Lattice cryptography: isomorphisms & dual attacks

Pulles, L.N.

Part III
Lattice Isomorphism Problem

Chapter 5

Hawk Signature Scheme

This chapter is based on joint work with L. Ducas, E.W. Postlethwaite and W.P.J. van Woerden published at ASIACRYPT 2022 [DPPvW22a], and the third round specification document of Hawk in NIST's standardisation process of additional digital signature schemes [BBD+26].

This chapter describes the signature scheme Hawk in detail, of which security is based on a variant of the Lattice Isomorphism Problem (LIP). Hawk uses modules of rank 2 over a cyclotomic number field of order a power of 2, which allows for efficient computations. The best known attacks that can break Hawk perform not much better than attacks that forget the module structure.

The specific module lattice makes sampling extremely easy: the distributions can be precomputed and stored compactly, and floating-point numbers are not needed. These design features are desirable for constrained devices, in which it can be challenging to use FN-DSA, a.k.a. Falcon [PFH+22].

Hawk's implementation is much faster than Falcon, and its signatures are a bit more compact than those of Falcon.

5.1 Introduction

Making the transition to post-quantum alternatives as straightforward as possible, requires construction of signature schemes having multiple good characteristics. It is important that keys and signatures are small, because many protocols, such as TLS and DNSSEC, are designed with the small sizes of pre-quantum schemes in mind [MdJvH+20].

Currently, the NIST has provided standards for the following three signature schemes: FN-DSA, ML-DSA and SLH-DSA, which are based respectively on Falcon [PFH+22], CRYSTALS-Dilithium [LDK+22] and Sphincs+ [HBD+22]. Of these three, Falcon has the smallest signatures, has smaller public keys than CRYSTALS-Dilithium, and is one of the most efficient post-quantum signature schemes. Yet, the NIST recommends CRYSTALS-Dilithium for general use, because generating Falcon signatures securely and efficiently is difficult [AAC+22].

Namely, Discrete Gaussian Sampling (DGS) [Kle00, GPV08] is used in Falcon to prevent signatures from leaking the private key [NR09]. DGS has become more efficient since its introduction [DN12a, DDLL13, GM18], in particular by exploiting ideal or module structures [Pei10, DP16] 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 operations and evaluation of transcendental functions.

The difficulties of DGS can be avoided completely by using a simple lattice, such as $\mathbb{Z}^n$ [DvW22, BGPS23], and an efficient, simple sampler for such lattice. One can build cryptography by rotating the simple lattice, and making this so-called *isomorphic* lattice public, while keeping the rotation private. In this case, the security is based on the intractability of recovering an isomorphism between two isomorphic lattices i.e., the *Lattice Isomorphism Problem* (LIP).

5.1.1 Contributions

In this chapter, we consider a concrete instantiation of the LIP-based cryptography framework [DvW22]. In their introductory work on LIP, Ducas and van Woerden mostly focus on theoretical and asymptotic results [DvW22], whereas this chapter focuses on the concrete construction of a signature scheme by combining their work with simple module

lattices, and verifying its practicality.

An attractive choice would be to consider the most structured option, namely modules of rank one over number fields, i.e., ideal lattices. However, this restricted version of LIP is solvable in classical polynomial time [GS02, LS17]. Instead, we work with rank two modules. It was quickly noted that NTRUSign signatures [HHP+03] were leaking the Gram matrix of the private key; recovering the private 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 [NR09], 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. While any ring of integers gets a lattice structure from its canonical embedding, this particular ring is viewed as an orthogonal lattice via the *coefficient embedding*, which allows for efficient computations. Moreover, LIP has already received some cryptanalytic attention in the case of an orthogonal lattice [Szy03].

To randomly generate a key pair, we must generate a basis for the module lattice R^2 following some distribution. We achieve this by mimicking NTRUSign key generation and setting the modulus q to 1, which allows us to use an efficient key generation technique [PP19]. Following the ideas presented by Ducas and van Woerden [DvW22], 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 Ducas and van Woerden [DvW22] hashes a message to $\left\{0, \frac{1}{q}, \ldots, \frac{q-1}{q}\right\}^{2n}$ for some q polynomial in the dimension n of the lattice. Another optimisation we propose is to hash the message to a smaller target space $\left\{0, \frac{1}{2}\right\}^{2n}$ 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 (omSVP). 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 inhomogeneous short integer solution

Table 5.1: Performance of Falcon and Hawk for $n = 512, 1024$ on an Intel® Core™ i5-4590 @3.30GHz processor with TurboBoost disabled. Falcon and Hawk are compiled with `-O3` and `-O2`, respectively. Timings of Sign correspond to batch usage, see Section 5.4.3.

Scheme	Falcon-512	Hawk-512	Falcon-1024	Hawk-1024
AVX2 KeyGen	7.95 ms	4.25 ms	23.60 ms	17.88 ms
Ref. KeyGen	19.32 ms	13.14 ms	54.65 ms	41.39 ms
AVX2 Sign	193 µs	50 µs	382 µs	99 µs
Ref. Sign	2449 µs	168 µs	5273 µs	343 µs
AVX2 Verify	50 µs	19 µs	99 µs	46 µs
Ref. Verify	53 µs	178 µs	105 µs	392 µs

problem [AKSY22] 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 efficiently with a few number theoretic transforms (NTT) or fast Fourier transforms (FFT). Moreover, we significantly compress the signature by dropping half of it, which is effectively computationally free. Decompressing a signature uses the Rounding-Off algorithm [Bab86]. 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

We propose a reference implementation and an AVX2 optimised implementation. The reference implementation makes no use of floating-points and only emulates them during key generation. The AVX2 version uses floating-points.

Table 5.1 lists performance of Hawk and compares with Falcon. On CPUs supporting AVX2, 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

Table 5.2: Key and signature sizes in bytes of Falcon and Hawk for $n = 512, 1024$.

Scheme	$\lvert\mathsf{sk}\rvert$	$\lvert\mathsf{pk}\rvert$	$\lvert\mathsf{Sig}\rvert$
Falcon-512 [PFH+22]	1281	897	652 ± 3
Hawk-512 (this work)	1153	1006 ± 6	542 ± 4
Falcon-1024 [PFH+22]	2305	1793	1261 ± 4
Hawk-512 (this work)	2561	2329 ± 11	1195 ± 6

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 Hawk uses a basis with smaller coefficients, its key generation is faster than Falcon.

Table 5.2 lists the key pair sizes and signature sizes of Falcon and 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, private keys are 128 bytes smaller. In Hawk-1024 we save a little on signatures, but our keys are larger. See Section 5.4.2 for the used encodings.

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

A recent variant of Falcon named Mitaka [EFG+22], also aims to make the signing procedure simpler and free from floating-point arithmetic. This is achieved with some loss in the signing quality compared to Falcon which makes signature forgeries somewhat easier, but the provided 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 [ASY22]. It might also be easier to mask against side-channel attacks, similar to how the lack of floating-points in the sampler simplifies the masking of Mitaka$_{\mathbb{Z}}$ [EFG+22, Sec. 7.3].

Implementation and source code

The *isochronous* C implementation, experimental data, and auxiliary scripts are open source:

https://github.com/ludopulles/hawk-aux

5.2 Scheme

In this section we present Hawk. We first give a version of Hawk that performs no compression on its signatures for simplicity, we call this uncompressed Hawk. We then introduce (compressed) Hawk and discuss how the security of Hawk directly reduces to that of the uncompressed Hawk. The auxiliary script `code/hawk.sage` is a proof of concept SageMath implementation of Hawk.

5.2.1 Uncompressed Hawk

The uncompressed version of our signature scheme is based on the scheme presented in [DvW22, 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 private key is some basis $\mathbf{B} \in \mathrm{SL}_2(R)$ where $\mathbf{B}$ (resp. $\mathrm{Rot}(\mathbf{B})$) generates R^2 (resp. $\mathbb{Z}^{2n}$). In the context of [DvW22, 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 $\mathbf{Q} = \mathbf{B}^\star \cdot \mathbf{B}$ associated to the basis $\mathbf{B}$. A signature on 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

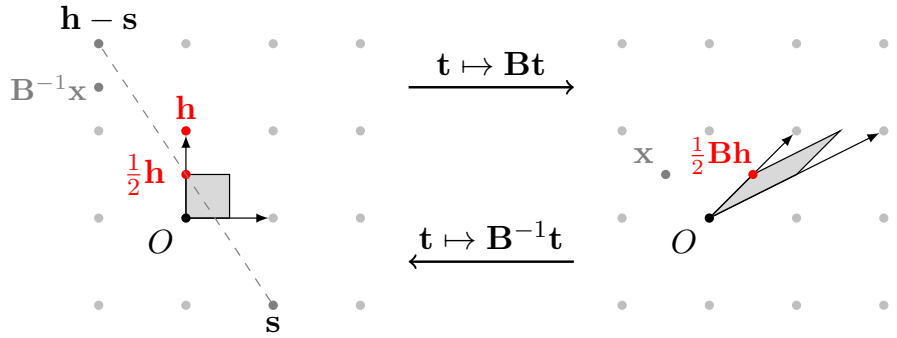

Figure 5.1: Illustration of signing. First $\mathbf{h}$ is sampled (left), then $\mathbf{B}$ is applied, a short lattice point $\mathbf{x}$ is sampled from a discrete Gaussian on $\mathbb{Z}^{2n} + \frac{1}{2}\mathbf{B} \cdot \mathbf{h}$ (right). Finally, $\mathbf{B}^{-1}$ applied to $\mathbf{x}$ is subtracted from $\frac{1}{2}\mathbf{h}$ to obtain a lattice point $\mathbf{s}$ close to $\mathbf{h}/2$ in $\|\cdot\|_{\mathbf{Q}}$, and we update $\mathbf{s}$ to $\mathbf{h} - \mathbf{s}$ to ensure SymBreak$(h_1 - 2s_1)$ is satisfied.

$\mathbb{Z} + 1/2$. 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 5.2 and below. See Figure 5.1 for a visualisation when $n = 1$. Verification checks if the distance $\left\|\frac{\mathbf{h}}{2} - \mathbf{s}\right\|_{\mathbf{Q}}$ between $\mathbf{s}$ and $\frac{1}{2}\mathbf{h}$ is not too large, which only requires the public key $\mathbf{Q} = \mathbf{B}^{\star}\mathbf{B}$ and not the private 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 a short coset vector,
4. σ_{ver}: controls the acceptable distance from a signature to a halved hash,
5. saltlen: controls the probability of hash collisions.

For uncompressed Hawk we present KeyGen in Algorithm 5.1, Sign in Algorithm 5.2 and Verify in Algorithm 5.3. The security parameter n is a power of two, and we assume the internal parameters can be computed from it. We use previous work to generate the unimodular transformation $\mathbf{B}$ efficiently [PP19, Alg. 4], by sampling the first basis

Algorithm 5.1. KeyGen(1^n): Hawk key generation

Input: parameter n
Output: a Hermitian form $\mathbf{Q} = \mathbf{B}^\star\mathbf{B}$, and the corresponding (randomly generated) basis $\mathbf{B} \in \mathrm{SL}_2(R)$ for R^{2n}

1: sample $f, g \in R$ with coefficients from $D_{\mathbb{Z},\sigma_{\mathsf{pk}}}$
2: $q_{00} = f^\star f + g^\star g$
3: **if** $2 \mid \mathrm{N}(f)$ **or** $2 \mid \mathrm{N}(g)$ **or** $\|(f,g)\|^2 \le \sigma_{\mathsf{sec}}^2 \cdot 2n$ **then**
4: **restart**
5: $(F,G)^\mathsf{T} \leftarrow \mathrm{TowerSolveI}_{n,1}(f,g)$ [PP19, Alg. 4], **if** $\perp$, **restart**
6: $\begin{pmatrix} F \\ G \end{pmatrix} \leftarrow \begin{pmatrix} F \\ G \end{pmatrix} - \mathrm{ffNP}_R\left(\frac{f^\star F + g^\star G}{q_{00}}, \mathrm{ffLDL}_R^\star(q_{00})\right) \cdot \begin{pmatrix} f \\ g \end{pmatrix}$ [DP16]
7: $\mathbf{B} = \begin{pmatrix} f & F \\ g & G \end{pmatrix}$
8: $\mathbf{Q} = \begin{pmatrix} q_{00} & q_{01} \\ q_{10} & q_{11} \end{pmatrix} = \mathbf{B}^\star \cdot \mathbf{B}$
9: **return** $(\mathsf{pk}, \mathsf{sk}) = (\mathbf{Q}, \mathbf{B})$

Algorithm 5.2. $\mathrm{Sign}_{\mathbf{B}}(\mathsf{m})$: Hawk signature generation

Input: private key $\mathbf{B}$ and message $\mathsf{m} \in \{0,1\}^*$
Output: a (randomly generated) valid signature $\mathsf{sig} \in \{0,1\}^{\mathsf{saltlen}} \times R^2$

1: $\mathsf{r} \leftarrow U(\{0,1\}^{\mathsf{saltlen}})$
2: $\mathbf{h} \leftarrow \mathsf{H}(\mathsf{m} \| \mathsf{r})$
3: $\mathbf{t} \leftarrow \mathbf{B} \cdot \mathbf{h} \pmod 2$
4: $\mathbf{x} \leftarrow \mathcal{D}_{\mathbb{Z}^{2n} + \frac{1}{2}\mathbf{t}, \sigma_{\mathsf{sign}}}$
5: **if** $\|\mathbf{x}\|^2 > \sigma_{\mathsf{ver}}^2 \cdot 2n$ **then**
6: **restart** (optional, see Section 5.5.2)
7: $\mathbf{s} = (s_0, s_1)^\mathsf{T} \leftarrow \frac{1}{2}\mathbf{h} - \mathbf{B}^{-1}\mathbf{x}$ (parse $\mathbf{x} \in \frac{1}{2}R^2$ via Vec^{-1})
8: **if** $\mathrm{SymBreak}(h_1 - 2s_1)$ is `False` **then**
9: $\mathbf{s} \leftarrow \mathbf{h} - \mathbf{s}$
10: **return** $\mathsf{sig} = (\mathsf{r}, \mathbf{s})$

Algorithm 5.3. $\textsc{Verify}_{\mathbf{Q}}(\mathsf{m}, \mathsf{sig})$: Hawk signature verification

Input: public key $\mathbf{Q}$, message $\mathsf{m} \in \{0,1\}^*$ and signature sig
Output: 1 if sig is a valid signature on m, otherwise 0

1: $(\mathsf{r}, \mathbf{s}) \leftarrow \mathsf{sig}$
2: $\mathbf{h} \leftarrow \mathsf{H}(\mathsf{m} \| \mathsf{r})$
3: **return** $\left[\!\!\left[\begin{array}{l} \mathbf{s} \in R^2 \textbf{ and } \left\| \frac{\mathbf{h}}{2} - \mathbf{s} \right\|_{\mathbf{Q}}^2 \leq \sigma_{\mathsf{ver}}^2 \cdot 2n \\ \textbf{and } \textsc{SymBreak}(h_1 - 2s_1) \text{ is } \texttt{True} \end{array} \right]\!\!\right]$

vector $(f, g)^{\mathsf{T}}$, and, if possible, completing it with a second basis vector $(F, G)^{\mathsf{T}}$ such that $\det(\mathbf{B}) = 1$. We combine this with the fast Babai's Nearest-Plane reduction [DP16] to obtain a shorter second basis vector $(F, G)^{\mathsf{T}}$. In KeyGen checks are performed prior to completing the basis $\mathbf{B}$. In TowerSolveI it is necessary for $\mathrm{N}(f)$ or $\mathrm{N}(g)$ to be an odd integer [PP19]. 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 5.3. Note that a signer also has easy access to $\mathbf{B}^{-1}$, because

$$\mathbf{B}^{-1} = \begin{pmatrix} G & -F \\ -g & f \end{pmatrix}, \tag{5.1}$$

follows from $fG - gF = \det(\mathbf{B}) = 1$.

In Sign and Verify we check the condition $\textsc{SymBreak}(h_1 - 2s_1)$, 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 $\textsc{SymBreak}(e)$ to be $\texttt{True}$ if and only if $e \neq 0$ and the first nonzero coefficient of $\textsc{Vec}(e)$ is positive. Checking this condition on $h_1 - 2s_1$ in Verify 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 $\textsc{SymBreak}(h_1 - 2s_1)$ is not satisfied during verification.

By lines 8 and 9 of Algorithm 5.2, this means SymBreak$(h_1 - 2s_1)$ and SymBreak$(2s_1 - h_1)$ 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 $\|\frac{\mathbf{h}}{2} - \mathbf{s}\|_{\mathbf{Q}}^2 = \|\mathbf{B}\left(\frac{\mathbf{h}}{2} - \mathbf{s}\right)\|^2 = \|\pm\mathbf{x}\|^2 \leq \sigma_{\mathsf{ver}}^2 \cdot 2n$, with $-\mathbf{x}$ given by Line 9 of Sign.

Key storage

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

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

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 KeyGen. 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 $q_{10} = q_{01}^\star$, so one may simply drop q_{10}. Moreover, since $\det(\mathbf{B}) = 1$ we have $q_{00}q_{11} - q_{01}q_{10} = \det(\mathbf{Q}) = \det(\mathbf{B}^\star)\det(\mathbf{B}) = 1$, therefore q_{11} can be dropped and reconstructed as $q_{11} = (1 + q_{01}^\star \cdot q_{01})/q_{00}$. 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 5.4.2.

5.2.2 (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 Verify 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 KeyGen. In Verify, 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 Rounding-Off and Babai's Nearest-Plane algorithm [Bab86]. 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 Rounding-Off for Hawk. Specifically, to reconstruct s_0, we use,

$$s_0' = \left\lceil \frac{h_0}{2} + \frac{q_{01}}{q_{00}} \left(\frac{h_1}{2} - s_1 \right) \right\rfloor , \tag{5.2}$$

where the rounding map $\lceil \cdot \rfloor \colon \mathbb{K} \to R$ acts coefficientwise, and satisfies $x - \lceil x \rfloor \in (-1/2, 1/2]$ for all $x \in \mathbb{R}$. Hence, Verify is adapted to read a signature $\mathsf{sig} = (\mathsf{r}, s_1)$ and reconstruct s_0' using Equation (5.2), before setting $\mathbf{s} = (s_0', s_1)^{\mathsf{T}}$. Observe that $s_0' = s_0$ holds 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 $\lceil \cdot \rfloor$ is equal to $\frac{f^\star x_0 + g^\star x_1}{f^\star f + g^\star g}$ because we have $(q_{00}, q_{01}) = (f^\star, g^\star) \cdot \mathbf{B}$ and $\mathbf{B} \cdot \left(\frac{\mathbf{h}}{2} - \mathbf{s} \right) = (x_0, x_1)^{\mathsf{T}}$. Thus, we certainly recover the correct s_0 from $\mathbf{Q}$, $\mathbf{h}$ and s_1 if

$$\frac{f^\star x_0 + g^\star x_1}{f^\star f + g^\star g} \in \left(-\frac{1}{2}, \frac{1}{2} \right)^n . \tag{5.3}$$

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

$$\left\| \left(\frac{f^\star}{q_{00}}, \frac{g^\star}{q_{00}} \right) \right\|^2 = \frac{1}{n} \operatorname{Tr}\left(\frac{f^\star f + g^\star g}{q_{00}^2} \right) = \frac{1}{n} \operatorname{Tr}\left(q_{00}^{-1} \right) = \langle 1, q_{00}^{-1} \rangle. \tag{5.4}$$

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 5.4.1. In summary, Algorithms 5.1, 5.2 and 5.3 are changed as follows for Hawk.

1. In KeyGen, 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 Equation (5.3).
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 Verify, given a signature (r, s_1) reconstruct s_0' using Equation (5.2) 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 Equation (5.3) holds except with small probability and forego item 2. in the list, see Section 5.5.

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 Section 5.6.3 we further reduce the forgery security of uncompressed Hawk to an assumption called the one more short vector problem, or omSVP.

5.3 Cryptanalysis

In this section we provide a concrete cryptanalysis of Hawk. Whereas the formal security arguments we make in Sections 5.6.1 and 5.6.3 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 private key from public information and forging a new signature given at most $q_s = 2^{64}$ signatures. Furthermore, we report the various parameters, probabilities and block sizes output by our cryptanalysis in Table 5.3.

The constraints on various quantities in our scheme are expressed 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 5.2. In particular for $\mathbf{x} \leftarrow \mathcal{D}_{\mathbb{Z}^d,\sigma}$, with σ above smoothing, $\mathbb{E}[\|\mathbf{x}\|] \approx \sigma\sqrt{d}$ [MR07, Sec. 4]. For example, the verification of signatures is determined by a distance, say ℓ. Instead of referring to ℓ, we use the shorthand $\sigma_{\mathsf{ver}} = \ell/\sqrt{d}$.

We stress that the relations of Figure 5.2 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 [NR09, DN12b] if an adversary is given access to signature transcripts. We then choose σ_{pk} which controls key generation in Algorithm 5.1. It must be large enough that recovering the private 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

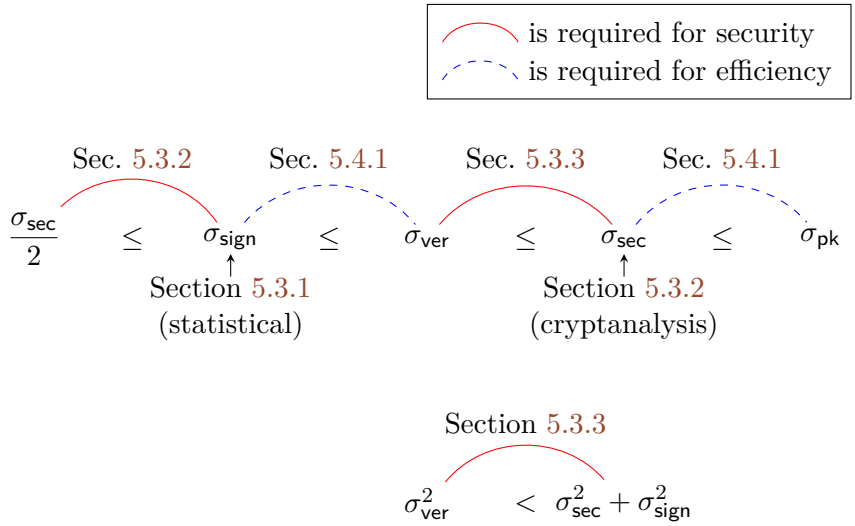

Figure 5.2: Summary of necessary relationships between various $\sigma_\bullet$

σ_{sec}, a parameter that represents the shortest a public basis can be before one recovers the private 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 5.4.1. 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.

5.3.1 Signature sampling

We choose σ_{sign} large enough to avoid signatures leaking information about a private key. Following the methodology in Falcon [PFH+22, Sec. 2.6], for security parameter λ in the face of an adversary allowed q_s signatures, it is enough to set

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

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 2.81. We ensure that, following the analysis of Falcon [PFH+22, Sec. 3.9.3], our probability distribution tables have

a Rényi divergence at order 513 from their ideal distributions of less than $1 + 2^{-79}$. These computations are done in the auxiliary script `code/generate_C_tables.py`.

5.3.2 Key recovery

In Hawk, the problem of recovering the private key $\mathbf{B} \in \mathrm{SL}_2(\mathcal{O}_\mathbb{K})$ from the public key $\mathbf{Q} = \mathbf{B}^\star \cdot \mathbf{B}$ 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 $\mathbf{U}^\star \cdot \mathbf{Q} \cdot \mathbf{U} = \mathbf{I}_2(\mathbb{K})$, i.e., reducing (any lattice basis corresponding to) $\mathbf{Q}$ to an orthonormal basis. As mentioned in [DvW22], all known algorithms to solve LIP for modules of rank at least two require finding at least one shortest vector. Therefore, we assume that the best key recovery attack requires one to find a single shortest vector.

Unusual-SVP

The shortest vectors in R^2 have length 1, which is a factor of order $\Theta(\sqrt{n})$ shorter than predicted by the Gaussian heuristic. Recovering such 'unusually' short vectors is easier than generic shortest vectors, and can be achieved by running the BKZ lattice reduction algorithm with block size 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 $\mathrm{Rot}(\mathbf{Q})$ 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 block size, see [AD21] for a survey. As an estimate, given that our lattice has unit volume, and we search for a vector of length one, we require a block size β such that

$$\sqrt{\beta/d} \approx \delta_\beta^{2\beta-d-1}, \tag{5.5}$$

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 [CN11, BSW18, DDGR20, PV21].

In Figure 5.3 we plot the estimate given by Equation (5.5) where the $o(d)$ term is concretised to some constant, the estimate given by the leaky-LWE-estimator [DDGR20], which applies the concrete improvements mentioned above, and experimental data. These experiments apply the BKZ 2.0 algorithm with lattice point enumeration [CN11] as implemented in fplll [FPL24] 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 [DvW22, 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 5.1. We see that below approximately dimension 80 instances can be solved with LLL reduction [LLL82], and that afterwards the required block size approximately increments by one when the dimension increases by two, as Equation (5.5) would suggest. Moreover, we see that above approximately block size 70 the model of Dachman-Soled et al. [DDGR20] appears especially accurate. Therefore, we use this model to determine β_{key} in Table 5.3. We use a simple progressive strategy where the block size increments by one after each tour, which we expect to require a block size perhaps two or three larger than a more optimal progressive strategy.

Minimal basis randomisation

For the experiments of Figure 5.3 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 5.4. For σ_{pk} below a certain threshold, instances can be solved by LLL. Then as σ_{pk} increases past this threshold the instances become harder, before reaching an empirical 'maximum hardness' (at least with respect to these experiments) where further increases in σ_{pk} appear to give no extra security.

When running BKZ one encounters shorter vectors as β grows. In a random unit-volume lattice, 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 [DDGR20] predicts that we suddenly jump from finding vectors of length $\ell_0 = \Theta(d)$

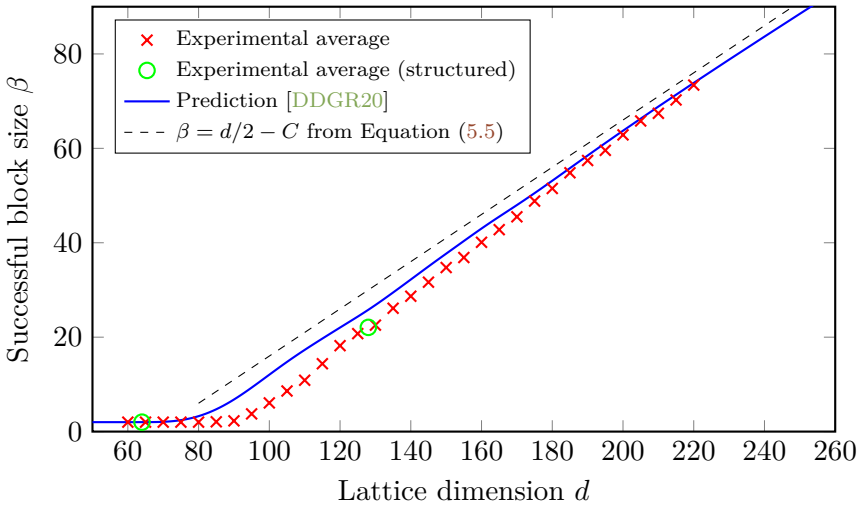

Figure 5.3: Block size required to recover a shortest vector via lattice reduction as a function of dimension d. We ran progressive BKZ (one tour per block size) over $\mathbb{Z}^d$ using an input form generated with $\sigma_{\mathsf{pk}} = 20$ and report the average successful β that recovered a length one vector over 40 instances. We used the BKZ simulator and probabilistic model of Dachman-Soled et al. [DDGR20], accounting for the d target solutions. See auxiliary scripts `code/BKZ_simulator.sage` and `code/exp_varying_n.sage`.

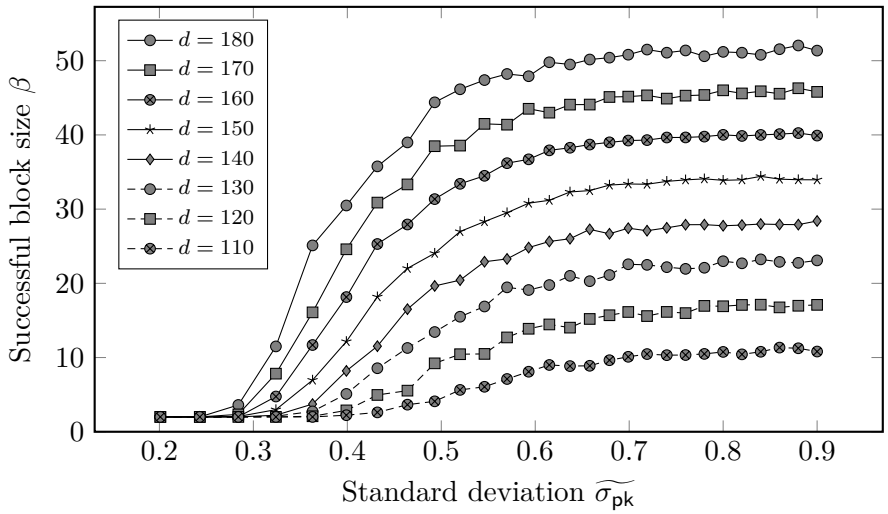

Figure 5.4: Block size required to recover a shortest vector via lattice reduction as a function of the standard deviation $\widetilde{\sigma_{\mathsf{pk}}}$. We ran progressive BKZ (one tour per block size) over $\mathbb{Z}^d$ using an input form generated with various σ_{pk} and report the average 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 auxiliary script `code/exp_varying_sigma.sage`.

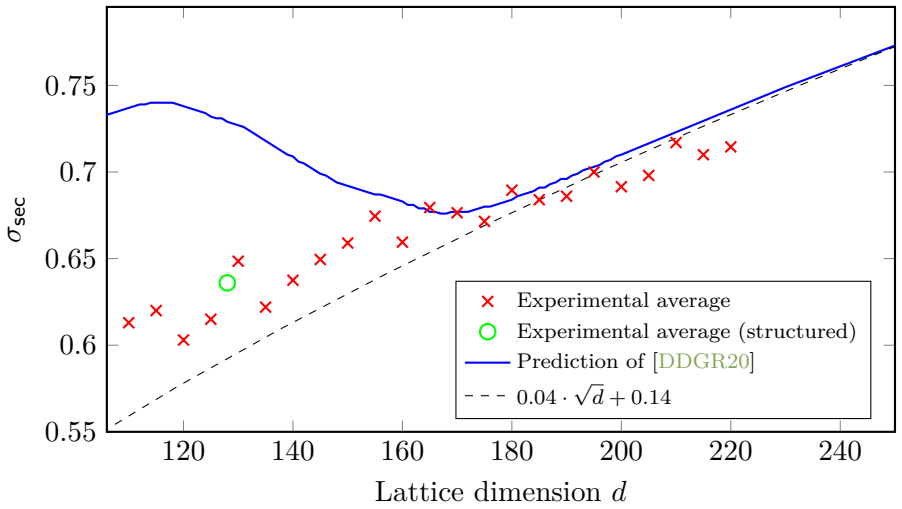

Figure 5.5: Shortest basis vector length before the recovery of a length one vector, given in terms of σ_{sec}. We ran progressive BKZ (one tour per block size) over $\mathbb{Z}^d$ using an input form generated with $\sigma_{\mathsf{pk}} = 20$ and report the average σ_{sec} determined by the shortest basis vector in the penultimate BKZ tour over 40 instances. See auxiliary scripts `code/exp_varying_n.sage` and `code/predict_varying_n.sage`.

to finding a shortest vector of length 1. This threshold behaviour was observed and discussed in [BGPS23, 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 5.5 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 5.3 we take σ_{sec} as the output from the prediction of [DDGR20]. 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 private key.

We note that the prediction of [DDGR20] in Figure 5.5 is inaccurate for $d \leq 180$, similarly (but more noticeably) to Figure 5.3. 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 block sizes (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 5.3.1 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 $\|2 \cdot (\frac{1}{2}\mathbf{h} - \mathbf{s})\|_{\mathbf{Q}} = 2\|\mathbf{x}\| \approx 2\sigma_{\mathsf{sign}}\sqrt{d} < \sigma_{\mathsf{sec}}\sqrt{d}$. Therefore, doubling a public quantity given by a signature may describe a shorter lattice vector than those seen just before private key recovery.

5.3.3 Signature forgery

Strong forgery

We consider the general problem of forging a signature on 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 $\|\frac{1}{2}\mathbf{h} - \mathbf{s}\|_{\mathbf{Q}} \leq \sigma_{\mathsf{ver}}\sqrt{d}$. We use the heuristic that solving such an approximate CVP instance is at least as hard as solving an approximate SVP instance with the same approximation factor over the same lattice. We determine the expected block size β using the BKZ simulator [DDGR20] such that our first basis vector has norm less than $\sigma_{\mathsf{ver}}\sqrt{d}$, and report it as β_{forge} in Table 5.3. Note that since we use the BKZ simulator of [DDGR20] 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 private 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 on some message, and hoping that it remains a valid signature on 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 5.3.2. We give arbitrarily many such length ℓ_0 vectors to the adversary for free. Using the function

`forgeProbability` from the auxiliary script `code/find_params.sage`, we estimate the probability the attack succeeds, i.e., that $\|\mathbf{x}+\mathbf{v}\|^2 \leq d \cdot \sigma_{\mathsf{ver}}^2$ for $\mathbf{x} \leftarrow \mathcal{D}_{\mathbb{Z}^{2n},\sigma_{\mathsf{sign}}}$ and $\mathbf{v}$ of length ℓ_0. If $\mathbf{x}$ were sampled from a spherical continuous Gaussian, then considering any such $\mathbf{v}$ would give the same distribution of squared lengths. We examine the distribution of $\|\mathbf{x}+\mathbf{v}\|^2$ for two 'extremal' choices of $\mathbf{v}$; the first has all its weight in one coordinate, $\mathbf{v} = (\lfloor \ell_0 \rfloor, 0, \ldots, 0)$, and the second is as balanced as possible, e.g. $\mathbf{v} = (1, \ldots, 1, 2, \ldots, 2)$ for $\|\mathbf{v}\| = \lfloor \ell_0 \rfloor$. Note that the distribution of $\|\mathbf{x}+\mathbf{v}\|^2$ is invariant under signed permutations of $\mathbf{v}$. We report our estimate for the success probability of this attack as Pr[weak forgery] in Table 5.3.

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 block sizes reported in Table 5.3. Our model makes use of recent improvements [DDGR20] and enjoys the experimental evidence above. The unusually short vectors in $\mathbb{Z}^d$ are a factor about 1.17 shorter than the shortest vector in the NTRU lattice of Falcon, after appropriate renormalisation. Thus, key recovery for Hawk will be slightly easier than for 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 for 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. However, we note that part of the key recovery methodology of Falcon is overconservative [DPPvW22b, App. D].

5.4 Parameters and performance

In Table 5.3 we list parameters and the output of the auxiliary script `code/find_params.sage` using our concrete cryptanalysis for Hawk.

Section 5.4.1 explains how these parameters were chosen. We explain the encoding used for public keys and signatures, and the simple encoding used for private keys, in Section 5.4.2. Finally, Section 5.4.3 contains the details behind Table 5.1. Note that the formal reductions in Section 5.6 require different parametrisations.

5.4.1 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 $q_s \cdot 2^{-\lambda}$ [PFH+22, 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 5.3 is such that the probability of $\|(f,g)\|^2 > \sigma_{\mathsf{sec}}^2 \cdot 2n$ is greater than 99.5% for f, g both with odd algebraic norm and sampled as in KeyGen. 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, Equation (5.3) 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, see the function `failProbability` in the auxiliary script `code/find_params.sage`. To obtain a strict upper bound on this probability one can use a looser tail bound analysis via Lemma 2.79 and Lemma 2.84, 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 private key $\mathsf{sk} = \mathbf{B}$, we provide a heuristic upper bound on the probability that decompression using Equation (5.2) gives $s_0' \neq s_0$, which is upper bounded by the probability that Equation (5.3) 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 $\mathbf{0}$

Table 5.3: Parameter sets for Hawk and their estimated security. The dimension, bit security and transcript size are used to determine σ_{sign}. Other standard deviations are determined in Section 5.3. We then estimate the probability that a signature fails for being too long. The estimated required block sizes 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 5.3.3.

	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 Recovery 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 Recovery (BKZ) β_{key}	211	452	940
Strong Forgery (BKZ) β_{forge}	211	452	1009
$\log_2(\Pr[\text{weak forgery}])$	-83	-143	-683

and standard deviation σ_{sign}. Following Section 5.2.2 the decompression succeeds if Equation (5.3) holds, or equivalently,

$$\frac{f^\star}{q_{00}} \cdot x_0 + \frac{g^\star}{q_{00}} \cdot x_1 \in \left(-\frac{1}{2}, \frac{1}{2}\right)^n.$$

Since $\mathbf{B}$ is fixed, each coefficient in the left hand side above is normally distributed with mean 0 and variance $\|(f^\star/q_{00}, g^\star/q_{00})\|^2 \cdot \sigma_{\mathsf{sign}}^2 = \langle 1, q_{00}^{-1}\rangle \cdot \sigma_{\mathsf{sign}}^2$, using Equation (5.4). Hence, the probability that a particular coefficient is outside $(-\frac{1}{2}, \frac{1}{2})$ is given by

$$\operatorname{erfc}\left(\frac{1/2}{\sigma_{\mathsf{sign}}\sqrt{2 \cdot \langle 1, 1/q_{00}\rangle}}\right).$$

Using a union bound, the probability that decompression fails is heuristically bounded from above by $n \cdot \operatorname{erfc}\Big(1/(\sigma_{\mathsf{sign}}\sqrt{8 \cdot \langle 1, 1/q_{00}\rangle})\Big)$. By rejecting keys having $\langle 1, q_{00}^{-1}\rangle \geq \nu_{\mathsf{dec}}$, signature decompression fails for any $\mathbf{B}$ heuristically with probability at most,

$$n \cdot \operatorname{erfc}\Big(1/\left(\sigma_{\mathsf{sign}}\sqrt{8\nu_{\mathsf{dec}}}\right)\Big).$$

If we take $\nu_{\mathsf{dec}} = 1/1000$ in Hawk-512, this probability is upper bounded by 2^{-105}. This condition on (f, g) in KeyGen 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 TowerSolveI 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 KeyGen.

5.4.2 Encoding

In Hawk both the private 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 encoding [Gol66]. This encoding is used for the signatures in Falcon [PFH+22].

For encoding, we use an altered absolute value,

$$|\cdot|' \colon \mathbb{Z} \to \mathbb{N}, \qquad x \mapsto \begin{cases} x & \text{for } x \geq 0 \\ -x-1 & \text{for } x < 0. \end{cases}.$$

The map that sends x to its sign and $|x|'$ gives a bijection $\mathbb{Z} \to \{0,1\} \times \mathbb{N}$. 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 [PFH+22, 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 encoding 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 private 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. Likewise, for the remaining polynomial F of the private key, we encode with one byte per coefficient (recall G can be reconstructed). We use this simple encoding and decoding for our private 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 $18 < 2^5$.

By using the Asymmetric Numeral Systems (ANS) encoding [Dud14, DTGD15] instead of Golomb–Rice encoding, signature sizes of Falcon shrink by 7%–14% [ETWY22]. Making use of the open-source rANS implementation by Fabien Giesen at https://github.com/rygorous/ryg_rans in the encoding and decoding of Hawk public keys, we are able to reduce public key sizes by 15 and 30 bytes for Hawk-512 and Hawk-1024 respectively. Because this only saves less than 2% on the public key

sizes but adds significantly to the code complexity, we did not use this encoding in the final version.

5.4.3 Performance

We report on the performance of our implementation of Hawk-512 and compare it to that of Falcon-512 in Table 5.1. Hawk was compiled with `gcc` version 12.1.0 using compilation flag `-O2` (and `-mavx2` for AVX2). Optimisation flag `-O2` gave better performance than `-O3`. 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` using compilation flags `-O3 -march=native`.

Memory usage

The reference implementation of Hawk-512 uses 24 kB, 15 kB and 18 kB of RAM for KeyGen, Sign and Verify respectively, versus 16 kB, 40 kB and 4 kB for Falcon-512 respectively. Hawk requires more RAM for KeyGen compared to Falcon to execute Line 6 of Algorithm 5.1. 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 27 kB and 24 kB for Sign and Verify respectively, prioritising speed over memory usage. For Hawk-1024 and Falcon-1024 memory usage roughly doubles.

Batched versus dynamic signing

For consistency with the Falcon report [PFH+22], Table 5.1 reports signing speeds for batched usage, that is, after some precomputation expanding the private key (called `expand_seckey`). If one needs to start from the private key without expanding it, the performance of Falcon is significantly affected while the precomputation is much lighter for Hawk. The timings for dynamic signing are given in Table 5.4.

5.5 Implementation details

We have written a reference and an AVX2 optimised implementation for Hawk-512 and Hawk-1024 in the C programming language. We have

Table 5.4: Performance of dynamic signing for Falcon and Hawk on same PC with same compilation flags as in Table 5.1.

Scheme	AVX2 Sign	Reference Sign
Falcon-512 [PFH+22]	320 µs	5427 µs
Hawk-512 (this work)	58 µs	168 µs
Speed-up (Falcon/Hawk)	↓ 5.5×	↓ 32×
Falcon-1024 [PFH+22]	656 µs	11 868 µs
Hawk-1024 (this work)	114 µs	343 µs
Speed-up (Falcon/Hawk)	↓ 5.8×	↓ 35×

reused a significant portion from the public implementation of Falcon, because Hawk and Falcon are algorithmically rather similar. This section provides details on the design choices that were made in these implementations.

All the implementations are *isochronous*, i.e., the runtime of KeyGen and Sign gives no information about the used private data. Verification is trivially isochronous as it only uses public information. Our implementations are not *constant-time* because of two reasons: KeyGen and Sign sometimes restart depending on internal probabilistic events, and time varies to encode the public quantities pk and sig.

Almost all processors that support AVX2 have a floating-point unit, and plenty of RAM. Therefore, the AVX2 optimised implementation makes extensive use of built-in floating-point numbers and FFTs for multiplications or divisions in a 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 to decompress signatures during verification. As floating-points are required in TowerSolveI 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.

Sampling and pseudorandomness

The extendable output function SHAKE256 is used to seed a pseudorandom number generator (PRNG) based on ChaCha20 [Ber08] during

sampling in KeyGen and Sign. As the Gaussian parameters used in the scheme are fixed, DGS can be performed efficiently with precomputed probability tables and this PRNG.

Fast polynomial arithmetic

We want all computations to be performed in time $O(n \log n)$. Addition and subtraction of two polynomials in $R = \mathbb{Z}[X]/(X^n + 1)$ are linear operations, 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 use the number theoretic transform (NTT), which works with a sufficiently large prime modulus $p \equiv 1 \pmod{2n}$ and an element $\omega \in \mathbb{F}_p^\times$ of order $2n$. As $\mathbb{F}_p^\times$ is a cyclic group of order $p - 1$, the NTT computes $f(\omega^i) \in R$ for all $i \in (\mathbb{Z}/2n\mathbb{Z})^\times$ in time $O(n \log n)$.

When polynomials are transformed using FFT or NTT, multiplication is coefficientwise. The resulting polynomial in R is then obtained via the inverse transformation. The NTT gives the correct result $a(X) \cdot b(X)$, if the used prime p is at least twice as large as the absolute value of any coefficient of $a(X)$, $b(X)$ or $a(X) \cdot b(X)$. We only use the NTT in the reference implementation.

Multiplications in R could also be implemented with Karatsuba or Toom–Cook style multiplication. For certain applications such as masking this may have comparable performance for the given parameters.

The FFT in our implementation uses double precision floating-point numbers (`double`). When a CPU supports AVX2, the FFT is faster than the NTT, but also requires more RAM. Like Falcon, we use the 'bit reversal' ordering of the roots in the FFT representation, 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 performance benefits [PFH+22, 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.

Divisions and signature decompression

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

5.5.1 Key generation

Key generation is the same for both implementations, as the majority of the work is the TowerSolveI procedure. This requires floating-points, so we emulate them in the reference implementation. At the time TowerSolveI was invented [PP19, Sec. 6], the problem of having a vector reduction in TowerSolveI 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 $\mathcal{D}_{\mathbb{Z},\sigma_{\mathsf{pk}}}$. This sampling is performed using a precomputed cumulative distribution 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

$$\left\lceil 2^{63} \cdot \Pr_{x \leftarrow \mathcal{D}_{\mathbb{Z},\sigma_{\mathsf{pk}}}} [|x| \geq k + 1] \right\rfloor,$$

so the statistical distance between $\mathcal{D}_{\mathbb{Z},\sigma_{\mathsf{pk}}}$ and the implemented sampling procedure is smaller than 2^{-64}.

We then check five conditions, before calling TowerSolveI.

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. $ff^\star + gg^\star$ 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 (mod $2n$) 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.

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 [PFH+22, 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 KeyGen (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 an `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.

Babai reduction

In KeyGen, after TowerSolveI outputs relatively short $(F, G)^{\mathsf{T}}$, we perform another reduction step using ffNP [DP16, Alg. 4] to further shrink the size of $(F, G)^{\mathsf{T}}$. TowerSolveI returns an element $(F, G)^{\mathsf{T}}$ whose projection onto the module lattice $M = (f, g)^{\mathsf{T}} \cdot R$, i.e., the rank
5 n real lattice with basis $\mathbf{C} = \text{Rot}((f, g)^{\mathsf{T}})$, 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 Algorithm 5.1 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. The ith vector $(i = 1, 2, \ldots, n)$ has an expected norm of $\sqrt{\frac{2n+1-i}{2n}} \cdot \|(f, g)\|$ [Pre15, Lem. 6.9]. 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 5.1. This shrinks the average public key size in Hawk-512 from 1027 bytes to 1006. We note that using ffNP in Falcon would not lead to smaller public key sizes, and only leads to a slightly better trapdoor sampler.

Our implementation of ffNP is written in a memory friendly manner, overwriting certain polynomials when they are no longer used.

Remark 5.1. Since the initial work on Hawk [DPPvW22a], multiple changes have been made to KeyGen. Namely, the following things were modified when Hawk was originally submitted to NIST [BBD+23].

- The centred binomial distribution (CBD) is used to sample coefficients for f and g, which only slightly changes the size distribution of public keys and signatures.
- Line 6 in Algorithm 5.1 is omitted, which changes reduction of (F, G) to a structured variant of Rounding-Off [Bab86].
- A new implementation of TowerSolveI is used [Por23] that only uses fixed-points and no floating-points anymore.
- The private key is essentially encoded as

$$(\mathsf{kgseed}, F \bmod 2, G \bmod 2),$$

  where kgseed is the seed used to sample a completable (f, g) in KeyGen. During signing, Line 3 of Algorithm 5.2 computes the target modulo 2, so it is sufficient to know $(F, G) \pmod 2$. Moreover, s_1 can be computed on Line 7 without (F, G) by Equation (5.1). The private key consists of 184 and 360 bytes for Hawk-512 and Hawk-1024 respectively.
- We apply a light version by Pornin and Stern [PS05] of the BUFF transform to Hawk [CDF+21, ADM+24]. Namely, the hash function on Line 2 of Algorithm 5.2 gets as input the message, salt and a hash $\mathsf{H}(\mathbf{Q})$ of the public key. Now, KeyGen computes $\mathsf{H}(\mathbf{Q})$ once to store it in the private key, so Sign can use it.

Subsequently, when Hawk advanced to the second round [BBD+24], there was one more modification. Vector reduction inside TowerSolveI was adjusted to reduce F with respect to f using Rounding-Off, and then update G accordingly. Namely, a multiple of (f, g) is subtracted from (F, G) such that $\lceil F/f \rfloor$ holds. This reduction often gives the same reduced vector (F, G) as calling Rounding-Off algorithm with (F, G) and (f, g) [Por25]. This allows for an easier implementation of KeyGen that uses only 12 kB of RAM for Hawk-512.

5.5.2 Signature generation

A message m is hashed to some element $h \in \{0,1\}^{2n}$ using SHAKE256 on the bit string $\mathsf{m}\|r$, with $r \in \{0,1\}^{320}$ a random salt. Whenever a check fails in signing, a new random salt is generated. Note 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 because the salt comes last.

In the case of Falcon, we do not see how reusing the salt is compatible with the security argument of Gentry et al. [GPV08]. 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.

First, there is a check if a signature is short enough, which is Line 5 in Algorithm 5.2. 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
5 double precision floating-points inside the FFT while the reference implementation uses the NTT with prime $p = 2^{16} + 1$.

AVX2 optimised implementation

During AVX2 signing we transform the whole basis $\mathbf{B}$ into FFT representation. Since G is not stored in the private 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 $\mathbf{x} = (x_0, x_1)^\mathsf{T} \leftarrow \mathcal{D}_{2\mathbb{Z}^{2n}+\mathbf{t},2\sigma_{\mathsf{sign}}}$ 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 $-gx_0 + fx_1$ using Equation (5.1). Now the signature is computed using

$$s_1 = \frac{h_1 - (-gx_0 + fx_1)}{2}.$$

Note here that h_1 does not need to be converted to FFT representation.

If one adds a compilation flag `-DHAWK_RECOVER_CHECK` explicitly during compilation, it is checked whether the vector $\mathbf{x}$ satisfies Equation (5.3)

during signing. In batched signing, the basis $\mathbf{B}$ is precomputed in FFT representation and therefore cannot be altered. If the code is compiled with the flag `-DHAWK_RECOVER_CHECK`, the expanded private key also contains $1/(f^\star f + g^\star g)$ 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 Fermat prime $p = 2^{16} + 1$ which is $1 \pmod{2048}$. 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 KeyGen holds, f is invertible modulo p.

One can reduce the memory usage of signing from 15 kB down to 8 kB 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 [Mon85] 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 using

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

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 Equation (5.3), 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 5.4.1.

5

Failure checks

By default, we catch invalid signatures before they are issued. The first failure check is Line 5 of Algorithm 5.2. A decompression may also output an incorrect $s_0' \neq s_0$. To catch this we could check with an FFT if Equation (5.3) 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 5.4.1).

Omitting the first check may 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 nonnegligible probability (see Section 5.4.1) of an invalid signature, which might be mitigated by reparametrising the scheme.

Discrete Gaussian sampling

We implement DGS for $\mathbb{Z} + \frac{1}{2}c$ with $c \in \{0, 1\}$ that is constant-time over input c and that uses 80 bits of randomness, sufficient for Section 5.3.1. Almost half of the time of Sign is spent on sampling.

This sampler uses a precomputed reverse cumulative distribution table scaled by a factor of 2^{78} similar to the case of Falcon [PFH+22,

Section 3.9.3]. To have a constant-time sampler, independent of the centre c and the outcome, the program reads the two cumulative distribution tables fully. Algorithm 5.4 explains the sampler in pseudocode. The algorithm uses a table where `RCDT[c][i]` contains the value

$$\left\lceil 2^{78} \cdot \Pr_{x \leftarrow \mathcal{D}_{2\mathbb{Z}+c,2\sigma_{\mathsf{sign}}}} [|x| \geq 2i+2] \right\rfloor.$$

Algorithm 5.4. SamplerZ(c): sampler for $2\mathbb{Z}+c$ with std. dev. $2\sigma_{\mathsf{sign}}$

Input: `RCDT` containing two reverse cumulative distribution tables for support $2\mathbb{Z}+c$ with $c \in \{0,1\}$
Output: a sample taken from a distribution close to $\mathcal{D}_{2\mathbb{Z}+c,2\sigma_{\mathsf{sign}}}$

$u \leftarrow \text{UniformBits}(78)$
$z_0 \leftarrow 0$
for `i` $= 0, 1, \ldots, 12$ **do**
    $z_0 \leftarrow z_0 + [\![u < (1-c) \cdot \texttt{RCDT[0][i]} + c \cdot \texttt{RCDT[1][i]}]\!]$
$z_0 \leftarrow 2 \cdot z_0 + c$
$z_0 \leftarrow z_0 - 2z_0 \cdot \text{UniformBits}(1)$
return z_0

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 $i \in \{5, \ldots, 12\}$.

5.5.3 Signature verification

The implementation for signature verification follows Algorithm 5.3 closely, but computes the norm with respect to $\mathbf{Q}$ efficiently.

Moreover, a signature s_1 is decompressed, i.e., s_0 is recovered. This requires a division with rounding to the closest integral point in R,

which the FFT can do efficiently. One can even do this using fixed-point arithmetic: it is highly unlikely that the numerical error yields an incorrect rounding. Especially, when we require that the quantity in Equation (5.3) has to be in $(-0.49, 0.49)^n$, an absolute error of 0.01 is tolerated.

AVX2 optimised implementation

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 Equation (5.2), 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 $\|(t_0, t_1)^{\mathsf{T}}\|_{\mathbf{Q}}^2 \leq 8n \cdot \sigma_{\mathsf{ver}}^2$. 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.

To compute the norm with respect to $\mathbf{Q}$, we use $\|\mathbf{t}\|_{\mathbf{Q}}^2 = \frac{1}{n}\operatorname{Tr}(\mathbf{t}^\star\mathbf{Q}\mathbf{t})$ as we have $\operatorname{Tr}(x) = \sum_\sigma \sigma(x)$ for $x \in R$, 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 $\mathbf{t}^\star\mathbf{Q}\mathbf{t}$ under the embeddings. For example, we have

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

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 an `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 FFTs and Equation (5.2), but 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 Equation (5.2) are so small that rounding to an incorrect s_0 is highly unlikely. By the parameters in Section 5.4.1, it is extremely unlikely (probability less than 2^{-105}) that s_0 is recovered incorrectly with Equation (5.2). 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 Equation (5.2) 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 key pair, none failed with this fixed-point FFT recovery of s_0 both for Hawk-512 and Hawk-1024. Note that fixed-point arithmetic improves the performance of the reference implementation significantly: using floating-points, verification is done in 945 µs and 2095 µs for Hawk-512 and Hawk-1024 respectively.

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 modular 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 all invalid signatures will be rejected.

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

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

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

For Hawk-1024, the nonconstant 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, KeyGen 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 [LM00, (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 invalid signatures are always rejected 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.

5.6 Formal reductions

In this section, we consider two formal reductions related to Hawk. These reductions provide confidence in the design of Hawk, whereas the cryptanalysis in Section 5.3 provides confidence in security of Hawk when using the parameters from Section 5.4.1.

Section 5.6.1 discusses a worst case to average case reduction, and its relation to Hawk. Section 5.6.2 explains why Hawk does not fit in the PSF framework [GPV08]. Section 5.6.3 reduces the strong signature forgery security of Hawk's design to so-called omSVP.

5.6.1 Module LIP self reduction

Our goal in this section, is to sketch a worst case to average reduction for the search module Lattice Isomorphism Problem (smLIP), which underlies private key recovery in Hawk, see 5.3.2. However, this reduction will use a different average case distribution than the distribution of $\mathbf{Q}$ output by Algorithm 5.1, and requires for an efficient reduction that σ_{pk} grows exponentially in n, which is larger than what is chosen in Section 5.4.1.

Specifically, in this section, we will consider a key generation algorithm with the following changes compared to Algorithm 5.1:

- certain conditions in KeyGen that cause restarts, are omitted;
- instead of TowerSolveI on Line 5, we use HermiteSolve, which successfully completes a basis if and only if it is possible to do so;
- we pick $\sigma_{\mathsf{pk}} = 2^{\Theta(n)}$ to make the reduction efficient. This value is larger than in Section 5.4.1.

The goal of this section is to define the worst case and average case versions of smLIP, and to show that an adversary against the average case can be transformed into a similar adversary against the worst case. From this, one may conclude that smLIP is as hard for a worst-case instance as an average case instance. However, the average case distribution does not match the output distribution of KeyGen, because it requires σ to be much larger than the chosen σ_{pk} and its support is much larger.

Worst-case search module LIP

In this section, we generalise LIP to module lattices.

Recall, for a CM-field $\mathbb{K}$ and $\ell \in \mathbb{Z}_{\geq 1}$, the set of matrices representing Hermitian positive definite forms, as

$$\mathcal{H}_\ell^{>0}(\mathbb{K}) = \left\{ \mathbf{Q} \in \mathbb{K}^{\ell\times\ell} \;\middle|\; \mathbf{Q} = \mathbf{Q}^\star \text{ and } \forall \mathbf{v} \in \mathbb{K}^\ell \setminus \{\mathbf{0}\}\colon \operatorname{Tr}(\mathbf{v}^\star \mathbf{Q} \mathbf{v}) > 0 \right\}.$$

This set is the generalisation of quadratic forms to the module setting. Quadratic forms $\mathbf{Q}, \mathbf{Q}'$ are considered equivalent if there exists $\mathbf{U} \in \mathrm{GL}_n(\mathbb{Z})$ such that $\mathbf{Q}' = \mathbf{U}^\mathsf{T}\mathbf{Q}\mathbf{U}$. The natural translation to Hermitian forms 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

$$[\mathbf{Q}]_{\mathrm{sl}} = \left\{ \mathbf{Q}' \in \mathcal{H}_\ell^{>0}(\mathbb{K}) \;\middle|\; \exists \mathbf{U} \in \mathrm{SL}_\ell(\mathcal{O}_\mathbb{K})\colon \mathbf{Q}' = \mathbf{U}^\star \cdot \mathbf{Q} \cdot \mathbf{U} \right\}.$$

Throughout we implicitly restrict to $\mathbb{K}$ that are CM fields, e.g. all cyclotomic fields. Using the Gentry–Szydlo algorithm [GS02] and its generalisations [Kir16, LS17], solving LIP under both actions is equivalent for such fields. Namely, if we have $\mathbf{Q}' = \mathbf{U}^\star\mathbf{Q}\mathbf{U}$, the Gentry–Szydlo algorithm can find $u \in \mathcal{O}_\mathbb{K}$ such that $u^\star u = \det(\mathbf{Q}')/\det(\mathbf{Q})$. Then, we have $\mathbf{Q}' \in [\mathbf{V}^\star\mathbf{Q}\mathbf{V}]_{\mathrm{sl}}$ where $\mathbf{V} = \begin{pmatrix} u & 0 \\ 0 & \mathbf{I}_{\ell-1} \end{pmatrix}$, so equivalence under $\mathrm{GL}_\ell(\mathcal{O}_\mathbb{K})$ reduces to equivalence under $\mathrm{SL}_\ell(\mathcal{O}_\mathbb{K})$.

Definition 5.2 (Worst-case smLIP)**.** Given rank $\ell \in \mathbb{Z}_{\geq 2}$, a CM-field $\mathbb{K}$, and $\mathbf{Q} \in \mathcal{H}_\ell^{>0}(\mathbb{K})$, an instance of $\text{wc-smLIP}^{\mathbf{Q}}_{\mathbb{K},\ell}$, or the worst-case search module Lattice Isomorphism Problem, is any $\mathbf{Q}' \in [\mathbf{Q}]_{\text{sl}}$. A solution is any $\mathbf{U} \in \text{SL}_\ell(\mathcal{O}_\mathbb{K})$ for which we have

$$\mathbf{Q}' = \mathbf{U}^\star \mathbf{Q} \mathbf{U}.$$

Note a solution is unique up to automorphism of $\mathbf{Q}$.

Average case search module LIP

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 cyclotomic fields $\mathbb{K} = \mathbb{Q}(\zeta_{2n})$ having conductor $2n$, where ζ_{2n} satisfies $(\zeta_{2n})^n + 1 = 0$.

We define our average case distribution for any class $[\mathbf{Q}]_{\text{sl}}$, but note that the average case distribution over $[\mathbf{I}_2(\mathbb{K})]_{\text{sl}}$ relates to our key generation in Algorithm 5.1. Finally, we define our average case distribution $\mathsf{AC}_{\mathbb{K},\sigma}([\mathbf{Q}]_{\text{sl}})$ algorithmically, see Algorithm 5.7. 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 $[\mathbf{Q}]_{\text{sl}}$. One can think of $\sigma = \sigma_{\mathsf{pk}}$ in the case of Hawk.

Algorithm 5.5. HermiteSolve$(\mathbb{K}, f, g)$: basis completion (if possible)

Input: cyclotomic field $\mathbb{K}$, and $f, g \in \mathcal{O}_\mathbb{K}$
Output: completion $F, G \in \mathcal{O}_\mathbb{K}$ s.t. $fG - gF = 1$ if these exist, else $\perp$

1: $\mathbf{X} \leftarrow [\text{Rot}(f), \text{Rot}(g)]$
2: compute $\mathbf{U} \in \text{GL}_{2n}(\mathbb{Z})$ such that $\mathbf{X} \cdot \mathbf{U}$ is in HNF
3: **if** $\mathbf{X} \cdot \mathbf{U} \neq [\mathbf{I}_n(\mathbb{Z}), \mathbf{0}^{n \times n}]$ **then**
4: **return** $\perp$
5: parse first column of $\mathbf{U}$ as $\begin{pmatrix} \text{Vec}(G) \\ -\text{Vec}(F) \end{pmatrix}$ with $F, G \in \mathcal{O}_\mathbb{K}$
6: **return** $\begin{pmatrix} F \\ G \end{pmatrix}$

Algorithm 5.5, upon input a vector $\mathbf{b}_1 \in \mathcal{O}_\mathbb{K}^2$, computes and outputs a vector $\mathbf{b}_2 \in \mathcal{O}_\mathbb{K}^2$ such that $\det([\mathbf{b}_1, \mathbf{b}_2]) = 1$ holds whenever this is

possible. Otherwise, the algorithm should output $\perp$, indicating it failed. This Algorithm is a subroutine of Algorithm 5.1 to complete a basis with a second basis vector after its first basis vector is sampled.

We also define Reduce with respect to the form $\mathbf{Q}$ in Algorithm 5.6. This is a simpler, but less efficient, version of ffNP used in Algorithm 5.1. 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 5.6. Reduce$(\mathbf{Q}, \mathbf{y}_1, \mathbf{y}_2)$: size reduction of $\mathbf{y}_2$ w.r.t. $\mathbf{Q}$ and $\mathbf{y}_1$

Input: cyclotomic field $\mathbb{K}$, and $\mathbf{y}_1, \mathbf{y}_2 \in \mathcal{O}_{\mathbb{K}}$, $\mathbf{Q} \in \mathcal{H}_2^{>0}(\mathbb{K})$
Output: canonical $\mathbf{y}_2'$ size reduced with respect to $\mathbf{Q}$ and $\mathbf{y}_1$

1: $\nu \leftarrow \left\lceil \frac{\mathbf{y}_1^\star \mathbf{Q} \mathbf{y}_2}{\mathbf{y}_1^\star \mathbf{Q} \mathbf{y}_1} \right\rfloor \quad \in \mathcal{O}_{\mathbb{K}}$
2: **return** $\mathbf{y}_2 - \nu \mathbf{y}_1$

The reduction in Reduce is canonical in the following way.

Lemma 5.3. *Let* $\mathbf{Q}' = \mathbf{U}^\star \mathbf{Q} \mathbf{U}$ *for* $\mathbf{Q} \in \mathcal{H}_2^{>0}(\mathbb{K})$ *and* $\mathbf{U} \in \mathrm{SL}_2(\mathcal{O}_{\mathbb{K}})$*, and suppose* $\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_2' \in \mathcal{O}_{\mathbb{K}}^2$ *satisfy* $\det([\mathbf{y}_1, \mathbf{y}_2]) = \det([\mathbf{y}_1', \mathbf{y}_2']) = 1$*, where* $\mathbf{y}_1' = \mathbf{U}^{-1}\mathbf{y}_1$*. Then, we have*

$$\textsc{Reduce}(\mathbf{Q}', \mathbf{y}_1', \mathbf{y}_2') = \mathbf{U}^{-1} \cdot \textsc{Reduce}(\mathbf{Q}, \mathbf{y}_1, \mathbf{y}_2).$$

Proof. First, let us write $\begin{pmatrix} f & F \\ g & G \end{pmatrix} = \mathbf{U}^{-1} [\mathbf{y}_1, \mathbf{y}_2]$ and $\begin{pmatrix} F' \\ G' \end{pmatrix} = \mathbf{y}_2'$. Note that we have $fG - gF = fG' - gF' = 1$. One readily verifies $F' = F + \lambda \cdot f$ and $G' = G + \lambda \cdot g$ holds for $\lambda = GF' - FG' \in \mathcal{O}_{\mathbb{K}}$. Indeed,

$$F + \lambda f = F(\underbrace{1 - fG'}_{-gF'}) + GfF' = F'(\underbrace{-gF + fG}_{1}) = F',$$

and similarly $G' = G + \lambda g$.

We have $\mathbf{y}_2' = \mathbf{U}^{-1}\mathbf{y}_2 + \lambda \mathbf{y}_1'$, so $\mathbf{y}_2'$ and $\mathbf{U}^{-1}\mathbf{y}_2$ are size reduced to the same output, say $\mathbf{z}' = \textsc{Reduce}(\mathbf{Q}', \mathbf{y}_1', \mathbf{y}_2') = \textsc{Reduce}(\mathbf{Q}', \mathbf{y}_1', \mathbf{U}^{-1}\mathbf{y}_2)$. By writing $\mathbf{z} = \textsc{Reduce}(\mathbf{Q}, \mathbf{y}_1, \mathbf{y}_2)$, we see that $\mathbf{U}^{-1}\mathbf{z}$ is size reduced

with respect to $\mathbf{Q}'$ and $\mathbf{y}'_1$. Indeed,

$$\left\lceil \frac{\mathbf{y}_1'^{\star} \cdot \mathbf{Q}' \cdot (\mathbf{U}^{-1}\mathbf{z})}{\mathbf{y}_1'^{\star} \cdot \mathbf{Q}' \cdot \mathbf{y}'_1} \right\rfloor = \left\lceil \frac{(\mathbf{U}^{-1}\mathbf{y}_1)^{\star} \cdot \mathbf{Q}' \cdot (\mathbf{U}^{-1}\mathbf{z})}{(\mathbf{U}^{-1}\mathbf{y}_1)^{\star} \cdot \mathbf{Q}' \cdot (\mathbf{U}^{-1}\mathbf{y}_1)} \right\rfloor = \left\lceil \frac{\mathbf{y}_1^{\star} \cdot \mathbf{Q} \cdot \mathbf{z}}{\mathbf{y}_1^{\star} \cdot \mathbf{Q} \cdot \mathbf{y}_1} \right\rfloor = 0.$$

We conclude $\mathbf{z}' = \mathbf{U}^{-1}\mathbf{z}$. □

Algorithm 5.7. $\text{AC}_{\mathbb{K},\sigma}(\mathbf{Q})$: sampler for $\mathsf{AC}_{\mathbb{K},\sigma}([\mathbf{Q}]_{\text{sl}})$

Input: cyclotomic field $\mathbb{K}$, parameter σ and $\mathbf{Q} \in \mathcal{H}_2^{>0}(\mathbb{K})$
Output: $(\mathbf{R}, \mathbf{Y}) \in [\mathbf{Q}]_{\text{sl}} \times \text{SL}_2(\mathcal{O}_{\mathbb{K}})$, where $\mathbf{R} = \mathbf{Y}^{\star}\mathbf{Q}\mathbf{Y}$

1: $\begin{pmatrix} f \\ g \end{pmatrix} = \mathbf{y}_1 \leftarrow D_{\mathbf{Q},\sigma}$ where $f, g \in \mathcal{O}_{\mathbb{K}}$
2: $\mathbf{y}_2 \leftarrow \text{HermiteSolve}(\mathbb{K}, f, g)$
3: **if** $\mathbf{y}_2 = \perp$ **then**
4: **restart**
5: $\mathbf{y}_2 \leftarrow \text{Reduce}(\mathbf{Q}, \mathbf{y}_1, \mathbf{y}_2)$
6: $\mathbf{Y} \leftarrow [\mathbf{y}_1, \mathbf{y}_2]$
7: $\mathbf{R} \leftarrow \mathbf{Y}^{\star} \cdot \mathbf{Q} \cdot \mathbf{Y}$
8: **return** $(\mathbf{R}, \mathbf{Y})$

The following lemma ensures that a sample $\mathbf{R} \in [\mathbf{Q}]_{\text{sl}}$ output by Algorithm 5.7 only depends on the class $[\mathbf{Q}]_{\text{sl}}$, and not on the input representative $\mathbf{Q}$. As a result the distribution $\mathsf{AC}_{\mathbb{K},\sigma}([\mathbf{Q}]_{\text{sl}})$ is well-defined.

Lemma 5.4. *For any power of two cyclotomic* $\mathbb{K}$, $\mathbf{Q} \in \mathcal{H}_2^{>0}(\mathbb{K})$, $\mathbf{Q}' \in [\mathbf{Q}]_{\text{sl}}$ *and* $\sigma > 0$ *the distributions of* $(\mathbf{R}, \cdot) \leftarrow \text{AC}_{\mathbb{K},\sigma}(\mathbf{Q})$ *and* $(\mathbf{R}, \cdot) \leftarrow \text{AC}_{\mathbb{K},\sigma}(\mathbf{Q}')$ *are equal.*

Proof. For $\mathbf{Q}' \in [\mathbf{Q}]_{\text{sl}}$ there exists a $\mathbf{U} \in \text{SL}_2(\mathcal{O}_{\mathbb{K}})$ such that $\mathbf{Q}' = \mathbf{U}^{\star} \cdot \mathbf{Q} \cdot \mathbf{U}$. It is sufficient to show that for any $\mathbf{Y}$ created during $\text{AC}_{\mathbb{K},\sigma}(\mathbf{Q})$, $\mathbf{Y}' = \mathbf{U}^{-1} \cdot \mathbf{Y}$ is created within $\text{AC}_{\mathbb{K},\sigma}(\mathbf{Q}')$ with the same probability. Having shown this, since $\mathbf{Y}^{\star} \cdot \mathbf{Q} \cdot \mathbf{Y} = (\mathbf{Y}')^{\star} \cdot \mathbf{Q}' \cdot \mathbf{Y}'$, the two distributions for $\mathbf{R}$ 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 $[\mathbf{Q}]_{\text{sl}}$, the probability of sampling $\mathbf{y}'_1 \leftarrow D_{\mathbf{Q}',\sigma}$ is equal to the probability of sampling $\mathbf{y}_1 \leftarrow D_{\mathbf{Q},\sigma}$.

Now, since $\mathbf{U}^{-1}\mathbf{y}_2$ is a way to complete $\mathbf{y}'_1$, Algorithm 5.5 will succeed and complete to some $\mathbf{y}'_2$. After Algorithm 5.6 has reduced $\mathbf{y}'_2$, Lemma 5.4 shows that $\mathbf{y}'_2 = \mathbf{U}^{-1}\mathbf{y}_2$. Hence, $\mathbf{Y}' = \mathbf{U}^{-1} \cdot \mathbf{Y}$. □

We can now define the average case version of module LIP.

Definition 5.5 (Average case smLIP)**.** Given some power of two cyclotomic $\mathbb{K}$ and $\mathbf{Q} \in \mathcal{H}_2^{>0}(\mathbb{K})$ an instance of $\text{AC-SMLIP}^{\mathbf{Q}}_{\mathbb{K},\sigma}$, the average case search module Lattice Isomorphism Problem, is given by $\mathbf{Q}$ and an element $\mathbf{Q}' \in [\mathbf{Q}]_{\text{sl}}$ sampled from $\mathsf{AC}_{\mathbb{K},\sigma}([\mathbf{Q}]_{\text{sl}})$. A solution is $\mathbf{U} \in \text{SL}_2(\mathcal{O}_{\mathbb{K}})$ such that $\mathbf{Q}' = \mathbf{U}^\star \cdot \mathbf{Q} \cdot \mathbf{U}$.

Reducing worst to average case

We expect that $\mathsf{AC}_{\mathbb{K},\sigma}([\mathbf{Q}]_{\text{sl}})$ becomes harder as the parameter σ increases. In fact, following [DvW22], if σ is large enough we have equivalence with the corresponding worst-case problem, assuming HermiteSolve does not fail too often.

Lemma 5.6 (Worst case to average case)**.** *Given a machine that can solve* $\text{AC-SMLIP}^{\mathbf{Q}}_{\mathbb{K},\sigma}$ *in time* T *with probability* $\varepsilon > 0$*, for some* $\sigma \geq 2^{\Theta(n)} \cdot \lambda_{2n}([\text{Rot}(\mathbf{Q})])$*, one may solve* $\text{WC-SMLIP}^{\mathbf{Q}}_{\mathbb{K},\ell}$ *in expected time*

$$T + \mathbb{E}_{\mathsf{samples}} \cdot \mathsf{poly}(n, \log \sigma)$$

with probability ε*. Here* $\mathbb{E}_{\mathsf{samples}}(n, \sigma) \geq 1$ *is the expected number of times* $(f, g)^{\mathsf{T}}$ *is (re)sampled in Algorithm 5.7.*

Proof. As input, we receive $\mathbf{Q} \in \mathcal{H}_2^{>0}(\mathbb{K})$ and a worst-case instance $\mathbf{Q}' \in [\mathbf{Q}]_{\text{sl}}$. By first LLL reducing $\text{Rot}(\mathbf{Q}) \in \mathbb{Q}^{2n \times 2n}$, we can sample efficiently from $D_{\mathbf{Q},\sigma}$ via Lemma 2.81. Thus, we can sample $(\mathbf{Q}'', \mathbf{U}'') \leftarrow \text{AC}_{\mathbb{K},\sigma}(\mathbf{Q})$ in time $\mathbb{E}_{\mathsf{samples}} \cdot \mathsf{poly}(n, \sigma)$ by Algorithm 5.7. The sample $\mathbf{Q}''$ is distributed as $\mathsf{AC}_{\mathbb{K},\sigma}([\mathbf{Q}]_{\text{sl}}) = \mathsf{AC}_{\mathbb{K},\sigma}([\mathbf{Q}']_{\text{sl}})$, and we have $\mathbf{Q}'' = \mathbf{U}''^\star \mathbf{Q} \mathbf{U}''$. Now $\mathbf{Q}''$ is an average case instance of $\text{AC-SMLIP}^{\mathbf{Q}'}_{\mathbb{K},\sigma}$, which the machine can solve in time T with probability ε. On success, we obtain some $\mathbf{U}' \in \text{SL}_2(\mathcal{O}_{\mathbb{K}})$ such that $\mathbf{Q}'' = \mathbf{U}'^\star \mathbf{Q}' \mathbf{U}'$, and $\mathbf{U} = \mathbf{U}''(\mathbf{U}')^{-1}$ gives a solution to the worst-case instance, i.e., $\mathbf{Q}' = \mathbf{U}^\star \mathbf{Q} \mathbf{U}$. □

Following the same argument as [DvW22, Lem 3.10] we may reduce σ in the reduction at the expense of an additive loss in runtime from the cost of stronger lattice reduction. There is a heuristic explanation and matching experimental evidence for why HermiteSolve fails with probability around 25% [DPPvW22b, App. A], which gives $\mathbb{E}_{\mathsf{samples}} \approx 4/3$, and thus the reduction above is in fact efficient.

Relation to Hawk key generation

There are multiple algorithmic differences between the average case distribution output by $\mathrm{AC}_{\mathbb{K},\sigma}(\mathbf{I}_2)$ and the public key output by KeyGen.

First, KeyGen uses a structured variant of Babai's Nearest-Plane algorithm to size reduce (F, G), whereas $\mathrm{AC}_{\mathbb{K},\sigma}(\mathbf{I}_2)$ uses a variant of Rounding-Off. This poses no notable issues, as the reduction on a Hermitian form $\mathbf{R}$ output by KeyGen can be reduced using Rounding-Off, and vice versa. Explicitly, given a Hermitian form $\mathbf{Q} \in \mathcal{H}_2^{>0}(\mathbb{K})$, the quadratic form $\mathbf{Q}' = \mathbf{U}^\star\mathbf{Q}\mathbf{U}$ is size reduced (using Rounding-Off), where

$$\mathbf{U} = \begin{pmatrix} 1 & -\lceil \mathbf{Q}_{01}/\mathbf{Q}_{00} \rfloor \\ 0 & 1 \end{pmatrix} .$$

Second, KeyGen in Algorithm 5.1 has more restart conditions because it uses TowerSolveI instead of HermiteSolve. Let p_r denote the probability that Algorithm 5.1 restarts when it internally samples a completable $(f, g)^\mathsf{T}$. If $\mathcal{A}$ is an adversary against Hawk's KeyGen using $\sigma = \sigma_{\mathsf{pk}}$, i.e., a machine that can return $\mathbf{B} \in \mathrm{SL}_2(\mathcal{O}_\mathbb{K})$ such that $\mathbf{B}^\star\mathbf{B} = \mathbf{Q}$ for a Hawk public key $\mathbf{Q}$ in time t and with probability ε, then $\mathcal{A}$ is an adversary for $\mathsf{AC}_{\mathbb{K},\sigma}([\mathbf{I}_2(\mathcal{O}_\mathbb{K})]_{\mathrm{sl}})$ that runs in time t with probability at least $(1 - p_r) \cdot \varepsilon$. There is experimental and theoretical evidence that the component of p_r, caused by the use of TowerSolveI and the requirement on $\mathrm{N}(f), \mathrm{N}(g)$ to be odd in Algorithm 5.1, is a constant [DPPvW22b, App. A]. Although this reduction from $\mathsf{AC}_{\mathbb{K},\sigma}([\mathbf{I}_2]_{\mathrm{sl}})$ to Hawk's KeyGen is achievable for $\sigma = \sigma_{\mathsf{pk}}$, note that the composed reduction from worst case to Hawk's KeyGen requires σ_{pk} to be larger than chosen in Section 5.4.1. Therefore, there is no efficient reduction possible from worst-case smLIP to the average case key recovery of Hawk KeyGen.

Last, as Remark 5.1 indicates, Hawk uses a centred binomial distribution (CBD) to sample f, g. In this case, the average case output by $\mathrm{AC}_{\mathbb{K},\sigma}(\mathbf{I}_2)$ differs significantly from the public keys output by KeyGen. For example, while the discrete Gaussian sampler assigns the same probability mass to (f, g) of the same ℓ_2 norm, CDB does not. We therefore cannot claim any worst case to average case reduction for the problem of recovering the secret key of Hawk from the public key.

5.6.2 Inapplicability of PSF framework

In this section, we introduce the framework of Preimage Sampleable Functions (PSF) [GPV08, Sec. 5.2]. Falcon uses this framework to show it is SUF-CMA secure, assuming it is hard to find an inhomogeneous short integer solution (ISIS) in an NTRU lattice [PFH+22, Sec. 2.2]. We then explain why Hawk does not fit in the framework.

Definition 5.7. The *PSF framework*, or *trapdoor collision-resistant hash functions with preimage sampling*, is a tuple of probabilistic polynomial-time algorithms (TrapGen, SampleDom, SamplePre) satisfying the following properties.

Trapdoor generation: The output $(a, t) \leftarrow \text{TrapGen}(1^n)$, consists of a (public) description a that allows one to efficiently compute a function $f_a \colon D_n \to R_n$, where D_n and R_n are some efficiently recognisable domain and range, respectively, and t is (private) trapdoor information for f_a.

Domain sampling: Let $(a, t) \leftarrow \text{TrapGen}(1^n)$. Then, output from $\text{SampleDom}(1^n)$ follows a distribution on D_n, not necessarily uniform, such that $f_a(x)$ for $x \leftarrow \text{SampleDom}(1^n)$ is the uniform distribution on R_n.

Trapdoor preimage sampling: Let $(a, t) \leftarrow \text{TrapGen}(1^n)$. For all $y \in R_n$, output from $\text{SamplePre}(t, y)$ follows a distribution on $f_a^{-1}(y)$ that is statistically indistinguishable from $x \leftarrow \text{SampleDom}(1^n)$ such that $f_a(x) = y$.

Moreover, for a given $y \in R_n$ the min-entropy of $x \leftarrow \text{SampleDom}(1^n)$ such that $f_a(x) = y$ holds, should be sufficient, and it should be hard to find a collision for f_a, i.e., distinct $x_1, x_2 \in D_n$ satisfying $f_a(x_1) = f_a(x_2)$.

We now consider the example of a hash-and-sign signature scheme that is based on the hardness of ISIS for q-ary lattices.

Hash-and-sign for q-ary lattices

We obtain a hash-and-sign signature scheme [GPV08, Sec. 6] based on the hardness of ISIS by using Ajtai's algorithm [Ajt99], for trapdoor generation: $(a, \mathbf{T}) \leftarrow \text{TrapGen}(1^n)$, where $a = (\mathbf{B}, \mathbf{A}) \in \mathbb{Z}^{m \times m} \times \mathbb{Z}^{n \times m}$. Here, $\mathbf{T} \in \mathbb{Z}^{m \times m}$ is a good trapdoor basis for the q-ary lattice $\Lambda =$

$\mathcal{L}_q^\perp(\mathbf{A})$ using Equation (2.5), and $\mathbf{B}$ is a bad basis for Λ. The domain is $D_n = \mathbb{Z}^m \cap (\sigma\sqrt{2\pi m} \cdot \mathcal{B}^m)$, and the range is $R_n = (\mathbb{Z}/q\mathbb{Z})^n$. We have

$$f_a \colon D_n \to R_n, \qquad \mathbf{x} \mapsto \mathbf{x}\mathbf{A} \pmod q \ ,$$

and $\textsc{SampleDom}(1^n) = \mathcal{D}_{\mathbb{Z}^m,\sigma}$. The algorithm $\textsc{SamplePre}(\mathbf{T}, \mathbf{y})$ first constructs arbitrary $\mathbf{x}_0 \in R_n$ such that $\mathbf{x_0}\mathbf{A} = \mathbf{y}$, and then outputs a sample from $\mathcal{D}_{\mathbf{T}\mathbb{Z}^m+\mathbf{x}_0,\sigma}$ using $\mathbf{T}$. The hash-and-sign signature scheme then uses a hash function $\mathsf{H} \colon \{0,1\}^* \to R_n$, and a valid signature on a message m is an element $\mathbf{x} \in D_n$ such that $f_a(\mathbf{x}) = \mathsf{H}(\mathsf{m})$.

Intuitively, it is easy for the public to generate $\mathbf{y}$ being the sum of a lattice point $\mathbf{x} \in \Lambda$ and a small error $\mathbf{e} \in D_n$, whereas not knowing the good basis $\mathbf{T}$ makes it hard to decode a particular $\mathbf{y} = \mathbf{x} + \mathbf{e}$ to its nearest lattice point $\mathbf{x}$.

The private key leakage from a signature transcript [NR09, DN12b] does not apply to this hash-and-sign signature scheme, because the list $(x_i)_{i=1}^{q_s}$ acquired from a signature transcript $(\mathsf{m}_i, x_i)_{i=1}^{q_s}$ is statistically indistinguishable from i.i.d. samples $x_i \leftarrow \textsc{SampleDom}(1^n)$ by definition of trapdoor preimage sampling.

Note, however, "for the security reduction to work, the signer must give out *at most one* preimage of a given point" [GPV08, Sec. 6]. Namely, if this is not the case, an adversary may obtain two *distinct* signatures $\mathbf{x}_1, \mathbf{x}_2$ on message m, i.e.,

$$f_a(\mathbf{x}_1) = f_a(\mathbf{x}_2) = \mathsf{H}(\mathsf{m}) \ ,$$

and subsequently, they will learn a short lattice vector $\mathbf{x}_1 - \mathbf{x}_2$ of norm at most $\sigma\sqrt{8\pi m}$, which solves SIS, the homogeneous variant of ISIS. The mentioned condition can be achieved either by adding sufficient so-called salt to messages before hashing, or by letting the signing algorithm return a cached signature when asked to sign a message for the second time.

Hash-and-sign for Hawk

The signature distribution is fundamentally different for Hawk. If one has the public key $\mathbf{Q}$, one can publicly sample from $\mathcal{D}_{\mathbf{Q},\mathbb{Z}^{2n},\sigma}$ when

$$\sigma \geq \sigma_{\mathsf{pk}}\sqrt{2n} \cdot (1/\pi) \cdot \sqrt{\log(2n+4)/2},$$

using Lemma 2.81. An average sample $\mathbf{x} \leftarrow \mathcal{D}_{\mathbf{Q},\mathbb{Z}^{2n},\sigma}$ has a squared $\mathbf{Q}$-norm of $2n\sigma^2 \geq (2n)^2\sigma_{\mathsf{pk}}^2 \log(2n+4)/(2\pi^2)$. On the other hand,

valid signatures provide lattice points of much smaller $\mathbf{Q}$-norm. Indeed, if anyone with a public key $\mathbf{Q}$ obtains a valid signature $\mathbf{s}$ on hashed message $\mathbf{h} = \mathsf{H}(\mathsf{m}) \in \{0,1\}^{2n}$, then they learn the lattice point $\mathbf{x} = \mathbf{h} - 2\mathbf{s} \in \mathbb{Z}^{2n}$ of average squared $\mathbf{Q}$-norm $8n\sigma_{\mathsf{sign}}^2$, which is a factor at least n smaller than the public samples for the parametrisations in Section 5.4.1.

In particular, the natural analogue of the PSF framework for q-ary lattices to Hawk is setting as domain $D_n = \{\mathbf{x} \in \mathbb{Z}^{2n} : \|\mathbf{x}\|_{\mathbf{Q}} \leq \sigma_{\mathsf{sign}}\sqrt{8n}$, and range $R_n = \{0,1\}^{2n}$, and $f_a \colon \mathbf{x} \mapsto \mathbf{x} \bmod 2$. However, there is no way to instantiate SampleDom in this case, as one cannot sample short enough vectors. Moreover, it appears that sampling from D_n is hard. This is why we introduce a new hardness assumption to base security on.

5.6.3 One more shortest vector 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 $\mathcal{H}_2^{>0}(\mathbb{K})$ and gives Gaussian samples according to this $\mathbf{Q}$. If one can find some nonzero $\mathbf{x}$ that is sufficiently short with respect to $\mathbf{Q}$, and that is in some sense nontrivially new, then one solves the problem. We show that for certain parameters, if one can forge a signature against 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.

Security definitions

We first define the ROM-SUF-CMA game for signature schemes. Here Sign and Verify are given 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 5.6.

Definition 5.8 (ROM-SUF-CMA security)**.** We say that a signature scheme $\Pi = (\mathrm{KeyGen}, \mathrm{Sign}, \mathrm{Verify})$ is $(t, \varepsilon, q_s, q_h)$-*ROM-SUF-CMA secure*, or strongly unforgeable under chosen message attacks in the

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

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

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

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

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

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

Figure 5.6: ROM-SUF-CMA game

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, we have

$$\Pr[\text{ROM-SUF-CMA}_{\Pi,\mathcal{A}} = 1] \leq \varepsilon.$$

Trivial automorphisms

In this section, we define a set of automorphisms of $\text{Vec}(R^2)$, which give rise to 'trivial' wins in our omSVP game. For example, negation preserves the inner product with respect to any Hermitian form: $\langle \mathbf{x}, \mathbf{y} \rangle_{\mathbf{Q}} = \langle -\mathbf{x}, -\mathbf{y} \rangle_{\mathbf{Q}}$.

Definition 5.9. Given a number field $\mathbb{K}$, let $\mu_{\mathbb{K}}$ be the set of roots of unity.

Lemma 5.10. *For $\alpha \in R$ we have $\alpha^\star \alpha = 1 \iff \alpha \in \mu_{\mathbb{K}}$.*

Proof. If $\alpha^\star \alpha = 1$ 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 $\sigma_i(\alpha^\star\alpha) = |\sigma_i(\alpha)|^2$ and noting that $\alpha \in \mu_{\mathbb{K}}$ must have $|\sigma_i(\alpha)| = 1$ else no power of it will equal 1. □

Lemma 5.11. *For any $\ell \in \mathbb{Z}_{\geq 1}$ some $\alpha \in R$ has $\|\alpha\mathbf{x}\|_{\mathbf{Q}} = \|\mathbf{x}\|_{\mathbf{Q}}$ for all $\mathbf{x} \in \mathbb{K}^\ell$ and $\mathbf{Q} \in \mathcal{H}^{>0}_\ell(\mathbb{K})$ if and only if $\alpha \in \mu_{\mathbb{K}}$.*

Proof. First note if $\alpha \in \mu_{\mathbb{K}}$ then by Lemma 5.10,

$$\|\alpha\mathbf{x}\|_{\mathbf{Q}} = \mathrm{Tr}(\alpha^\star\alpha\mathbf{x}^\star\mathbf{Q}\mathbf{x})/n = \mathrm{Tr}(\mathbf{x}^\star\mathbf{Q}\mathbf{x})/n = \|\mathbf{x}\|_{\mathbf{Q}}.$$

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 $\sum_{i=1}^n \sigma_i(\alpha^\star\alpha) = n$. 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 $\sum_{i=1}^n \sigma_i(\alpha^\star\alpha)^2 = n$. 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, $\sigma_i(\alpha^\star\alpha) = 1$ for all i, and $\alpha^\star\alpha = 1$, then $\alpha \in \mu_{\mathbb{K}}$ by Lemma 5.10. □

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

Lemma 5.12. *A power of two cyclotomic field $\mathbb{K}$ of conductor $2n$ has $\mu_{\mathbb{K}} = \{X^i : i = 0, \ldots, 2n-1\}$.*

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

Here $i = 0, n$ represent the identity and negation, respectively.

Identifying X with the matrix $X \cdot \mathbf{I}_2 \in \mathrm{GL}_2(R)$, note that the automorphism X of the Hermitian form $\mathbf{I}_2(R)$ provides the following automorphism of the standard basis $\textsc{Rot}(\mathbf{I}_2(R)) = \mathbf{I}_{2n}(\mathbb{Z})$:

$$\begin{pmatrix} x_0 \\ \vdots \\ x_{2n-1} \end{pmatrix} \mapsto (-x_{n-1}, x_0, x_1, \ldots, x_{n-2},\ -x_{2n-1}, x_n, x_{n+1}, \ldots, x_{2n-2})^\mathsf{T}\ .$$

In fact, X is an automorphism for every Hermitian form, similar to how -1 is an automorphism for every lattice.

Luo, Jiang, Pan and Wang observed that given $\mathbf{Q}$ there is another efficiently computable automorphism $\omega_{\mathbf{Q}}$ of $\textsc{Rot}(\mathbf{Q})$ [LJPW24], which we will now explain. Let us write $\overline{\mathbf{B}} = (\mathbf{B}^\star)^\mathsf{T}$ given a matrix (or vector) $\mathbf{B} \in \mathbb{K}^{k\times\ell}$, which compared to Definition 2.66, applies complex conjugation coefficientwise to $\mathbf{B}$ (not $\mathbf{B}^\mathsf{T}$). First, let

$$\mathbf{J}_2 = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix},$$

and observe $\mathbf{J}_2^2 = -\mathbf{I}_2(R)$. Via Equation (5.1) we relate the primal and dual $\mathbf{D} = (\mathbf{B}^{-1})^\star$ bases

$$\mathbf{D} = (\mathbf{B}^{-1})^\star = (-\mathbf{J}_2\mathbf{B}^\mathsf{T}\mathbf{J}_2)^\star = -\mathbf{J}_2 \cdot \overline{\mathbf{B}} \cdot \mathbf{J}_2. \tag{5.6}$$

Now, suppose it is publicly known that some vector $\mathbf{x}$ is short with respect to $\mathbf{Q}$, i.e., $\|\mathbf{x}\|_{\mathbf{Q}}$ is small. Then, the vector $\mathbf{z} = \mathbf{B}\cdot\mathbf{x}$ is short, i.e., $\|\mathbf{z}\| = \|\mathbf{B}\mathbf{x}\| = \|\mathbf{x}\|_{\mathbf{Q}}$ is small. Observe that conjugation and multiplication by $\mathbf{J}_2$ preserve inner products. Indeed, $\mathbf{z}' = \mathbf{J}_2 \cdot \overline{\mathbf{z}}$ satisfies $\|\mathbf{z}'\| = \|\mathbf{z}\|$. Using Equation (5.6), we get

$$\mathbf{z}' = \mathbf{J}_2 \cdot \overline{\mathbf{B}\mathbf{x}} = \mathbf{J}_2 \cdot \overline{\mathbf{B}} \cdot \overline{\mathbf{x}} = \mathbf{D} \cdot \mathbf{J}_2 \cdot \overline{\mathbf{x}}.$$

As $\mathbf{B}$ is not known, $\mathbf{z}$ and $\mathbf{z}'$ are not publicly known, but it is known that $\mathbf{J}_2\overline{\mathbf{x}}$ is a short vector with respect to the dual Hermitian form $\mathbf{Q}^{-1}$. Exploiting self-duality of the underlying unstructured lattice $\mathbb{Z}^{2n}$, allows us to find a short vector in terms of $\mathbf{Q}$. We express $\mathbf{z}'$ in terms of the primal basis $\mathbf{B}$:

$$\mathbf{z}' = \mathbf{B} \cdot \underbrace{\mathbf{B}^{-1}\mathbf{D}}_{\mathbf{Q}^{-1}} \cdot \mathbf{J}_2\overline{\mathbf{x}} = \mathbf{B} \cdot \mathbf{Q}^{-1} \cdot \mathbf{J}_2\overline{\mathbf{x}}.$$

Therefore, $\mathbf{x}' = \mathbf{Q}^{-1} \cdot \mathbf{J}_2\overline{\mathbf{x}}$ is also a publicly known lattice point of same $\mathbf{Q}$-norm $\|\mathbf{x}'\|_{\mathbf{Q}} = \|\mathbf{x}\|_{\mathbf{Q}}$. This motivates the following definition, which uses that $\mathbb{Z}^{2n}$ is a symplectic lattice [GHN06].

Definition 5.13 (Symplectic automorphism)**.** The so-called *symplectic automorphism* is the additive bijective map,

$$\omega\colon R^2 \to R^2, \qquad \begin{pmatrix} x \\ y \end{pmatrix} \mapsto \begin{pmatrix} y^\star \\ -x^\star \end{pmatrix} .$$

Given a Hermitian form $\mathbf{Q} \in \mathcal{H}_2^{>0}(\mathbb{K})$, the *symplectic automorphism with respect to* $\mathbf{Q}$ is the additive, bijective map

$$\omega_{\mathbf{Q}} \colon R^2 \to R^2, \qquad \mathbf{x} \mapsto \mathbf{Q}^{-1}\omega(\mathbf{x}) = \mathbf{Q}^{-1}\mathbf{J}_2 \cdot \overline{\mathbf{x}} \ .$$

Note that $\omega_{\mathbf{Q}}$ is not an automorphism of $\mathbf{Q}$, because it is not R-linear: $\omega_{\mathbf{Q}}(\alpha\mathbf{x}) = \alpha^\star\omega_{\mathbf{Q}}(\mathbf{x})$ holds for all $\mathbf{x} \in R^2$ and $\alpha \in R$. It is, however, an automorphism of $\textsc{Rot}(\mathbf{Q})$ by considering its representation as a matrix of order $2n$. Therefore, we have $\|\mathbf{x}\|_{\mathbf{Q}} = \|\omega_{\mathbf{Q}}(\mathbf{x})\|_{\mathbf{Q}}$ for all $\mathbf{x} \in R^2$. Using Equation (5.6), we see for any basis $\mathbf{B} \in \mathrm{GL}_2(R)$ and $\mathbf{x} \in R^2$ that

$$\mathbf{B} \cdot \omega_{\mathbf{B}^\star\mathbf{B}}(\mathbf{x}) = (\mathbf{B}^{-1})^\star\mathbf{J}_2\overline{\mathbf{x}} = \mathbf{J}_2\overline{\mathbf{B}\mathbf{x}} = \omega(\mathbf{B}\mathbf{x}). \tag{5.7}$$

This motivates the definition of the following automorphism.

Definition 5.14 (Group of trivial automorphism)**.** Given a Hermitian form $\mathbf{Q} \in \mathcal{H}_2^{>0}(\mathbb{K})$, the *group of trivial automorphisms with respect to* $\mathbf{Q}$ is given by

$$\mathcal{G}_{\mathbf{Q}} = \left\langle X \cdot \mathbf{I}_2(\mathbb{K}), \omega_{\mathbf{Q}} \right\rangle = \bigcup_{i=0}^{2n-1} \left\{ X^i \, \mathbf{I}_2(\mathbb{K}), X^i\omega_{\mathbf{Q}} \right\} \quad \subset \mathrm{O}(\textsc{Rot}(\mathbf{Q})).$$

We simply write $\mathcal{G} = \mathcal{G}_{\mathbf{Q}}$ for $\mathbf{Q} = \mathbf{I}_2(R)$.

This group is isomorphic to the generalised quaternion group Q_{4n}, which has order $4n$. Note, $\omega^2 = -\mathbf{I}_2(R)$. Using Equation (5.7), we get

$$\omega_{\mathbf{Q}}(\omega_{\mathbf{Q}}(\mathbf{x})) = \mathbf{B}^{-1}\omega(\omega(\mathbf{B}\mathbf{x})) = -\mathbf{x}.$$

Average case distribution

We are now ready to define omSVP in its full generality, for certain parameters that will become clear later. Although we focus on rank 2 as $\mathcal{G}_{\mathbf{Q}}$ is only defined for that rank, omSVP could be generalised to other ranks.

Definition 5.15 (Average case omSVP)**.** An *average case omSVP instance* is the pair $\textsc{ac-omSVP} = (\mathsf{Init}, \mathsf{samp})$. On input 1^n, Init returns a form $\mathbf{Q}$ sampled from some distribution over some $\mathcal{H}_2^{>0}(\mathbb{K})$, the roots of unity $\mu_{\mathbb{K}}$ for $\mathbb{K}$, a length bound L_n, and a Gaussian parameter σ_n. On input $\mathbf{Q}$, samp returns a sample from $D_{\mathbf{Q},\sigma_n}$.

$\text{SAMPLE}_{\text{AC-omSVP},\mathcal{A}}(1^n)$

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

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

1: $\mathbf{w} \leftarrow D_{\mathbf{Q},\sigma}$
2: $\mathcal{L}_{\mathsf{samples}} \leftarrow \mathcal{L}_{\mathsf{samples}} \cup \{\alpha(\mathbf{w})\}_{\alpha \in \mathcal{G}_{\mathbf{Q}}}$
3: **return** $\mathbf{w}$

Figure 5.7: SAMPLE game

The adversary $\mathcal{A}$ wins the game in Figure 5.7 whenever it can use the form $\mathbf{Q}$ and the samples it receives from samp to return some nontrivially new element of R^2 that is short enough.

Definition 5.16 (SAMPLE security)**.** Let AC-omSVP $= (\mathsf{Init}, \mathsf{samp})$ be an average case omSVP instance. We say 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, we have

$$\Pr[\text{SAMPLE}_{\text{AC-omSVP},\mathcal{A}} = 1] \leq \varepsilon.$$

Smoothing signatures

In the following, we will implicitly use the particular section $s\colon R/2R \to R$ of the natural surjection $R \to R/2R$ given by $s(h + 2R) = h_0 + h_1 X + \cdots h_{n-1} X^{n-1}$ with $(h_0, \ldots, h_{n-1}) \in \{0,1\}^n$. Thus, reduction modulo 2 can also be considered as a map $R \to R$ via s.

Definition 5.17. Let $\mathbf{B} \in \mathrm{GL}_2(R)$ and $\mathbf{Q} = \mathbf{B}^\star\mathbf{B}$. For every $\sigma > 0$ and $\mathbf{h} \in (R/2R)^2 \subset R^2$ we consider the following three $\mathbf{Q}$-dependent distributions.

- $\mathcal{D}_{\mathbf{Q},\sigma}$, the discrete Gaussian distribution over R^2 under $\|\cdot\|_{\mathbf{Q}}$ given by

$$\mathcal{D}_{\mathbf{Q},\sigma}\colon R^2 \to \mathbb{R}, \quad \mathbf{x} \mapsto \frac{1}{\sum_{\mathbf{y} \in R^2} \rho_{\mathbf{Q},\sigma}(\mathbf{y})} \rho_{\mathbf{Q},\sigma}(\mathbf{x}).$$

- $\widetilde{\mathcal{D}}_{\mathbf{Q},\sigma}[\mathbf{h}]$, the discrete Gaussian distribution over $\mathbf{h} + 2R^2 \subset R^2$ under $\|\cdot\|_{\mathbf{Q}}$ given by

$$\widetilde{\mathcal{D}}_{\mathbf{Q},\sigma}[\mathbf{h}] \colon \mathbf{h} + 2R^2 \to \mathbb{R}, \quad \mathbf{x} \mapsto \frac{1}{\sum_{\mathbf{y} \in \mathbf{h}+2R^2} \rho_{\mathbf{Q},\sigma}(\mathbf{y})} \rho_{\mathbf{Q},\sigma}(\mathbf{x}).$$

- $\widetilde{\mathcal{D}}_{\mathbf{Q},\sigma}$, the distribution implied by sampling a uniform $\mathbf{h} \leftarrow (R/2R)^2$ and returning a sample from $\widetilde{\mathcal{D}}_{\mathbf{Q},\sigma}[\mathbf{h}]$.

Given $\mathbf{B}$ we may sample the above distributions by sampling according to $\mathcal{D}_{\mathbf{I}_2(R),\sigma}$, in the coset defined by $\mathbf{t} = \mathbf{B}\mathbf{h}$ if required, and multiplying by $\mathbf{B}^{-1}$ as in Hawk signing, see Algorithm 5.2. Note that for every $\mathbf{h} \in (R/2R)^2$ and any outcome $\mathbf{w} \leftarrow \widetilde{\mathcal{D}}_{\mathbf{Q},\sigma}[\mathbf{h}]$ we have $\mathbf{w} \equiv \mathbf{h} \pmod 2$. A direct consequence of Lemma 2.84 is the following result, which tells us that $\mathcal{D}_{\mathbf{Q},2\sigma}$ assigns a similar probability to each coset $\mathbf{h} + 2R^2$.

Lemma 5.18. *Let $\sigma \geq \eta_\varepsilon(\mathbb{Z}^{2n})$ for some $\varepsilon \in (0,1)$. Then for all fixed $\mathbf{h} \in (R/2R)^2$ we have*

$$\Pr_{\mathbf{w} \leftarrow \mathcal{D}_{\mathbf{Q},2\sigma}} [\mathbf{w} \equiv \mathbf{h} \pmod 2] \leq 2^{-2n} \cdot \frac{1+\varepsilon}{1-\varepsilon}.$$

We will use the Rényi divergence of $\widetilde{\mathcal{D}}_{\mathbf{Q},2\sigma}$ from $\mathcal{D}_{\mathbf{Q},2\sigma}$, which can be computed using following lemma.

Lemma 5.19. *Let $\sigma > 0$, $\mathbf{Q} = \mathbf{B}^\star\mathbf{B}$ for some $\mathbf{B} \in \mathrm{GL}_2(R)$ and $a \in (1,\infty)$ be an order. Furthermore, let*

$$\alpha = \frac{\rho_{2\sigma}(\mathbb{Z})}{2\,\rho_\sigma(\mathbb{Z})}, \quad \text{and} \quad \beta = \frac{\rho_{2\sigma}(\mathbb{Z})}{2\,\rho_\sigma(\mathbb{Z} + \frac{1}{2})}.$$

Then

$$\mathrm{R}_a\Big(\widetilde{\mathcal{D}}_{\mathbf{Q},2\sigma} \,\Big\|\, \mathcal{D}_{\mathbf{Q},2\sigma}\Big) = \left(\frac{\alpha^{a-1} + \beta^{a-1}}{2}\right)^{\frac{2n}{a-1}}.$$

Proof. Since each $\mathbf{w} \in R^2$ lies in $2R^2 + \mathbf{h}$ for exactly one $\mathbf{h} \in (R/2R)^2$ we have $\widetilde{\mathcal{D}}_{\mathbf{Q},2\sigma}(\mathbf{w}) = 2^{-2n}\,\rho_{\mathbf{Q},2\sigma}(\mathbf{w}) / \rho_{\mathbf{Q},2\sigma}(2R^2 + \mathbf{h})$. Similarly, we have $\mathcal{D}_{\mathbf{Q},2\sigma}(\mathbf{w}) = \rho_{\mathbf{Q},2\sigma}(\mathbf{w}) / \rho_{\mathbf{Q},2\sigma}(R^2)$. Hence, by definition of the Rényi

divergence,

$$\mathrm{R}_a\Big(\widetilde{\mathcal{D}}_{\mathbf{Q},2\sigma} \,\Big\|\, \mathcal{D}_{\mathbf{Q},2\sigma}\Big)^{a-1} = \sum_{\substack{\mathbf{h}\in(R/2R)^2 \\ \mathbf{w}\in 2R^2+\mathbf{h}}} \widetilde{\mathcal{D}}_{\mathbf{Q},2\sigma}(\mathbf{w})^a / \mathcal{D}_{\mathbf{Q},2\sigma}(\mathbf{w})^{a-1}$$

$$= \sum_{\mathbf{h}\in(R/2R)^2} \frac{1}{2^{2n}} \left(\frac{\rho_{\mathbf{Q},2\sigma}(R^2)}{2^{2n}\,\rho_{\mathbf{Q},2\sigma}(2R^2+\mathbf{h})}\right)^{a-1} \sum_{\mathbf{w}\in 2R^2+\mathbf{h}} \frac{\rho_{\mathbf{Q},2\sigma}(\mathbf{w})}{\rho_{\mathbf{Q},2\sigma}(2R^2+\mathbf{h})} \ .$$

Note that $\sum_{\mathbf{w}} \rho_{\mathbf{Q},2\sigma}(\mathbf{w}) / \rho_{\mathbf{Q},2\sigma}(2R^2+\mathbf{h}) = 1$, simplifying the equation to

$$\mathrm{R}_a\Big(\widetilde{\mathcal{D}}_{\mathbf{Q},2\sigma} \,\Big\|\, \mathcal{D}_{\mathbf{Q},2\sigma}\Big)^{a-1} = \frac{1}{2^{2n}} \sum_{\mathbf{h}\in(R/2R)^2} \left(\frac{\rho_{\mathbf{Q},2\sigma}(R^2)}{2^{2n}\,\rho_{\mathbf{Q},2\sigma}(2R^2+\mathbf{h})}\right)^{a-1}$$

$$= \frac{1}{2^{2n}} \sum_{\mathbf{h}\in(R/2R)^2} \left(\frac{\rho_{2\sigma}(R^2)}{2^{2n}\,\rho_{2\sigma}(2R^2+\mathbf{B}\cdot\mathbf{h})}\right)^{a-1} ,$$

where the second equality relies on $\mathbf{B} \in \mathrm{GL}_2(R)$ and therefore $\mathbf{B}R^2 = R^2$. Note $\left\{2R^2+\mathbf{Bh} : \mathbf{h}\in(R/2R)^2\right\} = \left\{2R^2+\mathbf{t} : \mathbf{t}\in(R/2R)^2\right\}$, so we may sum over $\mathbf{t}\in(R/2R)^2$. We also pass to the unstructured setting, noting that $\rho_{2\sigma}(R^2) = \rho_{2\sigma}(\mathbb{Z}^{2n})$ and that, since $\mathbb{Z}^{2n}$ has an orthogonal basis $\mathbf{I}_{2n}(\mathbb{Z})$, $\rho_{2\sigma}(\mathbb{Z}^{2n}) = \rho_{2\sigma}(\mathbb{Z})^{2n}$. Similarly, we have $\rho_{2\sigma}(2R^2+\mathbf{t}) = \rho_{2\sigma}(2\mathbb{Z}^{2n}+\mathbf{t}) = \prod_{i=0}^{2n-1}\rho_{2\sigma}(2\mathbb{Z}+t_i)$, with $\mathbf{t}\in\{0,1\}^{2n}$ after the first equality. Moreover, $\rho_{2\sigma}(2\mathbb{Z}+b) = \rho_\sigma(\mathbb{Z}+b/2)$ holds ($b\in\{0,1\}$). Thus,

$$\mathrm{R}_a\Big(\widetilde{\mathcal{D}}_{\mathbf{Q},2\sigma} \,\Big\|\, \mathcal{D}_{\mathbf{Q},2\sigma}\Big)^{a-1} = \frac{1}{2^{2n}} \sum_{\mathbf{t}\in\{0,1\}^{2n}} \left(\frac{\rho_{2\sigma}(\mathbb{Z}^{2n})}{2^{2n}\,\rho_{2\sigma}(2\mathbb{Z}^{2n}+\mathbf{t})}\right)^{a-1}$$

$$= \frac{1}{2^{2n}} \sum_{\mathbf{t}\in\{0,1\}^{2n}} \prod_{i=0}^{2n-1} \left(\frac{\rho_{2\sigma}(\mathbb{Z})}{2\,\rho_\sigma(\mathbb{Z}+\frac{t_i}{2})}\right)^{a-1} .$$

By swapping these finite sums and products, and considering the contribution of a single t_i, half of which are zero and half of which are one over $\mathbf{t}\in\{0,1\}^{2n}$ for any fixed index i, we have

$$\mathrm{R}_a\Big(\widetilde{\mathcal{D}}_{\mathbf{Q},2\sigma} \,\Big\|\, \mathcal{D}_{\mathbf{Q},2\sigma}\Big)^{a-1} = \frac{1}{2^{2n}} \prod_{i=0}^{2n-1} \left[\left(\frac{\rho_{2\sigma}(\mathbb{Z})}{2\,\rho_\sigma(\mathbb{Z})}\right)^{a-1} + \left(\frac{\rho_{2\sigma}(\mathbb{Z})}{2\,\rho_\sigma(\mathbb{Z}+\frac{1}{2})}\right)^{a-1}\right]$$

$$= \left(\frac{\alpha^{a-1}+\beta^{a-1}}{2}\right)^{2n} ,$$

using the definitions of α and β. □

As σ grows the Rényi divergence decreases towards one. When σ is not too big, α and β in the above lemma can be efficiently numerically computed to any desired precision via a tail cut.

The following lemma, which is an adaptation of [DPPvW22b, Lem. 10] and [FH23, Lem. 1] to the symplectic automorphism, proves that a forgery in the SUF-CMA game of Hawk is with overwhelming probability a nontrivially new solution to ac-omSVP.

Lemma 5.20. *Let n be a power of 2, $\varepsilon \in (0,1)$, $\sigma \geq \eta_\varepsilon(\mathbb{Z}^{2n})$. For a sample $\mathbf{w} \leftarrow \mathcal{D}_{\mathbf{Q},2\sigma}$ we have*

$$\Pr_{\mathbf{w}}\left[\exists \alpha \in \mathcal{G}_{\mathbf{Q}} \setminus \{1,-1\} \colon \frac{\mathbf{h}+\alpha(\mathbf{w})}{2} \in R^2\right] \leq \frac{n+1}{2^n} \cdot \frac{1+\varepsilon}{1-\varepsilon}\ , \tag{5.8}$$

where $\mathbf{h} = \mathbf{w} \bmod 2$. For a fixed $\mathbf{h}^\circ \in (R/2R)^2$ we have

$$\Pr_{\mathbf{w}}\left[\exists \alpha \in \mathcal{G}_{\mathbf{Q}} \colon \frac{\mathbf{h}^\circ+\alpha(\mathbf{w})}{2} \in R^2\right] \leq \frac{2n}{2^{2n}} \cdot \frac{1+\varepsilon}{1-\varepsilon}\ . \tag{5.9}$$

Proof. To prove Equation (5.8), we split $\mathcal{G}_{\mathbf{Q}} = \mu_{\mathbb{K}} \cup \{\alpha\omega_{\mathbf{Q}} \mid \alpha \in \mu_{\mathbb{K}}\}$ in two sets and use a union bound. Moreover, for $\alpha \in \mathcal{G}_{\mathbf{Q}}$, note $(\mathbf{h}+\alpha(\mathbf{w}))/2 \in R^2$ happens if and only if $\mathbf{w} \equiv \alpha(\mathbf{w}) \pmod 2$ holds, as -1 acts trivially modulo $2R^2$.

First, if we have $\mathbf{w} \equiv \alpha\mathbf{w} \pmod 2$ for $\alpha \in \mu_{\mathbb{K}} \setminus \{1,-1\}$, then this statement is also true when replacing α by $\alpha^2, \alpha^4, \ldots, \alpha^{2n} = 1$. Now, Lemma 5.12 implies $\mathbf{w} \equiv X^{n/2}\mathbf{w} \pmod 2$. Note this event is rare as 2^n vectors $\mathbf{h} \in (R/2R)^2$ satisfy $\mathbf{h} \equiv X^{n/2}\mathbf{h} \pmod 2$. Hence, using Lemma 5.18 and a union bound over all such $\mathbf{h}$, we get

$$\Pr_{\mathbf{w}}\left[\exists \alpha \in \mu_{\mathbb{K}} \setminus \{1,-1\} \colon \frac{\mathbf{h}+\alpha\mathbf{w}}{2} \in R^2\right] \leq 2^n \cdot 2^{-2n}\frac{1+\varepsilon}{1-\varepsilon}.$$

Second, suppose we have $\mathbf{w} \equiv \alpha(\mathbf{w}) \pmod 2$ for some $\alpha \in \mathcal{G}_{\mathbf{Q}} \setminus \mu_{\mathbb{K}}$, i.e., for some $i \in \{0,1,\ldots,n-1\}$ we have $\mathbf{w} \equiv X^i \cdot \omega_{\mathbf{Q}}(\mathbf{w}) \pmod 2$ by Lemma 5.12. By a similar union bound and Lemma 5.18, now we have

$$\Pr_{\mathbf{w}}\left[\exists \beta \in \mu_{\mathbb{K}} \colon \frac{\mathbf{h}+\beta\omega_{\mathbf{Q}}(\mathbf{w})}{2} \in R^2\right] \leq 2^{-2n}\frac{1+\varepsilon}{1-\varepsilon} \cdot \sum_{i=0}^{n-1}|C_i|,$$

where $C_i = \{\mathbf{h} \in (R/2R)^2 \mid \mathbf{h} \equiv X^i \omega_{\mathbf{Q}}(\mathbf{h}) \pmod 2\}$. By taking $\mathbf{z} = (z_0, z_1)^\mathsf{T} = \mathbf{B}\mathbf{h}$, where $\mathbf{Q} = \mathbf{B}^\star\mathbf{B}$, we have both $z_0 \equiv X^i z_1^\star \pmod 2$ and $z_1 \equiv X^i z_0^\star \pmod 2$. Note that both conditions are equivalent, and any choice for $z_0 \in R/2R$ uniquely determines z_1. Hence, $|C_i| = 2^n$ for all i, which implies

$$\Pr_{\mathbf{w}}\left[\exists \alpha \in \mathcal{G}_{\mathbf{Q}} \setminus \mu_{\mathbb{K}} \colon \frac{\mathbf{h} + \alpha(\mathbf{w})}{2} \in R^2\right] \leq \frac{n}{2^n} \cdot \frac{1+\varepsilon}{1-\varepsilon}.$$

To prove Equation (5.9), observe $(\mathbf{h}^\circ + \alpha(\mathbf{w}))/2 \in R^2$ holds if and only if $\mathbf{w} \in \alpha^{-1}(\mathbf{h}^\circ) + 2R^2$. Since multiplication by -1 acts trivially modulo 2, there are at most $2n$ cosets of $2R^2$ to consider for $\mathbf{w}$ such that $\frac{1}{2}(\mathbf{h}^\circ + \alpha(\mathbf{w})) \in R^2$ holds for some $\alpha \in \mathcal{G}_{\mathbf{Q}} \setminus \mu_{\mathbb{K}}$. Together with Lemma 5.18, this proves Equation (5.9). □

Reduction

We now present the theorem that any ROM-SUF-CMA adversary against Hawk can be used as a subroutine for a SAMPLE adversary against an instance of ac-omSVP, while only losing a constant number of bits. There is also a reduction in the quantum ROM [FH23].

Below, let $\Pi_{\text{Hawk}}^{\text{tables}} = (\text{KeyGen}, \text{Sign}', \text{Verify})$ be Hawk as in Section 5.2.2, where Sign′ generates compressed signatures using precomputed tables like in the implementation, see Section 5.5.2. Moreover, $\kappa \in \mathbb{Z}_{\geq 1}$ denotes the maximum number of samples from samp needed to produce a valid Hawk signature. As invalid signatures are rare, one may take $\kappa = \lambda + 67$.

Theorem 5.21 (omSVP to SUF-CMA reduction)**.** *Let $\lambda \in \{128, 256\}$, $q_s \leq 2^{64}$ and $q_h \leq 2^\lambda$. Let $\mathcal{A}$ be a (quantum) adversary against the* (Q)SUF-CMA *game against $\Pi_{\text{Hawk}}^{\text{tables}}$ that runs in time at most $t_{\mathcal{A}} \geq 1$, makes (q_s, q_h) queries to $(\mathsf{OSign}, \mathsf{H})$, respectively, and has success probability*

$$\varepsilon_{\mathcal{A}} = \mathsf{Adv}^{\text{SUF-CMA}}_{\Pi_{\text{Hawk}}^{\text{tables}}, \mathcal{A}},$$

such that $t_{\mathcal{A}}/\varepsilon_{\mathcal{A}} < 2^\lambda$. In particular $\lambda = 128$ corresponds to Hawk-512 *and $\lambda = 256$ corresponds to* Hawk-1024*. Moreover, assume* $\mathsf{saltlen}$ *is taken arbitrarily large (in particular, at least as large as listed in Table 5.3).*

Then, there exists a (quantum) adversary $\mathcal{B}$ against the SAMPLE *game with the appropriate $\mathsf{Init}(1^n)$ running in time at most $t_{\mathcal{B}} = t_{\mathcal{A}} +$*

$\mathsf{Overhead}(\kappa \cdot q_s, q_h)$. *This overhead consists of making* $\kappa \cdot q_s$ *queries to* samp *and simulating* q_h *queries to* H. *The success probability of* $\mathcal{B}$ *satisfies*

$$\mathsf{Adv}^{\text{SAMPLE}}_{\text{AC-OMSVP},\mathcal{B}} > \frac{1}{2}\varepsilon_{\mathcal{A}}.$$

Proof. See Section 6 of https://hawk-sign.info/hawk-spec.pdf. □

Remark 5.22. We do not expect the reduction to be bidirectional. Intuitively even if one can find a $\mathbf{w}^\star$ 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.

Discussion

The AC-OMSVP instances implicit in our reduction from omSVP to Hawk are trivially solvable. In particular, without making any queries to the samp oracle, an adversary can submit $\mathbf{w}^\star = (1, 0)^\mathsf{T}$. In this case $\|\mathbf{w}^\star\|_{\mathbf{Q}} = \|(f, g)\| \approx \sqrt{2n}\sigma_{\mathsf{pk}} \leq 2\sqrt{2n}\sigma_{\mathsf{ver}} = L$. This is a consequence of us choosing σ_{pk} as small as our practical cryptanalysis will allow us, see Section 5.3.2, and not being able to choose σ_{ver} small enough to mitigate this without unacceptably increasing the restart probability. This is an instance of us choosing to follow practical cryptanalysis for setting parameters, and using our theoretical reduction to convince us of the soundness of our design; we could easily fix this by increasing σ_{pk}, but this would also increase the size of our public keys. We note that, while this decision makes our reduction vacuous, we do not know practically how to use these public key vectors to create forgeries against Hawk. More generally, one can consider an adversary that performs lattice reduction on the form $\mathbf{Q}$ and possibly combines the result with samples from the samp oracle.

In a more idealised setting, we are effectively asking a omSVP adversary to perform a relaxed notion of lattice sieving for just one vector. The standard heuristic from that literature is that the vectors $\mathbf{w}$ from samp are i.i.d. uniform points on a sphere of radius $2\sqrt{2n}\sigma_{\mathsf{sign}}$ [NV08]. Rather than asking for enough distinct shorter combinations of such $\mathbf{w}$ to recurse a sieving operation, the SAMPLE game asks for just one combination of $\mathbf{w}$, provided it does not fall into $\mathcal{L}_{\mathsf{samples}}$, that can actually be slightly longer, by a factor of $\sigma_{\mathsf{ver}}/\sigma_{\mathsf{sign}} \geq 1$. Optimising k-sieving techniques [BLS16, HK17] for this setting where a single relaxed combination is sought may be an interesting approach. Also, whether having

access to (lattice reduction on) $\mathbf{Q}$ can help an adversary in this setting, or whether one can reduce to a variant of omSVP where the adversary is not given $\mathbf{Q}$ by showing something equivalent can be obtained efficiently by using the samp oracle, are other interesting avenues for further work.

Finally, the applicability of learning style attacks [NR09, DN12b] to the particular ac-omSVP instances relevant to Hawk should be considered. Lemma 5.19 tells us that $\widetilde{\mathcal{D}}_{\mathbf{Q},2\sigma_{\mathsf{sign}}}$ and $\mathcal{D}_{\mathbf{Q},2\sigma_{\mathsf{sign}}}$ are close, which relate to $\mathbf{w}$ sampled internally during Hawk signing and the output of the samp oracle in the SAMPLE game. Recall that $\mathbf{w}$ is public in Hawk signing since it can be computed from $\mathbf{s}$ and $\mathbf{h}$. Therefore, if one can mount such a learning style attack against Hawk signatures, then one can mount such a learning attack against the SAMPLE game using $\mathbf{w}$ from samp with a very similar probability.