

Every Signing Leaks: Breaking Falcon via Floating-Point Conversion Leakage

Yuanyuan Zhou¹, Weijia Wang², Yiteng Sun², and Yu Yu³

¹ Synopsys

zhou.yuanyuan@gmail.com

² Shandong University

wjwang@sdu.edu.cn, sunyiteng@mail.sdu.edu.cn

³ Shanghai Jiao Tong University

yuyu@yuyu.hk

Abstract. Falcon offers compact signatures and well-studied mathematical security, but its side-channel security remains a critical challenge. In particular, its floating-point Gaussian sampler constitutes a major source of side-channel leakage, while protecting it efficiently is difficult because of its reliance on floating-point arithmetic. However, existing attacks targeting the sampler under realistic noisy-leakage conditions typically require thousands of traces and are evaluated on unoptimized implementations. In this paper, we present a new attack on Falcon’s sampler, which reduces the number of required traces by more than two orders of magnitude compared with prior attacks under realistic noisy-leakage conditions and works on both unoptimized (-00) and highly optimized (-03) builds. Our attack targets the floating-point Gaussian centers μ used by the sampler at the recursion leaves of the `ffSampling` procedure. Specifically, it exploits signed-exponent leakage arising when $\lfloor \mu \rfloor$ is converted from an integer back to floating-point representation, as well as the weaker sign-only leakage. Signed-exponent leakage places the integer part $\lfloor \mu \rfloor$ of each center in one of 17 signed power-of-two intervals, whereas sign-only leakage divides it into two intervals according to its sign. We propose CB-MLE, which uses gradient descent to obtain an estimate f^* whose computed Gaussian-center classifications are consistent with the observed classifications under the profiled confusion matrix. An ISD-style search tests subsets of the more reliable coefficients in f^* with LLL/BKZ on the corresponding reduced NTRU lattices to recover the full key. We collect power traces from the Falcon implementation in the PQClean library running on an ARM Cortex-M4 microprocessor. Our Gaussian-center classifier achieves single-trace classification accuracies of 0.9997 and 0.884 at the -00 and -03 optimization levels, respectively, confirming exploitable signed-exponent leakage from the Gaussian centers. Using these classification results, the full attack recovers Falcon-512 keys with success rates of 100% from 20 signatures at -00 and 100% from 56 signatures at -03. For Falcon-1024, it reaches 100% from 21 signatures at -00 and 100% from 100 signatures at -03.

Keywords: Falcon · Post-quantum cryptography · Side-channel attack · NTRU · Lattice-based signatures · Lattice reduction

1 Introduction

Falcon [21] is an NTRU-lattice instantiation of the GPV hash-and-sign framework [10], and is under standardization by NIST as FN-DSA in FIPS 206. Among the post-quantum signature schemes selected by NIST for standardization, Falcon offers the most compact signatures. However, implementations of Falcon remain vulnerable to side-channel attacks, in which attackers exploit physical leakage, such as timing, power consumption, or electromagnetic or acoustic emissions, to recover sensitive information.

It is widely known that applying conventional side-channel countermeasures, particularly masking, to Falcon remains quite challenging [19]. In particular, Falcon’s Gaussian sampler relies on numerous floating-point operations and intermediate values that are difficult to mask efficiently. This difficulty is inherent to the sampler and persists regardless of the compiler optimization level. To date, no efficient masking scheme is known for the complete sampler [9]. Avoiding floating-point arithmetic and facilitating side-channel protection were also key motivations for the design of Mitaka [8]. In this respect, Gaussian sampler constitutes a major source of Falcon’s side-channel leakage.

Several attacks have used leakage from Falcon’s sampler. Guerreau et al. [11] exploit leakage from Falcon’s **BaseSampler** to determine whether its output is zero. They use this information to filter the signatures and then apply a hidden-parallelepiped method to approximate the corresponding vector in the secret Gram–Schmidt basis. Zhang et al. [23] work with the same leakage through second-order covariance and also examine the random sign flip in the integer sampler. To handle noisy **BaseSampler** observations, Noh et al. [18] combine maximum a posteriori (MAP) decoding with ordinary least squares (OLS), with optional DDGR approximate-hint recovery [6]. Lin et al. [17] instead use NTRU symplecticity: one signature supplies several equivalent observations from its half-Gaussian and sign-only leakage, after which errors in the approximate key are decoded. Brinkmann et al. [2] return to the sign-only leakage from z and the Learning Halfspace Problem of Zhang et al. [23], but use logistic regression. Each coefficient gives one halfspace sample, so one signature provides 512 samples for Falcon-512 and 1024 for Falcon-1024. Their method uses the sign of z , while our attack setting with sign-only leakage exploits the sign of the Gaussian center μ . Section 5.2 compares the two methods under the same simulated sign-error rates.

The performance of these attacks, measured by the number of required signatures, naturally depends on the leakage quality, particularly its noise level. Brinkmann et al. recover Falcon-512 keys from 100 (on average) signatures when all simulated sign labels are correct [2]. When the label accuracy is reduced to 80%, their experiment uses 6,000 signatures and succeeds in 5% of trials. Results that account for measurement effects or classification errors still require thousands of observations. For measured Falcon sampler traces, Lin et al. report 25% key recovery from 6,500 traces [17]. Such measurement studies have also concentrated on unoptimized (–00) code. This leaves two practical questions:

how many signatures are needed with an imperfect classifier, and whether the leakage can still be used after compiler optimization.

Contributions. This work presents a new side-channel attack on Falcon’s sampler that substantially reduces the number of signatures required for key recovery compared with prior sampler-side attacks. On an ARM Cortex-M4 microprocessor, our Falcon-512 attack reaches a 100% success rate with 20 signatures at `-O0` and 56 at `-O3`. Compared with the previous result that requires 6,500 measured traces, this reduces the number of signatures by factors of approximately 325 and 116, respectively. We provide in Table 1 a comparison with prior side-channel attacks on Falcon’s sampler. Our contributions are summarized as follows.

First, we focus on the floating-point Gaussian centre μ at each leaf of the recursive `ffSampling` procedure. The sampler rounds this value down to the integer $s = \lfloor \mu \rfloor$. We first exploit the *signed-exponent leakage* produced by the normalization in `FPR_NORM64` when s is converted back to floating point. This signed-exponent leakage divides the possible values of s into 17 signed power-of-two intervals. We also consider the weaker case of *sign-only leakage*, which divides s into two intervals, one for negative values and one for non-negative values. For each type of leakage, we profile the classification errors to obtain a confusion matrix P , such that $P_{i,j}$ is the probability that a value in the i -th interval is classified into the j -th interval. This confusion matrix is critical to the following key-recovery procedure.

Second, we propose Confusion Matrix-based Maximum Likelihood Estimation (CB-MLE) to obtain an approximate estimate f^* of Falcon’s secret polynomial f . CB-MLE uses the relation between f and the Gaussian centers to refine a candidate f' by gradient descent. In each iteration, the current f' and the public challenges are used to compute the Gaussian centers and their classifications. Gradient descent iteratively updates f' so that the differences between the computed and observed Gaussian-center classifications fit the error distribution described by the profiled confusion matrix P . Finally, CB-MLE outputs an estimate f^* that may still contain some incorrect coefficients, and we propose an ISD-style (Information Set Decoding [20]) search over subsets of the more reliable coefficients, testing each subset with LLL/BKZ on the corresponding reduced NTRU lattice.

Third, we evaluate our attack using power traces collected from the PQClean implementation of Falcon running on an ARM Cortex-M4, compiled at `-O0` and `-O3`. The signed-exponent classifier achieves accuracies of 0.9997 and 0.884, respectively. With these classification results, CB-MLE recovers enough correct coefficients of f to meet the lattice-recovery criterion using at most 20 signatures at `-O0` and 56 at `-O3` for Falcon-512, and 21 and 100 signatures for Falcon-1024. To test the robustness of CB-MLE, we also evaluate sign-only leakage in simulation at different classification error rates. Even at an error rate of 30%, all 100 runs reach a key recovery success rate of 100% with 2,160 signatures.

Related works. It should be noted that Li et al. [16] target operations in Falcon’s key-generation procedure during signing and exploit very similar signed-

Table 1. Attacks on Falcon-512’s sampler.

Reference	#Signatures	Success Rate	Target Platform (Optimization)
Guerreau et al. [11]	10^6 *	n/a	Simulated with ELMO (-00)
Zhang et al. [23] [§]	45,000	25%	ARM Cortex-M4 (-00)
Lin et al. [17]	6,500	25%	ARM Cortex-M4 (-00)
Noh et al. [18]	25,000	~100%	ARM Cortex-M4 (-00)
Brinkmann et al. [2]	100	100%	Simulated
This work	20	100%	ARM Cortex-M4 (-00)
This work	56	100%	ARM Cortex-M4 (-03)

* This 10^6 -signature setting assumes roughly 1000 hours of lattice reduction.

§ Zhang et al. used filtering, and their 45 k post-filtered signatures correspond to 161 k raw signatures.

exponent leakage, but their partition uses only 11 intervals. By contrast, the Gaussian centers targeted by our attack span a much wider numerical range, and our signed-exponent leakage divides them into 17 classes. The subsequent key recovery strategies also differ. Li et al. directly recover a subset of the secret-key coefficients of both f and g and then apply an ISD-style search to recover the remaining coefficients. Our key-recovery method explicitly accounts for interval-classification errors before passing the recovered coefficients to the ISD-style search and lattice reduction.

Organization. Section 2 gives some necessary backgrounds. Section 3 describes the signed-exponent and sign-only leakage at the leaf nodes of the recursive **ffSampling** procedure. Section 4 introduces our proposed CB-MLE key-recovery method. Section 5 evaluates our attack both in practice and through simulation. Section 6 discusses some possible countermeasures, and Section 7 concludes the paper.

2 Preliminaries

2.1 Notation

Let $[n] = \{0, 1, \dots, n-1\}$ and $\mathbb{Z}_q = \mathbb{Z}/q\mathbb{Z}$. The main operations in Falcon are performed over the polynomial ring $R_q = \mathbb{Z}_q[x]/(x^n + 1)$, with $n \in \{512, 1024\}$ and $q = 12289$. For an element $f \in R_q$, we denote its i -th coefficient by f_i , i.e., $f = \sum_{i=0}^{n-1} f_i x^i$. For a vector $\mathbf{v}$, we write its infinity norm as $\|\mathbf{v}\|_\infty$. We use $\Pr[\cdot]$, $\mathbb{E}[\cdot]$, and $\text{Var}[\cdot]$ for probability, expectation, and variance. Finally, $\Phi(\cdot)$ is the standard normal cumulative distribution function (CDF), and $\text{GH}(L)$ is the Gaussian heuristic of a lattice L .

2.2 The Falcon Signature Scheme

Falcon follows the GPV hash-and-sign framework over NTRU lattices.

Key generation. Falcon’s secret key is a tuple of short polynomials $(f, g, F, G) \in R_q^4$ satisfying the NTRU equation $fG - gF = q$. Falcon samples the coefficients of f and g from a discrete Gaussian with standard deviation $\sigma_f = 1.17\sqrt{q/(2n)}$, which is about 4.05 for $n = 512$ and 2.87 for $n = 1024$. Falcon further requires each coefficient of f to satisfy $|f_i| \leq 2^{\max\_fg\_bits[\log_2 n] - 1} - 1$. Hence $f_i \in [-31, 31]$ for Falcon-512 and $f_i \in [-15, 15]$ for Falcon-1024. At the same time, Falcon’s public key is $h = g \cdot f^{-1} \bmod q \in R_q$.

Signing. To sign a message m , the signer:

1. Samples a salt r and computes $c = \text{HashToPoint}(r \| m) \in R_q$.
2. Computes the target vector in FFT domain:

$$t = (t_0, t_1) = \left(-\frac{1}{q} \text{FFT}(c) \odot \text{FFT}(F), \frac{1}{q} \text{FFT}(c) \odot \text{FFT}(f) \right). \quad (1)$$

3. Performs fast Fourier sampling: $z \leftarrow \text{ffSampling}(t, T)$, where T is the LDL* tree of the Gram matrix.
4. Computes $s = (t - z) \cdot \hat{B}$ and accepts if $\|s\|^2 \leq \lfloor \beta^2 \rfloor$.

Gaussian sampler. At the leaves of the FFT tree, the sampler generates discrete Gaussian samples via a base sampler and rejection sampling:

$$z_0 \leftarrow \text{BaseSampler}(), \quad b \leftarrow \{0, 1\}, \quad z \leftarrow b + (2b - 1) \cdot z_0. \quad (2)$$

Let $r = \mu - \lfloor \mu \rfloor$ and $\text{ccs} = \sigma_{\min} / \sigma'$. For the candidate z in Equation (2), **SamplerZ** computes

$$x = \frac{(z - r)^2}{2\sigma'^2} - \frac{z_0^2}{2\sigma_{\max}^2}.$$

BerExp accepts this candidate with approximate probability $\text{ccs} \cdot \exp(-x)$. If the candidate is rejected, **SamplerZ** draws a new z_0 and sign bit; otherwise, it returns $z + \lfloor \mu \rfloor$. This rejection step corrects the half-Gaussian proposal of **BaseSampler** to obtain the required discrete Gaussian distribution centered at μ with standard deviation σ' .

2.3 Floating-Point Representation

Falcon contains many floating-point operations and represents its floating-point values in binary64, which is the 64-bit IEEE-754 double-precision format. In this format, a floating-point value is represented by a 1-bit sign s , an 11-bit exponent e , and a 52-bit mantissa m , and its value equals $(-1)^s \cdot 2^{e-1023} \cdot (1 + m \cdot 2^{-52})$. It should be noted that, in this paper, we consider side-channel leakage from the sign and exponent of a floating-point value, rather than the more detailed information contained in the mantissa.

2.4 Negacyclic Matrix and Public Key Relation

For an index set $\mathcal{S} \subseteq [n]$, we write $f_{\mathcal{S}} = (f_i)_{i \in \mathcal{S}}$ for the coefficients of f whose indices are in $\mathcal{S}$. The public-key relation $g = h \cdot f \bmod (x^n + 1, q)$ can then be written as the matrix equation

$$g = H \cdot f \pmod{q} \quad (3)$$

where $H \in \mathbb{Z}_q^{n \times n}$ is the negacyclic matrix of h , with $H_{i,j} = (-1)^{\lfloor (i-j)/n \rfloor} \cdot h_{(i-j) \bmod n}$.

2.5 Lattice Attacks with Hints

The framework of Dachman-Soled et al. [6] formalizes side-channel leakage as “hints” about the secret vector; a *perfect hint* (v, l) asserts $\langle v, s \rangle = l$, and Cao, Jiang, and Cheng [3] integrate perfect hints efficiently by constructing the reduced lattice through Gaussian elimination. Each coefficient we determine is an *axis-aligned* perfect hint $\langle e_i, f \rangle = f_i$: it fixes one coordinate of f and thereby lowers the recovery dimension, but, unlike a generic dense hint, it contributes no lattice-determinant gain. Therefore, the hint counts for generic hints do not transfer to coordinate hints. Coordinate hints never decide whether f is determined but only lower the cost of finding it, and that cost decreases smoothly with the number of fixed coefficients rather than at a fixed threshold. Gaussian elimination without any lattice reduction (i.e. purely algebraic recovery) succeeds once the known coordinates form an information set of the rank- n relation $g = Hf$, which requires $d_f + d_g \geq n$ known coefficients of f and g combined. Li et al. [16] meet this bound with $n/2$ coefficients of each polynomial. Our leakage reveals f only, so the corresponding algebraic bound would be all n coefficients of f . During our experiments, we use a computational criterion instead of a combinatorial one, that is, we run the lattice step when the residual lattice of Section 4.2 can be reduced within the compute budget specified in Section 5. We retain $n/2$ only as a cheap screening condition on the candidate set because candidate sets below this size are never close to the compute budget, so it is a necessary condition for the instances we attempt but not a sufficient condition. Wrong hints are handled by retrying with another information set as detailed in Section 4.2. Our estimator provides far above $n/2$ correct coefficients at every reported operating point (Section 5).

2.6 Adaptive Moment Estimation

Adaptive moment estimation (Adam) was proposed by Kingma et al. [15], and it is a gradient-based method for stochastic optimization. Adam uses the gradients from previous iterations to determine the direction and size of each update. Specifically, let $L(\mathbf{z})$ be the function to be minimized, with $\mathbf{z} \in \mathbb{R}^d$ the vector of parameters. At the t -th iteration, $\mathbf{z}_{t-1}$ is the current parameter vector and

$\mathbf{g}_t = \nabla L(\mathbf{z}_{t-1})$ is the gradient evaluated at $\mathbf{z}_{t-1}$. Adam keeps exponential moving averages of the gradient and its coordinate-wise square:

$$\mathbf{m}_t = \beta_1 \mathbf{m}_{t-1} + (1 - \beta_1) \mathbf{g}_t, \quad \mathbf{v}_t = \beta_2 \mathbf{v}_{t-1} + (1 - \beta_2)(\mathbf{g}_t \odot \mathbf{g}_t),$$

where $\odot$ denotes coordinate-wise multiplication, and $\mathbf{m}_t$ and $\mathbf{v}_t$ are vectors with the same dimension as $\mathbf{z}$. The vector $\mathbf{m}_t$ is the exponential moving average of the gradient and records its average direction. The vector $\mathbf{v}_t$ is the exponential moving average of the coordinate-wise square of the gradient and records the gradient scale of each coordinate. We can see that β_1 and β_2 control how quickly the two averages change. The bias-corrected averages and the update are

$$\hat{\mathbf{m}}_t = \frac{\mathbf{m}_t}{1 - \beta_1^t}, \quad \hat{\mathbf{v}}_t = \frac{\mathbf{v}_t}{1 - \beta_2^t}, \quad \mathbf{z}_t = \mathbf{z}_{t-1} - \alpha \frac{\hat{\mathbf{m}}_t}{\sqrt{\hat{\mathbf{v}}_t} + \varepsilon}.$$

Here, α is the learning rate and ε prevents division by zero. The division and square root are computed coordinate by coordinate. Since $\hat{\mathbf{v}}_t$ is computed separately for each coordinate, Adam makes smaller changes to parameters whose gradients are usually larger. This prevents these parameters from dominating the optimization. Section 4.1 applies Adam to estimate the coefficients of secret polynomial f' .

3 Attack Target and Leakage of Gaussian Center

3.1 Attack Target

Threat model. We consider a profiled attack that can be divided into a profiling phase and an attack phase. During profiling, the adversary trains a leakage model on a clone device and may choose the secret keys used on that device. During the attack phase, the target device repeatedly signs with a fixed secret key (f, g, F, G) . Each signing operation gives the adversary one power trace and the corresponding public signature. The adversary also knows the public key h , but does not need to choose the messages or salts.

Falcon uses `ffSampling` to sample a short vector during signing. Algorithm 1 gives the recursive procedure from the Falcon specification [21]. For $n > 1$, it splits the input t_1 , samples the two resulting subproblems, and then updates and splits t_0 . Each recursive call reduces the dimension from n to $n/2$. The recursion stops at $n = 1$, where `SamplerZ` is called on the scalar components.

The t_1 component of the input $\mathbf{t}$ is computed from the public challenge c and the secret polynomial f as

$$t_1 = \frac{1}{q} \text{FFT}(c) \odot \text{FFT}(f).$$

The recursive `splitfft` calls carry t_1 down to the leaves of the recursion, where each scalar value is passed to `SamplerZ` as a Gaussian center. Specifically, the Gaussian center at position i is

$$\mu_i = \frac{(c \circledast f)_i}{q}, \tag{4}$$

Algorithm 1 Recursive `ffSampling`, adapted from Algorithm 11 of the Falcon specification [21].

Require: Target $\mathbf{t} = (t_0, t_1)$ and Falcon tree T

Ensure: Integer vector $\mathbf{z} = (z_0, z_1)$

```

1: if  $n = 1$  then
2:    $\sigma' \leftarrow T.\text{value}$ 
3:    $z_0 \leftarrow \text{SamplerZ}(t_0, \sigma')$ 
4:    $z_1 \leftarrow \text{SamplerZ}(t_1, \sigma')$ 
5:   return  $\mathbf{z} = (z_0, z_1)$ 
6:  $(\lambda, T_0, T_1) \leftarrow (T.\text{value}, T.\text{left}, T.\text{right})$ 
7:  $\mathbf{t}_1 \leftarrow \text{splitfft}(t_1)$ 
8:  $\mathbf{z}_1 \leftarrow \text{ffSampling}_{n/2}(\mathbf{t}_1, T_1)$ 
9:  $z_1 \leftarrow \text{mergefft}(\mathbf{z}_1)$ 
10:  $t'_0 \leftarrow t_0 + (t_1 - z_1) \odot \lambda$ 
11:  $\mathbf{t}_0 \leftarrow \text{splitfft}(t'_0)$ 
12:  $\mathbf{z}_0 \leftarrow \text{ffSampling}_{n/2}(\mathbf{t}_0, T_0)$ 
13:  $z_0 \leftarrow \text{mergefft}(\mathbf{z}_0)$ 
14: return  $\mathbf{z} = (z_0, z_1)$ 

```

where $\odot$ denotes negacyclic convolution. The challenge c is public and can be recomputed from the message and salt, while f is secret. Thus, each center is a linear function of the coefficients of f with publicly known coefficients.

At the recursive leaf, `SamplerZ` takes a Gaussian center μ and a standard deviation σ' . It begins by separating the integer and fractional parts of μ [21]. We write s for the integer part and present the step relevant to our attack in Algorithm 2.

Algorithm 2 `SamplerZ`, adapted from specification [21].

Require: Gaussian center μ and standard deviation σ'

```

1:  $s \leftarrow \lfloor \mu \rfloor$   $\triangleright$  integer part
2:  $r \leftarrow \mu - s$   $\triangleright$  fractional part, implemented by  $\mu - \text{fpr\_of}(s)$ 
    $\vdots$ 

```

3.2 Classification of the Gaussian Center

In the Falcon reference and PQClean implementations, `SamplerZ` begins by computing $r = \mu - \text{fpr\_of}(s)$, where `fpr_of` converts the integer s back to a 64-bit floating-point value before the subtraction. This conversion processes both the sign and exponent of s , and this operation is the main leakage source considered in this paper. We believe this leakage source is quite generic because an equivalent conversion should always be required in Falcon’s sampler when hardware floating-point support is unavailable.

For instance, in PQClean, `fpr_of(s)` calls `fpr_scaled(s, 0)`. This routine separates the sign, sets $m = |s|$, and saves the sign for the final 64-bit floating-

point encoding, naturally leaking the sign of s . It then passes m to `FPR_NORM64`. For a nonzero m , `FPR_NORM64` moves the highest set bit to bit 63 using a fixed sequence of tests for shifts by 32, 16, 8, 4, 2, and 1. In the first test, `nt = (uint32_t)(m >> 32)` is reduced to a one-bit value that indicates whether the upper half of m is nonzero. If the bit is zero, a mask selects `m << 32`; otherwise, m is left unchanged. The same bit updates the exponent: after an initial subtraction of 63, `e += nt << 5` restores 32 when no shift is applied. The remaining five tests follow the same pattern. If the total left shift is L , the output exponent equals the input exponent minus L . Since L is the number of leading zeros, the result reveals $\lfloor \log_2 |s| \rfloor$. For the shift order (32, 16, 8, 4, 2, 1), the `nt` sequence is all zero at $|s| = 1$, becomes (0, 0, 0, 0, 0, 1) for $|s| \in \{2, 3\}$, and is (0, 0, 0, 0, 1, 0) for $|s| \in \{4, 5, 6, 7\}$. Integers with the same binary exponent therefore share a normalization pattern and fall in the same interval $[2^k, 2^{k+1})$. Together, the interval and the saved sign form the *signed-exponent leakage*. The following function maps a value to its corresponding leakage class.

Definition 1 (signed-exponent function). For an integer cutoff $K \geq 1$, define $\exp_K : \mathbb{R} \rightarrow \{0, \pm 1, \dots, \pm K\}$ as

$$\exp_K(x) = \begin{cases} 0, & |x| < 1, \\ \text{sgn}(x) \cdot \min\{\lfloor \log_2 |x| \rfloor + 1, K\}, & |x| \geq 1. \end{cases} \quad (5)$$

The map $\exp_K$ keeps the sign and places the magnitude between consecutive powers of two. The two outer classes contain all values with $|x| \geq 2^{K-1}$. We set $K = 8$ for the rest of the paper and abbreviate $\exp_8$ as $\exp$, giving the alphabet $\{0, \pm 1, \dots, \pm 8\}$. In this case, only quite rare values with $|x| \geq 128$ fall in the tails.

Although the signed-exponent leakage can be identified with high accuracy in practice, as shown in Section 5.1, we also consider the weaker sign-only leakage, which reveals only whether μ or s is negative. This leakage contains only two classes and is easier to identify. We define the two classes using the following function.

Definition 2 (sign function). Define $\text{sign} : \mathbb{R} \rightarrow \{-1, +1\}$ as

$$\text{sign}(x) = \begin{cases} -1, & x < 0, \\ +1, & x \geq 0. \end{cases} \quad (6)$$

We use this sign-only leakage not only because it is easier to identify, but also to evaluate the robustness of key recovery under different noise levels. This is because, with only two classes, the classification error is much easier to control in simulation.

3.3 Confusion Matrix of the Classification

The previous subsection describes two ways to classify a Gaussian center: signed-exponent leakage gives 17 classes, while sign-only leakage gives two. We use

classification accuracy as a first measure of leakage quality. For a Gaussian center μ , let b be its true class and $\hat{b}$ the class identified from the leakage. The accuracy is defined as

$$p \stackrel{\text{def}}{=} \Pr [\hat{b} = b].$$

This definition applies to both signed-exponent and sign-only leakage.

The accuracy defined above gives only the proportion of correctly identified classes. It does not show how an observation is distributed among the incorrect classes. We therefore use a confusion matrix to record these error probabilities.

Definition 3 (row-normalized confusion matrix). *For a Gaussian center μ , let b be its true class, $\hat{b}$ the class identified from the leakage, and $\mathcal{C}$ the set of all classes. For labeled profiling traces, let $N_{i,j}$ be the number of traces with $b = i$ and $\hat{b} = j$. The row-normalized confusion matrix $P \in [0, 1]^{|\mathcal{C}| \times |\mathcal{C}|}$ has entries*

$$P_{i,j} \stackrel{\text{def}}{=} \frac{N_{i,j}}{\sum_{k \in \mathcal{C}} N_{i,k}} \approx \Pr[\hat{b} = j \mid b = i], \quad i, j \in \mathcal{C}.$$

Rows correspond to true classes and columns to observed classes, and each row sums to one.

This definition applies to both signed-exponent and sign-only leakage. The two leakage types determine their respective class sets $\mathcal{C}$, whose sizes are 17 and 2, respectively. For the sake of brevity, we refer to the row-normalized confusion matrix simply as the confusion matrix in the rest of the paper. The matrix is estimated during profiling and used by the key-recovery method presented in Section 4. It is therefore a critical component of the attack, rather than only a measure of leakage quality.

4 Key Recovery

Given the public key h , the public challenges $c^{(j)}$, the observed Gaussian-center classifications, and the profiled confusion matrix P , we recover the key in two steps. We first apply CB-MLE to obtain an estimate f^* of the secret polynomial f . Some coefficients may still be incorrect after f^* is rounded. We then use an ISD-style search to find a sufficiently large correct subset of the rounded coefficients and recover the complete key by lattice reduction. Section 4.1 describes the estimation of f , and Section 4.2 presents the ISD-style search and lattice-recovery step. Algorithm 3 summarizes the complete procedure.

Algorithm 3 Overall key-recovery procedure.

Require: Public key h , challenges $\{c^{(j)}\}_{j \in [N]}$, observed Gaussian centers' classes $\{\hat{b}_i^{(j)}\}_{j \in [N], i \in [n]}$, and profiled confusion matrix P

Ensure: Secret polynomial f or FAIL

- 1: Estimate f^* from $\{c^{(j)}\}$, $\{\hat{b}_i^{(j)}\}$, and P using Algorithm 4
 - 2: Recover f from f^* and h using the ISD-style lattice search in Algorithm 5
 - 3: **return** f or FAIL
-

4.1 Recovery of f

4.1.1 Overview of our method. We estimate f by adjusting a candidate f' until the classes computed from f' explain the observed classes under the profiled confusion matrix P . For each f' , we define $\mathcal{T}'$ as a set of pairs. Each pair consists of a Gaussian-center class reconstructed from f' and the class observed from the corresponding trace. The fitting target is

$$f^\star \stackrel{\text{def}}{=} \arg \min_{f'} D(\mathcal{T}', P),$$

where D is the negative log-likelihood score given by P .

4.1.2 The class pairs $\mathcal{T}'$. Let $j \in [N]$ be index of the signatures and $i \in [n]$ be index of the recursion leaves within each signature. For a candidate f' and a public challenge $c^{(j)}$, we define

$$s'_i{}^{(j)} \stackrel{\text{def}}{=} \left\lfloor \mu'_i{}^{(j)} \right\rfloor = \left\lfloor \frac{(c^{(j)} \circledast f')_i}{q} \right\rfloor,$$

where $\circledast$ denotes negacyclic convolution. The reconstructed classification of the leaf is

$$b'_i{}^{(j)} \stackrel{\text{def}}{=} \exp(s'_i{}^{(j)}) \in \mathcal{C}.$$

For each candidate f' , we collect the classification pairs in the set

$$\mathcal{T}' \stackrel{\text{def}}{=} \left\{ \left(b'_i{}^{(j)}, \hat{b}_i{}^{(j)} \right) \mid j \in [N], i \in [n] \right\}.$$

$\mathcal{T}'$ collects the pair associated with each of the Nn Gaussian centers. The first entry is computed from f' , while the second is exploited from the side-channel leakage.

4.1.3 The minimization procedure. The minimization evaluates a candidate f' by scoring all pairs in $\mathcal{T}'$:

$$D(\mathcal{T}', P) \stackrel{\text{def}}{=} - \sum_{j=1}^N \sum_{i=1}^n \log P_{b'_i{}^{(j)}, \hat{b}_i{}^{(j)}}, \quad (7)$$

where $P_{b,o} = \Pr[\hat{b} = o \mid b]$ is an entry of the profiled confusion matrix P .

A smaller score means that the classes computed from f' better explain the observed classes under the profiled matrix P . This use of P makes the minimization robust to classification errors. An incorrect classification does not immediately rule out a candidate, but instead contributes a penalty determined by the corresponding entry of P .

Equation (7), however, defines a hard score: each Gaussian center computed from f' is assigned to exactly one class b' . This hard assignment makes the score piecewise constant in f' , so it cannot be optimized directly by gradient descent.

To address this problem, we define the following smoothed observation probability from the confusion matrix P :

$$P_o^{\text{soft}}(u; \gamma) \stackrel{\text{def}}{=} \sum_{b \in \mathcal{C}} w_b(u; \gamma) P_{b,o}. \quad (8)$$

The weight $w_b(u; \gamma)$ smooths the hard class assignment. As a result, the score constructed from P_o^{soft} is differentiable in f' and can be optimized by gradient descent. Let $[L_b, U_b)$ be the continuous interval corresponding to class b . We define

$$w_b(u; \gamma) \stackrel{\text{def}}{=} \Phi\left(\frac{U_b - u}{\gamma}\right) - \Phi\left(\frac{L_b - u}{\gamma}\right).$$

where Φ is the standard normal cumulative distribution function, and γ controls the smoothing. A large value gives gradients over a wider range, while a small value is closer to the hard class assignment. The smoothed score is

$$D_\gamma(f') \stackrel{\text{def}}{=} - \sum_{j=1}^N \sum_{i=1}^n \log P_{b_i^{(j)}}^{\text{soft}}(\mu_i^{(j)}; \gamma) + \frac{\|f'\|^2}{2\sigma_f^2}. \quad (9)$$

The first term is the negative log-likelihood of the observed classifications under P^{soft} . The second term is an ℓ_2 regularizer. It penalizes candidates of f' with unusually large coefficients and keeps it within the expected range.

We minimize the resulting smoothed score with Adam, which requires an initial candidate of f' . We use the profiled confusion matrix P to estimate the corresponding Gaussian centers $\tilde{\mu}$ from the observed classifications $\hat{\mathbf{b}}$. The initial candidate f' can then be computed from $\tilde{\mu}$ and the public challenges using weighted least squares. Besides, during the optimization, γ is reduced geometrically from 0.30 to 0.12. The smoothing is scaled by the local interval width because the signed-exponent intervals have different sizes.

The optimization returns a real-valued estimate, which we denote by f^* . Algorithm 4 summarizes the recovery of f .

Algorithm 4 Recovery of f with CB-MLE.

Require: Challenges $\{c^{(j)}\}_{j \in [N]}$, observed Gaussian centers' classes $\{\hat{b}_i^{(j)}\}_{j \in [N], i \in [n]}$, profiled confusion matrix P , modulus q , smoothing schedule $\{\gamma_t\}_{t \in [T]}$, and the number of iterations T

Ensure: Real-valued estimate f^*

- 1: Initialize f' from P , $\{\hat{b}_i^{(j)}\}$, and $\{c^{(j)}\}$ using weighted least squares
 - 2: Initialize $D^* \leftarrow +\infty$
 - 3: **for** $t = 1$ to T **do**
 - 4: Compute $\mu_i^{(j)} = (c^{(j)} \otimes f')_i / q$ for all i, j
 - 5: Compute the smoothed score $D_{\gamma_t}(f')$ according to (9)
 - 6: Update f' with Adam using $\nabla_{f'} D_{\gamma_t}(f')$
 - 7: Form $\mathcal{T}'$ from the updated f' and compute $D' \leftarrow D(\mathcal{T}', P)$
 - 8: **if** $D' < D^*$ **then**
 - 9: $(D^*, f^*) \leftarrow (D', f')$
 - 10: **return** f^*
-

4.2 Key Recovery with an ISD-Style Search and Lattice Reduction

If the estimate f^* obtained in Section 4.1 is completely correct, the corresponding secret key can be recovered directly. However, obtaining a fully correct estimate inevitably requires many more traces than are actually needed. With fewer traces, f^* remains a real-valued estimate, and only some of its coefficients can be rounded to the correct integer values, i.e., values equal to the corresponding coefficients of the correct f . The rounded f^* alone does not reveal which of its coefficients are correct. We therefore propose an ISD-style search that tests candidate subsets of the rounded coefficients of f^* until a subset containing only correct coefficients is found.

Algorithm 5 summarizes the search and lattice-recovery procedure.

Algorithm 5 ISD-style search and lattice recovery.

Require: Estimate f^* , public key h , thresholds τ_1 and τ_2 , and maximum search depth $k_{\max}$

Ensure: Secret polynomial f or FAIL

- 1: Construct the negacyclic matrix H from h
- 2: Set $\hat{f}_i \leftarrow \text{round}(f_i^*)$ and $r_i \leftarrow |f_i^* - \hat{f}_i|$ for all $i \in [n]$
- 3: $\mathcal{D}_{\tau_1, \tau_2} \leftarrow \{i \in [n] : |\hat{f}_i| \leq \tau_1 \wedge r_i < \tau_2\}$
- 4: **if** $|\mathcal{D}_{\tau_1, \tau_2}| < n/2$ **then**
- 5: **return** FAIL
- 6: Order $\mathcal{D}_{\tau_1, \tau_2}$ as $i_1, i_2, \dots$ by decreasing r_i
- 7: Set $\mathcal{S} \leftarrow \emptyset$
- 8: **for** $k = 0$ to $\min(k_{\max}, |\mathcal{D}_{\tau_1, \tau_2}|)$ **do**
- 9: **for all** previously untested $\mathcal{S} \subseteq \{i_1, \dots, i_k\}$ **do**
- 10: $\mathcal{I} \leftarrow \mathcal{D}_{\tau_1, \tau_2} \setminus \mathcal{S}$
- 11: **if** $|\mathcal{I}| \geq n/2$ **then**
- 12: Choose $n - |\mathcal{I}| < m \leq n$ and construct the lattice in (11)
- 13: Run LLL/BKZ to obtain a candidate f
- 14: Compute $g \leftarrow Hf \bmod q$ and represent each coefficient in $[-\lfloor q/2 \rfloor, \lfloor q/2 \rfloor]$
- 15: **if** every centered coefficient of g satisfies $|g_i| \leq 5\sigma_f$ **then**
- 16: **return** f
- 17: **return** FAIL

Although f^* generally has non-integer coefficients, the correct polynomial f has integer coefficients. We therefore round each coefficient of f^* as

$$\hat{f}_i \stackrel{\text{def}}{=} \text{round}(f_i^*), \quad r_i \stackrel{\text{def}}{=} |f_i^* - \hat{f}_i|.$$

Before the ISD-style search, we apply two thresholds to form an initial set of candidate hints. The magnitude threshold $\tau_1 = 7$ keeps the estimates in the range where most Falcon coefficients lie. Under the coefficient distributions described in Section 2.2, this range contains approximately 93.7% of the coefficients for Falcon-512 and 99.1% for Falcon-1024. Thus, the magnitude filter alone retains

well above $n/2$ coefficients on average while removing large outlying estimates. The rounding threshold $\tau_2 = 0.25$ keeps estimates that are close to an integer. In particular, we use r_i as an indication of confidence: a smaller r_i means that rounding changes the estimate by less. A coefficient is retained as a candidate hint only when $|\hat{f}_i| \leq \tau_1$ and $r_i < \tau_2$. The resulting candidate index set is

$$\mathcal{D}_{\tau_1, \tau_2} \stackrel{\text{def}}{=} \left\{ i \in [n] : |\hat{f}_i| \leq \tau_1 \wedge r_i < \tau_2 \right\}.$$

After applying both filters, we proceed only when $|\mathcal{D}_{\tau_1, \tau_2}| \geq n/2$. We combine the ISD-style search with Cao et al.'s [3] Gaussian-elimination construction of the residual lattice. The search selects an information set $\mathcal{I} \subseteq \mathcal{D}_{\tau_1, \tau_2}$, and the values $\hat{f}_i$ for $i \in \mathcal{I}$ are used as hints in the lattice step. We consider only sets with $|\mathcal{I}| \geq n/2$, corresponding to at least 256 coefficients for Falcon-512 and 512 for Falcon-1024.

We order the candidate indices in $\mathcal{D}_{\tau_1, \tau_2}$ as $i_1, i_2, \dots$ by decreasing residual, so that i_1 has the lowest confidence. The first attempt uses all coefficients in $\hat{f}_{\mathcal{D}_{\tau_1, \tau_2}}$, i.e., $\mathcal{I} = \mathcal{D}_{\tau_1, \tau_2}$. If it fails, we remove i_1 and try again. We then consider the two least-confident coefficients and test the two new choices $\{i_2\}$ and $\{i_1, i_2\}$. After adding i_3 , we test all new combinations that contain i_3 : $\{i_3\}$, $\{i_1, i_3\}$, $\{i_2, i_3\}$, and $\{i_1, i_2, i_3\}$. The same procedure is repeated after adding i_4 and further low-confidence coefficients. In summary, let $\mathcal{S}$ be a removal set containing the indices of the candidate coefficients excluded in one attempt. For each $\mathcal{S}$, we use the information set $\mathcal{I} = \mathcal{D}_{\tau_1, \tau_2} \setminus \mathcal{S}$ and run the lattice-recovery step. The removal sets are selected in the order described above, starting with combinations of the least-confident coefficients.

The lattice construction treats every $\hat{f}_i$ with $i \in \mathcal{I}$ as an exact value. These values are substituted into the public-key relation during Gaussian elimination and are therefore not variables in the residual lattice. If one of them is wrong, the true f does not satisfy the resulting system and cannot be recovered from this lattice. The search then tries the next information set.

Each correct coefficient in $\mathcal{I}$ can be regarded as a perfect axis-aligned hint [6], meaning that it directly fixes one coefficient of f . Following the approach of Cao et al., we apply these hints to Equation (3), which relates the public key h to the short secret polynomials f and g . Let $\mathcal{U} = [n] \setminus \mathcal{I}$ be the unknown indices, and let $H_{\mathcal{I}}$ and $H_{\mathcal{U}}$ contain the columns indexed by $\mathcal{I}$ and $\mathcal{U}$, respectively. We can then write $Hf = H_{\mathcal{I}}f_{\mathcal{I}} + H_{\mathcal{U}}f_{\mathcal{U}}$. Replacing $f_{\mathcal{I}}$ with the selected coefficients $\hat{f}_{\mathcal{I}}$ gives

$$H_{\mathcal{U}} \cdot f_{\mathcal{U}} \equiv g - H_{\mathcal{I}} \cdot \hat{f}_{\mathcal{I}} \pmod{q}, \quad (10)$$

This reduced relation allows Gaussian elimination to be applied efficiently. The remaining task is to find the short pair $(f_{\mathcal{U}}, g)$ satisfying

$$H_{\mathcal{U}}f_{\mathcal{U}} - g \equiv -H_{\mathcal{I}}\hat{f}_{\mathcal{I}} \pmod{q}.$$

This is an inhomogeneous short-integer-solution instance solved by the standard Kannan embedding. We select m equations such that $n - |\mathcal{I}| < m \leq n$. The

superscript (m) denotes restriction to these same m equations: $H_{\mathcal{U}}^{(m)}$ contains the corresponding rows of $H_{\mathcal{U}}$, while $g^{(m)}$ and $(H_{\mathcal{I}}\hat{f}_{\mathcal{I}})^{(m)}$ contain the corresponding coordinates. The lattice

$$\Lambda = \left\{ (x, y, z) \in \mathbb{Z}^{(n-|\mathcal{I}|)+m+1} : H_{\mathcal{U}}^{(m)}x - y \equiv -z(H_{\mathcal{I}}\hat{f}_{\mathcal{I}})^{(m)} \pmod{q} \right\} \quad (11)$$

contains the target $(f_{\mathcal{U}}, g^{(m)}, 1)$. This target is short because $f_{\mathcal{U}}$ and $g^{(m)}$ contain coefficients of the short Falcon secret polynomials. Using more equations than unknown coefficients oversamples the system and further lowers the required block size, but each added equation also adds one dimension to the residual lattice Λ . These two effects work against each other, so m is chosen to minimize the required block size but not simply set to its maximum n . A larger correct information set leaves fewer unknown coefficients and makes this reduction easier by lowering the block size.

For a candidate f returned by lattice reduction, we compute $g = Hf \bmod q$ and represent each coefficient of g in the centered interval $[-\lfloor q/2 \rfloor, \lfloor q/2 \rfloor]$. We accept the candidate only if every centered coefficient of g satisfies $|g_i| \leq 5\sigma_f$. This check rejects candidates that are inconsistent with the expected coefficient size of the short secret polynomial g . When a candidate fails the check, the search moves to the next information set $\mathcal{I}$. The procedure stops when a candidate passes the check or all information sets up to the maximum search depth $k_{\max}$ have been tested. The number of correct hints is not itself a pass-or-fail condition. Each additional correct hint eliminates one unknown coefficient and lowers the block size required by the residual lattice, so key recovery becomes feasible gradually instead of at a fixed hint count. At $n/2$ correct hints the residual lattice is still out the reach of a practical lattice reduction budget for a normal PC, so the required number of signatures is determined by when the residual lattice becomes reducible within the computational budget, not by when $n/2$ correct hints become available.

5 Experimental Results

This section is to test whether the leakage described in Section 3.2 can be identified from real power traces and whether the resulting classification errors still allow key recovery. The measured confusion matrix connects these two parts: it records the errors observed in the side-channel experiment and is then used to evaluate the recovery of f and the subsequent ISD-style search and lattice-recovery step. We also study sign-only leakage at different error rates to examine how classification accuracy affects key recovery.

Measurement setup. We run PQCclean Falcon on a ChipWhisperer CW308 board equipped with an STM32F415 Cortex-M4 operating at 12.5 MHz. The implementation is compiled with `arm-none-eabi-gcc` 13.2.1 at either `-O0` or `-O3`. Power traces are acquired using a PicoScope 5244D at 65.5 MS/s for `-O0` and 125 MS/s for `-O3`. We use a lower sampling rate for `-O0` because its longer execution time would produce unnecessarily long traces. A GPIO trigger placed

immediately before `fpr_of(s)` is used to align the traces. For each compiler level, `-00` and `-03`, we collect 680,000 power traces. We use 80% of the traces for training and 20% for testing, with 20% of the training set reserved for validation. The accuracy reported in Table 3 is evaluated based on 136,000 traces for test set.

During profiling, we train the CNN classifier with root mean square propagation (RMSprop) for at most 15 epochs, using a learning rate of 10^{-3} and a batch size of 512. Both models use three Conv1D layers with ReLU activation and average pooling, followed by two fully connected layers and a 17-class softmax output. The model for `-00` setting contains about 17.92 M parameters, while the one for `-03` setting contains about 3.64 M. Note that `-03` setting uses a smaller network, since it is faster and produces short input window. Table 2 gives the full information of the network.

Table 2. CNN hyperparameters for signed-exponent classification.

Layer / parameter	-00 model	-03 model
Conv1D 1 (filters / kernel / pool)	82 / 10 / 2	66 / 10 / 4
Conv1D 2 (filters / kernel / pool)	85 / 9 / 3	91 / 14 / 2
Conv1D 3 (filters / kernel / pool)	71 / 17 / 2	100 / 14 / 2
Dense 1 (units / dropout)	401 / 0.26	202 / 0.11
Dense 2 (units / dropout)	447 / 0.43	147 / 0.21
Output (softmax)	17	17
Total parameters	~17.92 M	~3.64 M

5.1 Side-Channel Classification Results

Table 3 reports the accuracy of single-trace classification for both leakage settings. The accuracy of our signed-exponent classifier reaches 0.9997 at `-00` and 0.8838 at `-03`, and the classifier for sign-only leakage reaches an accuracy of 1 at both optimization levels. The lower signed-exponent accuracy at `-03` shows that compiler optimization makes the power-of-two intervals harder to distinguish.

Table 3. Measured single-trace classification accuracy.

Leakage	Class map	-00	-03
Signed exponent	<code>exp(s)</code>	0.9997	0.8838
Sign only	<code>sign(s)</code>	1.0	1.0

The accuracy of sign-only classification is 1.0 at both `-00` and `-03`. The corresponding confusion matrices are therefore identity matrices, with ones on the diagonal and zeros elsewhere. The two signed-exponent confusion matrices below give the full distribution of the classification errors at `-00` and `-03`. In each matrix, rows represent the true signed-exponent classes and columns represent the

observed signed-exponent classes. At -03, signed-exponent classification is less accurate. Its accuracy is only 0.8838, and the incorrect predictions are spread across many other classes rather than concentrated in a few classes. However, it should be noted that our proposed key-recovery method remains robust to classification errors. In the following subsection, we further evaluate this robustness by applying our attack to simulated sign-only leakage at different noise levels and reporting the number of required signatures.

$$P_{\text{exp}}^{-00} = \begin{pmatrix} 0.998 & 0.0025 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.998 & 0.0025 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & 0.0 \end{pmatrix}$$

$$P_{\text{exp}}^{-03} = \begin{pmatrix} 0.895 & 0.02 & 0.0033 & 0.0 & 0.0 & 0.0017 & 0.0017 & 0.0033 & 0.0017 & 0.0033 & 0.0017 & 0.0017 & 0.0017 & 0.0 & 0.0533 & 0.0 & 0.0117 \\ 0.0117 & 0.888 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0067 & 0.005 & 0.0017 & 0.0017 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0783 & 0.0017 & 0.005 \\ 0.0017 & 0.0167 & 0.9 & 0.0183 & 0.0 & 0.0 & 0.0017 & 0.005 & 0.0017 & 0.0017 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0017 & 0.0483 & 0.0 & 0.0033 \\ 0.0033 & 0.005 & 0.0117 & 0.86 & 0.025 & 0.0017 & 0.0067 & 0.0033 & 0.0033 & 0.0033 & 0.0 & 0.0017 & 0.0067 & 0.0 & 0.0633 & 0.0033 & 0.0017 \\ 0.0067 & 0.0 & 0.0083 & 0.0283 & 0.772 & 0.112 & 0.0033 & 0.0017 & 0.0 & 0.0033 & 0.0033 & 0.0017 & 0.005 & 0.0017 & 0.0467 & 0.0 & 0.0067 \\ 0.0017 & 0.0017 & 0.0 & 0.0 & 0.0083 & 0.828 & 0.0717 & 0.0017 & 0.0017 & 0.0033 & 0.0017 & 0.0033 & 0.0033 & 0.0017 & 0.06 & 0.005 & 0.0067 \\ 0.0067 & 0.005 & 0.0 & 0.0017 & 0.0 & 0.115 & 0.777 & 0.0267 & 0.0 & 0.0017 & 0.0017 & 0.0 & 0.0 & 0.0 & 0.0583 & 0.0033 & 0.0033 \\ 0.0017 & 0.0 & 0.0033 & 0.0 & 0.0 & 0.0033 & 0.005 & 0.915 & 0.0033 & 0.0033 & 0.0033 & 0.0 & 0.0017 & 0.0 & 0.0567 & 0.0 & 0.0033 \\ 0.005 & 0.0017 & 0.0033 & 0.0 & 0.0 & 0.0017 & 0.0033 & 0.0017 & 0.9 & 0.0017 & 0.0 & 0.0 & 0.0017 & 0.0033 & 0.065 & 0.0067 & 0.005 \\ 0.0 & 0.0017 & 0.0033 & 0.0 & 0.0017 & 0.0033 & 0.0017 & 0.0017 & 0.0017 & 0.913 & 0.0017 & 0.0017 & 0.0033 & 0.0 & 0.0517 & 0.005 & 0.0083 \\ 0.0067 & 0.0 & 0.0017 & 0.0 & 0.0017 & 0.0017 & 0.0033 & 0.0017 & 0.0017 & 0.0 & 0.908 & 0.0 & 0.0033 & 0.0017 & 0.0633 & 0.0033 & 0.0017 \\ 0.005 & 0.0033 & 0.005 & 0.0017 & 0.0 & 0.0033 & 0.0017 & 0.0033 & 0.0 & 0.0017 & 0.0 & 0.89 & 0.005 & 0.0033 & 0.07 & 0.0033 & 0.0033 \\ 0.0017 & 0.0 & 0.0033 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0067 & 0.0 & 0.0017 & 0.0 & 0.0 & 0.917 & 0.0 & 0.06 & 0.0033 & 0.0067 \\ 0.0033 & 0.0033 & 0.0017 & 0.0 & 0.0033 & 0.005 & 0.005 & 0.0 & 0.0033 & 0.0017 & 0.0 & 0.0 & 0.0 & 0.9 & 0.0683 & 0.0017 & 0.0033 \\ 0.005 & 0.005 & 0.0 & 0.0033 & 0.0017 & 0.0 & 0.0033 & 0.0 & 0.0 & 0.0033 & 0.0 & 0.0017 & 0.0033 & 0.0 & 0.963 & 0.0 & 0.01 \\ 0.0033 & 0.0033 & 0.0033 & 0.0 & 0.0017 & 0.0017 & 0.005 & 0.0033 & 0.0 & 0.005 & 0.0 & 0.0 & 0.0017 & 0.0033 & 0.0633 & 0.897 & 0.0 \\ 0.0017 & 0.0067 & 0.0033 & 0.0017 & 0.0033 & 0.0017 & 0.0 & 0.0017 & 0.0033 & 0.0017 & 0.0 & 0.0 & 0.0017 & 0.0033 & 0.065 & 0.0033 & 0.902 \end{pmatrix}$$

5.2 Key-Recovery Results

We evaluate both signed-exponent and sign-only leakage. For each setting, we repeat the experiment 100 times, using an independently generated key in each run. The success rate is the fraction of runs that succeed. For signed-exponent leakage, the observed classes are sampled from the measured confusion matrix for -00 or -03. CB-MLE only needs to recover enough reliable coefficients of f for the following ISD-style search and lattice-reduction step. We estimate the recovery cost as the number of ISD-style search trials multiplied by the cost of one lattice reduction. The latter is estimated from the required BKZ block size β using the Core-SVP cost $2^{0.292\beta+16.4}$ gates [1,6,3]. We attempt lattice recovery when the estimated cost is below 2^{50} gates. An attack is counted as successful only when the actual lattice reduction recovers the complete and correct f .

Table 4. Full key-recovery results with signed-exponent leakage.

	Falcon-512 ($n/2 = 256$)		Falcon-1024 ($n/2 = 512$)	
	-00	-03	-00	-03
Signatures	20	56	21	100
Success rate	100%	100%	100%	100%

Table 4 shows that the signed-exponent attack reaches a full key-recovery rate of 100% in all four settings. The -03 settings require more signatures because the classification accuracy falls from approximately 1.00 to 0.884. At the signature counts reported in Table 4, CB-MLE always recovers far more than $n/2$ correct coefficients of f , with at most one incorrect coefficient.

Figure 1 further shows how the fraction of runs meeting the coefficient condition changes with N for the four parameter-and-compiler settings. At $N = 10$, the rates are 90% and 86% for Falcon-512 and Falcon-1024 at -00, but only 40% and 9% at -03. At -03, the rates for Falcon-512 and Falcon-1024 rise to 93% and 89% at $N = 30$, and to 98% and 97% at $N = 40$. Most runs meet the condition quickly. Falcon-1024 at -03, for example, reaches 99% at $N = 60$ and 100% at $N = 87$. These curves describe the output of CB-MLE before the ISD-style search and lattice-reduction step; Table 4 reports the full key-recovery rates after those steps.

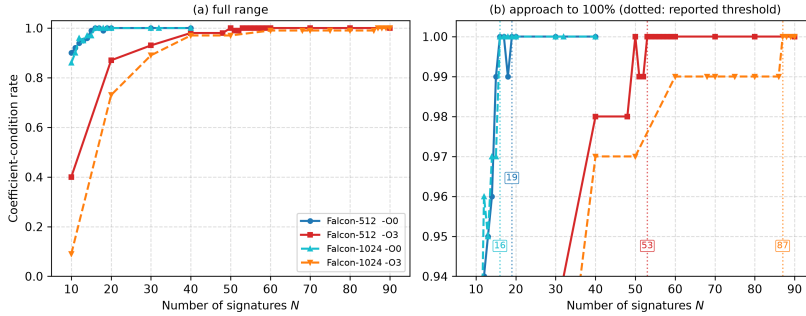


Fig. 1. Fraction of runs meeting the coefficient condition for each setting.

Sign-only leakage has two classes. Under the symmetric-error model used here, its 2×2 confusion matrix is specified by a single classification error rate ρ : the diagonal entries of P^{sign} are $1 - \rho$, and the off-diagonal entries are ρ . We can therefore vary ρ directly to study how the recovery method behaves under different noise levels. Doing the same for signed-exponent leakage would require a full 17×17 matrix.

We also compare our method with Brinkmann et al. [2], who study Falcon-512 under simulated sign-classification errors. We report the fraction of runs in which the selected coefficient set contains at least $n/2$ correct coefficients of f and no more than one wrong coefficient. This condition is checked against the known key in the experiment and indicates that the output can be passed to the subsequent recovery step. Similarly, we also attempt lattice recovery when the estimated cost is below 2^{50} gates. An attack is counted as successful only when the actual lattice reduction recovers the complete and correct f .

Table 5. Falcon-512 results with sign-only leakage.

ρ	Brinkmann et al. [2]		This work	
	Sigs.	Key Succ.	Sigs.	Key Succ.
0	100 [†]	100%	230	100%
0.05	2,400	40%	400	100%
0.10	3,800	25%	458	100%
0.15	5,100	20%	625	100%
0.20	6,000	5%	800	100%
0.25	10,000	5%	1,230	100%
0.30	<i>N/A</i>	<i>N/A</i>	2,160	100%

[†] Brinkmann et al. estimate that their dimension-halving step introduces a sign-error probability of about 1.0% even when $\rho = 0$; our $\rho = 0$ row contains no sign errors.

Table 5 compares the performance of our attack with that of Brinkmann et al. for ρ from 0 to 0.30 in steps of 0.05. Our attack achieves a 100% success rate at each listed number of signatures. For ρ from 0.05 to 0.25, the success rate reported by Brinkmann et al. falls from 40% to 5%. It should be noted that our attack and that of Brinkmann et al. exploit the signs of different intermediate variables (μ in our attack and z in theirs), and the number of required signatures is the maximum one for all tested 100 keys in our attack while the average number in theirs. Therefore, the results cannot be regarded as a complete end-to-end comparison.

6 Countermeasures

The leakage comes from two parts of `fpr_of( $\ell$ )`, i.e., the sign of ℓ carried by the binary64 sign bit and its exponent determining the shifts in `FPR_NORM64`. Changing only the control flow of `SamplerZ` removes none of them. Therefore, the defense must cover both parts of `fpr_of` and the preceding `fpr_floor( $\mu$ )`, which handles the same secret-dependent center.

Keeping binary64 signing would require masking `fpr_of` and `fpr_floor`, or otherwise equalizing their side-channel traces. Current masking schemes for

Falcon’s floating-point multiplication and addition cost $3\text{--}10\times$ [4], more importantly, they do not cover these floating-point conversions. Chen et al. [5] also show why incomplete protection is risky, that is, they broke a masked multiplication through an unmasked addition. At a sampler leaf, hiding only the sign or only the exponent likewise leaves useful information for an attacker. Profiling should therefore be performed at the protected conversion itself.

Existing sampler defenses cover different sources of leakage. Howe et al. remove timing leakage with an isochronous sampler [13]. Lin et al. reduce z-sign classification accuracy to at most 62% on Cortex-M4 with a constant-path **SamplerZ**, at a cost of about $3.5\times$ [17]. Maskaglia masks the base discrete-Gaussian sampler for the fixed- σ setting used by HAWK [12]. None of these defenses protects **fpr_floor** or **fpr_of**. A full sampler defense must protect these functions, harden the sampler control flow, and suppress z-sign leakage. Our attack does not use z-sign leakage. At -00 , the leakage from **fpr_of** gives a 100% recovery rate from 20 signatures for Falcon-512 and from 21 signatures for Falcon-1024 (Table 4).

A fixed-point signer would not contain the representation exploited by our attack. For example, the design of De Almeida Braga et al. [7] uses a public radix and therefore has neither an IEEE-754 exponent nor **FPR_NORM64**. This approach replaces the signing arithmetic instead of patching a single conversion. The current FIPS 206 validation plan requires signing to use native or emulated binary64 arithmetic and to match the specified known-answer tests, and it states that fixed point is allowed only for key generation [19]. Fixed-point signing is thus a possible future design but not a small modification to a conforming implementation. It would still need side-channel protection because an unmasked fixed-point value may also reveal its magnitude.

Key rotation reduces the number of available signatures to an attacker, so it only mitigates but does not eliminate the threat. Figure 1 shows that many keys already meet the coefficient condition for key recovery before the counts in Table 4, so these values are not safe rotation limits and they are already small considering a realistic usage of signings. In addition, other proposed defenses for Falcon, the one for the FFT conditional calculator [14], **mq_NTT** reduction [22], or floating-point pre-image computation [5] do not cover the sampler-leaf conversion either. In this regard, each of these leakage points must be treated separately while protected systematically.

7 Conclusion

We presented a profiled side-channel attack on Falcon that uses the sign and exponent leaked during the integer-to-binary64 conversion at each Gaussian-sampler leaf. We represent this leakage with 17 signed-exponent classes. CB-MLE uses the measured confusion matrix to score the observed classes and estimate the secret polynomial f . The recovered coefficients form candidate hints for the ISD-style search. For each information set, Gaussian elimination incorporates the selected hints into a reduced NTRU lattice, and LLL/BKZ searches for the

remaining key. On Cortex-M4, the signed-exponent classifier reaches single-trace accuracies of 0.9997 at -00 and 0.884 at -03. The full attack recovers Falcon-512 key with success rates of 100% from 20 signatures at -00 and 100% from 56 signatures at -03. For Falcon-1024, the corresponding results are 100% from 21 signatures and 100% from 100 signatures. These signature counts are one to three orders of magnitude below those of prior sampler attacks.

References

1. Albrecht, M.R., Göpfert, F., Virdia, F., Wunderer, T.: Revisiting the expected cost of solving usvp and applications to LWE. In: Takagi, T., Peyrin, T. (eds.) *Advances in Cryptology – ASIACRYPT 2017, Proceedings, Part I. Lecture Notes in Computer Science*, vol. 10624, pp. 297–322. Springer (2017). https://doi.org/10.1007/978-3-319-70694-8_11
2. Brinkmann, M., Kraus, N., May, A.: Halfspace learning for lattice signature key recovery from signs. In: Heninger, N., Rosulek, M. (eds.) *Advances in Cryptology – CRYPTO 2026, Proceedings, Part III. Lecture Notes in Computer Science*, vol. 16802, pp. 359–390. Springer Nature Switzerland (2026). https://doi.org/10.1007/978-3-032-35377-1_12
3. Cao, J., Jiang, H., Cheng, Q.: Refined attack on LWE with hints: Constructing lattice via gaussian elimination. In: Kalai, Y.T., Kamara, S.F. (eds.) *Advances in Cryptology – CRYPTO 2025, Proceedings, Part I. Lecture Notes in Computer Science*, vol. 16000, pp. 385–416. Springer (2025). https://doi.org/10.1007/978-3-032-01855-7_13
4. Chen, K.Y., Chen, J.P.: Masking floating-point number multiplication and addition of Falcon: First- and higher-order implementations and evaluations. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* **2024**(2), 276–303 (2024). <https://doi.org/10.46586/TCHES.V2024.I2.276-303>
5. Chen, K.Y., Ching, M.Q., Chen, J.P., Yang, B.Y.: When masking multiplication isn’t enough: Exploiting floating-point leakage in Falcon’s pre-image computation. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* **2026**(2), 686–708 (2026). <https://doi.org/10.46586/TCHES.V2026.I2.686-708>
6. Dachman-Soled, D., Ducas, L., Gong, H., Rossi, M.: LWE with side information: Attacks and concrete security estimation. In: Micciancio, D., Ristenpart, T. (eds.) *Advances in Cryptology – CRYPTO 2020, Proceedings, Part II. Lecture Notes in Computer Science*, vol. 12171, pp. 329–358. Springer (2020). https://doi.org/10.1007/978-3-030-56880-1_12
7. De Almeida Braga, D., Fouque, P.A., Lachguel, B., Prest, T.: Toward a secure fixed-point implementation of the Falcon signature scheme. In: Heninger, N., Rosulek, M. (eds.) *Advances in Cryptology – CRYPTO 2026, Proceedings, Part III. Lecture Notes in Computer Science*, vol. 16802, pp. 166–197. Springer Nature Switzerland (2026). https://doi.org/10.1007/978-3-032-35377-1_6
8. Espitau, T., Fouque, P., Gérard, F., Rossi, M., Takahashi, A., Tibouchi, M., Wallet, A., Yu, Y.: Mitaka: A simpler, parallelizable, maskable variant of Falcon. In: Dunkelman, O., Dziembowski, S. (eds.) *Advances in Cryptology – EUROCRYPT 2022, Proceedings, Part III. Lecture Notes in Computer Science*, vol. 13277, pp. 222–253. Springer (2022). https://doi.org/10.1007/978-3-031-07082-2_9

9. Fouque, P.A., Gérard, F., Rossi, M., Yu, Y.: Zalcon: An alternative FPA-free NTRU sampler for Falcon. In: Proceedings of the 3rd NIST PQC Standardization Conference (2021), <https://csrc.nist.gov/CSRC/media/Events/third-pqc-standardization-conference/documents/accepted-papers/>
10. Gentry, C., Peikert, C., Vaikuntanathan, V.: Trapdoors for hard lattices and new cryptographic constructions. In: Dwork, C. (ed.) Proceedings of the 40th Annual ACM Symposium on Theory of Computing (STOC 2008). pp. 197–206. ACM (2008). <https://doi.org/10.1145/1374376.1374407>
11. Guerreau, M., Martinelli, A., Ricosset, T., Rossi, M.: The hidden parallelepiped is back again: Power analysis attacks on Falcon. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2022**(3), 141–164 (2022). <https://doi.org/10.46586/TCHES.V2022.I3.141-164>
12. Haidar, C.A., Espitau, T., Hoffmann, C., Tibouchi, M.: Maskaglia: A new, efficient approach to masked discrete gaussian sampling. In: Heninger, N., Rosulek, M. (eds.) Advances in Cryptology – CRYPTO 2026, Proceedings, Part VII. Lecture Notes in Computer Science, vol. 16806, pp. 164–195. Springer Nature Switzerland (2026). https://doi.org/10.1007/978-3-032-35415-0_6
13. Howe, J., Prest, T., Ricosset, T., Rossi, M.: Isochronous gaussian sampling: From inception to implementation. In: Ding, J., Tillich, J. (eds.) Post-Quantum Cryptography – PQCrypto 2020 Proceedings. Lecture Notes in Computer Science, vol. 12100, pp. 53–71. Springer (2020). https://doi.org/10.1007/978-3-030-44223-1_4
14. Kim, G., Lee, J., Lee, M., Hong, S., Kim, H.: Secret key recovery of FALCON using simple power analysis in conditional calculator. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2026**(2), 953–975 (2026). <https://doi.org/10.46586/TCHES.V2026.I2.953-975>
15. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. In: 3rd International Conference on Learning Representations (ICLR 2015) (2015), <https://arxiv.org/abs/1412.6980>
16. Li, K., Ma, S., Dou, H., Guo, Q.: One fell swoop: A single-trace key-recovery attack on the Falcon signing algorithm. IACR Cryptol. ePrint Arch. **2025**, 2159 (2025), <https://eprint.iacr.org/2025/2159>
17. Lin, X., Zhang, S., Yu, Y., Wang, W., You, Q., Xu, X., Wang, X.: Thorough power analysis on Falcon gaussian samplers and practical countermeasure. In: Jager, T., Pan, J. (eds.) Public-Key Cryptography – PKC 2025, Proceedings, Part I. Lecture Notes in Computer Science, vol. 15674, pp. 229–258. Springer (2025). https://doi.org/10.1007/978-3-031-91820-9_8
18. Noh, J., Choi, H., Han, D., Shin, D.: Efficient error-tolerant side-channel attacks on GPV signatures based on ordinary least squares regression. IACR Cryptol. ePrint Arch. **2024**, 2043 (2024), <https://eprint.iacr.org/2024/2043>
19. Perlner, R.A.: FIPS 206: FN-DSA (Falcon) – status update. NIST CSRC 6th PQC Standardization Conference (2025), <https://csrc.nist.gov/CSRC/media/Presentations/fips206-fn-dsa-falcon-status-update/>
20. Prange, E.: The use of information sets in decoding cyclic codes. IRE Trans. Inf. Theory **8**(5), 5–9 (1962). <https://doi.org/10.1109/TIT.1962.1057777>
21. Prest, T., Fouque, P.A., Hoffstein, J., Kirchner, P., Lyubashevsky, V., Pornin, T., Ricosset, T., Seiler, G., Whyte, W., Zhang, Z.: Falcon: Fast-fourier lattice-based compact signatures over ntru. Tech. rep., NIST (2022), <https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022>, NIST Post-Quantum Cryptography Standardization, Selected Algorithm

- 22. Qiu, J., Aysu, A.: NTT-PEEL: Bit shift side-channel in FALCON's number theoretic transform. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* **2026**(2), 709–735 (2026). <https://doi.org/10.46586/TCHES.V2026.I2.709-735>
- 23. Zhang, S., Lin, X., Yu, Y., Wang, W.: Improved power analysis attacks on Falcon. In: Hazay, C., Stam, M. (eds.) *Advances in Cryptology – EUROCRYPT 2023, Proceedings, Part IV*. *Lecture Notes in Computer Science*, vol. 14007, pp. 565–595. Springer (2023). https://doi.org/10.1007/978-3-031-30634-1_19