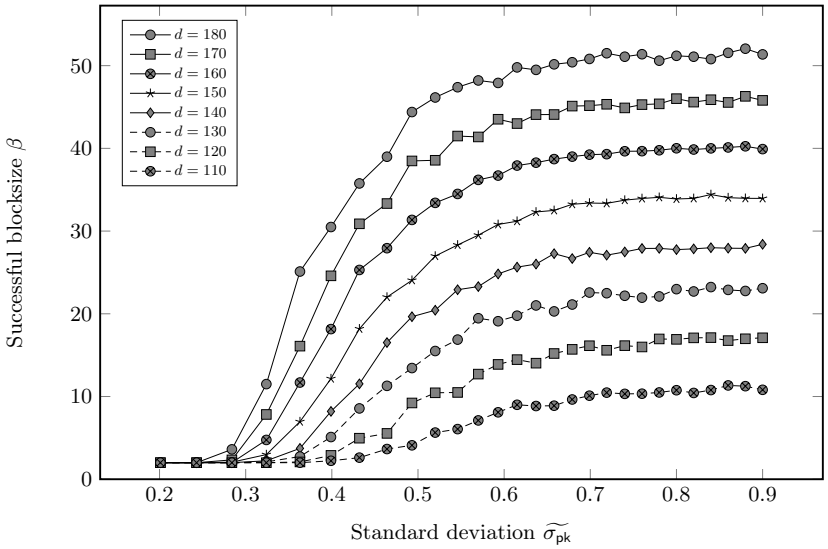


We ran progressive BKZ (one tour per blocksize) over $\mathbb{Z}^d$ using an input form generated with $\sigma_{\mathsf{pk}} = 20$ and report the average successful β that recovered a length one vector over 40 instances. We used the BKZ simulator and probabilistic model of [14], accounting for the d 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.

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 [14] 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 [11, 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 [14]. We assume the value σ_{sec} represents a lower 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 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`.

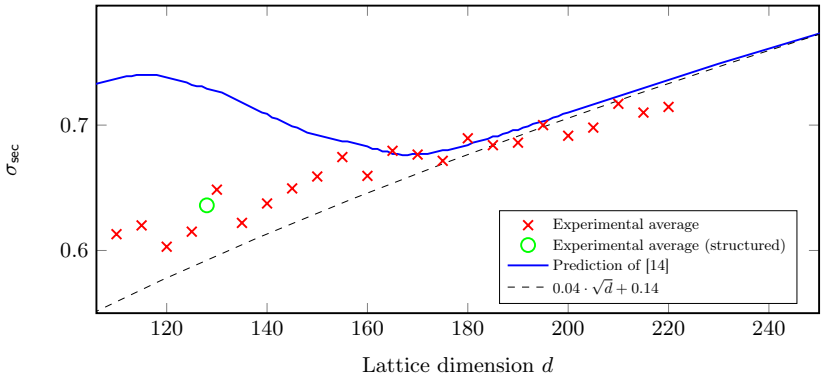
Fig. 4: Blocksize required to recover a shortest vector via lattice reduction as a function of the standard deviation $\widetilde{\sigma_{\mathsf{pk}}}$.

We note that the prediction of [14] 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.

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



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}.

SVP instance with the same approximation factor over the same lattice. we determine the expected blocksize β using the BKZ simulator [14] 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 [14] 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 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

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

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 [14] 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) 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 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.

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 $q_s \cdot 2^{-\lambda}$ [46, 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

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

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 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 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 we reduce the strong signature forgery security of Hawk's design to omSVP. To ensure the parametrisation of the omSVP problem we reduce to

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

	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.

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.

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 [29]. This encoding is used for the signatures in FALCON [46].

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 [46, 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 [44, 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 [46, 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 prime modulus $p \equiv 1 \pmod{2n}$ that is sufficiently large. Since $\mathbb{F}_p^\times$ is a cyclic group of order $p-1$, there exists an element $\omega \in \mathbb{F}_p^\times$ of order $2n$ and one can perform a discrete analog of the FFT by computing $f(\omega^i) \in R$ 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.

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 [39] 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`), requiring $8n$ bytes per polynomial. 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 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.

Note that, in contrast to FALCON, we sample a new salt and therefore receive a new target $\mathbf{t}$ whenever Sign restarts. We make this choice as we do not see how reusing the same $\mathbf{t}$ can be made compatible with the security argument of [27]. 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 [46], 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

	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.

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 TowerSolvel, 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 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 [28,33,31], 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 $\mathsf{HermiteSolve}$. 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 $\mathsf{ac}_\sigma(\mathbf{Q}, \mathbb{K})$: 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 $(\mathbf{R}, \cdot) \leftarrow \mathsf{ac}_\sigma(\mathbf{Q}, \mathbb{K})$ and $(\mathbf{R}', \cdot) \leftarrow \mathsf{ac}_\sigma(\mathbf{Q}', \mathbb{K})$ 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 $\mathsf{ac}_\sigma(\mathbf{Q}, \mathbb{K})$, $\mathbf{Y}' = \mathbf{U}^{-1} \cdot \mathbf{Y}$ is created within $\mathsf{ac}_\sigma(\mathbf{Q}', \mathbb{K})$ 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

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

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 $\mathsf{ac}_s(\mathbf{Q}, \mathbb{K})$. 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 [21], if σ is large enough we have equivalence with the corresponding worst case problem, assuming that $\mathsf{HermiteSolve}$ 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 [21, 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 we give a heuristic explanation and matching experimental evidence for why $\mathsf{HermiteSolve}$ 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., Bai, S., Ducas, L.: A subfield lattice attack on overstretched NTRU assumptions - cryptanalysis of some FHE and graded encoding schemes. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016, Part I. LNCS, vol. 9814, pp. 153–178. Springer, Heidelberg (Aug 2016). https://doi.org/10.1007/978-3-662-53018-4_6
4. 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
5. Albrecht, M.R., Ducas, L., Herold, G., Kirshanova, E., Postlethwaite, E.W., Stevens, M.: The general sieve kernel and new records in lattice reduction. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019, Part II. LNCS, vol. 11477, pp. 717–746. Springer, Heidelberg (May 2019). https://doi.org/10.1007/978-3-030-17656-3_25
6. Alkim, E., Ducas, L., Pöppelmann, T., Schwabe, P.: Post-quantum key Exchange—A new hope. In: 25th USENIX Security Symposium (USENIX Security 16). pp. 327–343. USENIX Association, Austin, TX (Aug 2016), https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/alkim
7. Babai, L.: On Lovász' lattice reduction and the nearest lattice point problem. Combinatorica **6**(1), 1–13 (1986)
8. 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
9. Banaszczyk, W.: New bounds in some transference theorems in the geometry of numbers. Mathematische Annalen **296**(1), 625–635 (1993)
10. Banaszczyk, W.: Inequalities for convex bodies and polar reciprocal lattices in R^n. Discrete Comput. Geom. **13**(2), 217–231 (dec 1995). https://doi.org/10.1007/BF02574039
11. 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
12. 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
13. 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
14. 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
15. Dubhashi, D.P., Panconesi, A.: Concentration of Measure for the Analysis of Randomized Algorithms. Cambridge University Press (2009). https://doi.org/10.1017/CBO9780511581274
16. Ducas, L.: Shortest vector from lattice sieving: A few dimensions for free. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018, Part I. LNCS, vol. 10820, pp. 125–145. Springer, Heidelberg (Apr / May 2018). https://doi.org/10.1007/978-3-319-78381-9_5
17. 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
18. 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
19. 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
20. 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
21. 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
22. Duda, J., Tahboub, K., Gadgil, N.J., Delp, E.J.: The use of asymmetric numeral systems as an accurate replacement for huffman coding. In: 2015 Picture Coding Symposium (PCS). pp. 65–69 (2015). https://doi.org/10.1109/PCS.2015.7170048
23. 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
24. Espitau, T., Tibouchi, M., Wallet, A., Yu, Y.: Shorter hash-and-sign lattice-based signatures. Cryptology ePrint Archive, Paper 2022/785 (2022), https://eprint.iacr.org/2022/785, https://eprint.iacr.org/2022/785
25. Ferraguti, A., Micheli, G.: On the Mertens–Cesàro theorem for number fields. Bulletin of the Australian Mathematical Society **93**(2), 199–210 (2016). https://doi.org/10.1017/S0004972715001288
26. 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
27. 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
28. 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
29. Golomb, S.: Run-length encodings (corresp.). IEEE Transactions on Information Theory **12**(3), 399–401 (1966). https://doi.org/10.1109/TIT.1966.1053907
30. 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
31. Kirchner, P.: Algorithms on ideal over complex multiplication order. Cryptology ePrint Archive, Report 2016/220 (2016), https://eprint.iacr.org/2016/220
32. Laurent, B., Massart, P.: Adaptive estimation of a quadratic functional by model selection. Annals of Statistics **28**(5), 1302–1338 (2000)
33. 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
34. Li, J., Nguyen, P.Q.: A complete analysis of the BKZ lattice reduction algorithm. Cryptology ePrint Archive, Report 2020/1237 (2020), https://eprint.iacr.org/2020/1237
35. 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
36. 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
37. Micciancio, D., Walter, M.: Practical, predictable lattice basis reduction. In: Fischlin, M., Coron, J.S. (eds.) EUROCRYPT 2016, Part I. LNCS, vol. 9665, pp. 820–849. Springer, Heidelberg (May 2016). https://doi.org/10.1007/978-3-662-49890-3_31
38. MILNOR, J.: Introduction to Algebraic K-Theory. (AM-72). Princeton University Press (1971), http://www.jstor.org/stable/j.ctt1b9x0xv
39. Montgomery, P.L.: Modular multiplication without trial division. Mathematics of computation **44**(170), 519–521 (1985)
40. 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
41. 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
42. 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
43. 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
44. Prest, T.: Gaussian sampling in lattice-based cryptography. Ph.D. thesis, Ecole normale supérieure-ENS PARIS (2015)

45. 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 (2017), available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-1-submissions
46. 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
47. 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
48. 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

A On the failure probability of completion

Crucial to HAWK (and also FALCON) is the completion of a vector $(f, g)^{\mathsf{T}}$ to a basis with determinant 1 (or q for FALCON). This completion can fail, either inherently because no such completion exists, or because the fast TowerSolveI method fails to find it. In this appendix we explain heuristically, and confirm experimentally, that approximately $\frac{1}{4}$ of f, g sampled from a discrete Gaussian are not completable. This implies that the average case sampler of Algorithm 6 is in fact efficient, for σ large enough to ensure efficient Gaussian sampling. This is the best case we can hope for, given that we expect $\frac{1}{4}$ of such f, g to both have even algebraic norm, which means they cannot be completed, see below. If we consider the extra conditions required by HAWK's KGen, Algorithm 1, most pertinently that f, g both have odd algebraic norm, then approximately $\frac{3}{4}$ of f, g will cause Algorithm 1 to restart. This is also the best case we can expect, as we expect half of all pairs to be such that f (exclusive) or g has even algebraic norm and therefore to fail in Algorithm 1, see below.

We then show that on completable samples TowerSolveI fails with only a small probability, both in the case of f, g being sampled from a discrete Gaussian, and also of such f, g that further satisfy the requirements of Algorithm 1. We can therefore adapt our worst case to average case reduction to HAWK's KGen. We begin with some relevant algebra, then examine the number of completable f, g in the two cases and discuss the probability that TowerSolveI succeeds on completable cases. We then discuss how to adapt our worst case to average case reduction to HAWK's KGen.

Requirements of HermiteSolve and TowerSolveI and algebra. Note that $\mathcal{O}_{\mathbb{K}}$ is a Dedekind domain, since $\mathbb{K}$ is a number field. For $f, g \in \mathcal{O}_{\mathbb{K}}$ we denote their principal ideals as $(f) = f\mathcal{O}_{\mathbb{K}}$ and $(g) = g\mathcal{O}_{\mathbb{K}}$. Elements f and g are completable if and only if (f) and (g) are comaximal; $(f) + (g) = \mathcal{O}_{\mathbb{K}}$. In a Dedekind domain this is equivalent to the ideal factorisations $(f) = \mathfrak{p}_1^{e_1} \cdots \mathfrak{p}_a^{e_a}$ and $(g) = \mathfrak{q}_1^{h_1} \cdots \mathfrak{q}_b^{h_b}$ being such that no $\mathfrak{p}_i = \mathfrak{q}_j$. Since HermiteSolve always finds a completion if one exists, it succeeds in exactly these cases. Elements f and g will be completed by TowerSolveI if and only if the greatest common divisor of algebraic norms $\mathrm{N}(f)$ and $\mathrm{N}(g)$ divides q. In our case, where $q = 1$, this is if and only if $\mathrm{N}(f)$ and $\mathrm{N}(g)$ are coprime. This is a potentially stronger condition than (f) and (g) being comaximal, as in a Dedekind domain it requires not only that no $\mathfrak{p}_i = \mathfrak{q}_j$ but also that no $\mathfrak{p}_i$ and $\mathfrak{q}_j$ lie above the same rational prime. (In the case of FALCON where q is the prime 12289, one allows (f) and (g) to both have prime ideals that lie above 12289 in their ideal factorisations, provided none of these prime ideals above 12289 are contained in both ideal factorisations.)

The facts in the following discussion are only necessarily true for power of two cyclotomics. In a power of two cyclotomic with degree n and conductor $m = 2n$ for a rational prime p we denote by r its ramification degree, by i its inertia degree and by s its splitting degree, and have the relationship $n = ris$. That is, $(p) = \mathfrak{p}_1^r \cdots \mathfrak{p}_s^r$ where $\mathfrak{p}_j$ has ideal norm $\mathfrak{N}(\mathfrak{p}_j) = [\mathcal{O}_{\mathbb{K}} : \mathfrak{p}_j] = p^i$. The only ramified rational prime is 2 which has $(r, i, s) = (n, 1, 1)$. For all other rational primes p

we have $r = 1$ and i equals the order of p in $(\mathbb{Z}/m\mathbb{Z})^\times$. We have $\phi(m) = n$ and for $m > 4$, i.e. $m \geq 8$, we know $(\mathbb{Z}/m\mathbb{Z})^\times$ is not cyclic. Therefore the order of p in this group is less than n and therefore $i < n$. Hence for $p > 2$ and $m > 4$ we know $s = n/i > 1$ and the rational prime is split. When these hypotheses are satisfied there are no inert rational primes.

In Dedekind domains (integral) ideals decompose uniquely into prime ideals, and the ideal norm is multiplicative, so $\mathfrak{N}((f)) = \mathfrak{N}(\mathfrak{p}_1)^{e_1} \cdots \mathfrak{N}(\mathfrak{p}_a)^{e_a}$. Here, the $\mathfrak{p}_j$ in the ideal factorisation of (f) do not necessarily all lie above the same rational prime. Note that $\mathrm{N}(f) = \mathfrak{N}((f))$, i.e. the algebraic norm of f equals the ideal norm of its principal ideal. In particular, this means that $\mathrm{N}(f)$ is even if and only if the ideal factorisation of (f) contains the unique prime ideal above 2. For some f, g, if both algebraic norms are even then the unique prime ideal above 2 is in the ideal factorisations of both (f) and (g), so they cannot be comaximal, and therefore f, g cannot be completed. Also note that in a Dedekind domain, for any pair of (integral) ideals $\mathfrak{a}, \mathfrak{b}$ such that $\mathfrak{a} \subset \mathfrak{b}$ there exists an ideal $\mathfrak{c}$ such that $\mathfrak{a} = \mathfrak{b}\mathfrak{c}$, see e.g. [38, p. 9]. In particular, this means that if $f \in \mathfrak{p}$ for some prime ideal $\mathfrak{p}$, then $(f) \subset \mathfrak{p}$ and therefore $(f) = \mathfrak{p}\mathfrak{c}$ for some $\mathfrak{c}$, so $\mathfrak{p}$ is a prime ideal in the ideal factorisation of (f).

For $c \in \mathbb{Z}_{\geq 1}$ a definition of a density $\mathbb{D}^c$ for subsets of $\mathcal{O}_\mathbb{K}^c$ is given in [25]. This density is not well defined for all subsets, but in the case of an (integral) ideal $\mathfrak{a}$ we have $\mathbb{D}^1(\mathfrak{a}) = 1/\mathfrak{N}(\mathfrak{a})$. If we let $C = \{(f, g) \in \mathcal{O}_\mathbb{K}^2 \colon (f) + (g) = \mathcal{O}_\mathbb{K}\}$ then we have $\mathbb{D}^2(C) = 1/\zeta_\mathbb{K}(2)$, where $\zeta_\mathbb{K}$ is the Dedekind zeta function for $\mathbb{K}$. We wish to define a heuristic for sampling elements from $\mathcal{O}_\mathbb{K}^c$ such that they fall into a subset with probability equal to its density. More precisely, for the appropriate c we denote by (†) the heuristic that we are sampling $x = (x_1, \ldots, x_c)$ from some distribution over $\mathcal{O}_\mathbb{K}^c$ such that for any $X \subset \mathcal{O}_\mathbb{K}^c$ for which $\mathbb{D}^c(X)$ is well defined, $\Pr[x \in X] = \mathbb{D}^c(X)$. We will also assume that as a given Gaussian parameter σ increases, the behaviour of sampling from $D_{\Lambda,\sigma}$ becomes closer to that of (†). In particular, we show experimental evidence for $\sigma = 1.5$ below, and assume that when σ is large enough to satisfy either Algorithm 6 or Lemma 5, heuristic (†) is at least as valid.

The proportion of completable f and g, and the efficiency of Algorithm 6. Recall the output of HermiteSolve is $\perp$ if and only if the input f, g are not comaximal ideals. We therefore need to estimate the probability that this occurs. Under (†) the probability of non comaximal f and g is $1 - 1/\zeta_\mathbb{K}(2)$. This quantity is investigated in [3, Lem. 1], where it is shown that as our power of two conductor tends to infinity, $\zeta_\mathbb{K}(2)$ tends to $4/3$. We therefore expect approximately $1/4$ of pairs (f, g) to be non comaximal. However, we may be concerned that when sampling f, g from $D_{\mathbf{Q},\sigma}$ we deviate too far from (†), or that our n is too small for [3, Lem. 1]. We therefore provide an estimate for this quantity and experimental results below. In Figure 6 we set $\mathbf{Q} = \mathbf{I}_2(\mathbb{K})$ for power of two cyclotomics $\mathbb{K}$ with $n \geq 4$ and $\sigma = 1.5$. The estimate is calculated under (†) as follows. First we consider the single ramified prime 2, with unique prime ideal $\mathfrak{p}$ above it. We have $\Pr[f \in \mathfrak{p}] = 1/\mathfrak{N}(\mathfrak{p}) = 1/2$ and the same independently for g, hence f, g are not completable because they share $\mathfrak{p}$ in their ideal factorisation with probability

$1/4$. We therefore assume a starting probability of $3/4$ for f, g being completable. For the next 10,000 rational primes p we calculate (i, s) and approximate the probability that f and g share none of the s prime ideals above p in their ideal factorisation as $(1 - 1/p^{2i})^s$. This is approximate since, for a *single* prime ideal $\mathfrak{p}$ above p, we have $\Pr[f \in \mathfrak{p}] = \Pr[g \in \mathfrak{p}] = 1/\mathfrak{N}(\mathfrak{p}) = 1/p^i$, and so the probability that at most one of f, g is in $\mathfrak{p}$ is $1 - 1/p^{2i}$. We are assuming this probability is independent over prime ideals above p. We also assume the probabilities for distinct p are independent and take the product of the above approximations with $3/4$.

In Figure 7 we present the same experiments where we further enforce that $\mathrm{N}(f)$ and $\mathrm{N}(g)$ are both odd and not too short, as required in Algorithm 1. We calculate our estimated probability by instead starting with probability $1/4$, before multiplying the approximate probabilities $(1 - 1/p^{2i})^s$, since we now only allow $f \notin \mathfrak{p}$ and $g \notin \mathfrak{p}$, for $\mathfrak{p}$ the unique prime ideal above 2. Our estimates in this case are therefore exactly one third of our estimates when we allow $\mathrm{N}(f)$ and $\mathrm{N}(g)$ to have different parities.

It remains to ensure the sampler within Algorithm 6 is efficient. By Lemma 1 this is ensured by taking $\sigma \geq \|\tilde{\mathbf{B}}_{\mathsf{rot}(\mathbf{Q})}\| \cdot \frac{1}{\pi} \cdot \sqrt{\log(4n+4)/2}$. Given the experimental accuracy of our theoretical estimates for the proportion of completable f, g in Figure 6, we assume that sampling from the average case distribution of Algorithm 6 requires a small constant number of internal repetitions.

On HermiteSolve vs. TowerSolve. While in theory HermiteSolve is efficient, and guarantees us the 'ground truth' on completability, in practice it is too costly in terms of both time and space. In Algorithm 1 we therefore make use of the elegant TowerSolve of [42]. Recall that TowerSolve can sometimes fail, even if a completion was possible.

We give a heuristic argument for the fraction of completable f, g that can be completed by TowerSolve. If we are considering f, g under (†) then we start with probability $3/4$ of them being completable via TowerSolve, which represents at most one of them having even algebraic norm. When, as in Algorithm 1, we further demand that both f and g have odd algebraic norm we start with probability $1/4$ of them being successful in Algorithm 1. Again our estimates in this second case are exactly a third of when we allow $\mathrm{N}(f)$ and $\mathrm{N}(g)$ to have different parities. We then proceed similarly to our estimate for completability, except for a rational prime p with (i, s) we estimate the probability that prime ideals above p are in the ideal factorisations of at most one of (f) and (g) as $1-(1-(1-1/p^i)^s)^2$. Here $1-(1-1/p^i)^s$ is an approximation of the probability that some prime ideals above p are in the ideal factorisation of e.g. (f), so this quantity squared is an approximation of the probability that some prime ideals above p (not necessarily the same ones) are in the the ideal factorisations of both (f) and (g). This is precisely the event we wish to avoid. See Figure 6 and Figure 7 for an experimental validation of this method.

Worst case to HAWK KGen. Formally, if we were to replace HermiteSolve with TowerSolve in Algorithm 6, or add any of the restart conditions of Algorithm 1,

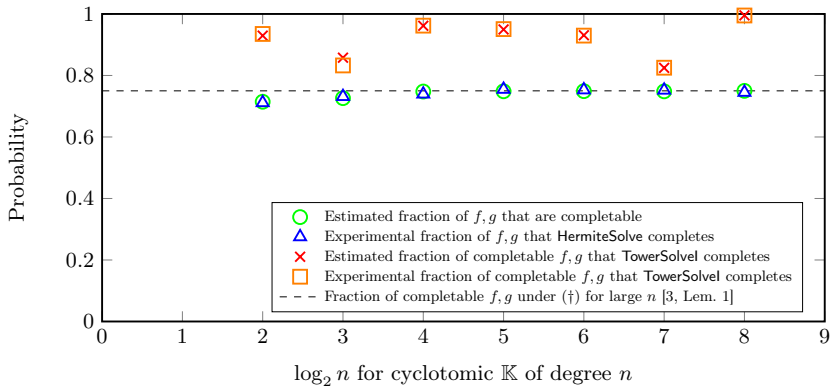
the distribution $\mathsf{AC}_\sigma([\mathbf{Q}]_{\mathrm{sl}}, \mathbb{K})$ would no longer necessarily be well defined. For example, setting $\mathbf{Q} = \mathbf{I}_2(\mathbb{K})$, if $(f, g)^\mathsf{T} = \mathbf{y}_1$ is completable, but not by TowerSolve, then $\mathbf{R} = \mathbf{Y}^*\mathbf{Y}$ would be output by the original Algorithm 6 but not by one where HermiteSolve has been replaced by TowerSolve. Since $\mathbf{R} \in [\mathbf{I}_2(\mathbb{K})]_{\mathrm{sl}}$ and $\mathbf{y}_1 = (1, 0)^\mathsf{T}$ is completable by both HermiteSolve and TowerSolve such that $\mathbf{Y} = \mathbf{I}_2(\mathbb{K})$ we have $\mathbf{R} = \mathbf{Y}^*\mathbf{R}\mathbf{Y}$ is output by both versions of Algorithm 6.

Our technique to show a reduction from $\mathsf{wc{-}smLIP}^{\mathbf{I}_2(\mathbb{K})}_{\mathbb{K},2}$ to Hawk KGen is as follows. We describe an average case distribution that is equivalent to $\mathsf{AC}_\sigma([\mathbf{I}_2(\mathbb{K})]_{\mathrm{sl}}, \mathbb{K})$ in terms of hardness, i.e. one that can be efficiently transformed to and from the original average case distribution. We do this to account for the different reduction methods used to determine $(F, G)^\mathsf{T} = \mathbf{y}_2$ in Algorithm 1 and Algorithm 6, namely ffNP and Algorithm 5 respectively. This new average case distribution has the exact same output as Algorithm 1 when completable $(f, g)^\mathsf{T} = \mathbf{y}_1$ that do not cause restarts in Algorithm 1 are internally sampled. Let p_r denote the probability that Algorithm 1 restarts when it internally samples a completable $(f, g)^\mathsf{T}$. If $\mathcal{A}$ is an adversary against Hawk's KGen using σ_{pk}, i.e. a machine that can return $\mathbf{B} \in \mathrm{SL}_2(\mathcal{O}_\mathbb{K})$ such that $\mathbf{B}^*\mathbf{B} = \mathbf{Q}$ for a Hawk public key $\mathbf{Q}$ in time t and with probability ε, then $\mathcal{A}$ is an adversary for $\mathsf{ac{-}smLIP}^{\mathbf{I}_2(\mathbb{K})}_{\mathbb{K},\sigma_{\mathsf{pk}}}$ that runs in time t with probability at least $(1 - p_r) \cdot \varepsilon$. We give experimental and theoretical evidence that the component of p_r caused by the use of TowerSolve and requiring both of $\mathrm{N}(f)$ and $\mathrm{N}(g)$ to be odd in Algorithm 1 is a constant in Figure 7. The other potential restarts, and their similarly small contribution to p_r, are discussed in Section 5 and Appendix C. If σ_{pk} is large enough then $\mathcal{A}$ is transformed into a worst case adversary via Lemma 5.

It remains to describe this equivalent average case distribution. We add a final step to Algorithm 6 which applies a Hermitian form version of ffNP with respect to $\mathbf{Q}$ to the output form $\mathbf{R}$. This is efficient, and can be efficiently reversed by applying (a Hermitian form version of) Algorithm 5, hence the two average case distributions are equivalently hard. If a completable $(f, g)^\mathsf{T} = \mathbf{y}_1$ is sampled in Algorithm 1 and Algorithm 6, Algorithm 1 does not restart, and we apply this extra postprocessing step to Algorithm 6, then the outputs will be identical.

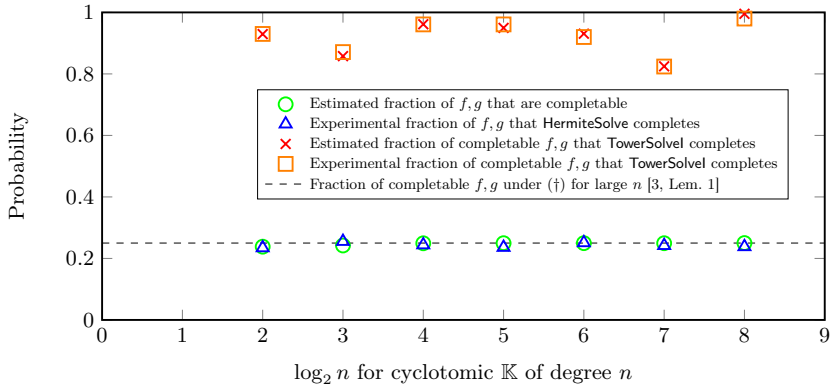
B The One More SVP Problem

The one more SVP problem, henceforth omSVP, is the problem upon which we base the forgery security of our signatures. Informally we define an average case omSVP instance that samples a $\mathbf{Q}$ from a distribution over some $\mathcal{H}^{>0}_\ell(\mathbb{K})$ and gives Gaussian samples according to this $\mathbf{Q}$. If one can find some non zero $\mathbf{x}$ that is sufficiently short with respect to $\mathbf{Q}$, and that is in some sense non trivially new, then one solves the problem. We show that when this $\mathbf{Q}$ is sampled as in Algorithm 6 and for certain parameters, if one can forge a signature against a 'provable' variant of Hawk then in the programmable random oracle model one can solve an instance of this problem. We then discuss parametrisations of our omSVP problem that we expect to be hard, with reference to Hawk parameters.



The estimated fraction of completable f, g, and the estimated fraction of these that TowerSolve is able to complete, are calculated as above. We then set $\sigma = 1.5$ and generate 5000 samples of f, g from a discrete Gaussian distribution over $\mathbb{Z}$ and check which can be completed by HermiteSolve and TowerSolve.

Fig. 6: Experimental justification that Algorithm 6 is efficient, and TowerSolve behaves similarly to HermiteSolve.



The estimated fraction of completable f, g, and the estimated fraction of these that TowerSolve is able to complete, are calculated as above in the case where f, g are sampled from Algorithm 1. We then set $\sigma = 1.5$ and generate 5000 samples of f, g from Algorithm 1 and check which can be completed by HermiteSolve and TowerSolve.

Fig. 7: Experimental justification that Algorithm 6 is efficient, and that TowerSolve behaves similarly to HermiteSolve, in the case where f, g are sampled as in Algorithm 1.

We first define the ROM-SUF-CMA game for signature schemes. Here Sign and Vf have access to a random oracle $\mathsf{RO}\colon \{0,1\}^* \to \{0,1\}^{\ell(n)}$, the output length of which depends on the security parameter n. The security notion we consider here is strong unforgeability under chosen message attacks in the random oracle model, or ROM-SUF-CMA, see Figure 8.

$\mathrm{ROM\text{-}SUF\text{-}CMA}_{\Pi,\mathcal{A}}(1^n)$

1 : $\mathcal{L}_{\mathsf{Sign}} \leftarrow \emptyset$
2 : $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KGen}(1^n)$
3 : $(\mathsf{m}^\star, \mathsf{sig}^\star) \leftarrow \mathcal{A}^{\mathsf{OSign}(\cdot),\mathsf{RO}(\cdot)}(\mathsf{pk})$
4 : **return** $[\![\mathsf{Vf}^{\mathsf{RO}(\cdot)}(\mathsf{pk}, \mathsf{m}^\star, \mathsf{sig}^\star) = 1 \wedge (\mathsf{m}^\star, \mathsf{sig}^\star) \notin \mathcal{L}_{\mathsf{Sign}}]\!]$

$\mathsf{OSign}(\mathsf{m})$

1 : $\mathsf{sig} \leftarrow \mathsf{Sign}^{\mathsf{RO}(\cdot)}(\mathsf{sk}, \mathsf{m})$
2 : $\mathcal{L}_{\mathsf{Sign}} \leftarrow \mathcal{L}_{\mathsf{Sign}} \cup \{(\mathsf{m}, \mathsf{sig})\}$
3 : **return** sig

$\mathsf{RO}(\mathsf{x})$

1 : **if** $T[\mathsf{x}] = \perp$ **then** // initially $T[\mathsf{x}] = \perp$ for all x
2 : $T[\mathsf{x}] \leftarrow U(\{0,1\}^{\ell(n)})$
3 : **return** $T[\mathsf{x}]$

Fig. 8: The ROM-SUF-CMA game.

Definition 9 (ROM-SUF-CMA **security).** *Let $\Pi = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vf})$ be a signature scheme. We say that Π is* $(t, \varepsilon, q_s, q_h)$-ROM-SUF-CMA *secure, or strongly unforgeable under chosen message attacks in the random oracle model, if for any adversary $\mathcal{A}$ running in time at most t, making at most q_s queries to its signing oracle, and making at most q_h queries to its random oracle, the probability* $\Pr[\mathrm{ROM\text{-}SUF\text{-}CMA}_{\Pi,\mathcal{A}} = 1] \leq \varepsilon$.

Next we define a set of length preserving functions of $\mathcal{O}_{\mathbb{K}}$ that will represent 'trivial' wins in our omSVP game. For example, when $\mathbb{Z} = \mathcal{O}_{\mathbb{Q}}$ one can always negate $\mathbf{x}$ to $-\mathbf{x}$ while preserving the length with respect to any Hermitian form.

Definition 10. *For a number field $\mathbb{K}$ let $\mu_{\mathbb{K}}$ be the set of roots of unity.*

Lemma 6. *For $\alpha \in \mathcal{O}_{\mathbb{K}}$ we have $\alpha^*\alpha = 1 \iff \alpha \in \mu_{\mathbb{K}}$.*

Proof. If $\alpha^*\alpha = 1$ then for any field embedding σ_i we have $|\sigma_i(\alpha)| = 1$ and therefore $|\sigma_i(\alpha)^j| = 1$ for all $j \in \mathbb{Z}$. Assume α is not a root of unity, so no such $\sigma_i(\alpha)^j = 1$, then $\{\sigma_i(\alpha)^j\}_{j\in\mathbb{Z}}$ is dense in the unit circle of $\mathbb{C}$, a contradiction. The converse follows from $\sigma_i(\alpha^*\alpha) = |\sigma_i(\alpha)|^2$ and noting that $\alpha \in \mu_{\mathbb{K}}$ must have $|\sigma_i(\alpha)| = 1$ else no power of it will equal 1.

Lemma 7. *For any $\ell \in \mathbb{Z}_{\geq 1}$ some $\alpha \in \mathcal{O}_{\mathbb{K}}$ has $\|\alpha\mathbf{x}\|_{\mathbf{Q}} = \|\mathbf{x}\|_{\mathbf{Q}}$ for all $\mathbf{x} \in \mathbb{K}^\ell$ and $\mathbf{Q} \in \mathcal{H}_\ell^{>0}(\mathbb{K})$ if and only if $\alpha \in \mu_{\mathbb{K}}$.*

Proof. First note if $\alpha \in \mu_{\mathbb{K}}$ then $\|\alpha\mathbf{x}\|_{\mathbf{Q}} = \mathrm{Tr}(\alpha^*\alpha\mathbf{x}^*\mathbf{Q}\mathbf{x})/n = \mathrm{Tr}(\mathbf{x}^*\mathbf{Q}\mathbf{x})/n = \|\mathbf{x}\|_{\mathbf{Q}}$ by Lemma 6. Conversely, if $\|\alpha\mathbf{x}\|_{\mathbf{Q}} = \|\mathbf{x}\|_{\mathbf{Q}}$ for all $\mathbf{x}$ and $\mathbf{Q}$ then it must be true for $\mathbf{x} = (1, 0, \ldots, 0)^{\mathsf{T}}$ and $\mathbf{Q} = \mathbf{I}_\ell(\mathbb{K})$, which by unrolling the definitions tells us $\sum_{i=1}^n \sigma_i(\alpha^*\alpha) = n$. Similarly, it must be true for $\mathbf{x} = (\alpha, 0, \ldots, 0)^{\mathsf{T}}$ and $\mathbf{Q} = \mathbf{I}_\ell(\mathbb{K})$, which by unrolling the definitions tells us $\sum_{i=1}^n \sigma_i(\alpha^*\alpha)^2 = n$. Note that for a set of real numbers $\{y_i\}_{i=1}^n$ such that $\sum_{i=1}^n y_i = \sum_{i=1}^n y_i^2 = n$ we have $y_1 = \cdots = y_n = 1$. Therefore $\sigma_i(\alpha^*\alpha) = 1$ for all i, and $\alpha^*\alpha = 1$, then $\alpha \in \mu_{\mathbb{K}}$ by Lemma 6.

In short, the above lemmas tell us that multiplying a solution in our omSVP game by a root of unity should be considered a trivial win, and disallowed.

Lemma 8. *For power of two cyclotomics of conductor $m = 2n$ we have $\mu_{\mathbb{K}} = \{X^i : i = 0, \ldots, 2n-1\}$.*

Proof. Recall $\mathcal{O}_{\mathbb{K}} = \mathbb{Z}[X]/(X^n + 1)$ and if $\alpha = \sum_{i=0}^{n-1} \alpha_i X^i$ then $\alpha^* = \alpha_0 - \sum_{i=1}^{n-1} \alpha_i X^{n-i}$. From $\alpha^*\alpha = 1$ we therefore have $\sum_{i=0}^{n-1} \alpha_i^2 = 1$ for $\alpha_i \in \mathbb{Z}$.

In this instance $i = 0$ represents the identity and $i = n$ represents negation.

Definition 11 (Average case omSVP). *An average case* omSVP *instance is the pair* ac−omSVP $= (\mathsf{Init}, \mathsf{samp})$. *On input 1^n,* Init *returns a form $\mathbf{Q}$ sampled from some distribution over some $\mathcal{H}_{\ell_n}^{>0}(\mathbb{K}_n)$, the roots of unity $\mu_{\mathbb{K}_n}$ for $\mathbb{K}_n$, a length bound L_n, and a Gaussian parameter σ_n. On input $\mathbf{Q}$,* samp *returns a sample from $D_{\mathbf{Q},\sigma_n}$.*

$\mathrm{SAMPLE}_{\mathsf{ac-omSVP},\mathcal{A}}(1^n)$

1 : $\mathcal{L}_{\mathsf{samples}} \leftarrow \{\mathbf{0}\}$
2 : $(\mathbf{Q}, \mu_{\mathbb{K}}, L, \sigma) \leftarrow \mathsf{Init}(1^n)$
3 : $\mathbf{x}^\star \leftarrow \mathcal{A}^{\mathsf{samp}(\mathbf{Q})}(\mathbf{Q})$
4 : **return** $[\![\|\mathbf{x}^\star\|_{\mathbf{Q}} \leq L \wedge \mathbf{x}^\star \notin \mathcal{L}_{\mathsf{samples}}]\!]$

$\mathsf{samp}(\mathbf{Q})$

1 : $\mathbf{x} \leftarrow D_{\mathbf{Q},\sigma}$
2 : $\mathcal{L}_{\mathsf{samples}} \leftarrow \mathcal{L}_{\mathsf{samples}} \cup \{\alpha\mathbf{x}\}_{\alpha \in \mu_{\mathbb{K}}}$
3 : **return** $\mathbf{x}$

Fig. 9: The SAMPLE game.

The adversary in Figure 9 wins whenever it can use the form $\mathbf{Q}$ and the samples it receives from samp to return some non trivial new element of $\mathcal{O}_{\mathbb{K}}^\ell$ that is short enough.

Definition 12 (SAMPLE security). *Let* ac−omSVP $= (\mathsf{Init}, \mathsf{samp})$ *be an average case* omSVP *instance. We say that* ac−omSVP *is (t, ε, q_o)-*SAMPLE *secure, if for any adversary $\mathcal{A}$ running in time at most t, and making at most q_o queries to* samp*, the probability* $\Pr[\mathrm{SAMPLE}_{\mathsf{ac-omSVP},\mathcal{A}} = 1] \leq \varepsilon$.

We use the following lemma in our reduction from ac−omSVP to Hawk. Each signature output by Hawk induces a sample from $D_{\mathbf{Q},\mathbb{Z}^{2n}+\frac{1}{2}\mathbf{h}_i,\sigma_{\mathsf{sign}}}$ for an i.i.d. uniformly chosen $\mathbf{h}_i$ in $\{0,1\}^{2n}$. We show that multiplying such a sample by two gives a distribution close to a discrete Gaussian over $\mathbb{Z}^{2n}$ with Gaussian parameter $2\sigma_{\mathsf{sign}}$.

Lemma 9. *Let D represent the distribution formed by first sampling $\mathbf{h} \leftarrow U(\{0,1\}^n)$ and then returning $\mathbf{x} \leftarrow D_{\mathbf{Q},\mathbb{Z}^n+\frac{1}{2}\mathbf{h},\sigma}$. If $\sigma \geq \eta_\varepsilon(\mathbb{Z}^n)$ then*

$$\Delta(2 \cdot D, D_{\mathbf{Q},2\sigma}) \leq \varepsilon/(1-\varepsilon).$$

Proof. For all $\mathbf{y} \in \mathbb{Z}^n$, $\mathbf{y}/2$ lies in a coset of the form $\mathbb{Z}^n + \frac{1}{2}\mathbf{h}$ for a unique $\mathbf{h} \in \{0,1\}^n$. We have

$$\Pr[D = \mathbf{y}/2] = 2^{-n} \cdot D_{\mathbf{Q},\mathbb{Z}^n+\frac{1}{2}\mathbf{h},\sigma}(\mathbf{y}/2) = 2^{-n} \cdot \rho_{\mathbf{Q},\mathbb{Z}^n+\frac{1}{2}\mathbf{h},\sigma}(\mathbf{y}/2)/\rho_{\mathbf{Q},\sigma}\left(\mathbb{Z}^n + \frac{1}{2}\mathbf{h}\right)$$

and $\Pr[D_{\mathbf{Q},2\sigma} = \mathbf{y}] = \rho_{\mathbf{Q},2\sigma}(\mathbf{y})/\rho_{\mathbf{Q},2\sigma}(\mathbb{Z}^n)$. Note that for any such $\mathbf{y}$ we have $\rho_{\mathbf{Q},\mathbb{Z}^n+\frac{1}{2}\mathbf{h},\sigma}(\mathbf{y}/2) = \rho_{\mathbf{Q},2\sigma}(\mathbf{y})$. It remains to argue about the sizes of $2^n \cdot \rho_{\mathbf{Q},\sigma}(\mathbb{Z}^n + \frac{1}{2}\mathbf{h})$ and $\rho_{\mathbf{Q},2\sigma}(\mathbb{Z}^n)$. From Lemma 3 we know that $2^n\rho_{\mathbf{Q},\sigma}(\mathbb{Z}^n + \frac{1}{2}\mathbf{h})$, $\rho_{\mathbf{Q},2\sigma}(\mathbb{Z}^n) \in [1-\varepsilon, 1+\varepsilon] \cdot 2^n(\sqrt{2\pi}\sigma)^n$. Therefore, for each such $\mathbf{y}$ we have $D(\mathbf{y}/2) \in [\frac{1-\varepsilon}{1+\varepsilon}, \frac{1+\varepsilon}{1-\varepsilon}] \cdot D_{\mathbf{Q},2\sigma}(\mathbf{y})$, and therefore $\Delta(2 \cdot D, D_{\mathbf{Q},2\sigma}) \leq \varepsilon/(1-\varepsilon)$.

We can now show that a ROM-SUF-CMA adversary against Hawk can be used as a subroutine for a SAMPLE adversary against an ac−omSVP instance. In the below we consider a 'provable' version of Hawk that samples its public keys exactly from the average case distribution of Algorithm 6. We also consider growing n which are powers of two, and assert that the internal values $(\sigma_{\mathsf{pk}}, \sigma_{\mathsf{sign}}, \sigma_{\mathsf{ver}}, \mathsf{saltlen})$ can be determined from n. Note that we do not expect the reduction to be bidirectional; intuitively even if one can find an $\mathbf{x}^\star$ that wins the SAMPLE game, one must also find a message and salt that hashes into a particular coset to make this a successful signature forgery.

Lemma 10. *Let $\Pi = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vf})$ be the provable version of Hawk described above with internal parameters $(\sigma_{\mathsf{pk}}, \sigma_{\mathsf{sign}}, \sigma_{\mathsf{ver}}, \mathsf{saltlen})$ such that $\sigma_{\mathsf{sign}} \geq \eta_\varepsilon(\mathbb{Z}^{2n})$ for some $\varepsilon(n) \in \mathsf{negl}(n)$ and $\sigma_{\mathsf{ver}} > \sigma_{\mathsf{sign}}$. Define ac−omSVP $= (\mathsf{Init}, \mathsf{samp})$ such that $\mathsf{Init}(1^n)$ samples $\mathbf{Q} \leftarrow \mathsf{AC}_{\sigma_{\mathsf{pk}}}([\mathbf{I}_2(\mathbb{K})]_{\mathrm{sl}}, \mathbb{K})$ for $\mathbb{K}$ the cyclotomic field of degree n. Also let $\mathsf{Init}(1^n)$ output the corresponding $\mu_{\mathbb{K}}$, the length bound $L = 2\sigma_{\mathsf{ver}}\sqrt{2n}$ and $\sigma = 2\sigma_{\mathsf{sign}}$. If $\mathcal{A}$ is a (t, δ, q_s, q_h)-ROM-SUF-CMA adversary against Π with random oracle RO that has output length $\ell(n) = 2n$, then in the reprogrammable random oracle model there exists a (t', δ', q_o)-SAMPLE adversary $\mathcal{B}$ against ac−omSVP such that $t' \approx t$, $q_o = q_s$ and*

$$\delta' \geq \delta - \mathsf{abort} - \mathsf{sim} - \mathsf{multitarget},$$

where $\mathsf{abort}, \mathsf{sim}$ and $\mathsf{multitarget}$ are made explicit in the proof. In particular, provided $q_s, q_h, |\mu_{\mathbb{K}}| \in \mathsf{poly}(n)$ and $\mathsf{saltlen} \in \omega(\log n)$, $\mathsf{abort}, \mathsf{sim}$ and $\mathsf{multitarget} \in \mathsf{negl}(n)$. Finally, t' is t plus the time required to query samp q_s times, answer q_h RO queries, and reprogram RO at q_s points.

Proof. We let $\mathcal{B}$ interact with $\mathcal{A}$ and reprogram RO. The form $\mathbf{Q}$ output by Init in the $\text{SAMPLE}_{\mathsf{ac-omSVP},\mathcal{B}}$ game is sent by $\mathcal{B}$ to $\mathcal{A}$ to act as pk in the $\text{ROM-SUF-CMA}_{\Pi,\mathcal{A}}$ game. Two types of queries from $\mathcal{A}$ must be simulated by $\mathcal{B}$. First, it must simulate RO, which it does via lazy sampling of bitstrings in $\{0,1\}^{2n}$. Secondly, it must simulate OSign. To do this, on receipt of message m_i from $\mathcal{A}$, $\mathcal{B}$ samples a uniform salt r_i from $\{0,1\}^{\mathsf{saltlen}}$ and checks whether $\mathsf{m}_i\|\mathsf{r}_i$ has already been queried by $\mathcal{A}$ to RO. If so, $\mathcal{B}$ aborts. If not, $\mathcal{B}$ queries samp, receives $\mathbf{x}_i$ and checks whether $\|\mathbf{x}_i\|_{\mathbf{Q}} > 2\sigma_{\mathsf{ver}}\sqrt{2n}$. If so, $\mathcal{B}$ aborts. If not, $\mathcal{B}$ reprograms $\mathsf{RO}(\mathsf{m}_i\|\mathsf{r}_i) = \mathbf{h}_i = \mathsf{parity}(\mathbf{x}_i)$ and responds to the $\mathsf{OSign}(\mathsf{m}_i)$ query with $\mathsf{sig}_i = (\mathsf{r}_i, \mathbf{s}_i = \mathbf{h}_i/2 \pm \mathbf{x}_i/2)$. The choice $\pm\mathbf{x}_i$ is made according to line 8 of Algorithm 2. Here $\mathsf{parity}\colon \mathbb{Z}^{2n} \to \{0,1\}^{2n}$, $(x_1, \ldots, x_{2n}) \mapsto (x_1 \bmod 2, \ldots, x_{2n} \bmod 2)$ maps an integer vector to the bitstring of its parities. When $\mathcal{A}$ outputs its forgery $(\mathsf{m}^\star, \mathsf{sig}^\star = (\mathsf{r}^\star, \mathbf{s}^\star))$ in the $\text{ROM-SUF-CMA}_{\Pi,\mathcal{A}}$ game, $\mathcal{B}$ computes $\mathbf{h}^\star = \mathsf{RO}(\mathsf{m}^\star\|\mathsf{r}^\star)$ and submits $\mathbf{x}^\star = \mathbf{h}^\star - 2\mathbf{s}^\star$ in the $\text{SAMPLE}_{\mathsf{ac-omSVP},\mathcal{B}}$ game. Note that since the length bound L output by Init is exactly twice the distance of a valid $\mathbf{s}^\star$ to $\frac{1}{2}\mathbf{h}^\star$, $\mathbf{x}^\star$ will always be short enough if $\mathcal{A}$ output a valid forgery.

We first argue about the probability that $\mathcal{B}$ aborts. By a union bound, the probability that $\mathcal{B}$ aborts because some $\mathsf{m}\|\mathsf{r}$ has already been queried to RO is at most $q_s \cdot q_h \cdot 2^{-\mathsf{saltlen}}$. Indeed, the probability is maximised if all q_h queries $\mathcal{A}$ makes to RO are of the form $\mathsf{m}\|\mathsf{u}$ for some fixed message and distinct strings u of length $\mathsf{saltlen}$, and then $\mathcal{A}$ makes q_s queries to OSign with m. By the tail bound of Lemma 2 and a union bound, since $\sigma_{\mathsf{ver}} > \sigma_{\mathsf{sign}}$ the probability that $\mathcal{B}$ aborts because a sample $\mathbf{x}_i$ from samp is too long is in $q_s \cdot \exp(-\Theta(n))$. Call the sum of these probabilities abort.

If $\mathcal{B}$ does not abort, we quantify the statistical distance between its simulations of RO and OSign and their true distributions. When $\mathcal{A}$ queries RO the lazy sampling method of $\mathcal{B}$ is exactly correct and thus uniform. When $\mathcal{B}$ reprograms the random oracle at some $\mathsf{m}_i\|\mathsf{r}_i$, it takes value $\mathsf{parity}(\mathbf{x}_i)$ for $\mathbf{x}_i \leftarrow D_{\mathbf{Q},2\sigma_{\mathsf{sign}}}$. We examine the probability of $2\mathbb{Z}^{2n} + \mathbf{c}$ under $D_{\mathbf{Q},2\sigma_{\mathsf{sign}}}$ for any $\mathbf{c} \in \{0,1\}^{2n}$, since these represent preimages of parity, and compare them to the ideal (uniform) probability of 2^{-2n} for a bitstring. Using Lemma 3 we can bound $\rho_{\mathbf{Q},2\sigma_{\mathsf{sign}}}(2\mathbb{Z}^{2n} + \mathbf{c}) = \rho_{\mathbf{Q},\sigma_{\mathsf{sign}}}(\mathbb{Z}^{2n} + \mathbf{c}/2)$ by recalling $\sigma_{\mathsf{sign}} \geq \eta_\varepsilon(\mathbb{Z}^{2n})$. Letting $\gamma = \rho_{\mathbf{Q},2\sigma_{\mathsf{sign}}}(\mathbb{Z}^{2n})$ we therefore know $D_{\mathbf{Q},2\sigma_{\mathsf{sign}}}(2\mathbb{Z}^{2n}+\mathbf{c}) \in [1-\varepsilon, 1+\varepsilon]\cdot(\sqrt{2\pi}\sigma_{\mathsf{sign}})^{2n}/\gamma$ for all $\mathbf{c}$. We also know that 2^{-2n} is in the same range, therefore the statistical distance between $\mathsf{parity}(\mathbf{x}_i)$ for $\mathbf{x}_i \leftarrow D_{\mathbf{Q},2\sigma_{\mathsf{sign}}}$ and the uniform distribution over $\{0,1\}^{2n}$ is at most $\varepsilon/(1-\varepsilon) \in \mathsf{negl}(n)$. Therefore the probability that $\mathcal{A}$ behaves differently given the simulated RO is at most $q_s \cdot \varepsilon/(1-\varepsilon)$. Next we consider the simulation of OSign by $\mathcal{B}$. By Lemma 9 we have that the statistical distance between $\mathbf{B}^{-1} \cdot \mathbf{x}$ on line 7 of Algorithm 2 and halving a sample from $D_{\mathbf{Q},2\sigma_{\mathsf{sign}}}$ is at most $\varepsilon/(1-\varepsilon) \in \mathsf{negl}(n)$. Therefore, the probability that $\mathcal{A}$ behaves differently given the simulated OSign is at most $q_s \cdot \varepsilon/(1-\varepsilon)$. Call the sum of these probabilities sim.

Finally, we discuss the probability that a successful signature forgery from $\mathcal{A}$ $(\mathsf{m}^\star, \mathsf{sig}^\star = (\mathsf{r}^\star, \mathbf{s}^\star))$ is such that $\mathbf{x}^\star = \mathbf{h}^\star - 2\mathbf{s}^\star$ is *not* a valid solution to the

$\text{SAMPLE}_{\mathsf{ac-omSVP},\mathcal{B}}$ game for $\mathcal{B}$. Recall $\mathbf{h}^\star = \mathsf{RO}(\mathsf{m}^\star\|\mathsf{r}^\star)$. This happens when $\mathbf{x}^\star \in \{\alpha \cdot \mathbf{x}_i\}_{i,\alpha\in\mu_\mathbb{K}}$ or $\mathbf{x}^\star = \mathbf{0}$, i.e. $\mathbf{x}^\star \in \mathcal{L}_{\mathsf{samples}}$. Since $\mathsf{sig}^\star$ is a successful forgery we have $\mathsf{RO}(\mathsf{m}^\star\|\mathsf{r}^\star) = \mathsf{parity}(\mathbf{x}^\star)$. In particular, $\mathcal{A}$ has found such an input output pair for RO where the output is from a set of size $|\mu_\mathbb{K}| \cdot q_s + 1$. This can happen in two ways. First, $\mathcal{A}$ may find a preimage of some $\mathsf{parity}(\mathbf{x})$ for $\mathbf{x} \in \mathcal{L}_{\mathsf{samples}}$ in one of its q_h RO queries. We assume that the RO queries of $\mathcal{A}$ never repeat an $\mathbf{m}_i\|\mathsf{r}_i$ query it has previously learnt from a signing query, as in doing so $\mathcal{A}$ would learn nothing new. Therefore, these queries give such a preimage with probability at most $q_h \cdot (q_s \cdot |\mu_\mathbb{K}| + 1) \cdot 2^{-2n}$. Second, the forgery submission of $\mathcal{A}$ may imply a preimage of $\mathsf{parity}(\mathbf{x})$ for $\mathbf{x} \in \mathcal{L}_{\mathsf{samples}}$. In the case where $(\mathsf{m}^\star\|\mathsf{r}^\star)$ has not been either queried or programmed for RO, this happens with probability $(q_s \cdot |\mu_\mathbb{K}| + 1) \cdot 2^{-2n}$. Finally, since $\mathcal{A}$ may return a different signature on the same message, it is possible that $\mathcal{A}$ returns $(\mathsf{m}_i, \mathsf{sig}^\star = (\mathsf{r}_i, \mathbf{s}^*))$ where the message and the salt are from a previous signature query. In this case $\mathbf{s}^\star \neq \mathbf{s}_i$ else it is not a weak forgery. Hence $\mathbf{x}^\star = \mathsf{RO}(\mathsf{m}_i\|\mathsf{r}_i) - 2\mathbf{s}^\star = \mathbf{h}_i - 2\mathbf{s}^\star$. As $\mathbf{s}^\star \neq \mathbf{s}_i$ we have $\mathbf{x}^\star \neq \mathbf{x}_i$, but $\mathbf{x}^\star$ does lie in the same coset $2\mathbb{Z}^{2n} + \mathbf{x}_i$. To ensure that $\mathbf{x}^\star$ is not in $\mathcal{L}' = \mathcal{L}_{\mathsf{samples}} \setminus \{\mathbf{x}_i\}$ we must argue that no $\mathbf{x} \in \mathcal{L}'$ has $\mathsf{parity}(\mathbf{x}) = \mathsf{parity}(\mathbf{x}_i)$. We begin by showing $\alpha\mathbf{x}_i$ for some $\alpha \in \mu_\mathbb{K} \setminus \{\pm 1\}$ has $\mathsf{parity}(\mathbf{x}_i) = \mathsf{parity}(\alpha\mathbf{x}_i)$ with negligible probability. Indeed, if this equality holds, then the two bitstrings of length n have a period strictly dividing n. As n is a power of two, this means the period divides $n/2$ and there are $2^{n/2}$ bitstrings of length n inside $\mathsf{parity}(\mathbf{x}_i)$ with such period. Therefore there is a $2^n/2^{2n} = 2^{-n}$ fraction of all cosets that $\mathbf{x}_i$ could fall into, such that one of its rotations has the same parity as itself. An output of RO is either chosen uniformly or as $\mathsf{parity}(\mathbf{x})$ for some $\mathbf{x} \leftarrow D_{\mathbf{Q},2\sigma_{\mathsf{sign}}}$. We know $2^{-2n}, D_{\mathbf{Q},2\sigma_{\mathsf{sign}}}(2\mathbb{Z}^{2n} + \mathbf{c}) \in [1-\varepsilon, 1+\varepsilon] \cdot (\sqrt{2\pi}\sigma_{\mathsf{sign}})^{2n}/\gamma$, and so the maximum probability an output of RO is any value is $2^{-2n} \cdot (1+\varepsilon)/(1-\varepsilon)$. Therefore the probability $\mathbf{x}_i$ lies in one of these "rotational" cosets is $2^{-n} \cdot 2^{-2n} \cdot (1+\varepsilon)/(1-\varepsilon) \in \mathsf{negl}(n)$. We now turn to $\alpha\mathbf{x}_j$ for $\alpha \in \mu_\mathbb{K}$ and $j \neq i$. If $\mathsf{parity}(\alpha\mathbf{x}_j) = \mathsf{parity}(\mathbf{x}_i)$ then $\mathsf{parity}(\mathbf{x}_j) = \mathsf{parity}(\alpha'\mathbf{x}_i)$ for some $\alpha' \in \mu_\mathbb{K}$. Since parity is invariant under negation, there are therefore n cosets each $\mathbf{x}_j$ can fall into such that some rotation of it will match the parity of $\mathbf{x}_i$. This happens with probability at most $(q_s-1)\cdot n \cdot 2^{-2n} \cdot (1+\varepsilon)/(1-\varepsilon) \in \mathsf{negl}(n)$. Finally, we discarded the possibility that $\mathsf{parity}(\mathbf{x}_i) = \mathbf{0}$ when considering $\mathbf{x}_i$ such that rotations share the same parity. Call the sum of the above probabilities $\mathsf{multitarget}$.

In total, if $\mathcal{B}$ does not abort, and $\mathcal{A}$ outputs a valid forgery, then $\mathcal{B}$ makes $q_o = q_s$ queries to samp, samples at most q_h uniform strings in $\{0,1\}^{\mathsf{saltlen}}$ to simulate RO, and reprograms RO at q_s points. Except with probability sim, $\mathcal{A}$ behaves as if it were interacting with the true $\text{ROM-SUF-CMA}_{\Pi,\mathcal{A}}$ game. Except with probability abort, $\mathcal{B}$ outputs some $\mathbf{x}^\star$, and except with probability $\mathsf{multitarget}$ it is a valid solution to the $\text{SAMPLE}_{\mathsf{ac-omSVP},\mathcal{B}}$ game. Therefore we have $\delta' \geq \delta - \mathsf{abort} - \mathsf{sim} - \mathsf{multitarget}$.

B.1 Discussion

Lemma 10 tells us that if $q_s, q_h, |\mu_{\mathbb{K}}| \in \mathsf{poly}(n)$ and 2^{saltlen} grows superpolynomially in n, then $\mathcal{B}$ effectively works in the same time and succeeds with the same probability as $\mathcal{A}$.

There are however instances $\mathsf{ac{-}omSVP} = (\mathsf{Init}, \mathsf{samp})$ which are easy to solve, even without a signature forging adversary against HAWK. We consider two such situations. Firstly, if $L \geq \sqrt{2}\sigma\sqrt{2n}$ then one can sum any two samples from samp and expect to win with high probability. In Lemma 10 this would be the case were $\sigma_{\mathsf{ver}} \geq \sqrt{2}\sigma_{\mathsf{sign}}$. Secondly, if $\mathbf{Q}$ contains short vectors, then one may solve the $\mathsf{ac{-}omSVP}$ instance without ever querying samp. In Lemma 10 this would be the case if $\sigma_{\mathsf{pk}} \leq 2\sigma_{\mathsf{ver}}$. In fact, the two conditions mentioned are not just sufficient, but also (close to) necessary for these attacks to work. We will show, by a concentration bound on the lengths of the samples, that the above attacks have probability negligible in n of working whenever

$$\sigma_{\mathsf{pk}} > C \cdot 2\sigma_{\mathsf{ver}}, \text{ and } \sigma_{\mathsf{sign}} > C \cdot \sigma_{\mathsf{ver}}/\sqrt{2},$$

for any constant $C > 1$.

We note that only one of these conditions seems relevant to HAWK. First, that omSVP requires $\sigma_{\mathsf{pk}} > 2\sigma_{\mathsf{ver}}$ to be hard seems an artefact of our reduction, since we must half omSVP samples to simulate signature queries. In HAWK our verification length bound $\sigma_{\mathsf{ver}}\sqrt{2n}$ is exactly half of L and we have $\sigma_{\mathsf{pk}} > \sigma_{\mathsf{ver}}$. On the other hand, if $\sigma_{\mathsf{ver}} > \sqrt{2} \cdot \sigma_{\mathsf{sign}}$ then the sum of two signatures will often be short enough to be a signature. This is not sufficient by itself for a forgery, but it does lead to the following situation. Let $\mathsf{H}(\mathsf{m}\|\mathsf{r}) = \mathbf{h}$, $\mathbf{t} = \mathbf{B} \cdot \mathbf{h}$, and $\mathbf{x} \leftarrow D_{\mathbb{Z}^{2n}+\frac{1}{2}\mathbf{t},\sigma_{\mathsf{sign}}}$, and similarly $\mathsf{m}', \mathsf{r}', \mathbf{h}', \mathbf{t}'$ and $\mathbf{x}'$. Since it is likely $\|\mathbf{x} + \mathbf{x}'\|^2 \leq \sigma_{\mathsf{ver}}^2 \cdot 2n$ and $\mathbf{x} + \mathbf{x}' \in \mathbb{Z}^{2n} + \frac{1}{2}(\mathbf{t} + \mathbf{t}')$, $\tilde{\mathbf{h}} = \mathbf{B}^{-1}(\mathbf{t} + \mathbf{t}')$ becomes another target in the codomain of H. That is, if an adversary can find some $\tilde{\mathsf{m}}, \tilde{\mathsf{r}}$ such that $\mathsf{H}(\tilde{\mathsf{m}}, \tilde{\mathsf{r}}) = \tilde{\mathbf{h}}$ they can use $\mathbf{x} + \mathbf{x}'$ to create a signature.

Our parameter sets all satisfy $\sigma_{\mathsf{ver}} < \sqrt{2} \cdot \sigma_{\mathsf{sign}}$ by some margin, though more expensive attack strategies might yield some extra targets for which to find preimages of H. Increasing $\mathsf{saltlen}$ slighty in HAWK would overcome any weakness brought about by these techniques, and we leave their exploration to future work.

Concentration bound. We use Lemma 13 to show for large enough σ the square length of a vector sampled from $D_{\mathbb{Z}^n,\sigma}$ is concentrated from below around the mean. To prove it we require the following lemmas. The first tells us that for well chosen values of σ the mean of the squared length for the discrete Gaussian is close to that of the continuous Gaussian. The second upper bounds the Gaussian mass of the integers outside of a centred open ball.

Lemma 11 ([36, Lem. 4.3] adapted). *For Λ of full rank n, $\sigma \geq 2\eta_\varepsilon(\Lambda)$ and $\mathbf{x} \leftarrow D_{\Lambda,\sigma}$ we have*

$$\mathbb{E}\left[\|\mathbf{x}\|^2\right] \in \left[1 - \frac{2\pi\varepsilon}{1-\varepsilon}, 1 + \frac{2\pi\varepsilon}{1-\varepsilon}\right] \cdot \sigma^2 n.$$

Lemma 12 ([10, Lem. 2.4] adapted). *For $t > 0$ we have*

$$\rho_\sigma(\mathbb{Z} \setminus (-t, t)) \leq 2\exp(-t^2/2\sigma^2) \cdot \rho_\sigma(\mathbb{Z}).$$

Lemma 13. *Let $\{X_i\}_{i=1}^n$ be i.i.d. as the square of $D_{\mathbb{Z},\sigma}$ and $X = X_1 + \cdots + X_n$. There exists an $\varepsilon(n) \in \mathsf{negl}(n)$ such that if $\sigma \geq 2\eta_\varepsilon(\mathbb{Z}^n)$ then there exists some $\gamma(n)$ tending to zero such that*

$$\Pr[X < (1-\gamma)\mu_{X,l}] \in \mathsf{negl}(n).$$

For example we may take $\varepsilon(n) = n^{-\log n}$, $\sigma \in \Theta(\log n)$ and $\gamma(n) = 1/\log n$.

Proof. We make use of the following form of the Chernoff–Hoeffding bound. If $\{Y_i\}_{i=1}^n$ are i.i.d. in $[0,1]$, $Y = Y_1 + \cdots + Y_n$ and $\mu_{Y,l} \leq \mathbb{E}[Y]$ then

$$\Pr[Y < (1-\gamma)\mu_{Y,l}] \leq \exp\left(-\frac{\gamma^2}{2}\mu_{Y,l}\right),$$

see for example [15, Sec. 1.6]. Our proof strategy is to use a lower bound on the mean of X. In particular, we calculate a value t such that $\rho_\sigma(\mathbb{Z}\setminus(-t,t)) \in \mathsf{negl}(n)$ and scale $Y_i = X_i/t^2$, $Y = X/t^2$, and $\mu_{Y,l} = \mu_{X,l}/t^2$. While we do not satisfy the hypothesis for the Chernoff–Hoeffding bound that each Y_i is in $[0,1]$, by a union bound over the X_i the probability we violate it is in $\mathsf{negl}(n)$. Let $\varepsilon(n) = n^{-\log n}$ then some $\sigma \in \Theta(\log n)$ implies $\mathbb{E}[X] \geq (1 - 2\pi\varepsilon/(1-\varepsilon)) \cdot n\sigma^2$ by Lemma 11 and $\rho_\sigma(\mathbb{Z}) \leq (1+\varepsilon)\cdot\sqrt{2\pi}\sigma$ by Lemma 3. If we set $t = \sigma n^{\delta/2}$ for any $\delta > 0$ then by Lemma 12

$$\begin{aligned}\rho_\sigma(\mathbb{Z} \setminus (-t,t)) &\leq 2\exp(-n^\delta/2)\rho_\sigma(\mathbb{Z}) \\ &\leq 2(1+\varepsilon)\exp(-n^\delta/2 + \log(\sqrt{2\pi}\sigma)) \in \mathsf{negl}(n).\end{aligned}$$

We therefore have $\Pr[Y < (1-\gamma)\mu_{Y,l}] \leq \exp\left(-\frac{\gamma^2}{2}\mu_{Y,l}\right) + \mathsf{negl}(n)$, for $\mu_{Y,l} = (1 - 2\pi\varepsilon/(1-\varepsilon))n\sigma^2/t^2 = (1 - 2\pi\varepsilon/(1-\varepsilon))n^{1-\delta}$. For $\gamma = 1/\log n$ and any $\delta \in (0,1)$ this upper bound is negligible in n. By unscaling we attain the same negligible probability for X.

In particular, when the hypotheses of Lemma 13 are satisfied we know that for any constant factor $C > 1$ the probability that any of $\mathsf{poly}(d)$ samples from $D_{\mathbb{Z}^d,\sigma}$ have squared length less than $d(\sigma/C)^2$ is negligible in $d = 2n$. Let $C > 1$ be any constant. Given that the public key is constructed from such samples with $\sigma = \sigma_{\mathsf{pk}}$, it is sufficient to have $\sigma_{\mathsf{pk}} > C \cdot 2\sigma_{\mathsf{ver}}$ to protect from trivial public key solutions. For the summations we know that if $\sigma_{\mathsf{sign}} > C^2 \cdot \sigma_{\mathsf{ver}}/\sqrt{2}$, then except with negligible probability all sampled vectors have squared length at least $d(\sigma_{\mathsf{sign}}/C)^2 > d(C \cdot \sigma_{\mathsf{ver}})^2/2$. Using similar concentration bounds, and that any tuple of $k = O(1)$ such vectors are nearly orthogonal, summing them gives, except with negligible probably, an element of square length at least $k \cdot d\sigma_{\mathsf{ver}}^2/2 \geq d\sigma_{\mathsf{ver}}^2$, which is not a solution.

We have shown asymptotic security for omSVP from these attacks if the parameters are chosen as mentioned. We leave the concrete security of attacks on omSVP for future work.

C Details on the implementation

In this appendix we explain in more detail the design choices made in the implementation and include more low-level information than in Section 5.3.

HAWK has a reference implementation and an AVX2 optimised implementation. Almost all processors with AVX2 support have a floating point unit and therefore the AVX2 optimised implementation makes extensive use of built in floating point numbers and will use the FFT when a multiplication or division is needed in the number ring. On the other hand, the reference implementation uses the NTT for multiplication in the number ring and only uses an FFT with fixed point precision for decompressing signatures in the verification step. As floating points are required in TowerSolve$\mathsf{I}$ during key generation, the reference implementation emulates floating points (IEEE-754 "binary64" format) in key generation, and is entirely free of them in signing and verification.

The ordering of the roots in the FFT representation is chosen as in FALCON, the "bit reversal" ordering, and only half the embeddings are stored since the other half are their complex conjugates. Moreover, of these n fixed-point or floating point numbers, the first $n/2$ are the real parts and the last $n/2$ are the imaginary parts, giving the performance benefits mentioned at the end of [46, Sec. 4.2.1]. The advantage of this ordering is that self-adjoint polynomials, like q_{00} and q_{11}, only have real embeddings and thus require only half the memory. An implementer is free to choose their own ordering as long as it is consistent within the implementation.

C.1 Key generation

Key generation is the same for both implementations, as the majority of the work is the TowerSolve$\mathsf{I}$ procedure. This requires floating points, so in the reference implementation we emulate them. In [42, Sec. 6], the problem of having a vector reduction in TowerSolve$\mathsf{I}$ using fixed point arithmetic was left open. If one is able to solve this problem, one can remove all floating point emulation from the reference implementation of HAWK.

In key generation, f, g are sampled in R with coefficients from $D_{\mathbb{Z},\sigma_{\mathsf{pk}}}$. This sampling is performed using a precomputed cumulative density table with 63 bits of precision and produces values of absolute value at most 13 for HAWK-512 and at most 18 for HAWK-1024. The kth index of the table (with $k = 0, 1, \ldots, 12$ for HAWK-512) contains the integer $\left\lfloor 2^{63} \cdot \Pr_{X \leftarrow D_{\mathbb{Z},\sigma_{\mathsf{pk}}}}[|X| \geq k+1] \right\rfloor$ so the statistical distance between $D_{\mathbb{Z},\sigma_{\mathsf{pk}}}$ and the implemented sampling procedure is smaller than 2^{-64}.

We then check five conditions.

1. $\mathrm{N}(f)$ and $\mathrm{N}(g)$ are both odd, i.e. both the sum of coefficients of f and the sum of coefficients of g are odd,
2. $\|(f, g)\|^2 > \sigma_{\mathsf{sec}}^2 \cdot 2n$,
3. $f(X)$ is invertible in $\mathbb{Z}[X]/(X^n + 1, 2^{16} + 1)$, i.e. the NTT of f modulo $2^{16} + 1$ has no entry equal to zero,

4. $ff^* + gg^*$ is invertible in $\mathbb{Z}[X]/(X^n + 1, 2147473409)$, i.e. the NTT of q_{00} modulo 2147473409 has no entry equal to zero,
5. $\langle (1, q_{00}^{-1}) \rangle < \nu_{\mathsf{dec}}$.

The primes $2^{16} + 1$ and 2147473409 are both $1 \pmod{2n}$ and will be motivated in sign and verify respectively. The conditions in 3. and 4. are required only in the reference implementation. However, we check these also in the AVX2 implementation as a signer should be able to switch to the reference implementation without changing the key pair and a verifier should be able to use the reference implementation.

If any of the checks fail, we reject this sample and restart key generation. We remark that in FALCON if the sum of coefficients of e.g. $\mathrm{N}(f)$ is even, then the final coefficient is resampled until its parity changes. As this slightly alters the key distribution we choose caution and resample both f and g completely.

The time to find a valid pair (f, g) is much smaller than the time to run TowerSolveI. The implementation of the TowerSolveI algorithm is taken from FALCON setting $q = 1$. Since the standard deviation of the coefficients of f and g is smaller for HAWK, the upper bound on the number of bits required to represent intermediate polynomials in the tower of fields is a little smaller. Moreover, in HAWK the final reduction of (F, G) in the top field is replaced by ffNP [20, Alg. 4], written in a memory friendly manner where certain polynomials overwrite memory that is no longer in use. This modification increases the public key size, whereas in Falcon this would only give a slightly better trapdoor sampler.

The TowerSolveI algorithm fails if $\mathrm{N}(f)$ and $\mathrm{N}(g)$ are not coprime over $\mathbb{Z}$, or the returned F, G have a coefficient with absolute value larger than 128, or F, G do not satisfy the sanity check $fG - gF = 1 \pmod{2147473409}$ which is rather unlikely. The sanity check catches rare cases where the intermediate coefficients of normed down f, g, F or G may be larger than is supported (see "Coefficient Sizes" [46, Sec. 4.4.3]). If TowerSolveI fails, we restart key generation. All these checks are also in the reference implementation of FALCON.

Otherwise, the public key is computed, and the program checks if the encoded public key is not larger than the allowed maximum size. The encoded public key first contains a header byte where the four most significant bits are set to zero and the remaining four bits contain the value $\log_2(n)$ for the n used in KGen (512 or 1024). After the header byte follows the used encoding of q_{00} and q_{01}.

Experimentally the average public key size is 1006.5 bytes (including its header byte) with a standard deviation of 6.1. To be able to reserve memory in advance for encoding a public key, we enforce the encoding of a public key to be of size at most 1043 bytes in HAWK-512, which is 6 standard deviations above the average, altering the key distribution only a little. In HAWK-1024 an average public key is of size 2329.2 bytes with a standard deviation of 11.2 and therefore the encoding of a public key can be at most 2396 bytes long. These averages were obtained by generating 10^5 keys.

Moreover, given the small σ_{pk}, we know the constant coefficient of q_{00} fits in a `int16_t`. It is checked for all the other coefficients of the quadratic form if these lie in a certain range. For example, in HAWK-512 the absolute values must

be (strictly) smaller than $2^9, 2^{12}$ and 2^{15} for q_{00}, q_{01} and q_{11} respectively. These bounds provide handles in verification when these elements are multiplied by a signature.

C.2 Signature generation

A message is hashed to some element $h \in \{0,1\}^{2n}$ using SHAKE256 on the bitstring $m\|r$, with $r \in \{0,1\}^{320}$ a random salt. Whenever a check fails in signing, a new random salt is generated. However, the message is not hashed again as it is enough to recover the internal state of SHAKE256 after consuming the message. This approach is only possible as we chose to have the salt last.

First, there is a check if a signature is short enough, which is line 5 in Algorithm 2. Moreover, it is checked that all coefficients of s_1 are not too large and the encoded signature cannot be too large as well, similar to the encoded public key.

In signing the two implementations differ: the AVX2 version uses double precision floating point numbers inside the FFT while the reference implementation uses NTT with prime $p = 2^{16} + 1$.

AVX2. During AVX2 signing we transform the whole basis $\mathbf{B}$ into FFT representation. Since G is not stored in the secret key, $G = (1 + gF)/f$ is calculated in FFT representation after f, g, F and 1 are put into FFT representation. Note that the Fourier transform of 1 is a vector with a one in every entry. Then, the hash $\mathbf{h} = (h_0, h_1)^\mathsf{T} \in R^2$ is also put into FFT representation after which we calculate $\mathbf{t} = \mathbf{B} \cdot \mathbf{h}$ in FFT representation and then invert the FFT. From this, the point $\mathbf{x} = (x_0, x_1)^\mathsf{T} \leftarrow D_{2\mathbb{Z}^{2n}+\mathbf{t},2\sigma_{\mathsf{sign}}}$ is sampled. This sampled point then overwrites the initial target and is converted to FFT representation. With the basis still in FFT representation we can easily compute the second component of $\mathbf{B}^{-1}\mathbf{x}$ by computing $(-g)x_0 + fx_1$. Now the signature is computed by

$$s_1 = \frac{h_1 - ((-g)x_0 + fx_1)}{2}.$$

Note here that h_1 does not need to be converted to FFT representation. In addition, there is a compilation flag `-DHAWK_RECOVER_CHECK`, which checks if a vector $\mathbf{x}$ satisfies (2).

In batched signing, the basis $\mathbf{B}$ is precomputed in FFT representation and therefore cannot be altered. If the compilation flag `-DHAWK_RECOVER_CHECK` is added, the expanded secret key also contains $1/(f^*f+g^*g)$ in FFT representation to make the check faster.

Note that the target $\mathbf{B} \cdot \mathbf{h}$ only needs to be known modulo 2 to sample $\mathbf{x}$. However, since f and g need to be in FFT representation to compute s_1 with $\mathbf{B}^{-1}$, as far as we know the best approach is to compute $\mathbf{Bh}$ exactly and then reduce it modulo 2. Having a fast multiplication in the ring $\mathbb{Z}[X]/(2, X^n + 1) = \mathbb{Z}[X]/(2, (X + 1)^n)$ may result in a faster signing algorithm and would reduce the memory usage of the current implementation, making this interesting future work.

Reference implementation. In the reference implementation we use the prime $p = 2^{16} + 1$ which is 1 (mod 2048). The coefficients in $\mathbf{B} \cdot \mathbf{h}$ are distributed around zero with a standard deviation of less than 400 for Hawk-1024. Since the prime used in signing is much larger than this standard deviation, the inverse NTT provides the correct lifting from $\mathbb{Z}/p\mathbb{Z}$ to $\mathbb{Z}$. Using the special property of the Fermat prime $2^{16} + 1$, multiplication is done using $2^{16} \equiv -1 \pmod{2^{16} + 1}$. Inverting an element $x \in \mathbb{Z}/p\mathbb{Z}$ is done by calculating $x^{p-2} \pmod{p}$, and can be done for this prime with an addition chain requiring 19 multiplications, which is faster than the naïve exponentiation by squaring. This inverse is only used to reconstruct G using $G = (1 + gF)/f$, and since the third condition in KGen holds, f is invertible modulo p.

One can reduce the memory usage of signing from 15kB down to 8kB for Hawk-512 by using a prime $p < 2^{16}$ for example $p = 12289$ or $p = 18433$ as then values can be stored in 16 bits rather than 32 bits. Here to have constant-time multiplications, one can use Montgomery modular reduction [39] with radix 2^{16}, which also has better performance than the built in modulo operator. Interestingly, there are addition chains for $p = 12289$ and $p = 18433$ both requiring 18 multiplications, which is much less than the 26 needed for exponentiation by squaring. Since a division is as costly for $p = 12289$ as for $p = 18433$, it is preferable to use the latter prime as larger integers are supported without incorrect lifting.

The NTT version of signing is essentially obtained by replacing all FFT invocations with NTT invocations. One of the few differences is that the NTT version calculates s_1 by

$$s_1 = (-g, f) \cdot \frac{\mathbf{t} - \mathbf{x}}{2},$$

where the division by two is performed in $\mathbb{Z}$ before passing to the NTT representation. Here, s_1 is calculated correctly with the NTT when the absolute value of the coefficients of the actual s_1 are all smaller than $p/2$, as otherwise an incorrect lift from $\mathbb{Z}/p\mathbb{Z}$ to $\mathbb{Z}$ may be chosen when inverting the NTT. If the equivalent equation from the AVX2 implementation is used, these coefficients must be bounded by $p/4$ in absolute value.

In the reference implementation, one cannot supply the compilation flag `-DHAWK_RECOVER_CHECK` to perform the recovery check as in (2), because this check can only be done with little overhead when basis and $\mathbf{x}$ are already in FFT representation. To still have this check without the use of emulated floating point numbers, one could simply run the verification algorithm inside signing, although this check almost never fails, see Section 5.1.

Sampling in signing. Sampling a single coefficient from $D_{2\mathbb{Z}+c,2\sigma_{\mathsf{sign}}}$ uses two precomputed cumulative density tables, one for each center $c \in \{0, 1\}$. To have the sampler run in constant time, independent of the centre c and the outcome, the program reads the two cumulative density tables fully. Algorithm 7 explains the sampler in pseudocode, where $[\![P]\!]$ is 1 when a proposition P holds and zero

otherwise. The algorithm uses a table where RCDT[c][i] contains the value

$$\left\lceil 2^{78} \cdot \Pr_{X \leftarrow D_{2\mathbb{Z}+c,2\sigma_{\mathsf{sign}}}} [\,|X| \geq 2i+2\,] \right\rceil .$$

As computers do not have built in support for 78 bit numbers, the tabulated values are split in the implementation into a high part of type `uint16_t` and a low part of type `uint64_t`. The high part contains 15 bits and the low part 63 bits. We get faster constant-time code by not having the highest bit set in both parts, because checking $a < b$ can be done by looking at the sign bit of $a - b$ for numbers a, b not having their highest bit set, making comparisons easier and more efficient. Moreover, after the first 5 entries in the table, the high parts are all zero and are thus omitted from the table to save memory. More importantly this makes the comparison of 78 bit numbers easier for $i \in \{5, \ldots, 12\}$.

Algorithm 7 Sampling from $2\mathbb{Z}+c$ with standard deviation $2\,\sigma_{\mathsf{sign}}$: SamplerZ($c$)

Require: RCDT having reverse cumulative density tables for support $2\mathbb{Z} + c$ with $c \in \{0, 1\}$
Ensure: A sample taken from a distribution close to $D_{2\mathbb{Z}+c,2\sigma_{\mathsf{sign}}}$
1: $u \leftarrow$ UniformBits(78)
2: $z_0 \leftarrow 0$
3: **for** i = 0, ..., 12 **do**
4: $\quad z_0 \leftarrow z_0 + [\![u < (1-c) \cdot \mathtt{RCDT[0][i]} + c \cdot \mathtt{RCDT[1][i]}]\!]$
5: $z_0 \leftarrow 2 \cdot z_0 + c$
6: $z_0 \leftarrow z_0 - 2z_0 \cdot$ UniformBits(1)
7: **return** z_0

C.3 Signature verification

The implementation for signature verification follows Algorithm 3 closely, but computes the norm with respect to $\mathbf{Q}$ in an efficient way.

AVX2. For the AVX2 version, one first computes q_{11} via the FFT and the equation $q_{11} = (1 + q_{10}q_{01})/q_{00}$, making use of the fact that numerator and denominator are both self-adjoint and thus saving some multiplications.

In the recovery of s_0 using (1), note that h_0 does not need to be in FFT representation. Moreover, $t_1 = h_1 - 2s_1$ is calculated in FFT representation, and after recovering s_0 one sets $t_0 = h_0 - 2s_0$ in FFT representation, after which we check $\left\|(t_0, t_1)^{\mathsf{T}}\right\|_{\mathbf{Q}}^2 \leq 8n \cdot \sigma_{\mathsf{ver}}^2$. Thus in total there are four FFT calls for t_0, t_1, q_{00} and q_{01} and an inverse FFT call on $t_1 \cdot q_{01}/q_{00}$ needed to calculate s_0.

For computing the norm of $\mathbf{t}$ with respect to $\mathbf{Q}$, we use $\|\mathbf{t}\|_{\mathbf{Q}}^2 = \frac{1}{n}\mathrm{Tr}(\mathbf{t}^*\mathbf{Q}\mathbf{t})$ as for a ring element x we have $\mathrm{Tr}(x) = \sum_\sigma \sigma(x)$, where σ ranges over all the field

embeddings of $\mathbb{Q}(\zeta_{2n})$ into $\mathbb{C}$. Recall that the FFT representation of x holds all the $\sigma(x)$ up to complex conjugation, making the trace easily computable once x is in FFT representation. Moreover, the norm is a real number so we only need the real part of $\mathbf{t}^*\mathbf{Qt}$ under the embeddings. For example, we have

$$\left\| \begin{pmatrix} t_0 \\ 0 \end{pmatrix} \right\|_{\mathbf{Q}}^2 = \frac{1}{n} \sum_{\sigma} \sigma\left(t_0^* q_{00} t_0\right) = \frac{1}{n} \sum_{\sigma} \sigma(q_{00}) |\sigma(t_0)|^2,$$

where the contribution of some σ is equal to that of its conjugate embedding. A benefit of calculating the geometric norm using t_0, t_1, q_{00}, q_{01} and q_{11} in FFT representation is that it requires no extra memory. When checking if the norm does not exceed the norm bound, first we check if the outcome fits in a `int32_t` and is positive. It is then rounded to an integer before being compared to the norm bound.

Reference implementation. In the reference implementation, we still recover s_0 using the FFT together with (1) but now using fixed-point arithmetic instead of floating point arithmetic. As there are bounds on coefficients of s_1 and q_{00}, q_{01} from signing and key generation respectively, it is possible to determine bounds on the absolute value of the numbers for all layers inside the FFT. The constant coefficient of q_{00} has a different bound to the other coefficients of q_{00}, so this one is excluded via setting it to zero before transforming q_{00} to the FFT representation. The FFT representation of the constant coefficient of q_{00} is a constant valued vector and can thus be easily added afterwards.

Initially we scale up the coefficients of s_1, q_{00} and q_{01} by a power of two such that they require at most 29 bits (excluding the sign bit at the beginning). Then, after each layer of computations within the FFT, all numbers are divided by two, dropping the least significant bit after the fixed point. It is then readily proven that no overflow happens during the FFT computation.

Although the computation here uses less precision than the AVX2 version, the precision errors before rounding in (1) are so small that rounding to an incorrect s_0 is highly unlikely. By the parameters in Section 5.1, it is extremely unlikely (probability less than 2^{-105}) that s_0 is recovered incorrectly with (1). By small adaptations to this analysis, it can be shown that heuristically for a fixed key pair in Hawk-512 with probability at least $1-2^{-100}$, all the coefficients in (1) lie in $\mathbb{Z} + (-0.49, 0.49)$ before being rounded to the nearest integer. This illustrates that the fixed-point computation can handle even "large" precision errors of 0.01 (which are rare) as such almost never affect the recovery of s_0.

Experimentally, out of $5 \cdot 10^8$ valid generated signatures, where we generate 100 signatures for a key pair before sampling a new one, none failed with this fixed-point FFT recovery of s_0 both for Hawk-512 and Hawk-1024.

For the norm calculation, q_{11} needs to be calculated from q_{00} and q_{01} using the NTT with the 31 bit prime. Here, the inverse of q_{00} is needed. We use the extended Euclidean algorithm for this, since verification is trivially isochronous, i.e. it operates only on public information and so there is no requirement for it

to be constant-time. The extended Euclidean algorithm has better performance than exponentiation.

For calculating the norm modulo a prime p, we exploit a property of the NTT, namely that $\mathrm{Tr}(x)$, the sum of FFT coefficients of x, is congruent to the sum of NTT coefficients of x modulo p for any element $x \in R$. The verification calculates the norm modulo two primes p_1, p_2, say n_1, n_2 with $n_i \in \{0, 1, \ldots, p_i - 1\}$ and then accepts the signature when $n_1 = n_2$ and n_1 is smaller than the norm bound. By theoretically determining the maximum norm possible with bounds on the signature coefficients and the public key coefficients, if this norm is known to be smaller than $p_1 \cdot p_2$ we know by the Chinese remainder theorem that no wrap around has happened, so no invalid signatures will be accepted.

First, denote the infinity norm of $f = f_0 + \cdots + f_{n-1}X^{n-1} \in R$ by

$$\left\|f_0 + f_1 X + \cdots + f_{n-1}X^{n-1}\right\|_\infty = \max_{0 \le i < n} \|f_i\|,$$

and write $\mathbf{t} = \mathbf{h} - 2\mathbf{s}$.

For HAWK-1024, the non-constant coefficients of q_{00} and q_{11} are at most 2^{10} and 2^{17} in absolute value respectively, while $\|q_{01}\|_\infty \le 2^{14}$. The encoding of signatures requires $\|s_1\|_\infty < 2^{10}$. In addition, for bounding the norm, we require that after recovering s_0, we must have $\|s_0\|_\infty < 2^{14}$. Thus, $\|t_0\|_\infty < 2^{15}$ and $\|t_1\|_\infty < 2^{11}$.

We will now use that for polynomials $a(X), b(X) \in \mathbb{Z}[X]/(X^n + 1)$, we have $\|a(X) \cdot b(X)\|_\infty \le \|a(X)\|_\infty \cdot \|b(X)\|_\infty \cdot n$. However, to use this bound, we handle the contribution of the constant terms of q_{00} and q_{11} separately as these are larger than the bounds mentioned above. First, KGen required that $\|f\|_\infty, \|g\|_\infty < 128$ and therefore, $\langle 1, q_{11}\rangle < 2n \cdot 128^2 = 2^{25}$. On the other hand, $\langle 1, q_{00}\rangle / \sigma_{\mathsf{pk}}^2$ behaves as a χ^2-distribution with $2n$ degrees of freedom, so by setting $D = 2n$ and $x = D/4$ in [32, (4.3)], the probability that this quantity is larger than $5n$ is at most $\exp(-n/2)$, which is miniscule for $n = 512$ and $n = 1024$. Therefore, we can safely say $\langle 1, q_{00}\rangle < 5n\sigma_{\mathsf{pk}}^2 < 2^{15}$. Now combining this with the bound on s_0 and s_1, the constant terms give a contribution of at most

$$2^{15} \cdot n \cdot \|t_0\|_\infty^2 + 2^{25} \cdot n \cdot \|t_1\|_\infty^2 \le n \cdot (2^{15+2\cdot 15} + 2^{25+2\cdot 11}) < 2^{58}.$$

These bounds on $\mathbf{Q}$ and $\mathbf{s}$ then imply that,

$$\begin{aligned}\|\mathbf{h} - 2\mathbf{s}\|_{\mathbf{Q}}^2 = \|\mathbf{t}\|_{\mathbf{Q}}^2 &\le n^2 \cdot \left(2^{2\cdot 15+10} + 2 \cdot 2^{15+11+14} + 2^{2\cdot 11+17}\right) + 2^{58} \\ &\le 2^{20}\left(2^{40} + 2^{41} + 2^{39}\right) + 2^{58} = 15 \cdot 2^{58}.\end{aligned}$$

Now given primes $p_1 = 2147473409$ and $p_2 = 2147389441$, one can see that $15 \cdot 2^{58} < p_1 p_2$. Of course, this also shows that HAWK-512 can calculate the norm with these two primes, as the bounds on all numbers are smaller. In both cases, this shows that no invalid signatures will get accepted as the geometric norm cannot be above $p_1 p_2$ for a signature that passes this NTT version of verification. On the other hand, the extra restriction on $\|s_0\|_\infty$ does reject some valid signatures, but this can be shown to happen with miniscule probability.

Namely, the coefficients for recovered s_0, averaged over key pairs and valid signatures s_1, follow roughly a Gaussian distribution with $\sigma \approx 354$ and $\sigma \approx 962$ for Hawk-512 and Hawk-1024 respectively. Hence, a simple union bound yields $\Pr[\|s_0\|_\infty \geq 2^{13}] < 2^{-382}$ for Hawk-512 and $\Pr[\|s_0\|_\infty \geq 2^{14}] < 2^{-203}$ for Hawk-1024.

Moreover, we choose to work with 31 bit primes as the (Montgomery) modular multiplication requires 64 bit numbers and if one takes a prime of e.g. 62 bits, this requires working with 128 bit integers which is not supported in the C language.

C.4 Encoding

Recently, Espitau et al. showed in [24] one can improve Falcon's signatures sizes by 7%-14% using asymmetric numeral systems (ANS) encoding ([22]) instead of Golomb–Rice encoding. We have tried encoding/decoding public keys in Hawk using the open-source rANS implementation at `https://github.com/rygorous/ryg_rans` and this showed a saving of 15 and 30 bytes for Hawk-512 and Hawk-1024 respectively. Because this saves less than 2% on the public key sizes but does add signficantly to the code complexity, we did not use this encoding in the final version.

D On the Falcon security estimates

In this appendix we describe and explain the security estimates for Falcon-512 and Falcon-1024 given in [46]. We go on to estimate the security of Hawk parameters using the Falcon methodology and discuss how part of this methodology is *over*conservative. One high level difference is that Falcon does not use state of the art BKZ simulation in its security estimates, as opposed to Hawk's use of [14]. Instead it uses different heuristics that describe the Gram–Schmidt lengths (resp. first basis vector length) of a basis after DBKZ [37] reduction for key recovery (resp. signature forgery). The structure of DBKZ allows for a cleaner theoretical proof of worst case bounds on e.g. the length of the first basis vector after DBKZ-β reduction, which are better than the best known for (standard) BKZ [34]. However, models of the average case behaviour of the two algorithms predict the same performance.

Falcon *key recovery.* Broadly speaking, Falcon estimates the cost of key recovery using [6, Sec. 6.3], which is hiding behind (4). Following the notation of [46, Sec. 2.5.1], their estimate for the left hand side is $\sqrt{B}\sigma_{\{f,g\}}$, i.e. an estimate for the length of f or g projected into the final projected sublattice of rank B. They then consider using DBKZ reduction and appeal to [37, Cor. 2] for its average case behaviour to obtain their value for their right hand side of (4). The factor of $\sqrt{4/3}$ will be discussed below. There are several things to note.

First, it appears there is a small mistake in [37, Cor. 2]. Indeed, in the notation of [37, Sec. 4], when replacing the condition of being less than or equal to $\sqrt{\gamma_k}$ in

(5) of their dynamical analysis, rather than taking $\sqrt{\mathrm{GH}(k)}$ they should choose $\mathrm{GH}(k)$. This is the difference between setting $\alpha = \frac{1}{2}\mathrm{GH}(k)$ and $\alpha = \mathrm{GH}(k)$ in [37, Cor. 2]. The effect of this change is

$$\|\mathbf{b}_i^*\| = \mathrm{GH}(k)^{\frac{n+1-2i}{k-1}} \mathrm{vol}(\Lambda)^{1/d},$$

that is, the factor of two in the denominator of the exponent disappears. As a sanity check, the $i = 1$ case now follows [37, Cor. 1]. It seems that the security estimates of FALCON implicitly make this correction.

Second, FALCON approximates $\mathrm{GH}(k)$ as $\sqrt{k/2\pi e}$ and also approximates its exponent to simplify presentation. Indeed, with FALCON's implicit correction and in their notation (where n becomes $d = 2n$), the exponent is $(2n+1-2i)/(B-1)$. The first index of the final DBKZ-B block is $i = d - B + 1$ when counting from 1, and therefore to use [37, Cor. 2] we must appeal to the discussion after it that allows $i \geq B$ and also ensure $B \leq (d+1)/2$ to make sure this is indeed the case. Then the exponent $(2B - 2n + 1)/(B - 1)$ is (conservatively) approximated as the larger $2(B - n)/B$. We finally arrive at

$$\left(\sqrt{B/2\pi e}\right)^{\frac{2(B-n)}{B}} = (B/2\pi e)^{1-n/B}.$$

Third, and most importantly, an early and targetted version of the "dimensions for free" technique [16] is used within the [6, Sec. 6.3] methodology. Indeed, consider the factor of $\sqrt{4/3}$ in the inequality $\sqrt{B}\sigma_{\{f,g\}} \leq \sqrt{4/3}\lambda$ of [46, Sec. 2.5.1]. This is precisely the condition [16, (3)]. The terms $\mathrm{gh}(\mathcal{L})$ and $\sqrt{4/3} \cdot \mathrm{gh}(\mathcal{L}_d)$ in [16, (3)] relate to $\sqrt{d}\sigma_{\{f,g\}}$ and $\sqrt{4/3}\lambda$ in [46, Sec. 2.5.1]. Since we are in a terminal block of size B we have that the Gram–Schmidt vector is the shortest vector in the given projected sublattice, which we expect to have length determined by the appropriate Gaussian heuristic. Note that we have $n = d$ and $d = n - B$ with the left hand side being the notation as in [16] and the right hand side as in [46]. Hence the $\sqrt{(n-d)/n} \cdot \mathrm{gh}(\mathcal{L})$ equaling $\sqrt{B}\sigma_{\{f,g\}}$ for FALCON and $\mathcal{L}_d$ being the correct projected sublattice to consider. The sentence "This is because all remaining Gram–Schmidt norms are larger..." alludes to ensuring [16, (4)]. As this technique is present in the FALCON v1.0 documentation [45], it appears to be a concurrent discovery to [16].

A note on "dimensions for free". For simplicity we chose to report the estimated BKZ blocksizes required to either recover the key or forge a signature without applying the dimensions for free techniques [16]. One is then free to take this "raw" BKZ blocksize and convert it into a cost as is required. In FALCON's key recovery methodology they not only use dimensions for free ideas within the methodology of [6, Sec. 6.3], as explained above, but also then apply directly the dimensions for free techniques of [16] to their estimated blocksize, i.e. they also apply dimensions for free within lattice reduction. We argue here that this is overconservative.

Intuitively, the dimensions for free technique works by sieving in a projected sublattice (of a smaller rank) than the lattice you wish to find a short vector in.

When the sieve terminates it contains exponentially many (in the smaller rank) short vectors in the projected sublattice. These vectors are then lifted to the full lattice and the shortest returned. If the basis of the full lattice has sufficiently good properties, e.g. after BKZ-β reduction for some well chosen β, and the rank of the projected sublattice is not too small, then we expect one of these lifted vectors to be the shortest vector in the full lattice.

In particular, if Λ is our lattice of rank d, and Λ_i our projected sublattice of rank $d-i$, we assume that on termination our sieve contains $\{\mathbf{v} \in \Lambda_i \setminus \{\mathbf{0}\} \colon \|\mathbf{v}\| \leq \sqrt{4/3} \cdot \mathrm{gh}(\Lambda_i)\}$. Under the Gaussian heuristic we expect this set to contain approximately $(4/3)^{(d-i)/2}$ vectors. On the other hand, a sieve that is run on Λ is assumed to contain $\{\mathbf{v} \in \Lambda \setminus \{\mathbf{0}\} \colon \|\mathbf{v}\| \leq \sqrt{4/3} \cdot \mathrm{gh}(\Lambda)\}$ when it terminates, a set of expected size $(4/3)^{d/2}$. Immediately we see a problem: the bespoke dimensions for free idea used in Falcon's key recovery, as explained in the "Falcon key recovery" paragraph above, requires *all* vectors in the final projected sublattice of rank B of length less than or equal to $\sqrt{4/3} \cdot \left\|\mathbf{b}^*_{d-B+1}\right\|$ to be in the sieve database. However, if we use the general dimensions for free technique of [16] within lattice reduction, in this final block we only sieve in some further projected sublattice of a smaller rank and lift to it. This means that, regardless of how long or short the lifted vectors are, there are simply *not enough* of them.

More specifically, following [46, Sec. 2.5.1] we set $i \in \Theta(B/\log B)$ so that the rank of the projected sublattice that is sieved in is of the order $B(1 - 1/\log B)$. The sieve database after termination therefore represents a

$$(4/3)^{(B(1-1/\log B)-B)/2} = (4/3)^{-B/2\log B}$$

fraction of the required number of vectors. It may be possible to use ideas for solving LWE from [5, Sec. 6] in this instance. Here one final sieve is performed on a terminal projected sublattice of a larger rank than is used for lattice reduction, but it would require any security estimation to cost at least one call to an SVP oracle on a lattice of rank larger than $B(1 - 1/\log B)$.

Signature forgeries. To estimate the cost of forging signature on a message Falcon examines the cost of sufficient lattice reduction on an embedding lattice. However, their choice of embedding parameter $K = \sqrt{q}$ makes this approach the same as assuming that approximate CVP is at least as hard as approximate SVP with the same approximation factor on the same lattice, as we do. The difference is that for Hawk we use BKZ simulation to determine the required blocksize to solve this approximate SVP, whereas Falcon again appeals to [37, Cor. 2]. Specifically, they set $i = 1$, approximate $\mathrm{GH}(B)$ as $\sqrt{B/2\pi e}$ and perform a simplification of the exponent. Indeed, $(2n-1)/(B-1)$ is (conservatively) simplified to the smaller $2n/B$ and we obtain

$$\begin{aligned}\left(\sqrt{B/2\pi e}\right)^{(2n-1)/(B-1)} \cdot (q^n \cdot K)^{1/(2n+1)} &\approx \left(\sqrt{B/2\pi e}\right)^{2n/B} \cdot \sqrt{q} \\ &= (B/2\pi e)^{n/B} \cdot \sqrt{q} \\ &\leq \beta\end{aligned}$$

as our condition. Note that here β is a bound on the norm of a signature and not a blocksize for lattice reduction algorithms.

HAWK *and* FALCON *compared.* In this section we apply the FALCON security methodology to HAWK to be able to give a comparison in equal terms. We first describe what we expect by phrasing the problem of key recovery as unusual SVP, and the problem of signature forgery as approximate SVP.

Firstly, for key recovery the approach of [6, Sec. 6.3] requires less lattice reduction effort the shorter the projections of the unusually short vector, i.e. the left hand side of (4), are. As a more general principal, the shorter, i.e. more unusual, a lattice vector is compared to what we may expect from a random lattice using either a heuristic such as the Gaussian heuristic, or worst case bounds such as Minkowski, the easier it is to find with lattice reduction. This conflicts slightly with the idea that finding short vectors in lattices is hard – this is true provided a given vector is not much shorter than all others. To understand how HAWK and FALCON compare under any security estimates that make use of the fact that their lattices contain vectors shorter than expected, we should compare how much shorter than expected they are in either case. For HAWK we have vectors of length 1 and expect vectors of length $\mathrm{gh}(\mathbb{Z}^d)$. For FALCON we have vectors of expected length $\sqrt{d} \cdot \sigma_{\{f,g\}}$ and expect vectors of length $\mathrm{gh}(\mathbb{Z}^d) \cdot \sqrt{q}$. Given that $\sigma_{\{f,g\}} = 1.17 \cdot \sqrt{q/d}$ for FALCON, we see that the short vectors of HAWK are a factor of $(\sqrt{d} \cdot \sigma_{\{f,g\}}/\mathrm{gh}(\mathbb{Z}^d) \cdot \sqrt{q})/(1/\mathrm{gh}(\mathbb{Z}^d)) = 1.17$ times more 'unusual' than those of FALCON. This suggests that key recovery for HAWK should be slightly easier than in FALCON.

Signature forgery tells the opposite story. Recall that we are assuming the cost of solving approximate CVP with some approximation factor is at least the cost of solving approximate SVP with the same approximation factor over the same lattice, and that for the particular choice of embedding parameter K, FALCON's security methodology is the same. Here we are looking for lattice vectors that are longer than what we expect to be the shortest, and the difficulty of doing so grows as their lengths get closer to that of the shortest vector. Note that below we consider Hermite SVP, where the approximation factor is relative to the normalised volume of the lattice, and not to the actual shortest vector. In HAWK we have $\mathrm{vol}(\mathbb{Z}^d)^{1/d} = 1$, and search for a vector of length $\sqrt{d}\sigma_{\mathsf{ver}}$. That is the approximation factor is $\sqrt{d}\sigma_{\mathsf{ver}}$. In FALCON we have $\mathrm{vol}(\Lambda)^{1/d} = \sqrt{q}$ and search for vectors of length $\beta = 1.1\sqrt{d} \cdot \sigma$. That is, the approximation factor is $1.1\sqrt{d}\sigma/\sqrt{q}$. The FALCON condition is a factor of $1.1\sigma/\sqrt{q}\sigma_{\mathsf{ver}}$ times looser, which is approximately 1.15 and 1.06 for FALCON-512 and FALCON-1024 respectively. As such we expect it to be slightly easier to forge signatures for FALCON.

In Table 4 we report the estimated blocksizes for key recovery and signature forgery against HAWK-$\{512, 1024\}$ and FALCON-$\{512, 1024\}$ using the FALCON methodology. For key recovery we report the blocksizes both with the factor of $\sqrt{4/3}$ found in [46, Sec. 2.5.1], and then without it. With it, we argued above that it is overconservative to further apply dimensions for free to the lattice reduction, whereas without it we choose to do so.

Parameters	β_{key} $(\sqrt{4/3})$	β_{key} (d4f)	β_{forge}	β_{forge} (d4f)
Hawk-512	436	416	434	395
Falcon-512	458	440	411	374
Hawk-1024	898	866	970	901
Falcon-1024	936	904	952	884

Table 4: Estimating the security of Hawk and Falcon using the Falcon methodology. Here β_{key} followed by $(\sqrt{4/3})$ denotes Falcon's approach of using dimensions for free techniques in the final block during key recovery. On the other hand, (d4f) denotes using dimensions for free techniques *only* within lattice reduction [16]. Note that β_{key} $(\sqrt{4/3})$, β_{forge} and β_{forge} (d4f) match the table in [46, Sec. 2.5.1]. Only β_{key} (d4f) differs, and is higher, since we argue above the "double" application of dimensions for free techniques is overconservative. See function `falcon_blocksizes` at `https://github.com/ludopulles/hawk-aux/blob/main/code/find_params.sage`.

As we can see, in each case key recovery is a little easier for Hawk and signature forgery a little easier for Falcon. In particular, since signature forgery is easiest for both Hawk-512 and Falcon-512, under the Falcon security methodology Hawk-512 is slightly more secure than Falcon-512 and vice versa for Hawk-1024 and Falcon-1024. Our arguments regarding the double application of dimensions for free techniques in key recovery attacks do not change the security of Falcon-512, since for these parameters forgeries are estimated to be easier both in Table 4 and in the table of [46, Sec. 2.5.1]. However, in Falcon-1024 we find that the security increases slightly compared to the table of [46, Sec. 2.5.1].