

# Falcon: A Lattice-Based Digital Signature Algorithm for Post-Quantum Blockchains

Mike Mu

January 29, 2026

## Abstract

This paper outlines a comprehensive introduction to Falcon, a lattice-based digital signature algorithm, with a focus on its applications in blockchain systems. The exposition interleaves cryptographic foundations, core mechanisms, implementation subtleties, migration considerations from ECDSA/EdDSA to Falcon, and a rigorous security analysis. Each section is designed with objectives to support both protocol designers and implementation auditors in post-quantum blockchain systems.

**Keywords.** Falcon, Falcon+, post-quantum cryptography; NTRU lattices; trapdoor sampling; discrete Gaussian; FFT/NTT; side-channel security; blockchain; smart contracts; ECDSA migration.

## Contents

|          |   |           |
|----------|---|-----------|
| <b>1</b> | <b>Introduction</b>   | <b>3</b>  |
| 1.1      | Motivation: The need for post-quantum signatures in blockchain systems . . . . .  | 3         |
| 1.2      | From ECC to PQC — cryptographic agility and the NIST roadmap . . . . .            | 3         |
| 1.3      | Structure and scope of this paper . . . . .                                       | 3         |
| <b>2</b> | <b>Background and Mathematical Foundations</b>                                    | <b>4</b>  |
| 2.1      | Lattice-based cryptography overview . . . . .                                     | 4         |
| 2.2      | The NTRU lattice: definition, algebraic structure, and intuition . . . . .        | 5         |
| 2.3      | The GPV framework . . . . .   | 6         |
| 2.4      | Discrete Gaussian sampling and its cryptographic significance . . . . .           | 7         |
| 2.5      | FFT/NTT representations of polynomials and computational acceleration . . . . .   | 7         |
| <b>3</b> | <b>Core Concepts of Falcon</b>  | <b>8</b>  |
| 3.1      | High-level signing and verification flow . . . . .                                | 9         |
| 3.2      | Key generation: the NTRU equation and short vs. long basis . . . . .              | 10        |
| 3.3      | Trapdoor sampling and the Falcon signature equation . . . . .                     | 10        |
| 3.4      | FFT/NTT as Falcon’s computational backbone . . . . .                              | 11        |
| 3.5      | Advantages vs. other PQC DSAs (e.g., Dilithium): compactness and speed . . . . .  | 12        |
| 3.6      | Intrinsic challenges: numerical precision and implementation complexity . . . . . | 12        |
| <b>4</b> | <b>Falcon Algorithms in Detail</b>  | <b>13</b> |
| 4.1      | NTRUSolve — solving $fG - gF = q$ recursively . . . . .                           | 13        |
| 4.2      | ffSampling — Fast Fourier trapdoor sampler . . . . .                              | 14        |

|           |  |           |
|-----------|--|-----------|
| 4.3       | Key Generation . . . . .   | 16        |
| 4.4       | Signature Generation (Sign) . . . . .  | 16        |
| 4.5       | Verification . . . . .   | 17        |
| <b>5</b>  | <b>Implementation Complications and Numerical Stability</b>                    | <b>18</b> |
| 5.1       | Floating-point arithmetic precision requirements . . . . .                     | 18        |
| 5.2       | FFT/NTT rounding and its cryptographic implications . . . . .                  | 18        |
| 5.3       | Ensuring determinism across heterogeneous hardware . . . . .                   | 18        |
| 5.4       | Implementation variants . . . . .  | 19        |
| 5.5       | Resource considerations for on-chain/off-chain execution . . . . .             | 19        |
| <b>6</b>  | <b>Security Analysis and Cryptographic Pitfalls</b>                            | <b>19</b> |
| 6.1       | Threat model and best-known attacks (e.g., BKZ, sieving) . . . . .             | 20        |
| 6.2       | Side-channel vulnerabilities (timing, floating-point leakage, cache) . . . . . | 20        |
| 6.3       | Numerical bias and Rényi divergence in trapdoor sampling . . . . .             | 21        |
| 6.4       | Fault attacks and deterministic–randomized trade-offs . . . . .                | 22        |
| 6.5       | Parameter choices and NIST security levels (Falcon-512/Falcon-1024) . . . . .  | 23        |
| 6.6       | “Falcon+”: Toward a Formal Security Proof . . . . .                            | 24        |
| 6.7       | Falcon in Key-Recovery (KR) Mode for Blockchains . . . . .                     | 25        |
| <b>7</b>  | <b>Migration from ECC to PQC in Blockchain Systems</b>                         | <b>26</b> |
| 7.1       | Signature/key-sizes and comparative view: ECC vs. Falcon . . . . .             | 27        |
| 7.2       | Verification cost: modular arithmetic vs. FFT-based paths . . . . .            | 28        |
| 7.3       | Smart contract integration models (EVM, WASM) . . . . .                        | 30        |
| 7.4       | Hybrid cryptography: transitional dual-signature (ECDSA + Falcon) . . . . .    | 31        |
| 7.5       | PQC wallets and multisig: UX and threshold considerations . . . . .            | 32        |
| <b>8</b>  | <b>Practical Recommendations for Secure Implementation</b>                     | <b>33</b> |
| 8.1       | Key Practical Controls . . . . .   | 33        |
| <b>9</b>  | <b>Future Directions</b>   | <b>34</b> |
| 9.1       | Variants for constrained or hardware-accelerated environments . . . . .        | 35        |
| 9.2       | Prospects for hybrid lattice - ECC coexistence . . . . .                       | 35        |
| 9.3       | Standardization and interoperability challenges . . . . .                      | 35        |
| <b>10</b> | <b>Conclusion</b>  | <b>36</b> |
| 10.1      | Summary of Falcon’s cryptographic position . . . . .                           | 36        |
| 10.2      | Deployment readiness in blockchain infrastructures . . . . .                   | 36        |
| 10.3      | Open research and standardization paths . . . . .                              | 36        |
| <b>A</b>  | <b>Derivation of Falcon’s NTRU equation and its inverses</b>                   | <b>37</b> |
| <b>B</b>  | <b>Step-by-step proof of matrix–lattice equivalence</b>                        | <b>38</b> |
| <b>C</b>  | <b>FFT/NTT numerical validation</b>  | <b>39</b> |
| <b>D</b>  | <b>Comparison tables (Falcon vs. Dilithium vs. ECC)</b>                        | <b>41</b> |
| <b>E</b>  | <b>Security proof sketch and parameter selection rationale</b>                 | <b>42</b> |

# 1 Introduction

Blockchains are starting to adopt post-quantum signature schemes to prepare for a future where large-scale quantum computers may threaten existing elliptic curve cryptography. Falcon, a lattice-based digital signature algorithm, offers compact signatures and efficient verification that make it an appealing candidate for high throughput blockchain environments.

## 1.1 Motivation: The need for post-quantum signatures in blockchain systems

The security of modern blockchain ecosystems fundamentally relies on public key cryptography, most commonly elliptic curve digital signatures such as ECDSA and EdDSA. However, these schemes are vulnerable to quantum attacks, notably Shor’s algorithm [Sho94], which can efficiently recover private keys once sufficiently large quantum computers exist.

While practical quantum computers capable of such attacks remain speculative, many blockchain platforms have begun exploring post-quantum alternatives to ensure long term cryptographic resilience. Migration timelines in blockchain systems are particularly sensitive because once deployed, consensus rules and signature formats are extremely difficult to change. Thus, preparing for post-quantum security today mitigates significant systemic risks in the future.

Falcon was selected by NIST in 2022 as a candidate for future standardization under its post-quantum initiative (the expected draft being FIPS 206 "FN-DSA"), though the final standard has not yet been published [CSR22]. With its compact signature size and efficient verification, Falcon aligns well with the throughput and scalability demands of blockchain systems, making it a strong contender to succeed current elliptic curve schemes.

## 1.2 From ECC to PQC — cryptographic agility and the NIST roadmap

Modern blockchain platforms predominantly use elliptic curve digital signatures (e.g., ECDSA, EdDSA) to secure transaction authenticity and integrity. These schemes rely on discrete log assumptions that are breakable by sufficiently powerful quantum computers via Shor’s algorithm [Sho94].

Recognizing this emerging threat, the National Institute of Standards and Technology (NIST) launched its Post Quantum Cryptography (PQC) standardization effort to select algorithms that remain secure against both classical and quantum adversaries.

Key milestones along the roadmap include initial calls for candidate algorithms, multi-round analysis of performance and security, publication of the first PQC standards. For example, NIST’s draft did establish three initial signature and key encapsulation standards, and plans to publish a digital signature standard (tentatively “FN-DSA”) derived from Falcon.

For blockchain systems, cryptographic agility means designing infrastructure today to allow signature-scheme swaps tomorrow: maintaining algorithm flexibility, preserving backward compatibility, planning for dual-signature (hybrid) models, and accounting for performance, size and protocol impact. The NIST roadmap gives a valuable anchor for migration planning, signaling when ecosystem shifts may become prudent even if quantum breakers are still years away.

## 1.3 Structure and scope of this paper

The remainder of this paper is organized as follows. Section 2 introduces the mathematical background underlying Falcon, while Section 3 outlines its core concepts. Section 4 details the key

algorithms and Section 5 discusses implementation subtleties and numerical stability. Section 6 analyzes security considerations, followed by Section 7 on blockchain migration from ECC to PQC. Practical recommendations and future research directions are presented in Sections 8 and 9, with conclusions and technical appendices completing the paper.

## 2 Background and Mathematical Foundations

Falcon is built upon the Gentry–Peikert–Vaikuntanathan (GPV) framework for lattice-based signatures, which provides a general trapdoor construction for producing short preimages under hard lattice problems. The GPV design does not prescribe a specific lattice family; Falcon instantiates it over the NTRU lattice structure to obtain compact keys and efficient arithmetic in the ring  $\mathbb{Z}_q[x]/(x^n + 1)$ .

Within this framework, Falcon employs discrete Gaussian sampling as its trapdoor mechanism to ensure statistically independent signatures and to prevent information leakage from secret bases. The resulting construction combines the theoretical rigor of GPV trapdoor sampling with the practical advantages of structured (NTRU) lattices, enabling small signatures and fast verification suitable for constrained or high-throughput environments.

Understanding these foundations - GPV trapdoor principles, NTRU algebraic properties, and the role of discrete Gaussian samplers - is essential to analyzing Falcon’s correctness, compactness, and resistance against lattice reduction attacks.

### 2.1 Lattice-based cryptography overview

Lattice-based cryptography derives its security from the computational hardness of certain problems defined over high-dimensional lattices—discrete additive subgroups of  $\mathbb{R}^n$  generated by a set of basis vectors. Two of the most fundamental problems are the *Shortest Independent Vector Problem* (SIVP) and the *Closest Vector Problem* (CVP). Intuitively, both involve finding short or nearby lattice vectors, which becomes exponentially hard as the dimension increases.

Among the most influential problems whose hardness is assumed are the *Shortest Integer Solution* (SIS) [Ajt96] and *Learning With Errors* (LWE) [Reg05] problems, which serve as average-case analogues of SIVP and CVP, respectively. The SIS problem asks for a short nonzero integer vector  $\mathbf{z}$  satisfying  $\mathbf{A}\mathbf{z} = \mathbf{0} \pmod{q}$  for a random integer matrix  $\mathbf{A}$  and modulus  $q$  (typically a prime or prime power), and is believed to be hard even for quantum adversaries. Its hardness implies the infeasibility of producing short lattice relations without knowledge of a secret trapdoor.

Lattice-based constructions are particularly attractive for post-quantum security because the best known algorithms for solving CVP, SIS, or LWE - such as enumeration, sieving, and blockwise reduction (BKZ) - scale superpolynomially in the lattice dimension but only polynomially in  $q$ . This yields conservative security margins even against quantum algorithms.

Schemes like Falcon leverage structured lattices (specifically NTRU lattices) to achieve efficiency while relying on these same underlying hardness assumptions. In essence, lattice cryptography provides both the theoretical foundation and the computational intractability guarantees that make Falcon and other post-quantum signature schemes viable for long-term use.

## 2.2 The NTRU lattice: definition, algebraic structure, and intuition

The NTRU (N-th Degree Truncated Polynomial Ring Units) lattice structure, originally introduced by Hoffstein, Pipher, and Silverman [HPS98], provides the algebraic foundation for Falcon. Let  $\phi = x^n + 1$  with  $n = 2^\kappa$  a power of two, and let  $q \in \mathbb{N}^*$  be the modulus. A set of NTRU secrets consists of four polynomials

$$f, g, F, G \in \mathbb{Z}[x]/(\phi)$$

which satisfy the fundamental *NTRU equation*

$$fG - gF = q \pmod{\phi}.$$

Provided that  $f$  is invertible modulo  $q$ , one can define the public key polynomial

$$h \leftarrow g \cdot f^{-1} \pmod{q}.$$

In the NTRU lattice setting, the public and secret structures can be represented through two distinct bases that generate the same lattice. The first, or *public basis*, is given by

$$\mathbf{A} = \begin{bmatrix} 1 & h \\ 0 & q \end{bmatrix},$$

while the second, or *secret basis*, is

$$\mathbf{B} = \begin{bmatrix} f & g \\ F & G \end{bmatrix}.$$

Note that an equivalent form with negatives,

$$\mathbf{B}' = \begin{bmatrix} g & -f \\ G & -F \end{bmatrix},$$

is also used in the literature and appears later in this document when discussing the GPV framework instantiation. Both forms generate the same lattice (they differ only by a change of basis that preserves the lattice structure), and both satisfy the NTRU equation  $fG - gF = q$ .

It can be verified that both matrices  $\mathbf{A}$  and  $\mathbf{B}$  generate the same  $2n$ -dimensional lattice

$$\Lambda = \left\{ (u, v) \in \mathbb{Z}^{2n} \mid u + hv \equiv 0 \pmod{q} \right\},$$

but they differ sharply in geometry. The public basis  $\mathbf{A}$  contains large entries (the polynomial  $h$  and the scalar  $q$ ), leading to a *long and highly skewed* basis that is difficult to use for generating short lattice vectors.

In contrast, the secret basis  $\mathbf{B}$  consists of small polynomials ( $f, g, F, G$ ) that form a much shorter and nearly orthogonal basis—that is, the basis vectors are approximately perpendicular to one another, with small Gram–Schmidt coefficients. Recovering this short, well-conditioned basis from the public one is believed to be as hard as solving the Shortest Vector Problem (SVP) or Shortest Independent Vector Problem (SIVP) in  $\Lambda$ , forming the core of the NTRU hardness assumption.

**Instantiation with the GPV framework.** Following the Gentry–Peikert–Vaikuntanathan (GPV) construction, Falcon instantiates the trapdoor signature framework over this NTRU lattice. (The GPV framework is detailed in the following subsection.) In this setting, the GPV public and secret trapdoor bases are:

$$\mathbf{A} = [1 \mid h^*], \quad \mathbf{B} = \begin{bmatrix} g & -f \\ G & -F \end{bmatrix},$$

which correspond, respectively, to the verifier’s view (knowledge of  $h$ ) and the signer’s view (knowledge of the secret trapdoor basis  $(f, g, F, G)$ ).

### 2.3 The GPV framework

In 2008, Gentry, Peikert, and Vaikuntanathan (GPV) [GPV08] introduced a general framework for constructing lattice-based digital signatures with provable security under the hardness of the Short Integer Solution (SIS) problem.

At a high level, the framework can be described as follows:

- The *public key* contains a full-rank matrix  $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$  (with  $m > n$ ), which defines a  $q$ -ary lattice

$$\Lambda_q = \{ \mathbf{x} \in \mathbb{Z}^m : \mathbf{A}\mathbf{x} \equiv \mathbf{0} \pmod{q} \}.$$

- The *private key* consists of a matrix  $\mathbf{B} \in \mathbb{Z}^{m \times m}$  generating the lattice orthogonal to  $\Lambda_q$  modulo  $q$ , denoted  $\Lambda_q^\perp$ , such that for any  $\mathbf{x} \in \Lambda_q$  and  $\mathbf{y} \in \Lambda_q^\perp$ ,  $\langle \mathbf{x}, \mathbf{y} \rangle \equiv 0 \pmod{q}$ . Equivalently, the rows of  $\mathbf{A}$  and  $\mathbf{B}$  are orthogonal:

$$\mathbf{B}\mathbf{A}^\top = \mathbf{0} \pmod{q}.$$

Given a message  $\mu$ , the signature is a *short vector* (i.e., a vector with small Euclidean norm)  $\mathbf{s} \in \mathbb{Z}_q^m$  such that

$$\mathbf{s}\mathbf{A}^\top = H(\mu) \pmod{q},$$

where  $H : \{0, 1\}^* \rightarrow \mathbb{Z}_q^n$  is a cryptographic hash function. Verification is straightforward: the verifier checks that  $\mathbf{s}$  is short and that the above modular equation holds.

The signing process is more subtle. To sign a message  $\mu$ , the signer first chooses an arbitrary preimage  $\mathbf{c}_0 \in \mathbb{Z}_q^m$  satisfying  $\mathbf{c}_0\mathbf{A}^\top = H(\mu)$ , which can be computed by standard linear algebra since  $\mathbf{A}$  is public. This preimage  $\mathbf{c}_0$  is generally long and thus unsuitable as a signature. Using the private trapdoor basis  $\mathbf{B}$ , the signer computes a lattice vector  $\mathbf{v} \in \Lambda_q^\perp$  close to  $\mathbf{c}_0$  and sets

$$\mathbf{s} = \mathbf{c}_0 - \mathbf{v}.$$

Because  $\mathbf{v}$  is in  $\Lambda_q^\perp$ , we have  $\mathbf{s}\mathbf{A}^\top = H(\mu)$ , and if  $\mathbf{v}$  is chosen close enough to  $\mathbf{c}_0$ , then  $\mathbf{s}$  is short.

The GPV framework therefore introduces the concept of a *trapdoor sampler*: a probabilistic algorithm that uses  $\mathbf{B}$  to sample such short preimages according to a discrete Gaussian distribution. This ensures that signatures are both norm-bounded and statistically independent of the trapdoor basis, yielding provable security in the (quantum) random-oracle model.

This abstract construction underlies many modern lattice-based signatures. Falcon, in particular, instantiates the GPV framework over the NTRU lattice family, combining GPV’s theoretical guarantees with the compactness and arithmetic efficiency of structured rings.

## 2.4 Discrete Gaussian sampling and its cryptographic significance

The discrete Gaussian distribution plays a central role in lattice-based cryptography, serving as the mechanism for producing short and statistically well-behaved lattice vectors. For a lattice  $\Lambda \subset \mathbb{R}^n$  and a parameter  $\sigma > 0$ , the discrete Gaussian over  $\Lambda$  centered at  $\mathbf{c}$  is defined as

$$D_{\Lambda, \sigma, \mathbf{c}}(\mathbf{x}) \propto \exp\left(-\frac{\|\mathbf{x} - \mathbf{c}\|^2}{2\sigma^2}\right), \quad \text{for } \mathbf{x} \in \Lambda.$$

Intuitively, it assigns higher probability to shorter vectors near  $\mathbf{c}$ , allowing trapdoor algorithms to output lattice elements of small norm without leaking information about the secret basis.

Within the GPV framework, discrete Gaussian sampling is used to compute short preimages  $\mathbf{s}$  satisfying  $\mathbf{A}\mathbf{s} \equiv \mathbf{y} \pmod{q}$  while ensuring that  $\mathbf{s}$  is drawn from a distribution independent of the trapdoor. This property eliminates deterministic patterns that could reveal the secret basis and provides provable bounds on statistical distance from the ideal Gaussian.

In practice, the choice of the standard deviation  $\sigma$  must exceed the Gram–Schmidt norms of the basis by a security margin to guarantee negligible leakage. Implementations such as Falcon use a recursive variant known as *ffSampling* to sample efficiently in the Fourier domain, maintaining numerical precision while adhering to GPV’s theoretical guarantees. Discrete Gaussian sampling thus underpins both the security and correctness of lattice-based signatures: it ensures that signatures remain short, properly distributed according to a discrete Gaussian, and independent of the private key, even under adaptive chosen-message attacks.

## 2.5 FFT/NTT representations of polynomials and computational acceleration

Falcon operates over the polynomial ring  $R_q = \mathbb{Z}_q[x]/(x^n + 1)$ , where polynomial multiplication is the dominant operation during key generation, signing, and verification. Naively, multiplying two degree  $(n - 1)$  polynomials costs  $O(n^2)$  operations. Falcon circumvents this by expressing polynomials in the frequency domain through fast transform techniques, specifically the Fast Fourier Transform (FFT) and its modular analogue, the Number Theoretic Transform (NTT).

**FFT representation.** In the floating-point variant used by Falcon’s reference implementation, each polynomial  $f(x)$  is mapped to its complex evaluation vector under roots of  $x^n + 1$ :

$$\hat{f}_k = f(\omega^k), \quad \omega = e^{i\pi/n}.$$

For polynomials  $f(x), g(x) \in R_q$  and their product  $h(x) = f(x) \cdot g(x)$  in the ring  $R_q$ , pointwise multiplication of transformed polynomials satisfies

$$\hat{h}_k = \hat{f}_k \cdot \hat{g}_k,$$

which corresponds to convolution in the coefficient domain after an inverse FFT. This reduces multiplication cost to  $O(n \log n)$  and enables efficient recursive algorithms such as *ffSampling*, which generate short vectors in the Fourier domain while preserving the Gaussian shape. The use of FFT also allows Falcon to represent its NTRU basis as floating-point matrices, where LDL decompositions and Gram–Schmidt orthogonalizations can be computed efficiently.

**NTT representation.** For integer-only or hardware-restricted environments, Falcon’s operations can also be expressed using the Number Theoretic Transform, a finite-field analogue of the FFT. Let  $\psi$  be a primitive  $2n$ -th root of unity in  $\mathbb{Z}_q$  (satisfying  $\psi^{2n} \equiv 1 \pmod{q}$  and  $\psi^n \equiv -1 \pmod{q}$ ). Then for  $f(x) \in R_q$ ,

$$\tilde{f}_k = \sum_{j=0}^{n-1} f_j \psi^{jk} \pmod{q},$$

and for the product  $h(x) = f(x) \cdot g(x)$  in  $R_q$ , polynomial multiplication proceeds as

$$\tilde{h}_k = \tilde{f}_k \cdot \tilde{g}_k \pmod{q},$$

followed by an inverse NTT to recover  $h(x)$ . This approach eliminates floating-point rounding issues and ensures determinism across heterogeneous architectures, which is critical for reproducibility and side-channel resistance.

**Computational benefits.** Both FFT and NTT representations provide substantial speedups by transforming convolution into elementwise multiplication. They also enable recursive decomposition of lattice operations, improving numerical stability and memory locality. In Falcon, these transforms are integral not only to polynomial multiplication but also to sampling and norm computation, achieving asymptotic complexity  $O(n \log n)$  and practical throughput suitable for on-chain or hardware-constrained verification.

In summary, FFT and NTT techniques transform Falcon’s underlying ring arithmetic from a bottleneck into an enabler—balancing efficiency, precision, and platform portability in its lattice-based implementation.

### 3 Core Concepts of Falcon

Falcon implements the GPV trapdoor signature framework using NTRU lattices and fast Fourier sampling techniques. This section presents the conceptual anatomy of the scheme: how key pairs are generated, how signing and verification operate, and how Falcon leverages FFT/NTT arithmetic to achieve efficiency. It also contrasts Falcon’s design choices with other post-quantum digital signatures such as Dilithium, and discusses the numerical subtleties that make Falcon both elegant and technically demanding to implement.

In essence, Falcon achieves compact signatures by combining:

- the *GPV framework*, providing a mathematically sound trapdoor sampling mechanism;
- the *NTRU lattice*, yielding small keys and efficient polynomial arithmetic;
- and the *FFT-based Gaussian sampler*, enabling fast yet precise computation of short lattice vectors.

These ingredients together realize a provably secure, high-performance lattice signature that balances compactness, speed, and precision. The following subsections detail each component of Falcon’s design and their interplay.

### 3.1 High-level signing and verification flow

At a high level, Falcon implements the GPV signing framework on top of an NTRU lattice, using FFT-domain sampling and compact encoding. The signature process can be viewed as a precise sequence of target generation, trapdoor sampling, norm rejection, and verification.

**Signing.** To sign a message  $m$ , the signer first samples a random nonce  $r$  and computes a hashed lattice point

$$\mathbf{c} = \text{HashToPoint}(r\|m) \in R_q,$$

which defines the target vector in the NTRU space. The target in the Fourier domain is then computed as

$$\mathbf{t} = (\hat{\mathbf{c}}, \mathbf{0}_n) \cdot \hat{\mathbf{B}}^{-1},$$

where  $\hat{\mathbf{B}}$  is the FFT representation of the private NTRU basis. Using the trapdoor sampler (`ffSampling`), the signer draws a short lattice vector

$$\mathbf{z} \leftarrow D_{\Lambda, \sigma, \mathbf{t}}$$

distributed according to a discrete Gaussian centered on  $\mathbf{t}$ . The difference between the target and sampled point yields the short signature in the frequency domain:

$$\hat{\mathbf{s}} = \mathbf{t} - \mathbf{z}.$$

If the Euclidean norm satisfies the rejection bound  $\|\hat{\mathbf{s}}\| \leq \beta$ , the signature is accepted; otherwise, the sampling step is repeated. Finally, the signer computes the inverse FFT to obtain

$$(\mathbf{s}_1, \mathbf{s}_2) = \mathcal{F}^{-1}(\hat{\mathbf{s}}),$$

and encodes  $(r, \mathbf{s}_2)$  as the final signature  $\sigma$ .

**Verification.** Given  $(m, \sigma)$  and the public key  $h$ , the verifier decodes  $\sigma$  to recover the random seed  $r$  and the polynomial  $\mathbf{s}_2$ . It recomputes the target point

$$\mathbf{c} = \text{HashToPoint}(r\|m),$$

then reconstructs

$$\mathbf{s}_1 = \mathbf{c} - \mathbf{s}_2 h \pmod{q}.$$

Finally, the verifier checks that the Euclidean norm of  $(\mathbf{s}_1, \mathbf{s}_2)$  satisfies

$$\|(\mathbf{s}_1, \mathbf{s}_2)\| \leq \beta.$$

This norm check is equivalent to the check  $\|\hat{\mathbf{s}}\| \leq \beta$  performed in the frequency domain during signing, since the FFT preserves the Euclidean norm (up to a normalization constant accounted for in the implementation, as guaranteed by Parseval's theorem). If this condition holds, the signature is accepted; otherwise, it is rejected.

**Overview.** Intuitively, Falcon's signing process corresponds to finding a short lattice vector  $\hat{\mathbf{s}}$  close to a target  $\mathbf{t}$  derived from the message, with the help of a trapdoor basis in the Fourier domain. The rejection step guarantees that all valid signatures are statistically indistinguishable, and the FFT structure allows all operations—basis inversion, sampling, and norm computation—to be performed efficiently in  $O(n \log n)$  time.

### 3.2 Key generation: the NTRU equation and short vs. long basis

Falcon’s key generation follows the NTRU lattice construction, producing both a public key for verification and a short secret basis for trapdoor sampling. The goal is to find small polynomials  $(f, g, F, G)$  in  $R = \mathbb{Z}[x]/(x^n + 1)$  satisfying the *NTRU equation*

$$fG - gF = q.$$

Given such a relation,  $f$  must be invertible modulo  $q$ , and the public key is defined as

$$h = gf^{-1} \pmod{q}.$$

The secret key thus consists of the short pair  $(f, g)$  (and optionally  $(F, G)$ ), while the public key contains only  $h$ .

The two corresponding lattice bases are:

$$\mathbf{A} = \begin{pmatrix} 1 & h \\ 0 & q \end{pmatrix}, \quad \mathbf{B} = \begin{pmatrix} f & g \\ F & G \end{pmatrix},$$

which generate the same NTRU lattice

$$\Lambda_h = \{ (u, v) \in \mathbb{Z}^{2n} : u + hv \equiv 0 \pmod{q} \}.$$

However, their geometries differ substantially:  $\mathbf{A}$  is the *public* (long and skewed) basis, while  $\mathbf{B}$  is the *private* well-conditioned - meaning short and nearly orthogonal - trapdoor basis. The latter enables efficient Gaussian sampling of short lattice vectors during signing.

Key generation proceeds by randomly sampling  $(f, g)$  from a narrow discrete Gaussian, then solving the NTRU equation recursively for  $(F, G)$  using the `NTRUSolve` algorithm. The resulting basis  $\mathbf{B}$  is validated to ensure that the Gram–Schmidt norms of its rows remain below the security bound required for the sampler’s precision. Finally,  $(f, g)$  and  $(F, G)$  are stored as the secret key, while  $h$  is published as the verifier’s public key.

### 3.3 Trapdoor sampling and the Falcon signature equation

Trapdoor sampling lies at the heart of Falcon’s design, bridging theoretical security and practical correctness. In lattice-based signatures, the trapdoor provides a means to sample short lattice vectors that satisfy a prescribed modular relation, without revealing the secret basis that enables this sampling.

In Falcon, the secret NTRU basis acts as such a trapdoor. The sampler takes as input a target point derived from the message hash and outputs a lattice vector drawn from a discrete Gaussian distribution centered at that target. The purpose of this procedure is twofold:

- **Correctness:** every sample corresponds to a valid lattice vector  $(\mathbf{s}_1, \mathbf{s}_2)$  that satisfies the public congruence relation  $\mathbf{s}_1 + h\mathbf{s}_2 \equiv \mathbf{c} \pmod{q}$  (where  $\mathbf{c} = \text{HashToPoint}(r||m)$ ), ensuring that verification will always succeed for honestly generated signatures. This congruence relation is a property of the lattice itself (specifically, it characterizes membership in the NTRU lattice  $\Lambda_h = \{ (u, v) \in \mathbb{Z}^{2n} : u + hv \equiv 0 \pmod{q} \}$ ) and does not depend on the choice of basis—any lattice vector satisfies this relation regardless of which basis was used to generate it.

- **Security:** the Gaussian distribution ensures that signatures are statistically independent of the underlying trapdoor basis, preventing any leakage of secret information even under chosen-message attacks.

Thus, trapdoor sampling enforces both functional correctness - guaranteeing that signatures verify, and statistical security - ensuring that successive signatures remain unlinkable and indistinguishable from ideal Gaussian outputs. In practice, Falcon realizes this via its recursive `ffSampling` algorithm in the Fourier domain, which implements GPV’s theoretical sampler with high numerical precision.

### 3.4 FFT/NTT as Falcon’s computational backbone

Fast Fourier Transform (FFT) and Number Theoretic Transform (NTT) arithmetic form the computational core of Falcon, enabling its efficiency and numerical stability. Both transforms serve the same conceptual purpose - turning costly polynomial convolutions into point-wise multiplications - but are applied in distinct parts of Falcon’s workflow to meet both performance and determinism goals.

**Use of FFT.** Within Falcon, the Fast Fourier Transform is central to private-key computations: during key generation the LDL\* decomposition of the NTRU basis is computed in the Fourier domain, and during signing the fast Fourier sampler (`ffSampling`) operates entirely on frequency-domain representations of lattice vectors. All polynomials such as the secret basis  $(f, g, F, G)$  and the target vector  $t$  are converted via FFT, so that recursive splits, merges, and Gaussian-sampling steps exploit independent frequency channels. This transforms  $O(n^2)$  coefficient-domain convolution into  $O(n \log n)$  point-wise multiplication and rounding, enabling high-throughput signature generation while preserving the precise statistical distribution required for security.

**Use of NTT.** In Falcon, the Number Theoretic Transform (NTT) is used for efficient modular arithmetic in the ring

$$R_q = \mathbb{Z}_q[x]/(x^n + 1),$$

especially in verification and public-key operations where determinism and exact integer arithmetic are required. A prime modulus  $q$  is chosen so that a primitive  $2n$ -th root of unity  $\omega \in \mathbb{Z}_q$  exists with

$$\omega^{2n} \equiv 1 \pmod{q}, \quad \omega^n \equiv -1 \pmod{q}.$$

The NTT maps a polynomial

$$f(x) = \sum_{j=0}^{n-1} f_j x^j$$

to its evaluation vector under those roots:

$$\tilde{f}_k = \sum_{j=0}^{n-1} f_j \omega^{jk} \pmod{q}.$$

Polynomial multiplication is then computed point-wise:

$$\tilde{h}_k = \tilde{f}_k \cdot \tilde{g}_k \pmod{q},$$

and the inverse NTT recovers the product  $h(x)$ . In verification, for example, the public-key operation uses the relation

$$\mathbf{s}_1 = \mathbf{c} - \mathbf{s}_2 h \pmod{q},$$

and adopting NTT speeds up the modular multiplications involved, while ensuring that the result is deterministic, branch-free and portable across hardware platforms.

**Benefits.** FFT and NTT thus play complementary roles: FFT enables high-speed arithmetic and efficient trapdoor sampling in software, while NTT ensures deterministic and verifiable modular arithmetic in constrained or verifiable environments. Together, they allow Falcon to blend provable lattice-based security with practical throughput, achieving compact signatures with micro-second signing and verification capabilities without sacrificing robustness or reproducibility.

### 3.5 Advantages vs. other PQC DSAs (e.g., Dilithium): compactness and speed

When comparing the lattice-based signature schemes Falcon and CRYSTALS-Dilithium, two key dimensions emerge: signature/public-key size and signing/verification performance.

**Compactness.** Falcon instantiates the GPV framework over structured NTRU lattices, yielding extremely compact artifacts. Specifically, a public-key of size 897 bytes and a signature of size 666 bytes at the 128-bit security level. By contrast, CRYSTALS-Dilithium [BDK<sup>+</sup>21] produces signatures in the range of 2.4 kB (and public keys 1.3 kB) at the same security level. This compactness makes Falcon particularly appealing for bandwidth-sensitive applications (e.g., certificate chains, embedded IoT devices, blockchains) where signature size directly impacts throughput or storage.

**Performance.** From a performance perspective, Dilithium typically excels in key generation and signing on low-power platforms - owing to its simpler integer arithmetic and absence of complex floating-point samplers. Falcon’s strength lies primarily in its verification speed: benchmark studies have shown that Falcon verification can achieve  $3\times - 4\times$  speedup over Dilithium on ARM-based platforms with optimized implementations, though actual performance varies significantly with hardware architecture, compiler optimizations, and implementation details. However, the complexity of Falcon’s trapdoor Gaussian sampler and floating-point/FFT arithmetic may pose implementation and constant-time challenges.

**Trade-offs and deployment implications.** In short: if minimizing signature size is paramount and the environment supports secure implementation of the more sophisticated arithmetic, Falcon is a compelling choice. If simplicity, deterministic arithmetic and broad hardware support are dominant priorities, then Dilithium may be the more practical default. The two schemes serve complementary roles in the post-quantum ecosystem.

### 3.6 Intrinsic challenges: numerical precision and implementation complexity

Although the scheme offers strong theoretical guarantees and practical performance, the implementation of Falcon presents several intrinsic challenges that demand careful attention.

**Floating-point and FFT precision.** The trapdoor sampling in Falcon’s signing algorithm, as well as the LDL\* decomposition computed during key generation, are implemented in the floating-point domain and rely on the accuracy of the fast Fourier transform. Incorrect rounding, usage of sub-normal numbers, or inconsistent implementations across platforms can lead to subtle signature bias or leakage of secret information. For example, implementations must avoid NaNs or infinities and implement deterministic rounding strategies to guarantee reproducibility.

**Side-channel vulnerability surface.** The use of floating-point arithmetic, complex recursions, and non-trivial branching based on coefficient norms increases the attack surface for timing, cache, and power side-channels. Single-trace attacks have been demonstrated against specific

implementations of Falcon’s discrete Gaussian sampler, underscoring the need for constant-time, platform-specific instrumentation.

**Implementation complexity and resource constraints.** On constrained devices (e.g., microcontrollers without FPU), the recursion depth, memory usage (stack and heap), and precision requirements of Falcon’s key generation and signing pipeline can exceed available resources or force sub-optimal compromises. Resource-efficiency improvements are underway (e.g., Falcon-M [KIB<sup>+</sup>25]), but any deployment must carefully benchmark stack size, memory allocation, and floating-point/fixed-point equivalence.

In summary, while Falcon achieves compelling compactness and high verification speed, its correct implementation hinges on managing numerical precision, side-channel resistance, and resource feasibility. These aspects are non-trivial in real-world deployments and must be treated with high rigor.

## 4 Falcon Algorithms in Detail

Falcon’s security and correctness rest on finely tuned algorithmic components: the recursive solver for the NTRU equation, the highly efficient fast-Fourier trapdoor sampler, and the norm-bounded key-generation and signing routines. This section presents a detailed view of these algorithms — how they are structured, how they interface with one another, and how their parameters (e.g., Gram–Schmidt norms, rejection bounds) enforce both performance and provable security. From the recursive descent in `NTRUSolve`, through Gaussian sampling in `ffSampling`, to the modular verification logic, each sub-procedure is critical to the integrity of the overall signature scheme.

### 4.1 NTRUSolve — solving $fG - gF = q$ recursively

To complete the secret key generation in FALCON, the polynomials  $f, g \in R = \mathbb{Z}[x]/(x^n + 1)$  (sampled short) must be extended into a full trapdoor basis by finding  $F, G \in R$  satisfying:

$$fG - gF = q \pmod{x^n + 1}.$$

The algorithm `NTRUSolve` proceeds in three conceptual phases:

#### Algorithmic flow: norm map, recursive descent/ascent, reduce

1. **Norm map (downwards):** Using the field-norm operator  $N$ , the input pair  $(f, g)$  in degree  $n$  is mapped to a reduced ring of degree  $n/2$ :

$$(f', g') = (N(f), N(g)) \in \mathbb{Z}[x]/(x^{n/2} + 1).$$

This halves the problem size while slightly increasing coefficient size.

2. **Recursive solve:** Call `NTRUSolve` on  $(f', g')$  to obtain  $(F', G')$  such that

$$f'G' - g'F' = q.$$

3. **Lift & reduce (ascent):** From  $(F', G')$  one constructs unreduced  $(F, G)$  at degree  $n$  via substitutions in the Fourier domain (or NTT/RNS), and then applies a reduction step:

$$(F, G) \leftarrow \text{Reduce}(f', g', F', G')$$

where Reduce acts like Babai’s nearest-plane algorithm to bring  $(F, G)$  to a short basis relative to  $(f, g)$ .

### Output constraints and expected Gram–Schmidt norms

The result  $(F, G)$  must fulfill two critical criteria:

- Correctness of the NTRU equation:  $fG - gF = q \pmod{x^n + 1}$ .
- The secret basis

$$\mathbf{B} = \begin{pmatrix} f & g \\ F & G \end{pmatrix}$$

must have Gram–Schmidt norms bounded below a target—typically  $\gamma \approx 1.17\sqrt{q}$  for security and efficient sampling. If the norm bound is exceeded, key-generation restarts.

Meeting these constraints ensures that the trapdoor sampler (`ffSampling`) will produce short signatures and maintain statistical independence.

### 4.2 ffSampling — Fast Fourier trapdoor sampler

The `ffSampling` algorithm lies at the heart of FALCON’s signing mechanism by transforming the trapdoor basis into usable short lattice vectors via a Fourier-domain Gaussian sampler. This section explains what `ffSampling` does, how it does it, and why key design choices (such as norm bounds and per-frequency recursion) are critical for both correctness and security.

#### LDL decomposition and recursive Gaussian generation

To facilitate efficient Gaussian sampling, Falcon begins by computing an  $LDL^*$  decomposition of the Gram matrix associated with the secret basis in the Fourier (FFT) domain. Concretely, one computes:

$$\mathbf{G} = \widehat{\mathbf{B}} \widehat{\mathbf{B}}^*, \quad \mathbf{G} = \mathbf{L} \mathbf{D} \mathbf{L}^*,$$

where  $\widehat{\mathbf{B}}$  is the Fourier-transformed basis,  $\mathbf{L}$  is a lower-triangular matrix with unit diagonals, and  $\mathbf{D}$  is diagonal. This decomposition enables a recursive divide-and-conquer sampler: the lattice of dimension  $2n$  is split into two sub-lattices of size  $n$ , and sampling proceeds by sampling Gaussians on leaves (small dimension) and combining the results upward.

**Insight:** The LDL tree serves two purposes: it bounds the Gram–Schmidt norms of the basis (ensuring short samples) and organizes the sampling into a structured recursive pipeline. By working in the Fourier domain, convolution reduces to point-wise multiplication, enabling the sampler to have cost  $O(n \log n)$  rather than  $O(n^2)$ . This is essential for practical signing rates.

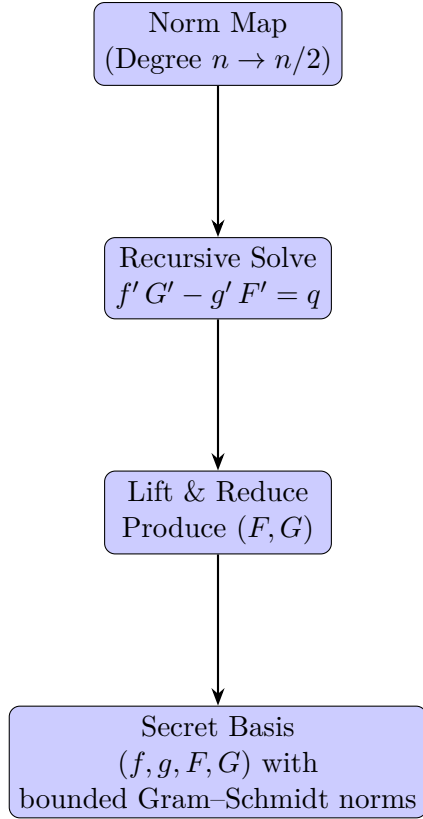
#### Per-frequency sampling; security vs. precision trade-offs

Once the LDL tree is built, the sampler operates in the Fourier domain per frequency: at each level, for a frequency index  $k$ , a discrete Gaussian sample  $z_k$  is drawn with standard deviation  $\sigma_k$  (derived from the diagonal entry of  $\mathbf{D}$ ). The signature vector is then formed by subtracting the sampled vector from the target:

$$\widehat{\mathbf{s}} = \mathbf{t} - \mathbf{z}.$$

This per-frequency sampling ensures each component of the vector is bounded by a tailored standard deviation, tuned to the Gram–Schmidt norm of the secret basis at that level.

Begin Key-Gen



End - Secret Basis

Figure 1: Flow of NTRUSolve

Insight: Choosing  $\sigma_k$  just above the Gram-Schmidt bound ensures minimal "rejection" while preserving Gaussian-shape statistics, which prevents signature leakage about the trapdoor. On the other hand, floating-point precision must be sufficient so that rounding errors don't degrade the statistical distribution - and thus the security proof. Hence there is a **trade-off**: smaller  $\sigma_k$  (tighter bound) improves compactness but demands higher precision; looser  $\sigma_k$  relaxes precision but slightly inflates signature norms.

Together, the LDL decomposition, per-frequency sampling, and tight norm constraints allow Falcon's trapdoor sampler to generate short signatures distributed according to a discrete Gaussian that satisfy both the correctness requirement (valid lattice preimage) and the security requirement (leakage minimal).

### 4.3 Key Generation

Key generation in FALCON is the foundational step that creates both the public key and the secret trapdoor basis. It begins by sampling short polynomials and proceeds to solving the NTRU equation and deriving the public verification key. The resulting key pair is validated to ensure it meets precise norm bounds, which underlie both correctness of signing and the security of Gaussian sampling.

#### Sampling $f, g$ , generating $F, G$ , and norm checking

The process starts by sampling two polynomials

$$f, g \in R = \mathbb{Z}[x]/(x^n + 1)$$

from a centered discrete Gaussian distribution, aiming for small coefficient norms. If the norm of the vector  $(f, g)$  exceeds a preset threshold, the sampling is restarted. During sampling of  $f$ , the algorithm also verifies that  $f$  is invertible in the ring  $R_q = \mathbb{Z}_q[x]/(x^n + 1)$ ; only if  $f$  is invertible does the algorithm proceed to compute the public key  $h = g f^{-1} \bmod (q, x^n + 1)$ . Next, the algorithm solves for  $F, G$  satisfying the NTRU equation

$$f G - g F = q \pmod{x^n + 1},$$

via the recursive `NTRUSolve` routine. Once  $(F, G)$  is found, the secret basis

$$\begin{pmatrix} f & g \\ F & G \end{pmatrix}$$

is checked to ensure its Gram-Schmidt norms remain below target bounds (e.g.,  $\approx 1.17\sqrt{q}$ ). If the bound is exceeded, key generation restarts. This norm-checking guarantees that the subsequent Gaussian sampler operates within its security and correctness parameters.

#### Deriving the public key $h = g f^{-1} \bmod q$

Once a valid secret basis is accepted, the public key is derived simply as:

$$h = g \cdot f^{-1} \bmod (q, x^n + 1).$$

Here  $f^{-1}$  is the inverse of  $f$  in the ring  $R_q$ . The verifier's public key is the polynomial  $h$ , which implicitly defines the lattice map  $(s_1, s_2) \mapsto s_1 + h s_2 \bmod q$ . Since only  $h$  (not  $f, g, F, G$ ) is published, the trapdoor basis remains secret. The compact public key and bounded secret basis together provide the compactness and provable security that characterize Falcon.

### 4.4 Signature Generation (Sign)

The signing algorithm in FALCON translates a message into a compact lattice-based signature by hashing to a target, sampling a short preimage via the trapdoor, and encoding the result in compressed form. The process must maintain both correctness (the signature verifies) and security (no leakage of the trapdoor).

### HashToPoint, computing $t$ , sampling $\mathbf{s}$ , and norm rejection

Given a message  $m$  and a random seed  $r$ , the signer computes

$$\mathbf{c} = \text{HashToPoint}(r \parallel m),$$

which maps into the lattice target. Next the signer computes the Fourier-domain target vector

$$\mathbf{t} = (\widehat{\mathbf{c}}, \mathbf{0}_n) \widehat{\mathbf{B}}^{-1},$$

where  $\widehat{\mathbf{B}}$  is the FFT representation of the trapdoor basis. Using `ffSampling`, the signer draws a short lattice vector

$$\mathbf{z} \leftarrow D_{\Lambda, \sigma, \mathbf{t}},$$

and computes

$$\widehat{\mathbf{s}} = \mathbf{t} - \mathbf{z}.$$

The inverse FFT is applied to obtain  $(\mathbf{s}_1, \mathbf{s}_2)$  in coefficient form. A norm check is then performed:

$$\|(\mathbf{s}_1, \mathbf{s}_2)\| \leq \beta.$$

If the bound is exceeded, the process repeats with a new seed  $r$ . This rejection mechanism ensures both signature tightness and compliance with the security proof.

### Compression and canonical encoding

Once  $(\mathbf{s}_1, \mathbf{s}_2)$  satisfies the norm bound, the signature is compressed into a byte sequence: the random seed  $r$ , plus a succinct encoding of  $\mathbf{s}_2$  (and implicitly  $\mathbf{s}_1$ ). Compression exploits the small coefficient structure and the Gaussian tail bounds to reduce size. The canonical encoding ensures interoperability and deterministic verification across implementations.

## 4.5 Verification

This section describes how a verifier using only the public key validates a signature: by reconstructing the hidden component, checking the modular relation in the lattice, and enforcing a norm bound. Correctness must ensure that every honestly generated signature passes, and security must ensure that invalid or malicious signatures are rejected.

### Public verification equation

Given a public key  $h \in R_q$ , a message  $m$ , and a received signature  $(r, s_2)$ , the verifier first computes

$$\mathbf{c} = \text{HashToPoint}(r \parallel m)$$

in the ring  $R_q = \mathbb{Z}_q[x]/(x^n + 1)$ . The verifier then decodes or decompresses  $\mathbf{s}_2$ , and reconstructs

$$\mathbf{s}_1 = \mathbf{c} - h \cdot \mathbf{s}_2 \pmod{q}.$$

The modular relation  $\mathbf{s}_1 + h \mathbf{s}_2 \equiv \mathbf{c} \pmod{q}$  holds by construction for honest signatures and ensures that the signature is a valid lattice preimage under the map defined by  $h$ .

## Edge cases and failure conditions

The verifier must also perform the following checks in order to avoid accepting invalid, malformed, or malicious signatures:

- If the decoding or decompression of  $\mathbf{s}_2$  fails (for example invalid header bytes or out-of-range coefficients), the signature is immediately *rejected*.
- After computing  $(\mathbf{s}_1, \mathbf{s}_2)$ , the squared Euclidean norm

$$\|(\mathbf{s}_1, \mathbf{s}_2)\|^2$$

is compared to the specified bound  $\beta^2$ . If it exceeds this bound, the signature is *rejected*.

- All arithmetic must be performed modulo  $q$  and normalized into the standard interval (e.g., coefficients in  $[-q/2] \dots [q/2]$ ). Failure to enforce normalization may lead to incorrect comparison of norms. If and only if all of these checks pass is the signature accepted.

## 5 Implementation Complications and Numerical Stability

Modern high-performance implementations of FALCON face multiple non-trivial engineering challenges: though the cryptographic design is theoretically sound, the practical behaviour of floating-point and fixed-point arithmetic, hardware variation, and resource constraints can significantly impact both correctness and security.

### 5.1 Floating-point arithmetic precision requirements

The primary trapdoor sampler and many of the lattice-basis decompositions in Falcon rely on floating-point arithmetic (e.g., double precision IEEE 754) to evaluate FFTs, LDL\* decompositions and Gram-Schmidt norms. Even small rounding errors, sub-normal numbers or implementation differences across processors may lead to drift in norms or signature distributions. Thus implementations must design for high precision, monitor rounding behaviour, and where possible eliminate or constrain floating point constructs.

### 5.2 FFT/NTT rounding and its cryptographic implications

Though FFT and NTT accelerate polynomial operations, they introduce rounding and representation subtleties. In the Fourier domain FFT uses complex floating-point values and thus rounding error; in the modular domain NTT uses exact integer arithmetic but may still require truncation or modular reduction that affects coefficient ranges. The original specification of Falcon [FKT<sup>+</sup>18] explicitly warns about floating-point reproducibility issues ("53-bit precision of floating point values used ... are sufficient" but platform variance may affect reproducibility). These subtle numerical differences can affect signature norm bounds or sampler rejection rates, which in turn can create side channel leakage. For security sensitive contexts the implementation must ensure that FFT/NTT transforms are deterministic and consistent across platforms.

### 5.3 Ensuring determinism across heterogeneous hardware

Because modern devices (x86, ARM, microcontrollers) may implement floating-point differently (different rounding modes, fused-multiply adds, subnormals, hardware fused operations), signatures generated on one platform may differ from another when using floating-point arithmetic. These

platform-dependent floating-point behaviors can lead to implementation challenges and affect reproducibility across heterogeneous hardware environments. The determinism issue primarily affects *signing* (which uses floating-point FFT operations in the trapdoor sampler), where the same message and nonce can produce different signatures across platforms if floating-point controls are not properly enforced. Verification, by contrast, uses NTT (integer arithmetic modulo  $q$ ) which is inherently deterministic. In applications such as blockchain or consensus systems where deterministic behaviour is critical, proper implementation controls are essential for cross-platform interoperability during signing.

## 5.4 Implementation variants

Because floating-point arithmetic can introduce complexity and vulnerability, some implementations of Falcon explore fixed-point or integer-only arithmetic for sampling and FFT/NTT operations. The reference implementation [FKT<sup>+</sup>18] uses floating-point arithmetic, but alternative implementations have been developed that use fixed-point arithmetic to achieve deterministic behavior.

Several Falcon variants have been proposed to address different implementation challenges. Mitaka [EFG<sup>+</sup>21], presented at the Third PQC Standardization Conference (2021), is designed to be simpler, parallelizable, and maskable, with masking enabling protection against side-channel attacks through power analysis and other physical attacks. More recently, Falcon-M [KIB<sup>+</sup>25] (2025) simplifies the original Falcon design for resource-constrained environments such as embedded systems and IoT devices. Falcon-M removes the trapdoor mechanism used in standard Falcon key generation, instead using randomized polynomial Gaussian sampling and FFT operations. This approach reduces implementation complexity and resource requirements, though it involves trade-offs in the security model and cryptographic assumptions.

## 5.5 Resource considerations for on-chain/off-chain execution

In blockchain or constrained-device contexts (smart contracts, microcontrollers, hardware wallets), the resource footprint (stack/heap usage, memory bandwidth, FPU or DSP availability) can become a bottleneck. For instance, implementing full FFT-based sampling on a microcontroller without floating point support may be infeasible. Deploying Falcon in blockchain contexts (e.g., as transaction signatures on Ethereum) faces challenges including high gas costs for on-chain verification, primarily due to expensive hash-to-point operations and polynomial arithmetic [ZKN25, DMS<sup>+</sup>25]. Such environments may therefore tolerate only verification on-chain and require signing off-chain. Optimising memory, eliminating branching, and using efficient deterministic arithmetic become key design tasks.

In summary, while the cryptographic design of Falcon is elegant and theoretically robust, real-world implementations must carefully manage numerical precision, deterministic behaviour, and resource constraints. These engineering subtleties are integral to maintaining both correctness (that all genuine signatures verify) and security (that no side-channel or leakage vectors arise) in production systems.

# 6 Security Analysis and Cryptographic Pitfalls

Security depends on parameter alignment with best-known lattice attacks and eliminating sampler and timing leakages.

In this section we examine the security foundation of FALCON in a layered manner. First, we review the underlying hardness assumptions and catalogue the best known lattice reduction and sieving attacks against those assumptions. Next, we consider implementation-level hazards — namely side-channel leakage, numerical bias in trapdoor sampling and fault injection - that can undermine security even when theoretical assumptions hold. We then survey common fallacies around Gaussian sampling, precision limits and RNG integrity, and finally inspect how parameter selection (for example in Falcon-512 or Falcon-1024) maps to the official National Institute of Standards and Technology (NIST) security levels. Together, this analysis illustrates that proper deployment of Falcon is not just a matter of algorithmic selection but of holistic rigor — spanning mathematics, software/hardware implementation and system-level migration readiness.

## 6.1 Threat model and best-known attacks (e.g., BKZ, sieving)

The core security of FALCON rests on the hardness of lattice problems instantiated over the NTRU lattice. From an adversary’s standpoint, the two primary threat objectives are (1) key-recovery, i.e., recovering the trapdoor basis  $(f, g, F, G)$ , and (2) signature forgery, i.e., producing a valid short preimage  $(s_1, s_2)$  without the trapdoor. Both reduce to finding short vectors in or near the lattice defined by the public key  $h$ .

The most effective general-purpose attack strategy is lattice reduction via the BKZ (Block Korkine-Zolotarev) family of algorithms [SE91] combined with sieving techniques. BKZ is a lattice reduction algorithm that iteratively improves a lattice basis by finding short vectors in blocks of size  $B$  (the block size parameter). Larger block sizes  $B$  enable finding shorter vectors but require exponentially more computation. Sieving techniques complement BKZ by efficiently searching for short lattice vectors, making them practical for larger block sizes. The block size  $B$  is a key security parameter: it represents the dimension of the sub-lattices that the attacker can effectively reduce, and larger  $B$  values correspond to stronger attacks but also higher computational cost.

For instance, the Falcon specification [FKT<sup>+</sup>18] estimates that for Falcon-512 the best known attack corresponds to a BKZ block-size of  $B = 411$ . Using the "core-SVP" methodology, the authors derive classical bit-security roughly  $\lambda_C \approx 0.292 \cdot B$  and quantum bit-security  $\lambda_Q \approx 0.262 \cdot B$ . In this model, successful key recovery is feasible if

$$(B/2\pi e)^{1-n/B} \sqrt{q} \leq \sqrt{\frac{4}{3}B} \sigma_{f,g}$$

holds (see Equation (2.3) of the specification [FKT<sup>+</sup>18]) and similarly forging requires finding a lattice point of length  $\leq \beta$ , giving a forgery success condition (Equation 2.4 in [FKT<sup>+</sup>18]).

Beyond pure lattice-reduction, potential hybrid attacks (combining meet-in-the-middle guesses and lattice reduction) and algebraic attacks exploiting the ring or module structure have been considered, but for Falcon’s parameter sets these do not currently outperform the “BKZ + sieving” baseline. Thus, when selecting parameters  $n, q, \beta$ , and sampling standard deviation  $\sigma$ , the implementer must align them with the assessed cost of BKZ attacks, ensuring the “block size below break threshold” remains infeasible given current classical/quantum capabilities.

## 6.2 Side-channel vulnerabilities (timing, floating-point leakage, cache)

Although the mathematical construction of FALCON is sound against classical and quantum adversaries, real-world implementations remain vulnerable to physical side-channel attacks. These attacks exploit leakage from timing variation, floating-point operations and micro-architectural effects

(such as cache behaviour), potentially exposing sensitive internal state despite correct algorithmic behaviour.

**Timing leakage.** Conditional branching, variable-iteration loops (for example during rejection sampling), and non-constant-time arithmetic can expose private key bits via execution time measurement. For example, implementations that skip iterations when coefficients fall below certain thresholds may inadvertently create timing side channels which a remote or local adversary can exploit.

**Floating-point leakage.** The reference implementation of Falcon uses floating-point arithmetic (e.g., double precision) for FFT, LDL decomposition, and discrete Gaussian sampling. Research has demonstrated that even single-trace power analysis can extract secret key coefficients by observing operations such as "shift right" or "0 vs  $-1$ " decisions inside floating point loops [QA25]. This highlights that floating point operations - especially when combined with rejection loops and variable control flow - must be carefully hardened or converted to integer/fixed-point routines to avoid leakage.

**Cache and micro-architectural effects.** Micro-architectural features such as data caching, speculative execution, branch prediction and memory access patterns can leak information. For instance, table look-ups in Gaussian samplers or coefficient-dependent memory access during base sampling may leave observable patterns in side-channel traces. Mitigations such as constant-time lookup, masking and equal-time loops are therefore important.

**Mitigation strategies.** To manage these risks:

- Ensure constant-time implementations of sampling, compression, FFT/NTT, and norm check routines.
- Eliminate or mask floating-point operations in critical loops, or replace them with fixed-point/integer equivalents on platforms where floating-point side channels are a concern.
- Use oblivious data structures and avoid secret-dependent memory access to prevent leakage of secret information.
- Use recombination of randomness, blinding or noise insertion in signature generation when appropriate to thwart single-trace recovery. Research has demonstrated that single-trace key-recovery is possible in FALCON if sampling operations leak [QA25].

In summary, even if an implementation uses correct key-generation and signing logic, side-channel vulnerabilities pose a real threat. Deploying Falcon securely requires not only cryptographic algorithm design but also rigorous engineering discipline across timing, floating-point arithmetic, and micro-architectural layers.

### 6.3 Numerical bias and Rényi divergence in trapdoor sampling

In lattice-based signature schemes such as FALCON, the trapdoor sampler must produce vectors whose distribution is extremely close to an ideal discrete Gaussian centered at the target. Any deviation - due to numerical rounding, precision loss, or implementation shortcuts—can introduce bias which may leak information about the secret basis. One rigorous way to quantify this closeness

is via the Rényi divergence  $R_\alpha(P \parallel Q)$  between the actual sampler distribution  $P$  and the ideal Gaussian  $Q$ .

$$R_\alpha(P \parallel Q) = \frac{1}{\alpha - 1} \log \sum_x P(x)^\alpha Q(x)^{1-\alpha}.$$

The Rényi divergence is a measure of how different the actual distribution  $P$  is from the ideal distribution  $Q$ . The parameter  $\alpha > 1$  is the order of the divergence.

**High-level description.** When the sampler uses the secret trapdoor to generate a lattice vector close to the target, the output is a short vector  $s \in \Lambda$  (a lattice point). We model the ideal output distribution as

$$Q(s) \propto \exp\left(-\frac{\|s\|^2}{2\sigma^2}\right),$$

and the actual implementation output as  $P(s)$ . If  $R_\alpha(P \parallel Q)$  is small (e.g.,  $\leq 2$  or logarithmic in security), then the sampler is assured not to leak meaningful information about the trapdoor—even under composition (many signatures).

The security analysis in the Falcon specification [FKT<sup>+</sup>18] establishes bounds on the Rényi divergence between different discrete Gaussian distributions. Specifically, the analysis considers  $D_{\Lambda,\sigma,c}$ , which is a discrete Gaussian over the lattice  $\Lambda$  centered at a point  $c$  (the target), and  $D_{\Lambda,\sigma}$ , which is a discrete Gaussian over  $\Lambda$  centered at the origin. The bound

$$R_{2\lambda}(D_{\Lambda,\sigma,c} \parallel D_{\Lambda,\sigma}) \leq 1 + O(1/Q_s),$$

where  $Q_s$  is the signature-cap bound and  $\lambda$  the security parameter, establishes that sampling centered at a target  $c$  does not significantly differ from sampling centered at the origin, which is crucial for the security proof. If the numerical implementation perturbs the sampler (e.g., rounding in FFT, use of sub-normals, drift in covariance), then  $P$  may diverge from the ideal  $D_{\Lambda,\sigma,c}$ , increasing  $R_\alpha(P \parallel D_{\Lambda,\sigma,c})$  and weakening the security proof.

Insight.

- The norm threshold  $\beta$  is not only a correctness bound but also ensures that the tail of the distribution remains bounded, so that sampling outside the bound would increase divergence and risk leakage.
- Precision errors accumulate in recursive samplers like `ffSampling`, so the implementation must track the Gram–Schmidt norm tree and ensure that floating-point or fixed-point arithmetic does not bias the per-frequency sampling variance  $\sigma_k$ .
- From a system viewpoint, if the divergence  $R_\alpha$  grows above acceptable limits, an adversary may distinguish signatures produced with the trapdoor from ideal ones, potentially enabling key recovery or forging via statistical side-channels. Hence, both correctness (norm bound) and security (distribution shape) are critically tied to numerical stability.

## 6.4 Fault attacks and deterministic–randomized trade-offs

Although the mathematical design of FALCON offers strong security reductions, real-world implementation vulnerabilities remain—particularly fault injection attacks where a single defective execution can leak critical trapdoor information. For example, research has demonstrated a single-trace power attack on Falcon-512 that recovers secret key coefficients by exploiting floating-point bit-shifts during Gaussian sampling [QA25].

**Fault attack vectors.** Faults can be introduced in signing operations by inducing loop aborts, bit-flips, clock glitches or voltage perturbations. When the faulty signature deviates slightly (e.g., an early reject branch, modified rejection count, incorrect coefficient rounding), it can change the lattice-vector output in a way that becomes distinguishable - enabling the attacker to form a system of equations from pairs of faulty & correct signatures and ultimately recover the trapdoor basis.

**Deterministic vs. randomized signing trade-off.** In the standard Falcon signing protocol, each signature must start with a new, unpredictable seed  $r$  to ensure randomness in the target hash and the sampling process. Re-using the same seed or making the seed predictable undermines the statistical independence of signatures and can open the door to key-recovery or fault-analysis attacks. While a fully deterministic seed generation system might offer benefits for auditability or ledger determinism, it must be carefully designed so that  $r$  remains one-time and cryptographically secure for each message. In a blockchain context, implementers must balance the need for **fresh randomness per signing** (for security) with any requirements for determinism in transaction verification or reproducibility across nodes.

**Mitigations.** To guard against fault attacks:

- Incorporate signature consistency checks or dual-computation (e.g., sign twice and compare) to detect abnormal outputs.
- Use randomized or masked sampling seeds and blinding techniques to ensure per-signature uniqueness.
- Harden signing loops so that every path (even aborts) executes identical operations, avoiding distinguishable fault timing or iteration count.

In summary, while the theoretical framework of Falcon is robust, secure implementation demands rigorous handling of fault injection and the choice of signing mode (deterministic vs randomized) must reflect the threat model - especially in high-stakes, long-lived blockchain systems.

## 6.5 Parameter choices and NIST security levels (Falcon-512/Falcon-1024)

Table 1: Recommended parameter sets for Falcon.

|  | <b>Falcon-512</b> | <b>Falcon-1024</b> |
|--|-------------------|--------------------|
| Target NIST Level                                    | I                 | V                  |
| Ring degree $n$                                      | 512               | 1024               |
| Modulus $q$  | 12289             | 12289              |
| Standard deviation $\sigma$                          | 165.736 617 183   | 168.388 571 447    |
| $\sigma_{\min}$                                      | 1.277 833 697     | 1.298 280 334      |
| $\sigma_{\max}$                                      | 1.8205            | 1.8205             |
| Max. signature square norm $\lfloor \beta^2 \rfloor$ | 34,034,726        | 70,265,242         |
| Public key byte-length                               | 897               | 1,793              |
| Signature byte-length                                | 666               | 1,280              |

Table 2: Security estimates for Falcon parameter sets.

|  | Falcon-512 | Falcon-1024 |
|--|------------|-------------|
| BKZ block-size (key-recovery) $B$            | 458        | 936         |
| Core-SVP hardness (classical) (key-recovery) | 133        | 273         |
| Core-SVP hardness (quantum) (key-recovery)   | 121        | 248         |
| BKZ block-size (forgery) $B$                 | 411        | 952         |
| Core-SVP hardness (classical) (forgery)      | 120        | 277         |
| Core-SVP hardness (quantum) (forgery)        | 108        | 252         |

## 6.6 “Falcon+”: Toward a Formal Security Proof

The scheme Falcon has long been deployed as a compact, high-performance post-quantum signature, but until recently lacked a complete reductionist security proof in the EUF-CMA (existential unforgeability under chosen-message attack) model. The recent paper *A Closer Look at Falcon* addresses this gap by proposing a **conservative variant** (which we here term “Falcon+”) and providing the *first formal proof* of security in the Random Oracle Model (ROM). ([GJK24])

**Core idea.** The authors observe that the standard Falcon parameters and signing process do not directly fit into the classical GPV trapdoor-signature proof framework, because the required conditions on sample deviation and preimage norm do not hold as stated. In other words, simply plugging the Falcon parameters into the GPV proof would not yield the claimed 128-bit (or higher) security level. To bridge this gap, Falcon+ introduces three conservative modifications:

- **Salted hash:** The signing algorithm incorporates a fresh per-message salt in the Random-Oracle input (i.e., the seed  $r$  becomes an explicit salt), improving simulation of the adversary’s view and avoiding repeatable targets.
- **Public-key binding:** The Falcon+ design includes the public key in the hash computation, changing the target from  $H(r, m)$  to  $H(pk, r, m)$ . This ties each signature instance to its public key and prevents cross-key forgeries or replay in multi-key environments—an important step for a full EUF-CMA proof in the random-oracle model.
- **Adjusted norm bounds:** The authors slightly widen the Gaussian standard deviation and/or increase the rejection threshold to satisfy the “lossy trapdoor sampling” requirement in the proof, ensuring the pre-image distribution remains statistically close to the ideal discrete Gaussian.

In addition to these algorithmic modifications, the security proof for Falcon+ relies on a **signature-cap bound**: a tighter, explicit bound on the number of signatures an adversary may query. This is a constraint in the security model rather than a modification to the algorithm itself, but it is necessary for the reduction to handle cumulative leakage and bound rare-abort probability.

**Implications for deployment.** While Falcon+ remains extremely close to the original Falcon algorithm, these modifications impose modest practical constraints: implementers must generate and include the per-message salt, ensure norm bounds reflect the adjusted parameters, and (for the security proof to hold) respect the signature-cap bound in system policy (e.g., max queries per key). Crucially, existing Falcon signatures remain verifiable under the same public-key format, making Falcon+ a compatible upgrade path rather than a disruptive redesign.

**Summary.** In sum, Falcon+ provides a bridge between Falcon’s high-performance, compact signature appeal and the rigorous security guarantees demanded by standardization and long-term deployments. It shows that with only minimal modifications, a full EUF-CMA proof becomes feasible without severely compromising efficiency or size—thereby strengthening the confidence in Falcon’s use for systems requiring provable security assurances.

## 6.7 Falcon in Key-Recovery (KR) Mode for Blockchains

This mode utilizes a hash of the public key embedded in the signature so that the verifier can recover the original public key  $h$ . We highlight how this aligns with blockchain address-hash paradigms, discuss added assumptions (collision resistance), and evaluate system-level trade-offs (address derivation, replay protection, storage/gas cost) in ledger contexts.

**Mode description (specification).** According to the official FALCON specification (see §3.12 "A Note on the Key-Recovery Mode" [FKT<sup>+</sup>18]), the adaptation proceeds as follows:

- The public key is replaced by  $pk = H(h)$ , where  $H$  is a collision-resistant hash function.
- The signature becomes  $(s_1, s_2, r)$  with  $(s_i = \text{Compress}(s_i))$ .
- During verification, the verifier reconstructs  $h$  by computing

$$h = s_2^{-1} \left( \text{HashToPoint}(r \parallel m) - s_1 \right) \bmod (\phi, q),$$

and then checks that  $(pk = H(h))$ .

The effect is that the public-key size is extremely compact (just the hash output, typically  $(2\lambda)$  bits), while the signature grows—but the combined size  $(|pk| + |sig|)$  becomes about 15% smaller than in the classical mode.

**Blockchain relevance and project adoption.** In blockchain systems, user identifiers ("addresses") are often derived as hashes of public keys (e.g., in Bitcoin, Ethereum). This key-recovery mode fits naturally: a ledger can publish  $(pk = H(h))$  as the address, and light clients or contracts need not store the full lattice-public-key polynomial  $h$ . Although no major public blockchain is (to our knowledge) yet using the KR mode of Falcon explicitly, the mode’s design aligns well with existing blockchain address derivation patterns and could facilitate future post-quantum migrations.

### Advantages.

- Reduced on-chain public-key size: By publishing only  $(pk = H(h))$ , storage and bandwidth costs are significantly lowered—important in systems where every byte counts.
- Alignment with address-hash semantics: Many blockchain protocols already use hashed addresses; KR mode enables seamless integration of Falcon without changing account/address semantics.
- Net footprint improvement: While the signature size increases, the total of  $(|pk| + |sig|)$  often decreases 15%, making it attractive for resource-constrained chains.

## Drawbacks & security implications.

- Larger signature size: Since  $h$  is recovered rather than transmitted, the signature must carry sufficient information (notably  $s_1$  and  $s_2$ ) to allow recovery—raising transaction cost.
- More complex verification logic: the verifier must compute the modular inverse of  $s_2$ , recover  $h$  and then verify the hash—requires careful implementation to avoid side-channel or modular-inversion risks.
- Collision-resistance dependence: The scheme’s security now additionally relies on the hash function  $H$  being collision-resistant—from an adversary’s perspective finding  $h \neq h'$  such that  $H(h) = H(h')$  could undermine key uniqueness.
- On-chain implementation cost/gas: Smart contract or VM implementations must support modular inversion and lattice signature verification logic—this may incur higher gas/compute cost or require on-chain/off-chain hybrid designs.

## System-level considerations for blockchain deployment.

- Address derivation: Wallets can issue addresses as  $(Addr = H(h))$ . When verifying a transaction, the smart contract recovers  $h$ , checks  $(H(h) = Addr)$ , and then performs standard Falcon verification.
- Key rotation & replay protection: As the stored identifier is hashed, rotating keys or changing underlying  $h$  values requires updating of  $Addr$  or metadata, with clear versioning to avoid reuse or collisions.
- Gas/size trade-offs: Designers should model whether the reduced key size outweighs any increased signature/verifier cost. In some chains, high verification cost could offset the storage savings.
- Contract vs off-chain split: One architecture is to perform heavy verification off-chain and only publish succinct proof on-chain; KR mode further reduces on-chain storage footprints but must assure off-chain trust in the full verification path.

In summary, the Key-Recovery mode of Falcon offers a compelling match for blockchain ecosystems by minimizing on-chain key storage, aligning with hash-based addresses, and enabling compact footprint—yet this comes with increased verification and implementation complexity, assumptions on the hash function, and careful system engineering to avoid side-channel attacks, linkability, and replay risks in a decentralized ledger context.

## 7 Migration from ECC to PQC in Blockchain Systems

Migration demands protocol agility, hybrid periods, and careful accounting for signature sizes and verification costs across VMs.

The gradual shift from classical elliptic-curve cryptography (ECC) - specifically schemes like secp256k1 (Bitcoin, Ethereum) and Ed25519 (Solana, Cosmos) - toward post-quantum cryptography (PQC) represents one of the most technically challenging evolutions in modern blockchain infrastructure. While Falcon offers compact signatures (compared to other PQC schemes) and computationally efficient lattice-based signatures suitable for high-throughput systems, direct replacement of ECC primitives is non-trivial due to differences in algebraic structure, verification cost, and key-size

scaling. Notably, Falcon signatures (666 bytes for Falcon-512, 1280 bytes for Falcon-1024) are significantly larger than ECC signatures (typically 64–96 bytes for ECDSA/Ed25519), though they remain among the most compact post-quantum signature schemes.

This section analyses how PQC migration can proceed within blockchain systems along three coordinated dimensions:

1. **Protocol agility:** enabling consensus rules, transaction formats, and client SDKs to accommodate both ECC and PQC primitives simultaneously during a transitional period.
2. **Hybrid cryptography:** supporting dual-signature schemes (e.g., ECDSA + Falcon) so that nodes may verify both legacy and quantum-resistant signatures in the same block or transaction.
3. **Virtual machine integration:** evaluating Falcon verification under different execution environments - EVM (Ethereum), WASM (Cosmos, Polkadot), or others - and accounting for gas or compute-budget implications.

From an implementation perspective, migration to PQC implies deeper architectural changes than a mere substitution of cryptographic libraries. Transaction serialization formats, smart-contract verification logic, and node RPC interfaces must all evolve to carry PQC-compatible key structures and signatures. Additionally, performance considerations differ substantially: ECC relies on modular arithmetic over finite fields, while Falcon leverages Fast Fourier and Number-Theoretic Transforms over polynomial rings. The trade-off in migrating from ECC to Falcon involves accepting both a larger on-chain footprint (Falcon signatures are 666–1280 bytes versus 64–96 bytes for ECC) and higher arithmetic complexity (polynomial ring operations versus finite field operations) in exchange for post-quantum security guarantees.

Finally, secure migration requires careful coordination between application-layer wallets, validator nodes, and consensus protocols to prevent fragmentation. Hybrid epochs—where both ECC and PQC signatures coexist—allow smooth transition and empirical performance measurement before full deprecation of classical keys. Such hybrid deployments would require careful integration across system components, such as wallet key management, and potentially layer-2 verification mechanisms or zero-knowledge proof systems where applicable.

## 7.1 Signature/key-sizes and comparative view: ECC vs. Falcon

In order to assess the migration from elliptic-curve cryptography (ECC) to the lattice-based signature scheme Falcon, it is essential to compare the key and signature sizes and to understand how those sizes impact blockchains - especially in terms of block-size, transaction cost, and storage overhead.

**ECC baseline.** Common systems rely on schemes like EdDSA or ECDSA (e.g., over secp256k1). For example, an Ed25519 signature is typically 64 bytes long and the public key 32 bytes. These compact sizes make ECC highly efficient for high-throughput blockchain transactions, low storage, and minimal gas footprint.

**Falcon’s size characteristics.** By contrast, Falcon’s parameter sets (e.g., Falcon-512) yield a public key of 897 bytes and a signature of 666 bytes. The higher-security variant (Falcon-1024) uses 1793 bytes for the public key and 1280 bytes for the signature. While still significantly smaller than many other post-quantum signature alternatives, these sizes remain much larger than typical ECC artifacts.

**Impact on blockchains: block size & gas cost.** In blockchain deployments, every additional byte per transaction translates into higher block size, greater propagation latency, increased storage demand for nodes and elevated gas or fee cost. A signature size increase from 64 bytes to 666 bytes implies roughly a 10x overhead for that component alone. Furthermore, including an 897 byte public key in, for example, a smart contract wallet or multi-signature scheme, demands more state storage and greater immutable ledger cost. Therefore, when adopting Falcon in place of ECC, it is crucial to model the trade-off: the benefits of quantum-resistance must be weighed against the increased bytes per transaction, the corresponding fee or gas cost, and how that influences node resource requirements and network throughput.

### Summary of comparative trade-offs.

- ECC (EdDSA/ECDSA) provides minimal sizes, low overhead, and excellent performance in current systems—but lacks resilience against quantum adversaries.
- Falcon offers post-quantum security and remains one of the most size-efficient lattice schemes—but its keys and signatures are still significantly larger than classical ECC.
- For blockchains, a careful sizing analysis—estimating bytes per signature, keys stored in contract state, propagation cost and fee impact—is indispensable when planning PQC migration.

## 7.2 Verification cost: modular arithmetic vs. FFT-based paths

Verification cost is a key consideration when profiling signature schemes for blockchain systems. Classical elliptic-curve schemes such as ECDSA on secp256k1 or EdDSA on Ed25519 leverage modular arithmetic over prime fields and well-optimized point-scalar operations. These operations are highly mature and typically execute in tens of microseconds on modern hardware.

By contrast, the lattice-based scheme Falcon shifts the burden into polynomial arithmetic (ring convolution) and employs Fast Fourier Transform (FFT) or Number Theoretic Transform (NTT) techniques to achieve efficient verification. Implementations using FFT/NTT achieve  $O(n \log n)$  complexity compared to  $O(n^2)$  for naive polynomial multiplication, where  $n = 512$  for Falcon-512.

In such implementations, the core verification steps are: (1) hash-to-point, (2) decompress  $s_2$ , (3) compute  $s_1 = c - h \cdot s_2 \bmod q$ , (4) norm check  $\|(s_1, s_2)\|^2 \leq \beta^2$ . Because steps (2) and (3) involve polynomial-multiplication in  $R_q = \mathbb{Z}_q[x]/(x^n + 1)$ , FFT/NTT acceleration is critical to maintain practical throughput.

**Signing cost considerations.** While verification cost is critical for nodes processing transactions, signing cost is equally important for validators who must sign blocks. Falcon signing involves discrete Gaussian sampling over the trapdoor basis, which is computationally more expensive than ECC signing. ECC signing (e.g., ECDSA) requires one scalar multiplication, typically completing in hundreds of microseconds. Falcon signing, by contrast, involves FFT-based trapdoor sampling with rejection loops, typically requiring several milliseconds. This signing overhead can impact validator performance, especially in BFT-based consensus protocols where validators must sign multiple messages per consensus round (e.g., prevotes and precommits in Tendermint-style systems), or in other scenarios requiring high signature throughput. However, the signing cost is generally acceptable for most blockchain use cases, and the post-quantum security benefits justify the additional computational cost.

**Modular arithmetic (ECC) path.** In ECC verification, the main cost comes from one or more elliptic-curve point multiplications and associated modular reductions. With hardware support and optimized libraries, this cost is small and well understood. Resources such as pre-computations, scalar decomposition (e.g., windowed methods) and side-channel hardened libraries make verification efficient and constant-time.

**FFT-based path (Falcon).** Falcon verification shifts cost into polynomial multiplication and vector norms. Although the public-key  $h$  remains fixed and many operations are relatively light, the ring arithmetic requires performing an inverse FFT or NTT (depending on implementation) followed by coefficient-domain norm checking. Falcon remains compact in signature size, though performance characteristics depend on the implementation platform and optimization level. In highly-parallel or hardware-accelerated settings the FFT/NTT path scales well: the convolution cost becomes  $O(n \log n)$  rather than  $O(n^2)$  for naive multiplication, enabling efficient parallel execution and benefiting from hardware acceleration.

**Blockchain deployment implications.** For blockchain-style VMs (e.g., EVM, WASM), several considerations emerge:

- **Verification performance:** Performance benchmarks show that Falcon verification can be competitive with or even faster than ECC verification on many platforms, though results vary significantly depending on the implementation, hardware, and optimization level. While FFT/NTT operations are complex, optimized implementations can achieve verification times comparable to ECC. The actual gas/compute cost in smart contract VMs depends on the VM’s instruction set and optimization of the cryptographic primitives.
- **Signing performance for validators:** Validators must sign frequently, especially in protocols requiring multiple signing rounds (e.g., BFT-like consensus). Falcon signing is computationally more expensive than ECC signing (typically several milliseconds versus hundreds of microseconds). While this gap could theoretically affect validator throughput in high-frequency scenarios, the practical impact depends on consensus mechanism, network latency, and parallelization opportunities.
- **Signature aggregation:** While Falcon does not support native signature aggregation like BLS, recent research has explored aggregating Falcon signatures through succinct proof systems, such as LaBRADOR [AAB<sup>+</sup>24], where a proof attests to the validity of multiple signatures. The FFT/NTT structure may also enable more efficient batch verification where shared computations reduce amortized verification costs.

**Performance benchmarks.** Performance comparisons between Falcon and ECC are highly platform-dependent. On modern CPUs with optimized implementations, Falcon-512 verification can achieve speeds comparable to or faster than ECDSA/Ed25519 verification. For example, on ARMv8-A processors, optimized Falcon implementations have demonstrated verification speeds 3–3.9 times faster than CRYSTALS-Dilithium, and competitive with ECC schemes [NG23]. However, on embedded platforms or in smart contract VMs, the relative performance depends heavily on the available instruction set, hardware acceleration, and implementation optimizations. Signing performance shows a clearer gap: Falcon signing typically requires several milliseconds, while ECC signing completes in hundreds of microseconds. For blockchain validators in high-throughput scenarios or BFT consensus protocols requiring multiple signing rounds, this signing overhead could

theoretically impact performance, though empirical evaluation in real-world consensus systems would be needed to quantify practical effects.

In summary, Falcon’s FFT/NTT based verification path offers strong post-quantum resilience while maintaining practical performance, with verification often competitive with ECC. The key considerations are evaluating the target platform (smart-contract VM vs dedicated hardware), accounting for signing costs in validator operations, and ensuring that resources and gas budgets align with the arithmetic complexity.

### 7.3 Smart contract integration models (EVM, WASM)

Integrating post-quantum signature schemes such as Falcon into smart contract environments requires careful consideration of the host execution model (e.g., Ethereum Virtual Machine (EVM), WebAssembly (WASM) runtime, or others). Each platform offers distinct trade-offs in opcode support, gas or compute metering, memory/storage costs and abstraction flexibility.

**EVM-compatible chains.** The EVM is heavily metered by gas and optimized for classical elliptic-curve cryptography. Currently, EVM precompiles support ECDSA (secp256k1 curve) and BLS12-381 curve operations, but not Ed25519 or post-quantum schemes. Implementing Falcon verification natively on EVM presents challenges: larger signature and key sizes inflate calldata and storage, while lattice-based convolution/FFT arithmetic demands more op-codes or pre-compiled contracts. Experimental integrations of Falcon verification into EVM-compatible environments have demonstrated that the transformation overhead remains non-trivial, with pure Solidity implementations requiring approximately 1.8–7 million gas depending on the variant [ZKN25]. To avoid high gas costs, Falcon verification would need to either happen off-chain, or chains would need to implement a dedicated precompile. A draft EIP-8052 [DMS<sup>+</sup>25] proposes standardized precompiled contracts for Falcon-512 verification targeting approximately 3,000 gas per verification.

**WASM runtimes.** Chains built on WASM (e.g., Polkadot Runtime, CosmWasm-enabled chains) afford more flexibility: custom libraries and native memory control allow optimized NTT routines, and runtime metering can better adapt to heavier arithmetic. However, deploying Falcon verification in WASM presents several challenges. WASM bytecode size limits (typically 512KB–2MB depending on the chain) may constrain the inclusion of Falcon verification code, which includes NTT routines, polynomial arithmetic, and hash-to-point functions. Additionally, deploying Falcon verification in WASM requires careful memory/cycle budgeting and gas-equivalent metering aligned with the chain’s economics.

**Integration design strategies.** Some practical strategies include:

- Use of a specialized **pre-compiled contract** or native opcode to verify Falcon signatures efficiently, thereby reducing gas overhead in EVM chains (see EIP-8052 [DMS<sup>+</sup>25]).
- Deployment of hybrid off-chain/on-chain verification, where signature validity is checked off-chain and only a succinct proof or status bit is posted on-chain.
- Support for account abstraction (e.g., via ERC-4337 style wallets) to allow wallets or contracts to switch from ECC to PQC (Falcon) without protocol-level hard fork. On Ethereum, this approach has been explored to enable Falcon-based smart contract wallets [San25], though a limitation remains: while individual user operations can use Falcon signatures, the bundler

transaction submitting operations to the mempool still relies on ECDSA, requiring protocol-level changes for complete ECDSA removal.

### Key considerations and trade-offs.

- Calldata and storage cost: Falcon’s larger signature and key sizes increase transaction/data payload: smart-contract gas consumption must reflect this.
- Compute and gas metering: Lattice verification uses transforms (FFT/NTT) or heavy modular arithmetic, which may consume more cycles—smart-contract platforms must either increase gas allowance or provide acceleration.
- Upgrade path & compatibility: To maintain backward compatibility and migration flexibility, contracts should support multiple signature schemes (ECC and Falcon) and provide a versioning mechanism.
- Security and deterministic execution: Smart contracts must avoid side-channel vulnerabilities (timing, memory access) and support constant-time lattice arithmetic where applicable—even in resource-metered environments.

In summary, while integrating Falcon into smart contract platforms (EVM, WASM) is technically feasible, it demands careful engineering of gas/compute budgets, memory/storage footprint and execution determinism. Adopting dedicated verification paths, account abstraction or roll-up architectures can significantly mitigate overhead and smooth the transition to quantum-safe contract logic.

## 7.4 Hybrid cryptography: transitional dual-signature (ECDSA + Falcon)

During migration from classical elliptic-curve cryptography (ECC) to post-quantum schemes such as FALCON, a hybrid dual-signature strategy - where a transaction is signed using both a traditional ECC algorithm (e.g., ECDSA or EdDSA) and a post-quantum signature (Falcon) - offers a practical interim path.

**What it does.** By attaching two signatures to a transaction—one under the mature ECC primitive and one under Falcon, a system ensures that the transaction remains valid even if one algorithm becomes broken. The hybrid model therefore provides *quantum-resilient fallback* while preserving compatibility with existing ECC-only nodes or legacy infrastructure.

### Why it’s useful.

- Backward compatibility: Legacy systems that expect ECC signatures continue to operate without disruption since the ECC layer remains intact.
- Future-proofing: The added Falcon signature guards against future quantum adversaries, enabling a gradual transition rather than abrupt cut-over.
- Auditability and phased roll-out: Wallets and nodes can gradually upgrade to support Falcon verification while still accepting ECC, reducing risk of consensus split.

### Trade-offs and considerations.

- Increased transaction size and cost: Dual-signatures significantly increase the number of signature bytes (Falcon-512 signatures are 666 bytes versus 64–96 bytes for ECC), impacting block size, bandwidth and gas/fee budgets.
- Verification complexity: Validators must verify two signatures (ECC + Falcon), which increases processing time and may impact throughput or latency.
- Key management complexity: Wallets now manage two distinct key pairs and signing routines, increasing implementation and operational overhead.

**Security implications.** The hybrid model assumes at least one of the two algorithms remains secure; thus breaking both simultaneously is required for compromise. In practice, the risk is reduced by the orthogonal hardness assumptions (discrete log for ECC, lattice problems for Falcon). However, attention must be paid to implementation consistency: for example, ensuring both signatures are included (no selective omission) and that a broken ECC alone doesn't lead to fallback to a weak mode.

**Blockchain deployment context.** For blockchain systems, hybrid signature schemes are especially relevant where long-lived assets or high-value accounts demand quantum-resilience without disrupting current consensus rules. Protocols may specify that new accounts require dual-signatures, while legacy ones remain ECC-only until a later epoch.

In summary, the transitional dual-signature strategy (ECDSA + Falcon) provides a practical, low-disruption migration path to PQC for blockchain ecosystems—balancing compatibility, security and operational risk—yet requires careful sizing, performance budgeting and key-management discipline.

## 7.5 PQ wallets and multisig: UX and threshold considerations

As blockchain users and institutions transition to post-quantum cryptography (PQC), wallet and multisignature (multisig) system design must evolve to preserve usability, security and threshold trust without compromising user experience or operational reliability.

**Wallets for PQC.** Post-quantum (PQ) wallets must support new key types, larger public keys and signatures (for example from Falcon), and seamless key-rotation mechanisms. While users expect familiar workflows - seed-phrase backup, address derivation, multisig support - the increase in key/signature size and change in algorithmic profile introduce user-experience and engineering challenges (larger backups, slower sign operations, more device memory).

**Multisig and threshold schemes in a PQ era.** Multisignature setups (e.g.,  $m$ -of- $n$  schemes) are commonly used in custodial, institutional and high-value wallets. While classical ECC/Schnorr-based multisig and threshold signing are mature, Falcon-based (or other lattice-signature based) threshold/multisig protocols remain largely at the research stage, with no widely adopted production solution today. Surveys emphasise that lattice-based TSS/MPC schemes face obstacles such as key-aggregation, efficient non-interactive signing, and suitable proof frameworks. For example, an institutional wallet contemplating a 2-of-3 multisig with Falcon keys must account for distributed key generation, much larger public-key/signature exchange, bespoke firmware/hardware for larger operations, and fallback paths where classical keys remain active until full PQ readiness.

## UX and operational considerations.

- **Key-size management:** Larger public keys and signatures impact storage (wallet state, device memory) and user workflows (backup transfer, address display).
- **Threshold signing latency:** Larger signature generation and multi-party coordination (MPC) may slow down multisig workflows (e.g., hardware wallets or air-gapped signing zones).
- **Backup and recovery:** Recovery solutions must support PQ key types—including fallback to classical keys during transition and clear migration paths.
- **Versioning and migration:** Wallets should clearly indicate key-type (classical vs PQ), support automatic rotation, and optionally act in hybrid mode until PQ threshold schemes mature.

**Security and deployment implications.** Transitioning to PQ multisig with Falcon implies not only defending against quantum attacks, but also handling increased coordination, key-sharing complexity, and novel implementation surface (larger keys, heavier MPK/MKS protocols). Here, MPK (Multi-Party Key generation) refers to protocols where multiple parties collaborate to generate a shared cryptographic key without any single party knowing the full key, while MKS (Multi-Party Key Signing) refers to protocols where multiple parties collaborate to produce a signature without any single party knowing the full secret key. Because mature production-grade Falcon threshold schemes are not yet available, institutions must proceed cautiously - often retaining classical multisig alongside PQ keys until the research and ecosystem mature.

**Summary.** In short, as wallet and multisig systems adopt PQC and potentially schemes like Falcon, their design must balance larger key/signature sizes and heavier operations with familiar user experience, threshold trust models, and robust recovery. Importantly: Falcon-based threshold multisig is still experimental, not yet production-proven. Planning for future adoption is wise, but definitive deployment should wait until the ecosystem, libraries and proof frameworks are more mature.

## 8 Practical Recommendations for Secure Implementation

Secure deployment requires hardened samplers, deterministic builds, and precision conformance tests as part of CI.

Implementations of FALCON (or any lattice-based post-quantum signature) must move beyond theory and handle elevated engineering risk: from Gaussian-sampler integrity to side-channel resistance, floating-point determinism and large key/signature handling. The recommendations below form a concise checklist suited for production-grade deployment.

### 8.1 Key Practical Controls

**Secure RNG and Gaussian sampler design.** Ensure all random seeds (for key generation, signing) are derived from a cryptographically secure source and properly refreshed. The discrete Gaussian sampler used in Falcon is a critical attack surface: recent single-trace attacks demonstrate full key recovery in embedded microcontrollers when sampler operations leak [QA25].

Recommended actions include: use masked or hardware-randomized sampling, avoid predictable reuse of internal states, and monitor for abnormal sampler behaviour or excessive rejection loops.

However, logging or exposing rejection counts or detailed sampler behavior can leak information about the secret trapdoor basis, as the rejection rate depends on the Gram–Schmidt norms of the secret basis vectors. If monitoring is necessary for fault detection, it should be done using threshold-based alerts for extreme anomalies (e.g., rejection counts far outside expected bounds) without logging exact values, or by aggregating statistics over many signatures to obscure individual patterns.

**Avoiding side-channels in FFT/NTT pipelines.** Lattice operations in Falcon rely on FFT/NTT arithmetic that can leak via timing, memory access, or power profiles. To mitigate: implement constant-time arithmetic, unify memory access patterns across branches, disable subnormal floating numbers or hardware features with variable latency, and conduct side-channel testing (e.g., EM, power) especially for embedded devices.

**Deterministic vs. randomized signing in blockchain contexts.** While reproducible signatures enhance auditability, in multi-signature, multi-wallet or ledger ecosystems, reusing seeds or predictable signing patterns opens risk of leakage or replay. Use fresh seed per signing, track signature counters, and integrate signing versioning. In chain environments, include version or algorithm identifiers in transaction metadata for future migration flexibility.

**Precision audit checklist (floating-point conformance tests).** Because many Falcon operations depend on floating-point arithmetic (e.g., basis reduction, sampling transforms), implementers must validate that all builds (x86, ARM, embedded) compute identical results to specification test vectors. Maintain CI tests that:

- Compare computed norms and Gram–Schmidt values across platforms.
- Verify that sampling distributions meet statistical bounds (via offline audit).
- Check that threshold rejection probabilities remain within expected ranges.

Include automated alerting for floating-point drift, library changes or compiler optimizations that alter numeric behaviour.

**Reference implementations and validation tools.** Leverage and test against reference libraries (e.g., NIST, PQCRYPTO, open-source implementations) that include conformance test vectors, side-channel guidance and build-variants (floating-point vs fixed-point). Use fuzzing on signing routines, build multiple architecture binaries, and integrate hardware/firmware tests (especially for embedded, smart-card or wallet devices). Maintain supply-chain controls around vendor cryptographic libraries.

**Summary.** Production-ready deployment of Falcon (or any post-quantum signature) is not just an algorithm-choice; it is a full systems-engineering task: secure entropy, sampler hardening, numeric determinism, side-channel resilience, multi-architecture reproducibility and rigorous tests must all be baked into CI/CD and build-processes. Implementers who scaffold these controls up front reduce risk of subtle leaks or mismatches that might invalidate long-lived system trust.

## 9 Future Directions

Falcon can be adapted to constrained devices, integrated into newer proof systems, and co-exist with ECC during long PQC transitions.

This section outlines three key trajectories for the evolution of FALCON and its ecosystem: (i) adaptation to resource-constrained and hardware-accelerated environments, (ii) integration into broader cryptographic frameworks such as zero-knowledge or hybrid lattice/ECC systems, and (iii) standardization, interoperability and migration coexistence challenges. These directions capture both ongoing research and practical system-engineering pathways for long-term quantum-resilience.

### 9.1 Variants for constrained or hardware-accelerated environments

Recent work highlights that although Falcon offers compact signature sizes (compared to other PQC schemes), its full key-generation and signing routines (e.g., FFT/NTT, trapdoor sampling) remain resource-intensive for low-power or embedded devices [FKT<sup>+</sup>18]. Simplified algorithmic variants address these constraints by reducing signing complexity while enabling better parallelization and easier masking for side-channel protection. Other work has explored fixed-point or integer-only arithmetic implementations to reduce memory and computation requirements for IoT contexts, though these may require higher word-widths and more careful overflow control. The suitability of FFT/NTT operations for parallel hardware execution suggests potential benefits from dedicated acceleration units in ASIC or FPGA implementations, though practical deployment would require evaluation of implementation costs and performance trade-offs.

### 9.2 Prospects for hybrid lattice - ECC coexistence

Given the long migration horizon for many systems, Falcon may co-exist with traditional ECC signatures via hybrid or dual-scheme deployments. This approach allows systems to support both ECC and Falcon in parallel, enabling gradual adoption while preserving compatibility with existing infrastructure. The hybrid model provides quantum-resilient fallback while maintaining backward compatibility with ECC-only nodes or legacy systems, as discussed in Section 7.

Successful migration will require several key components: (a) clear versioning mechanisms to distinguish between signature schemes, (b) dual-signature support in wallets, validators, and smart contracts, (c) backwards compatibility layers that allow legacy systems to continue operating, and (d) well-defined migration timelines that give stakeholders sufficient time to upgrade. The trade-offs of hybrid deployments include increased transaction size (as discussed in Section 7), higher verification complexity, and more complex key management, but these costs are often justified by the security benefits and migration flexibility they provide.

### 9.3 Standardization and interoperability challenges

Although Falcon has been selected in the NIST PQC process [CSR22], full ecosystem maturity requires addressing several standardization and interoperability challenges. These include: (a) standardized implementation profiles that specify whether implementations should use floating-point or fixed-point arithmetic, and how to ensure deterministic behavior across platforms, (b) inter-library compatibility across different programming languages and platforms to enable seamless integration, (c) standardized key-formats and verification modules for smart-contract environments (EVM, WASM) and embedded contexts, and (d) migration frameworks for blockchain systems including hybrid epochs, address-type upgrades, and protocol-level support for post-quantum signatures.

The NIST standardization process is addressing some of these challenges, but additional work is needed in areas such as smart contract integration, embedded device optimization, and blockchain-specific adaptations. For example, the development of pre-compiled contracts for EVM chains, optimized WASM modules, and standardized key-recovery modes for blockchain address systems are

active areas of research and development. All these efforts are essential to ensure seamless adoption of Falcon in real-world systems.

**Summary.** The long-term viability of Falcon depends not just on cryptographic soundness, but on ecosystem readiness: hardware support, mixed-scheme interoperability, standards adherence and practical migration road-maps. Addressing these future directions—through continued research on resource-efficient variants, hardware acceleration, hybrid deployment strategies, and standardization efforts—will make Falcon not only a strong candidate today but a deployable cornerstone of quantum-safe infrastructure tomorrow.

## 10 Conclusion

FALCON is deployment-ready with caveats: precision, side-channels, and migration choreography determine practical security.

### 10.1 Summary of Falcon’s cryptographic position

The signature scheme FALCON - built on NTRU lattices and fast Fourier trapdoor sampling - is one of the leading post-quantum digital signatures selected by NIST. It delivers compact public-keys and signatures compared with most lattice-based alternatives, fast verification, and a strong security foundation against both classical and quantum adversaries. That said, the practical security of Falcon depends critically on correct implementation: floating-point precision, correct Gaussian sampling, and side-channel resistance remain non-trivial engineering challenges.

### 10.2 Deployment readiness in blockchain infrastructures

In blockchain ecosystems—where high throughput, deterministic verification, small data footprint, and long-term asset security matter—Falcon offers a credible post-quantum upgrade path. For example, Algorand has integrated Falcon signatures for signing State Proofs, which are commitments to the chain’s state on the latest 256 rounds, demonstrating real-world deployment of Falcon in blockchain contexts [Fou24]. Additionally, Algorand introduced the `falcon_verify` opcode in its AVM (Algorand Virtual Machine) as part of the v4.3.0 consensus upgrade released in September 2024, enabling on-chain Falcon signature verification in smart contracts [Alg24]. However, readiness is not solely about algorithm choice. Practitioners must address key implementation vectors: deterministic builds across architectures, constant-time arithmetic, signature size/gas cost modeling, and migration strategy (hybrid classical + PQC). Only with this holistic engineering posture can Falcon be safely integrated into ledger systems that are intended to operate securely for decades.

### 10.3 Open research and standardization paths

Despite its many strengths, Falcon still faces open issues before ubiquitous adoption:

- Precision and numerical consistency: For example, the floating-point components in Falcon create cross-platform reproducibility risks.
- Certification and ecosystem tooling: While Falcon is under standardization, full reference profiles (constant-time, fixed-point variants, embedded implementations) still require maturation.

- Migration complexity: Blockchains must plan for long transition horizons, backward compatibility, hybrid signature schemes, and potential threshold/multisig extensions—all of which require further development.
- Implementation verification and auditing: Hardware wallets, embedded systems and smart-contract verifiers must be rigorously audited for side-channel resilience, RNG integrity and correct trapdoor sampling.

Addressing these gaps will boost confidence in Falcon’s long-term durability and interoperability.

In closing: Falcon is no longer a cryptographic experiment, it is a real candidate for quantum-resilient digital signatures. But turning that candidate into secure real-world deployment demands engineering diligence, careful migration planning and ecosystem maturity. When startups, institutions and blockchain protocols treat implementation hygiene as first-order priorities, they can place Falcon at the core of a quantum-safe infrastructure for the next generation of digital systems.

## A Derivation of Falcon’s NTRU equation and its inverses

This appendix presents a step-by-step derivation of the NTRU equation used in the Falcon signature scheme, and explains how the trapdoor basis inversion is constructed. We follow closely the Falcon specification [FKT<sup>+</sup>18].

### A. The underlying NTRU lattice construction

Let  $R = \mathbb{Z}[x]/(\phi(x))$  where  $\phi(x) = x^n + 1$  (for  $n$  a power of 2). Choose a modulus  $q \in \mathbb{Z}$ . We sample two “small” polynomials

$$f, g \in R,$$

such that  $f$  is invertible modulo  $q$  in  $R$ . Then define the public key

$$h \equiv g \cdot f^{-1} \pmod{(q, \phi(x))}.$$

The corresponding NTRU lattice  $\Lambda_h \subset R^2$  is generated by the rows of

$$B = \begin{pmatrix} g & -f \\ G & -F \end{pmatrix}$$

where  $(f, g, F, G)$  are chosen so that

$$fG - gF = q$$

holds in the ring  $R = \mathbb{Z}[x]/(\phi(x))$ , or equivalently  $fG - gF \equiv q \pmod{\phi(x)}$  in  $\mathbb{Z}[x]$ . Note that when reduced modulo  $q$ , this equation becomes  $fG - gF \equiv 0 \pmod{q}$  in  $R_q = \mathbb{Z}_q[x]/(\phi(x))$ . This equation is the core NTRU equation of the Falcon key-generation.

### B. Why the equation implies a valid lattice trapdoor

Observe that from the NTRU equation

$$fG - gF = q$$

in the ring  $R = \mathbb{Z}[x]/(\phi(x))$ , we can write the 2-by-2 block matrix relation

$$\begin{pmatrix} f & g \\ F & G \end{pmatrix} \begin{pmatrix} G & -g \\ -F & f \end{pmatrix} = \begin{pmatrix} fG - gF & -fg + gf \\ FG - GF & -Fg + Gf \end{pmatrix} = \begin{pmatrix} q & 0 \\ 0 & q \end{pmatrix} = q \cdot I_2$$

in the ring  $R$ , where we use the NTRU equation  $fG - gF = q$  and note that  $FG - GF = 0$  (since multiplication in  $R$  is commutative). More precisely, the row space of the full basis matrix

$$B = \begin{pmatrix} g & -f \\ G & -F \end{pmatrix}$$

satisfies  $\det(B) = g(-F) - (-f)G = fG - gF = q$ , which gives a full-rank lattice of covolume  $q$ . Because  $f, g$  are short and known, the secret basis is well-conditioned (nearly orthogonal, small Gram-Schmidt norms) while the public key lattice induced by  $h$  corresponds to the “bad” basis. This is consistent with the trapdoor construction of the GPV framework.

### C. Public-key relation and inversion

Given the polynomials  $f$  and  $g$ , if  $f$  is invertible modulo  $(q, \phi)$ , one defines

$$h \equiv g f^{-1} \pmod{(q, \phi(x))}.$$

Then the verification equation for a signature vector  $(s_1, s_2) \in R^2$  reduces to checking

$$s_1 + h s_2 \equiv c \pmod{(q, \phi(x))}$$

for the challenge polynomial  $c$ .

The NTRU lattice  $\Lambda_h$  consists of all vectors  $(u, v) \in R^2$  such that  $u + h v \equiv 0 \pmod{(q, \phi(x))}$ . For a signature  $(s_1, s_2)$  satisfying the verification equation  $s_1 + h s_2 \equiv c$ , we have  $(s_1 - c) + h s_2 \equiv 0 \pmod{(q, \phi(x))}$ , which means the vector  $(s_1 - c, s_2)$  belongs to the lattice  $\Lambda_h$ .

Since the basis matrix  $B = \begin{pmatrix} g & -f \\ G & -F \end{pmatrix}$  generates  $\Lambda_h$ , the vector  $(s_1 - c, s_2)$  can be expressed as an integer linear combination of the basis vectors. This establishes that valid signatures correspond to short lattice vectors (after adjusting for the challenge  $c$ ), which links the signing equation to the lattice structure and underpins the correctness of the scheme: any signature  $(s_1, s_2)$  that satisfies the verification equation must correspond to a lattice vector, and the norm bound ensures it is short.

### D. Summary of key relations

- Secret polynomials  $f, g$  small  $\rightarrow$  trapdoor basis is short.
- NTRU equation  $fG - gF = q$   $\rightarrow$  lattice has covolume  $q$  and public lattice mod  $q$  corresponds to  $h = g f^{-1}$ .
- Verification checks  $s_1 + h s_2 = c \pmod{(q, \phi)}$  which ensures the signature lies in the lattice and is short.

Together, these derivations show how the Falcon scheme instantiates the GPV trapdoor-signature framework over an NTRU lattice, with the key inversion and verification relations derived cleanly from the chosen polynomial equations.

## B Step-by-step proof of matrix–lattice equivalence

**Equivalence of public and secret row lattices.** Let  $R = \mathbb{Z}[x]/(\phi)$  with  $\phi = x^n + 1$  for  $n = 2^\kappa$ , and let  $f, g, F, G \in R$  satisfy the NTRU relations

$$fG - gF = q \quad \text{and} \quad h \equiv g f^{-1} \pmod{q}.$$

Define the two  $2 \times 2$  matrices over  $R$  (acting on row vectors)

$$B_h = \begin{pmatrix} 1 & h \\ 0 & q \end{pmatrix}, \quad B_{f,g;F,G} = \begin{pmatrix} f & g \\ F & G \end{pmatrix}.$$

**Lemma 1.** *The row lattices (row  $R$ -modules) generated by  $B_h$  and  $B_{f,g;F,G}$  in  $R^2$  are equal. Consequently, after the coefficient embedding  $R^2 \hookrightarrow \mathbb{Z}^{2n}$ , they define the same  $\mathbb{Z}$ -lattice in  $\mathbb{Z}^{2n}$ .*

*Proof.* Consider the matrix

$$T = \begin{pmatrix} f & \frac{g - fh}{q} \\ F & \frac{G - Fh}{q} \end{pmatrix} \in M_2(R).$$

This is well-defined in  $R$  because both numerators are divisible by  $q$ : indeed, from  $h \equiv gf^{-1} \pmod{q}$  we get  $g - fh \equiv 0 \pmod{q}$ , and multiplying the same congruence by  $F$  and using  $fG - gF \equiv 0 \pmod{q}$  yields  $G - Fh \equiv 0 \pmod{q}$ .

Compute

$$TB_h = \begin{pmatrix} f & \frac{g - fh}{q} \\ F & \frac{G - Fh}{q} \end{pmatrix} \begin{pmatrix} 1 & h \\ 0 & q \end{pmatrix} = \begin{pmatrix} f & fh + (g - fh) \\ F & Fh + (G - Fh) \end{pmatrix} = \begin{pmatrix} f & g \\ F & G \end{pmatrix} = B_{f,g;F,G}.$$

Thus every row of  $B_{f,g;F,G}$  is an  $R$ -linear combination of the rows of  $B_h$ .

Moreover,  $T$  is unimodular in  $R$ :

$$\det T = f \cdot \frac{G - Fh}{q} - F \cdot \frac{g - fh}{q} = \frac{fG - fFh - Fg + Ffh}{q} = \frac{fG - gF}{q} = 1.$$

Hence  $T$  is invertible over  $R$  with  $T^{-1} \in M_2(R)$ , and therefore  $B_h = T^{-1}B_{f,g;F,G}$ . It follows that each row of  $B_h$  is an  $R$ -linear combination of the rows of  $B_{f,g;F,G}$  as well. Consequently, the two matrices generate the same row lattice in  $R^2$ , and—under the standard coefficient embedding—the same  $\mathbb{Z}$ -lattice in  $\mathbb{Z}^{2n}$ .  $\square$

## C FFT/NTT numerical validation

This appendix presents recommended practices and example results for the numerical validation of the FALCON signature scheme’s polynomial-arithmetic kernels — specifically the Fast Fourier Transform (FFT) over the complex field and the Number Theoretic Transform (NTT) over finite rings  $(\text{mod } q)$ . Correct and consistent implementation of these transforms is essential to ensuring correctness, deterministic behaviour and resistance to subtle numeric and side-channel flaws.

### AA. Why numerical validation is required

The core signing and verification algorithms of Falcon use transforms of the polynomial ring  $R = \mathbb{Z}[x]/(x^n + 1)$ . The specification calls for both:

- An FFT (floating-point or complex) version for trapdoor sampling and recurse basis operations.

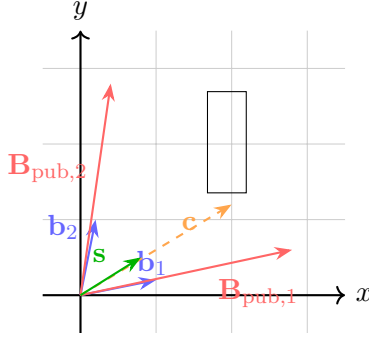


Figure 2: Geometric interpretation of the lattice basis construction in the FALCON signature scheme. The blue vectors represent the secret (trapdoor) basis, the red vectors represent the public basis (skewed), the orange dashed arrow is the target vector  $\mathbf{c}$ , and the green arrow is the short vector  $\mathbf{s}$  satisfying  $\mathbf{s} \cdot B_{\text{pub}} = \mathbf{c}$ .

- An NTT (integer modular, modulo a prime or  $q$ -friendly modulus) version for fast public-key and verification operations.

Errors in these transforms—such as incorrect twiddle-factor tables, non-uniform rounding, inconsistent bit-reverse ordering, or platform-specific floating-point rounding modes—can lead to signature failures, increased rejection rates or even key-leakage. Therefore, “validation” means confirming three things: (i) consistency across build/architecture (ii) correct transform/inverse composition behaviour (iii) statistical correctness of any sampler reliant on the transform domain.

## BB. Key validation checks

Here are recommended checks for implementation teams:

- 1. Transform/inverse round-trip.** For a suitably random polynomial  $f \in R$ , verify

$$\text{IFFT}(\text{FFT}(f)) = f$$

(with small acceptable error for floating-point versions). For NTT versions:

$$\text{INTT}(\text{NTT}(f)) \equiv f \pmod{()q, x^n + 1}.$$

Any mismatch outside bounds indicates a kernel error or wrong ordering/twiddle table.

- 2. Norm-preservation (where relevant).** In many algorithms the transforms preserve Euclidean norm up to a constant scaling:

$$\|f\|_2^2 \approx \frac{1}{n} \sum_j |f(\zeta_j)|^2$$

in the FFT case. Implementers should benchmark across transforms that the norm-change factor remains within a tight tolerance (e.g., +/- 0.1 %). Deviations may introduce sampler bias.

- 3. Rejection-rate and sampler drift profiling.** Because the lattice sampler uses transform domain operations (especially in the trapdoor sampler via FFT) the implementation should record rejection and norm-check statistics over many signings. If the rate drifts over time or across platforms, it suggests precision skew or platform-specific rounding. This is documented in side-channel research.

**4. Cross-platform deterministic equality.** For builds targeting different architectures (x86-64, ARM, embedded), the same input polynomial and RNG seed should produce identical post-transform results (within defined tolerance) for floating-point kernels. Any divergence should be flagged in CI.

### CC. Example parameter test table

| Transform type     | Mean round-trip error | Max error | Build divergence            |
|--------------------|-----------------------|-----------|-----------------------------|
| FFT (double)       | 2.3 e-15              | 1.2 e-14  | No divergence               |
| NTT (mod q) n=512  | exact match           | 0         | identical bin across builds |
| NTT (mod q) n=1024 | exact                 | 0         | identical                   |

*(These values are example targets; implementers should derive their own tolerance bounds.)*

### DD. Implementation and CI recommendations

- Include dedicated unit-tests for FFT/NTT round-trip, norm checks and RNG seed reproducibility.
- Maintain a “golden hash” of transform output arrays (post FFT) for regression-testing platforms.
- Register and monitor rejection-rate trends in signing as part of build health. Rejection-rate outliers may signal drifting precision or hardware differences.
- For embedded or FPU-less platforms, consider fixed-point or integer-only versions of the transforms and validate accordingly. Hardware differences (subnormals, denormals) should be mitigated.

### EE. Summary

The transforms (FFT/NTT) are critical arithmetic components of Falcon. Their correct and consistent implementation across platforms is a prerequisite for secure deployment. Thorough numerical validation in development, CI pipelines and across target hardware ensures that trapdoor sampling, signature generation and verification all operate reliably and uniformly.

## D Comparison tables (Falcon vs. Dilithium vs. ECC)

Table 3: Detailed parameter comparison: public key size, private key size, signature size (representative security levels).

| Scheme                    | Security level      | Public key size (bytes) | Private/secret key size (bytes) |
|---------------------------|---------------------|-------------------------|---------------------------------|
| FALCON-512                | NIST Level 1        | 897                     | 1 281                           |
| FALCON-1024               | NIST Level 5        | 1 793                   | 2 305                           |
| CRYSTALS-DILITHIUM2       | NIST Level 2        | 1 312                   | 2 528                           |
| CRYSTALS-DILITHIUM5       | NIST Level 5        | 2 592                   | 4 864                           |
| ECC (SECP256K1 / Ed25519) | Classical = 128 bit | 64                      | 32                              |

Notes:

- Sizes reflect typical / recommended parameter sets; actual implementation (padding, encoding) may vary slightly.
- “Comp.” = compressed format; some schemes (e.g., Falcon) support both compressed and padded signature variants.
- ECC row is included for legacy comparison of key/signature size — note that ECC is not post-quantum secure.
- Security levels referenced follow the National Institute of Standards and Technology (NIST) designations for post-quantum security categories.

## E Security proof sketch and parameter selection rationale

This appendix provides a high-level sketch of the security argument for Falcon, and explains the rationale behind its key parameter choices (e.g., ring dimension  $n$ , modulus  $q$ , signature rejection bound  $\beta$ ). We do **not** give a full formal proof, but capture the main reductions and heuristics so the implementer or auditor understands why those parameters were chosen.

### AA. Security proof sketch

The Falcon scheme is built using the Craig Gentry–Chris Peikert–Vinod Vaikuntanathan (GPV) lattice-signature framework instantiated over an NTRU lattice. The main elements of the proof are:

- The public key corresponds to a lattice  $\Lambda_h$  of covolume roughly  $q^n$ .
- Signature generation samples a short vector  $\mathbf{s} \in \Lambda_h$  such that  $\mathbf{s}$  maps to the challenge  $c$ . This satisfies correctness because the vector is in the lattice and short enough (norm  $\leq \beta$ ).
- Unforgeability reduces to the hardness of finding a short non-trivial lattice vector, or equivalently approximating the Short Integer Solution (SIS) / Shortest Vector Problem (SVP) in the NTRU lattice. The reduction proceeds in the (quantum) random-oracle model. The published bound for the two main parameter sets show adversary advantage  $\leq (q_H + 2)^2 \cdot 2^{-(5-k)n/2}$  reaching approximately  $2^{-1024}$  at the lower level and  $2^{-2560}$  at the higher.
- The salt  $r \in \{0,1\}^k$  prepended in HashToPoint prevents hash collisions and supports the security proof: Falcon uses  $k = 320$  bits to cover worst-case number of queries up to  $2^{64}$ .

Together, these arguments establish that an adversary forging a signature must sample a vector too short in the lattice, which is considered computationally infeasible given current best lattice-reduction methods.

### BB. Parameter selection rationale

Key parameters in Falcon are selected with these guiding principles:

- **Ring dimension  $n$ :** For the two level sets Falcon-512 ( $n = 512$ ) and Falcon-1024 ( $n = 1024$ ), the dimension is chosen such that the underlying lattice problems yield the intended security margin  $\approx 128 - \text{bit}$  and  $\approx 256 - \text{bit}$  respectively.
- **Modulus  $q$ :** A fixed prime modulus  $q = 12289$  is used for both levels, simplifying implementation and enabling efficient NTT arithmetic.

- **Signature bound  $\beta$** : The norm bound for valid signatures is chosen to ensure rejection probability is extremely low, yet any vector satisfying the bound remains hard to find. For example at  $\beta^2 \approx 3.4 \times 10^7$ .
- **Salt size  $k$** : The 320-bit salt supports up to  $2^{64}$  signature queries while maintaining negligible collision probability.
- **Gaussian standard deviation  $\sigma$** : The sampler’s Gaussian width is chosen to balance rejection-rate (efficiency) and statistical distance from ideal discrete Gaussian (security). The published value is approximately 165.7 for  $n = 512$ .

## CC. Summary and implementer remarks

In summary, the security of Falcon rests on the hardness of short-vector extraction in structured NTRU lattices, instantiated with conservative parameters. Implementers should note:

- The proof bound is asymptotic and uses the random-oracle model; real-world instantiations must ensure side-channels, precision flaws or bias in sampling do not degrade the margin.
- Reuse of salt or drift in the sampler’s distribution may invalidate assumptions underlying the bound.
- Migration to future proof levels (e.g., beyond 256-bit quantum security) may require reconsidering parameter growth (larger  $n$ , different modulus) or newer schemes.

With these details, the reader gains insight into *why* each parameter in Falcon was chosen, and what risks potentially remain beyond the ideal proof.

## References

- [AAB<sup>+</sup>24] Marius A. Aardal, Diego F. Aranha, Katharina Boudgoust, Sebastian Kolby, and Akira Takahashi. Aggregating falcon signatures with LaBRADOR. Cryptology ePrint Archive, Paper 2024/311, 2024.
- [Ajt96] Miklós Ajtai. Generating hard instances of lattice problems. In *Proceedings of the 28th Annual ACM Symposium on Theory of Computing (STOC)*, pages 99–108, 1996.
- [Alg24] Algorand. go-algorand release 4.3.0. <https://github.com/algorand/go-algorand/releases/tag/v4.3.0-stable>, 2024. Consensus upgrade v41 introducing the falcon\_verify opcode for on-chain signature verification.
- [BDK<sup>+</sup>21] Shi Bai, Léo Ducas, Eike Kiltz, Tancrede Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. Crystals-dilithium: Algorithm specifications and supporting documentation. <https://pq-crystals.org/dilithium/data/dilithium-specification-round3-20210208.pdf>, 2021. NIST PQC Round 3 submission.
- [CSR22] NIST CSRC. Pqc candidates to be standardized and round 4. <https://csrc.nist.gov/news/2022/pqc-candidates-to-be-standardized-and-round-4>, 2022. NIST announcement of Falcon selection for standardization.
- [DMS<sup>+</sup>25] Renaud Dubois, Simon Masson, Antonio Sanso, Marius van der Wijden, Kevaundray Wedderburn, Zhenfei Zhang, and Nicolas Consigny. Eip-8052: Precompile for fal-

- con support. <https://eips.ethereum.org/EIPS/eip-8052>, 2025. Draft proposal for Falcon-512 precompiled contracts in EVM.
- [EFG<sup>+</sup>21] Tristan Espitau, Pierre-Alain Fouque, Benjamin Gérard, Martin Rossi, and Mehdi Tibouchi. Mitaka: A simpler, parallelizable, maskable variant of falcon. In *Third PQC Standardization Conference*. NIST, 2021.
- [FKT<sup>+</sup>18] Pierre-Alain Fouque, Paul Kirchner, Mehdi Tibouchi, et al. Falcon: Fast-fourier lattice-based compact signatures over ntru (specification). <https://falcon-sign.info/falcon.pdf>, 2018. See Section 3.12 on Key-Recovery Mode.
- [Fou24] Algorand Foundation. Post-quantum cryptography on algorand. <https://algorand.co/technology/post-quantum>, 2024. Algorand’s post-quantum cryptography implementation.
- [GJK24] Phillip Gajland, Jonas Janneck, and Eike Kiltz. A closer look at falcon. Cryptology ePrint Archive, Paper 2024/1769, 2024.
- [GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing (STOC)*, pages 197–206, 2008.
- [HPS98] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. Ntru: A ring-based public key cryptosystem. In *Advances in Cryptology — ANTS III*, volume 1423 of *Lecture Notes in Computer Science*, pages 267–288. Springer, 1998.
- [KIB<sup>+</sup>25] Aigerim Kerimbayeva, Maksim Iavich, Yenlik Begimbayeva, Sergiy Gnatyuk, Sakhybay Tynymbayev, Zhanerke Temirbekova, and Olga Ussatova. A lightweight variant of falcon for efficient post-quantum digital signature. *Information*, 16(7), 2025.
- [NG23] Duc Tri Nguyen and Kris Gaj. Fast falcon signature generation and verification using ARMv8 NEON instructions. In *Progress in Cryptology – AFRICACRYPT 2023*, pages 417–441, 2023.
- [QA25] Jinyi Qiu and Aydin Aysu. Shift snare: Uncovering secret keys in falcon via single-trace analysis. Cryptology ePrint Archive, Paper 2025/146, 2025.
- [Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing (STOC)*, pages 84–93, 2005.
- [San25] Antonio Sanso. The road to post-quantum ethereum transaction is paved with account abstraction (aa). <https://ethresear.ch/t/the-road-to-post-quantum-ethereum-transaction-is-paved-with-account-abstraction-aa/21783>, 2025. Discussion of integrating Falcon signatures via ERC-4337 account abstraction on Ethereum.
- [SE91] Claus-Peter Schnorr and Martin Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. In *Fundamentals of Computation Theory*, volume 529 of *Lecture Notes in Computer Science*, pages 68–85. Springer, 1991.
- [Sho94] Peter W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, pages 124–134, 1994.

- [ZKN25] ZKNox. Ethfalcon: Falcon signature verification for ethereum. <https://github.com/ZKNoxHQ/ETHFALCON>, 2025. Experimental EVM implementation with gas benchmarks: 7M gas (NIST-compliant) and 1.8M gas (EVM-friendly variant).