Toward a Secure Fixed-Point Implementation of the Falcon Signature Scheme

Daniel De Almeida Braga[1], Pierre-Alain Fouque[1], Bachir Lachguel[1,2], and Thomas Prest[2]

[1] IRISA/Inria, Rennes Univ., France
{daniel.de-almeida-braga,pierre-alain.fouque}@irisa.fr
[2] PQShield
{bachir.lachguel,thomas.prest}@pqshield.com

Abstract. Falcon was selected by NIST in 2022 for standardization as a post-quantum digital signature scheme. Among all standardized signature schemes, Falcon achieves the smallest signature size. Its main drawback, however, is its reliance on floating-point arithmetic, which plays a critical role in the security analysis. This reliance poses significant challenges for practical implementations: some platforms lack floating-point units, floating-point division is not constant time on many processors, and protecting floating-point computations against side-channel attacks using masking techniques is particularly difficult on embedded devices.

To address portability issues, Pornin (ePrint 2019/893) proposed an implementation of Falcon that emulates floating-point arithmetic using integer operations. While it enables deployment on a wider range of platforms, this approach incurs a substantial performance penalty compared to the native floating-point implementation.

This work studies the theory and practice of implementing Falcon's signing procedure in *fixed-point arithmetic*. This requires a specific analysis of the *boundedness* and *precision* of intermediate variables.

1. Our boundedness analysis revolves around a key fact: almost every intermediate variable arising during key expansion and signing is bounded by a function of four quantities that can be computed at key generation time. Our modified key generation enforces thresholds on these quantities through a light rejection step that rejects less than 50% of initial Falcon keys. This then yields sharp, unconditional bounds on all fixed-point variables.
   Establishing these bounds is highly nontrivial, and relies on Gaussian concentration arguments as well as on *symplectic pairs*, a generalization of symplecticity.
2. Our precision analysis remains, for now, partly empirical. Following a Rényi divergence argument, our main theorem proves the security of fixed-point Falcon conditioned on error bounds of certain intermediate values. These error bounds are derived empirically based on extensive experiments.

We provide a C fixed-point implementation. It is approximately a factor of two slower than the original floating-point Falcon implementation, but achieves a speedup of an order of magnitude compared to emulated floating-point implementations.

1 Introduction

The transition to post-quantum cryptography has brought renewed attention to the practical deployability of lattice-based cryptographic schemes. Among them, the Falcon digital signature scheme [PFH+20] stands out for its exceptionally compact signatures and keys. Selected by NIST in 2022 for standardization, Falcon is currently the most size-efficient standardized post-quantum signature scheme. This efficiency, however, comes at the cost of a rather complex implementation, as Falcon's implementation fundamentally relies on floating-point arithmetic in the signing algorithm. It is worth pointing out that the verification does not require floating-point, and a subset of the key generation has already been implemented using fixed-point arithmetic [Por23].

The Falcon signature scheme follows the hash-and-sign paradigm: the message is first hashed to a target point c, and the secret key enables sampling a *short* vector (s_1, s_2) such that $s_1 + s_2 \cdot h = c$. Recent analysis [FGdG+26] proves that the unforgeability of (a slightly modified) Falcon is equivalent to a multi-target version of the inhomogeneous SIS problem, while recovering the secret key from the public key is the NTRU key-recovery problem.

This paradigm originates from the seminal work of Gentry, Peikert, and Vaikuntanathan (GPV) [GPV08], which introduced efficient sampling from discrete Gaussian distributions over lattices based on Klein's sampler [Kle00]. Stehlé and Steinfeld later studied the NTRU instantiation of the GPV framework from an asymptotic perspective [SS13], extending it to a ring setting and reducing its security to standard worst-case problems over ideal lattices.

Subsequently, Ducas, Lyubashevsky, and Prest [DLP14] proposed a concrete instantiation of this framework using NTRU lattices, significantly reducing both public key and signature sizes. Their construction relies on working over a specific module in the ring of integers $\mathcal{R}$ of a power-of-two cyclotomic field, together with carefully chosen concrete parameters. The tower-of-rings structure of $\mathcal{R}$ can be further exploited to accelerate Gaussian sampling to quasi-linear time, as shown in [DP16], in contrast to the quadratic-time complexity of Klein's sampler. This optimized sampler, commonly referred to as the FFO sampler, derives its name from its resemblance to Fast Fourier-type algorithms and relies on an efficient orthogonalization process. Building on the same tower-of-rings structure, Pornin and Prest [PP19] further improved Falcon by significantly accelerating its key generation procedure. Unlike `NTRUEncrypt` [HPS98], Falcon requires a full lattice basis as a secret key, which necessitates solving Bézout-like equations in $\mathcal{R}$. With these successive improvements, Falcon achieves a high level of efficiency and security, with its most notable feature being the compactness of both signatures and public keys. An additional advantage is that the scheme naturally supports the construction of identity-based encryption schemes within the GPV framework [GPV08,DLP14].

At the core of Falcon lies a Gaussian sampling procedure. The original analysis, due to Prest [Pre17], relies on relative error bounds arising from floating-point

computations. While this approach is mathematically sound, it poses serious challenges for implementations on constrained or security-critical platforms. In particular, many embedded processors lack a floating-point unit; floating-point division is often not constant time; and masking floating-point arithmetic against side-channel attacks remains a largely open problem. Westerbaan[3] (Cloudflare) pointed out these limitations:

> *Falcon, crucially, is the first big cryptographic algorithm to use double-precision floating-point arithmetic. [...] Despite this achievement, Falcon's constant-timeness is built on shaky grounds. The next generation of Intel CPUs might add an optimization that breaks Falcon's constant-timeness. Also, many CPUs today do not even have fast constant-time double-precision operations. And then still, there might be an obscure bug that has been overlooked. In time, it might be figured out how to do constant-time arithmetic on the FPU robustly, but we feel it's too early to deploy Falcon where the timing of signature minting can be measured. Notwithstanding, Falcon is a great choice for offline signatures such as those in certificates.*

These issues hinder the deployment of Falcon in embedded and high-assurance environments. To improve portability, Pornin proposed an alternative implementation that emulates floating-point arithmetic using integer operations only. While this solution removes the dependence on hardware floating-point units, it introduces a substantial performance penalty over the reference implementation, a 20× factor degradation, making it significantly slower than the original floating-point implementation. This motivates the search for an implementation strategy that simultaneously avoids floating-point arithmetic and retains high performance.

Floating-point arithmetic (FPA) currently appears in three components of Falcon implementations.

1. *Key generation*: FPA is used to accelerate the NTRUSolve algorithm. Pornin already proposed a fixed-point implementation of this component in [Por23,Por25]; the NTRU equation can be checked exactly at runtime, so that precision issues affect efficiency but not correctness. The switch to fixed-point arithmetic leads to the rejection of a constant fraction of the admissible keys, therefore it is also easy to argue that impact on security is minimal. Making NTRUSolve fixed-point is therefore outside the scope of this document; we do, however, modify the surrounding key generation in order to enforce bounds that the two other components rely on.
2. *Key expansion*: the so-called Falcon tree is precomputed from the secret key. This step also relies on FPA, is not covered by [Por23,Por25], and it was previously unclear how much precision it requires.
3. *Signing*: the Fast Fourier Orthogonalization (FFO) sampler consumes the tree to sample signatures. Its security analysis bounds the divergence between the implemented distribution and the ideal discrete Gaussian through relative error

[3] https://blog.cloudflare.com/another-look-at-pq-signatures/, https://blog.cloudflare.com/nist-post-quantum-surprise/

bounds [Pre17] — an analysis originally conducted for Klein's sampler rather than for the FFO sampler.

Unlike key generation, the correctness and security of key expansion and signing cannot be checked at runtime: they can only be established through a proof, which is the object of this work.

In this work, we tackle the problem of having a secure and FPA-free implementation of Falcon and we propose a *complete* fixed-point implementation based on a revised security analysis of the sampling algorithm. Our starting point is a reinterpretation of the sampler's correctness and security conditions using absolute error bounds rather than relative ones. This change of perspective enables the use of fixed-point arithmetic while preserving the statistical guarantees required for security. However, moving to absolute error bounds necessitates a careful and global control of numerical errors throughout the computation.

Our methodology combines two ingredients. First, we recast the sampler's correctness and security conditions in terms of absolute error bounds, and derive sufficient conditions on the numerical precision of the sampler inputs — the centers and standard deviations — that guarantee security. Our main theorem is stated conditionally on such precision bounds; we instantiate them empirically in Section 7.5, leaving their fully analytical (or automated) derivation to future work.

Second, to make these conditions easier to meet — that is, to obtain tighter bounds on the sampler inputs and lower the required precision — we slightly modify the key generation procedure, controlling the quantity α_{hybrid} that governs the precision of the fast Fourier sampler, in the spirit of the Antrag [ENS+23] and Mitaka [EFG+22] key generations. This modification affects neither the functionality nor the security level of Falcon. Combined with a formal boundedness analysis, these bounds determine the fixed-point format of every intermediate variable, allowing for a complete fixed-point implementation of Falcon.

Based on this methodology, we present a fixed-point implementation of Falcon, designed to be constant-time, that is suitable for a wide range of processors, including those without floating-point support. Our implementation is approximately a factor of two slower than the original floating-point Falcon implementation, but achieves a speedup of five to ten times compared to integer-only implementations that emulate FPA. This demonstrates that fixed-point arithmetic provides a practical and efficient compromise between portability, performance, and security.

1.1 Our Contributions

We summarize our contributions as follows:

Absolute-error-based analysis of the Falcon sampler. We introduce a new security analysis of Falcon's Gaussian sampling algorithm based on absolute error bounds, replacing the relative-error framework used in prior work. We show that

absolute error control is sufficient to preserve the statistical guarantees required for security, thereby enabling implementations that do not rely on FPA.

Refined input distributions through modified key generation. We propose a slight modification of Falcon's key generation procedure, inspired by the Antrag key generation used in the Mitaka signature scheme. This modification yields tighter a priori bounds on the sampler inputs, leading to improved precision requirements in the fixed-point analysis without altering the underlying hardness assumptions or security level.

Soundness of fixed-point sampling. We show that if bounds on the absolute errors on the sampling centers and standard deviations can be obtained, Falcon can be proven to be secure when implemented in fixed-point. This is done via a Rényi divergence argument.

Concrete efficiency gains. We demonstrate that the resulting fixed-point implementation achieves significantly better performance than integer-only emulation of FPA, while remaining within a small constant factor of the original floating-point implementation.

1.2 Previous works

The first proposed implementation of Gaussian sampling [DN12] relies on FPA. Since the introduction of Falcon, several works aimed at modifying the scheme to eliminate the FFO sampler.

In [DGPY20], the authors propose a Gaussian sampling method over general lattices that avoids FPA. Their construction is based on Peikert's sampler, which differs from Klein's sampler and generally results in larger sizes (signature and public key). The practicality of this approach still remains to be assessed.

Prest's thesis introduces another alternative, the so-called Hybrid sampler [Pre15]. This approach has inspired several subsequent works, including the Mitaka signature scheme [EFG+22] and the Antrag key generation procedure [ENS+23]. Our results suggest that these can be ported to fixed-point arithmetic more easily than Falcon.

Finally, a masked implementation of Falcon relying on FPA has been proposed in [CC24]. Nevertheless, masking floating-point variables is significantly less efficient than masking integers, as used in fixed-point implementations. Since one of the goals of our work is deployment on embedded devices, memory consumption is a critical concern. We therefore follow an approach similar to [SZZ+22] to reduce by two the memory space of the tree, which exploits the symplectic structure of NTRU bases.

Three recent works are directly related to ours. Halmans et al. [HvVS+26] implemented Falcon using triple-word floating-point arithmetic. Lin et al. [LTYZ25] show that mathematically equivalent implementations of Falcon may output diverging signatures; we encountered this phenomenon first-hand (see Section 5.1).

Finally, Fouque et al. [FGdG+26] give the first formal security proof of a (slightly modified) Falcon; their analysis assumes an idealized sampler, and is thus complementary to ours, which quantifies the gap between implemented and idealized samplers.

1.3 Difference with the conference version

This document is an extended version of the eponymous paper published at IACR CRYPTO 2026. Compared to the conference version, this document has several additions, including but not limited to:

1. Full proofs;
2. Section 3.4 introduces the Type 1 and Type 2 terminology;
3. Lemma 4 is stronger than the lemma initially used.
4. Theorem 2 presents a precision analysis based on the root mean square error (RMSE), instead of the worst-case error analysis of Theorem 1.
5. Section 7 presents a Type 2 Python fixed-point implementation

2 Technical Overview

Falcon leverages floating/fixed-point arithmetic in two areas: Key generation and Signature (see Fig. 1). The key generation procedure outputs a key pair $(\mathsf{sk}, \mathsf{pk})$. The secret key $\mathsf{sk} = (\hat{\mathbf{B}}, \mathsf{T})$ consists of two structured objects defined over the complex numbers: a matrix $\hat{\mathbf{B}}$ and a tree structure T, which are used to enable efficient Gaussian sampling. The signing procedure uses sk to produce (integer arithmetic) signatures. While the distribution of Falcon signatures can be formally characterized under an idealized model of complex arithmetic, real-world implementations operate with a finite precision. This introduces numerical artifacts, leading to a measurable deviation between the actual signature distribution and its ideal theoretical counterpart.

Boundedness analysis When using fixed-point arithmetic, arithmetic overflows can easily occur since, unlike in floating-point arithmetic, there is no exponent that can capture large variations of magnitude. To illustrate this, in the IEEE-754 `binary64` format, representable numbers are of the form $(-1)^b \cdot (1 + m) \cdot 2^{E-1023}$ with $m \in [0, 1)$ and an 11-bit exponent E, so that the ratio between the largest and smallest positive representable numbers is about $2^{2^{11}-1} = 2^{2046}$ (and even 2^{2098} when counting subnormal numbers). For a 64-bit fixed-point number, this ratio is at most 2^{63}. Thus it becomes vital to understand and control the growth of intermediate values. We perform a complete boundedness analysis, deriving bounds by leveraging algebraic properties and Falcon-specific properties.

Bounding variables tightly is crucial. On one hand, underestimating the magnitude of fixed-point variables can lead to *overflow* and voiding the correctness and

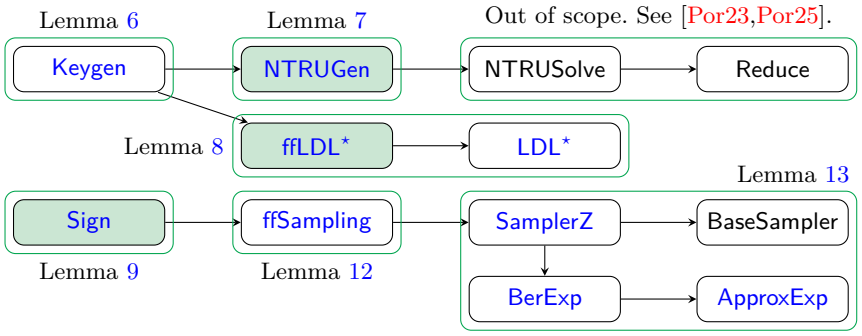

Fig. 1: Diagram of the high-level algorithmic structure of Falcon. An arrow $A \to B$ indicates that B is a subroutine of A. Algorithms for which we propose conservative tweaks are highlighted in green.

security of an implementation. On the other hand, overestimating it leads to *underflow*: the leading bits are zeroes, thus precious bits are wasted to encode trivial information and precision is suboptimal. In some cases, we propose tweaks to Falcon's algorithms in order to improve our boundedness analysis. These tweaks are highlighted in green boxes in the algorithms.

ffLDL⋆ and the Falcon tree (Section 4.3). The key generation procedure outputs a keypair sk, pk, where the secret key sk contains two complex objects: a matrix $\hat{\mathbf{B}}$ and a tree T. While the magnitude of the entries of $\hat{\mathbf{B}}$ is easy to bound, T is constructed via the ffLDL⋆ subroutine, which performs a number $O(n \log n)$ of divisions of the form $c \leftarrow a/b$ for $a, b \in \mathbb{C} \times \mathbb{R}^+$. Fixed-point divisions are notoriously difficult since, in addition to upper-bounding the numerator and denominator, they require to *lower*-bound the denominator.

In Lemma 8, we show that in Falcon, ffLDL⋆ only performs divisions of the form $c \leftarrow a/b$ in which the divisor satisfies $q/(\alpha_{\text{hybrid}})^2 \leq b \leq q \cdot (\alpha_{\text{hybrid}})^2$, with α_{hybrid} a quantity that is computed prior to any division (α_{hybrid} is formally defined in Eq. (7); informally, it measures how far the embeddings of (f, g) deviate from $\sqrt{q}$). This has two benefits: (i) the divisor is lower-bounded a priori, and (ii) except at the root, the quotient admits a simple bound $\|c\|_\infty < 1$.

Lemma 8 is interesting in two aspects. First, the quantity α_{hybrid} that quantifies the amplitude of a, b previously appeared in variants of Falcon such as Mitaka [EFG+22] or Antrag [ENS+23], which both seek to minimize α_{hybrid} in order to optimize the security of Prest's hybrid sampler [Pre15]. Lemma 8 shows that minimizing α_{hybrid} also helps with the precision analysis of Falcon's fast Fourier sampler[4]. Second, its proof heavily relies on symplecticity, a geometric property that

[4]We stress that the hardness of the mathematical assumptions underlying Falcon remains unchanged and still relies on α_{GPV} (Eq. (6)), not α_{hybrid}.

NTRU lattices possess. We find that this property transfers to the tree T via the generalized notion of *symplectic pairs*, illustrated in Fig. 6. This allows to cleanly lower-bound the divisors that appear in ffLDL*. To our knowledge, this is the first time that symplecticity is exploited to enable a clean precision analysis of NTRU-related algorithms.

We also leverage symplectic pairs at the algorithmic level. In [SZZ+22], it was observed that both halves of the FALCON tree T are the negation of each other (except their leaves, which are essentially the inverse of each other). Their technique, which we make explicit in Symplectree, also allows avoiding one operation that represents a significant loss in precision.

Rejection sampling in NTRUGen (Section 4.2). Our analysis of Keygen and its subroutines reveals that critical fixed-point values are bounded by quantities that are easily computed from the private NTRU polynomials f, g, F, G, such as α_{hybrid} or $\|\mathsf{FFT}(f)\|_\infty$. While these quantities admit polynomial upper bounds from fixed parameters, these are typically worse than the average-case behavior.

To close this gap, we adopt a pragmatic approach: if the quantities associated to candidate keys (f, g, F, G) exceed given thresholds, the candidate keys are rejected and the key generation process restarts. This is performed in Lines 7, 8, 13, and 15 of NTRUGen. The resulting guarantees are formalized in Lemma 7.

From an efficiency point of view, this provides unconditional guarantees on the magnitude of fixed-point values output by the key generation procedure. The number of restarts is increased by a small constant factor.

The impact on security is minimal. Indeed, these checks reject a constant fraction of the set of admissible keys, thus a standard Rényi divergence argument (for example) shows that the bit-security is reduced by no more than a small additive term $O(1)$.

Integer/fractional decomposition in Sign (Section 5.1). The signing algorithm Sign constructs a floating/fixed-point target $\hat{\mathbf{t}}$ from the message m. That target is input into the subroutine ffSampling in order to produce a signature. In the original FALCON specification, Sign computes a challenge c uniform in $\mathbb{Z}_q[x]/(x^n + 1)$, and sets $\mathbf{t} = \frac{c}{q}(-F, f)$. $\|\hat{\mathbf{t}}\|_\infty$ may be as large as $n \cdot \gamma_{F,G}$ — where $\gamma_{F,G} \geq \|\mathsf{FFT}(F)\|_\infty, \|\mathsf{FFT}(G)\|_\infty$ is a threshold enforced by our tweaked key generation (Section 4.2) — which is detrimental to the fixed-point precision analysis of ffSampling.

In order to address this gap, we split $\mathbf{t}$ as $\mathbf{t} = \mathbf{t}_{\text{frac}} + \mathbf{t}_{\text{int}}$, where $\mathbf{t}_{\text{int}}$ and $\mathbf{t}_{\text{frac}}$ are the integer and fractional parts of $\mathbf{t}$, respectively. We then input $\mathbf{t}_{\text{frac}}$ into ffSampling (instead of $\mathbf{t}$), and re-add $\mathbf{t}_{\text{int}}$ into the result. Correctness follows from basic properties of discrete Gaussians. $\mathbf{t}_{\text{frac}}$ is guaranteed to satisfy $\|\hat{\mathbf{t}_{\text{frac}}}\|_\infty \leq n/2$, a bound that is better than the bound on $\|\hat{\mathbf{t}}\|_\infty$ by a factor $2\,\gamma_{F,G}$. The correctness of our technique is formalized in Lemma 9.

Analysis of ffSampling (Section 5.2). Finally, we need to perform a boundedness analysis for ffSampling, which is the main subroutine of Sign and the only one (besides the FFT) that requires floating/fixed-point arithmetic. Due to the algorithm's complexity, this is a highly nontrivial task.

Thankfully, we notice that every intermediate variable of ffSampling can be expressed as a constant plus the inner product between a discrete Gaussian and a fixed vector. This allows to leverage concentration bounds on the magnitude of such inner products. Interestingly, our analysis relies once again on symplectic pairs. Our result is formalized in Lemma 12.

2.1 Precision analysis

In a fixed-point implementation, a single integer word is used to represent a real value by conceptually dividing it into two parts: an *integral part* and a *fractional part*. This separation does not exist at the hardware level; it is entirely defined by the programmer. The more bits allocated to the fractional part, the higher the precision of the representation. The boundedness analysis makes it possible to control the range of each variable, and therefore determines the required size of the integral part. However, one must also carefully choose the number of bits devoted to the fractional part in order to guarantee sufficient precision and security. To ensure correct computations, it is necessary to control the distance between the real value x and its encoded approximation $\bar{x}$.

Given the bounds we compute in this paper, it is then possible to deduce the format of the fixed-point variables in a manner that *provably* prevents the occurrence of overflows. Then, we deduce the number of bits that are left for the fractional part in the fixed-point format.

We believe that error bounds could be obtained by automated tooling [MNR14] or pen-and-paper proofs in further work, thus we focus our work on how to prove the security of a fixed-point Falcon implementation *if we are given such error bounds*. In particular, in Falcon's sampler, the absolute error for the variable storing the centers c and for the standard deviations σ_j allow to compute the Rényi divergence between the ideal and sampled distribution which would give a provably secure fixed-point implementation of Falcon. In Section 7, we compute empirical error bounds which indicate that 64-bit are sufficient to securely implement Falcon in (so-called *Type 2*) fixed-point arithmetic.

Table 1 summarizes the use of floating- or fixed-point operations throughout the Falcon implementation. While they represent a small part of the runtime, the division and square root operations are usually considered as non-constant time operations on FPA.

Table 1: Summary of F(x)p (floating- or fixed-point) operations. A ✓ mark indicates that the algorithm performs this type of operation; a function name in the In/Out columns indicates the subroutine that F(x)p values are received from (resp. sent to). In: takes one or more F(x)p as input (including from subroutines). `fadd`: addition or subtraction. `fcmul`: multiplication by a public constant. `fmul`: multiplication between two sensitive values. `fdiv`: division. `fsqrt`: square root. Out: outputs one or more F(x)p (including to subroutines).

Function	In	`fadd`	`fcmul`	`fmul`	`fdiv`	`fsqrt`	Out
Common functions							
splitfft	✓	✓	✓				✓
mergefft	✓	✓	✓				✓
FFT		✓	✓				✓
invFFT	✓	✓	✓				
Key generation							
Keygen		✓	✓	✓	✓	✓	ffLDL*
→ NTRUGen		✓	✓	✓	✓		
→ NTRUSolve							
→ Reduce		✓		✓	✓		
→ ffLDL*	ffLDL*						ffLDL*
→ LDL*	ffLDL*	✓		✓	✓		ffLDL*, LDL*
Signing							
Sign	ffSampling	✓	✓	✓			ffSampling
→ ffSampling	Sign, ffSampling, SamplerZ	✓	✓	✓			ffSampling, SamplerZ
→ SamplerZ	ffSampling	✓	✓	✓			BerExp
→ BaseSampler							
→ BerExp	SamplerZ	✓	✓				ApproxExp
→ ApproxExp	BerExp	✓		✓			

3 Preliminaries

In this section, we define cyclotomic fields, NTRU lattices, discrete Gaussian distributions, and the Rényi divergence, and give some background on fixed-point arithmetic.

3.1 Cyclotomic Fields

Let m be a positive integer, and $d = \varphi(m)$ be the degree of the m-th cyclotomic polynomial $\Phi_m(x)$, where $\varphi(m)$ denotes Euler's totient function. Let ζ be a primitive m-th root of unity. Let $\mathbb{K}_m$ be the cyclotomic field associated with $\Phi_m(x)$, and its

ring of integers $\mathcal{R} = \mathbb{Z}[\zeta]$. It follows that $\mathbb{K}_m = \mathbb{Q}(\zeta)$ is isomorphic to $\mathbb{Q}[x]/(\Phi_m(x))$ and $\mathcal{R}$ to $\mathbb{Z}[x]/(\Phi_m(x))$. Both $\mathbb{K}_m$ and $\mathcal{R}$ are in $\overline{\mathbb{K}} = \mathbb{K}_m \otimes \mathbb{R} = \mathbb{R}[x]/(\Phi_m(x))$.

Any element $f = \sum_{i=0}^{d-1} f_i \zeta^i \in \overline{\mathbb{K}}$ can be represented as $(f_0, \ldots, f_{d-1})$. The field automorphism $\zeta \mapsto \zeta^\star$ over $\mathbb{K}$ is an involution corresponding to the complex conjugation, and $f^\star$ is the image of f under this automorphism. The number field $\mathbb{K}$ has d complex embeddings, called canonical, $\varphi_i : \mathbb{K} \to \mathbb{C}$ that can be extended over $\overline{\mathbb{K}}$. Elements of $\mathbb{K}_m$ represented by polynomials are sent in $\mathbb{C}$ by evaluating the polynomials at all the primitive m-th roots of unity, which form the FFT representation. Throughout the paper, we denote by $\hat{a}$ the FFT representation of a. An element $e \in \overline{\mathbb{K}}$ is invertible if and only if all its embeddings are non-zero.

In Falcon, m is a power of two, $d = n = \varphi(m) = m/2$, and $\Phi_m(x) = x^n + 1$, $\mathbb{K}_m = \mathbb{Q}[x]/(x^n + 1)$. The hermitian structure of $\overline{\mathbb{K}}$ follows from the fact that $\mathbb{K}$ can be seen as a $\mathbb{Q}$-vector space, it can be equipped with the sesquilinear map $\langle \cdot, \cdot \rangle$: $\mathbb{K} \times \mathbb{K} \to \mathbb{C}$, $\langle a, b \rangle = \mathrm{Tr}(ab^\star) = \sum_{i=1}^{d} \varphi_i(a)\varphi_i(b^\star)$. The map extends to $\overline{\mathbb{K}}$ and the associated norm $\|x\|$ is $\sqrt{\langle x, x \rangle} \in \mathbb{R}$. For vector space $\mathbb{K}^2$, the sesquilinear product is $\langle \mathbf{a}, \mathbf{b} \rangle_{\mathbb{K}} = \sum a_i b_i^\star$.

3.2 Lattices and Discrete Gaussian

A lattice $\Lambda \subset \mathbb{R}^n$ is the set of all integer linear combinations of n linearly independent vectors $\mathbf{B} = (\mathbf{b}_1, \ldots, \mathbf{b}_n)$, called a basis of Λ. The dual of a lattice Λ in $\mathbb{R}^n$, denoted Λ^*, is the set of all vectors $\mathbf{y} \in \mathbb{R}^n$ such that $\langle \mathbf{x}, \mathbf{y} \rangle \in \mathbb{Z}$ for all vectors $\mathbf{x} \in \Lambda$. The Gram-Schmidt orthogonalization of a basis $\mathbf{B}$ is $\tilde{\mathbf{B}} = (\tilde{\mathbf{b}}_1, \ldots, \tilde{\mathbf{b}}_n)$ and the Gram-Schmidt norm of $\mathbf{B}$ is $\max_{1 \le i \le n} \|\tilde{\mathbf{b}}_i\|$.

For any $\sigma > 0$, we define the Gaussian function $\rho_\sigma \colon \mathbb{R}^n \to [0, 1)$ as

$$\rho_\sigma(\mathbf{x}) = \exp\big(-\|\mathbf{x}\|^2 / 2\sigma^2\big).^{5} \tag{1}$$

For a discrete set $A \subset \mathbb{R}^n$, we define $\rho_\sigma(A) = \sum_{\mathbf{v} \in A} \rho_\sigma(\mathbf{v})$. The discrete Gaussian distribution over a lattice coset $\Lambda + \mathbf{c}$ with parameter σ, $D_{\Lambda + \mathbf{c}, \sigma}$, is the probability distribution over $\Lambda + \mathbf{c}$ that to each vector $\mathbf{v} \in \Lambda + \mathbf{c}$ assigns the probability

$$\frac{\rho_\sigma(\mathbf{v})}{\rho_\sigma(\Lambda + \mathbf{c})}.$$

When the standard deviation is above the so-called smoothing parameter [MR07], the discrete Gaussian behaves as a continuous one. Given $\varepsilon > 0$, and a lattice Λ, with smoothing loss ε, the smoothing parameter is

$$\eta_\varepsilon(\Lambda) = \inf\{s > 0 : \rho_{1/(2\pi s)}(\Lambda^*) = 1 + \varepsilon\}.$$

[5] In the lattice literature, two definitions co-exist: for example, [Lyu12,Pre17] and we use the one in Eq. (1), while [MR07] uses $\rho_s(\mathbf{x}) = \exp\big(-\pi\|\mathbf{x}\|^2/s^2\big)$. This entails $s = \sqrt{2\pi}\sigma$.

We recall that $\eta_\epsilon(\mathbb{Z}^n) \leq \eta^+_\epsilon(\mathbb{Z}^n)$, where:

$$\eta^+_\epsilon(\mathbb{Z}^n) = \frac{1}{\pi}\sqrt{\frac{1}{2}\log\left(2n\left(1+\frac{1}{\epsilon}\right)\right)}.$$

We note that $\eta^+_\epsilon(\mathbb{Z}^n) \geq \eta^+_{\epsilon/n}(\mathbb{Z})$. FALCON sets the main standard deviation σ as:

$$\sigma = 1.17\eta^+_\epsilon(\mathbb{Z}^{2\,n})\sqrt{q}. \tag{2}$$

Lemma 1 (Lemma 4.3 in [Lyu12]). *Let $\mathbf{v} \in \mathbb{R}^n$, $\sigma > 0$ and $\tau > 0$.*

$$\mathbb{P}[|\langle \mathbf{z}, \mathbf{v}\rangle| > \tau\sigma\|\mathbf{v}\|; \mathbf{z} \leftarrow D_{\mathbb{Z}^n,\sigma}] \leq 2e^{-\tau^2/2}. \tag{3}$$

If $\tau \geq \sqrt{2(\lambda+1)\log 2}$, then the right side of Eq. (3) is at most $2^{-\lambda}$.

In Lemma 2, we provide a refinement of Lemma 4.2 from [MR07]. The initial lemma requires $\sigma \geq 2\,\eta_\epsilon(\mathbb{Z})$, which does not capture the regimes $\sigma \approx \eta_\epsilon(\mathbb{Z})$ that are used in lattice-based cryptography. Lemma 2 is valid for any $\sigma \geq \eta_\epsilon(\mathbb{Z})$. In addition, it covers the fourth moment $\mathbb{E}[(z-c)^4]$. The proof is deferred to Section B.1.

Lemma 2 (Analogue of [MR07, Lemma 4.2] over $\mathbb{Z}$, for $\sigma \geq \eta_\epsilon(\mathbb{Z})$). *Let $\epsilon \in (0, 1/4]$, $\sigma \geq \eta_\epsilon(\mathbb{Z})$ and $c \in \mathbb{R}$. Let $z \sim D_{\mathbb{Z},\sigma,c}$ and $m_i = \mathbb{E}[(z-c)^i]$. Then:*

$$\begin{aligned}
|m_1| &\leq \frac{2\pi\,\epsilon\,\sigma^2}{(1-\epsilon)^2},\\
\left|m_2 - \sigma^2\right| &\leq \frac{4\pi^2\,\epsilon\,\sigma^4}{(1-\epsilon)^2},\\
\left|m_4 - 6\,\sigma^2\,m_2 + 3\,\sigma^4\right| &\leq \frac{16\pi^4\,\epsilon\,\sigma^8}{(1-\epsilon)^2}.
\end{aligned}$$

In particular this implies that:

$$\mathrm{Var}[(z-c)^2] \coloneqq m_4 - m_2^2 = 2\,\sigma^4 + O(\epsilon\,\sigma^6 + \epsilon\,\sigma^8)$$

NTRU Lattices. Given $f, g \in \mathcal{R}$ such that f is invertible in $\mathbb{Z}[x]/(q, \Phi_m(x))$, where $q = 12289$ is prime, the NTRU public-key is $h = f^{-1}g \bmod q$. The NTRU lattice is defined as $\{(u,v) \in \mathcal{R}^2 : u + vh = 0 \bmod q\}$, and it has two bases, one given by h and a short one defined by (f,g) and (F,G) such that $fG - gF = q$:

$$B_b = \begin{pmatrix} -h & 1 \\ q & 0 \end{pmatrix} \text{ and } B_{f,g} = \begin{pmatrix} g & -f \\ G & -F \end{pmatrix}.$$

This lattice can be seen as a $2d$-dimensional lattice over $\mathbb{Z}$ and each polynomial f can be seen as the matrix representing the multiplication by f in $\mathcal{R}$:

$$C(f) = \begin{pmatrix} f_1 & f_2 & f_3 & \dots & f_d \\ -f_d & f_1 & f_2 & \dots & f_{d-1} \\ \dots & \dots & \dots & \dots & \dots \\ -f_2 & -f_3 & \dots & \dots & f_1 \end{pmatrix}.$$

3.3 Rényi Divergence

The Rényi divergence is a common tool in lattice-based cryptography [BLR+18,Pre17].

Definition 1 (Rényi divergence). *Let $\mathcal{P}, \mathcal{Q}$ be two discrete distributions such that $Supp(\mathcal{P}) \subseteq Supp(\mathcal{Q})$, and $\alpha \in (1; +\infty)$. The Rényi divergence of order α is:*

$$R_\alpha(\mathcal{P}; \mathcal{Q}) = \left(\sum_{x \in Supp(\mathcal{P})} \frac{\mathcal{P}(x)^\alpha}{\mathcal{Q}(x)^{\alpha-1}} \right)^{\frac{1}{\alpha-1}} = \left(\mathbb{E}_{x \sim \mathcal{Q}} \left(\frac{\mathcal{P}(x)}{\mathcal{Q}(x)} \right)^\alpha \right)^{1/(\alpha-1)}.$$

Following Csiszár's f-divergence framework [Csi63], $(R_\alpha^{\alpha-1} - 1)$ is an f-divergence for $f : x \mapsto x^\alpha - 1$. We recall in Lemma 3 a few properties that follow from this interpretation; proofs can also be found in [vEH12,BLR+18].

Lemma 3. *For distributions $\mathcal{P}, \mathcal{Q}$ the following properties hold for all $\alpha > 1$:*

1. ***Data processing inequality.*** *For a (randomized) function f,*

$$R_\alpha(f(\mathcal{P}); f(\mathcal{Q})) \leq R_\alpha(\mathcal{P}; \mathcal{Q}).$$

2. ***Probability preservation.*** *For any event $E \subseteq Supp(\mathcal{Q})$:*

$$\mathcal{P}(E) \leq (\mathcal{Q}(E) \cdot R_\alpha(\mathcal{P}; \mathcal{Q}))^{\frac{\alpha-1}{\alpha}}.$$

3. ***Monotonicity.*** *$\alpha \mapsto R_\alpha(\mathcal{P}; \mathcal{Q})$ is non-decreasing on $(1, +\infty)$.*

We need the following lemmas for our main proof. Lemmas 4 and 5 bound the Rényi divergence between Gaussians of slightly different centers and standard deviations, respectively. In the conference version of this paper, we relied on Lemma 4.2 from [LSS14] instead of Lemma 4. This only guaranteed $\ln R_\alpha \leq O\left(\frac{\alpha\,\delta^2}{\sigma^2} + \epsilon\right)$, and since $\epsilon \approx 2^{-36}$ in Falcon, this was not sufficient for our purposes. In regimes of interest, Lemma 4 guarantees $\ln R_\alpha \leq O\left(\frac{\alpha\,\delta^2}{\sigma^2}(1+\epsilon)\right)$, a stronger statement which allows our proofs to go through. Due to page constraints, the proofs of Lemmas 4 and 5 are given in Sections B.2 and B.3, respectively; the proof strategies are very similar.

Lemma 4. *Let $\epsilon \in (0, 1/4]$, $\sigma \geq \eta_\epsilon(\mathbb{Z})$, $\alpha > 1$, and $c, \delta \in \mathbb{R}$. Then:*

$$R_\alpha(D_{\mathbb{Z},\sigma,c+\delta},\, D_{\mathbb{Z},\sigma,c}) \leq \exp\left(\frac{\alpha\,(\alpha+1)\,\delta^2}{2\,(\alpha-1)\,\sigma^2}\,(1 + O(\epsilon)) \right) \tag{4}$$

Lemma 5. *Let $c \in \mathbb{R}$, $\alpha \geq 2$, $\epsilon \in (0, 1/4)$ and $\delta > 0$ such that $\alpha\,\delta \leq 1/2$. Let $\sigma_1, \sigma_2 > 0$ such that $\sigma_1 \geq \sqrt{1+\alpha\delta} \cdot \eta_\epsilon(\mathbb{Z})$ and $|\sigma_1^2/\sigma_2^2 - 1| \leq \delta$. Then:*

$$\ln R_\alpha(D_{\mathbb{Z},\sigma_2,c}, D_{\mathbb{Z},\sigma_1,c}) \leq \frac{3\,\alpha\,\delta^2}{4\,(1-\alpha\,\delta)^2}\,(1 + O(\epsilon\,\sigma_1^4)).$$

3.4 Fixed-point arithmetic

We replace the floating-point operations of FALCON with signed fixed-point arithmetic [Yat24]. This decision is motivated by implementation considerations: fixed-point arithmetic can be implemented efficiently on platforms where floating-point arithmetic is slow, unavailable, or undesirable for constant-time and portability reasons. From a numerical perspective, floating-point format gives values with a *relative precision* over a wide dynamic range, whereas fixed-point provides a *constant absolute precision*. The precision is fixed, and does not scale with the magnitude of the represented values.

Representation A signed fixed-point value is stored as an integer, with an *implicit* scaling factor 2^{-f}. Let $f \in \mathbb{Z}^+$ be the number of fractional bits and let X be a k-bit signed integer (in two's complement). The represented real value is then

$$x = X \cdot 2^{-f}.$$

We denote the corresponding format $\mathbf{Q}_{i,f}$ with i such that $i = k - f$, and i including the sign bit (see Fig. 2). We may refer to X as the integer representation of x, k as the *word length* of the fixed-point format (in order to disambiguate from the bit-length of the platform's words, we refer to the latter as *machine word length*), and f as its *fraction length*.

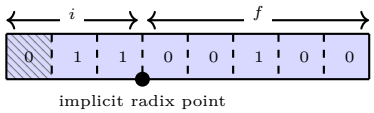

Fig. 2: A $\mathbf{Q}_{3,5}$ fixed-point number, its value is 3.125, and its sign bit is greyed-out

Equivalently, the values lie in $[-2^{i-1}, 2^{i-1} - 2^{-f}]$ with step 2^{-f}. When converting a real value to $\mathbf{Q}_{i,f}$, rounding to nearest induces an absolute error bounded by 2^{-f-1} (and truncation by $< 2^{-f}$), assuming no overflow.

Main operations. Let $x = X \cdot 2^{-f}$ and $y = Y \cdot 2^{-f}$ be in the same format $\mathbf{Q}_{i,f}$ (for simplicity).

1. **Addition and subtraction.** This is implemented as an integer add/sub, with conservative range analysis to avoid overflow.

$$x \pm y = (X \pm Y) \cdot 2^{-f}.$$

2. **Multiplication.** The product satisfies

$$xy = (XY) \cdot 2^{-2f}.$$

To return to $\mathbf{Q}_{i,f}$ we rescale by 2^f and round to nearest:

$$\mathrm{mul}_f(X, Y) = \left\lfloor \frac{XY}{2^f} \right\rceil, \qquad xy \approx \mathrm{mul}_f(X, Y) \cdot 2^{-f},$$

where $\lfloor \cdot \rceil$ denotes rounding to the nearest integer. At exact half-points ($XY \equiv 2^{f-1} \bmod 2^f$), our reference implementation breaks ties to even; the bias-and-floor variant $\left\lfloor (XY + 2^{f-1})/2^f \right\rfloor$ instead rounds ties towards $+\infty$. Either tie-break yields the same absolute error bound 2^{-f-1}. Practically, this is computed with a widened integer product (e.g., $k \times k \to 2k$) followed by a shift and rounding, as depicted in Fig. 3. The fractional length is the sum of the length of the two fractional parts, an overflow can happen for the integral part, and at the end, we have to put the value in the input format: $i_s = 2i$ and $f_s = 2f$.

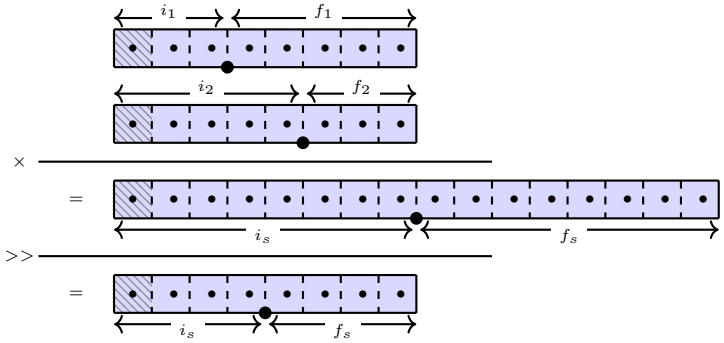

Fig. 3: Multiplication in fixed-point for different formats

3. **Shifts and rescaling.** Multiplication/division by powers of two correspond to integer shifts and are exact (up to overflow). Virtual shifts, or rescalings, can reinterpret the format by moving the radix point implicitly. Moving from the $\mathbf{Q}_{i_1,f_1}$ format to the $\mathbf{Q}_{i_2,f_2}$ format is illustrated in Fig. 4.

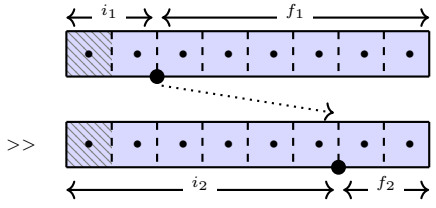

Fig. 4: Virtual right-shift over 4 bits

4. **Division via Newton–Raphson.** Division can be reduced to computing a reciprocal. For $y \neq 0$, we compute an approximation of $r \approx 1/y$ and return $x/y \approx x \cdot r$. Newton–Raphson for the reciprocal uses:

$$r_{t+1} = r_t + r_t\,(1 - y\,r_t).$$

   In fixed point, each iteration consists of a small number of multiplications and additions; the number of correct bits roughly doubles per iteration once r_t is in the convergence region. An initial approximation can be obtained by normalizing y to an interval; in our implementation we use such a normalization-based seed, and then apply a few iterations.
5. **Square root via Newton–Raphson.** Similarly, square roots can be computed with Newton–Raphson on $g(z) = z^2 - a$, yielding:

$$z_{t+1} = \frac{1}{2}\left(z_t + \frac{a}{z_t}\right).$$

   This requires one division (or reciprocal) per iteration. Alternatively, one may compute $u \approx 1/\sqrt{a}$ with

$$u_{t+1} = u_t\left(\frac{3}{2} - \frac{a}{2}\,u_t^2\right), \quad \text{then } \sqrt{a} \approx a\,u,$$

   which uses only multiplications once u_t is initialized. Given that in FALCON we only use square roots to normalize the (well-bounded) leaves of the tree (Line 7 of Algorithm 1), this method is very effective. As with reciprocal computation, convergence is quadratic and a small number of iterations suffices to reach the precision required by our bounds.

Type 1 vs Type 2 fixed-point formats. There are several flavors of fixed-point arithmetic. For example, distinct operands may or may not have the same word length k and fraction length f. Our implementations may use one of these two formats:

1. **Type 1 – uniform k and f.** This format uses uniform word length k and fraction length f across all operands. This is a simple format, however the integer length $i = k - f$ needs to be dimensioned for the worst-case variable, which may increase the word length. We use the notation `i.f` as shorthand for Type 1 format with integer length `i` and fraction length `f`.
2. **Type 2 – uniform k, per-operand f.** This format uses uniform word length k across all operands, but the fraction length $f(x)$ is per-operand x. We note that $f(x)$ is still known at compile time (otherwise it would be a floating-point format). Type 2 allows shorter word lengths than Type 1, however taking full advantage of it requires more careful analysis at the design stage, and more tedious bookkeeping of $f(x)$ at the implementation stage.

3.5 Notation on Binary Trees

A tree T is defined recursively from the *root* node.

1. $\mathsf{T}.\mathsf{leftchild}$ the left subtree of the root
2. $\mathsf{T}.\mathsf{rightchild}$ the right subtree of the root
3. $\mathsf{T}.\mathsf{value}$ the value stored in the root

A *child* node is a root that has a parent (i.e. every node except the root). A node with no child is called a *leaf*, otherwise it is an *internal node*. The *level* of a node is its distance from the root, and the *depth* of a tree is the maximal level of its nodes. A binary tree is *perfect* if all leaves have the same level ℓ, in which case the tree has $2^{\ell+1} - 1$ nodes.

4 Key Generation

This section performs a boundedness analysis for the key generation: Keygen (Section 4.1) and its subroutines NTRUGen (Section 4.2) and ffLDL* (Section 4.3).

4.1 Keygen

Keygen is the key generation algorithm in Falcon. While the verification key pk is an integer polynomial $h \in \mathbb{Z}_q[x]/(x^n + 1)$, the signing key $\mathsf{sk} = (\hat{\mathbf{B}}, \mathsf{T})$ contains a 2×2 matrix $\hat{\mathbf{B}}$ and a tree T. The entries of $\hat{\mathbf{B}}$ and T are FFT representations of polynomials $a \in \mathbb{R}[x]/(x^k + 1)$, for $k \in \{1, 2, 4, \ldots, n/2, n\}$, stored as arrays of floating-point (or, in our case fixed-point) complex values.

When using floating-point arithmetic, the precision used to store sk has an impact on the distribution of signatures output when using sk. When using fixed-point arithmetic, in addition to the precision, the magnitude of fixed-point values becomes a crucial parameter; upper bounds are given in Lemma 6.

Lemma 6 (Boundedness of Keygen). *Let* $\mathsf{sk}, \mathsf{pk} \leftarrow \mathsf{Keygen}(x^n + 1, q)$, *where* $\mathsf{sk} = (\hat{\mathbf{B}}, \mathsf{T})$. *For every node* $\hat{a}$ *in* T:

$$\begin{cases} \|\hat{a}\|_\infty \leq \gamma_{\text{root}} & \text{if } \hat{a} \text{ is the root} \\ \|\hat{a}\|_\infty < 1 & \text{if } \hat{a} \text{ is an internal node} \\ \|\hat{a}\|_\infty \leq 1.17^2 \cdot \eta_\epsilon^+(\mathbb{Z}^{2n}) & \text{if } \hat{a} \text{ is a leaf node} \end{cases} \tag{5}$$

Furthermore, every entry of $\hat{\mathbf{B}}$ *is upper-bounded by* $\max(\gamma_{f,g}, \gamma_{F,G})$.

Proof. The structure of this proof is as follows: (i) bounding the root of T, (ii) bounding the internal nodes of T, (iii) bounding the leaves of T.

Bounding the root node. The value at the root of T is exactly $\hat{k}$, where $k = \frac{F f^\star + G g^\star}{f f^\star + g g^\star}$. As per Lemma 7 (boundedness of NTRUGen), $\|\hat{k}\| \leq \gamma_{\text{root}}$.

Algorithm 1 Keygen(ϕ, q)

Require: A monic polynomial $\phi \in \mathbb{Z}[x]$, a modulus q
Ensure: A secret key sk, a public key pk

1: $f, g, F, G \leftarrow \mathsf{NTRUGen}(\phi, q)$ ▷ Solving the NTRU equation
2: $\mathbf{B} \leftarrow \left[\begin{array}{c|c} g & -f \\ \hline G & -F \end{array}\right]$
3: $\hat{\mathbf{B}} \leftarrow \mathsf{FFT}(\mathbf{B})$ ▷ `fcmul`, `fadd`
4: $\hat{\mathbf{G}} \leftarrow \hat{\mathbf{B}} \odot \hat{\mathbf{B}}^\star$ ▷ `fadd`, `fmul`
5: $\mathsf{T} \leftarrow \mathsf{ffLDL}^\star(\hat{\mathbf{G}})$ ▷ Computing the LDL* tree
6: **for each** leaf of T **do** ▷ Normalization step
7: | leaf.value $\leftarrow \sigma/\sqrt{\text{leaf.value}}$ ▷ σ as in Eq. (2). ▷ `fdiv`, `fsqrt`
8: $\mathsf{sk} \leftarrow (\hat{\mathbf{B}}, \mathsf{T})$
9: $h \leftarrow gf^{-1} \bmod q$
10: $\mathsf{pk} \leftarrow h$
11: **return** sk, pk

Bounding internal nodes. As per Lemma 8 (boundedness of ffLDL*), each internal node (i.e., any node except the root and leaves) is bounded by 1.

Bounding the leaves. After ffLDL*, each leaf contains a value a that satisfies $q/(\alpha_{\text{GPV}})^2 \leq a \leq q \cdot (\alpha_{\text{GPV}})^2$. This is true because ffLDL* is equivalent to computing the Gram-Schmidt orthogonalization of $\mathbf{B}$ [DP16]. The result follows from Line 7 of Keygen and Eq. (2).

Bounding $\hat{\mathbf{B}}$. By construction of NTRUGen (Lines 7 and 13), every entry of $\hat{\mathbf{B}}$ is upper-bounded by $\max(\gamma_{f,g}, \gamma_{F,G})$. □

4.2 NTRUGen

We recall the notations α_{GPV} and α_{hybrid} from [ENS+23]. Given polynomials f, g, $\alpha_{\text{GPV}} = \alpha_{\text{GPV}}(f, g)$ and $\alpha_{\text{hybrid}} = \alpha_{\text{hybrid}}(f, g)$ are the only positive values that satisfy:

$$(\alpha_{\text{GPV}})^2 = \max\left(\frac{1}{n}\sum_{\zeta\in\Omega_n}\frac{|f(\zeta)|^2+|g(\zeta)|^2}{q}, \frac{1}{n}\sum_{\zeta\in\Omega_n}\frac{q}{|f(\zeta)|^2+|g(\zeta)|^2}\right), \tag{6}$$

$$(\alpha_{\text{hybrid}})^2 = \max_{\zeta\in\Omega_n}\left(\max\left(\frac{|f(\zeta)|^2+|g(\zeta)|^2}{q}, \frac{q}{|f(\zeta)|^2+|g(\zeta)|^2}\right)\right), \tag{7}$$

where $\Omega_n = \{\zeta_n^i \mid i \in \mathbb{Z}_{2n}^\times, \zeta_n = \exp(i\pi/n)\}$ is the set of the n complex roots of $x^n + 1$. As observed by [ENS+23], $\alpha_{\text{GPV}}(f, g) \leq \alpha$ if and only if the embeddings of $(ff^\star + gg^\star)/q$ and of its inverse are less than α^2 on average, whereas $\alpha_{\text{hybrid}}(f, g) \leq \alpha$ if all of the embeddings of these values are at most α^2. We note that by applying norm equivalences, α_{hybrid} and α_{GPV} are equivalent up to a factor $\sqrt{n}$:

$$\alpha_{\text{GPV}} \leq \alpha_{\text{hybrid}} \leq \sqrt{n} \cdot \alpha_{\text{GPV}}. \tag{8}$$

Algorithm 2 NTRUGen(ϕ, q)

Require: A monic polynomial $\phi \in \mathbb{Z}[x]$ of degree n, a modulus q
Ensure: Polynomials f, g, F, G

1: $\sigma_{\{f,g\}} \leftarrow 1.17\sqrt{q/2n}$ ▷ Precomputed
2: **for** i from 0 to $n-1$ **do**
3: $\quad f_i \leftarrow D_{\mathbb{Z},\sigma_{\{f,g\}},0}$ ▷ Table-based
4: $\quad g_i \leftarrow D_{\mathbb{Z},\sigma_{\{f,g\}},0}$ ▷ Table-based
5: $f \leftarrow \sum_i f_i x^i$ ▷ $f \in \mathbb{Z}[x]/(\phi)$
6: $g \leftarrow \sum_i g_i x^i$ ▷ $g \in \mathbb{Z}[x]/(\phi)$
7: **if** $\|\mathsf{FFT}(f), \mathsf{FFT}(g)\|_\infty > \gamma_{f,g}$ **then restart** ▷ `fadd`, `fcmul`, `fmul`
8: **if** $\alpha_{\text{hybrid}}(f,g) > \gamma_{\text{hybrid}}$ **then restart** ▷ `fadd`, `fcmul`, `fmul`
9: **if** $\alpha_{\text{GPV}}(f,g) > 1.17$ **then restart** ▷ `fadd`, `fcmul`, `fmul`, `fdiv`
10: **if** $\mathsf{NTT}(f)$ contains 0 **then restart** ▷ Check that f is invertible mod q
11: $F, G \leftarrow \mathsf{NTRUSolve}_{n,q}(f,g)$ ▷ Computing F, G such that $fG - gF = q \bmod \phi$
12: **if** $(F,G) = \perp$ **then restart** ▷ [Por25] also imposes $\|F,G\|_\infty \leq 127$, which is good for FxP.
13: **if** $\|\mathsf{FFT}(F), \mathsf{FFT}(G)\|_\infty > \gamma_{F,G}$ **then restart** ▷ `fadd`, `fcmul`, `fmul`
14: Compute $k = \frac{F\,f^\star + G\,g^\star}{f\,f^\star + g\,g^\star}$ ▷ `fadd`, `fcmul`, `fmul`, `fdiv`
15: **if** $\|\mathsf{FFT}(k)\|_\infty > \gamma_{\text{root}}$ **then restart** ▷ Bound root node of Falcon tree
16: **return** f, g, F, G

Our tweaks: bounded checks. Algorithm 2 presents our tweaked version of NTRUGen. Compared to [PFH+20], it adds four boundedness checks on candidate keys (f, g, F, G). For Falcon-512, we take $(\gamma_{f,g}, \gamma_{\text{hybrid}}, \gamma_{F,G}, \gamma_{\text{root}}) = (255, 4, 3500, 24)$.

1. Line 7 according to a threshold $\gamma_{f,g}$
2. Line 8 according to a threshold γ_{hybrid},
3. Line 13 according to a threshold $\gamma_{F,G}$,
4. Lines 14 and 15 according to a threshold γ_{root}.

The order of checks is deliberate. Lines 7 to 10 are placed before the time-consuming call to NTRUSolve, so even if they increase the number of restarts by a noticeable amount, their impact on the running time is small. On the other hand, Lines 13 and 15 can only be placed after the call to NTRUSolve, so we set $\gamma_{F,G}$ and γ_{root} so that they almost never trigger a restart.

Also note that Line 7 is placed before Line 8, since knowing an upper bound on $\|\mathsf{FFT}(f), \mathsf{FFT}(g)\|_\infty$ ensures we can compute α_{hybrid} in fixed-point with no overflow. Similarly, after NTRUSolve, Line 13 is placed before Lines 14–15, since an upper bound on $\|\mathsf{FFT}(F), \mathsf{FFT}(G)\|_\infty$ ensures we can compute k in fixed-point with no overflow. The remaining check (Line 9) is the pre-existing Falcon Gram–Schmidt test; it is placed after Line 8, since its orthogonalized component performs a fixed-point division whose divisor is then guaranteed to be lower-bounded by $q/\gamma_{\text{hybrid}}^2$.

Impact. The impact of our tweaks on NTRUGen is given in Table 2. A visual illustration of the distributions and rejection thresholds can be found in Fig. 5. One

can see that the impact on the running time is small, and the number of restarts due to NTRUSolve failing even seems to slightly decrease, though the number of runs is too small to establish a clear trend. Furthermore, since a small α_{hybrid} correlates with a small α_{GPV}, imposing a threshold on α_{hybrid} reduces the number of restarts for the α_{GPV} check.

Table 2: Number of times each check triggered a restart, while generating 1000 keys with NTRUGen. "-" indicates the check is not present. $\hat{a}$ is shorthand for $\mathsf{FFT}(a)$. Restarts on Lines 12, 13 and 15 have a substantially higher cost.

Check	$\|\hat{f},\hat{g}\|_\infty$	α_{hybrid}	α_{GPV}	$\mathsf{NTT}(f)$	$(F,G)\neq\perp$	$\|\hat{F},\hat{G}\|_\infty$	$\|\hat{k}\|_\infty$
Line	Line 7	Line 8	Line 9	Line 10	Line 12	Line 13	Line 15
[Por25]	–	–	9911	264	102	–	–
Ours	3208	8735	3716	122	92	0	0

Lemma 7 (Boundedness of NTRUGen). *Let* $(f,g,F,G) \leftarrow \mathit{NTRUGen}(x^n+1,q)$. *It holds that:*

$$\alpha_{\text{hybrid}}(f,g) \leq \gamma_{\text{hybrid}}, \tag{9}$$

$$\|\mathit{FFT}(a)\|_\infty \leq \gamma_{f,g}, \quad \text{where } a \in \{f,g\}, \tag{10}$$

$$\|\mathit{FFT}(A)\|_\infty \leq \gamma_{F,G}, \quad \text{where } A \in \{F,G\}, \tag{11}$$

$$\|\mathit{FFT}(k)\|_\infty \leq \gamma_{\text{root}}, \quad \text{where } k = \frac{F\,f^\star + G\,g^\star}{f\,f^\star + g\,g^\star}. \tag{12}$$

Proof. The result is immediate by construction: Eq. (10) is enforced by Line 7, Eq. (9) is enforced by Line 8, Eq. (11) is enforced by Line 13, and Eq. (12) is enforced by Lines 14 and 15. □

4.3 ffLDL⋆ and LDL⋆

We study ffLDL⋆ and its subroutine LDL⋆ jointly. In fixed-point, the main challenge of ffLDL⋆ and LDL⋆ is the presence of a division in Line 2 of LDL⋆, which can produce an arbitrarily large result. Somewhat surprisingly, we show in Lemma 8 that, except at the root, the result of this division is bounded by 1.

Using the symplectic structure of the NTRU trapdoor, we are able to store only the left subtree. This technique was first described in [SZZ+22], but we give an interpretation of this technique in terms of symplecticity.

The idea is that the internal nodes of the right subtree are the opposite of the internal nodes of the left subtree, and the leaves of the right subtree are in symplectic relation (in the sense of Definition 2 below) with those of the left subtree. We thus

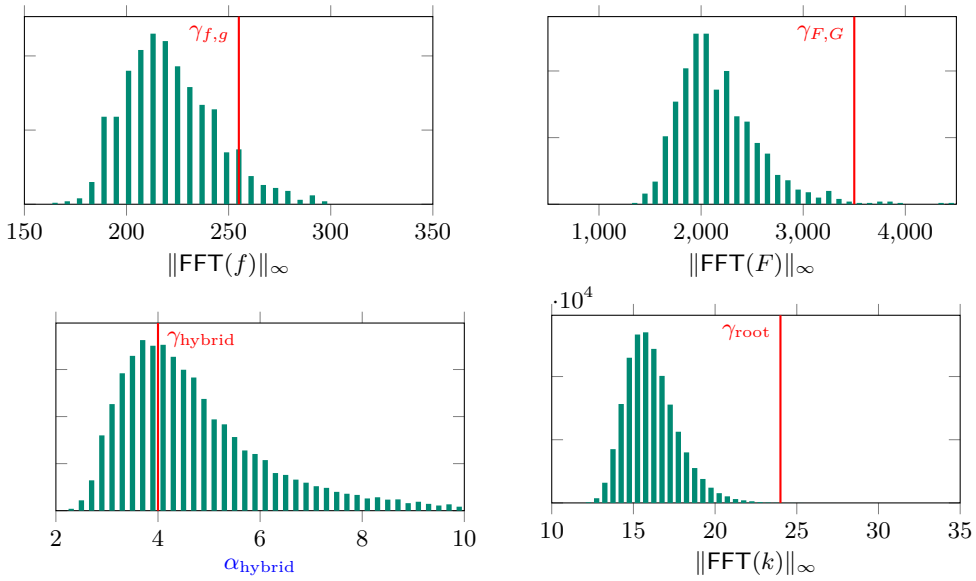

Fig. 5: Empirical distribution of the quantities $\|\mathsf{FFT}(f)\|_\infty$, $\|\mathsf{FFT}(F)\|_\infty$, α_{hybrid} and $\|\mathsf{FFT}(k)\|_\infty$ for candidate f, g, F, G generated in NTRUGen, each quantity being measured before its rejection step; discretized for readability. Red lines: rejection thresholds $\gamma_{f,g}$, $\gamma_{F,G}$, γ_{hybrid} and γ_{root}.

only generate the left subtree. This allows for a better precision analysis, as bounding the root of the left subtree is much easier than bounding the root of the right subtree, which stems from $\hat{D}_{11} = q^2 \cdot \hat{u}^{-1}$ (with $u = f\,f^\star + g\,g^\star$) and is harder to control directly. From an implementation perspective, it also cuts down the time for the generation of the tree and the storage requirements (about 40 kB for a full Falcon-512 tree in double precision) by half. For simplicity, we do not show here how to generate the tree on-the-fly, but this technique is used in the C implementation.

The ffLDL$^\star_{\mathsf{Top}}$ algorithm only generates the left subtree at the beginning of the recursion, and the Symplectree algorithm builds the right subtree. Our boundedness analysis below is stated for ffLDL*; it applies unchanged to the ffLDL$^\star_{\mathsf{Top}}$+Symplectree variant, since both compute the same tree.

Remark 1. Fig. 6 represents the tree T output by ffLDL*. The tree has depth $d = \log_2 n$, meaning there are $d+1$ levels including the root level 0. Each level $\ell \in \{0, \ldots, d\}$ contains 2^ℓ nodes, each of which has a value $\hat{a} = \mathsf{FFT}(a)$, where a is a polynomial in $\mathbb{R}[x]/(x^{n/(2^\ell)} + 1)$.

To make our proofs formal, we index intermediate values in ffLDL* and LDL* according to the recursion in which they appear: $\mathbf{G}^{[i]}, \mathbf{L}^{[i]}, \mathbf{D}^{[i]}$ and similarly for their individual entries. We set $i = 1$ the top-level call to ffLDL* and LDL*, and $i = 2 \cdot i_0$ (resp. $i = 2 \cdot i_0 + 1$) if i is the left-child (resp. right-child) recursive call to i_0. This

Algorithm 3 $\mathsf{LDL}^\star(\hat{\mathbf{G}})$

Require: A full-rank self-adjoint matrix $\hat{\mathbf{G}} = (\hat{G}_{ij}) \in \mathsf{FFT}(\mathbb{Q}[x]/(\phi))^{2\times 2}$
Ensure: The $\mathrm{LDL}^\star$ decomposition $\hat{\mathbf{G}} = \hat{\mathbf{L}} \odot \hat{\mathbf{D}} \odot \hat{\mathbf{L}}^\star$ over $\mathsf{FFT}(\mathbb{Q}[x]/(\phi))$
Format: All polynomials are in $\mathsf{FFT}()$ representation.

$\hat{D}_{00} \leftarrow \hat{G}_{00}$
$\hat{L}_{10} \leftarrow \hat{G}_{10}/\hat{G}_{00}$ ▷ `fdiv`
$\hat{D}_{11} \leftarrow \hat{G}_{11} - \hat{L}_{10} \odot \hat{L}_{10}^\star \odot \hat{G}_{00}$ ▷ `fadd`, `fmul`
$\hat{\mathbf{L}} \leftarrow \left[\begin{array}{c|c} 1 & \hat{0} \\ \hline \hat{L}_{10} & \hat{1} \end{array}\right], \hat{\mathbf{D}} \leftarrow \left[\begin{array}{c|c} \hat{D}_{00} & \hat{0} \\ \hline \hat{0} & \hat{D}_{11} \end{array}\right]$
return $(\hat{\mathbf{L}}, \hat{\mathbf{D}})$

Algorithm 4 $\mathsf{ffLDL}^\star_{\mathsf{Top}}(\hat{\mathbf{G}})$

Require: A full-rank Gram matrix $\hat{\mathbf{G}} \in \mathsf{FFT}\left(\mathbb{Q}[x]/(x^k+1)\right)^{2\times 2}$
Ensure: A binary tree T
Format: All polynomials are in FFT representation.

$(\hat{\mathbf{L}}, \hat{\mathbf{D}}) \leftarrow \mathsf{LDL}^\star(\hat{\mathbf{G}})$ ▷ Discard $\hat{\mathbf{D}}_{11}$
$\mathsf{T}.\mathsf{value} \leftarrow \hat{L}_{10}$ ▷ $\hat{\mathbf{L}} = \left[\begin{array}{c|c} \hat{1} & \hat{0} \\ \hline \hat{L}_{10} & \hat{1} \end{array}\right], \hat{\mathbf{D}} = \left[\begin{array}{c|c} \hat{D}_{00} & \hat{0} \\ \hline \hat{0} & \hat{D}_{11} \end{array}\right]$
$\hat{d}_{00}, \hat{d}_{01} \leftarrow \mathsf{splitfft}(\hat{D}_{00})$ ▷ $\hat{d}_{ij} \in \mathsf{FFT}(\mathbb{Q}[x]/(x^{k/2}+1))$
$\hat{\mathbf{G}}_0 \leftarrow \left[\begin{array}{c|c} \hat{d}_{00} & \hat{d}_{01} \\ \hline \hat{d}_{01}^\star & \hat{d}_{00} \end{array}\right]$
$\mathsf{T}.\mathsf{leftchild} \leftarrow \mathsf{ffLDL}^\star(\hat{\mathbf{G}}_0)$
$\mathsf{T}.\mathsf{rightchild} \leftarrow \mathsf{Symplectree}(\mathsf{T}.\mathsf{leftchild}, k/2)$ ▷ Right subtree derived by symplecticity [SZZ+22]
return T

is illustrated in Fig. 6. At internal nodes, normalized and raw trees coincide, so the right-node value remains $-\widehat{L}_{10}$. Only the leaf rule changes. For σ_L the leaf from the left subtree and σ_R the leaf from the right subtree, we have:

$$\sigma_R \longmapsto \frac{\sigma^2}{q\sigma_L}.$$

Symplectic pairs. In Definition 2, we define *symplectic pairs*, which generalize symplectic matrices. In Falcon, symplectic pairs are helpful in many ways:

1. In the proof of Lemma 8, we use them to lower bound divisors, which is crucial for correctness.
2. At an algorithmic level, they are used to compute half of the tree, and get the other half for free.
3. Looking ahead, we will also use symplectic pairs in Lemma 12 in order to perform the boundedness analysis of ffSampling.

Algorithm 5 $\mathsf{ffLDL}^\star(\hat{\mathbf{G}})$

Require: A full-rank Gram matrix $\hat{\mathbf{G}} \in \mathsf{FFT}\left(\mathbb{Q}[x]/(x^n+1)\right)^{2\times 2}$
Ensure: A binary tree T
Format: All polynomials are in FFT representation.

1: $(\hat{\mathbf{L}}, \hat{\mathbf{D}}) \leftarrow \mathsf{LDL}^\star(\hat{\mathbf{G}})$ ▷ $\hat{\mathbf{L}} = \left[\begin{array}{c|c} \hat{1} & \hat{0} \\ \hline \hat{L}_{10} & \hat{1} \end{array}\right], \hat{\mathbf{D}} = \left[\begin{array}{c|c} \hat{D}_{00} & \hat{0} \\ \hline \hat{0} & \hat{D}_{11} \end{array}\right]$
2: $\mathsf{T}.\mathsf{value} \leftarrow \hat{L}_{10}$
3: **if** $(n = 2)$ **then**
4: $\quad \mathsf{T}.\mathsf{leftchild} \leftarrow \hat{D}_{00}$
5: $\quad \mathsf{T}.\mathsf{rightchild} \leftarrow \hat{D}_{11}$
6: $\quad$ **return** T
7: **else**
8: $\quad \hat{d}_{00}, \hat{d}_{01} \leftarrow \mathsf{splitfft}(\hat{D}_{00})$ ▷ $\hat{d}_{ij} \in \mathsf{FFT}(\mathbb{Q}[x]/(x^{n/2}+1))$
9: $\quad \hat{d}_{10}, \hat{d}_{11} \leftarrow \mathsf{splitfft}(\hat{D}_{11})$
10: $\quad \hat{\mathbf{G}}_0 \leftarrow \left[\begin{array}{c|c} \hat{d}_{00} & \hat{d}_{01} \\ \hline \hat{d}_{01}^\star & \hat{d}_{00} \end{array}\right], \hat{\mathbf{G}}_1 \leftarrow \left[\begin{array}{c|c} \hat{d}_{10} & \hat{d}_{11} \\ \hline \hat{d}_{11}^\star & \hat{d}_{10} \end{array}\right]$
11: $\quad \mathsf{T}.\mathsf{leftchild} \leftarrow \mathsf{ffLDL}^\star(\hat{\mathbf{G}}_0)$
12: $\quad \mathsf{T}.\mathsf{rightchild} \leftarrow \mathsf{ffLDL}^\star(\hat{\mathbf{G}}_1)$
13: $\quad$ **return** T

Definition 2 (Symplectic pairs). *Given two real matrices* $\mathbf{M} \in \mathbb{R}^{2\ell\times 2\ell}$ *and* $\mathbf{N} \in \mathbb{R}^{2\ell\times 2\ell}$*, we say that* $(\mathbf{M}, \mathbf{N})$ *is a* β*-symplectic pair*[6] *if:*

$$\mathbf{M}^t \cdot \mathbf{J} \cdot \mathbf{N} = \beta \cdot \mathbf{J}, \quad \textit{where} \quad \mathbf{J} = \begin{bmatrix} 0 & \mathbf{I}_\ell \\ -\mathbf{I}_\ell & 0 \end{bmatrix}. \tag{13}$$

For diagonal matrices $\begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix}$ *and* $\begin{bmatrix} c & 0 \\ 0 & d \end{bmatrix}$*, Eq. (13) simplifies to:*

$$a \cdot d = b \cdot c = \beta. \tag{14}$$

We use symplectic pairs in order to bound the intermediate variables and outputs of $\mathsf{ffLDL}^\star$. Due to page constraints, the proof is deferred to Section B.4.

Lemma 8 (Boundedness of $\mathsf{ffLDL}^\star$). *Let (i)* $f, g, F, G \leftarrow \mathit{NTRUGen}(x^n+1, q)$*, (ii)* $\mathbf{B} \leftarrow \left[\begin{array}{c|c} g & -f \\ \hline G & -F \end{array}\right]$*, (iii)* $\hat{\mathbf{B}} \leftarrow \mathit{FFT}(\mathbf{B})$*, (iv)* $\hat{\mathbf{G}} \leftarrow \hat{\mathbf{B}} \times \hat{\mathbf{B}}^\star$*, and (v)* $T \leftarrow \mathsf{ffLDL}^\star(\hat{\mathbf{G}})$*, as in Lines 1 to 5 of Keygen. For every real matrix* $\hat{\mathbf{D}}^{[i]}$*, with* $i \geq 1$*, each of its diagonal coefficients* $(\hat{d}_{jj})_j$ *is upper- and lower-bounded as:*

$$q/(\alpha_{\text{hybrid}})^2 \leq \hat{d}_{jj} \leq q \cdot (\alpha_{\text{hybrid}})^2. \tag{15}$$

Furthermore, every non-root $\hat{L}_{10}^{[i]}$*, with* $i \geq 2$*, is upper-bounded as follows:*

$$\|\hat{L}_{10}\|_\infty \leq \frac{(\alpha_{\text{hybrid}})^4 - 1}{(\alpha_{\text{hybrid}})^4 + 1} < 1. \tag{16}$$

[6] This terminology is inspired by the fact that a matrix $\mathbf{M}$ is said to be symplectic if $\mathbf{M}^t \cdot \mathbf{J} \cdot \mathbf{M} = \mathbf{J}$. In particular, $\mathbf{M}$ is symplectic if and only if $(\mathbf{M}, \mathbf{M})$ is a 1-symplectic pair.

Algorithm 6 Symplectree(T_1, k)

Require: A binary Falcon tree T_1, a power-of-two ring degree k
Ensure: A binary Falcon tree T_2, in symplectic relation with T_1
Format: All polynomials are in FFT representation. This algorithm can be done on-the-fly during signing.

$\mathsf{T}_2.\mathsf{value} \leftarrow -\mathsf{T}_1.\mathsf{value}$
if $(k = 2)$ **then**
  $\mathsf{T}_2.\mathsf{leftchild} \leftarrow \frac{q^2}{\mathsf{T}_1.\mathsf{rightchild}}$ ▷ fdiv
  $\mathsf{T}_2.\mathsf{rightchild} \leftarrow \frac{q^2}{\mathsf{T}_1.\mathsf{leftchild}}$ ▷ fdiv
  return T_2
$\mathsf{T}_2.\mathsf{leftchild} \leftarrow$ Symplectree($\mathsf{T}_1.\mathsf{rightchild}, k/2$)
$\mathsf{T}_2.\mathsf{rightchild} \leftarrow$ Symplectree($\mathsf{T}_1.\mathsf{leftchild}, k/2$)
return T_2

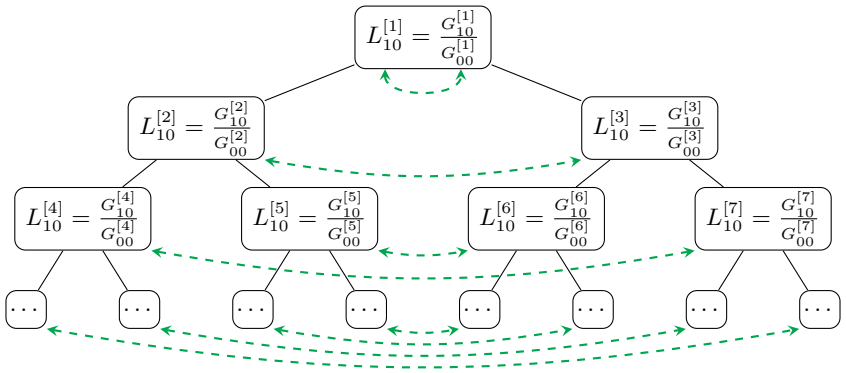

Fig. 6: Falcon Tree T. q^2-symplectic pairs $(\hat{\mathbf{D}}^{[i]}, \hat{\mathbf{D}}^{[i']})$ (in the sense of Definition 2) between matrices are illustrated in green dashed arrows. The tree nodes store the $\hat{L}_{10}^{[i]}$, while the pairing relates the associated diagonal matrices $\hat{\mathbf{D}}^{[i]}$; the self-loop on the root reflects that $\hat{\mathbf{D}}^{[1]}$ forms a q^2-symplectic pair with itself.

The root $i = 1$ is excluded from Eq. (16): $\hat{L}_{10}^{[1]} = \hat{k}$ does not arise from a splitfft and may exceed 1; its magnitude is instead controlled by the dedicated γ_{root} check in NTRUGen (Line 15 of Algorithm 2).

5 Signing

5.1 Sign

Sign produces a target $\mathbf{t}$ from the message, and feeds it to its subroutine ffSampling. We propose a simple tweak to Sign which improves the boundedness of ffSampling's input. We notice that $D_{\Lambda(\mathbf{B}),\sigma,\mathbf{t}\cdot\mathbf{B}} = \mathbf{t}_{\text{int}} \cdot \mathbf{B} + D_{\Lambda(\mathbf{B}),\sigma,\mathbf{t}_{\text{frac}}\cdot\mathbf{B}}$ which is what allows us to gain precision here. We show in Lemma 9 that a Gaussian sampling on $\mathbf{t}_{\text{frac}}$ is equivalent to a Gaussian sampling on $\mathbf{t}$: the signature is independent of $\mathbf{t}_{\text{int}}$.

Sign with the NTT-based target construction. Let $\mathbf{t} = \mathbf{t}_{\text{int}} + \mathbf{t}_{\text{frac}}$ be the integer/fractional decomposition (cf. Lemma 9) of the target $\mathbf{t} = \frac{c}{q} \cdot (-F, f)$, and let $\mathbf{qt}_{\text{frac}} := q \cdot \mathbf{t}_{\text{frac}} \in \mathbb{Z}^{2n}$. Since $q \cdot \mathbf{t} = c \cdot (-F, f)$ and $\|\mathbf{qt}_{\text{frac}}\|_\infty \leq q/2$, we have $\mathbf{qt}_{\text{frac}} \equiv (-c \cdot F,\ c \cdot f) \bmod^{\pm} q$, so we can compute it directly by a pair of NTT multiplications modulo q, avoiding inexact FFT operations. Algorithm 7 implements this construction. The remainder of the algorithm is unchanged compared to the original specification [PFH+20].

Lemma 9 shows that the principle applied via our tweak results in a distributional equivalence. This goes a bit further, and KAT vectors can, in principle, be preserved. Indeed, making the randomness rand of ffSampling explicit, for any $\mathbf{t}_{\text{int}}$ being the FFT of an integer vector, it holds that $\text{ffSampling}(\mathbf{t}_{\text{frac}} + \mathbf{t}_{\text{int}}, \mathsf{T}; \mathsf{rand}) = \mathbf{t}_{\text{int}} + \text{ffSampling}(\mathbf{t}_{\text{frac}}, \mathsf{T}; \mathsf{rand})$. However, the paragraph "*LTYZ25 divergences and a fix*" below shows that reality is a bit more subtle.

Algorithm 7 Sign (m, sk, $\lfloor\beta^2\rfloor$), NTT-based target

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

```
r ← {0,1}^320 uniformly
c ← HashToPoint(r‖m, q, n)
qt_frac ← ((−c·F) mod± q, (c·f) mod± q)        ▷ via NTT; integer, ‖qt_frac‖∞ ≤ q/2
t_frac ← FFT(qt_frac) · 1/q                    ▷ fadd, fcmul
do
    do
        z ← ffSampling_n(t_frac, T)            ▷ ‖t_frac‖∞ ≤ 1/2 in coefficient domain
        s = (t_frac − z)B̂                      ▷ fadd, fmul
    while ‖s‖² > ⌊β²⌋
    (s1, s2) ← invFFT(s)                       ▷ fadd, fcmul
    s ← Compress(s2, 8·sbytelen − 328)
while (s = ⊥)
return sig = (r, s)
```

Lemma 9 (Correctness of our tweaked Sign). *Let $\Lambda(\mathbf{B})$ be our lattice and $\mathbf{t}_{\text{int}} \in \mathbb{Z}^{2n}$ and $\mathbf{t}_{\text{frac}} \in [-\frac{1}{2}, \frac{1}{2}]^{2n}$, such that $\mathbf{t} = \mathbf{t}_{\text{frac}} + \mathbf{t}_{\text{int}}$. Then:*

$$D_{\Lambda(\mathbf{B}),\sigma,\mathbf{t}\cdot\mathbf{B}} = \mathbf{t}_{\text{int}} \cdot \mathbf{B} + D_{\Lambda(\mathbf{B}),\sigma,\mathbf{t}_{\text{frac}}\cdot\mathbf{B}}. \tag{17}$$

Proof. Let $\mathbf{z} \in \mathbb{Z}^{2n}$; for brevity, we write $\rho_{\mathbf{c}}(\mathbf{x}) := \rho_\sigma(\mathbf{x} - \mathbf{c})$. The probability of $\mathbf{z}$ being sampled from $D_{\Lambda,\mathbf{t}_{\text{frac}}+\mathbf{t}_{\text{int}}}$ is:

$$\frac{\rho_{(\mathbf{t}_{\text{int}}+\mathbf{t}_{\text{frac}})\cdot\mathbf{B}}(\mathbf{z}\cdot\mathbf{B})}{\sum_{\mathbf{x}\in\Lambda}\rho_{(\mathbf{t}_{\text{int}}+\mathbf{t}_{\text{frac}})\cdot\mathbf{B}}(\mathbf{x})} = \frac{\rho_{(\mathbf{t}_{\text{int}}+\mathbf{t}_{\text{frac}})\cdot\mathbf{B}}(\mathbf{z}\cdot\mathbf{B})}{\sum_{\mathbf{x}\in\Lambda}\rho_{\mathbf{t}_{\text{frac}}\cdot\mathbf{B}}(\mathbf{x})} = \frac{\rho_{\mathbf{t}_{\text{frac}}\cdot\mathbf{B}}\left((\mathbf{z}-\mathbf{t}_{\text{int}})\cdot\mathbf{B}\right)}{\sum_{\mathbf{x}\in\Lambda}\rho_{\mathbf{t}_{\text{frac}}\cdot\mathbf{B}}(\mathbf{x})}. \tag{18}$$

This is equal to the probability of $\mathbf{z}$ being sampled from $D_{\Lambda(\mathbf{B}),\sigma,\mathbf{t}_{\text{frac}}\cdot\mathbf{B}} + \mathbf{t}_{\text{int}} \cdot \mathbf{B}$. □

LTYZ25 divergences and a fix. When applying the tweak in Algorithm 7, we started observing noticeable discrepancies between the reference floating-point implementation and ours: both implementations produced valid, yet distinct signatures. The rate of divergence was about 1/8000.

While we initially thought this was due to an implementation mistake, we found that applying the tweak to Pornin's "c-fn-dsa" implementation (`https://github.com/pornin/c-fn-dsa`) resulted in a similar divergence rate, suggesting that it is independent of the choice of representation (floating-point or fixed-point). Upon closer inspection, we found that whenever the standard and tweaked signing processes diverged, they did at either the *first two or last two* calls to SamplerZ, and this happened when the corresponding centers c_i were exact integers.

As it turns out, this behavior was predicted and studied in a paper by Lin, Tibouchi, Yu and Zhang [LTYZ25], or LTYZ25. Heuristic 1 of [LTYZ25] posits that the first two (resp. last two) calls to SamplerZ have a probability $1/q$ (resp. $1/\|g, -f\|^2$) of having a center c_i that is an exact integer. Since the first step of SamplerZ computes $\lfloor c_i \rfloor$, if two implementations compute ever-so-slightly distinct centers $c_i + \epsilon$ and $c_i - \epsilon$, then they will diverge no matter how small ϵ is. This is exactly what happened in our case. For Falcon, we have $1/q + 1/\|g, -f\|^2 \approx 1/7101$, which is consistent with our empirical divergence rate of about 1/8000. [LTYZ25] presented examples of implementation choices (fused multiply-add instructions, emulated floating-point arithmetic) that can trigger this behavior, and our tweak is another such example.

Section 7.1 of [LTYZ25] suggests to fix this by applying two joint fixes:

1. Tweaking SamplerZ to apply rounding rather than truncating;
2. Forcing $\|g, -f\|^2$ odd.

After applying both fixes, the divergence completely disappeared. We note that these fixes, especially the first one, result in a different set of KATs than those of the Falcon reference implementation. Nevertheless, our take-away from this experience is that slight idiosyncrasies of the code can result in KAT divergences at a rate of up to 1/8000, even when two versions are mathematically equivalent, and even with arbitrarily high precision. The only robust countermeasure, in our opinion, is to apply the LTYZ25 fixes [LTYZ25] to the final FN-DSA standard.

5.2 ffSampling

This section performs a boundedness analysis of ffSampling. We first recall in Lemma 10 a result from [FGdG+26], stating that the output distribution of ffSampling is statistically close to a true discrete Gaussian. We make explicit a few assumptions that are implicit in the original lemma.

Lemma 10 (Distribution of ffSampling, Lemma 7 of [FGdG+26] adapted). *Let* $\mathbf{B} \in (\mathbb{Q}[x]/(x^k+1))^{2\times 2}$, $\hat{\mathbf{B}} \leftarrow \mathit{FFT}(\mathbf{B})$, $\hat{\mathbf{G}} \leftarrow \hat{\mathbf{B}} \cdot \hat{\mathbf{B}}^\star$, *and* $T \leftarrow \mathsf{ffLDL}^\star(\hat{\mathbf{G}})$. *Let* $\epsilon \in (0, 1/4)$, *a standard deviation* $\sigma \geq \eta_\epsilon(\mathbb{Z}^{2k}) \cdot \|\mathbf{B}\|_{GS}$, *and* $\mathbf{t} \in (\mathbb{Q}[x]/(x^k+1))^2$. *Let*

Algorithm 8 $\mathsf{ffSampling}_k(\hat{\mathbf{t}}, \mathsf{T})$

Require: $\hat{\mathbf{t}} = (\hat{t}_0, \hat{t}_1) \in \mathsf{FFT}\left(\mathbb{Q}[x]/(x^k+1)\right)^2$, a FALCON tree T
Ensure: $\hat{\mathbf{z}} = (\hat{z}_0, \hat{z}_1) \in \mathsf{FFT}\left(\mathbb{Z}[x]/(x^k+1)\right)^2$
Format: All polynomials are in FFT representation.

if $k = 1$ **then**
  $\sigma' \leftarrow \mathsf{T.value}$ ▷ It holds that $\sigma' \in [\sigma_{\min}, \sigma_{\max}]$.
  $\hat{z}_0 \leftarrow \mathsf{SamplerZ}(\hat{t}_0, \sigma')$ ▷ For $k = 1$, $(\hat{t}_i, \hat{z}_i) = t_i, z_i$. Assume $z_0 \sim D_{\mathbb{Z},\sigma',t_0}$
  $\hat{z}_1 \leftarrow \mathsf{SamplerZ}(\hat{t}_1, \sigma')$ ▷ Assume $z_1 \sim D_{\mathbb{Z},\sigma',t_1}$
  return $\hat{\mathbf{z}} = (\hat{z}_0, \hat{z}_1)$
$(\hat{\ell}, \mathsf{T}_0, \mathsf{T}_1) \leftarrow (\mathsf{T.value}, \mathsf{T.leftchild}, \mathsf{T.rightchild})$
$\hat{\mathbf{t}}_1 \leftarrow \mathsf{splitfft}(\hat{t}_1)$ ▷ $\hat{\mathbf{t}}_1 \in \mathsf{FFT}\left(\mathbb{Q}[x]/(x^{k/2}+1)\right)^2$, likewise $\hat{\mathbf{t}}_0$ below. ▷ `fadd`, `fcmul`.
$\hat{\mathbf{z}}_1 \leftarrow \mathsf{ffSampling}_{k/2}(\hat{\mathbf{t}}_1, \mathsf{T}_1)$ ▷ First recursive call to $\mathsf{ffSampling}_{k/2}$
$\hat{z}_1 \leftarrow \mathsf{mergefft}(\hat{\mathbf{z}}_1)$ ▷ $\hat{\mathbf{z}}_1 \in \mathsf{FFT}\left(\mathbb{Z}[x]/(x^{k/2}+1)\right)^2$, likewise $\hat{\mathbf{z}}_0$ below. ▷ `fadd`, `fcmul`.
$\hat{t}'_0 \leftarrow \hat{t}_0 + (\hat{t}_1 - \hat{z}_1) \odot \hat{\ell}$ ▷ `fadd`, `fmul`.
$\hat{\mathbf{t}}_0 \leftarrow \mathsf{splitfft}(\hat{t}'_0)$ ▷ `fadd`, `fcmul`.
$\hat{\mathbf{z}}_0 \leftarrow \mathsf{ffSampling}_{k/2}(\hat{\mathbf{t}}_0, \mathsf{T}_0)$ ▷ Second recursive call to $\mathsf{ffSampling}_{k/2}$
$\hat{z}_0 \leftarrow \mathsf{mergefft}(\hat{\mathbf{z}}_0)$ ▷ `fadd`, `fcmul`.
return $\hat{\mathbf{z}} = (\hat{z}_0, \hat{z}_1)$

$\hat{\mathbf{z}} \leftarrow \mathit{ffSampling}(\hat{\mathbf{t}}, T)$, $\mathbf{s} := (\mathbf{t} - \mathbf{z}) \cdot \mathbf{B}$, D_1 *be the distribution of* $\mathbf{s}$ *and* $D_0 = D_{\Lambda_{\mathbf{t}},\sigma}$ *be the centered Gaussian distribution of parameter* σ *discretized over the coset* $\Lambda_{\mathbf{t}} := \Lambda(\mathbf{B}) + \mathbf{t} \cdot \mathbf{B}$. *Let us note* $a \lesssim b$ *as shorthand for* $a \leq b \cdot (1 + o(1))$. *For any* $\mathbf{s}^*$ *in* $\Lambda_{\mathbf{t}}$*:*

$$\exp(-2\epsilon) \lesssim \left(\frac{1 - \epsilon/2k}{1 + \epsilon/2k}\right)^{2k} \leq \frac{D_1(\mathbf{s}^*)}{D_0(\mathbf{s}^*)} \leq \left(\frac{1 + \epsilon/2k}{1 - \epsilon/2k}\right)^{2k} \lesssim \exp(2\epsilon). \tag{19}$$

Lemma 12 bounds with overwhelming probability all intermediate variables in ffSampling. The proof expresses them as inner products between a (translated) discrete Gaussian and a fixed vector. This allows to bound them using concentration bounds (Lemma 1). Lemma 12 uses a technical lemma (Lemma 11) on the coefficients of inverse square roots of the matrices $\mathbf{G}^{[i]}$ appearing in ffLDL*. The proof of Lemma 12 is deferred to Section B.5.

Lemma 11. *For* $i \geq 1$, *let* $\hat{\mathbf{G}} = \hat{\mathbf{G}}^{[i]} \in \mathit{FFT}\left(\mathbb{Q}[x]/(x^k+1)\right)^{2\times 2}$ *be as appearing in* ffLDL* *(see Remark 1). There exists a factorization* $\hat{\mathbf{G}} = \hat{\mathbf{B}} \cdot \hat{\mathbf{B}}^\star$, *such that* $\mathbf{B} \in \left(\mathbb{R}[x]/(x^k+1)\right)^{2\times 2}$ *and every complex entry of* $\hat{\mathbf{B}}^{-1}$ *is bounded by* B_{inv}*:*

$$B_{\text{inv}} = \frac{\gamma_{F,G}}{q} \quad \textit{if } i = 1, \qquad B_{\text{inv}} = \frac{\alpha_{\text{hybrid}}}{\sqrt{q}} \quad \textit{otherwise.} \tag{20}$$

Proof. For the special case $i = 1$, if $\mathbf{B} = \mathbf{B}^{[1]}$, then $\mathbf{B}^{-1} = \frac{1}{q} \cdot \begin{bmatrix} -F & f \\ -G & g \end{bmatrix}$ and the entries of $\hat{\mathbf{B}}^{-1}$ are therefore bounded by $\frac{\gamma_{F,G}}{q}$.

Now we study the general case. The factorization of $\hat{\mathbf{G}}$ is immediate from LDL* (Algorithm 3): if we note $\hat{\mathbf{G}} = \begin{bmatrix} \hat{G}_{00} & \hat{G}_{01} \\ \hat{G}_{01}^\star & \hat{G}_{11} \end{bmatrix}$, then

$$\hat{\mathbf{G}} = \hat{\mathbf{L}} \cdot \hat{\mathbf{D}} \cdot \hat{\mathbf{L}}^\star, \text{ where } \hat{\mathbf{L}} = \begin{bmatrix} 1 & 0 \\ \hat{L}_{10} & 1 \end{bmatrix}, \hat{\mathbf{D}} = \begin{bmatrix} \hat{D}_{00} & 0 \\ 0 & \hat{D}_{11} \end{bmatrix}, \begin{cases} \hat{L}_{10} = \frac{\hat{G}_{10}}{\hat{G}_{00}}, \\ \hat{D}_{00} = \hat{G}_{00}, \\ \hat{D}_{11} = \hat{G}_{11} - \hat{L}_{10}^\star \hat{G}_{10} \end{cases} \tag{21}$$

We may set $\hat{\mathbf{B}} = \hat{\mathbf{L}} \cdot \sqrt{\hat{\mathbf{D}}}$ ($\hat{\mathbf{D}}$ is diagonal, and its diagonal entries are positive in $\mathbb{R}$). Therefore

$$\hat{\mathbf{B}}^{-1} = \hat{\mathbf{D}}^{-1/2} \cdot \hat{\mathbf{L}}^{-1} = \begin{bmatrix} \hat{D}_{00}^{-1/2} & 0 \\ 0 & \hat{D}_{11}^{-1/2} \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 \\ -\hat{L}_{10} & 1 \end{bmatrix} = \left[\begin{array}{c|c} \hat{D}_{00}^{-1/2} & 0 \\ \hline -\hat{L}_{10} \cdot \hat{D}_{11}^{-1/2} & \hat{D}_{11}^{-1/2} \end{array}\right] \tag{22}$$

Following Lemma 8, each complex entry of $\hat{\mathbf{B}}^{-1}$ is upper-bounded by $\alpha_{\text{hybrid}}/\sqrt{q}$. □

Lemma 12 (Boundedness of ffSampling). *Let $k \leq n$ be a power of two. Suppose that $\|\hat{\mathbf{t}}^{[1]}\|_\infty \leq B$. For any $i \geq 1$, the intermediate variables $\hat{\mathbf{z}} = \hat{\mathbf{z}}^{[i]}$ and $\hat{\mathbf{t}} = \hat{\mathbf{t}}^{[i]}$ in ffSampling are bounded with probability $\geq 1 - 2^{-\lambda}$:*

$$\|\hat{\mathbf{t}}\|_\infty \leq B + 2\, A_{\text{child}} \sqrt{n} \left(\gamma_{\text{root}} + \sqrt{2} + 1\right) \tag{23}$$

$$\|\hat{\mathbf{z}}\|_\infty \leq \|\hat{\mathbf{t}}\|_\infty + A \cdot \sqrt{k}, \begin{cases} A = A_{\text{root}} \coloneqq 2\,\tau\,\sigma \cdot \frac{\gamma_{F,G}}{q} & \text{if } i = 1, \\ A = A_{\text{child}} \coloneqq 2\,\tau\,\sigma \cdot \frac{\alpha_{\text{hybrid}}}{\sqrt{q}} & \text{if } i \geq 2, \end{cases} \tag{24}$$

where $\tau = \sqrt{2(\lambda + \log_2(2n \log n) + 1) \log 2}$.

5.3 SamplerZ

Due to space constraints, the formal description of SamplerZ is given in Section A. However, we provide bounds on its internal variables in Lemma 13.

Lemma 13 (Boundedness of variables in SamplerZ and its subroutines). *Let $x_{\max} = \frac{19^2}{2\sigma_{\min}^2} - \frac{18^2}{2\sigma_{\max}^2}$, where 19 and 18 are the extremal values of $|z|$ and z_0 in SamplerZ. Note that $x_{\max} < 62$ for Falcon parameters. Suppose that the input $\mu \in \mathbb{R}$ of SamplerZ(Algorithm 9) is B-bounded. Then:*

1. *In SamplerZ, every floating/fixed-point variable is B_1-bounded, for $B_1 = B + x_{\max}$, and its output is B_2-bounded, for $B_2 = B + 19$.*
2. *In the BerExp subroutine of SamplerZ, z is an integer in $\{0, \ldots, 2^{64} - 1\}$, and the comparison variable w is an integer in $\{-255, \ldots, 255\}$. The floating/fixed-point variables s, r are B_3-bounded, for $B_3 = x_{\max}/\ln(2)$.*
3. *In the ApproxExp subroutine of BerExp, the input x is $\ln(2)$-bounded. The other variables y, z are integers in $\{0, \ldots, 2^{63}\}$.*

5.4 Distribution of Fixed-Point Falcon

We first recall a result from [HPRR20]. At its heart, Lemma 14 is a Rényi divergence argument, casting the security game as a single algorithm making Q_s calls to distributions. The main difference compared to usual Rényi divergence arguments is that Lemma 14 handles the case where the i-th call depends on the previous one.

Lemma 14 (Lemma 11 in [HPRR20]). *Consider a cryptosystem $C_{\mathcal{P}}$ in which a Q_s-uple $(x_0, \dots x_{Q_s-1})$ is drawn according to a distribution $\mathcal{P}$. Assume that $C_{\mathcal{P}}$ is λ-bit secure against a search problem. Let us consider a copy of the latter cryptosystem, denoted $C_{\mathcal{Q}}$, where the only difference is that $(x_0, \dots x_{Q_s-1})$ is drawn according to a distribution $\mathcal{Q}$ such that $Supp(\mathcal{Q}) \subseteq Supp(\mathcal{P})$.*

We denote $(x_0, \dots x_{Q_s-1})$ the random variables corresponding to $(x_0, \dots x_{Q_s-1})$ that follow $\mathcal{P}$ or $\mathcal{Q}$ depending on the case. For $0 \leq i < Q_s$, we denote by $\mathcal{P}_{i|<i}(\cdot|x_{<i})$ (resp. $\mathcal{Q}_{i|<i}(\cdot|x_{<i})$) the conditional distribution of x_i given the knowledge of $(x_1, \dots, x_{i-1}) = x_{<i}$ under $\mathcal{P}$ (resp. $\mathcal{Q}$). If, for all $0 \leq i < Q_s$, and for all i-uple $x_{<i}$ in the support of $\mathcal{Q}$ restricted to its first i variables,

$$R_{2\lambda-1}(\mathcal{Q}_{i|x_{<i}}, \mathcal{P}_{i|x_{<i}}) \leq 1 + \frac{1}{4Q_s}, \tag{25}$$

then $C_{\mathcal{Q}}$ is $(\lambda - 1)$-bit secure against the search problem.

On SamplerZ. Theorems 1 and 2 model SamplerZ as an ideal Gaussian sampler. In practice, SamplerZ is an adaptation of a sampler from [HPRR20], which is shown in their Theorem 7 to be statistically close to an ideal Gaussian sampler. For pragmatic implementation reasons, SamplerZ makes one tweak (Line 3 of BerExp) that voids out Theorem 7 of [HPRR20], preventing us from directly plugging in SamplerZ in these theorems.[7] However, we do not expect this tweak to make any change in the concrete security of SamplerZ, therefore we assume that the bounds obtained via Theorems 1 and 2 are applicable to the real Falcon signature scheme.

Theorem 1. *Consider Falcon $=$ (Keygen, Sign, Verify) the Falcon signature implemented with arbitrary precision, and $\overline{\text{Falcon}} =$ ($\overline{\text{Keygen}}$, $\overline{\text{Sign}}$, Verify) be Falcon implemented in fixed-point arithmetic, where Sign (resp. $\overline{\text{Sign}}$) can be queried at most Q_s times. We assume that SamplerZ is a perfect Gaussian: for $z \leftarrow \text{SamplerZ}(t, \sigma')$, $z \sim D_{\mathbb{Z},\sigma',t}$. During an execution of Sign, the center of each call to SamplerZ (resp. the restart decisions) is a deterministic function of the message and of the history (the salts drawn and the integers returned by the previous calls). Assume there are global constants Δ_σ and $\Delta_{\mathbf{t}}$ such that:*

[7] Line 3 caps s at 63, so that the acceptance probability of BerExp is lower bounded by about 2^{-64}. Therefore the sampled distribution has a slightly heavier tail than the ideal one. We do not expect this to impact security, although proving it formally is a non-trivial and interesting question.

1. *For any (f, g, F, G) output by NTRUGen, the leaves $(\sigma_i)_{i\in[2\,n]}$ and $(\overline{\sigma}_i)_{i\in[2\,n]}$ of the trees computed from (f, g, F, G) in exact and fixed-point arithmetic, respectively, satisfy:*

$$\left|\frac{\overline{\sigma}_i}{\sigma_i} - 1\right| \leq \Delta_\sigma.$$

2. *For $(\mathsf{sk}, \overline{\mathsf{sk}})$ as in the previous item, for any message and any history, the centers t_i and $\overline{t}_i$ computed by Sign and $\overline{\text{Sign}}$ from the same history satisfy:*

$$|\overline{t}_i - t_i| \leq \Delta_{\mathbf{t}}.$$

Furthermore, let $\alpha = 2\lambda - 1$, and let $\epsilon < 1/4$ be a smoothing loss such that $\sigma_{\min} \geq \max(1, \sqrt{1 + 3\,\alpha\,\Delta_\sigma} \cdot \eta_\epsilon(\mathbb{Z}))$. Let $N = 2\,n\,Q_s(1 + c)$ for a small constant $c < 1$. Assume that $\alpha\,\Delta_\sigma \leq 1/8$ and that:

$$\exp\left(3\,\alpha\,\Delta_\sigma^2\,(1 + O(\epsilon + \alpha\,\Delta_\sigma))\right) \leq 1 + \frac{1}{4\,N}, \tag{26}$$

$$\exp\left(\frac{\alpha\,\Delta_{\mathbf{t}}^2}{2}\,(1 + O(\epsilon + \Delta_\sigma + 1/\alpha))\right) \leq 1 + \frac{1}{4\,N}. \tag{27}$$

If Falcon has a bit-security λ in the UF-CMA game (unforgeability against chosen-message attacks), then $\overline{\text{Falcon}}$ has a bit-security at least $\lambda - 2$ in the same game.

Proof. Our goal is to apply Lemma 14 twice. Falcon is a cryptosystem making at most N queries to SamplerZ, modeled as a perfect Gaussian sampler $D_{\mathbb{Z},\sigma_i,t_{i,j}}$, where $i \in [2\,n]$ and $j \in [Q'_s]$ where $Q'_s = Q_s(1 + c)$. Note that a signing query restarts whenever the candidate signature is too long or fails to compress (Line 9 and 12 of Algorithm 7). Both events have probability at most 2^{-14} per iteration for Falcon-512 [PFH+20], which we capture with the additive term c.

First hop. We introduce an intermediate $\overline{\overline{\text{Falcon}}}$, where the leaves of T are swapped to $(\overline{\sigma}_i)_i$, but the algorithm is otherwise identical to Falcon. This swaps each $D_{\mathbb{Z},\sigma_i,t_{i,j}}$ for $D_{\mathbb{Z},\overline{\sigma}_i,t_{i,j}}$, whose divergence we bound with Lemma 5. Let $\delta_i := |\sigma_i^2/\overline{\sigma}_i^2 - 1|$ be the relative error on σ_i^2, so that $\delta_i \leq 2\,\Delta_\sigma + O(\Delta_\sigma^2)$; in particular $\delta_i \leq 3\,\Delta_\sigma$, so that $\sigma_i \geq \sigma_{\min} \geq \sqrt{1 + \alpha\,\delta_i} \cdot \eta_\epsilon(\mathbb{Z})$ and Lemma 5 applies. We have:

$$\forall i, j, \quad R_\alpha(D_{\mathbb{Z},\overline{\sigma}_i,t_{i,j}}, D_{\mathbb{Z},\sigma_i,t_{i,j}}) \leq \exp\left(3\,\alpha\,\Delta_\sigma^2\,(1 + O(\epsilon + \alpha\,\Delta_\sigma))\right).$$

Since Eq. (26) is true, we can apply Lemma 14 and $\overline{\overline{\text{Falcon}}}$ is unforgeable under chosen-message attacks with at most a 1-bit loss.

Second hop. Swapping from $\overline{\overline{\text{Falcon}}}$ to $\overline{\text{Falcon}}$ is equivalent to replacing every $t_{i,j}$ by $\overline{t}_{i,j}$. We apply Lemma 4 at $\sigma = \overline{\sigma}_i \geq \sigma_{\min}\,(1 - \Delta_\sigma) \geq \eta_\epsilon(\mathbb{Z})$, folding its $\frac{\alpha+1}{\alpha-1} = 1 + O(1/\alpha)$ and $1/\overline{\sigma}_i^2 = 1 + O(\Delta_\sigma)$ into the $O(\cdot)$:

$$\forall i, j, \quad R_\alpha(D_{\mathbb{Z},\overline{\sigma}_i,\overline{t}_{i,j}}, D_{\mathbb{Z},\overline{\sigma}_i,t_{i,j}}) \leq \exp\left(\frac{\alpha\,\Delta_{\mathbf{t}}^2}{2}\,(1 + O(\epsilon + \Delta_\sigma + 1/\alpha))\right).$$

Since Eq. (27) is true, we can apply Lemma 14 and $\overline{\text{FALCON}}$ is unforgeable under chosen-message attacks with at most a 2-bit loss. The second application of Lemma 14 is made at the security level $\lambda - 1$ reached after the first hop, so it nominally requires the order $2(\lambda-1)-1$; the bound above holds a fortiori at that order, since R_α is non-decreasing in α (Item 3 of Lemma 3). □

Instead of working with the worst-case error for the Rényi divergence, as is done in [HvVS+26], we can work instead with the average error, by leveraging the multiplicativity of the Rényi divergence. In Theorem 2 below, we show that the suitable average is the quadratic mean, also known as the root mean square.

We will first need two tools. The first one is a variant of Lemma 14, in which the per-call conditions (25) are replaced by a single bound on the joint divergence; its proof is deferred to Section B.6.

Lemma 15 (Slight generalization of [HPRR20, Lemma 11]). *Consider a cryptosystem $C_\mathcal{P}$ in which an N-uple $\mathbf{x}$ is drawn according to a distribution $\mathcal{P}$. Assume that $C_\mathcal{P}$ is λ-bit secure against a search problem. Let $C_\mathcal{Q}$ be a copy of this cryptosystem, where the only difference is that $\mathbf{x}$ is drawn according to a distribution $\mathcal{Q}$ such that $Supp(\mathcal{Q}) \subseteq Supp(\mathcal{P})$. If*

$$R_{2\lambda-1}(\mathcal{Q},\mathcal{P}) \leq \left(1+\frac{1}{4N}\right)^N, \tag{28}$$

then $C_\mathcal{Q}$ is $(\lambda-1)$-bit secure against the search problem.

The second tool is a chain rule for the Rényi divergence of adaptively generated tuples. Bounding each factor by its supremum over the histories $x_{<i}$ recovers the iterated Proposition 4 of [HPRR20]; the point of this version is that the supremum is taken over full executions of the pathwise product, so that the per-call divergences may be averaged along each execution, which is exactly what the RMSE hypotheses of Theorem 2 provide.

Lemma 16 (Chain rule along executions). *Let $\alpha > 1$. Let $\mathcal{Q},\mathcal{P}$ be two distributions over tuples $x = (x_1,\ldots,x_m)$, specified by their conditionals $\mathcal{Q}_{i|x_{<i}}$ and $\mathcal{P}_{i|x_{<i}}$, with $Supp(\mathcal{Q}) \subseteq Supp(\mathcal{P})$. Then:*

$$R_\alpha(\mathcal{Q},\mathcal{P}) \;\leq\; \sup_{x\in Supp(\mathcal{Q})} \prod_{i=1}^{m} R_\alpha\left(\mathcal{Q}_{i|x_{<i}},\, \mathcal{P}_{i|x_{<i}}\right). \tag{29}$$

Proof. For a prefix $x_{\leq j}$, write:

$$M_i(x_{<i}) = \sum_\xi \mathcal{Q}_{i|x_{<i}}(\xi)^\alpha\, \mathcal{P}_{i|x_{<i}}(\xi)^{1-\alpha} = R_\alpha(\mathcal{Q}_{i|x_{<i}},\mathcal{P}_{i|x_{<i}})^{\alpha-1} \tag{30}$$

$$w(x_{\leq j}) = \prod_{i\leq j} \mathcal{Q}_{i|x_{<i}}(x_i)^\alpha\, \mathcal{P}_{i|x_{<i}}(x_i)^{1-\alpha} \tag{31}$$

$$f(x_{\leq j}) = \prod_{i\leq j} M_i(x_{<i}) \tag{32}$$

We claim that $\sum_{x_{\le j}} w(x_{\le j})/f(x_{\le j}) = 1$ for every $j \le m$, by induction on j. For $j = 1$ this is $M_1/M_1 = 1$. For the inductive step:

$$\sum_{x_{\le j}} \frac{w(x_{\le j})}{f(x_{\le j})} = \sum_{x_{\le j-1}} \frac{w(x_{\le j-1})}{f(x_{\le j-1})} \cdot \frac{1}{M_j(x_{<j})} \sum_{x_j} \mathcal{Q}_{j|x_{<j}}(x_j)^{\alpha}\, \mathcal{P}_{j|x_{<j}}(x_j)^{1-\alpha} = \sum_{x_{\le j-1}} \frac{w(x_{\le j-1})}{f(x_{\le j-1})},$$

since the inner sum is exactly $M_j(x_{<j})$. Therefore

$$R_\alpha(\mathcal{Q},\mathcal{P})^{\alpha-1} = \sum_x w(x) = \sum_x \frac{w(x)}{f(x)} \cdot f(x) \;\le\; \Big(\sup_x f(x)\Big) \cdot \sum_x \frac{w(x)}{f(x)} = \sup_x \prod_i M_i(x_{<i}),$$

and taking $(\alpha-1)$-th roots concludes the proof. □

Theorem 2 (Working with the RMSE). *Using the notations of Theorem 1. Assume there are global constants Δ_σ and $\Delta_{\mathbf{t}}$ such that:*

1. *For any (f,g,F,G) output by NTRUGen, the leaves $(\sigma_i)_{i\in[2\,n]}$ and $(\overline{\sigma}_i)_{i\in[2\,n]}$ of the trees computed from (f,g,F,G) in exact and fixed-point arithmetic, respectively, have the quadratic mean of their relative error bounded as:*

$$\sqrt{\frac{1}{2n}\sum_{i\in[2n]} \left|\frac{\overline{\sigma}_i}{\sigma_i} - 1\right|^2} \le \Delta_\sigma. \tag{33}$$

2. *For $(\mathsf{sk}, \overline{\mathsf{sk}})$ as in the previous item, for any messages $(\mathsf{m}_j)_{j\in[Q'_s]}$ and any history, the centers $(t_{i,j})_{i,j}$ and $(\bar{t}_{i,j})_{i,j}$ computed by Sign and $\overline{\text{Sign}}$ from the same history have the quadratic mean of their absolute error bounded as:*

$$\sqrt{\frac{1}{2nQ_s}\sum_{j\in[Q'_s]}\sum_{i\in[2n]} |\bar{t}_{i,j} - t_{i,j}|^2} \le \Delta_{\mathbf{t}}. \tag{34}$$

Furthermore, let $\alpha = 2\lambda - 1$, let δ_∞ be such that $\max_i |\overline{\sigma}_i/\sigma_i - 1| \le \delta_\infty$ for any (f,g,F,G) output by NTRUGen, and let $\epsilon < 1/4$ be a smoothing loss such that $\sigma_{\min} \ge \max(1, \sqrt{1+3\,\alpha\,\delta_\infty}\cdot\eta_\epsilon(\mathbb{Z}))$. Assume that $\alpha\,\delta_\infty \le 1/8$ and that:

$$\exp\left(3\,\alpha\,\Delta_\sigma^2\,(1+O(\epsilon+\alpha\,\delta_\infty))\right) \le 1 + \frac{1}{4\,N}, \tag{35}$$

$$\exp\left(\frac{\alpha\,\Delta_{\mathbf{t}}^2}{2}\,(1+O(\epsilon+\delta_\infty+1/\alpha))\right) \le 1 + \frac{1}{4\,N}. \tag{36}$$

If Falcon has a bit-security λ in the UF-CMA game (unforgeability against chosen-message attacks), then $\overline{\text{Falcon}}$ has a bit-security at least $\lambda - 2$ in the same game.

Proof. As in Theorem 1, we view Falcon as making $N = 2\,n\,Q_s(1+c)$ queries to a perfect Gaussian sampler $D_{\mathbb{Z},\sigma_i,t_{i,j}}$, and we go through the same intermediate

$\overline{\overline{\text{FALCON}}}$, where only the leaves $(\sigma_i)_i$ are swapped for $(\overline{\sigma}_i)_i$. Let $\mathcal{P}, \mathcal{Q}, \mathcal{R}$ denote the distributions of the view of the adversary in FALCON, $\overline{\overline{\text{FALCON}}}$ and $\overline{\text{FALCON}}$, respectively. By Lemma 15, it suffices to show that $R_\alpha(\mathcal{Q}, \mathcal{P})$ and $R_\alpha(\mathcal{R}, \mathcal{Q})$ are both at most $\left(1 + \frac{1}{4N}\right)^N$.

First hop. Lemma 5 bounds $R_\alpha(D_{\mathbb{Z},\overline{\sigma}_i,t_{i,j}}, D_{\mathbb{Z},\sigma_i,t_{i,j}})$ uniformly in the center c, hence uniformly in the (history-dependent) $t_{i,j}$; iterating [HPRR20, Proposition 4] therefore gives:

$$R_\alpha(\mathcal{Q}, \mathcal{P}) \leq \prod_{i\in[2n]} \prod_{j\in[Q'_s]} \sup_{c\in\mathbb{R}} R_\alpha(D_{\mathbb{Z},\overline{\sigma}_i,c}, D_{\mathbb{Z},\sigma_i,c}).$$

We bound each factor by Lemma 5, exactly as in Theorem 1 but keeping the individual errors. Let

$$\delta_i := |\sigma_i^2/\overline{\sigma}_i^2 - 1| \tag{37}$$

$$\Rightarrow \delta_i \leq 2\,|\overline{\sigma}_i/\sigma_i - 1|\,(1 + O(\delta_\infty)) \leq 3\,\delta_\infty \tag{38}$$

$$\Rightarrow \sigma_i \geq \sigma_{\min} \geq \sqrt{1 + \alpha\,\delta_i} \cdot \eta_\epsilon(\mathbb{Z}) \tag{39}$$

Thus Lemma 5 applies, and each factor is at most $\exp\left(\frac{3\,\alpha\,\delta_i^2}{4}\,(1 + O(\epsilon + \alpha\,\delta_\infty))\right)$. Multiplying, then applying Eq. (33) and Eq. (35):

$$\begin{aligned} R_\alpha(\mathcal{Q}, \mathcal{P}) &\leq \exp\left(\frac{3\,\alpha}{4}\,(1 + O(\epsilon + \alpha\,\delta_\infty))\,Q'_s \sum_{i\in[2n]} \delta_i^2\right) \\ &\leq \exp\left(N \cdot 3\,\alpha\,\Delta_\sigma^2\,(1 + O(\epsilon + \alpha\,\delta_\infty))\right) \\ &\leq \left(1 + \frac{1}{4N}\right)^N \end{aligned}$$

Second hop. Swapping from $\overline{\overline{\text{FALCON}}}$ to $\overline{\text{FALCON}}$ replaces every $t_{i,j}$ by $\bar{t}_{i,j}$, and these centers vary with the execution, which is why we need the RMSE. Lemma 16 bounds $R_\alpha(\mathcal{R}, \mathcal{Q})$ by the supremum, over the executions (a history of sampler outputs and adversarial messages), of the product of the per-call divergences realized along that execution. We fix such an execution. As in Theorem 1, Lemma 4 applied at $\sigma = \overline{\sigma}_i$ bounds the call (i, j) by $\exp\left(\frac{\alpha\,|\bar{t}_{i,j} - t_{i,j}|^2}{2}\,(1 + O(\epsilon + \delta_\infty + 1/\alpha))\right)$, so that:

$$\begin{aligned} \prod_{i\in[2n]} \prod_{j\in[Q'_s]} R_\alpha(D_{\mathbb{Z},\overline{\sigma}_i,\bar{t}_{i,j}}, D_{\mathbb{Z},\overline{\sigma}_i,t_{i,j}}) &\leq \exp\left(\frac{\alpha}{2}\,(1 + O(\epsilon + \delta_\infty + 1/\alpha)) \sum_{i,j} |\bar{t}_{i,j} - t_{i,j}|^2\right) \\ &\leq \exp\left(N \cdot \frac{\alpha\,\Delta_{\mathbf{t}}^2}{2}\,(1 + O(\epsilon + \delta_\infty + 1/\alpha))\right), \end{aligned}$$

where the last inequality is Eq. (34) applied to this very execution – the "*for any history*" quantifier of Item 2 is exactly the supremum required by Lemma 16. Taking the supremum and applying Eq. (36), we get $R_\alpha(\mathcal{R}, \mathcal{Q}) \leq \left(1 + \frac{1}{4N}\right)^N$, which concludes the proof. □

We instantiate Theorem 2 with experimental numbers computed in Section 7.5. This gives us 2^{60} queries. We note that this is higher than expected using the methodology of [Pre17]. It would be interesting to apply our proof technique to the experimental errors found in [HvVS+26].

6 Type 1 Implementation in C and Benchmarking

6.1 Implementation overview

Our implementation is derived from the public FALCON reference code submitted to NIST, and is primarily intended to validate the algorithmic and analytic changes introduced in this work. For this reason, the codebase currently contains three variants sharing a common structure: (i) the original *native floating-point*; (ii) an *integer-only emulation of floating-point arithmetic* (also from the original implementation); and (iii) our new *fixed-point* implementation incorporating the optimizations described in this paper. This allows us to perform relevant comparisons (including KATs, differential testing, and benchmarking) while keeping the surrounding control flow identical. Each variant can be chosen at compilation time by using the appropriate compilation flag.

A key design goal is modularity: our core routines (FFT-related code, tree construction, and sampling) are written so that the same high-level algorithm can be instantiated either with native doubles or with fixed-point arithmetic.

`https://github.com/fixed-point-fndsa/fixed-point-c`

6.2 Implemented optimizations

Our implementation includes several optimizations, some of which are *generic* and beneficial even when using native floating-point ($\bullet$), and some of which are specifically aimed at *eliminating floating-point arithmetic* ($\star$).

- $\bullet$ **Symplectree.** We exploit the symplectic structure of NTRU bases to store and generate only one half of the Falcon tree, reconstructing the other half on-the-fly. Working only with the left tree saves us from doing one LDL* step, which improves the precision. This reduces the tree memory footprint by (almost) a factor two and also cuts the tree-generation cost accordingly (Section 4.3).
- $\bullet$ **Using $\mathbf{t}_{\text{frac}}$ in ffSampling.** We implement the decomposition $\mathbf{t} = \mathbf{t}_{\text{int}} + \mathbf{t}_{\text{frac}}$ and feed only $\mathbf{t}_{\text{frac}}$ to ffSampling. This reduces the amplitude of sampler inputs (hence precision pressure) without changing the distribution of valid signatures.
- $\star$ **Float-free signing path.** In our fixed-point variant, no floating-point arithmetic is used in key generation, signing, or verification: all real/complex operations involved in FFT-domain computations and sampling are performed with fixed-point integers.

- ⋆ **Fixed-point key generation components.** For the NTRU solving portion of key generation, we reuse and adapt Pornin's fixed-point NTRUGEN implementation [Por25] as a module, and we integrate our additional bound checks (e.g., on α_{hybrid} and FFT norms) into this pipeline. These checks provide unconditional bounds on intermediate magnitudes and greatly simplify fixed-point range management.

6.3 KAT preservation and current engineering pitfalls

KAT-preserving strategy. We emphasize that our fixed-point implementation is designed to stay as close as possible to the original FALCON KATs *without modifying the deterministic randomness stream.* To that end, we keep the same control flow and random draws as the reference implementation whenever feasible, and we avoid changes that would desynchronize the RNG state even if they produce valid signatures.

In particular, we currently do *not* deploy the square-root removal technique inside SamplerZ, because this change modifies how randomness is consumed (even if the resulting signatures remain valid) and the speed tradeoff is unclear. Instead, we keep the original structure and implement a fast constant-time iterative square-root routine.

In addition, since our NTRUGen includes additional bound checks, it rejects a larger fraction of candidate keys than the baseline reference. This does not affect security or correctness, but it changes the key generation flow. Therefore, for strict KAT verification we currently rely on the reference key generation, while we benchmark our modified key generation separately.

Fixed-point formats. At this stage, for simplicity and stress-testing, the implementation is Type 1: it uses a uniform `64.64` representation for fixed-point values. This is intentionally conservative and comfortably satisfies the bounds derived in the paper, but it is not optimal: it incurs a performance and memory overhead compared to the multi-format design implied by our precision analysis. Additional engineering work is required to implement the heterogeneous fixed-point formats described by the analysis, and to systematically apply format-specialization across all subroutines. A Python Type 2 implementation is detailed in Section 7.

Constant time policy. The current implementation has been tested for constant-time using timecop[8]. While this tool is neither sound, nor complete, it allows for a quick overview of the current state of the implementations, and is easy to learn for developers [JFD+22,FDJ+24]. Running the analysis currently returns some false positives (manually confirmed as benign), but a more extensive and sound testing may be suitable before real-world deployment. This is out of scope of this contribution, and is considered future work.

[8] https://www.post-apocalyptic-crypto.org/timecop/

6.4 Benchmark

We benchmark on a commodity laptop equipped with an `Intel(R) Core(TM) Ultra 7 155H`. To reduce measurement noise, we disable TurboBoost, fix the CPU frequency, and pin the benchmark process to a single core. In this configuration, the measured core frequency is stable at 4.8 GHz. For all tests, the code was compiled with `gcc 13.3.0`, optimization level `-O3`, on Ubuntu 24.04.

We compare (when applicable) the three variants: native floating-point, emulated floating-point, and our fixed-point implementation. We also report the results for the native version including the generic optimizations.

Table 3 reports the current end-to-end performance of our implementation. Since the current fixed-point signing path uses a conservative `64.64` format throughout, these numbers are pessimistic with respect to the final design.

Table 3: Benchmark results for Falcon-512. Speedups: $\Delta_{\text{native}} = \mathsf{Native}_{\text{ref}}/\mathsf{Native}_{\text{new}}$ (generic optimizations), $\Delta_{\text{fp}} = \mathsf{Emu}_{\text{ref}}/\mathsf{Fixed}_{\text{new}}$ (no-float variants), $\Delta_{\text{fp/fx}} = \mathsf{Native}_{\text{ref}}/\mathsf{Fixed}_{\text{new}}$ (native to fixed point). Values > 1 indicate faster execution.

	Native			No-float			Speedup
Operation	ref	**new**	Δ_{native}	emu	**fixed**	Δ_{fp}	$\Delta_{\text{fp/fx}}$
Keygen (ms)	4.0	**2.7**	1.48×	8.6	**2.7**	3.18×	1.48×
Expand key (µs)	45.5	**39.8**	1.14×	1292.0	**141.1**	9.16×	0.32×
Sign (µs)	101.7	**106.7**	0.95×	1412.4	**198.2**	7.13×	0.51×
Verify (µs)	15.1	**14.7**	1.02×	15.0	**14.8**	1.01×	1.02×

The generic optimizations primarily impact key generation and tree handling, yielding a 1.48× speedup for Falcon-512 key generation in the native build. More importantly, replacing software-emulated floating-point arithmetic with fixed-point yields large gains: key expansion and signing improve by about 7–9× compared to the emulated-float reference. Verification is essentially unchanged across variants, as expected. Compared to the native floating-point build, fixed point incurs an overhead, since floating-point operations are executed in hardware with a single (or a few) instructions, whereas fixed-point arithmetic must be implemented in software. Crucially, however, fixed point is substantially faster than floating-point emulation, which was until now the standard approach for running Falcon on platforms without native floating-point support.

Regarding memory usage, the only changes that affect memory footprint are:

– **Symplectree.** Storing only half of the Falcon tree saves almost half of the expanded key storage. The beneficial impact of this optimization has already been evaluated in [SZZ+22]. Additionally, avoiding the full LDL computation at the root of the tree saves on both instructions and temporary storage. We gain

an additional $2 \times \mathrm{FPR}_{\mathrm{len}} \times n$ bytes, with $\mathrm{FPR}_{\mathrm{len}}$ the byte-length of an element representing a real.

– **Fixed-point representation.** The uniform `64.64` representation increases memory usage, since values that fit in one word in the native implementation require two words in our conservative format. Once the heterogeneous fixed-point formats are implemented, values will match the storage footprint of the native variant. See Section 7.5 for the precision of a 64-bit implementation.

Overall, the final implementation retains the memory savings induced by Symplectree and improves over the current fixed-point timings by moving from the conservative `64.64` format to the specialized formats prescribed by our analysis.

7 Type 2 Implementation in Python and Precision Benchmarking

We present a 64-bit Type 2 implementation in typed Python, see Section 3.4 for the terminology. All fixed-point values fit in 64-bit machine words, paving the way for memory-efficient fixed-point implementations. Our implementation is derived from Prest's falcon.py (https://github.com/tprest/falcon.py), which we adapted by porting all floating-point functions to the fixed-point format. Some of the code of this proof-of-concept was generated using Claude Code, with models Opus 4.7–4.8 and Fable 5. We make the code available here:

https://github.com/fixed-point-fndsa/fxp-falcon-py

Throughout this section, when we say that a property P is *enforced*, it means that either: (i) P is always true, or (ii) the code raises an `AssertionError` if P is not satisfied.

7.1 Types

Our types for fixed-point representation of real and complex numbers are `FxR` and `FxC`, respectively. `FxR(x, m, p)` is the exact representation of $v = \mathtt{x} * 2^{\mathtt{m}-\mathtt{p}}$, with the conditions $\mathtt{x} \in \mathbb{Z}$, $|\mathtt{x}| < 2^{\mathtt{p}}$ and $\mathtt{m} \leq \mathtt{p}$ enforced. We prefer to reason with $\mathtt{m} = k-1-f$ rather than f, since `m` is independent of the word size k and only depends on the characteristics of the (idealized) algorithm being implemented. A global $\mathtt{p} = k-1$ is enforced throughout the implementation.

```
@dataclass(frozen=True)
class FxR:
    x: int  # Integer repr.
    m: int  # m = p - f
    p: int  # p+1 = wordsize k
```

```
@dataclass(frozen=True)
class FxC:
    re: FxR
    im: FxR
```

Retagging. The methods `retag_fxr` and `retag_fxc` allow changing the value `m` of a `FxR` and `FxC`, respectively, so that the new `FxR`/`FxC` encodes the same value, possibly up to minor imprecisions.

m-budgets. Given a real value a, we use the notation $\mathtt{m}(a)$ as shorthand for a value `m` such that $|a| < 2^{\mathtt{m}}$. This means a admits the lossy representation `FxR(x, m, p)`, with $\mathtt{x} = \lfloor a \cdot 2^{\mathtt{p} - \mathtt{m}} \rceil$. The notation extends seamlessly to complex numbers. We refer to $\mathtt{m}(a)$ as the m-budget of a. m-budgets are instrumental in our implementation, as they need to be set as tight as possible for maximal precision.

7.2 Basic arithmetic operations

We now discuss the main arithmetic operations: addition, subtraction, multiplication, inversion and square root. The first three are implemented via operator overload: `a + b`, `a - b` and `a * b`.

1. **Addition and subtraction.** Addition of two `FxR` values `a` and `b` enforces `a.m == b.m`, and sets `c = FxR(x=a.x + b.x, m=a.m, p=p)`. Subtraction is implemented in a similar manner. Design choice:
   - `m` is left unchanged. This allows tighter bounds, but it is left to the implementer to ensure that there is no overflow.
2. **Multiplication.** Multiplication of two `FxR` values `a` and `b` sets `c = FxR(x=x, m=a.m + b.m, p=p)`, where `x` contains the `p` most significant bits of `x0 = a.x * b.x`. Design choices:
   - Setting `m = a.m + b.m` makes this a Type 2 implementation. In addition, the method `FxC.mul_to` allows to select the output `m`.
   - "Rounding" `x0` is done via round-half-to-even, aka the Banker's rounding.
3. **Inversion and square root.** Inversion and inverse square root are performed via Newton-Raphson, by the functions `nr_reciprocal` and `rsqrt`.

7.3 Falcon algorithms

1. **FFT.** This operation is implemented by `fft_fxp(f, certified=False)`. Consider a polynomial f of degree n, and $\hat{f} = \mathsf{FFT}(f)$. There are two modes:
   - `certified=False`: no assumption is made on the norm of $\hat{f}$. We have the trivial bound $\|\hat{f}\|_\infty \leq n \cdot \|f\|_\infty$. Therefore we set $\mathtt{m}(\hat{f}) = \mathtt{m}(f) + \log_2 n$.
   - `certified=True`: a bound on $\hat{f}$ is certified ahead of time. For example, from construction of NTRUGen we know that $\|\hat{f}\|_\infty, \|\hat{g}\|_\infty \leq \gamma_{f,g} = 255$, so that we may set $\mathtt{m}(\hat{f}) = \mathtt{m}(\hat{g}) = 8$.[9] This is much better than the bound $\mathtt{m}(\hat{f}) = \mathtt{m}(\hat{g}) = 5 + 9 = 14$ one would obtain in the non-certified case.

[9] One may object that an upper bound on the output $\mathsf{FFT}(f)$ does not guarantee an upper bound for intermediate variables. However, for the FFT, this is indeed the case. Indeed, the intermediate variables of $\mathsf{FFT}(f)$ are the same as for $\mathsf{invFFT}(\hat{f})$. Since invFFT does not increase the norm, bounding $\hat{f}$ also bounds the intermediate values of $\mathsf{invFFT}(\hat{f})$ and therefore of $\mathsf{FFT}(f)$.

In both cases, f is retagged to the m-budget $\mathtt{m} = \mathtt{m}(\hat{f})$ of the output $\hat{f}$, then the entire algorithm is done at an m-budget $\mathtt{m}$.

2. **invFFT.** This is implemented by `ifft_fxp`. This is much simpler to study than FFT: since invFFT does not increase the norm, we can use the input m-budget during the entire algorithm, and no retagging is required.
3. ffLDL***.** We recall that this is a recursive algorithm. We distinguish between the top-level call (root) and the inner recursive calls (inner). They are implemented by `ffldl_fft_fxp_ntru_root` and `ffldl_fft_fxp`, respectively.
   Out of the four entries of $\hat{\mathbf{G}}$, we only need to store two: $\hat{G}_{00}$ and $\hat{G}_{10}$. The other two are implicit: $\hat{G}_{01} = \hat{G}_{10}^\star$ by self-adjunction, at the root level $\hat{D}_{11} = q^2/\hat{G}_{00}$ by symplecticity so we do not need $\hat{G}_{11}$, and at the inner levels $\hat{G}_{11} = \hat{G}_{00}$. Note the following tricks:
   (a) $q^2/\hat{G}_{00}$ is computed as $q \cdot (q \cdot \hat{G}_{00}^{-1})$ so the constant q stays at $m = 14$ instead of 28 for q^2;
   (b) $\hat{D}_{jj}$ is renormalized to $m = 18$ between levels.
4. **Sign.** This is implemented by `sign`. There are two modes: the standard one (std) and the one with the tweak described in Section 5.1.
   In the standard variant, it holds that $\mathbf{s} = (\mathbf{t} - \mathbf{z}) \cdot \mathbf{B} = (c, 0) - \mathbf{z} \cdot \mathbf{B}$. This allows computing $\mathbf{s}$ using only pure integer arithmetic. Note that:
   (a) this is exact mod $\pm q$, since $\|\mathbf{s}\|_\infty \leq \tau\sigma < q/2$;
   (b) the tweak variant can use the same formula with $\mathbf{z}' = \mathbf{z} + \mathbf{t}_{\text{int}}$, though the execution is more complicated.
5. **ffSampling.** This is implemented by `ffsampling_fxp`. Our implementation makes it so that all intermediate values run at the same tag $m =$ `m_sign` (21 in std, 18 in tweak); so internally, `ffsampling_fxp` is effectively Type 1.
6. **SamplerZ.** This is implemented by `samplerz_fxp`. Note that in the Falcon specification, the subroutine ApproxExp is already in Type 1 fixed-point (`0.64`).

7.4 m-budgets

Since this is a Type 2 implementation, each variable has its own m-budget. After several optimizations, we managed to upper-bound all m-budgets by **21**. This indicates that a Type 1 implementation with `22.64` is possible with minimal adaptations, or `32.64` for a more natural alignment with 32-bit architectures. The m-budgets are given in Table 4.

7.5 Experiments

In this section, we provide experiments to evaluate the precision errors in our implementation. We implement Falcon-512 in 256-bit precision arithmetic (`mpmath`), which serves as ground truth. We then compare and plot the difference between this implementation and our Type 2 implementation, which gives us experimental errors. This is similar to the work of [HvVS+26].

Table 4: m-budgets for FALCON-512

a	$\mathtt{m}(a)$	Explanation
Key expansion (ffLDL* and children functions)		
$f, g, \hat{f}, \hat{g}$	8	By construction, $\|\hat{f}\|_\infty, \|\hat{g}\|_\infty \leq \gamma_{f,g} = 255$. We align f, g with $\hat{f}, \hat{g}$ for simplicity.
$F, G, \hat{F}, \hat{G}$	12	By construction, $\|\hat{F}\|_\infty, \|\hat{G}\|_\infty \leq \gamma_{F,G} = 3500$. We align F, G with $\hat{F}, \hat{G}$ for simplicity.
$\hat{G}_{00}$ (root)	18	$\|\hat{G}_{00}\|_\infty \leq \|\hat{f}\|^2 + \|\hat{g}\|^2 < 2 \cdot \gamma_{f,g}^2 < 2^{17}$. We set $\mathtt{m}(\hat{G}_{00}) = 18$ to align with the $\hat{D}_{jj}$'s downstream.
$\hat{G}_{10}$ (root)	21	$\hat{G}_{10} = \hat{f}^\star \cdot \hat{F} + \hat{g}^\star \cdot \hat{G}$, so $\|\hat{G}_{10}\|_\infty \leq 2 \cdot \gamma_{f,g} \cdot \gamma_{F,G}$.
$\hat{L}_{10}$ (root)	5	By construction, $\|\hat{L}_{10}\|_\infty \leq \gamma_{\text{root}} = 24 < 2^5$.
$\hat{D}_{jj}$	18	From Lemma 8, $\|\hat{D}_{jj}\| \leq q \cdot \gamma_{\text{hybrid}}^2$.
$\hat{G}_{ij}$ (inner)	18	$\hat{G}_{00}, \hat{G}_{01} = \mathsf{splitfft}(\hat{D}_{jj})$ for some $\hat{D}_{jj}$, and $\hat{G}_{10}, \hat{G}_{11} = \hat{G}_{01}^\star, \hat{G}_{00}$. splitfft does not increase the norm.
$\hat{L}_{10}$ (inner)	0	From Lemma 8, $\|\hat{L}_{10}\|_\infty < 1$.
Signing		
c (std)	14	$\|c\|_\infty < q$
c/q (std)	0	$\|c/q\|_\infty < 1$
$\hat{\mathbf{t}}, \hat{\mathbf{z}}$ (std)	21	Lemma 12
$\hat{\mathbf{t}}, \hat{\mathbf{z}}$ (tweak)	18	Lemma 12
SamplerZ		
dss, ccs	0	Leaf constants: $\mathtt{dss} = 1/(2\sigma_i^2) < 1$, and $\mathtt{ccs} = \sigma_{\min}/\sigma_i < 1$. Precomputed at key expansion.
$r,\ z - r$	5	From Lemma 13, $\|z\| \leq 19$ and $\|r\| \leq 1$, so $\|z - r\| \leq 20 < 2^5$. r is aligned with z since SamplerZ adds them up.
x	10	$\mathtt{m}(x) = 2 \times 5$ follows from the multiplication rule. We could retag to a smaller $\mathtt{m}(x)$, since $x < 62$ but this would cost additional retags and provides no gain.

We can then plug our experimental errors into Theorem 2, whose hypotheses are stated in terms of the root mean square errors (RMSE). The RMSE of the centers t_0' at the base level of ffSampling is lower than $2^{-39.5}$, see Fig. 7. The relative RMSE on the σ_i, measured separately, is about 2^{-60}, with a large margin; Fig. 8 shows the precision of the key-expansion chain feeding them, per level of the FALCON tree.

8 Recommendations for FN-DSA

In this section, we take a look back at tweaks that have been proposed in this work and others. For each tweak, we express whether we recommend them for a hypothetic fixed-point version of FN-DSA. This is summarized in Table 5. In particular, a Type 1 `32.64` version of FN-DSA can be easily specified and implemented if the tweaks marked **Yes** in Table 5 are added to the FN-DSA specification.

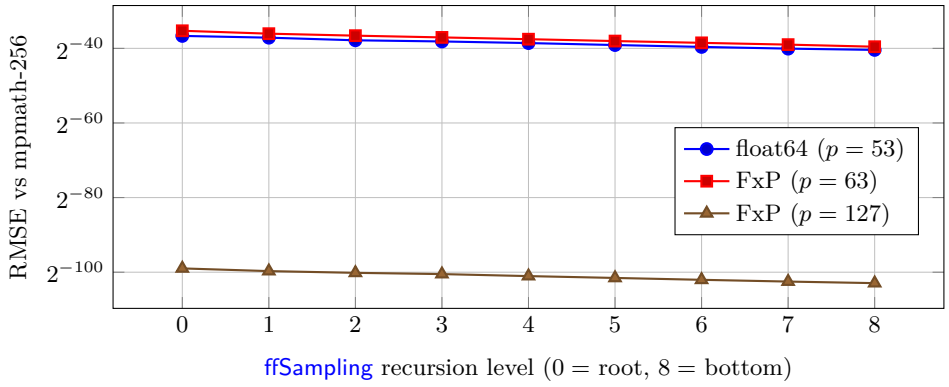

Fig. 7: Root mean square error (RMSE) of the centers t'_0 of ffSampling per level of recursion, for Falcon-512. Reference is `mpmath`-256.

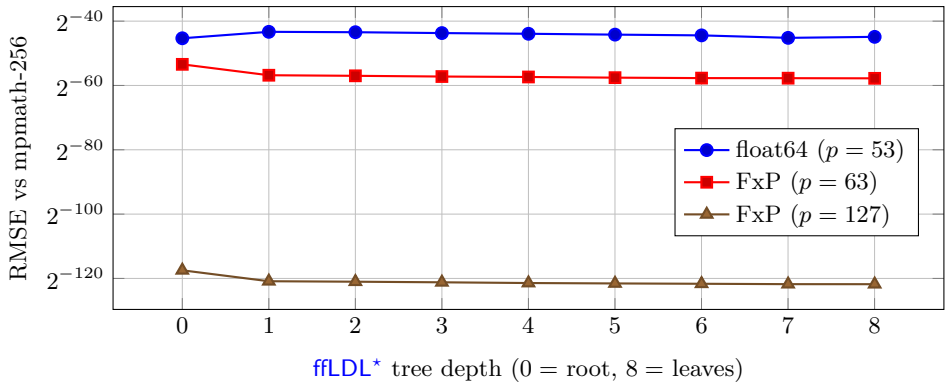

Fig. 8: RMSE of the Gram–Schmidt polynomials L_{10} computed by ffLDL* during key expansion, per level of the FALCON tree, for Falcon-512. Reference is `mpmath`-256.

8.1 Implement the LTYZ25 countermeasures

In Section 5.1, we described how the countermeasures from [LTYZ25] allow eliminating an entire class of divergences. This class of divergences is *fundamental* to how FALCON is currently specified: any two implementations, even if mathematically equivalent on the surface, run the risk of triggering it every ~ 8000 signatures. The divergences we observed in Section 5.1 are merely a symptom of this situation.

We *recommend* adopting the LTYZ25 countermeasures in the FN-DSA specification. They require a moderate rewriting of SamplerZ, BaseSampler and ApproxExp, however they are the only known robust way to guarantee that correct implementations will pass KAT vectors.[10]

[10] Alternatively, we can generate KAT vectors that are guaranteed to *not* trigger LTYZ25 divergences. However this countermeasure sits on the side of KAT vectors, not of the implementation.

Table 5: Possible tweaks for FN-DSA

Tweak	Section	Effort to implement?	Recommended for FN-DSA?
Other works			
LTYZ25 tweak	Section 8.1	Moderate	**Yes**
Bound $\|a\|_\infty$ for $a \in \{f, g, F, G\}$	Section 8.2	Straightforward	**Yes**
This work			
Enforce $\|\hat{f}\|_\infty, \|\hat{g}\|_\infty \leq \gamma_{f,g}$	Section 8.3	Straightforward	**Suggested**
Enforce $\|\hat{F}\|_\infty, \|\hat{G}\|_\infty \leq \gamma_{F,G}$	Section 8.4	Straightforward	**Yes**
Enforce $\alpha_{\text{hybrid}} \leq \gamma_{\text{hybrid}}$	Section 8.5	Straightforward	**Yes**
Enforce $\|\hat{k}\|_\infty \leq \gamma_{\text{root}}$	Section 8.6	Straightforward	**Suggested**
Signing tweak in Section 5.1	Section 8.7	Moderate	No

8.2 Bound $\|F\|_\infty, \|G\|_\infty, \|f\|_\infty, \|g\|_\infty$

In the FALCON specification, no bound is specified on $\|a\|_\infty$ for $a \in \{f, g, F, G\}$. However, Pornin's C implementation ensures $\|f\|_\infty, \|g\|_\infty \leq 24$ by construction, and enforces $\|F\|_\infty, \|G\|_\infty \leq 127$ via a check.

We *recommend* adopting these checks in the FN-DSA specification: they are straightforward to implement, facilitate a fixed-point implementation and make it easier to store f, g, F, G.

8.3 Enforce $\|\hat{f}\|_\infty, \|\hat{g}\|_\infty \leq \gamma_{f,g} = 255$

If this check is absent, then we can replace it by one of these slightly weaker bounds:

1. The check in Section 8.2 combined with the fact $\|\hat{f}\|_\infty \leq n \|f\|_\infty$;
2. The check in Section 8.5 combined with the fact $|f(\zeta)|^2 + |g(\zeta)|^2 \leq q\,(\alpha_{\text{hybrid}})^2$, which gives $\|\hat{f}\|_\infty, \|\hat{g}\|_\infty \leq \sqrt{q} \cdot \gamma_{\text{hybrid}}$.

We note that this check intervenes in the same places as the one in Section 8.4. However $\gamma_{f,g} \ll \gamma_{F,G}$, so the former quantity is never a bottleneck in practice.

We *suggest* adopting this check in the FN-DSA specification: it is not essential and a weaker version can be derived from other checks, but it is straightforward to implement.

8.4 Enforce $\|\hat{F}\|_\infty, \|\hat{G}\|_\infty \leq \gamma_{F,G} = 3500$

If this check is absent, then we can only rely on the weaker bounds $\|\hat{F}\|_\infty \leq n \|F\|_\infty$ and $\|\hat{G}\|_\infty \leq n \|G\|_\infty$; combined with the check of Section 8.2, this gives $\|\hat{F}\|_\infty, \|\hat{G}\|_\infty \leq 127\, n \approx 2^{16}$, about 4 bits looser than $\gamma_{F,G}$. The main impact of $\gamma_{F,G}$ is in the boundedness analysis of ffSampling (Lemma 12).

We *recommend* adopting this check in the FN-DSA specification: it is straightforward to implement and facilitate a fixed-point implementation of signing.

8.5 Enforce $\alpha_{\mathbf{hybrid}} \leq \gamma_{\mathbf{hybrid}} = 4$

This check mainly impacts the following quantities:

1. *Key expansion.* Upper- *and* lower-bounding the values $\hat{d}_{jj}$'s in ffLDL*, see (15); In particular, tighter bounds provides stronger correctness, precision and efficiency guarantees with performing inversion in LDL*.
2. *Signing.* The boundedness analysis of ffSampling (Lemma 12);

If this check is absent, we can fall back on the bound $\alpha_{\text{hybrid}} \leq \sqrt{n} \cdot \alpha_{\text{GPV}}$ (8), which is looser by a bit less than 3 bits.

We *recommend* adopting this check in the FN-DSA specification: it is straightforward to implement, is essential for secure and efficient inversion during key expansion (LDL*) and enables a tight boundedness analysis.

8.6 Enforce $\|\hat{k}\|_\infty \leq \gamma_{\mathbf{root}} = 24$

We recall that $k = \frac{F\,f^\star + G\,g^\star}{f\,f^\star + g\,g^\star} = L_{10}^{[1]}$. Checking $\|\hat{k}\|_\infty$ against γ_{root} pins the value of $L_{10}^{[1]}$. If removed, we can fall back on this formula, which is looser by $\sim$3 bits:

$$\left\| \frac{F\,f^\star + G\,g^\star}{f\,f^\star + g\,g^\star} \right\|_\infty \leq \frac{\max_\zeta \sqrt{|\hat{F}|^2 + |\hat{G}|^2}}{\min_\zeta \sqrt{|\hat{f}|^2 + |\hat{g}|^2}} \leq \frac{\sqrt{2} \cdot \gamma_{F,G} \cdot \gamma_{\text{hybrid}}}{\sqrt{q}} \approx 179. \tag{40}$$

γ_{root} also impacts the boundedness analysis of ffSampling (Lemma 12), however it is not the primary bottleneck; removing it has an impact of one bit (for std signing).

We *suggest* adopting this check in the FN-DSA specification: it is not essential, but it is straightforward to implement.

8.7 Signing tweak in Section 5.1

The tweak in Sign presented in Section 5.1 provides a tighter boundedness analysis for Sign and ffSampling, by about 3 bits.

We *do not suggest* adopting this check in the FN-DSA specification: it requires some small changes to Sign, and reconstruction is not as straightforward as in the standard case. Furthermore, it seems that a reasonable number of queries is already guaranteed without this tweak (see Section 7.5).

9 Conclusion

We present a fixed-point implementation of Falcon, designed to be constant-time, together with a complete boundedness analysis, a security analysis conditional on error bounds, and extensive benchmarking of these error bounds. This work addresses a key limitation of Falcon in the context of online signing. In particular, our implementation eliminates the reliance on floating-point arithmetic while preserving both security and efficiency. Moreover, it is well suited for platforms without a floating-point unit, as it outperforms the software-emulated floating-point implementation included in Pornin's reference code of Falcon.

The next step is to compute more formal bounds on the errors. One solution can be to use interval arithmetic to obtain such bounds or other techniques to prove concrete bounds on the errors. We also consider developing an alternative variant based on a dedicated 64-bit `int` representation to further reduce memory consumption depending on the previous security analysis. Another important direction for future work is the formal verification of the error bounds using a proof assistant such as `gappa` (https://gappa.gitlabpages.inria.fr/). Beyond its software applicability, our implementation provides a solid foundation for hardware realizations and for masked implementations designed to mitigate side-channel attacks.

Acknowledgments. The authors would like to express their gratitude to the anonymous reviewers of CRYPTO for their many insightful comments and remarks, which improved significantly the quality of this paper. We thank Amine Najahi for providing some figures, and Thomas Pornin for useful comments.

This work has been supported by the French Agence Nationale de la Recherche, under the project ANR-25-CE39-4214-01 RELATE, as well as the EU Horizon Europe research and innovation programme under Grant Agreement No 101225722 (FORTRESS).

AI disclosure. Post-submission to CRYPTO, AI tools (Claude Fable 5 and Opus 4.8-5) have been used, in interaction with the fourth author, to proofread this paper, rework proofs and generate the Python proof-of-concept. We take full responsibility for the content of this paper.

References

BLR+18. Shi Bai, Tancrède Lepoint, Adeline Roux-Langlois, Amin Sakzad, Damien Stehlé, and Ron Steinfeld. Improved security proofs in lattice-based cryptography: Using the Rényi divergence rather than the statistical distance. Journal of Cryptology, 31(2):610–640, April 2018.

CC24. Keng-Yu Chen and Jiun-Peng Chen. Masking floating-point number multiplication and addition of falcon first- and higher-order implementations and evaluations. IACR TCHES, 2024(2):276–303, 2024.

Csi63. Imre Csiszar. Eine informationstheoretische ungleichung und ihre anwendung auf den beweis der ergodizitat von markoffschen ketten. Magyar Tud. Akad. Mat. Kutato Int. Kozl., 8:85–108, 1963.

DGPY20. Léo Ducas, Steven Galbraith, Thomas Prest, and Yang Yu. Integral matrix gram root and lattice gaussian sampling without floats. In Anne Canteaut and Yuval Ishai, editors, EUROCRYPT 2020, Part II, volume 12106 of LNCS, pages 608–637. Springer, Cham, May 2020.

DLP14. Léo Ducas, Vadim Lyubashevsky, and Thomas Prest. Efficient identity-based encryption over NTRU lattices. In Palash Sarkar and Tetsu Iwata, editors, ASIACRYPT 2014, Part II, volume 8874 of LNCS, pages 22–41. Springer, Berlin, Heidelberg, December 2014.

DN12. Léo Ducas and Phong Q. Nguyen. Faster Gaussian lattice sampling using lazy floating-point arithmetic. In Xiaoyun Wang and Kazue Sako, editors, ASIACRYPT 2012, volume 7658 of LNCS, pages 415–432. Springer, Berlin, Heidelberg, December 2012.

DP16. Léo Ducas and Thomas Prest. Fast fourier orthogonalization. In Sergei A. Abramov, Eugene V. Zima, and Xiao-Shan Gao, editors, Proceedings of the ACM on International Symposium on Symbolic and Algebraic Computation, ISSAC 2016, Waterloo, ON, Canada, July 19-22, 2016, pages 191–198. ACM, 2016.

EFG+22. Thomas Espitau, Pierre-Alain Fouque, François Gérard, Mélissa Rossi, Akira Takahashi, Mehdi Tibouchi, Alexandre Wallet, and Yang Yu. Mitaka: A simpler, parallelizable, maskable variant of falcon. In Orr Dunkelman and Stefan Dziembowski, editors, EUROCRYPT 2022, Part III, volume 13277 of LNCS, pages 222–253. Springer, Cham, May / June 2022.

ENS+23. Thomas Espitau, Thi Thu Quyen Nguyen, Chao Sun, Mehdi Tibouchi, and Alexandre Wallet. Antrag: Annular NTRU trapdoor generation - making mitaka as secure as falcon. In Jian Guo and Ron Steinfeld, editors, ASIACRYPT 2023, Part VII, volume 14444 of LNCS, pages 3–36. Springer, Singapore, December 2023.

FDJ+24. Marcel Fourné, Daniel De Almeida Braga, Jan Jancar, Mohamed Sabt, Peter Schwabe, Gilles Barthe, Pierre-Alain Fouque, and Yasemin Acar. "These results must be false": A usability evaluation of constant-time analysis tools. In Davide Balzarotti and Wenyuan Xu, editors, USENIX Security 2024. USENIX Association, August 2024.

FGdG+26. Pierre-Alain Fouque, Phillip Gajland, Hubert de Groote, Jonas Janneck, and Eike Kiltz. A closer look at falcon. In Joan Daemen and Emmanuel Thomé, editors, Advances in Cryptology - EUROCRYPT 2026 - 45th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Rome, Italy, May 10-14, 2026, Proceedings, Part IV, volume 16544 of Lecture Notes in Computer Science, pages 3–31. Springer, 2026.

GPV08. Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In Richard E. Ladner and Cynthia Dwork, editors, 40th ACM STOC, pages 197–206. ACM Press, May 2008.

HPRR20. James Howe, Thomas Prest, Thomas Ricosset, and Mélissa Rossi. Isochronous gaussian sampling: From inception to implementation. In Jintai Ding and Jean-Pierre Tillich, editors, Post-Quantum Cryptography - 11th International Conference, PQCrypto 2020, pages 53–71. Springer, Cham, 2020.

HPS98. Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. NTRU: A ring-based public key cryptosystem. In Joe Buhler, editor, Algorithmic Number Theory, Third International Symposium, ANTS-III, Portland, Oregon, USA, June 21-25, 1998, Proceedings, volume 1423 of Lecture Notes in Computer Science, pages 267–288. Springer, 1998.

HvVS+26. Stef Halmans, Christine van Vredendaal, Tobias Schneider, Frank Custers, and Tim Güneysu. Twfalcon: Triple-word arithmetic for falcon: Giving falcon the precision to fly securely. IACR Transactions on Cryptographic Hardware and Embedded Systems, 2026(2):347–371, Apr. 2026.

JFD+22. Jan Jancar, Marcel Fourné, Daniel De Almeida Braga, Mohamed Sabt, Peter Schwabe, Gilles Barthe, Pierre-Alain Fouque, and Yasemin Acar. "They're not that hard to mitigate": What cryptographic library developers think about timing attacks. In 2022 IEEE Symposium on Security and Privacy, pages 632–649. IEEE Computer Society Press, May 2022.

Kle00. Philip N. Klein. Finding the closest lattice vector when it's unusually close. In David B. Shmoys, editor, 11th SODA, pages 937–941. ACM-SIAM, January 2000.

LSS14. Adeline Langlois, Damien Stehlé, and Ron Steinfeld. GGHLite: More efficient multilinear maps from ideal lattices. In Phong Q. Nguyen and Elisabeth Oswald, editors, EUROCRYPT 2014, volume 8441 of LNCS, pages 239–256. Springer, Berlin, Heidelberg, May 2014.

LTYZ25. Xiuhan Lin, Mehdi Tibouchi, Yang Yu, and Shiduo Zhang. Do not disturb a sleeping falcon - floating-point error sensitivity of the falcon sampler and its consequences. In Serge Fehr and Pierre-Alain Fouque, editors, EUROCRYPT 2025, Part II, volume 15602 of LNCS, pages 213–244. Springer, Cham, May 2025.

Lyu12. Vadim Lyubashevsky. Lattice signatures without trapdoors. In David Pointcheval and Thomas Johansson, editors, EUROCRYPT 2012, volume 7237 of LNCS, pages 738–755. Springer, Berlin, Heidelberg, April 2012.

MNR14. Matthieu Martel, Amine Najahi, and Guillaume Revy. Toward the synthesis of fixed-point code for matrix inversion based on cholesky decomposition. In Proceedings of the 2014 Conference on Design and Architectures for Signal and Image Processing, pages 1–8, 2014.

MR07. Daniele Micciancio and Oded Regev. Worst-case to average-case reductions based on gaussian measures. SIAM J. Comput., 2007.

PFH+20. Thomas Prest, Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. FALCON. Technical report, National Institute of Standards and Technology, 2020. available at https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions.

Por23. Thomas Pornin. Improved key pair generation for falcon, BAT and hawk. Cryptology ePrint Archive, Report 2023/290, 2023.

Por25. Thomas Pornin. Improved (again) key pair generation for falcon, BAT and hawk. Cryptology ePrint Archive, Paper 2025/1239, 2025.

PP19. Thomas Pornin and Thomas Prest. More efficient algorithms for the NTRU key generation using the field norm. In Dongdai Lin and Kazue Sako, editors, PKC 2019, Part II, volume 11443 of LNCS, pages 504–533. Springer, Cham, April 2019.

Pre15. Thomas Prest. Gaussian Sampling in Lattice-Based Cryptography. Theses, École Normale Supérieure, December 2015.

Pre17. Thomas Prest. Sharper bounds in lattice-based cryptography using the Rényi divergence. In Tsuyoshi Takagi and Thomas Peyrin, editors, ASIACRYPT 2017, Part I, volume 10624 of LNCS, pages 347–374. Springer, Cham, December 2017.

SS13. Damien Stehlé and Ron Steinfeld. Making NTRUEncrypt and NTRUSign as secure as standard worst-case problems over ideal lattices. Cryptology ePrint Archive, Report 2013/004, 2013.

SZZ+22. Shuo Sun, Yongbin Zhou, Rui Zhang, Yang Tao, Zehua Qiao, and Jingdian Ming. Fast fourier orthogonalization over NTRU lattices. In Cristina Alcaraz, Liqun Chen, Shujun Li, and Pierangela Samarati, editors, Information and Communications Security - 24th International Conference, ICICS 2022, Canterbury, UK, September 5-8, 2022, Proceedings, volume 13407 of Lecture Notes in Computer Science, pages 109–127. Springer, 2022.

vEH12. Tim van Erven and Peter Harremoës. Rényi divergence and kullback-leibler divergence. CoRR, abs/1206.2459, 2012.

Yat24. Randy Yates. Fixed-point arithmetic: An introduction, 2024. http://www.digitalsignallabs.com/downloads/fp.pdf.

A SamplerZ and its subroutines: BerExp and ApproxExp

In this section, we give the formal description of SamplerZ and its subroutines BerExp and ApproxExp, as specified in [PFH+20] and used in our implementation; Lemma 13 bounds their internal variables. At the end of this section, we also discuss a square-root-free variant of BerExp that we considered but do not deploy.

In BerExp, the final Bernoulli trial compares a uniform 64-bit integer against $z \approx 2^{64} \cdot ccs \cdot e^{-x}$ *lazily*: the comparison proceeds byte by byte, from the most significant byte down, and a fresh random byte is drawn only as long as all previous bytes were equal. This leaves the acceptance probability unchanged while consuming less randomness on average.

Algorithm 9 SamplerZ(μ, σ')

Require: Fixed-point values $\mu, \sigma' \in \mathbb{R}$ such that $\sigma' \in [\sigma_{\min}, \sigma_{\max}]$
Ensure: An integer $z \in \mathbb{Z}$ sampled from a distribution very close to $D_{\mathbb{Z},\sigma',\mu}$

$r \leftarrow \mu - \lfloor \mu \rfloor$ ▷ $r \in [0, 1)$
$ccs \leftarrow \sigma_{\min}/\sigma'$ ▷ $ccs \in [\frac{\sigma_{\min}}{\sigma_{\max}}, 1]$. ▷ `fdiv` (precomputed).
while (1) **do**
  $z_0 \leftarrow$ BaseSampler() ▷ $z_0 \in \{0, \ldots, 18\}$
  $b \leftarrow$ UniformBits(8) `&` `0x1` ▷ $b \in \{0, 1\}$.
  $z \leftarrow b + (2 \cdot b - 1)z_0$ ▷ $z \in \{-18, \ldots, 19\}$
  $x \leftarrow \frac{(z-r)^2}{2\sigma'^2} - \frac{z_0^2}{2\sigma_{\max}^2}$ ▷ $x \in [0, x_{\max}]$ (Lemma 13). `fadd`, `fcmul`, `fmul`.
  if (BerExp$(x, ccs) = 1$) **then**
    return $z + \lfloor \mu \rfloor$ ▷ `fadd`.

Proof (Proof of Lemma 13). We treat SamplerZ and each of its subroutines in a separate paragraph.

SamplerZ. Depending on the sign of b, either $|z-r| = |z_0+1-r|$ or $|z-r| = |z_0+r|$. Either way, $z_0 \le |z - r| \le z_0 + 1$, and by construction $z_0 \in \{0, \ldots, 18\}$. In addition, $\sigma_{\min} \le \sigma' \le \sigma_{\max}$ and $0 \le z_0 \le 18$, therefore we may bound x as:

$$0 \le \frac{z_0^2}{2\sigma'^2} - \frac{z_0^2}{2\sigma_{\max}^2} \le x \le \frac{(z_0+1)^2}{2\sigma_{\min}^2} - \frac{z_0^2}{2\sigma_{\max}^2} \le x_{\max}. \tag{41}$$

Finally, $|z + \lfloor \mu \rfloor| \le 19 + B$.

BerExp. Since $x \in [0, x_{\max}]$ (see above), $s = \lfloor x/\ln(2) \rfloor \in [0, x_{\max}/\ln(2)]$. Then $s \cdot \ln(2) \in [0, x_{\max}]$ and $r = x - \lfloor x/\ln(2) \rfloor \cdot \ln(2) \in [0, \ln(2))$. For z: ApproxExp outputs values in $\{0, \ldots, 2^{63} - 1\}$, and since $ccs \ge \sigma_{\min}/\sigma_{\max}$, its output is at least 1; hence $z = (2 \cdot \text{ApproxExp}(r, ccs) - 1)\texttt{>>}s$ lies in $\{0, \ldots, 2^{64} - 1\}$. Each w is the difference between a uniform byte and a byte of z, hence $w \in \{-255, \ldots, 255\}$.

ApproxExp. The bound on the input x holds by construction (see BerExp). The variables y, z are in $\{0, \ldots, 2^{63} - 1\}$ by construction. □

Algorithm 10 BerExp(x, ccs)

Require: Fixed-point values $x \geq 0$ and $ccs \in [\sigma_{\min}/\sigma_{\max}, 1]$
Ensure: A single bit, equal to 1 with probability $\approx ccs \cdot \exp(-x)$

$s \leftarrow \lfloor x/\ln(2) \rfloor$ ▷ $s \in [0, x_{\max}/\ln(2)]$. `fcmul`
$r \leftarrow x - s \cdot \ln(2)$ ▷ $r \in [0, \ln 2)$. `fadd`, `fcmul`
$s \leftarrow \min(s, 63)$ ▷ $s \in [0, 63]$
$z \leftarrow (2 \cdot \text{ApproxExp}(r, ccs) - 1) \texttt{>>} s$ ▷ $z \approx 2^{64-s} \cdot ccs \cdot \exp(-r) = 2^{64} \cdot ccs \cdot \exp(-x)$
$i \leftarrow 64$
do
  $i \leftarrow i - 8$
  $w \leftarrow \text{UniformBits}(8) - ((z \texttt{>>} i)$ `&` `0xFF`$)$ ▷ Lazy byte-wise comparison of UniformBits(64) to z
while $(w = 0)$ and $(i > 0)$
return $[\![w < 0]\!]$ ▷ Return 1 with probability $2^{-64} \cdot z \approx ccs \cdot \exp(-x)$

Algorithm 11 ApproxExp(x, ccs)

Require: Fixed-point values $x \in [0, \ln(2)]$ and $ccs \in [0, 1]$
Ensure: An integral approximation of $2^{63} \cdot ccs \cdot \exp(-x)$

$C =$
`[0x00000004741183A3,0x00000036548CFC06,0x0000024FDCBF140A, 0x0000171D939DE045,`
`0x0000D00CF58F6F84, 0x000680681CF796E3, 0x002D82D8305B0FEA, 0x011111110E066FD0,`
`0x0555555555070F00, 0x155555555581FF00, 0x400000000002B400, 0x7FFFFFFFFFFF4800,`
`0x8000000000000000]`
$y \leftarrow C[0]$ ▷ y and z remain in $\{0, ..., 2^{63} - 1\}$ the whole algorithm.
$z \leftarrow \lfloor 2^{63} \cdot x \rfloor$ ▷ Can be done by shifting the fixed-point value.
for $i = 1, \ldots, 12$ **do**
  $y \leftarrow C[i] - (z \cdot y) \texttt{>>} 63$ ▷ `fadd`, `fmul`
  ▷ $(z \cdot y)$ fits in 126 bits, but we only need the top 63 bits
$z \leftarrow \lfloor 2^{63} \cdot ccs \rfloor$ ▷ Precomputed.
$y \leftarrow (z \cdot y) \texttt{>>} 63$ ▷ `fmul`
return y

A square-root-free variant (not deployed). The only square root in the signing pipeline is the one producing the normalized leaves $\sigma' = \sigma/\sqrt{\text{leaf.value}}$ at key generation (Line 7 of Algorithm 1), from which the precomputed constants $1/(2\sigma'^2)$ and $ccs = \sigma_{\min}/\sigma'$ are derived. It can be avoided altogether by storing $1/(2\sigma'^2)$ only and working with $ccs^2 = \sigma_{\min}^2/\sigma'^2$: drawing $u \leftarrow \text{UniformBits}(64)$ and comparing u^2 against $\left(2^{64} \cdot ccs \cdot \exp(-x)\right)^2$ — which is computable from ccs^2 — samples the same Bernoulli distribution as BerExp without any square root. This consumes about

$\times 1.72$ as much randomness per iteration of SamplerZ, but makes the consumption pattern simpler, as BerExp then always uses the same amount of randomness. Since randomness is consumed differently, this variant is KAT-breaking; we therefore do not deploy it, and instead keep the original BerExp above and compute the square root with a fast constant-time iterative routine (see Section 6.3).

B Deferred Proofs

B.1 Proof of Lemma 2

Proof. Let $\beta_k \coloneqq e^{-2\pi^2 k^2 \sigma^2}$. The Poisson summation formula gives, for all $c \in \mathbb{R}$:

$$S(c) \coloneqq \rho_\sigma(\mathbb{Z} - c) = \sigma\sqrt{2\pi}\,(1 + \tilde{y}(c)), \qquad \tilde{y}(c) \coloneqq 2\sum_{k\geq 1} \beta_k \cos(2\pi k c). \tag{42}$$

Both sides of (42) are sums that converge uniformly on compacts together with their derivatives (Gaussian decay on the left, doubly exponential on the right), so we may differentiate (42) four times term by term. This gives, on the left and right respectively:

$$S'(c) = S(c)\cdot m_1/\sigma^2 \qquad S'(c) = \sigma\sqrt{2\pi}\,\tilde{y}'(c) \tag{43}$$

$$S''(c) = S(c)\cdot (m_2/\sigma^4 - 1/\sigma^2) \qquad S''(c) = \sigma\sqrt{2\pi}\,\tilde{y}''(c) \tag{44}$$

$$S''''(c) = S(c)\cdot (m_4/\sigma^8 - 6\,m_2/\sigma^6 + 3/\sigma^4) \qquad S''''(c) = \sigma\sqrt{2\pi}\,\tilde{y}''''(c) \tag{45}$$

Equating both sides gives:

$$m_1 = \sigma^2 \cdot \frac{\tilde{y}'(c)}{1+\tilde{y}(c)}, \qquad \tilde{y}'(c) = -4\pi \sum_{k\geq 1} k\,\beta_k\,\sin(2\pi k c), \tag{46}$$

$$m_2 = \sigma^2 + \sigma^4 \cdot \frac{\tilde{y}''(c)}{1+\tilde{y}(c)}, \qquad \tilde{y}''(c) = -8\pi^2 \sum_{k\geq 1} k^2\,\beta_k\,\cos(2\pi k c), \tag{47}$$

$$m_4 - 6\,\sigma^2\,m_2 + 3\,\sigma^4 = \frac{\sigma^8\cdot \tilde{y}''''(c)}{1+\tilde{y}(c)}, \qquad \tilde{y}''''(c) = 32\pi^4 \sum_{k\geq 1} k^4\,\beta_k\,\cos(2\pi k c). \tag{48}$$

We now bound the Fourier sums. By definition of $\eta_\epsilon(\mathbb{Z})$, $\beta_1 \leq \sum_{k\geq 1}\beta_k \leq \epsilon/2 \leq 1/8$. We note that by definition $\beta_k = \beta_1^{k^2}$, so the terms $k \geq 2$ are dominated by β_1^3: for $j \in \{1,2,4\}$ and $\beta_1 \leq 1/8$,

$$\sum_{k\geq 2} k^j \beta_1^{k^2-1} = \beta_1^3 \sum_{k\geq 2} k^j \beta_1^{k^2-4} \leq 17\,\beta_1^3 \leq \frac{17\,\epsilon^3}{8} \leq \epsilon,$$

where the middle inequality isolates the term $k = 2$ (which equals $k^j \leq 16$) and the last one uses $\epsilon \leq 1/4$. Hence:

$$\sum_{k\geq 1} k^j \beta_k = \beta_1\Big(1 + \sum_{k\geq 2} k^j \beta_1^{k^2-1}\Big) \leq \beta_1\,(1+\epsilon) \leq \frac{\epsilon}{2\,(1-\epsilon)}.$$

Multiplying by the prefactors of $\tilde{y}'$, $\tilde{y}''$, $\tilde{y}''''$ gives

$$|\tilde{y}'| \leq 2\pi\epsilon/(1-\epsilon) \tag{49}$$
$$|\tilde{y}''| \leq 4\pi^2\epsilon/(1-\epsilon) \tag{50}$$
$$|\tilde{y}''''| \leq 16\pi^4\epsilon/(1-\epsilon) \tag{51}$$

Furthermore, $1+\tilde{y} \geq 1-\epsilon$. Injecting into (46) concludes the proof. □

B.2 Proof of Lemma 4

Proof. We start from the following identity:

$$\alpha(y-\delta)^2 + (1-\alpha)y^2 = (y-\alpha\delta)^2 - \alpha(\alpha-1)\delta^2$$

Setting $y = x - c$, dividing by $2\,\sigma^2$, exponentiating and summing over $x \in \mathbb{Z}$ gives:

$$\sum_{x\in\mathbb{Z}} \rho_\sigma(x-c-\delta)^\alpha\, \rho_\sigma(x-c)^{1-\alpha} = e^{\frac{\alpha(\alpha-1)\delta^2}{2\sigma^2}}\, \rho_\sigma(\mathbb{Z}-c-\alpha\delta)$$

This is equivalent to:

$$\begin{aligned}(\alpha-1)\ln R_\alpha(D_{\mathbb{Z},\sigma,c+\delta}, D_{\mathbb{Z},\sigma,c}) &= \frac{\alpha(\alpha-1)\delta^2}{2\sigma^2} + \ln\left(\frac{\rho_\sigma(\mathbb{Z}-c-\alpha\delta)}{\rho_\sigma(\mathbb{Z}-c-\delta)^\alpha\, \rho_\sigma(\mathbb{Z}-c)^{1-\alpha}}\right) \\ &= \frac{\alpha(\alpha-1)\delta^2}{2\sigma^2} + \underbrace{\varphi(c+\alpha\,\delta) - \alpha\,\varphi(c+\delta) - (1-\alpha)\varphi(c)}_{\coloneqq \Delta(\delta)}\end{aligned} \tag{52}$$

where $\varphi(u) \coloneqq \ln \rho_\sigma(\mathbb{Z}-u)$. Our goal in the rest of the proof is to bound $\Delta(\delta)$. We have $\Delta'(\delta) = \alpha\,(\varphi'(c+\alpha\,\delta) - \varphi'(c+\delta))$ and $\Delta''(\delta) = \alpha^2\varphi''(c+\alpha\,\delta) - \alpha\varphi''(c+\delta)$. We note that $\Delta(0) = \Delta'(0) = 0$, so Taylor's theorem gives us:

$$|\Delta(\delta)| \leq \frac{\alpha(\alpha+1)\,\delta^2}{2} \sup|\varphi''| \tag{53}$$

We now bound $|\varphi''|$. Derivating φ twice gives, with $z \sim D_{\mathbb{Z},\sigma,c}$, and applying Lemma 2 to bound $\mathbb{E}[(z-c)^2] - \sigma^2$ and $\mathbb{E}[z-c]$ gives, with $m_i \coloneqq \mathbb{E}[(z-c)^i]$:

$$\varphi''(c) = \frac{\mathrm{Var}[z]}{\sigma^4} - \frac{1}{\sigma^2} = \frac{m_2 - m_1^2 - \sigma^2}{\sigma^4} = O(\epsilon) \tag{54}$$

Combining Eqs. (52) to (54) concludes the proof. □

B.3 Proof of Lemma 5

Proof. Set $\tau_i \coloneqq 1/\sigma_i^2$ and $s \coloneqq \tau_2 - \tau_1$. Since $D_{\sigma,c}(z) = \rho_\sigma(z-c)/\rho_\sigma(\mathbb{Z}-c)$, and using $\frac{\alpha}{\sigma_2^2} + \frac{1-\alpha}{\sigma_1^2} = \alpha\tau_2 + (1-\alpha)\tau_1 = \tau_1 + \alpha s$, we can write:

$$\begin{aligned} R_\alpha(D_{\mathbb{Z},\sigma_2,c}, D_{\mathbb{Z},\sigma_1,c})^{\alpha-1} &= \sum_{z\in\mathbb{Z}} \frac{D_{\sigma_2,c}(z)^\alpha}{D_{\sigma_1,c}(z)^{\alpha-1}} \\ &= \rho_{\sigma_1}(\mathbb{Z}-c)^{\alpha-1}\,\rho_{\sigma_2}(\mathbb{Z}-c)^{-\alpha} \sum_{z\in\mathbb{Z}} e^{-\frac{(z-c)^2}{2}(\tau_1+\alpha s)} \\ &= \rho_{\sigma_1}(\mathbb{Z}-c)^{\alpha-1}\,\rho_{\sigma_2}(\mathbb{Z}-c)^{-\alpha}\,\rho_{1/\sqrt{\tau_1+\alpha s}}(\mathbb{Z}-c). \end{aligned} \tag{55}$$

Taking logarithms and noting $\psi(\tau) \coloneqq \ln \rho_{1/\sqrt{\tau}}(\mathbb{Z}-c)$, (55) is equivalent to:

$$(\alpha-1)\ln R_\alpha(D_{\mathbb{Z},\sigma_2,c}, D_{\mathbb{Z},\sigma_1,c}) = \underbrace{\psi(\tau_1+\alpha s) - \alpha\,\psi(\tau_1+s) - (1-\alpha)\,\psi(\tau_1)}_{\coloneqq\, \Gamma(s)}. \tag{56}$$

Our goal in the rest of the proof is to bound $\Gamma(s)$. We have $\Gamma'(u) = \alpha\,(\psi'(\tau_1+\alpha u) - \psi'(\tau_1+u))$ and $\Gamma''(u) = \alpha^2\,\psi''(\tau_1+\alpha u) - \alpha\,\psi''(\tau_1+u)$. We note that $\Gamma(0) = \Gamma'(0) = 0$, so Taylor's theorem gives us:

$$|\Gamma(s)| \le \frac{\alpha(\alpha+1)\,s^2}{2} \sup_J |\psi''|, \tag{57}$$

where J is the segment between τ_1 and $\tau_1+\alpha s$. We note that $|s| \le \delta\,\tau_1$ and therefore:

$$\tau_1(1-\alpha\,\delta) \le \min J \le \max J \le \tau_1(1+\alpha\,\delta)$$

It remains to bound $|\psi''|$ on J. Differentiating ψ twice yields, for $X \sim D_{\mathbb{Z},1/\sqrt{\tau},c}$:

$$\psi''(\tau) = \frac{1}{4}\,\mathrm{Var}\left[(X-c)^2\right] \ge 0. \tag{58}$$

Since $\sigma(\tau) \coloneqq 1/\sqrt{\tau}$ satisfies $\sigma(\tau) \ge \sigma_1/\sqrt{1+\alpha\,\delta} \ge \eta_\epsilon(\mathbb{Z})$, Lemma 2 applies:

$$\mathrm{Var}\left[(X-c)^2\right] = 2\,\sigma(\tau)^4 + O(\epsilon\,\sigma(\tau)^6 + \epsilon\,\sigma(\tau)^8) \tag{59}$$

Combining Eqs. (56) to (59), and using $\frac{\alpha+1}{\alpha-1} \le 3$ concludes the proof. □

B.4 Proof of Lemma 8

Proof. We recall that $\hat{\mathbf{G}}^{[1]} = \hat{\mathbf{G}} = \hat{\mathbf{B}} \odot \hat{\mathbf{B}}^\star$ where $\hat{\mathbf{B}} = \begin{bmatrix} \hat{g} & -\hat{f} \\ \hat{G} & -\hat{F} \end{bmatrix}$. A simple computation, using the NTRU equation $f\,G - g\,F = q$ (so that $\det \hat{\mathbf{G}}^{[1]}[j] = q^2$ for every

index j), shows that $(\hat{\mathbf{L}}^{[1]}, \hat{\mathbf{D}}^{[1]}) = \mathsf{LDL}^\star(\hat{\mathbf{G}}^{[1]})$ satisfy:

$$\hat{\mathbf{L}}^{[1]} = \left[\begin{array}{c|c} \hat{1} & \hat{0} \\ \hline \hat{L}_{10} & \hat{1} \end{array}\right], \qquad \text{with } L_{10} = \frac{F\,f^\star + G\,g^\star}{f\,f^\star + g\,g^\star} \tag{60}$$

$$\hat{\mathbf{D}}^{[1]} = \left[\begin{array}{c|c} \hat{u} & \hat{0} \\ \hline \hat{0} & q^2 \cdot \hat{u}^{-1} \end{array}\right], \quad \text{with } u = ff^\star + gg^\star. \tag{61}$$

We prove Items 1 and 2, which together imply Eq. (15).

1. *B-boundedness.* All matrices $\hat{\mathbf{D}}^{[i]}$ are upper-bounded by $B = q \cdot (\alpha_{\text{hybrid}})^2$.
2. *q^2-symplectic pairs.* At each level $\ell \geq 0$, for any $2^\ell \leq i_0, i_1 < 2^{\ell+1}$ such that $i_0 + i_1 = 2^\ell + 2^{\ell+1} - 1$, $(\hat{\mathbf{D}}^{[i_0]}, \hat{\mathbf{D}}^{[i_1]})$ are a q^2-symplectic pair (see Fig. 6 for an illustration).

It is clear from Eq. (61) that Items 1 and 2 are true for $\ell = 0$, and in particular $\hat{\mathbf{D}}^{[1]}$ is symplectic. We will show by induction that they are true for $i > 1$. To make the proof more concise, for a matrix $\hat{\mathbf{M}} = (m_{i,i'}) \in (\mathbb{C}^k)^{2\times 2}$ and $j \in [k]$, we note $\hat{\mathbf{M}}[j] = \left[\begin{array}{c|c} m_{0,0}[j] & m_{0,1}[j] \\ \hline m_{1,0}[j] & m_{1,1}[j] \end{array}\right] \in \mathbb{C}^{2\times 2}$.

B-boundedness (Item 1). For $i_0 \geq 1$, assume that $\hat{\mathbf{D}}^{[i_0]} = \begin{bmatrix} \hat{D}_{00}^{[i_0]} & 0 \\ 0 & \hat{D}_{11}^{[i_0]} \end{bmatrix}$ is B-bounded. Our goal is to show that its two "children" matrices $\hat{\mathbf{D}}^{[2\cdot i_0]}$ and $\hat{\mathbf{D}}^{[2\cdot i_0+1]}$ are also B-bounded. For $j \in [k/2]$, let $a = \hat{D}_{00}^{[i_0]}[j]$ and $b = \hat{D}_{00}^{[i_0]}[k/2 - j - 1]$; in Falcon's FFT ordering, these two indices hold the evaluations at a pair of opposite roots $\pm\zeta$. By definition of splitfft:

$$\hat{\mathbf{G}}^{[2\cdot i_0]}[j] = \hat{\mathbf{G}_0}^{[i_0]}[j] = \begin{bmatrix} \frac{a+b}{2} & \frac{\zeta^\star(a-b)}{2} \\ \frac{\zeta(a-b)}{2} & \frac{a+b}{2} \end{bmatrix}. \tag{62}$$

Applying $\mathsf{LDL}^\star$ to $\hat{\mathbf{G}}^{[2\cdot i_0]}$ gives $\hat{\mathbf{D}}^{[2\cdot i_0]}$:

$$\hat{\mathbf{D}}^{[2\cdot i_0]}[j] = \begin{bmatrix} a' & 0 \\ 0 & b' \end{bmatrix}, \quad \text{where} \quad (a', b') = f(a, b). \tag{63}$$

Where f is defined as follows:

$$\begin{aligned} f : (\mathbb{R}_+^*)^2 &\longrightarrow (\mathbb{R}_+^*)^2 \\ (a, b) &\longmapsto \left(\frac{a+b}{2}, \frac{2ab}{a+b}\right). \end{aligned}$$

Since $a, b > 0$, we have $\frac{a+b}{2} \leq \max(a, b)$. Similarly, via the arithmetic mean–harmonic mean (AM–HM) inequality, $\frac{2ab}{a+b} \leq \frac{a+b}{2} \leq \max(a, b)$. In other words,

$\|f(a,b)\|_\infty \leq \|(a,b)\|_\infty$. Note also that f preserves products — $\frac{a+b}{2} \cdot \frac{2ab}{a+b} = a\,b$, the product of the arithmetic and harmonic means being the squared geometric mean — which is the mechanism by which the determinant q^2 propagates down the tree in Item 2 below. Since the bound above is true for $j \in [k/2]$, it implies that $\hat{\mathbf{D}}^{[2\cdot i_0]}$ is B-bounded. By the exact same process, we can show that $\hat{\mathbf{D}}^{[2\cdot i_0+1]}$ is B-bounded. By induction, this proves Item 1.

q^2-symplectic pairs. With the same notations as above, we assume by the induction hypothesis that $(\hat{\mathbf{D}}^{[i_1]}, \hat{\mathbf{D}}^{[i_0]})$ is a symplectic pair, for i_1 defined in Item 2. For $j \in [k/2]$, let $c = \hat{D}_{11}^{[i_1]}[j]$ and $d = \hat{D}_{11}^{[i_1]}[k/2-j-1]$. Similarly to Eq. (63):

$$\hat{\mathbf{D}}^{[2\cdot i_1+1]}[j] = \begin{bmatrix} c' & 0 \\ 0 & d' \end{bmatrix}, \quad \text{where} \quad (c', d') = f(c, d). \tag{64}$$

From Eq. (14), the induction hypothesis reads entrywise as $\hat{D}_{00}^{[i_0]}[j'] \cdot \hat{D}_{11}^{[i_1]}[j'] = q^2$ for every index j'; in particular, $a \cdot c = b \cdot d = q^2$, and it follows that:

$$a' \cdot d' = \frac{a+b}{2} \cdot \frac{2\,c\,d}{c+d} = \frac{(a+b)c\,d}{c+d} = \frac{(a+b)\cdot q^2/a \cdot q^2/b}{q^2/a + q^2/b} = q^2. \tag{65}$$

Similarly, $b' \cdot c' = q^2$. Since this is true for any $j \in [k/2]$, $(\hat{\mathbf{D}}^{[2\cdot i_0]}, \hat{\mathbf{D}}^{[2\cdot i_1+1]})$ is a q^2-symplectic pair. Similarly, $(\hat{\mathbf{D}}^{[2\cdot i_0+1]}, \hat{\mathbf{D}}^{[2\cdot i_1]})$ is a q^2-symplectic pair. By induction, this proves Item 2.

Putting everything together. Every $\hat{\mathbf{D}}^{[i_0]}$ is B-bounded (Item 1), and they all admit an (also B-bounded) $\hat{\mathbf{D}}^{[i_1]}$ with which they form a q^2-symplectic pair (Item 2). Consequently, every $\hat{\mathbf{D}}^{[i_0]}$ has every entry of their diagonal coefficients *lower-bounded* by $q^2/B = q/(\alpha_{\text{hybrid}})^2$. This proves Eq. (15).

We now prove Eq. (16). Consider $\hat{L}_{10}^{[i]}$ as computed in Line 2 of LDL*. Since the input $\hat{\mathbf{G}}^{[i]}$ was computed as in Line 10 of ffLDL*, we have:

$$\hat{L}_{10}^{[i]}[j] = \frac{\zeta(a-b)}{a+b}, \tag{66}$$

for $q/(\alpha_{\text{hybrid}})^2 \leq a, b \leq q \cdot (\alpha_{\text{hybrid}})^2$ and $|\zeta| = 1$. In absolute value, the extremal values of Eq. (66) are attained when a is minimal and b is maximal, or vice-versa. Eq. (16) follows. □

B.5 Proof of Lemma 12

Proof. In order to facilitate our proof, we number the recursive calls to ffSampling and the corresponding variables using the same notations as in Remark 1: ffSampling$^{[i]}$, $\hat{\mathbf{t}}^{[i]}$, $\hat{\mathbf{z}}^{[i]}$, $\hat{\mathbf{s}}^{[i]}$, and so on. Let $\hat{\mathbf{G}}^{[i]}$ be the matrix in the i-th node of the FALCON tree, as defined by ffLDL*. We define $\hat{\mathbf{B}}^{[i]}$ via the factorization $\hat{\mathbf{G}}^{[i]} = \hat{\mathbf{B}}^{[i]} \cdot (\hat{\mathbf{B}}^{[i]})^\star$ of Lemma 11.

Bounding $\|\hat{\mathbf{z}} - \hat{\mathbf{t}}\|_\infty$. In ffSampling, we need to bound the intermediate variables $\hat{\mathbf{t}}, \hat{\mathbf{z}}$ and $\hat{t}'_0$. Since ffSampling is a recursive algorithm, this is best achieved by determining inference rules. Suppose that $\hat{\mathbf{t}} = \hat{\mathbf{t}}^{[i]}$ is B-bounded. We bound $\hat{\mathbf{z}}$ via concentration bounds on affine projections of discrete Gaussians. We have:

$$\mathbf{s} = (\mathbf{t} - \mathbf{z}) \cdot \mathbf{B} \tag{67}$$

$$\Leftrightarrow \quad (\hat{\mathbf{t}} - \hat{\mathbf{z}}) = \hat{\mathbf{s}} \cdot \hat{\mathbf{B}}^{-1} \tag{68}$$

$$\Rightarrow \quad \|\hat{\mathbf{t}} - \hat{\mathbf{z}}\|_\infty \leq \|\hat{\mathbf{s}}\|_\infty \cdot \|\hat{\mathbf{B}}^{-1}\|_\infty, \tag{69}$$

where $\|\hat{\mathbf{B}}^{-1}\|_\infty$ is the operator norm of $\hat{\mathbf{B}}^{-1}$ for the infinity norm, and is upper bounded by 2 times the maximum of $\hat{\mathbf{B}}^{-1}$'s coefficients, since $\hat{\mathbf{B}}^{-1}$ is a 2×2 matrix. We bound $\|\hat{\mathbf{s}}\|_\infty$ and $\|\hat{\mathbf{B}}^{-1}\|_\infty$ separately.

Noting $\mathbf{s} = (s_0, s_1)$, for $j \in [k]$, we can express the j-th entry of $\hat{s}_0$ as the inner product between the vector of coefficients of s_0 and $\boldsymbol{\zeta}_j = (1, \zeta_j, \zeta_j^2, \ldots, \zeta_j^{k-1})$, where ζ_j is the j-th complex root of $x^k + 1$. We model the distribution of $\mathbf{s}$ as the discrete Gaussian $D_{\mathbb{Z}^{2n},\sigma}$[11]. This allows to leverage Lemma 1:

$$\mathbb{P}[|\langle s_0, \boldsymbol{\zeta}_i \rangle| > \tau\, \sigma\, \sqrt{k}] \leq 2e^{-\tau^2/2}. \tag{70}$$

We set $\tau = \sqrt{2(\lambda + \log_2(2n \log n) + 1) \log 2}$, so that this is true with probability $\geq 1 - 2^{-\lambda}$ for all the entries of all the $\mathbf{s}^{[i]}$ of a given execution. Note that each $\mathbf{s}$ is real-valued, so its FFT entries at conjugate embeddings share the same magnitude: at most $n(\log_2 n + 1) \leq 2\, n \log n$ of these events are distinct, which the $\log_2(2n \log n)$ slack in τ covers. This bounds $\|\hat{\mathbf{s}}\|_\infty$ with overwhelming probability.

The entries of $\hat{\mathbf{B}}^{-1}$ are B_{inv}-bounded, for B_{inv} defined in Lemma 11. Therefore:

$$\|\hat{\mathbf{z}} - \hat{\mathbf{t}}\|_\infty \leq A \cdot \sqrt{k}, \text{ where } \begin{cases} A = A_{\text{root}} \coloneqq 2\, \tau\, \sigma \cdot \frac{\gamma_{F,G}}{q} & \text{if } i = 1 \\ A = A_{\text{child}} \coloneqq 2\, \tau\, \sigma \cdot \frac{\alpha_{\text{hybrid}}}{\sqrt{q}} & \text{if } i \geq 2. \end{cases}$$

We obtain Eq. (24) by applying the triangle inequality.

Bounding every $\|\hat{\mathbf{t}}^{[i]}\|_\infty$. Let $B_{\mathbf{t}}^{[i]}$ be an upper bound on $\|\hat{\mathbf{t}}^{[i]}\|_\infty$. We recall that

$$\|\mathsf{splitfft}(\hat{t})\|_\infty \leq \|\hat{t}\|_\infty \tag{71}$$

$$\|\mathsf{mergefft}(\hat{\mathbf{t}})\|_\infty \leq 2\|\hat{\mathbf{t}}\|_\infty \tag{72}$$

$$\hat{\mathbf{t}}^{[2\,i+1]} = \mathsf{splitfft}(\hat{t}_1^{[i]}) \tag{73}$$

$$\hat{\mathbf{t}}^{[2\,i]} = \mathsf{splitfft}(\hat{t}_0'^{[i]}). \tag{74}$$

[11] In fact, $\mathbf{s}$ is Rényi-close to a Gaussian discretized, not over $\mathbb{Z}^{2n}$, but *the coset of preimages of* c. However, in the random oracle model c is uniformly random so that, under the randomness of c, $\mathbf{s}$ is a discrete Gaussian over $\mathbb{Z}^{2n}$.

We now show that:

$$\forall i \geq 1: \qquad B_{\mathbf{t}}^{[2\,i+1]} \leq B_{\mathbf{t}}^{[i]} \tag{75}$$

$$\forall i \geq 2: \qquad B_{\mathbf{t}}^{[2\,i]} \leq B_{\mathbf{t}}^{[i]} + 2\,A_{\text{child}}\,\sqrt{k} \tag{76}$$

$$i = 1: \qquad B_{\mathbf{t}}^{[2]} \leq B_{\mathbf{t}}^{[1]} + 2\,\gamma_{\text{root}}\,A_{\text{child}}\sqrt{n}. \tag{77}$$

Eq. (75) is true because splitfft does not increase the norm (Eq. (71)). For Eqs. (76) and (77), we start from Line 10 and apply Eqs. (73) and (74):

$$\begin{aligned} & t_0'^{[i]} = t_0^{[i]} + (t_1^{[i]} - z_1^{[i]}) \odot \ell^{[i]} \\ \Rightarrow \quad & \mathbf{t}^{[2\,i]} = \mathsf{splitfft}(t_0^{[i]}) + (\mathbf{t}^{[2\,i+1]} - \mathbf{z}^{[2\,i+1]}) \odot \mathbf{M}, \end{aligned}$$

where $\mathbf{M} = \begin{bmatrix} \ell_0 & \ell_1 \\ \ell_1^\star & \ell_0 \end{bmatrix}$, with $(\ell_0, \ell_1) \coloneqq \mathsf{splitfft}(\ell^{[i]})$. In particular, the entries of $\mathbf{M}$ are bounded by $\|\ell^{[i]}\|_\infty$. Applying the triangle inequality and Eq. (71) gives:

$$\|\mathbf{t}^{[2\,i]}\|_\infty \leq \|\mathbf{t}^{[i]}\|_\infty + \|\mathbf{t}^{[2\,i+1]} - \mathbf{z}^{[2\,i+1]}\|_\infty \cdot 2\,\|\ell^{[i]}\|_\infty \tag{78}$$

$$\Rightarrow \qquad B_{\mathbf{t}}^{[2\,i]} \leq B_{\mathbf{t}}^{[i]} + A_{\text{child}}\sqrt{k} \cdot 2\,\|\ell^{[i]}\|_\infty. \tag{79}$$

We finally obtain Eqs. (76) and (77) by combining Eq. (79) with $\|\ell^{[i]}\|_\infty < 1$ if $i \geq 2$, and $\|\ell^{[1]}\|_\infty \leq \gamma_{\text{root}}$.

Eqs. (75) to (77) allow us to bound all the $\mathbf{t}^{[i]}$. The worst-case is attained at the bottom leftmost node $i = n$, where we have:

$$\begin{aligned} B_{\mathbf{t}}^{[n]} &\leq B_{\mathbf{t}}^{[1]} + 2\,A_{\text{child}}\left(\gamma_{\text{root}}\sqrt{n} + \left(\sqrt{n/2} + \sqrt{n/4} + \cdots + 1\right)\right) \\ B_{\mathbf{t}}^{[n]} &\leq B_{\mathbf{t}}^{[1]} + 2\,A_{\text{child}}\left(\gamma_{\text{root}}\sqrt{n} + \frac{\sqrt{n}-1}{\sqrt{2}-1}\right) \\ B_{\mathbf{t}}^{[n]} &\leq B_{\mathbf{t}}^{[1]} + 2\,A_{\text{child}}\,\sqrt{n}\left(\gamma_{\text{root}} + \sqrt{2} + 1\right). \end{aligned}$$

This gives Eq. (23) and concludes the proof. □

B.6 Proof of Lemma 15

Proof. The argument is essentially that of Lemma 14, and we reproduce it. We work by contraposition: assume that $C_{\mathcal{Q}}$ is not $(\lambda-1)$-bit secure, i.e., there is an attacker breaking $C_{\mathcal{Q}}$ with probability $\epsilon_{\mathcal{Q}} > 2^{-(\lambda-1)}$. Let $\epsilon_{\mathcal{P}}$ denote the probability that the same attacker breaks $C_{\mathcal{P}}$, where the N-uple is drawn from $\mathcal{P}$ instead of $\mathcal{Q}$. By probability preservation, for all $a \in (1, +\infty)$:

$$\epsilon_{\mathcal{P}} \geq \epsilon_{\mathcal{Q}}^{a/(a-1)} / R_a(\mathcal{Q}, \mathcal{P}). \tag{80}$$

Set $a \coloneqq 2\lambda - 1$ in Eq. (80). For the numerator, we have

$$\epsilon_{\mathcal{Q}}^{a/(a-1)} > \left(2^{-(\lambda-1)}\right)^{\frac{2\lambda-1}{2(\lambda-1)}} = \sqrt{2} \cdot 2^{-\lambda},$$

while Eq. (28) bounds the denominator as

$$R_{2\lambda-1}(\mathcal{Q}, \mathcal{P}) \leq \left(1 + \frac{1}{4N}\right)^{N} \leq e^{1/4} \leq \sqrt{2}.$$

Thus $\epsilon_{\mathcal{P}} > 2^{-\lambda}$: the attacker also breaks $C_{\mathcal{P}}$ with success probability strictly higher than $2^{-\lambda}$, contradicting the λ-bit security of $C_{\mathcal{P}}$. □