Flying, Not Floating: Precise Fixed-Point Falcon on Embedded Processors

Stef Halmans [1], Amit Kumar [1], Tim Güneysu [1,2]

[1] Ruhr University Bochum, Bochum, Germany
stef.halmans@rub.de, ak10495@nyu.edu, tim.gueneysu@rub.de
[2] German Research Center for Artificial Intelligence (DFKI), Bremen, Germany

Abstract. The Falcon digital signature scheme, under standardization as FN-DSA, requires real-number arithmetic during signing, with its security dependent on numerical precision. Recently, De Almeida Braga et al. (CRYPTO 2026) showed that Falcon-512 can sign in 64.64 fixed-point arithmetic on 64-bit hosts. In this work, we present the first fixed-point FN-DSA signer for microcontrollers, using 32.64 arithmetic in constant-time Cortex-M4 assembly for Falcon-512 and Falcon-1024. Although a straightforward fixed-point implementation is less precise than double precision, our new techniques successfully restore precision. They keep reciprocals, inverse square roots and small constants wide and round once at the consumer. We evaluate them with known techniques on both arithmetics and both degrees. Our signer needs around 45 % fewer cycles than double-precision emulation for Falcon-512 / Falcon-1024, and its errors support an estimated $2^{99.7}$ / $2^{96.7}$ queries per key, against $2^{61.5}$ / $2^{58.7}$ for double precision and the target of 2^{64}. Our 12-byte words raise signing RAM from $59n$ to $83n$ bytes. We also show which implementation choices lift double-precision c-fn-dsa above 2^{64} while making it faster. We derive key generation bounds for Falcon-1024, accepting 72.3 % of keys where the Falcon-512 values accept 8.5 %, and provide recommendations for the fixed-point FN-DSA that NIST plans.

Keywords: FN-DSA · Falcon · Fixed-Point Arithmetic · ARM Cortex-M4 · Numerical Precision · Post-Quantum Signatures

1 Introduction

Large-scale quantum computers will break currently deployed RSA and elliptic-curve digital signatures [Sho94]. In response, NIST standardized ML-DSA [Nat24a] and SLH-DSA [Nat24b], and is standardizing Falcon [PFH+20] as FN-DSA in FIPS 206. Falcon features the smallest signature sizes among the three, making it attractive for certificates and constrained devices. However, its signing algorithm relies on fast Fourier sampling [DP16] over real numbers, where the underlying Rényi divergence security proof requires numerical precision well beyond single precision [Pre17; PFH+20]. Recently, Halmans et al. [HVS+26] demonstrated that standard double precision fails to satisfy the specification's precision bound and resolved the gap using triple-word arithmetic at approximately twice the computational cost. While key generation can avoid floating-point operations [Por23; Por25b] and verification uses only integer arithmetic, signing remains uniquely dependent on floating-point arithmetic. On embedded platforms, this dependency presents significant challenges: many microcontrollers lack a double-precision floating-point unit (FPU), hardware floating-point division is frequently non-constant-time, and side-channel masking is expensive [CC24]. Fixed-point arithmetic addresses these issues by executing entirely on the integer unit, yielding absolute rounding errors independent of operand magnitude, and providing better suitability for masking [DFLP26a]. For instance, on the Cortex-M4F,

where the FPU supports only single precision, double precision must be emulated in constant time [Por19; Por25a]. On this platform, an emulated double-precision addition requires 70 cycles compared to 3 cycles for a fixed-point addition (Table 6), with emulated operations accounting for up to two-thirds of the total signing latency [Por26a].

De Almeida Braga et al. [DFLP26b] made fixed-point signing possible for Falcon: key generation checks give the integer budget for every intermediate value, and a Rényi argument[1] replaces the specification's worst-case bound on the sampler inputs by their RMSE and turns it into a number of supported signing queries. Their implementations are a 64.64 C signer for 64-bit hosts [DAB26] and a Type 2 Python reference [Pre26], whose precision figures, like their thresholds, cover Falcon-512 only. In response to their findings, NIST announced plans to specify FN-DSA signing in 64.64 fixed-point arithmetic, with 32.64 expected to yield equivalent KATs if key generation follows their recommendations [Per26]. In concurrent work, Pornin [Por26a] restructures double-precision `ffSampling` to cut RAM to $20n$ bytes and, with a new FFT and integer arithmetic modulo q and q', removes over half the floating-point operations, also speeding up signing on the Cortex-M4. In this work we change the arithmetic beneath c-fn-dsa [Por26b], the structure of [PFH+20] whose integer budgets [DFLP26a] proved; as the `ffSampling` changes need new budgets, we use only its integer Gram matrix, signature modulo q' and FFT-new, whose budgets we derive.

To date, no fixed-point signing of Falcon has been demonstrated on a microcontroller platform, and, in particular, it remains unclear, if such an implementation would be precise enough to cover both parameter sets. We now close this gap and present a first fixed-point signer for the Cortex-M4. This requires new fixed-point techniques, which we evaluate for precision and speed together with known techniques from earlier work. We measure precision by Q_s, the number of signatures per key up to which Theorem 2 preserves at least $\lambda - 2$ of the λ bits of security [DFLP26a]; FN-DSA is sized for 2^{64} [PFH+20]. Its input errors come from the worst of 10^5 keys against a 200-bit reference, so Q_s is measured, not proven. Implemented straightforwardly, fixed point reaches a lower Q_s than double precision; with our techniques, it reaches an even higher one. The known techniques raise Q_s in both arithmetics, and fixed point keeps its lead. We make the following contributions.

- The first fixed-point FN-DSA signer on a microcontroller: 32.64 arithmetic, constant time, on the Cortex-M4 at both parameter sets, requiring 44.9 / 45.7 % fewer signing cycles than double-precision emulation for Falcon-512 / 1024 (Sections 3 and 4.2).
- New fixed-point techniques that make the signing sufficiently precise: on an absolute scale, small values such as the reciprocals and the constants $1/q$ and $1/\sigma$ keep few significant bits, so we keep them in a wider format and round only once after multiplication. Without these techniques, fixed point signing is less precise than double precision. By using them we achieve $Q_s = 2^{99.7} / 2^{96.7}$ compared to $2^{61.5} / 2^{58.7}$ for double-precision c-fn-dsa and a target of 2^{64} (Sections 3.4 and 4.1).
- Known techniques from [DFLP26a; Por26a], evaluated separately and in combination on both arithmetics, bring double-precision c-fn-dsa above the 2^{64} target to $Q_s = 2^{75.7} / 2^{72.6}$ and also make it faster (Sections 4.1 and 4.2).
- Recommendations for a fixed-point FN-DSA: key generation bounds for Falcon-1024, accepting 72.3 % of keys where the Falcon-512 values accept 8.5 %, and the operation chain to pin, as exact KATs cannot check precision (Sections 3.3 and 5).

Our 96-bit values, against binary64's 64, raise signing RAM from $59n$ to $83n$ bytes. Pornin [Por26a] reaches $20n$ bytes via its structure; we need 29 % fewer cycles. Combining both, a 64-bit format and masking are future work. Our code is at https://github.com/Chair-for-Security-Engineering/Falcon-FxP-M4.

[1] The conference version proves the worst-case Theorem 1 and uses the root mean square error (RMSE) heuristically; the extended version [DFLP26a] proves it as Theorem 2, cited from here on.

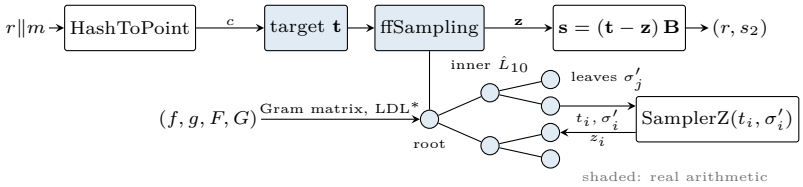

Figure 1: Falcon signing chain. Highlighted boxes are real arithmetic (ieee or fxp32.64), white boxes integer. The pairs (t_i, σ'_i) at `SamplerZ` are what Section 4.1 measures.

Related Work Precision analyses of lattice Gaussian sampling trace back to the lazy floating-point technique of Ducas and Nguyen [DN12]. For Falcon, Hwang [Hwa24] formally verified the FFT's emulated floating-point arithmetic, while Garg et al. [GEP+26] analyzed a fixed-point Klein sampler for threshold signatures, explicitly identifying the fixed-point analysis of `ffSampling` as an open problem. The first microcontroller implementation of Falcon, by Oder et al. [OSHG19] on the Cortex-M4, signs in double precision and computes the Falcon tree during signing rather than storing it. Beyond the implementations by [DFLP26a], no existing Falcon signer executes `ffSampling` in fixed-point arithmetic; other fixed-point FFTs for Falcon serve key generation, in software [Por23] and in an ASIC whose authors retained floating point for signing [AAS+26]. Side-channel and fault attacks on Falcon signing frequently target its IEEE 754 binary64 representation, exploiting the preimage multiplication, integer-to-binary64 conversions, and square-root operations [KA21; GMRR22; LMDG25; KLL+26; ZWSY26; KHTA26]. Masking binary64 arithmetic incurs a 23–118× overhead per operation, resulting in multi-second signing times [CC24; BPT+24], while masking only the multiplication offers insufficient protection [CCCY26]. Notably, Zhou et al. [ZWSY26] note that fixed-point representations eliminate exponent leakage, though unmasked fixed-point values may still reveal their magnitude. Finally, Kaihara et al. [KHTA26] eliminated the square-root operation without performance loss by squaring the Bernoulli test in `SamplerZ`; while integrating this optimization into our signer would simplify the fused leaf (Section 3.4) to a single multiplication, it alters the specific operation chain that FIPS 206 would need to pin.

2 Background

2.1 Falcon in a Nutshell

Falcon follows the hash-and-sign paradigm of Gentry, Peikert, and Vaikuntanathan [GPV08] over NTRU lattices [HPS98]: the private key is a short basis of the lattice, the public key describes the same lattice without it, and a signature is a lattice point close to the hash of the message, which only the short basis can find. It operates over $R = \mathbb{Z}[x]/(x^n + 1)$ with $n \in \{512, 1024\}$ and $q = 12\,289$. The private key consists of $f, g, F, G \in R$ with $fG - gF = q$, forming the NTRU basis $\mathbf{B} = \left[\begin{smallmatrix} g & -f \\ G & -F \end{smallmatrix}\right]$. We write $\hat{f} = \mathsf{FFT}(f)$ for the FFT representation and $\odot$ for the pointwise product in that domain, bold for vectors and matrices, and $\|\hat{a}\|_\infty$ for the largest magnitude over the $n/2$ FFT slots.

To sign a message m, the signer draws a salt r and hashes $c \leftarrow \mathtt{HashToPoint}(r\|m) \in \mathbb{Z}_q^n$. The real target vector $\mathbf{t} = (-cF/q,\ cf/q)$ satisfies $\mathbf{t}\mathbf{B} = (c, 0)$. `ffSampling` [DP16] returns a lattice point $\mathbf{z}$ near $\mathbf{t}$, giving the short signature $\mathbf{s} = (\mathbf{t} - \mathbf{z})\,\mathbf{B}$, restarting in the rare case that $\mathbf{s}$ is longer than verification's bound β. De Almeida Braga et al. [DFLP26a] modify Sign so that `ffSampling` sees only the fractional part of $\mathbf{t}$, the *split* of Section 3.2.

`ffSampling` (Figure 1) draws $\mathbf{z}$ from a discrete Gaussian centered at $\mathbf{t}$, one coefficient at a time: each of the $2n$ coefficients of $\mathbf{z}$ comes from $\mathtt{SamplerZ}(t_i, \sigma'_i)$ [PFH+20, Alg. 12–15], a discrete Gaussian over $\mathbb{Z}$ with center t_i and standard deviation σ'_i. The σ'_i come from

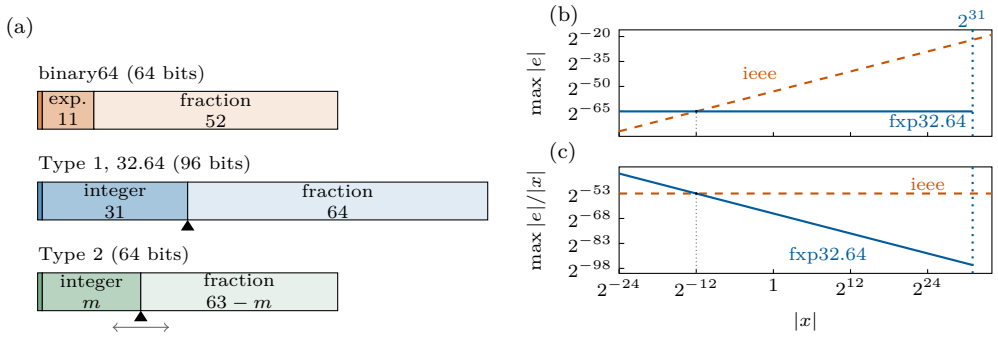

Figure 2: Storage formats (a) and worst-case absolute (b) and relative (c) rounding error. Dark cells are sign bits, triangles binary points.

the key: an LDL* decomposition of the Gram matrix $\mathbf{B}\mathbf{B}^*$, halved in degree level by level, gives a binary tree (the Falcon tree) whose root and inner nodes hold coupling values $\hat{L}_{10}$ and whose n leaves hold the standard deviations. The centers t_i depend on the message and on the samples already drawn: after each subtree the remaining target is corrected with the node's $\hat{L}_{10}$, so an error in $\hat{L}_{10}$ reaches every later center. Following [OSHG19] and the dynamic mode of [Por19], c-fn-dsa, and therefore this paper, builds the tree inside signing and never stores it, one level at a time; this is why reciprocals, square roots and the Gram matrix are signing costs in Tables 6 and 7. `SamplerZ` works on integers and on c-fn-dsa's real type `fpr`, fixed point here, within the $\hat{\mathbf{t}}, \hat{\mathbf{z}}$ budget [DFLP26a, Lemma 13], is isochronous [HPRR20], and is the same in every configuration, treated as ideal. Only the ApproxExp input $\lfloor 2^{63} \cdot x \rfloor$ overflows 32.64 and should read the fraction bits (Section 5).

The basis and the signature are integers, but the target and the tree are real numbers, computed in the FFT domain, with magnitudes from about 2^{-19} (the reciprocals of Section 3.4) to 2^{24} (the transformed hash of Section 3.1). The security argument needs the $2n$ pairs (t_i, σ'_i) handed to `SamplerZ` to be accurate. Theorem 2 of [DFLP26a] turns the RMSE of those pairs into the number of signatures per key, Q_s, that the proof covers (Section 4.1). We reason where errors enter, and which techniques remove them.

2.2 Floating-Point and Fixed-Point Arithmetic

The IEEE 754 binary64 format has a 53-bit significand, and every operation rounds with relative error at most 2^{-53}. The Cortex-M4F floating-point unit handles only single precision, so binary64 is emulated with integer instructions in constant time [Por19; Por25a], and we call this emulation "ieee" from here on. Triple-word (tw) arithmetic [HVS+26] is the only arithmetic using the M4F FPU, representing each value as three single-precision words with a rounding error well below ieee at roughly twice the cost.

A fixed-point value is $X \cdot 2^{-f}$ with X a two's-complement integer. A format "$i.f$" has i integer bits including the sign and f fractional bits, and we abbreviate "fixed-point" to "fxp" from here on. A Type 1 format uses one format for every value, a Type 2 format uses one word length but a per-variable fraction length fixed at compile time [DFLP26a] (Figure 2a). We write m for the integer budget of a value a, meaning $|a| < 2^m$. A Type 1 format with a signed integer part needs $m+1$ integer bits to hold such a value, so a budget of $m = 21$ calls for a 22.64 format. These budgets are provable: [DFLP26a] add threshold checks to key generation that bound every intermediate value (Section 3.3). The minimal Type 1 format of [DFLP26a] is 22.64, their C code uses 64.64 [DAB26], and we use 32.64, which they suggest for 32-bit architectures, in our implementation fxp32.64.

One unit in the last place (ULP) of 32.64 is 2^{-64} regardless of the value. For values $|x| \geq 1$ this gives exactly 64 fractional bits against binary64's 52 (53 with the implicit 1),

but for values much smaller than 1 this format wastes leading zero bits that a floating-point exponent would shift away. Such values hold fewer significant bits, and a multiplication by a larger value amplifies the loss (Section 3.4). The consequence for the evaluation is that fxp rounding errors are absolute and do not depend on the key's magnitudes, while binary64 errors scale with them (Figure 2b–c, equal at $|x| = 2^{-12}$; below, binary64 is more precise). The flip side is range, the dotted line at 2^{31}: without an exponent, a two's-complement overflow wraps silently, so the integer budgets must be proven.

Adding or subtracting two $i.f$ values with the same format is an exact integer operation on the underlying X and needs no rounding. Multiplication of two values with f fractional bits produces $2f$ fractional bits, which are rounded once back to f. On 32.64, this means 128 fractional bits rounded to 64, with an error of at most 2^{-65}, or $2^{-64}/\sqrt{12} = 2^{-65.8}$ in RMSE if uniform over one ULP. We use two rounding rules: round to nearest with ties to even, the IEEE 754 default, and round to nearest with ties up, $\lfloor (XY + 2^{f-1})/2^f \rfloor$ [DFLP26a]. Division and square root are replaced by Newton–Raphson iterations (Section 3.4).

3 Fixed-Point FN-DSA Signing

We implement a Type 1 fxp format, specifically 32.64, for Falcon. A 32.64 value is three 32-bit limbs, and a full 96×96-bit product is built from nine $32 \times 32 \to 64$-bit partial products. These 32 integer bits are plenty for both the split and non-split integer budgets (Table 1, $m \leq 31$), and as we focus on the 32-bit Cortex-M4, it is more efficient than a 64.64 format used by the C implementation from [DFLP26a; DAB26]. Two things can go wrong: a value too large for the integer budget wraps around silently (Section 2.2), and a value much smaller than 1 loses significant bits. Sections 3.1–3.3 keep every value in range: the target order avoids the overflow that Sign [PFH+20, Alg. 10] causes for non-split, the split [DFLP26a, Alg. 7] computes the final signature in integers, and we extend the bounds of [DFLP26a] to Falcon-1024. Section 3.4 recovers the bits lost to small values with wide internal formats, and Section 3.5 covers speed and memory.

3.1 Target Computation Order

In FFT domain, the target as described in Section 2.1 is obtained by

$$\hat{\mathbf{t}} \leftarrow \left(-\tfrac{1}{q}\,\hat{c} \odot \hat{F},\ \tfrac{1}{q}\,\hat{c} \odot \hat{f}\right).$$

The instruction order is not defined, which in fixed point arithmetic lets us control the dynamic range of intermediate values. Here, we distinguish three ordering options for the $\hat{F}$ product: *multiply-first* [PFH+20; Por25a; DAB26], *scale-first* (ours) and *coefficient-domain* [Pre26]. With $\|\hat{c}\|_\infty \leq n(q-1) < 2^{22.6}$ and the key checks of Section 3.3 ($\|\hat{F}\|_\infty < \gamma_{F,G} < 2^{11.8}$, $\|\hat{f}\|_\infty < \gamma_{f,g} < 2^8$ for Falcon-512), the highest m in each order is:

Order	Computation	m (512 / 1024)	32.64?
multiply-first	$-(\hat{c} \odot \hat{F})/q$	35 / 37	no (64.64)
scale-first	$(-\hat{c}/q) \odot \hat{F}$	23 / 24	yes
coefficient-domain	$\mathsf{FFT}(-c/q) \odot \hat{F}$	21 / 23	yes

Multiply-first overflows the 32.64 budget ($m \leq 31$), resulting in an infinite signing loop. Reordering is free as it requires the same number of operations. We denote *scale-first* as our baseline fxp32.64 implementation, which we later extend with options like the split. [DFLP26a, Table 4] gives $m(\hat{\mathbf{t}}) = 21$, so their suggested 22.64 format implicitly assumes coefficient-domain order for non-split. In a Type 2 implementation, coefficient-domain is a natural fit: each coefficient of c/q lies in $[0, 1)$, so Type 2 assigns $m = 0$ and gives the full fraction to the $1/q$ multiplication. *Coefficient-domain* saves two (Falcon-512)

or one (Falcon-1024) integer bits over *scale-first*, but both fit in 32.64 ($m \leq 31$), and *coefficient-domain* rounds c/q before the FFT, which *scale-first* keeps exact. Our precision analysis (Section 4.1) focuses on *scale-first*; on ieee, both orders give the same center error δ_t within 0.1 bit.

3.2 Target Split and Final Signature

According to Lemma 9 of [DFLP26a], shifting the center by a lattice vector shifts the sample by the same vector, so `ffSampling` only needs the fractional part of $\mathbf{t}$ and the integer part is added back afterwards. De Almeida Braga et al. [DFLP26a, Alg. 7] split the target as $\mathbf{t} = \mathbf{t}_{\text{int}} + \mathbf{t}_{\text{frac}}$ with $\mathbf{t}_{\text{int}} \in \mathbb{Z}^{2n}$ and $\|\mathbf{t}_{\text{frac}}\|_\infty \leq 1/2$. Because $q\,\mathbf{t} = (-cF, cf)$, the fractional part $q\,\mathbf{t}_{\text{frac}} = ((-cF) \bmod^{\pm} q,\ (cf) \bmod^{\pm} q)$, with $\bmod^{\pm}$ the centered remainder, comes from two NTT products mod q without floating- or fixed-point arithmetic. `ffSampling` runs on $\hat{\mathbf{t}}_{\text{frac}}$ and returns $\mathbf{z}'$, giving $\mathbf{z} = \mathbf{t}_{\text{int}} + \mathbf{z}'$ and $\mathbf{s} = (\mathbf{t}_{\text{frac}} - \mathbf{z}')\,\mathbf{B}$. This reduces $\|\hat{\mathbf{t}}_{\text{frac}}\|_\infty$ to at most $n/2$, from up to $n\,\gamma_{F,G}$ for the non-split $\hat{\mathbf{t}}$, improving the m budget and the precision (Section 4.1). The non-split case is the specification's original Sign algorithm [PFH+20, Alg. 10].

In the non-split configuration, the final signature can be computed without the target, as $\mathbf{s} = (\mathbf{t} - \mathbf{z})\mathbf{B} = (c, 0) - \mathbf{z}\mathbf{B}$, fully in integer domain. For the split configuration, the fractional or the integer part of the target is required. The most straightforward method is to store $\mathbf{t}_{\text{frac}}$ during signature computation, and use $\mathbf{s} = (\mathbf{t}_{\text{frac}} - \mathbf{z}')\mathbf{B}$, requiring the FFT, as $\mathbf{t}_{\text{frac}}$ is not integer [DFLP26a]. Storing $\mathbf{t}_{\text{frac}}$ throughout `ffSampling` requires $24n$ additional bytes in our 32.64 format. Alternatively, recomputation of $\mathbf{t}_{\text{frac}}$ requires both the NTT and the FFT on top of the final signature computation. We describe two methods that, as in the non-split configuration, compute $\mathbf{s}$ fully in the integer domain after only the inverse FFT of $\mathbf{z}'$; our implementation uses the first.

The first method, from concurrent work by Pornin [Por26a], relies on reduction over a small prime $q' = 18\,433$ instead of recovering $\mathbf{t}_{\text{int}}$, using $q^{-1} \equiv 3 \pmod{q'}$ to turn $q\,\mathbf{t}_{\text{frac}}$ into $\mathbf{t}_{\text{frac}} \bmod q'$. The part s_2 is computed mod q'. The other part is recomputed as $s_1 = c - s_2 h \bmod q$ with a norm check in place for the verification computation. This can cause a restart, but never results in an invalid signature. Every accepted signature has $\|s_2\|_\infty \leq \beta < q'/2$, with β the signature bound of Falcon [PFH+20] ($\beta \approx 8\,382$ for Falcon-1024), so a 15-bit prime suffices without any coefficient wrap. This method is 1.5 / 1.6 % faster compared to that on auxiliary prime and halves the NTT table flash; our implementation uses it by default.

The second method computes the signature from $\mathbf{t}_{\text{int}}$ by use of NTT, given by $\mathbf{s} = (c, 0) - (\mathbf{t}_{\text{int}} + \mathbf{z}')\mathbf{B}$ [DFLP26a, Lemma 9]. Writing the Euclidean division of the integer products by q coefficient-wise as $(cF)[i] = q\,k_0[i] + r_0[i]$, with centered remainders $r_j[i]$, yields $\mathbf{t}_{\text{int}} = (-k_0,\, k_1)$ with split residues $q\,\mathbf{t}_{\text{frac}} = (-r_0,\, r_1)$. The remainders are already available from the NTT products mod q that the norm check requires so that only quotients $k_j[i]$ are missing. We recompute the products modulo an auxiliary prime $p \equiv 1 \pmod{2n}$ using the NTT, giving residues $a_j[i]$. As $q\,k_j[i] \equiv a_j[i] - r_j[i] \pmod{p}$, the quotient is $(a_j[i] - r_j[i])\,q^{-1} \bmod p$, exact whenever $p > 2\,\|\mathbf{t}_{\text{int}}\|_\infty$. With $\|F\|_\infty, \|f\|_\infty \leq 127$ and $c \in [0, q-1]$, each coefficient of cF sums n terms, resulting in $\|\mathbf{t}_{\text{int}}\|_\infty \leq 130\,037$ for Falcon-1024. As $2 \cdot 130\,037 > 2^{16}$, halfword packing is not possible. We thus choose $p = 8\,380\,417$, the 23-bit ML-DSA prime [Nat24a], whose lazy signed-Montgomery NTT needs at most one reduction pass.

3.3 Falcon-1024 Bounds

De Almeida Braga et al. [DFLP26a, Alg. 2] add four threshold checks to Falcon's key generation, restarting when any check fails. First, $\gamma_{f,g}$ bounds the FFT coefficients of the short key polynomials ($\|\hat{f}, \hat{g}\|_\infty < \gamma_{f,g}$). Second, γ_{hybrid} bounds α_{hybrid}: in every FFT slot,

$|\hat{f}|^2 + |\hat{g}|^2$ stays within a factor γ^2_{hybrid} of q [DFLP26a, Eq. 7], originating as the trapdoor quality in Mitaka [EFG+22]. Both run before `NTRUSolve`. After `NTRUSolve`, $\gamma_{F,G}$ bounds the FFT coefficients of the large key polynomials ($\|\hat{F}, \hat{G}\|_\infty < \gamma_{F,G}$) as third check, and, finally, γ_{root} bounds the root value of the LDL* decomposition ($\|\hat{L}_{10}\|_\infty < \gamma_{\text{root}}$). Existing tests ensure that $\alpha_{\text{GPV}} \leq 1.17$, $\mathsf{NTT}(f)$ has no zero entry, and `NTRUSolve` does not fail.

Every intermediate value of key generation and signing is bounded by a function of the four checked quantities [DFLP26a], giving each variable an m budget (Table 1). Besides the key polynomials and $\hat{\mathbf{t}}, \hat{\mathbf{z}}$, these variables are the Gram matrix $\hat{\mathbf{G}} = \hat{\mathbf{B}}\hat{\mathbf{B}}^*$ with entries $\hat{G}_{00} = |\hat{g}|^2 + |\hat{f}|^2$, $\hat{G}_{10} = \hat{G}\,\hat{g}^* + \hat{F}\,\hat{f}^*$, and $\hat{G}_{11} = |\hat{G}|^2 + |\hat{F}|^2$ ($\hat{G}$ without indices is the key polynomial), and its LDL* decomposition [PFH+20, Alg. 8],

$$\hat{L}_{10} = \frac{\hat{G}_{10}}{\hat{G}_{00}}, \qquad \hat{D}_{00} = \hat{G}_{00}, \qquad \hat{D}_{11} = \hat{G}_{11} - \frac{|\hat{G}_{10}|^2}{\hat{G}_{00}}. \tag{1}$$

The ffLDL* algorithm [PFH+20, Alg. 9] halves the degree at each level down to degree 2, building the Falcon tree: every inner node holds its own $\hat{L}_{10}$, and the n leaves are the real values $\sigma'_j = \sigma/\sqrt{\hat{D}_{jj}}$, where σ is Falcon's degree-dependent standard deviation, each serving two `SamplerZ` calls. Independently of thresholds, $\|\hat{L}_{10}\|_\infty < 1$ holds below the root since Gram matrices have equal diagonals and are positive definite. By Lemma 8 of [DFLP26a], every diagonal $\hat{D}_{jj}$ of the tree satisfies

$$\frac{q}{\gamma^2_{\text{hybrid}}} \;\leq\; \hat{D}_{jj} \;\leq\; q\,\gamma^2_{\text{hybrid}}, \tag{2}$$

including $\hat{G}_{00} = \hat{D}_{00}$ and $\hat{D}_{11}$ at the root. The checks bound only FFT outputs, but this also bounds every FFT intermediate value: read backwards, the FFT is an inverse FFT, whose stages never increase the largest absolute value [DFLP26a].

De Almeida Braga et al. [DFLP26a] derived and evaluated the thresholds for Falcon-512 only, $(\gamma_{f,g}, \gamma_{\text{hybrid}}, \gamma_{F,G}, \gamma_{\text{root}}) = (255, 4, 3500, 24)$. However, if used as such for Falcon-1024, only 8.5 % of keys would be accepted. Our goal was to minimize the integer budget increase for Falcon-1024 while keeping acceptance comparable to the Falcon-512 thresholds.

Table 1 shows the m budget of each variable for the Falcon-512 thresholds and for the same thresholds at Falcon-1024. With probability of at least $1 - 2^{-\lambda}$, $\hat{\mathbf{t}}, \hat{\mathbf{z}}$ budgets follow from Eq. (3) [DFLP26a, Eq. 23]. Here, B depends on the split or non-split case and σ, λ on the security level [PFH+20]:

$$\begin{aligned} &\|\hat{\mathbf{t}}\|_\infty \leq B + 2\,A_{\text{child}}\,\sqrt{n}\,\big(\gamma_{\text{root}} + \sqrt{2} + 1\big), \\ &A_{\text{child}} = \frac{2\,\tau\,\sigma\,\gamma_{\text{hybrid}}}{\sqrt{q}}, \qquad \tau = \sqrt{2\big(\lambda + \log_2(2n\ln n) + 1\big)\ln 2}. \end{aligned} \tag{3}$$

Note that $\hat{\mathbf{t}}, \hat{\mathbf{z}}$ budgets grow even with the same bounds since n doubles in B and $\sqrt{n}$. The slightly larger bound on $\hat{\mathbf{z}}$ [DFLP26a, Eq. 24] still fits in the same budgets as $\hat{\mathbf{t}}$.

Increasing the threshold, leads to acceptance of more keys. The border of a threshold is the largest value at which every variable it bounds still fits its m budget. One step beyond costs an integer bit, so a threshold raised past its border is best raised all the way to the next one, so thresholds are chosen at borders. $\gamma_{f,g}$, $\gamma_{F,G}$, and γ_{root} directly set the integer range for their variable, so their border is a power of two. γ_{hybrid} has no budget of its own, but enters through $\hat{D}_{jj}$: $q \cdot \gamma^2_{\text{hybrid}} < 2^m$. $\gamma_{F,G}$ and γ_{root} also enter the $\hat{\mathbf{t}}, \hat{\mathbf{z}}$ budget from Eq. (3), where $\gamma_{F,G}$ sets B and γ_{root} and γ_{hybrid} appear in the second term, so these three share one border. This shared border explains the Falcon-512 thresholds: $B = 512 \cdot 3500 \cdot (q-1)/q \approx 2^{20.77}$, and the second term with $\gamma_{\text{root}} = 24$ brings the bound close to 2^{21}. Together, 3500 and 24 nearly exhaust the $m = 21$ budget.

Table 1: Variable m budgets and how they relate to the rejection thresholds. Both columns use the Falcon-512 thresholds from [DFLP26a].

Variable	Bound	m_{512}	m_{1024}
$\hat{f}, \hat{g}$	$\|\hat{f}\|_\infty, \|\hat{g}\|_\infty < \gamma_{f,g}$	8	8
$\hat{F}, \hat{G}$	$\|\hat{F}\|_\infty, \|\hat{G}\|_\infty < \gamma_{F,G}$	12	12
$\hat{G}_{00}$ (root)‡	$\|\hat{G}_{00}\|_\infty \le \|\hat{f}\|^2 + \|\hat{g}\|^2 < 2\,\gamma_{f,g}^2$	18	18
$\hat{G}_{10}$ (root)	$\|\hat{G}_{10}\|_\infty < 2\,\gamma_{f,g}\,\gamma_{F,G}$	21	21
$\hat{L}_{10}$ (root)	$\|\hat{L}_{10}\|_\infty < \gamma_{\text{root}}$	5	5
$\hat{D}_{jj}$	$\|\hat{D}_{jj}\|_\infty \le q\,\gamma_{\text{hybrid}}^2$	18	18
$\hat{L}_{10}$ (inner)†	$\|\hat{L}_{10}\|_\infty \le \dfrac{\alpha_{\text{hybrid}}^4 - 1}{\alpha_{\text{hybrid}}^4 + 1} < 1$	0	0
$\hat{\mathbf{t}}, \hat{\mathbf{z}}$ (split)	Eq. (3), $B = n/2$	18	19
$\hat{\mathbf{t}}, \hat{\mathbf{z}}$ (non-split)	Eq. (3), $B = n\,\gamma_{F,G}\,(q-1)/q$	21	22

† Independent of the thresholds. ‡ The bound gives $m = 17$, aligned to $\hat{D}_{jj}$ as in [DFLP26a].

Table 2: Threshold tuples, joint acceptance over unfiltered key generation, attempts per accepted key, and m budgets. Superscripts mark a threshold at its border and the budget it raises first. The $\hat{\mathbf{t}}, \hat{\mathbf{z}}$ column lists split / non-split; $\hat{G}_{10}$ follows from $\hat{f}, \hat{g}$ and $\hat{F}, \hat{G}$.

	Thresholds				Accepted	Attempts	m budgets				
Tuple	$\gamma_{f,g}$	γ_{hybrid}	$\gamma_{F,G}$	γ_{root}	keys	per key	$\hat{f}, \hat{g}$	$\hat{D}_{jj}$	$\hat{F}, \hat{G}$	$\hat{G}_{10}$	$\hat{\mathbf{t}}, \hat{\mathbf{z}}$
512, unfiltered	–	–	–	–	100 %	10.6	–	–	–	–	–
512 [DFLP26a]	255	4	3500	24	66.0 %	16.0	8	18	12	21	18 / 21
1024, unfiltered	–	–	–	–	100 %	16.6	–	–	–	–	–
1024, same	255	4	3500	24	8.5 %	196.4	8	18	12	21	19 / 22
1024, A	256^a	5.39	7510^c	32^c	67.4 %	24.6	8	19	13	22	20 / 23
1024, B	256^a	6.53^b	7365^c	32^c	72.3 %	23.0	8	19	13	22	20 / 23
1024, C	512^{ab}	6.53^b	7365^c	32^c	98.8 %	16.8	9	19	13	23	20 / 23

^{a} $m(\hat{f}, \hat{g})$ ^{b} $m(\hat{D}_{jj})$ ^{c} $m(\hat{\mathbf{t}}, \hat{\mathbf{z}})$

To evaluate the candidates of Table 2, we ran 10^5 key generation instances at $n = 1024$ with all four checks disabled (unfiltered) and replayed each tuple on this set, so acceptance is the joint rate over all four checks with their correlations. An attempt is one `NTRUGen` iteration, counting rejections by the existing tests. The Falcon-512 row was replayed the same way on 10^5 keys at $n = 512$, hence its 66.0 % instead of a value from [DFLP26a].

Tuples A and B have the same m budgets, spending the shared border in two ways. Tuple B sets $\gamma_{f,g} = 256$, keeping $m(\hat{f}, \hat{g}) = 8$, and $\gamma_{\text{hybrid}} = \sqrt{2^{19}/q} \approx 6.53$, the largest value keeping $m(\hat{D}_{jj}) = 19$. $\gamma_{\text{root}} = 32$ keeps $m(\hat{L}_{10}) = 5$, and $\gamma_{F,G} = 7365$ follows from the shared border. Because 256 and 32 sit exactly at a power-of-two budget ceiling ($\le 2^k$ would need $m = k + 1$), we check all thresholds with strict $< \gamma$. De Almeida Braga et al. [DFLP26a] accepts $\le \gamma$, the tighter check rejects no additional keys in practice. Tuple A spends the border on $\gamma_{F,G}$ instead: with $\gamma_{\text{hybrid}}^2 = 29$, $\gamma_{F,G} = 7510$ is the largest integer below the non-integer $\hat{\mathbf{t}}, \hat{\mathbf{z}}$ border. Tuple B accepts 5 % more: γ_{hybrid} rejects many keys while $\gamma_{F,G}$ above ∼7000 almost none. Tuple C spends one more bit on $m(\hat{f}, \hat{g})$ and $m(\hat{G}_{10})$, buying 98.8 % acceptance at 16.8 attempts per key, mostly the same as unfiltered key generation at 16.6. Tuple C also gets B's γ_{hybrid} at no cost: with $\gamma_{f,g} = 512$ the root diagonal $\hat{G}_{00} < 2\,\gamma_{f,g}^2 = 2^{19}$ already needs $m(\hat{D}_{jj}) = 19$ (Table 1, ‡), the budget that $\gamma_{\text{hybrid}} = 6.53$ fills. We choose tuple B, because it has the lowest m budget increase with a reasonable acceptance rate: every budget increases by one over the Falcon-512 values except $m(\hat{f}, \hat{g})$, which stays at 8. Its 72.3 % acceptance (1.38× unfiltered key generation)

Table 3: Rejections per check of [DFLP26a, Alg. 2] while generating 1000 Falcon-1024 keys with tuple B. The last three checks are after `NTRUSolve`.

Check	$\|\hat{f},\hat{g}\|_\infty$	α_{hybrid}	α_{GPV}	$\mathsf{NTT}(f)$	$(F,G)\neq\perp$	$\|\hat{F},\hat{G}\|_\infty$	$\|\hat{L}_{10}\|_\infty$
Tuple B	7 950	3 832	10 091	87	8	0	3

is even better than Falcon-512's 66.0 % (1.51×). Its highest budgets are $m = 23$ for the non-split target and $m = 24$ for $\hat{c}$ (Section 3.1), both fitting 32.64. Without Symplecroot (Section 3.4), the generic LDL* also computes $\hat{G}_{11} < 2\,\gamma^2_{F,G}$ ($m = 27$), which still fits.

Table 3 breaks down the rejections per 1000 accepted keys for tuple B, led by the existing α_{GPV} test. The two new checks that fire most, $\gamma_{f,g}$ and γ_{hybrid}, are evaluated before `NTRUSolve`. Each early reject costs only a polynomial generation and a few comparisons, with negligible impact on key generation performance. The $\gamma_{F,G}$ and γ_{root} checks fire rarely, because most keys that would fail them are already caught by the earlier thresholds. A failure after `NTRUSolve` is expensive, but the table shows it almost never happens.

Mitaka [EFG+22] and BAT [FKPY22] derive n-dependent thresholds for the checks before `NTRUSolve`. Mitaka's trapdoor quality is α_{hybrid}, and BAT bounds the norms of (f, g) and (F, G), though (F, G) is given as a function of f, g. We reproduced our Table 3 rejection rates for $\gamma_{f,g}$ and γ_{hybrid} using their tail formulas. These formulas are not effective at retrieving the bounds for Falcon, as they do not take our m budgets into account.

3.4 Higher Precision Operations

On the absolute scale of fixed point (Section 2.2), values much smaller than 1 lose significant bits. The reciprocals in the LDL* decomposition have denominators up to $q\,\gamma^2_{\text{hybrid}}$ by Eq. (2), $2^{17.6}$ for Falcon-512, giving values as low as $2^{-17.6}$. In fxp32.64, such a value keeps about 46 significant bits out of 64 fractional, a relative error of $2^{-47.4}$ instead of 2^{-65}, and once stored the loss enters every product. We avoid that by keeping reciprocals, inverse square roots and small constants in a wide internal format, rounding once at the consumer's output. Following [DFLP26a], we also remove a cancellation at the LDL* root with a direct computation.

Symplecroot The subtraction $\hat{D}_{11} = \hat{G}_{11} - |\hat{G}_{10}|^2/\hat{G}_{00}$ in Eq. (1) cancels most of the magnitude, amplifying the rounding error of its operands. The NTRU identity $fG - gF = q$ implies $\hat{G}_{00}\hat{G}_{11} - |\hat{G}_{10}|^2 = q^2$ in every slot, so at the root $\hat{D}_{11} = q^2/\hat{G}_{00}$, which can be computed directly. The identity holds only at the root, as the children's determinants are not constant; we call this *Symplecroot*. Extending the idea to inner nodes gives *Symplectree*, which we found to give the same precision at a higher cost (Section 4.1) [SZZ+22; DFLP26a].

Fused Reciprocal In Falcon, division computes $\hat{L}_{10} = \hat{G}_{10}/\hat{G}_{00}$ at every LDL* node and $q^2/\hat{G}_{00}$ at the Symplecroot root, with denominators bounded by Eq. (2).

Following [DFLP26a], we replace division by a Newton–Raphson reciprocal followed by a multiplication. The divisor is normalized as $y = 2^k d$ with $d \in [1/2, 1)$. The iteration

$$r_{t+1} = r_t(2 - dr_t)$$

squares the relative error at each step ($e_{t+1} = e_t^2$). The seed $r_0(d) = 48/17 - (32/17)\,d$ has relative error at most $1/17$, giving 4.09 correct bits on $[1/2, 1)$, which double to 65 after four iterations ($4.1 \to 8.2 \to 16.3 \to 32.7 \to 65$). The first two run on 32-bit words against the top limb, one 32×32-bit multiply per product. The third runs on the upper 64 bits, four 32×32-bit partial products. At each stage the word width exceeds the correct bits by at least 14, so the next iteration squares the truncation error away. The fourth uses the

complete 96-bit divisor and its nine partial products, squaring away all prior truncations. The result is kept as an unrounded 126-bit fractional value, *without* rounding it to fxp32.64, with a relative error of at most $2^{-65.3}$. The reciprocal is computed once per FFT slot at each LDL* node and shared across its consumers (real and imaginary parts of $\hat{L}_{10}$, and at the Symplecroot root $q^2/\hat{G}_{00}$); each consumer multiplies the wide reciprocal by its own numerator and rounds the product once. Because the reciprocal is never stored outside registers, this does not increase stack usage. For the non-root $\hat{L}_{10}$, with $\|\hat{L}_{10}\|_\infty < 1$, this results in an absolute error of at most $2^{-65} + 2^{-65.3} < 2^{-64}$, instead of up to $2^{-47.4}$ for Falcon-512 when the reciprocal is rounded first, and $q^2/\hat{G}_{00}$ shows the same improvement.

Fused Square Root The sampler consumes each leaf $x = \hat{D}_{jj}$ as the scale factor $\sqrt{x}/\sigma = 1/\sigma'_j$ (Section 3.3), which spares `SamplerZ` a division. Without fusion, this requires two roundings: one for $\sqrt{x} = x \cdot (1/\sqrt{x})$, and one for the multiplication by $1/\sigma$. The constant $1/\sigma < 1$ ($\approx 0.006 \approx 2^{-7.4}$ for Falcon-512) would lose about 7 significant bits if stored in the 32.64 format. We fold the entire chain into a single product $x \cdot (1/\sqrt{x}) \cdot (1/\sigma)$, where the unrounded inverse square root is multiplied directly by x, and $1/\sigma$ is stored in a normalized 71-bit fractional representation with the full 64 significant bits, rounded once at the output. The precision gain comes from the wide constant: its actual relative error goes from $2^{-59.0}$ to $2^{-64.9}$ for Falcon-512 ($2^{-58.4}$ to $2^{-66.2}$ for Falcon-1024), while rounding $\sqrt{x}$ in between would add only $2^{-65}/\sigma < 2^{-72}$. The resulting absolute leaf error is at most $2^{-64.1}$ instead of $2^{-59.3}$ for Falcon-512, and $2^{-64.6}$ instead of $2^{-58.8}$ for Falcon-1024.

Following [DFLP26a], the inverse square root $1/\sqrt{x}$ is computed by the iteration

$$u_{t+1} = \frac{u_t}{2}\left(3 - a\,u_t^2\right),$$

where x is normalized to $a = 2^{2k}x \in [1/4, 1)$; the bound in Eq. (2) ensures positivity. Convergence is quadratic, but a factor 3/2 in the error recurrence costs about half a bit per step. The seed $u_0 = 2.2119 - \frac{4}{3}a$ has about 3 correct bits and reaches 80 after five iterations, the early ones on narrower words as for the reciprocal; the fifth produces 94 fractional bits on the full 96-bit value.

Wide $1/q$ The constant $1/q \approx 2^{-13.6}$ keeps only 50.4 significant bits in 32.64, a relative error of $2^{-51.5}$. It multiplies FFT slots up to $n(q-1) \approx 2^{22.6}$, amplifying the storage error into the target (Section 3.1). We store $1/q$ as $\lfloor 2^{77}/q \rceil$, a normalized 64-bit word with relative error $2^{-65.1}$, gaining $+5.4$ / $+4.2$ bits of δ_t at Falcon-512 / 1024 (Table 4). Each multiplication is one 96×64-bit product rounded once, cheaper than a 96×96-bit multiply. It is applied after the FFT of the residues on the split path and in the scale-first multiply on the non-split path, at no cycle cost.

3.5 Speed and Memory Improvements

The remaining improvements are mostly fxp-specific, targeting speed and memory rather than precision. We measured them to not influence the final error values, besides the integer Gram and FFT-new, which are discussed in Section 4.1.

$\hat{L}_{10}$ in 64 Bits At the non-root nodes, $\|\hat{L}_{10}\|_\infty < 1$, so the 32 integer bits of 32.64 are wasted. Storing $\hat{L}_{10}$ in 64 bits (1 sign, 63 fractional) loses one fractional bit, raising its error from at most 2^{-65} to 2^{-64}, or from $2^{-65.8}$ to $2^{-64.8}$ in RMSE (Section 2.2), still 5.5 bits below the center error $\delta_t = 2^{-59.3}$ of the recommended configuration (Section 4.1). Without the fused reciprocal (Section 3.4), the error of $\hat{L}_{10}$ would far exceed the loss of one fractional bit. This saves 2 / 4 KB of signing memory with faster 96×64 multiplies.

Single-Rounding Complex Multiplication Rounding is about 30 % of each fxp multiplication. In a complex multiply $(ac - bd, ad + bc)$, each output has 2 multiplications and 1 addition; the two products run without rounding and only the combined result is rounded, giving an error of at most 2^{-65} instead of 2^{-64} with 2 roundings instead of 4, or $2^{-65.8}$ instead of $2^{-65.3}$ in RMSE, as independent roundings add in squares. We measured no effect, as both are at least 6 bits below δ_t.

FFT Twiddle Table Storage The FFT twiddles are strictly smaller than 1 in absolute value, so 64 bits instead of 96 suffice, cutting the memory cost of the table by a third at faster 96×64 multiplications. Unlike $\hat{L}_{10}$, we do not want to lose a fractional bit, as the twiddles enter every FFT, inverse FFT and split/merge. We keep the full 64 fractional bits, bit-identical to the 96-bit format, by storing the twiddles unsigned: sine entries are always positive, cosine entries negative exactly at odd indices, and a butterfly absorbs all negations, including the inverse-FFT conjugation, by swapping its operands. The index parity is public, so this optimization is constant-time and saves flash and cycles.

Rounding Mode [DFLP26a] use ties to even; the alternative, ties up, has the same worst case error (2^{-65}) and RMSE ($2^{-65.8}$) but a per-rounding bias of $\Pr[\text{tie}] \cdot \mathbb{E}[e \mid \text{tie}] = 2^{-64} \cdot 2^{-65} = 2^{-129}$, assuming uniform discarded bits. In practice, the fractional bits are not fully uniform, so ties can be more frequent, but the bias stays far below the RMSE of $2^{-65.8}$. Ties up is faster as it skips the parity check; because of the very low bias the precision loss is negligible, and we measured no effect.

Integer Gram and FFT-new Pornin [Por26a] computes $\mathbf{G} = \mathbf{B}\mathbf{B}^*$ exactly in integers via the CRT, transforms only G_{00} and G_{10} to the FFT domain, and rebuilds $\hat{G}_{11} = q^2/\hat{G}_{00} + |\hat{G}_{10}|^2/\hat{G}_{00}$ from the LDL*'s own reciprocal. For split, the NTT operations are shared with the target computation, saving FFT operations; for non-split the NTTs come on top, with a minor performance loss. On ieee without Symplecroot this gains sampler precision from the exact $\mathbf{G}$; on fxp32.64 it matches Symplecroot and costs 0.5 bits once the reciprocal is fused and $1/q$ is wide. [Por26a] also suggests FFT-new for these transforms of integer polynomials, the basis, or the Gram entries: it splits the input into self-adjoint parts, so that the first stages are exact integer additions and every later butterfly is one real multiplication by a recombination constant $R_j = 1/(2\cos\theta_j)$ and an add–sub, instead of a complex twiddle multiplication. These constants grow large where $\cos\theta_j$ is small, up to $R_{\max} = 81$ for Falcon-512 (163 for Falcon-1024), amplifying rounding errors and costing precision on both arithmetics (Section 4.1). Unlike for the classic FFT, the intermediate values of FFT-new can exceed its output (non-unit R_j), but stay at most $2n$ times the largest input coefficient. For the largest input, $\|G_{10}\|_\infty \le 4144 \cdot 2^7 \,/\, 5822 \cdot 2^7$ at Falcon-512 / 1024 [Por26a], this gives $2^{29.0} \,/\, 2^{30.5}$, so $m = 30 \,/\, 31$, which fits 32.64.

4 Evaluation

We evaluate the implementation choices for precision and for speed in order to find the best trade-offs. Section 4.1 shows how each configuration affects the errors at the Gaussian sampler and the resulting Q_s. Section 4.2 presents cycle counts and recommended configurations. Both ieee and fxp32.64 implement the same `fpr` interface of c-fn-dsa so that every configuration runs identical signing code and the two arithmetics are compared at equal structure. The baseline configuration is non-split fxp32.64 or ieee with no additional techniques from Section 3.

4.1 Precision Evaluation

This section defines the error metric and how it gives Q_s, locates the error sources (Table 5), describes what remains in each arithmetic, and compares fixed point with binary64.

Theorem 2 of [DFLP26a] replaces the worst-case error analysis in the specification [PFH+20; Pre17] with an RMSE-based argument at the `SamplerZ` inputs. The two inputs of each `SamplerZ` call are a center t_i, whose error δ_t is absolute, and a standard deviation σ'_i, whose error δ_σ is relative. Theorem 2 turns their RMSE into Q_s. Writing $\bar{\sigma}'_i$ and $\bar{t}_{i,j}$ for the values an implementation computes, the per-key RMSE values are

$$\delta_\sigma = \sqrt{\frac{1}{2n}\sum_{i\in[2n]}\left(\frac{\bar{\sigma}'_i}{\sigma'_i}-1\right)^2}, \qquad \delta_t = \sqrt{\frac{1}{2nQ_s}\sum_{j}\sum_{i\in[2n]}\left|\bar{t}_{i,j}-t_{i,j}\right|^2}\ \text{[DFLP26a]}, \tag{4}$$

where i runs over the $2n$ calls of one `ffSampling` execution and j over the executions, restarts included. δ_σ depends only on the key, while δ_t depends on the message history. Theorem 2 of [DFLP26a] additionally requires a worst-case bound $\delta_\infty \geq \max_i |\bar{\sigma}'_i/\sigma'_i - 1|$ with $(2\lambda-1)\,\delta_\infty \leq 1/8$, which every configuration meets by more than 28 bits. It bounds the number of signing queries Q_s for which λ-bit UF-CMA security keeps at least $\lambda-2$ bits as a function of δ_σ, δ_t, and the restart rate c. Theorem 2 needs its bounds for every key, so we report the worst sampled key.

The precision analysis of [HVS+26] was re-implemented to measure the RMSE. For δ_σ we used 10^5 keys, storing the largest $|\bar{\sigma}'_i/\sigma'_i - 1|$ for δ_∞. For δ_t we used 10^5 keys with 100 signatures each. As δ_t depends on the messages, 100 signatures only estimate its per-key value, so for five configurations of Table 4 we re-ran the keys with the worst and the median δ_t using 10^6 signatures each, which changed their exponent $-\log_2 \delta_t$ by at most 0.02 bit (fxp32.64) and 0.12 bit (ieee). Unless stated otherwise, all results in this section come from the first two runs. Every error is taken against a 200-bit reference computed with MPFR [FHL+07] that follows the same control flow as the evaluated code. The harness runs the portable C operations, which the Cortex-M4 assembly reproduces identically in per-operation tests and test vectors on the board. The keys come from the same seeds for every configuration, so the rows are paired, and all keys pass the γ checks of Section 3.3.

Theorem 2 counts $N = 2nQ_s(1+c)$ sampler calls. Inverting its two conditions, without their $1+O(\cdot)$ factors, gives the number of signatures per key the proof supports,

$$Q_s \leq \frac{1}{4\,\alpha\, n\,(1+c)\,\delta_t^2} \qquad \text{and} \qquad Q_s \leq \frac{1}{24\,\alpha\, n\,(1+c)\,\delta_\sigma^2}, \qquad \alpha = 2\lambda - 1, \tag{5}$$

with $\lambda = 128$ for Falcon-512 and $\lambda = 256$ for Falcon-1024. Both bounds must hold, and the δ_t bound is always the binding one: every δ_t in Table 4 is at least 4 bits worse than the corresponding δ_σ, and the smaller constant of the δ_t bound (4 against 24) makes up only 1.3 of these bits. Table 4 lists the exponent $x = -\log_2 \delta$ of the error values. Filling in α and n in the δ_t bound gives, for Falcon-512 $Q_s \leq 2^{2x}/(4\cdot 255\cdot 512) \approx 2^{2x-19}$, and for Falcon-1024 $Q_s \leq 2^{2x}/(4\cdot 511\cdot 1024) \approx 2^{2x-21}$. We measured the restart rate c at most 10^{-3} in our runs, changing the bound by a negligible factor of $2^{0.002}$. From this, $Q_s \geq 2^{64}$ requires $x \geq 41.5\,/\,42.5$ for Falcon-512 / 1024, and every bit of δ_t is worth two bits of Q_s. A Q_s below 2^{64} means that Theorem 2 does not cover 2^{64} queries for that configuration with the measured errors, not that an attack exists. A Q_s above 2^{64} covers more queries than the scheme supports, as the specification sizes the salt, σ and `SamplerZ`, which Theorem 2 treats as ideal, for 2^{64} queries [PFH+20]. The part above 2^{64} is a margin for unsampled keys with larger errors and the $1+O(\cdot)$ factors dropped in Eq. (5).

Where Precision Is Lost The two inputs come from two places (Figure 1): the centers from the target $\mathbf{t} = (-cF/q,\ cf/q)$ of the hash c (Section 2.1), and the standard deviations

Table 4: Sampler errors $-\log_2 \delta$ (larger is better) at the worst of 10^5 keys, Falcon-512 / Falcon-1024. A "+" row adds one technique to the row above. Shaded: $Q_s < 2^{64}$. From Eq. (5), $Q_s \geq 2^{64}$ needs $-\log_2 \delta_t \geq 41.5 / 42.5$.

Configuration	ieee		fxp32.64	
	$-\log_2(\delta_t)$	$-\log_2(\delta_\sigma)$	$-\log_2(\delta_t)$	$-\log_2(\delta_\sigma)$
baseline (non-split)	40.3 / 39.8	46.3 / 45.6	39.1 / 37.8	43.4 / 42.5
Symplecroot	40.3 / 39.8	52.7 / 52.7	45.3 / 45.0	52.6 / 51.2
split	41.9 / 40.8	46.3 / 45.6	39.1 / 37.8	43.4 / 42.5
split, int. Gram	43.7 / 43.1	48.3 / 48.0	45.6 / 45.2	52.6 / 51.2
split, Symplecroot	47.5 / 47.0	52.7 / 52.7	45.6 / 45.2	52.6 / 51.2
+ int. Gram	47.3 / 46.8	52.1 / 51.9	45.6 / 45.2	52.6 / 51.2
+ FFT-new	43.0 / 41.7	48.4 / 47.5	45.6 / 45.2	52.6 / 51.2
non-split, fused reciprocal	—	—	45.9 / 45.9	59.0 / 57.9
split, fused reciprocal	—	—	50.8 / 50.3	59.0 / 57.9
+ fused sqrt	—	—	50.8 / 50.3	60.5 / 59.1
+ wide $1/q$	—	—	56.2 / 54.5	60.5 / 59.1
+ Symplecroot	—	—	59.8 / 59.3	64.1 / 64.3
+ int. Gram (recommended)	—	—	59.3 / 58.9	64.2 / 64.3
non-split, fused recip. / sqrt, wide $1/q$, Symplecroot	—	—	52.4 / 51.8	64.1 / 64.3
recommended, FFT-new	—	—	55.7 / 54.4	61.2 / 58.6

Table 5: Error sources, the technique that removes or adds each, and its gain in $-\log_2 \delta$ at the worst of 10^5 keys, Falcon-512; negative gains are losses.

Source	Enters	Size	Technique	ieee δ_t	ieee δ_σ	fxp32.64 δ_t	fxp32.64 δ_σ
Small values in fixed point							
reciprocal rounding	root, inner	rel. $2^{-47.4}$	fused reciprocal	—	—	+11.7	+15.6
$1/q$ rounding	target	rel. $2^{-51.5}$	wide $1/q$	—	—	+5.4	0.0
$1/\sigma$ rounding	leaves	rel. $2^{-59.0}$	fused sqrt	—	—	0.0	+1.5
Error amplifiers							
root cancellation	root	$\times$ up to $2^{14.4}$	Symplecroot	+5.6	+6.4	+3.6	+3.6
FFT-new constants	Gram matrix	$\times$ up to 41	FFT-new	−4.3	−3.7	−3.6	−3.0
Large operands							
non-split target	target	up to $2^{20.8}$	split	+7.2	0.0	+7.4	0.0
integer Gram FFT	Gram matrix	up to $2\gamma_{f,g}^2 \approx 2^{17}$	int. Gram	−0.2	−0.6	−0.5	+0.1

from the Falcon tree, built from the key's Gram matrix. Each tree node divides by $\hat{G}_{00}$ to get $\hat{L}_{10} = \hat{G}_{10}/\hat{G}_{00}$ and $\hat{D}_{11}$ (Eq. (1)). At the root $\hat{D}_{11} = q^2/\hat{G}_{00}$, and the leaves are $\sigma'_j = \sigma/\sqrt{\hat{D}_{jj}}$. After each subtree, `ffSampling` corrects the remaining target with the node's $\hat{L}_{10}$ so that an error near the root reaches every later center, while one at a leaf affects only that leaf's σ'_j. Precision is lost in three ways. *Small values in fixed point:* a value much smaller than 1 keeps few significant bits on the absolute grid (Figure 2c), for instance $1/q \approx 2^{-13.6}$ keeps 50.4 of 64. *Error amplifiers:* an operation of the algorithm that scales errors already present, such as a subtraction of two nearly equal values or a multiplication by a large constant. *Large operands:* key or message values that are large: binary64's absolute error grows with them (Figure 2b), and in fixed point they scale the rounding error of a rounded operand.

The error of each row in Table 4 is set by the largest remaining source for that configuration, and each technique removes only one source, possibly shifting the largest error to a different point. A technique therefore shows its gain only once it removes the largest source, so gains depend on the order. Table 5 lists the sources grouped by these

three kinds, with the stage of Figure 1 where each error enters. Its "size" column is a relative error (rel.), an amplification factor (×), or an operand magnitude, and each gain is the Table 4 difference at the step where that source is the largest. For example, the $1/q$ rounding is a small-value source, enters the target, has relative error $2^{-51.5}$, and the wide $1/q$ removes it, gaining +5.4 in δ_t and 0 in δ_σ, because σ' never uses $1/q$. As the δ_t bound always sets Q_s, the text discusses δ_σ only where its source differs from δ_t's. All values below are for Falcon-512; Falcon-1024 is within two bits unless stated.

Small Values in Fixed Point These three sources exist only in fixed point. The largest baseline source, the reciprocal rounding of $1/\hat{G}_{00}$, sits in every tree node and moves both δ_t and δ_σ, while $1/q$ enters only the target (δ_t) and $1/\sigma$ only the leaves (δ_σ). Rounded to a 32.64 value, the reciprocal keeps as few as 46 of its 64 significant bits, a relative error up to $2^{-47.4}$ (Section 3.4), which enters every $\hat{L}_{10} = \hat{G}_{10}/\hat{G}_{00}$. In the baseline, this error, further amplified by the root cancellation (see below), sets both δ_t and δ_σ. The fused reciprocal removes it, giving the largest gains of Table 5: +15.6 bits of δ_σ (43.4 → 59.0) and +11.7 of δ_t with the split (39.1 → 50.8), where the $1/q$ rounding then sets δ_t. Without the split, the $1/q$ rounding floor is lower (45.9, a gain of +6.8), as the rounded $\hat{c}/q$ is also multiplied by $\hat{F}$ and $\hat{f}$. The wide $1/q$ of Section 3.4 removes that floor, giving +5.4 bits (50.8 → 56.2). After the fused reciprocal, δ_σ is set by the $1/\sigma$ rounding: every row that still has the unfused sqrt reads 59.0, regardless of Symplecroot or the split. The fused sqrt with the wide $1/\sigma$ constant gives +1.5 without Symplecroot and +5.0 with it, as Symplecroot's remaining share of δ_σ becomes visible only once the sqrt is fused. With these sources left in, the best fxp32.64 δ_t is 45.6, behind the recommended ieee row (47.3); without them it is 59.3.

Error Amplifiers At the root, the generic LDL* computes $\hat{D}_{11} = \hat{G}_{11} - |\hat{G}_{10}|^2/\hat{G}_{00}$ (Eq. (1)), whose true value is $q^2/\hat{G}_{00}$ by the NTRU identity. As both terms share the denominator $\hat{G}_{00}$, the subtracted term is $\kappa = |\hat{G}_{10}|^2/q^2$ times larger than the result, so the subtraction cancels almost all of the operands' magnitude and turns a relative error ε in either operand into about $\kappa\varepsilon$ in $\hat{D}_{11}$, in both arithmetics. κ depends on the key and the slot; the key checks of Section 3.3 bound it by $(2\gamma_{f,g}\gamma_{F,G}/q)^2 \approx 2^{14.4}$ for Falcon-512 ($2^{16.5}$ for Falcon-1024). In the baseline, this amplified error sets both fxp32.64 errors (39.1 for δ_t, 43.4 for δ_σ), but on ieee only δ_σ (46.3), as the non-split target sets δ_t (see below).

Symplecroot exploits the NTRU identity and computes $\hat{D}_{11} = q^2/\hat{G}_{00}$ directly, without the subtraction. On the fxp32.64 baseline, it gives +6.2 bits of δ_t and +9.2 of δ_σ. On ieee, it gives +6.4 of δ_σ but no δ_t on the baseline, and +5.6 of δ_t once the split has reduced the target error. After the fused reciprocal and the wide $1/q$, where the cancellation is again the largest fxp32.64 source, Symplecroot gives Table 5's +3.6 bits in both δ_t and δ_σ: the gain is smaller because the error it amplifies is already smaller. The cancellation depends on the key: while the $1/q$ rounding dominates, the per-key spread (median minus worst key) is only 0.18 bits, the wide $1/q$ grows it to 3.1 bits, and Symplecroot shrinks it to 0.36, as Symplecroot gains +3.6 at the worst key but only +0.9 at the median key.

On fxp32.64, the integer Gram achieves the same result as Symplecroot. The κ amplification requires the two operands to carry independent errors, but the integer Gram builds $\hat{G}_{11}$ from $\hat{G}_{00}$ and $\hat{G}_{10}$ using the same reciprocal and the same products that the LDL* will use (Section 3.5). Hence, the operands share all their errors and the subtraction removes them instead of amplifying them. In fixed point, addition and subtraction are exact so that $\hat{D}_{11}$ reduces to $q^2/\hat{G}_{00}$ at a non-amplified reciprocal error. In binary64, rounding errors during intermediate additions fail to fully cancel; as a result, integer Gram recovers only +1.8 bits of δ_t, compared to +5.6 for Symplecroot.

FFT-new (Section 3.5) trades speed for amplification. Its largest constant $R_{\max}$ acts in the last stage which also halves its outputs. Thus, a rounding error from any earlier

stage is amplified by up to $R_{\max}/2 = 41$ (81 for Falcon-1024), compared to ≤ 1 for the unit-modulus twiddles of the classic FFT. In binary64, rounding errors scale with the rounded value, so the larger constants also produce proportionally larger errors. In fxp32.64, each rounding is absolute at 2^{-64} but amplification gives a per-coefficient error floor of $2^{-64} \cdot 41 \approx 2^{-58.6}$. FFT-new comes for free while δ_t is still $2^{-50.8}$ or worse (Table 4). This larger error hides the $2^{-58.6}$ floor, which only shows once the wide $1/q$ is added.

On the recommended ieee row, FFT-new takes 4.3 bits of δ_t at the worst key (47.3 → 43.0), about 2 at the median key, and 3.7 of δ_σ. At Falcon-1024, it costs 5.1 bits and puts the worst key at $Q_s = 2^{62.3}$, below 2^{64}. The amplified error enters the Gram entries before the LDL* and sets a floor that no way of computing $\hat{D}_{11}$ at the root repairs: the generic LDL*, Symplecroot and Symplectree all give 43.0 on the integer Gram. The signer of Pornin [Por26a] uses Symplecroot with FFT-new, so we expect a similar loss. On the recommended fxp32.64 row, FFT-new costs 3.6 bits of δ_t (59.3 → 55.7, $Q_s = 2^{92.4}$) and 3.0 of δ_σ (5.7 at Falcon-1024). In exchange, FFT-new saves 4.2 / 4.3 % of cycles on ieee, but only 1.5 / 1.6 % on fxp32.64 for +9.0 KB of flash (Section 4.2).

Large Operands Without the split, the target is the product of the rounded $\hat{c}/q$ with $\hat{F}$ and $\hat{f}$ (Section 3.1). A rounded factor such as a twiddle adds an error proportional to the value it multiplies, so $\hat{c}$, up to $2^{22.6}$, leaves its FFT with errors of up to $2^{-42.4}$ per butterfly in fxp32.64; the $1/q$ scales value and error down together, but the product with $\hat{F}$, up to $\gamma_{F,G} < 2^{11.8}$, scales the error of $\hat{c}/q$ up again, and binary64 also rounds the product at its own magnitude of up to $2^{20.8}$. With the split (Section 3.2), $\hat{F}$ and $\hat{f}$ multiply c only in the exact integer NTT, and no large factor follows a rounding in the real domain. On ieee, the split gives +1.6 on the baseline, where the root cancellation then sets δ_t, and +7.2 once Symplecroot has removed the cancellation. On fxp32.64, it gives nothing on the baseline, where the reciprocal rounding dominates, and +7.4 with the precise operations of Section 3.4 and Symplecroot (Table 5). Paired per key, the median gain on the recommended fxp32.64 configuration is +4.4 bits. The non-split fxp32.64 row has 2.8 bits of per-key spread, as $|\hat{F}|$ and $|\hat{f}|$ vary with the key.

The integer Gram's FFT of the large entries introduces rounding errors that scale with the largest entry, not with each slot's own value, costing up to 1.9 bits in the slots with the smallest $\hat{G}_{00}$. Only keys with such slots pay, and the integer Gram is taken for speed (−5.6 % of ieee cycles, −0.5 bits and −2.6 % on fxp32.64). Reading Table 5's fxp32.64 δ_t column along the chain of Table 4 from the split row gives the recommended value: $39.1 + 11.7 + 5.4 + 3.6 - 0.5 = 59.3$, and for ieee $41.9 + 5.6 - 0.2 = 47.3$.

Opportunities for Further Investigation In each arithmetic, δ_σ is close to the limitation of the respective format. `SamplerZ` receives $1/\sigma'_j \in [1/\sigma_{\max}, 1/\sigma_{\min}] \approx [0.55, 0.78]$ [PFH+20], for which one rounding (Section 2.2) has a relative RMSE of $2^{-53.9}$ to $2^{-54.4}$ on ieee and $2^{-64.9}$ to $2^{-65.4}$ on fxp32.64. The recommended δ_σ sits about 2 bits short of that on ieee ($2^{-52.1}$) and 1 bit short on fxp32.64 ($2^{-64.2}$). For δ_t, one rounding of a value near 1 has an RMSE of $2^{-64}/\sqrt{12} = 2^{-65.8}$ on fxp32.64 and $2^{-52}/\sqrt{12} = 2^{-53.8}$ on binary64. The measured values fall about 6.5 bits short: 59.3 against 65.8 on fxp32.64, 47.3 against 53.8 on ieee (at Falcon-1024 both are 6.9–7.0 bits short). The 12-bit lead of fxp32.64 is the 12-bit difference of the two formats near 1 (Figure 2b). The same factor in both arithmetics is consistent with an amplifier in the shared signing code, not in either format. A possible amplifier, which we did not further investigate, is the target update at the root: it multiplies by $\hat{L}_{10}$ with $|\hat{L}_{10}|$ up to $\gamma_{\text{root}} = 24$ (32 at Falcon-1024), about $2^{4.6}$ (2^5), while below the root $|\hat{L}_{10}| < 1$. We left this open, as at 59.3 the recommended configuration is already 17.8 bits of δ_t above the 2^{64} requirement (Q_s margin 35.7 bits).

Table 6: Cycles per call on the Cortex-M4. "mul · reciprocal" is the multiply consuming the unrounded reciprocal; "sqrt · $1/\sigma$" is the sampler leaf (ieee: sqrt then multiply).

	add	add–sub	mul	complex mul	reciprocal	mul · reciprocal	sqrt · $1/\sigma$	FFT, $n = 512$
ieee	70	84	34	336	461	34	646	620 k
fxp32.64	3	6	26	182	90	89	219	217 k
ratio	0.04	0.07	0.76	0.54	0.20	2.6	0.34	0.35

Fixed Point against Binary64 The baseline fxp32.64 build reaches $Q_s = 2^{59.2} / 2^{54.5}$, and the recommended configuration $2^{99.7} / 2^{96.7}$ ($2^{85.9} / 2^{82.7}$ without the split). The ieee baseline reaches $Q_s = 2^{61.5} / 2^{58.7}$, and the recommended ieee row (split, Symplecroot, int. Gram) $2^{75.7} / 2^{72.6}$ while being 2.6 / 2.5 % faster (Section 4.2). At equal structure, the recommended fxp32.64 row leads ieee by about 12 bits in both δ_t and δ_σ. The values that matter are near 1, where 32.64 has 12 more significant bits than binary64 (Figure 2b). Its one weakness, small values, sits in three places that the precise operations of Section 3.4 remove. Binary64's weakness covers every large value and has no local fix. The split and Symplecroot give similar gains on both arithmetics and are representation-independent.

The per-key spread differs between the two arithmetics. The ieee baseline worst key is 2.7 bits below its median (40.3 against 43.0), the recommended fxp32.64 row only 0.34 bits: binary64 errors scale with the key-dependent magnitudes of $\hat{G}_{00}$, fixed-point rounding is absolute. The fxp worst-key value of the recommended row is therefore robust to the key sample, while the ieee one is not. Our port of the triple-word arithmetic of [HVS+26] onto the same signing code takes about twice the ieee cycles; the split, the integer Gram and Symplecroot make it faster as they do ieee, and improve its δ_t the same way from a worst-key starting point of $2^{-59.8}$ (10^4 keys), where it needs no improvement. The precision figures of [DFLP26a] are not directly comparable, as they report one level above the leaves, one key, and a 64-bit Type-2 Python implementation.

Implementation Choices Not Affecting Sampler Errors The following choices, measured on 10^5 keys with 10 signatures each, do not change the errors in Table 4 beyond its printed precision of 0.1 bit. Ties-up rounding saves 2.2 / 2.3 % of cycles (Table 7). Storing $\hat{L}_{10}$ in 64 bits saves 2 / 4 KB of signing memory. The single-rounding complex multiply is precision-neutral. Running the early Newton iterations on narrower words (Section 3.4) saves 1.7 % of cycles. The unsigned 64-bit twiddle table is bit-identical to the 96-bit format and saves 3.1 / 3.3 % of cycles. Symplectree equals Symplecroot within 0.15 bit on both arithmetics but costs +0.5 / +0.6 %, and is more complex, so we use Symplecroot.

4.2 Performance

All cycle counts are medians on a NUCLEO-L4R5ZI (Cortex-M4F, 20 MHz) under pqm4, using the DWT cycle counter. The ieee baseline rows run c-fn-dsa, unchanged except for the Keccak, and are faster than its published M4 figures [Por25a], so the 45 % lead of the recommended fxp32.64 row is conservative. Since [Por25a], c-fn-dsa optimized its `fpr` emulation and signing code (scratch $78n \to 59n$ bytes): its older version in pqm4 [KRSS19] signs Falcon-512 in 22.28 M cycles on the same board, against 17.45 M now with the same Keccak. We measured a restart rate of 0.1 % at Falcon-1024; restarts double a signature's cost but move the benchmark mean by only about 0.1 %. All rows except the first use the Keccak of Huang et al. [HAZ+24], shipped with pqm4, on both arithmetics; the "original SHAKE" row uses c-fn-dsa's own Keccak, for comparison with measurements that use it.

Table 6 lists the per-operation cycle costs. Addition gains by a factor of 23× because fixed-point addition is one 96-bit wide integer add, with no exponent handling. Addition,

subtraction, add–sub, negation and halving take only a few instructions each, so we inline them. All other assembly routines use a custom calling convention, with operands in fixed registers and no register saving, similar to the `fpr_add_sub` of c-fn-dsa [Por25a]. The multiplication reveals the weakest point of the format: a 96×96-bit product needs nine 32×32-bit partial products at a ratio of only 0.76. This hints at a 64-bit format, especially on platforms without a single-cycle $32 \times 32 \to 64$-bit multiplier. The reciprocal (0.20) and the sampler leaf (0.34) gain from the algorithm: the emulated binary64 division and square root compute one result bit per iteration (55 and 54 iterations) to get the correctly rounded result, while the kernels of Section 3.4 use Newton–Raphson iterations on the single-cycle 32×32-bit multiplier and round only once, at the output of the consuming multiply. The sampler leaf additionally folds the multiplication by $1/\sigma$ into the same product. The one operation where fxp32.64 is slower is the multiply consuming the unrounded reciprocal (89 against 34, ratio 2.6); this is a deliberate trade, spending cycles in the consumer to keep the reciprocal unrounded, and the reciprocal and its consumers combined are still faster than on ieee. At the polynomial level, the classic FFT takes only 0.35 of the ieee cycles, less than a single multiply (0.76) or complex multiply (0.54), because fxp32.64 combines the complex multiply and the add–sub of each butterfly in one assembly routine, with a single rounding per output, saving 6 % of signing cycles. A fused butterfly would help ieee less, as its butterfly is dominated by the emulated arithmetic, which needs nearly all registers, so operands cannot stay in registers between operations. In portable C, without assembly, fxp32.64 signs in 28.93 / 62.36 M cycles, against 40.63 / 88.11 M for ieee at equal structure, still about 29 % faster. On 64-bit hosts, scalar C code for fxp32.64 showed no gain over the 64.64 format from [DFLP26a]; vectorized implementations were not evaluated.

Memory, Verification and Key Generation The fxp32.64 signing scratch is $83n$ bytes (41.5 / 83.0 KB for Falcon-512 / 1024), with $\hat{L}_{10}$ stored in 64 bits (Section 3.5), against $59n$ (29.5 / 59.0 KB) for ieee; measured with pqm4, the peak sign stack including this scratch is 45.9 / 87.7 KB, against 31.3 / 60.8 KB for ieee. The +41 % overhead is the price of the 12-byte word, not of fixed point as such. A 64-bit word would restore ieee's footprint, but requires a Type 2 error evaluation, likely using techniques similar to our unrounded reciprocal and sqrt; the $20n$ layout of [Por26a] also reduces both.

Verification touches no `fpr` value and is identical on both arithmetics. With the faster Keccak it takes 0.326 / 0.646 M cycles, down from 0.407 / 0.799 M. It costs about 4 KB of flash: a verify-only build grows from 11.4 to 15.5 KB (+36 %), a signing build from 47.2 to 51.2 KB (+9 %). In a combined sign and verify build, sign has already paid for the Keccak; for a verify-only device such as a bootloader, the original Keccak avoids the flash cost. The γ checks of Section 3.3 need 1.51× the key generation attempts at Falcon-512 and 1.38× at Falcon-1024. Most rejections come from $\gamma_{f,g}$ and γ_{hybrid}, which depend only on f and g and are checked before `NTRUSolve`, so the cycle cost is well below those factors. Over 200 key generations, the checks raise the mean from 75.9 to 89.2 M cycles at Falcon-512 (+18 %) and from 298.8 to 339.7 M cycles at Falcon-1024 (+14 %). Key generation, including the γ checks, runs in its own 32.32 fixed point [Por23] and does not depend on the signing arithmetic, so this holds for both ieee and fxp32.64.

Constant Time For signing in every configuration of Table 7, on 100 keys × 100 signatures per degree, TIMECOP [Nei19] finds no branch or memory address depending on the private key or signing randomness in host builds at gcc `-O3` and `-O0`, except values public by construction: the salt, the public key, key validity, the signature, restarts, and the accept decisions of the isochronous `SamplerZ` [HPRR20]. TIMECOP runs on the host and sees neither our assembly nor branches the Cortex-M4 compiler may introduce. A scan of the disassembled Cortex-M4 images shows that each branch is a C condition checked by TIMECOP or tests a public counter, length or parameter, and that signing executes no

Table 7: Sign cycle medians on the Cortex-M4, 100 keys × 100 signatures. "vs. baseline" is the ieee baseline row. Highlighted: $Q_s < 2^{64}$ at the worst key of 10^5. The fxp rows other than "unfused reciprocal" store $\hat{L}_{10}$ in 64 bits, which is precision-neutral (Section 4.1).

Configuration	Sign (Mcycles)		vs. baseline		$\log_2 Q_s$	
	512	1024	512	1024	512	1024
ieee						
baseline, original SHAKE	17.45	37.44	+4.7 %	+4.3 %	61.5	58.7
baseline (non-split)	16.66	35.91	—	—	61.5	58.7
baseline, FFT-new	15.50	33.27	−7.0 %	−7.3 %	59.2	54.3
split, int. Gram	16.51	35.58	−0.9 %	−0.9 %	68.3	65.2
+ Symplecroot	16.22	35.01	−2.6 %	−2.5 %	75.7	72.6
+ FFT-new	15.54	33.49	−6.7 %	−6.7 %	67.1	62.3
fxp32.64						
split, fused recip. / sqrt, wide $1/q$, Symplecroot, int. Gram	9.38	19.95	−43.7 %	−44.4 %	99.7	96.7
unfused reciprocal	9.83	20.95	−41.0 %	−41.7 %	72.2	69.5
ties-up (recommended)	9.17	19.49	−44.9 %	−45.7 %	99.7	96.7
ties-up, FFT-new	9.04	19.18	−45.8 %	−46.6 %	92.4	87.8
non-split, fused recip. / sqrt, wide $1/q$, Symplecroot	9.11	19.31	−45.3 %	−46.2 %	85.9	82.7
ties-up	8.88	18.81	−46.7 %	−47.6 %	85.9	82.7
ties-up, FFT-new	8.80	18.58	−47.2 %	−48.3 %	84.2	79.8

division, the only Cortex-M4 arithmetic instruction whose timing depends on its operands (2 to 12 cycles [Arm20; BBB+25]), not even through compiler-runtime calls. On the board, each `fpr` operation takes the same number of cycles for 10^4 random and boundary inputs, and each FFT-domain routine for 100 random polynomials.

Recommended Configurations Table 7 lists the cycle counts and Q_s per configuration. The recommended fxp32.64 build uses the split, the fused reciprocal and sqrt, the wide $1/q$, Symplecroot, the integer Gram, $\hat{L}_{10}$ in 64 bits, and ties-up rounding. Every technique except the split is faster than the build without it, and the split's 3 % of cycles buys about 14 bits of Q_s. The error is nearly key-independent (0.34 / 0.24 bit spread at 10^5 keys), so the worst-key Q_s is robust to the key sample. We also see that fused reciprocal is essential: compared with the same build, leaving it out costs +5 % cycles and −27 bits of Q_s.

The non-split configuration with ties-up (8.88 / 18.81 M) is the fastest and smallest option (−6 KB flash from the NTT tables), at $Q_s = 2^{85.9} / 2^{82.7}$, about 20 bits above the 2^{64} target. Its 2.8 bit per-key spread makes that figure less certain than the recommended row's. FFT-new on the recommended row gives 9.04 / 19.18 M at $Q_s = 2^{92.4} / 2^{87.8}$, costing +9.0 KB (+17.6 %) of sign flash for −1.5 / −1.6 % of cycles, a bad trade unless flash is not a constraint. On binary64, the ieee baseline falls below $Q_s = 2^{64}$ at the worst key. The recommended ieee row (split, Symplecroot, int. Gram) is both 2.6 / 2.5 % faster and 14 bits of Q_s better than the baseline. FFT-new on top saves another 4.2 / 4.3 % but drops Falcon-1024 to $Q_s = 2^{62.3}$, below the 2^{64} target. At equal structure, fxp32.64 is about 44 % faster than ieee and +24 bits of Q_s better. On fxp32.64, speed and precision do not trade off: every technique that improves precision is also faster, and only the split costs cycles. This advantage holds on cores without a double-precision FPU, which emulate binary64.

Comparison with c-fn-dsa-alt On our board, with the Keccak of [HAZ+24], c-fn-dsa-alt [Por26a] (alt) takes 12.93 / 27.58 M cycles, and our recommended fxp32.64 needs 29 % fewer cycles, at about 4× its signing RAM ($83n$ against $20n$). Alt is faster than c-fn-dsa because it performs fewer fpr operations: a restructured `ffSampling`, which adds the target only at the leaves, and the integer Gram, which our recommended row already

uses, need fewer FFTs, and FFT-new makes each FFT cheaper. On fxp32.64, where fpr operations are already cheap (Table 6), the same reduction saves less: FFT-new, for instance, saves 4.2 % of cycles on ieee but only 1.5 % on fxp32.64 (Table 7). We therefore do not expect alt's structure to speed up fxp32.64 as much as it speeds up ieee. [Por26a] argues that, to sign within a deadline with probability $1 - 10^{-9}$, Falcon-512 must account for three attempts (40 M) against 78 iterations of ML-DSA-44 [Nat24a] (about 52 M); with our recommended fxp32.64, three attempts take 27.5 M, strengthening that case.

Several of our techniques apply to any Type 1 fixed-point Falcon signer, alt included (Sections 3.4 and 3.5). The fused reciprocal is the essential one: +11.7 bits of δ_t (Table 4) and −5 % of cycles (Table 7, unfused-reciprocal row). The fused sqrt with the wide $1/\sigma$ constant and ties-up rounding also apply directly, but not $\hat{L}_{10}$ in 64 bits: alt stores $L/(1+X)$ instead, which is not bounded by 1. More generally, the m budgets of [DFLP26a] (Table 1) cover the `ffSampling` of [PFH+20], which alt restructures to save memory, so its budgets and precision need a new evaluation. The parts we use from alt stay outside `ffSampling` and are easy to bound: the integer Gram and the signature modulo q' are exact, and FFT-new needs at most $m = 31$ (Section 3.5). Combining fxp32.64 with alt's structure is future work, as [Por26a] also proposes.

5 Suggestions for FIPS 206

This section provides input and recommendations for a fixed-point FN-DSA specification based on the 64.64 arithmetic of [DFLP26a]. This includes (a) what must be pinned for equivalence across word sizes, (b) which operation chain to pin, (c) key generation bounds for Falcon-1024, and (d) what exact KATs can test.

Word-Size Independence Equivalence between word sizes requires pinning the scale-first target order of Section 3.1. The reference code, c-fn-dsa, and fixed-point-c [DAB26] multiply first, which overflows 32 integer bits at both degrees. With scale-first and the γ checks, every value fits in 32 integer bits (Table 1). ApproxExp's input $\lfloor 2^{63} \cdot x \rfloor$, formed as a product in the reference and c-fn-dsa, needs 63 integer bits and wraps silently in 32.64. Defining it as the top 63 fraction bits of x [DFLP26a, Alg. 11] avoids this. With the same rounding, the Type 1 formats 32.64 and 64.64 then give identical bits, so the specification should pin the fractional bits and the proven integer ranges rather than a word size, letting an embedded target use 12-byte words instead of 16. We suggest ties-up rounding, which is precision-neutral and saves 2.2 / 2.3 % of signing cycles (Table 7).

Pinning a Precise Chain Our fxp32.64 precision rows are 64.64 rows for the same chain, because both have 64 fraction bits and nothing overflows, so every rounding decision is identical. Table 4 therefore applies to a 64.64 specification too. An absolute scale loses relative precision on small values, such as reciprocals and $1/q$, and the fix belongs in the chain, not the format. According to Table 4 and Eq. (5), the baseline structure yields $Q_s = 2^{59.2} / 2^{54.5}$, below 2^{64} and below binary64. Adding Symplecroot gives $2^{71.5} / 2^{69.1}$. Adding the one-rounding division and $\sqrt{x} \cdot 1/\sigma$ of Section 3.4 with the normalized $1/q$ and $1/\sigma$ constants gives $2^{85.9} / 2^{82.7}$. Adding the split gives $2^{99.7} / 2^{96.7}$ (Table 7). We recommend pinning Symplecroot or Symplectree, one-rounding division and $\sqrt{x} \cdot 1/\sigma$, and normalized small constants. The fused operations are faster than their unfused counterparts (Table 7) and each rounds once, bounding the error to one ULP for results below 1. They are not correctly rounded, so exact KATs must pin their Newton–Raphson seeds, iterations and truncations, reproducible on any word size, and the precision-neutral but bit-changing ties-up, $\hat{L}_{10}$ in 64 bits and single-rounding complex multiply; our recommended build is one such chain. Correctly rounded results would free these steps at the cost of a correction step. [DFLP26a] does not recommend the split, as it changes signing, makes

the reconstruction less straightforward, and their non-split chain already supports enough queries. But we recommend it: with the integer epilogue of Section 3.2 the reconstruction costs +3.0 / +3.3 % of cycles, and we measure +6.9 / +7.1 bits of δ_t from the target path (52.4 → 59.3 in Table 4), which no other technique of that table removes. The split also cuts the per-key spread from 2.8 to 0.34 / 0.24 bits, so the worst-key Q_s is robust to the key sample. Type 2 benefits twice: the 3 integer bits the split saves on $\hat{\mathbf{t}}, \hat{\mathbf{z}}$ (Table 2) become fraction bits, on top of the target-path gain, and opens lower-memory signing layouts [Por26a]. The split computes $\mathbf{t}_{\text{frac}}$ in the coefficient domain, which Type 2 favors over scale-first, as each coefficient then needs no integer bits (Section 3.1).

Key Generation [DFLP26a] give γ values only for Falcon-512, and the Falcon-1024 rows in fixed-point-c [DAB26] are placeholders. Table 2 gives our Falcon-1024 tuple. The γ checks cost +18 % key generation cycles at Falcon-512 and +14 % at Falcon-1024. The γ values fix every variable's integer budget (Table 1), which is what allows a 32.64 format and exactly what a Type 2 implementation needs. A key generation with these values admits a later Type 2 version with no further change. The validation of [BCCP26] checks the NTRU equation, the Gram–Schmidt norm, $\|(F, G)\|_\infty$ and the seed stream, but not γ. A key generator that skips the γ checks passes the validator, yet 32.64 relies on γ for its integer ranges (Table 1), so the validator or the signer at key load must check γ.

Exact KATs and the LTYZ25 Fixes Given the same key, message and randomness, two implementations diverge when the floor $\lfloor t_i \rfloor$ of the center in `SamplerZ` differs, changing every subsequent center through the target update, or when the rejection step [PFH+20, Alg. 14] reads a different number of random bytes. At the first two and last two `SamplerZ` calls, exact-integer centers make the floor a coin flip [LTYZ25], giving a structural divergence rate of about 1/7100 per signature [DFLP26a]. At all other calls, the centers have far larger denominators, making exact-integer centers negligible, and rounding errors in t_i and $1/\sigma'_i$ produce a noise class at both decision points, orders of magnitude lower. We measured the structural rate at 1/7000 to 1/7600 on 10^7 / 10^6 signatures at Falcon-512 / 1024, comparing binary64 and fxp32.64 centers against the 200-bit reference of Section 4.1, which gives a 1000-signature KAT set an 87 % chance of passing a binary64 signer against a pinned fixed-point chain. The LTYZ25 fixes [LTYZ25], rounding the center in `SamplerZ` instead of taking its floor and forcing $\|(g, -f)\|^2$ odd, remove the dominant structural class. We recommend adopting them: the divergence rate between two arithmetics drops from 1/7100 to the noise class, and randomized signing removes the same-randomness pairs that the LTYZ25 attack requires. KATs still cannot check precision. A center lies within an implementation's error of a half-integer with probability proportional to that error and then rounds up or down with the error sign: an implementation with too little precision still passes KATs while reducing the security level, and a sufficiently precise one still diverges with a small probability. Precision assurance must therefore come from analysis of the pinned chain. They are recommended in [DFLP26a] too, with divergence-free KATs as the alternative. This alternative generalizes to KATs that every sufficiently precise implementation matches, letting binary64 and fixed-point signers share one set.

6 Conclusion

We have presented the first fixed-point FN-DSA signer on a microcontroller, in fxp32.64 constant-time Cortex-M4 assembly for Falcon-512 and Falcon-1024. It needs 45 % fewer cycles than double-precision emulation (ieee) at $Q_s = 2^{99.7} / 2^{96.7}$, estimated from 10^5 keys, against $2^{61.5} / 2^{58.7}$ for ieee and the 2^{64} target. Signing RAM is $83n$ against $59n$ bytes for ieee; we need 29 % fewer cycles than c-fn-dsa-alt [Por26a] at about 4× its RAM.

Without our new operations, which keep reciprocals, inverse square roots and small constants wide and round once at the consumer, fxp32.64 is less precise than ieee at equal structure; with them it leads by about 24 bits of Q_s, and they are also faster. For ieee, the split, the integer Gram and Symplecroot raise Q_s to $2^{75.7}$ / $2^{72.6}$ while being faster. Each technique's precision depends on the rest of the chain and KATs cannot check precision, so a standard must evaluate its pinned chain as a whole (Section 5). Pinning the fraction bits and the proven integer ranges rather than a word size gives identical bits in 32.64 and 64.64, so our precision results hold for the 64.64 arithmetic that NIST plans. Our Falcon-1024 bounds accept 72.3 % of keys where the Falcon-512 values accept 8.5 %.

Combining fxp32.64 with the structure of c-fn-dsa-alt, a 64-bit Type 2 format, masking, and KATs that every sufficiently precise implementation matches, are future work.

References

[AAS+26] Muna Alshehhi, Ghada Alsuhli, Hani Saleh, Thanos Stouraitis, and Florian Caullery. "Design and implementation of a fixed-point FFT/IFFT processor for NTRU lattices". In: *Microprocessors and Microsystems* 122 (2026), p. 105282. ISSN: 0141-9331. DOI: 10.1016/j.micpro.2026.105282. URL: https://www.sciencedirect.com/science/article/pii/S0141933126000396.

[Arm20] Arm. *Cortex-M4 Processor Technical Reference Manual*. Revision r0p1, Issue 0001_04. 2020. URL: https://developer.arm.com/documentation/100166/0001/.

[BBB+25] Daniel J. Bernstein, Karthikeyan Bhargavan, Shivam Bhasin, Anupam Chattopadhyay, Tee Kiah Chia, Matthias J. Kannwischer, Franziskus Kiefer, Thales B. Paiva, Prasanna Ravi, and Goutam Tamvada. "KyberSlash: Exploiting secret-dependent division timings in Kyber implementations". In: *IACR TCHES* 2025.2 (2025), pp. 209–234. DOI: 10.46586/tches.v2025.i2.209-234.

[BCCP26] Maxime Bros, Christopher Celi, Pierre Ciadoux, and Ray Perlner. *Black-box validation of Falcon key generation under numerical instability*. Cryptology ePrint Archive, Paper 2026/954. 2026. URL: https://eprint.iacr.org/2026/954.

[BPT+24] Pierre-Augustin Berthet, Justine Paillet, Cédric Tavernier, Lilian Bossuet, and Brice Colombier. "Masked Computation of the Floor Function and Its Application to the FALCON Signature". In: *CiC* 1.4 (2024), p. 9. DOI: 10.62056/ay73zl7s.

[CC24] Keng-Yu Chen and Jiun-Peng Chen. "Masking Floating-Point Number Multiplication and Addition of Falcon First- and Higher-order Implementations and Evaluations". In: *IACR TCHES* 2024.2 (2024), pp. 276–303. DOI: 10.46586/tches.v2024.i2.276-303.

[CCCY26] Keng-Yu Chen, Ming Qing Ching, Jiun-Peng Chen, and Bo-Yin Yang. "When Masking Multiplication Isn't Enough: Exploiting Floating-Point Leakage in Falcon's Pre-Image Computation". In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2026.2 (Apr. 2026), pp. 686–708. DOI: 10.46586/tches.v2026.i2.686-708. URL: https://tches.iacr.org/index.php/TCHES/article/view/12905.

[DAB26] Daniel De Almeida Braga. *Fixed-Point C Prototype for Falcon – Type 1 Implementation*. Source code, commit `ad1e5bf`. 2026. URL: https://github.com/fixed-point-fndsa/fixed-point-c.

[DFLP26a] Daniel De Almeida Braga, Pierre-Alain Fouque, Bachir Lachguel, and Thomas Prest. *Toward a Secure Fixed-Point Implementation of the Falcon Signature Scheme*. Cryptology ePrint Archive, Paper 2026/1531. Extended version of [DFLP26b]. 2026. URL: https://eprint.iacr.org/2026/1531.

[DFLP26b] Daniel De Almeida Braga, Pierre-Alain Fouque, Bachir Lachguel, and Thomas Prest. "Toward a Secure Fixed-Point Implementation of the Falcon Signature Scheme". In: *Advances in Cryptology – CRYPTO 2026*. Ed. by Nadia Heninger and Mike Rosulek. Cham: Springer Nature Switzerland, 2026, pp. 166–197. ISBN: 978-3-032-35377-1.

[DN12] Léo Ducas and Phong Q. Nguyen. "Faster Gaussian Lattice Sampling Using Lazy Floating-Point Arithmetic". In: *ASIACRYPT 2012*. Ed. by Xiaoyun Wang and Kazue Sako. Vol. 7658. LNCS. Springer, Berlin, Heidelberg, Dec. 2012, pp. 415–432. DOI: 10.1007/978-3-642-34961-4_26.

[DP16] Léo Ducas and Thomas Prest. "Fast Fourier Orthogonalization". In: *Proceedings of the 2016 ACM International Symposium on Symbolic and Algebraic Computation*. ISSAC '16. Waterloo, ON, Canada: Association for Computing Machinery, 2016, pp. 191–198. ISBN: 9781450343800. DOI: 10.1145/2930889.2930923. URL: https://doi.org/10.1145/2930889.2930923.

[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: *EUROCRYPT 2022, Part III*. Ed. by Orr Dunkelman and Stefan Dziembowski. Vol. 13277. LNCS. Springer, Cham, May 2022, pp. 222–253. DOI: 10.1007/978-3-031-07082-2_9.

[FHL+07] Laurent Fousse, Guillaume Hanrot, Vincent Lefèvre, Patrick Pélissier, and Paul Zimmermann. "MPFR: A multiple-precision binary floating-point library with correct rounding". In: *ACM Trans. Math. Softw.* 33.2 (June 2007). ISSN: 0098-3500. DOI: 10.1145/1236463.1236468. URL: https://doi.org/10.1145/1236463.1236468.

[FKPY22] Pierre-Alain Fouque, Paul Kirchner, Thomas Pornin, and Yang Yu. "BAT: Small and Fast KEM over NTRU Lattices". In: *IACR TCHES* 2022.2 (2022), pp. 240–265. DOI: 10.46586/tches.v2022.i2.240-265.

[GEP+26] Radhika Garg, Daniel Escudero, Antigoni Polychroniadou, Akira Takahashi, and Xiao Wang. *Thresholdizing Standardized FALCON Signatures*. Cryptology ePrint Archive, Paper 2026/1300. 2026. URL: https://eprint.iacr.org/2026/1300.

[GMRR22] Morgane Guerreau, Ange Martinelli, Thomas Ricosset, and Mélissa Rossi. "The Hidden Parallelepiped Is Back Again: Power Analysis Attacks on Falcon". In: *IACR TCHES* 2022.3 (2022), pp. 141–164. DOI: 10.46586/tches.v2022.i3.141-164.

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. "Trapdoors for hard lattices and new cryptographic constructions". In: *40th ACM STOC*. Ed. by Richard E. Ladner and Cynthia Dwork. ACM Press, May 2008, pp. 197–206. DOI: 10.1145/1374376.1374407.

[HAZ+24] Junhao Huang, Alexandre Adomnicai, Jipeng Zhang, Wangchen Dai, Yao Liu, Ray C. C. Cheung, Çetin Kaya Koç, and Donglong Chen. "Revisiting Keccak and Dilithium Implementations on ARMv7-M". In: *IACR TCHES* 2024.2 (2024), pp. 1–24. DOI: 10.46586/tches.v2024.i2.1-24.

[HPRR20] James Howe, Thomas Prest, Thomas Ricosset, and Mélissa Rossi. "Isochronous Gaussian Sampling: From Inception to Implementation". In: *PQCrypto 2020*. Ed. by Jintai Ding and Jean-Pierre Tillich. Springer, Cham, 2020, pp. 53–71. DOI: 10.1007/978-3-030-44223-1_4.

[HPS98] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. "NTRU: A Ring-Based Public Key Cryptosystem". In: *Third Algorithmic Number Theory Symposium (ANTS)*. Vol. 1423. LNCS. Springer, June 1998, pp. 267–288.

[HVS+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". In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2026.2 (Apr. 2026), pp. 347–371. DOI: 10.46586/tches.v2026.i2.347-371. URL: https://tches.iacr.org/index.php/TCHES/article/view/12892.

[Hwa24] Vincent Hwang. "Formal Verification of Emulated Floating-Point Arithmetic in Falcon". In: *IWSEC 24*. Ed. by Kazuhiko Minematsu and Mamoru Mimura. Vol. 14977. LNCS. Springer, Singapore, Sept. 2024, pp. 125–141. DOI: 10.1007/978-981-97-7737-2_7.

[KA21] Emre Karabulut and Aydin Aysu. "FALCON Down: Breaking FALCON Post-Quantum Signature Scheme through Side-Channel Attacks". In: *2021 58th ACM/IEEE Design Automation Conference (DAC)*. 2021, pp. 691–696. DOI: 10.1109/DAC18074.2021.9586131.

[KHTA26] Hiroto Kaihara, Calvin Abou Haidar, Mehdi Tibouchi, and Masayuki Abe. *Square Root of All Evil: The Dangers of Falcon's Superfluous Square Roots*. Cryptology ePrint Archive, Paper 2026/2046. 2026. URL: https://eprint.iacr.org/2026/2046.

[KLL+26] GyuSang Kim, JeongHwan Lee, Myeonghoon Lee, Seokhie Hong, and HeeSeok Kim. "Secret Key Recovery of FALCON using Simple Power Analysis in Conditional Calculator". In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2026.2 (Apr. 2026), pp. 953–975. DOI: 10.46586/tches.v2026.i2.953-975. URL: https://tches.iacr.org/index.php/TCHES/article/view/12915.

[KRSS19] Matthias J. Kannwischer, Joost Rijneveld, Peter Schwabe, and Ko Stoffelen. *pqm4: Testing and Benchmarking NIST PQC on ARM Cortex-M4*. Cryptology ePrint Archive, Report 2019/844. 2019. URL: https://eprint.iacr.org/2019/844.

[LMDG25] Kang Li, Shouran Ma, Haochen Dou, and Qian Guo. *One Fell Swoop: A Single-Trace Key-Recovery Attack on the Falcon Signing Algorithm*. Cryptology ePrint Archive, Report 2025/2159. 2025. URL: https://eprint.iacr.org/2025/2159.

[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: *EUROCRYPT 2025, Part II*. Ed. by Serge Fehr and Pierre-Alain Fouque. Vol. 15602. LNCS. Springer, Cham, May 2025, pp. 213–244. DOI: 10.1007/978-3-031-91124-8_8.

[Nat24a] National Institute of Standards and Technology. *Module-Lattice-Based Digital Signature Standard (ML-DSA)*. FIPS 204. National Institute of Standards and Technology, 2024. URL: https://csrc.nist.gov/pubs/fips/204/final.

[Nat24b] National Institute of Standards and Technology. *Stateless Hash-Based Digital Signature Standard (SLH-DSA)*. FIPS 205. National Institute of Standards and Technology, 2024. URL: https://csrc.nist.gov/pubs/fips/205/final.

[Nei19] Moritz Neikes. *TIMECOP: Automated Dynamic Analysis for Timing Side-Channels*. 2019. URL: https://www.post-apocalyptic-crypto.org/timecop/.

[OSHG19] Tobias Oder, Julian Speith, Kira Höltgen, and Tim Güneysu. "Towards Practical Microcontroller Implementation of the Signature Scheme Falcon". In: *PQCrypto 2019*. Ed. by Jintai Ding and Rainer Steinwandt. Springer, Cham, 2019, pp. 65–80. DOI: 10.1007/978-3-030-25510-7_4.

[Per26] Ray Perlner. *New Plan for FN-DSA*. Post to the NIST pqc-forum mailing list, 28 September 2026. 2026. URL: https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/7z-y5qp-W2Y.

[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*. Tech. rep. Specification v1.2, 01/10/2020; round-3 submission to the NIST Post-Quantum Cryptography Standardization project. National Institute of Standards and Technology, 2020. URL: https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions.

[Por19] Thomas Pornin. *New Efficient, Constant-Time Implementations of Falcon*. Cryptology ePrint Archive, Report 2019/893. 2019. URL: https://eprint.iacr.org/2019/893.

[Por23] Thomas Pornin. *Improved Key Pair Generation for Falcon, BAT and Hawk*. Cryptology ePrint Archive, Report 2023/290. 2023. URL: https://eprint.iacr.org/2023/290.

[Por25a] Thomas Pornin. *Falcon on ARM Cortex-M4: an Update*. Cryptology ePrint Archive, Report 2025/123. 2025. URL: https://eprint.iacr.org/2025/123.

[Por25b] Thomas Pornin. *Improved (Again) Key Pair Generation for Falcon, BAT and Hawk*. Cryptology ePrint Archive, Report 2025/1239. 2025. URL: https://eprint.iacr.org/2025/1239.

[Por26a] Thomas Pornin. *A RAM-Efficient Implementation of Falcon*. Cryptology ePrint Archive, Paper 2026/1915. 2026. URL: https://eprint.iacr.org/2026/1915.

[Por26b] Thomas Pornin. *FN-DSA (in C)*. Source code, commit `3aa5048`. 2026. URL: https://github.com/pornin/c-fn-dsa.

[Pre17] Thomas Prest. "Sharper Bounds in Lattice-Based Cryptography Using the Rényi Divergence". In: *ASIACRYPT 2017, Part I*. Ed. by Tsuyoshi Takagi and Thomas Peyrin. Vol. 10624. LNCS. Springer, Cham, Dec. 2017, pp. 347–374. DOI: 10.1007/978-3-319-70694-8_13.

[Pre26] Thomas Prest. *Fixed-Point Python Prototype for Falcon*. Source code. 2026. URL: https://github.com/fixed-point-fndsa/fxp-falcon-py.

[Sho94] Peter W. Shor. "Algorithms for Quantum Computation: Discrete Logarithms and Factoring". In: *35th FOCS*. IEEE Computer Society Press, Nov. 1994, pp. 124–134. DOI: 10.1109/SFCS.1994.365700.

[SZZ+22] Shuo Sun, Yongbin Zhou, Rui Zhang, Yang Tao, Zehua Qiao, and Jingdian Ming. "Fast Fourier Orthogonalization over NTRU Lattices". In: *ICICS 22*. Ed. by Cristina Alcaraz, Liqun Chen, Shujun Li, and Pierangela Samarati. Vol. 13407. LNCS. Springer, Cham, Sept. 2022, pp. 109–127. DOI: 10.1007/978-3-031-15777-6_7.

[ZWSY26] Yuanyuan Zhou, Weijia Wang, Yiteng Sun, and Yu Yu. *Every Signing Leaks: Breaking Falcon via Floating-Point Conversion Leakage*. Cryptology ePrint Archive, Paper 2026/2124. 2026. URL: https://eprint.iacr.org/2026/2124.