One Fell Swoop: A Single-Trace Key-Recovery Attack on the Falcon Signing Algorithm

Kang Li[1], Shouran Ma[2], Haochen Dou[2] and Qian Guo[2*]

[1] Jinan University, Guangzhou, China
kanglee175@163.com
[2] Lund University, Lund, Sweden
shouran.ma@eit.lth.se, haochen.dou@eit.lth.se, qian.guo@eit.lth.se

Abstract. Falcon, a lattice-based signature scheme selected for NIST post-quantum standardization, is notable for its compact signature size alongside a complex signing procedure involving extensive floating-point arithmetic. Prior side-channel attacks on Falcon, while demonstrating vulnerabilities, have consistently required a large number of power traces for successful key recovery; this critical efficiency gap means previously reported attacks are often impractical in real-world scenarios where trace collection is limited.

This paper presents a new single-trace attack on the Falcon. We identify and exploit novel leakage points within the floating-point conversion and Fast Fourier Transform (FFT) routines during the secret key expansion, which allow us to progressively partition the possible values of the secret key coefficients. By identifying a sufficient number of these coefficients, we establish a system of linear equations that can be solved to recover the entire secret key. Our attack is particularly critical for the `sign_dyn` design—the memory-efficient implementation adopted in important cryptographic libraries and reference implementations—as it executes key expansion during every signature operation. We emphasize that this is the **first single-trace attack on the Falcon signing procedure itself**, providing a more compelling threat scenario than previous work.

We validate our attack on an ARM Cortex-M4 microcontroller, demonstrating a 100% key recovery success rate with just a single power trace for both Falcon-512 and Falcon-1024 in both signing designs—`sign_tree` and `sign_dyn`, compiled at the `-O0` level. While the `-O3` optimization level mitigates some leakages, our multi-trace attack remains effective in the practically used `sign_dyn` design, recovering 80 out of 100 Falcon-512 keys with only 5 traces. Our findings expose a critical implementation vulnerability in Falcon, highlighting the urgent necessity of integrating countermeasures to protect Falcon in real-world applications.

Keywords: NIST post-quantum cryptography standardization · Lattice-based cryptography · Side-channel attacks · Falcon

1 Introduction

Falcon [PFH+22] is a lattice-based post-quantum signature scheme built on the NTRU lattice and has been selected by NIST as one of the three signatures for standardization. Compared with the other two signatures (Dilithium [LDK+22], SPHINCS+ [HBD+22]), Falcon achieves the smallest combined size of public key and signature, making it especially attractive for embedded platforms. It also offers efficient signing and verification procedures.

*Corresponding author.

As a product of many years of work, Falcon's theoretical foundations have undergone extensive scrutiny. However, as Falcon relies on the GPV hash-and-sign framework [GPV08] and is instantiated through fast Fourier sampling [DP16], the signing procedure becomes inherently complex and involves extensive floating-point (FP) operations. These characteristics expose Falcon to some side-channel attacks. Fouque et al. [FKT+20] first proposed a timing attack against the signing procedure, which was later addressed by the introduction of the isochronous Gaussian sampler [HPRR20]. Building upon this sampler, a constant-time implementation of Falcon is implemented by Pornin [Por19]. Yet, even in its constant-time form, Falcon remains vulnerable to power or electromagnetic leakage attacks.

Currently, two operations in Falcon's signing procedure are known to leak secret information: pre-image computation and integer Gaussian sampling. Regarding the pre-image computation, Karabulut et al. [KA21] first exploited FP multiplications in the Fourier domain to launch a correlation power analysis (CPA) for key recovery targeting the Falcon-512, and the CPA attack requires 9,000 – 10,000 electromagnetic traces to succeed. Later, Guerreau et al. [GMRR22] improved their method by significantly reducing the enumeration space when attacking the mantissa multiplication. Their experiments show that 5,000 power traces are needed to recover the secret key. More recently, Chen et al. [CSM+25] proposed the SERP-SCA attack framework, which further advances this research line and achieves notable improvements in both success rate and efficiency. The required number of power traces is reduced to about 3,000.

As for the integer Gaussian sampling, Guerreau et al. [GMRR22] introduced the first attack targeting the so-called half Gaussian leakage in the integer Gaussian sampler. Their attack is inspired by the hidden parallelepiped attack [NR06, DN12], and needs at least 1,000,000 power traces to have a reasonable post-processing time for recovering the key. Subsequently, Zhang et al. [ZLYW23] enhanced this attack and identified a new leakage named the sign leakage in the integer Gaussian sampler. By the enhanced attack, at least 45,000 traces are required by combining the two leakages. Later, their attack is refined by Lin et al. [LZY+25] by exploiting the symmetry of the NTRU lattice structure and introducing a more powerful decoding technique. The minimum number of power traces required is reduced to 6,500. It is worth noting that the attacks in [ZLYW23] and [LZY+25] can only achieve about a 25% success rate.

Motivation As summarized in Table 1, prior side-channel analyses of Falcon have consistently relied on large datasets, typically requiring thousands of power or electromagnetic traces to succeed. While these studies demonstrate theoretical vulnerabilities, the requirement for such extensive data collection often renders them impractical against real-world targets where access may be limited or transient. In contrast, an attack requiring only a **single trace or a small number of traces** represents the most severe threat model, as it can succeed with minimal measurements.

While single-trace attacks have been explored for various lattice-based schemes [PPM17, PP19a, KH18, EFGT17, QLZ+24, ACLZ20, KAA21], research on Falcon in this rigorous setting remains scarce. To the best of our knowledge, the only existing single-trace attack on Falcon is the work of Qiu et al. [QA25]. However, their method targets the discrete Gaussian sampling process during the *key generation* phase. This is a significant limitation: key generation is typically a "one-off" event performed in a controlled environment, making it difficult for an attacker to capture the specific trace.

This paper addresses the critical need for a single-trace attack on the *signing* procedure, which is executed repeatedly and is thus far more exposed. Our motivation stems from the distinct implementation choices available in Falcon:

- `sign_tree`: A design optimized for **speed**, which pre-computes and stores the expanded key.

Table 1: Comparison of the number of traces required for key-recovery attacks on the Falcon-512. Numbers with a superscript * indicate there is a trade-off between the number of traces and post-processing computation. We present numbers corresponding to a reasonable post-processing time. Except for work [KA21], which collects electromagnetic traces, all other works collect power traces. Abbreviations: PIC = Pre-image computation, HGL = Half Gaussian leakage, SL = Sign leakage, DGS = discrete Gaussian sampling, FPCM = Floating Point Conversion and Multiplication.

Work	**Target**	**#Trace**	**Phase**
[KA21]	PIC	9,000 – 10,000	signing
[GMRR22]	PIC	5,000	signing
	HGL	1,000,000*	
[ZLYW23]	HGL	220,000*	signing
	SL	170,000*	
	HGL & SL	45,000*	
[LZY+25]	HGL	27,500*	signing
	SL	25,000*	
	HGL & SL	6,500*	
[CSM+25]	PIC	3000	signing
[QA25]	DGS	1	key generation
This Work (-O0)	**FPCM**	**1**	**signing**
This Work (-O3)	**FPCM**	**5**	**signing**

- `sign_dyn`: A design optimized for **memory efficiency**, which re-computes the expanded key on-the-fly during every signature generation.

Because the `sign_dyn` design allows Falcon to fit on memory-constrained devices (where side-channel attacks are most relevant), it is the standard choice for embedded libraries. By identifying novel leakages in the FP conversions and FP multiplications at the beginning of the Fast Fourier Transform (FFT) that occur during this on-the-fly expansion, we turn the memory-saving feature of `sign_dyn` into a security liability. Unlike key generation, which happens once, `sign_dyn` performs these vulnerable expansion operations for *every* signature. This allows us to mount a single-trace attack on the signing process itself, significantly elevating the practical threat level compared to prior work.

Contributions Our contributions are as follows:

- We present a novel key-recovery side-channel attack that is independent of all previous attacks on Falcon. The method exploits newly identified side-channel leakage points within the FP conversion and FP multiplication steps at the beginning of the FFT routine. These leakages allow us to restrict the possible values of secret key coefficients to smaller candidate sets.
- We successfully conduct a single-trace attack against both reference designs at the `-O0` optimization level. It is important to note that while the `sign_tree` design performs key expansion only once during the key generation phase, the `sign_dyn` design invokes key expansion during the signing phase. Therefore, our attack on the practically used `sign_dyn` design constitutes the first single-trace attack on the Falcon signing procedure itself.
- Our attack framework also provide a multi-trace attack to the practically used `sign_dyn` design by incorporating maximum likelihood estimation and an ISD-style key recovery strategy under noisy side-channel conditions (e.g., caused by the advanced compilation level). The extended attack exhibits a certain degree of error tolerance, which significantly improves the success rate of key recovery.

- We validate the attack against Falcon's reference implementation on an ARM Cortex-M4 microcontroller. At the `-O0` optimization level, we achieve 100% key recovery with a single power trace for Falcon-512 and Falcon-1024 in both signing designs. At the `-O3` level, although some leakages are mitigated, our multi-trace attack recovers 80 out of 100 Falcon-512 keys with only 5 traces in the `sign_dyn` design, demonstrating its continued practicality. This result significantly improves the state-of-the-art, as shown in Table 1.

Organizations The remainder of this paper is organized as follows. Section 2 introduces the threat model and provides the necessary background on the Falcon signature scheme. In Section 3, we detail our novel attack methodology. Section 4 presents the experimental validation of our attack under both `-O0` and `-O3` optimization levels. A discussion of the implications and countermeasures of our attacks is provided in Section 5. Finally, Section 6 concludes the paper.

2 Preliminary

Notations. We use bold uppercase letters for matrices and bold lowercase letters for vectors. A complex number is represented as $a + i \cdot b$, where a is the real part and $i \cdot b$ is the imaginary part. For a complex number z, $\bar{z}$ denotes its conjugate. For a real number a, $|a|$ denotes its absolute value. Define the polynomial ring $\mathcal{R} := \mathbb{Z}[x]/(x^n + 1)$. For a polynomial $P \in \mathcal{R}$, it can be expressed either as $P(x) = P_0 + P_1 x + \cdots + P_{n-1}x^{n-1}$, or equivalently, as a vector of coefficients $(P_0, P_1, \ldots, P_{n-1})$. The Hermitian adjoint of P is denoted as $P^\star$, which is defined as $P_0 - \sum_{i=1}^{n-1} P_i x^{n-i}$.

2.1 The Falcon Signature Scheme

In this subsection, we provide a brief introduction to the Falcon signature scheme. Some parts are omitted as they are not relevant to our work; for a comprehensive description, we refer the reader to the NIST submission of Falcon [PFH+22].

Let $q \in \mathbb{N}^*$ and let n be a power of two. These two values serve as public parameters in Falcon, where the modulus q is fixed to 12889, and the ring degree n is set to 512 or 1024. These two values of n correspond to two sets of parameters that address security levels I and V as defined by NIST, part of which are presented in Table. 2

Table 2: Some parameters for Falcon-512 and Falcon-1024.

	Falcon-512	**Falcon-1024**
Target NIST Level	I	V
Ring degree n	512	1024
Modulus q	12289	
Standard deviation σ	165.736 617 183	168.388 571 447
Max. Signature Square Norm $\lfloor \beta^2 \rfloor$	34 034 726	70 265 242

The overall process for generating the key pair of Falcon is shown in Alg. 1. During key generation, the coefficients of f and g ($f, g \in \mathcal{R}$) are sampled from a discrete Gaussian distribution centered at 0 with standard deviation $\sigma_{f,g} = 1.17\sqrt{q/2n}$ and restricted to small integer ranges, namely $[-31, 31]$ for $n = 512$ and $[-15, 15]$ for $n = 1024$. Subsequently, using the relation $fG - gF = q$, together with the algorithm proposed in [PP19b], the secret keys F and G can be computed efficiently. These operations are all implemented in the `NTRUGen` subroutine. Afterward, $\mathbf{B}$ is transformed into its FFT representation $\hat{\mathbf{B}}$,

Algorithm 1 Keygen(ϕ, q) [PFH+22]

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

$(f, g, F, G) \leftarrow \mathsf{NTRUGen}(\phi, q)$ ▷ Solving the NTRU equation
$\mathbf{B} \leftarrow \begin{bmatrix} g & -f \\ G & -F \end{bmatrix}$
$\hat{\mathbf{B}} \leftarrow \mathsf{FFT}(\mathbf{B})$ ▷ Compute the FFT for each of the 4 components $\{g, -f, G, -F\}$
$\mathbf{G} \leftarrow \hat{\mathbf{B}} \times \hat{\mathbf{B}}^{\star}$
$\mathbf{T} \leftarrow \mathsf{ffLDL}^{\star}(\mathbf{G})$ ▷ Computing the LDL* tree
for each leaf ℓ of $\mathbf{T}$ **do**
    $\ell.\text{value} \leftarrow \sigma/\sqrt{\ell.\text{value}}$ ▷ Normalization step
$\mathbf{sk} \leftarrow (\hat{\mathbf{B}}, \mathbf{T})$
$h \leftarrow gf^{-1} \bmod q$
$\mathbf{pk} \leftarrow h$
return $(\mathbf{sk}, \mathbf{pk})$

Algorithm 2 Sign($\mathbf{m}$, $\mathbf{sk}$, $\lfloor\beta^2\rfloor$) [PFH+22]

Input: A message $\mathbf{m}$, a secret key $\mathbf{sk}$, a bound $\lfloor\beta^2\rfloor$
Output: A signature of $\mathbf{m}$

$\mathbf{r} \leftarrow \{0,1\}^{320}$ uniformly
$c \leftarrow \mathsf{HashToPoint}(\mathbf{r} \| \mathbf{m}, q, n)$
$\mathbf{t} \leftarrow \left(-\frac{1}{q}\mathsf{FFT}(c) \odot \mathsf{FFT}(F), \frac{1}{q}\mathsf{FFT}(c) \odot \mathsf{FFT}(f)\right)$ ▷ $\odot$ denotes the multiplication in the FFT domain
do
    do
        $\mathbf{z} \leftarrow \mathsf{ffSampling}_n(\mathbf{t}, \mathbf{T})$
        $\mathbf{s} \leftarrow (\mathbf{t} - \mathbf{z})\hat{\mathbf{B}}$
    while $\|\mathbf{s}\|^2 > \lfloor\beta^2\rfloor$
    $(s_1, s_2) \leftarrow \mathsf{invFFT}(\mathbf{s})$
    $\mathbf{s} \leftarrow \mathsf{Compress}(s_2, 8 \cdot \mathsf{sbytelen} - 328)$
while $s = \perp$
return $(\mathbf{r}, \mathbf{s})$

based on which the Falcon Tree $\mathbf{T}$ is constructed using the LDL* decomposition. Finally, Alg. 1 outputs the secret key $\mathbf{sk}$, which consists of $\hat{\mathbf{B}}$ and $\mathbf{T}$, and the public key $\mathbf{pk}$.

The signing procedure of Falcon is illustrated in Alg. 2. It takes as input a message $\mathbf{m}$, the secret key parameter $\mathbf{sk}$, and a boundary parameter β. The algorithm begins by sampling a salt $\mathbf{r}$ uniformly at random and computing the hash c. After transforming c into the FFT domain, it is multiplied by the inverse of the secret key $\mathbf{B}$ in the FFT representation to obtain the preimage $\mathbf{t}$. Next, the subroutine `ffsampling` is invoked to execute fast Fourier sampling to produce the vector $\mathbf{z}$. Then, a short vector $\mathbf{s} = (s_1, s_2)$ is calculated such that $s_1 + s_2 h = c \pmod q$. Finally, if $\mathbf{s}$ is short enough, it is encoded and output together with $\mathbf{r}$ as the final signature; otherwise, it repeats the fast Fourier sampling to produce the short vector $\mathbf{s}$.

2.2 Two Signing Designs

The reference implementation of Falcon offers two signing designs, `sign_tree` and `sign_dyn`, which realize different ways of generating signatures. The `sign_tree` design strictly follows

Algs 2, where the input secret key **sk** consists of $\hat{\mathbf{B}}$ and $\mathbf{T}$. This form of secret key is referred to as an expanded key. In contrast, `sign_dyn` accepts the original key $\mathbf{B}$ and computes $\hat{\mathbf{B}}$ and $\mathbf{T}$ on demand during signing. Both designs invoke a function called `smallints_to_fpr` to convert integer coefficients into FP numbers, followed by an FFT to obtain $\hat{\mathbf{B}}$ and construct $\mathbf{T}$. In the `sign_tree` design, these computations are performed once and the expanded key is stored, whereas `sign_dyn`, which stores only the original key $\mathbf{B}$, repeats them for every signing operation.

These two designs reflect a trade-off between signing speed and memory consumption. In `sign_tree`, since $\hat{\mathbf{B}}$ and $\mathbf{T}$ are precomputed, the signing process is faster. The `sign_tree` design is meant to be used when several signatures are to be computed with the same private key. According to the comments in the reference code [1], the amortized cost per signature is roughly halved when using the `sign_tree` design for short messages, although it depends on the underlying architecture and implementation choices. However, the expanded key size is much larger than that of the original key. For instance, in Falcon-512 (resp. Falcon-1024), the original key size is 1281 bytes (resp. 2305 bytes), while the expanded key reaches $57,352$ bytes (resp. $122,888$ bytes). Such large keys impose significant memory pressure on resource-constrained platforms. Specifically, the Falcon-1024 implementation based on the `sign_tree` design does not fit into 128KiB of RAM, rendering this implementation infeasible on Cortex-M4 microcontrollers [KRSS19]. Consequently, mainstream Falcon implementation repositories (e.g., PQClean [2]) adopt `sign_dyn` for external interfaces.

2.3 Floating-Point Conversion and FFT in Falcon

In this subsection, we detail the FP conversion and the FFT in Falcon. As mentioned above, before applying the FFT to the secret key matrix $\mathbf{B}$, Falcon utilizes `smallints_to_fpr` to convert the coefficients of the key into the FP type, called `fpr`, which follows the IEEE-754 standard with 53 bits of precision. More precisely, Falcon encodes FP numbers using unsigned 64-bit integers. The 64 bits are divided into three fields: the most significant bit represents the sign s, the next 11 bits represent the exponent e, and the remaining 52 bits represent the mantissa m. These components (s, e, m) together define a FP number $x = (-1)^s \cdot 2^{e-1023} \cdot \left(1 + m \cdot 2^{-52}\right)$.

Falcon employs FFT to accelerate the polynomial multiplications. Although FFT is a well-known algorithm, certain implementation details in Falcon's reference code deserve special attention. Let ω denote a $2n$-th primitive root of unity, i.e., $\omega = \exp(i\pi/n)$. Define $\omega_j = \omega^{2j+1}$ for $j \in [0, n-1]$. These values are the roots of $x^n + 1$ over the complex field, and they satisfy the property $\omega_{n-1-j} = \overline{\omega_j}$. After applying FFT to f, the vector representation is given by $(f(\omega_0), f(\omega_1), \ldots, f(\omega_{n-1}))$. Since $f \in \mathcal{R}$ has real coefficients, it holds that $f(\omega_{n-1-j}) = \overline{f(\omega_j)}$. Therefore, the first point to note is that Falcon only needs to keep half of the complex values during FFT. Second, Falcon stores the real part of each complex value at index j of the original array holding the coefficients of f, while the corresponding imaginary part is stored at index $j + n/2$, thereby achieving an in-place representation. Finally, the FFT starts from the inputs whose form is $f_j + i \cdot f_{j+n/2}$, where $j \in [0, n/2 - 1]$.

2.4 Threat Model

In our threat model, we consider a passive adversary who is capable of measuring the power consumption of the target device while it executes cryptographic operations. The adversary is assumed to have knowledge of the software implementation and can approximately identify the locations of specific cryptographic computations. The attack follows a profiled

[1] https://falcon-sign.info/impl/falcon.h.html, see lines 174-186 in falcon.h
[2] https://github.com/PQClean/PQClean

attack setting, which requires building precise leakage models using template-based or machine-learning-based approaches.

During the profiling phase, the adversary is assumed to possess a clone of the target device, over which the adversary has full control of both the secret key and the input data. This enables the collection of corresponding power traces for the construction of accurate leakage models, which is performed in an offline manner. In the subsequent attack phase, however, the adversary can only acquire one or a few power traces from the actual target device, without access to the secret key. Based on the leakage models obtained in the profiling phase, the adversary aims to recover the secret key from the limited information available during the attack phase.

3 The Attack Methodology

According to the reference implementation of Falcon, the key expansion in the `sign_tree` design can be regarded as part of the key generation phase, whereas in the `sign_dyn` design it is performed during the signing phase. However, as mentioned earlier, both signing designs invoke the same routines for the FP conversions and FFT during key expansion. Therefore, the models developed for leakages in these operations can be leveraged to construct potential attacks targeting both designs. It should be noted, however, that only single-trace attacks can be designed for implementations adopting the `sign_tree` design of Falcon, while the `sign_dyn` design additionally allows for the possibility of multi-trace attacks. Consequently, in this section, we develop an attack methodology to recover the secret key with a single trace or a few traces. We begin with a brief introduction to the inspiration behind our methodology and then describe in detail how our attack is developed and carried out. The Falcon-512 is used as the representative case in this section.

3.1 Core Principle of the Attack

In Falcon, the secret keys and the public key satisfy $h = gf^{-1} \pmod q$. We can rewrite it in the following form:

$$\begin{bmatrix} g_0 \\ g_1 \\ \vdots \\ g_{n-1} \end{bmatrix} = \begin{bmatrix} h_0 & -h_{n-1} & \cdots & -h_1 \\ h_1 & h_0 & \cdots & -h_2 \\ \cdots & \cdots & \ddots & \vdots \\ h_{n-1} & h_{n-2} & \cdots & h_0 \end{bmatrix} \begin{bmatrix} f_0 \\ f_1 \\ \vdots \\ f_{n-1} \end{bmatrix} \pmod q. \tag{1}$$

We observe that once the total number of coefficients of f and g that can be determined reaches greater than or equal to n, the remaining key coefficients can be determined by solving the equation. In Falcon, f and g follow the same distribution and are processed identically. Consequently, if we can recover roughly half of the coefficients from each of f and g, then an equation where the dimension of the coefficient matrix is approximately $\frac{n}{2} \times \frac{n}{2}$ can be constructed and solved to recover the remaining key coefficients.

A natural question then arises: where can sufficient leakages be found to determine about half of these coefficients? Through careful analysis, we identify that some FP operations leak substantial information about the secret key. In particular, when the coefficients of f and g are converted into FP representations and subsequently multiplied by the twiddle factors during the FFT, their value ranges leak. In the following, we exploit the leakages that arise from these locations, along with our method, to determine about half of the coefficients of f or g (so that the total number of coefficients determined is at least n).

3.2 Exploiting Leakages from the Floating-Point Conversion

`smallints_to_fpr` invokes its subroutine `fpr_scaled` to convert every secret key coefficient into a FP number, which is illustrated in Listing 1[3]. This function can roughly be divided into four stages: extracting the absolute value and sign (lines 3–5), left-shifting the absolute value into a prescribed range (line 8), handling the case when the absolute value is zero (lines 10–12), and packing the result into `fpr` for return (line 13).

It is not hard to notice that the first stage and the third stage are vulnerable to side-channel analysis to extract the information of the sign and whether the coefficient equals zero, because there exist negation operations for 0 or 1. The more subtle leakage comes from `FPR_NORM64` in the second stage, which is shown in Listing 2. This function uses a binary search approach to left-shift the absolute value of the component `m` until its most significant bit becomes 1 [4]. There are 6 similar code snippets in Listing 2, and we omitted some of them here. These snippets check in turn whether the leading $32, 16, 8, 4, 2$ or 1 bit(s) of current `m` are all zeros. If so, it left-shifts `m` by the corresponding number of bits; otherwise, `m` remains unchanged. The condition used for each check is whether `nt` is 0 or 1, and the operations `(u64)nt-1` show obvious side-channel leakages to distinguish these two cases. After that, we know whether `m` is left-shifted by the corresponding number of bits and are able to partition the absolute value range of the secret key coefficients.

```
1  fpr fpr_scaled(s64 i, int sc) {
2      ...
3      s = (int)((u64)i >> 63);
4      i ^= -(s64)s;
5      i += s;
6      m = (u64)i;
7      e = 9 + sc;
8      FPR_NORM64(m, e);
9      ...
10     t = (u32)((u64)(i | -i) >> 63);
11     m &= -(u64)t;
12     e &= -(int)t;
13     return FPR(s, e, m);}
```

Listing 1: The fragment code of `fpr_scaled`.

```
1  #define FPR_NORM64(m, e)    do { \
2      u32 nt; \
3      (e) -= 63; \
4      nt = (u32)((m) >> 32); \
5      nt = (nt | -nt) >> 31; \
6      (m) ^= ((m) ^ ((m) << 32)) & ((u64)nt - 1); \
7      (e) += (int)(nt << 5);
8      \
9      nt = (uint32_t)((m) >> 48); \
10     nt = (nt | -nt) >> 31; \
11     (m) ^= ((m) ^ ((m) << 16)) & ((u64)nt - 1); \
12     (e) += (int)(nt << 4); \
13     ...
14     nt = (u32)((m) >> 63); \
15     (m) ^= ((m) ^ ((m) << 1)) & ((u64)nt - 1); \
16     (e) += (int)(nt); \
17 } while (0)
```

Listing 2: The fragment code of `FPR_NORM64`.

[3] We use `u64` to denote an unsigned 64-bit integer (C's `uint64_t`) and `s64` to denote a signed 64-bit integer (C's `int64_t`), etc, without further notice.

[4] `FPR_NORM64` assumes the input is nonzero. Notably, passing 0 to the function does not cause a runtime error; it simply renders the computation meaningless. The subsequent code behind it checks whether the input is zero, thereby correcting the behavior of the preceding code.

Combining the above three leakages in the FP conversion, we can deduce the sign of each coefficient of f (or g), and classify the absolute values of each coefficient into different sets. Recall that the absolute value range of coefficients is $[0, 31]$, each coefficient belongs to one of the following sets:

$$\{0\},\ \{1\},\ \{2,3\},\ \{4,..,7\},\ \{8,..,15\},\ \{16,...,31\}. \tag{2}$$

This means we can determine coefficients with values of 0 and ± 1. According to the distribution of the coefficients of f (or g), the expected number of coefficients that are 0 and ± 1 is about 148.

3.3 Exploiting Leakages from the Floating-Point Multiplication

Building on the partition of the coefficient range presented in (2), we find that leakages from the FP multiplication in the FFT enable a finer-grained partition of the secret key coefficients that are multiplied with the twiddle factor. We focus only on the FP multiplications at the start of the FFT because this is when the secret key coefficients are multiplied directly by the twiddle factor $(\frac{\sqrt{2}}{2} + i \cdot \frac{\sqrt{2}}{2})$. Falcon employs `fpr_mul` to do the FP multiplication, as shown in Listing 3.

```
1  fpr_mul(fpr x, fpr y) {
2      ...
3      xu = (x & (((u64)1 << 52) - 1)) | ((u64)1 << 52);
4      yu = (y & (((u64)1 << 52) - 1)) | ((u64)1 << 52);
5      x0 = (u32)xu & 0x01FFFFFF;
6      x1 = (u32)(xu >> 25);
7      y0 = (u32)yu & 0x01FFFFFF;
8      y1 = (u32)(yu >> 25);
9      w = (u64)x0 * (u64)y0;
10     ...
11     w = (u64)x1 * (u64)y0;
12     ...
13     zu = (u64)x1 * (u64)y1;
14     ...
15     zu += z2;
16     ...
17     zv = (zu >> 1) | (zu & 1);
18     w = zu >> 55;
19     zu ^= (zu ^ zv) & -w;
20     ...
21     return FPR(s, e, zu);
22 }
```

Listing 3: The fragment code of `fpr_mul`.

In Listing 3, two main tasks are performed. First, it multiplies the mantissas of the two FP numbers (lines 9–15); second, it normalizes the product of the mantissas to the range $[2^{54}, 2^{55} - 1]$ (lines 17–19). Since the twiddle factor is fixed, the mantissa multiplication naturally serves as a target for the profilied attack, where the relationship between the power traces and the mantissa values is modeled.

We further observe, however, that the normalization procedure introduces additional leakage. Specifically, during normalization, the implementation checks whether the product is greater than or equal to 2^{55}. Since `x1` and `y1` effectively represent 28-bit values whose the most significant bit is 1, the value range of `zu` is $[2^{54}, 2^{56} - 1]$. As a result, right-shifting `zu` by 55 bits (see line 18) yields `w` = 1 if `zu` is greater than or equal to 2^{55}; otherwise, `w` = 0. Side-channel analysis can be conducted to learn the leakage information about the comparison result. Since the absolute value range of the coefficient is $[0, 31]$ when starting FFT, we can enumerate all possible values of the coefficient to calculate whether the results

exceed 2^{55}. Taking f as an example, this will divide the absolute values (excluding the determined absolute value of 0 and 1) into 2 different sets:

$$\begin{aligned}\{|f_j| \mid \texttt{zu} < 2^{55}\} &= \{2,4,5,8,9,10,11,16,17,18,19,20,21,22\},\\ \{|f_j| \mid \texttt{zu} \geq 2^{55}\} &= \{3,6,7,12,13,14,15,23,24,25,26,27,28,29,30,31\}.\end{aligned} \tag{3}$$

As a result, it can be determined which set $|f_j|$ ($j \in [128,255] \cup [384,511]$) belongs to based on the leakage from the normalization procedure. Combining the partition outlined in (2), coefficients in the set f_j ($j \in [128,255] \cup [384,511]$) with values of ± 2 and ± 3 can be determined. Moreover, the absolute value of the remaining uncertain coefficients belongs to one of the following finer sets:

$$\{4,5\},\ \{6,7\},\ \{8,\ldots,11\},\ \{12,\ldots,15\},\ \{16,\ldots,22\},\ \{23,\ldots,31\}.$$

At this point, the expected number of coefficients that can be determined with values of $0, \pm 1, \pm 2, \pm 3$ is approximately 207.

This finer value range facilitates side-channel analysis on the multiplication of mantissas to partition the value range further. However, it is not necessary to recover all of these remaining uncertain coefficients. Instead, it suffices to construct two binary classifiers based on the collected side-channel information: one to distinguish between 4 and 5, and the other to distinguish between 6 and 7. Once the coefficients in the set f_j ($j \in [128,255] \cup [384,511]$) that take the values ± 4 and ± 5 are also determined, the total expected number of determined coefficients is approximately 246. By additionally considering coefficients with values ± 6 and ± 7, this expectation increases to about 266, exceeding half the number of coefficients of the secret key f.

3.4 Launching the Attack

The overall attack consists of two phases: a profiling phase and an attack phase. For the profiling phase, power traces from three target operations (FP conversion, normalization procedure, and mantissa multiplication) are collected to construct corresponding classifiers. For the attack phase, the constructed classifiers are utilized sequentially to determine enough secret key coefficients to recover the key by solving the equation (1). To increase the success rate of key recovery, several strategies are also proposed. The workflow of the attack is shown in Fig. 1.

3.4.1 Profiling Phase

Based on the collected power traces for the operations of FP conversion, normalization procedure and mantissa multiplication, we build three types of classifiers required by the attack phase. Specifically, We first build an 11-class classifier for the FP conversion. This classifier aims to predict both the sign of a coefficient and the set of its absolute value, as shown in (2). Then, a binary classifier for the normalization procedure is constructed to distinguish which set shown in (3) the key coefficients belong to. In addition, two binary classifiers for distinguishing elements in $\{4,5\}$ and $\{6,7\}$ during mantissa multiplication are constructed. These classifiers are implemented using either template-based or machine-learning-based approaches, depending on the strength of the observed leakages.

Template-based Profiling For leakage distributions exhibiting clear inter-class separability, we adopt a classical template-based profiling methodology [CRR03]. To identify informative samples, we first perform pairwise Welch's t-tests between all label groups and compute the Sum of Squared t-values (SOST) metric,

$$\mathrm{SOST} = \sum_{i,j=1,i\neq j}^{\text{groups}} \left(t_{i,j}^{\mathsf{Welch}}\right)^2,\ t_{i,j}^{\mathsf{Welch}} = \frac{m_i - m_j}{\sqrt{\sigma_i^2/n_i + \sigma_j^2/n_j}}, \tag{4}$$

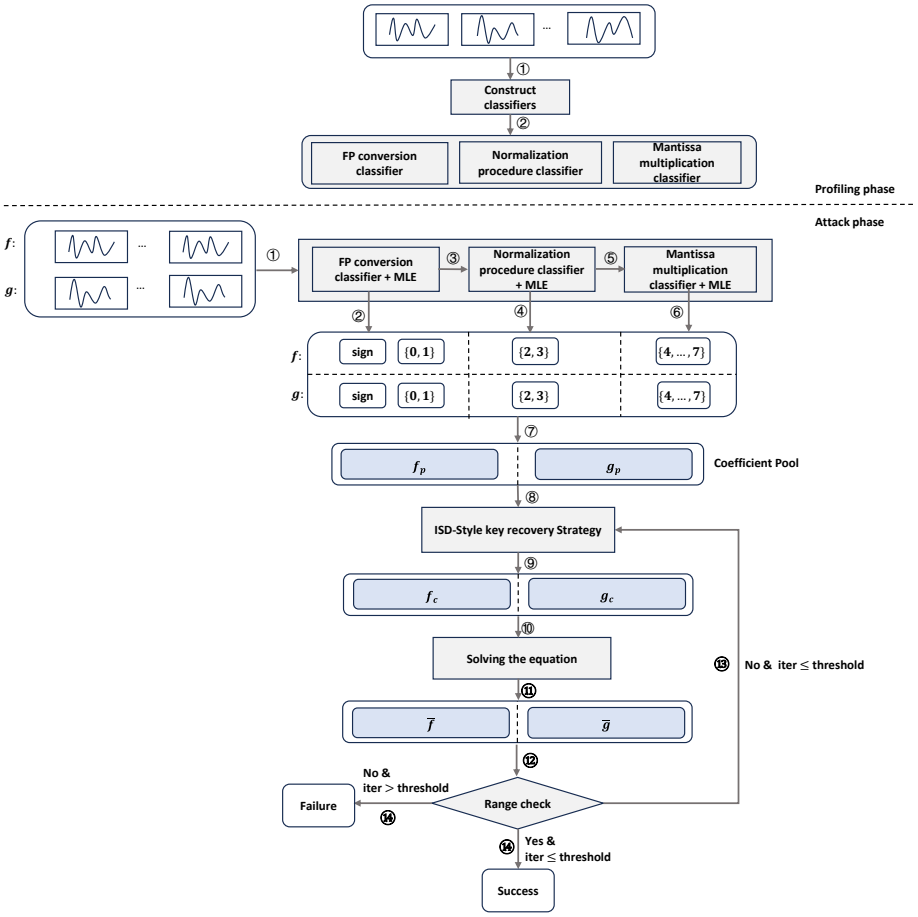

Figure 1: Workflow of the proposed attack. f_p and g_p denote the predicted secret key coefficients obtained from three types of classifiers. f_c and g_c represent the 512 coefficients selected according to the ISD-style key recovery strategy. $\bar{f}$ and $\bar{g}$ denote the recovered secret keys that require further value range check. When multiple power traces are replaced by a single trace and the maximum likelihood estimation (MLE) strategy is removed, the workflow simplifies to that of a single-trace attack.

where m_i, n_i and σ_i denote the mean, number and standard deviation of the i-th group of pwoer traces. This metric highlights time points with the strongest multi-class leakage separation. The points of interest (PoIs) are then selected by ranking samples according to their SOST values. Using these PoIs, we construct the Gaussian templates—means and covariance matrices per class — following the widely used approach in side-channel analysis. During the attack phase, the classifiers evaluate the likelihood or the possibility that traces collected in the attack phase belong to each class. This method is particularly effective when leakage closely follows Gaussian models and PoIs are well separated.

Machine-Learning-based Profiling When leakage distributions exhibit weak separability, selecting PoIs based solely on SOST becomes a labor-intensive and unreliable procedure, as meaningful features are obscured by noise and micro-architectural variations. To avoid manual PoIs selection, we adopt convolutional neural networks (CNN), which directly learn discriminative features from raw traces [CDP17, PSB+18]. Prior work shows that CNN and compact Multilayer Perceptron (MLP) achieve robust profiling performance under noisy or misaligned leakage [MPP16]. In our experiments, these models consistently outperform template-based profiling, precisely because they bypass PoI extraction and instead rely on end-to-end feature learning tailored to subtle signal variations.

3.4.2 Attack Phase

For the `sign_dyn` design, the attack phase can be carried out in two different settings, depending on the accuracy of the classifiers: a single-trace attack when accuracy is sufficiently high, and a multi-trace attack when additional traces are needed to compensate for lower accuracy. However, for the `sign_tree` design, only a single-trace attack can be conducted. This is because the key expansion from the original key is just performed once, where only one power trace can be captured during the attack phase.

Single-Trace Attack Once the constructed classifiers achieve consistently high accuracy, we launch a single-trace attack. Specifically, we collect one power trace covering the FP conversions and the FP multiplications performed at the beginning of the FFT for both f and g, and then segment it accordingly. Following the methods described above, we apply the corresponding classifiers sequentially to progressively narrow the coefficient ranges and recover certain coefficients. When the coefficients of f and g that have been determined reach at least 512 in total, we substitute them into the equation (1) to recover the remaining coefficients by solving the equation.

Multi-Trace Attack When the classifiers do not achieve near-perfect accuracy, we resort to a multi-trace attack. In this setting, multiple power traces corresponding to the three target operations are collected, and a maximum likelihood estimation (MLE) strategy is applied to ensure that the predicted coefficients are sufficiently accurate. More specifically, when refining the range of a key coefficient using a classifier, we obtain multiple classification probability distributions for candidate sets from multiple power traces. To reduce the probability of misclassification, we apply the MLE strategy to determine the most likely class of the key coefficient. The same procedure is repeated when applying the next classifier. Once enough coefficients are identified, they are substituted into the equation (1) to recover the remaining coefficients.

ISD-Style Key Recovery Strategy To further increase the success rate of key recovery, we employ an ISD-style (Information Set Decoding [Pra62]) approach. In single-trace and multi-trace settings, we are able to determine slightly more than 512 coefficients and form a coefficient pool. From this pool, we randomly sample subsets of 512 coefficients and

test them as candidate reconstructions. This process is repeated until a consistent and error-free set is obtained (verified by solving the corresponding equation and checking the value range of every recovered coefficient), or until the number of iterations exceeds a preset threshold (indicating a failure to recover the secret key). This approach tolerates a small fraction of prediction errors and significantly enhances the success rate of key recovery.

4 Experiment Validation

In this section, we validate our single-trace attack on real-world devices compiled with `-O0`, and then extend the experiment to `-O3`. While some leakages are mitigated under `-O3`, the attack remains effective with multiple traces under the `sign_dyn` design, confirming its practicality.

4.1 Experimental Setup

Our experimental setup consists of the ChipWhisperer Husky together with two base boards, CW308 and CW313. Each base board is equipped with an STM32F405 microcontroller target featuring an ARM Cortex-M4 core. We target the reference code from PQClean [5]. The operating frequency is set to 30MHz, with a sample rate of 120MHz. The compiler version is `arm-none-eabi-gcc 14.2.0`. Because the targeted code segments and leakage source are the same for `sign_tree` and `sign_dyn` designs, it is enough to run the functions `smallints_to_fpr` and the subsequent `PQCLEAN_FALCON512_CLEAN_FFT` (where line 218 of the file `fft.c` is modified from `u < logn` to `u < 2` because we only focus on the first layer of FFT) to conduct our experiment for both designs. Trigger functions are incorporated in the code to capture the power trace. After collection, power traces were processed using a 4th-order low-pass filter with a cutoff frequency of 15MHz.

4.2 The Single-Trace Attack under `-O0`

At `-O0`, the leakages at the side-channel-susceptible locations identified in Section 3.2 and 3.3 are obvious. Therefore, we perform a single-trace attack using the template-based approach [CRR02]. The power traces are collected on the CW308 base board.

4.2.1 The Profiling Phase

We first construct an 11-class template targeting the leakages arising from FP conversions. In `fpr_scaled`, the leakages observed in both the first and third stages (lines 3–5 and 10–12 in Listing 1) originate from the well-studied negation operation. Since the variables `s` and `t` are either 0 or 1, the resulting data exhibits a Hamming weight difference of 64 bits, which translates into a clearly distinguishable power variation at the corresponding locations. In `FPR_NORM64`, six similar code snippets determine whether the input must be left-shifted by the respective number of bits, depending on whether `nt`=0 or `nt`=1. The outcome of the subsequent subtraction `(u64)nt -1` (e.g., line 6 in Listing 2) is either an all-zero or an all-one word, corresponding to a Hamming weight of 0 or 64. This also manifests as a highly discernible feature in the power trace.

For each class, we collect 200 power traces containing the leakages from the three stages to construct the template by controlling the coefficients of f and g. The PoIs are selected using the SOST metric, which is shown in Figure 2. In Figure 2, five distinct clusters of peaks can be observed (shown in the red boxes). From left to right, these correspond to one peak related to the sign, three peaks caused by left-shifts in `FPR_NORM64`, and the final

[5] https://github.com/PQClean/PQClean/tree/master/crypto_sign/falcon-512/clean

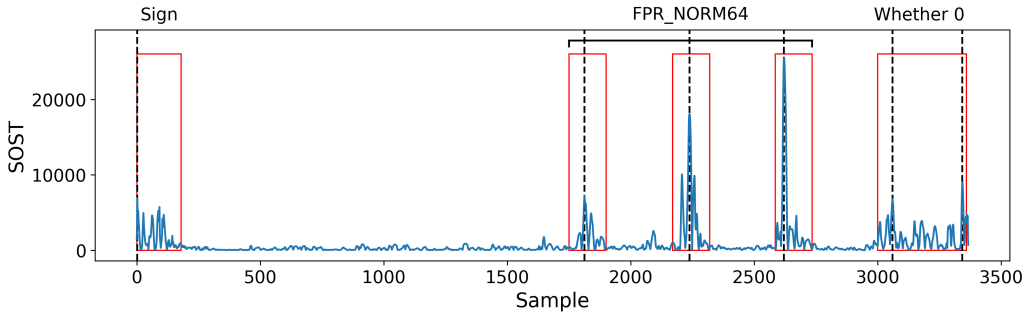

Figure 2: The result of the t-test conducted on the collected power traces that contain the leakages in `fpr_scaled`.

cluster of peaks indicating whether the input equals zero. Based on these observations, we select 6 PoIs indicated by the vertical dashed lines in the figure. We then collect an additional 1000 traces per class for testing and the 11-class template achieves an accuracy of 100%.

We then construct a 2-class template targeting the leakage of the normalization process. This leakage also comes from the negation operation (line 19 in Listing 3) because the variable `w` can only take values of 0 or 1. As before, we collect 200 power traces for each class to construct the template. Figure 3 illustrates the difference between two randomly selected power traces from different classes, which clearly demonstrates their distinct characteristics. Based on the t-test results of the two classes, we identify the position with the maximum SOST as the point of interest (PoI) and construct a univariate template. For accuracy evaluation, an additional 1,000 power traces per class are collected as the test set, and the univariate template achieves an accuracy of 100%.

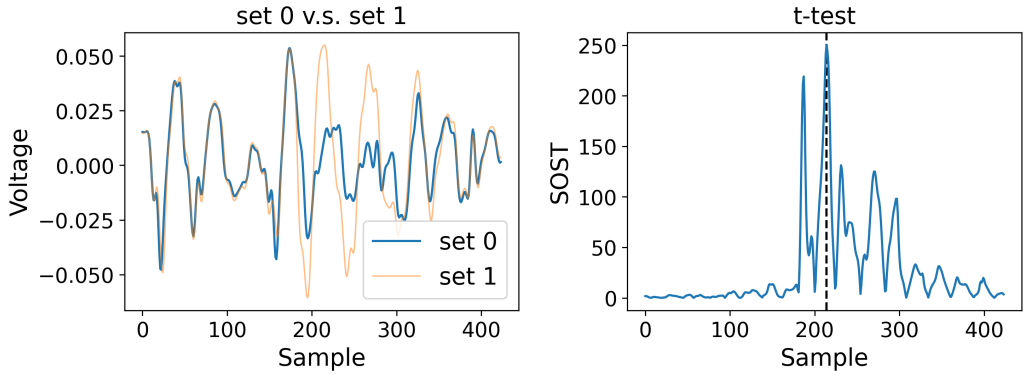

Figure 3: The left figure shows the result of randomly selecting one trace from each of the two sets for comparison, where `set0` corresponds to the class of mantissa multiplication results less than 2^{55}, and `set1` corresponds to that greater than or equal to 2^{55}. The right figure presents the result of the t-test conducted on the two classes of traces.

Finally, according to our discussion in Section 3.3, it would be sufficient to construct only two templates for the mantissa multiplication, namely one to distinguish between 4 and 5 and another to distinguish between 6 and 7. For each class within every template, we collect 200 power traces. Across both templates, we fix the spacing between PoIs at 80 samples and select 10 positions with the largest SOST values as PoIs. We collect 1000 traces per class for each template to form the test sets and both templates achieve an

Table 3: CNN architecture. The shape convention follows (#`trace`, #`sample`, #`channel`), abbreviated as (B, L, C). Convention $\texttt{op}_{k,s,p}$ is used to represent a operator, either `Conv` or `AvgPool`, with kernel, stride, padding, resp. All convolution layers use $k = 5$, padding $\lfloor k/2 \rfloor$. All fully-connected layers use 0.1 dropout. All optimizers use `Adam`.

Layer	**Output shape**	**Details**
Input	$(B, \ell, 1)$	1D sequence, batch size B, length (#`sample`) ℓ
Conv 1	$(B, \lfloor \ell/2 \rfloor, 4)$	$\texttt{Conv}_{5,1,2} \to \texttt{BN} \to \texttt{Sigmoid} \to \texttt{AvgPool}_{2,2,0}$
Conv 2	$(B, \lfloor \ell/4 \rfloor, 4)$	$\texttt{Conv}_{5,1,2} \to \texttt{BN} \to \texttt{Sigmoid} \to \texttt{AvgPool}_{2,2,0}$
Flatten	$(B, \lfloor \ell/4 \rfloor \times 4)$	
FC 1	$(B, 128)$	$\texttt{FC} \to \texttt{ReLU} \to \texttt{Dropout}_{0.1}$
FC 2	$(B, 128)$	$\texttt{FC} \to \texttt{ReLU} \to \texttt{Dropout}_{0.1}$
Output	(B, N_{class})	`FC`

accuracy of 100% on their datasets.

4.2.2 The Attack Results

After preparing the required templates, we randomly generate 100 Falcon-512 keys. In the attack phase, we collect a single power trace encompassing the FP conversion and the FP multiplication at the beginning of the FFT of both f and g, and segment it accordingly. Because classifiers from the profiling phase all achieve perfect accuracy, we conduct a single-trace attack for both signing designs. The experimental results show that the single-trace attack achieves a 100% success rate, meaning that this single-trace attack applies to both signing designs. Meanwhile, we carried out our attack on Falcon-1024. Our attack also achieves a success rate of 100% for 100 randomly generated Falcon-1024 keys. Since the templates constructed under `-O0` exhibit perfect classification accuracy, the ISD-style key recovery method is not necessary here.

4.3 The Multi-Trace Attack under `-O3`

The experiments under the `-O0` optimization level provide meaningful validation for our analysis of the leakage source and demonstrate the feasibility of the proposed attack. Nevertheless, since `-O0` corresponds to an unoptimized compilation, we further conduct experiments under the `-O3` optimization level to evaluate the practicality of the attack in a more realistic setting.

At `-O3`, the obvious leakages under `-O0` are mitigated by instruction optimization and reordering. For instance, during the normalization procedure (line 19 in Listing 3), `zu` is derived via flags and conditional moves rather than by explicitly calculating `-w`. As a result, the operation `-w` no longer induces a 64-bit Hamming weight difference. Nevertheless, t-test to the selected segments show that it has ≥ 4.5 decision threshold, meaning that there are still strong leakages.

4.3.1 Profiling Phase

The leakages are less distinguishable at `-O3`, and thus selecting PoIs based solely on SOST becomes a labor-intensive and relatively unreliable procedure. We use a CNN model to construct a discriminative classifier, which can learn from the entire power trace segment and perform classification based on the extracted features. The CNN model's architecture is summarized in Table 3.

For training the 11-class classifier corresponding to the FP conversion, the 2-class classifier corresponding to the normalization procedure, and the two 2-class classifiers

corresponding to the mantissa multiplication, we sample 5632, 10752, 1024, and 1024 power traces as the training sets, respectively. In addition, we collect double number of power traces for each classifier to form the test sets. The classification accuracies of the four classifiers on their test sets are summarized in Table 4.

Table 4: The Accuracy of 4 CNN Classifiers.

Classifiers	**Target Operation**	**Accuracy**
11-class	FP Conversion	99.96%
2-class	Normalization Procedure	89.92%
2-class	Mantissa Multiplication (4 and 5)	98.44%
2-class	Mantissa Multiplication (6 and 7)	90.92%

4.3.2 The Attack Results

Since some classifiers fail to achieve near-perfect accuracy, the single-trace attack failed. Consequently, our attack under `O3` targets the `sign_dyn` design because we can turn to a multi-trace attack in this setting.

As multiple traces are required in the attack phase, the MLE strategy can be used to improve the prediction accuracy of the secret key coefficients by every classifier. It is worth noting that, since the real and imaginary parts of the twiddle factor $(\frac{\sqrt{2}}{2} + i \cdot \frac{\sqrt{2}}{2})$ are equal at the beginning of the FFT, each power trace provides two independent observations for both the normalization procedure and the mantissa multiplication (this information is not used for the key recovery at `-O0`). This further enhances the prediction accuracy when applying the MLE strategy. In addition, we adopt the ISD-style key recovery strategy to improve the success rate of key recovery. Specifically, since the 11-class classifier achieves near-perfect accuracy, we treat the coefficients determined by it as error-free. Let N denote the number of coefficients taking values of 0 and ± 1. We then repeatedly sample subsets of size $512 - N$ from the coefficients predicted as $\pm 2, \pm 3, \pm 4, \pm 5, \pm 6$, or ± 7, until the sampled subset contains only correct elements. The threshold for the number of trials is set to 1,000,000.

With this multi-trace attack, we recover 80 out of 100 randomly generated Falcon-512 keys using only 5 traces. Furthermore, we also conduct attacks on Falcon-1024, where 10 traces allow us to recover 85 out of 100 randomly generated keys. More detailed results for Falcon-512 and Falcon-1024 are provided in Table 5.

Table 5: The number of the successfully recovered keys out of 100 randomly generated keys with `-O3` in the `sign_dyn` design.

#Trace	**#Success of Falcon-512**	**#Success of Falcon-1024**
15	99	95
10	98	85
5	80	31
1	0	0

4.3.3 The Attack Complexity

When only a limited number of power traces are collected during the attack phase, the prediction accuracy of the classifiers decreases accordingly. In this case, the most time-consuming part of our attack arises from the ISD-style key recovery strategy, which involves solving a large number of systems of linear equations. In the worst case, recovering a Falcon-512 (Falcon-1024) secret key requires solving 1,000,000 systems of equations, leading to a computational complexity of about 2^{44} (2^{47}) basic operations. However, even under

this worst-case scenario, such a level of complexity remains well within the capabilities of modern computers.

We implemented a Gaussian elimination algorithm in C to solve these systems of equations efficiently. On a personal computer equipped with a 13th Gen Intel(R) Core(TM) i7-1365U CPU and 16 GB of RAM, constructing and solving one system of dimension approximately 256×256 (for Falcon-512) and 512×512 (for Falcon-1024) takes about 0.0169 s and 0.135 s on a single thread, respectively. These timings correspond to a workload of approximately 4.7 (37.5) core-hours to complete the equation-solving phase of a Falcon-512 (Falcon-1024) key-recovery attack in the worst-case scenario. Furthermore, this computation can be efficiently parallelized by distributing the 1,000,000 equation-solving tasks in a key-recovery attack across multiple threads or processing units. For example, when executed on a typical 64-core server with 128 threads (e.g., AMD EPYC™ 9554), the equation-solving phase of a Falcon-512 (Falcon-1024) key-recovery attack would require just about 2.2 min (17.6 min) based on the measured per-system solving time. Note that the average-case complexity is significantly lower than the worst case. For example, when attacking Falcon-512 with 5 power traces, the attempts in the ISD-style key recovery strategy in 80 successful attacks ranged from 2 to 207,134, with 75 of them requiring fewer than 10,000 attempts—far below 1,000,000. These results confirm that the proposed attack constitutes a fully practical end-to-end attack.

5 Discussion

This section analyzes the impacts, limitations and future directions of our proposed attack against Falcon implementations. A discussion of countermeasures is also presented.

5.1 On the Proposed Attack

Based on the newly found leakages in the FP conversion and the FP multiplication at the beginning of the FFT routine associated with the secret key, our new attack demonstrates a very effective attack on Falcon implementations, particularly on the practicality used `sign_dyn` design. Specifically, under the `-O0` compilation level, our proposed attack can recover the secret key from a single trace for both `sign_dyn` and `sign_tree` designs. This is a new type of single-track attack, completely different from the single-track attack proposed by Qiu et al [QA25]. Under the `-O3` compilation level, the key can still be recovered with high probability with our attack using only a few power traces in the `sign_dyn` design. Compared with the previous attacks in the signing procedure, our attack demonstrates substantial improvements in the number of required traces (see Tab. 1). This significant improvement stems from the effective exploitation of the key distribution, public-key information, and strong leakages revealed during the FP conversion and FFT routines.

Considering the memory constraints of embedded platforms and the portability of the code, mainstream repositories that contain Falcon implementations (such as PQClean, pqm4 [6], liboqs [7], etc.) employ the `sign_dyn` design as the standard signing interface. As a result, all these implementations are highly threatened by our attack, which reveals that this interface design choice introduces a serious vulnerability in terms of side-channel security. The presence of our attack makes the trade-off between the practicality and security of the signing interface an issue that cannot be overlooked. Therefore, exploring designs that preserve the practicality of Falcon while minimizing the leakages represents an important direction for future research.

[6] https://github.com/mupq/pqm4
[7] https://github.com/open-quantum-safe/liboqs

Despite our attack opening a new avenue to attack the Falcon implementation, limitations remain. Our attack fails against Falcon implementations that adopt the `sign_tree` design compiled under the `-O3` optimization level. However, the `sign_tree` design is impractical because its substantial storage overhead, in particular on the memory-constrained embedded platforms. Although the `sign_tree` design has not been adopted by the mainstream Falcon implementation repositories, achieving a single-trace attack on the Falcon signing procedure under the `-O3` setting would itself be highly appealing and meaningful. A promising direction for future research is to design a single-trace attack under the `-O3` setting with the help of the Soft-Analytical Side-Channel Attack (SASCA) technique [VCGS14] based on this work.

5.2 Countermeasures

We discuss possible countermeasures for Falcon implementations adopting the `sign_dyn` design. Our attack relies on the leakages originating from FP conversions and FP multiplications in the FFT, as well as on the distribution of the secret key coefficients. Therefore, mitigating or preventing the attack can be achieved by reducing the exploitable leakages or altering the key distribution.

Since the FP conversion and FFT are inherent to Falcon, a certain level of leakage is unavoidable. Introducing the hiding countermeasure can reduce the amount of exploitable leakage and thus make our attack more difficult. However, the overall security and performance impact of applying hiding to Falcon still requires further investigation.

The masking countermeasure with provable security can effectively protect against our attack. Specifically, masking divides each secret coefficient into multiple random shares, thereby ruining the original distribution of the key coefficients. However, current research on masked implementations of Falcon remains limited. For example, both the masked software implementation in [CC24] and the masked hardware implementation in [KA24] mainly consider attacks targeting FP multiplications. These existing works need to be further extended to resist our attack. It is also worth noting that masking introduces significant computational overhead, making it challenging to balance efficiency and security. A practically deployable masked Falcon will likely require substantial future research efforts.

6 Conclusion Remarks

We have presented a new single-trace side-channel attack on the Falcon; in particular, our attack in the `sign_dyn` design is the first single-trace attack on the Falcon signing procedure. By exploiting novel leakages in the FP conversion and FP multiplication at the beginning of the FFT, we determine enough secret key coefficients to solve for the entire key via a system of linear equations. Our attack achieves 100% key recovery on an ARM Cortex-M4 with one power trace for Falcon-512/1024 compiled at `-O0` under both `sign_tree` and `sign_dyn` designs. With the multi-trace version of the attack, even with `-O3`, we recover 80% of Falcon-512 keys using just five traces under the practicality used `sign_dyn` design. Compared with the previous attacks, our attack significantly reduces the number of required traces to recover the secret key while keeping a very high success rate.

These findings highlight the concerns about the signing design choice and the urgent need for implementing countermeasures in Falcon, particularly for embedded platforms. Our attack reveals that the `sign_dyn` design, which is adopted by the signing interface in mainstream Falcon repositories, suffers from severe leakage. Since the standardized version of Falcon has not yet been finalized, it is crucial that future standardization processes thoroughly evaluate and justify both the security and practicality of the signing design

to be adopted. Masking is an effective countermeasure against our attack; however, the current development of masking techniques for Falcon remains limited. To fully mitigate the threat posed by our attack, future work should consider implementing a comprehensive masking scheme that specifically addresses leakages from FP operations. Moreover, given that Falcon is intended for deployment on embedded platforms, the significant performance overhead introduced by masking calls for thorough investigation.

Acknowledgment

Kang Li thanks the China Scholarship Council (CSC) support under Grant No. 202406780039. This work was also supported in part by the Swedish Research Council (grant numbers 2021-04602 and 2023-03654), the Crafoord Foundation, and the Wallenberg AI, Autonomous Systems and Software Program (WASP), funded by the Knut and Alice Wallenberg Foundation. The computations were enabled by the supercomputing resource Berzelius provided by National Supercomputer Centre at Linköping University and the Knut and Alice Wallenberg foundation.

References

[ACLZ20] Dorian Amiet, Andreas Curiger, Lukas Leuenberger, and Paul Zbinden. Defeating newhope with a single trace. In *International Conference on Post-Quantum Cryptography*, pages 189–205. Springer, 2020.

[CC24] Keng-Yu Chen and Jiun-Peng Chen. Masking floating-point number multiplication and addition of falcon: First-and higher-order implementations and evaluations. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2024(2):276–303, 2024.

[CDP17] Eleonora Cagli, Cécile Dumas, and Emmanuel Prouff. Convolutional neural networks with data augmentation against jitter-based countermeasures. In *Cryptographic Hardware and Embedded Systems – CHES 2017*, volume 10529 of *Lecture Notes in Computer Science*, pages 45–68. Springer, 2017.

[CRR02] Suresh Chari, Josyula R Rao, and Pankaj Rohatgi. Template attacks. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 13–28. Springer, 2002.

[CRR03] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template attacks. In Burton S. Kaliski, çetin K. Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002*, pages 13–28, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[CSM+25] Shuyi Chen, Honglin Shao, Jingdian Ming, Yuejun Liu, Yiwen Gao, and Yongbin Zhou. Serp-sca: A strengthened side-channel attack framework for falcon signature. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2025.

[DN12] Léo Ducas and Phong Q Nguyen. Learning a zonotope and more: Cryptanalysis of ntrusign countermeasures. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 433–450. Springer, 2012.

[DP16] Léo Ducas and Thomas Prest. Fast fourier orthogonalization. In *Proceedings of the 2016 ACM International Symposium on Symbolic and Algebraic Computation*, pages 191–198, 2016.

[EFGT17] Thomas Espitau, Pierre-Alain Fouque, Benoît Gérard, and Mehdi Tibouchi. Side-channel attacks on bliss lattice-based signatures: Exploiting branch tracing against strongswan and electromagnetic emanations in microcontrollers. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1857–1874, 2017.

[FKT+20] Pierre-Alain Fouque, Paul Kirchner, Mehdi Tibouchi, Alexandre Wallet, and Yang Yu. Key recovery from gram–schmidt norm leakage in hash-and-sign signatures over ntru lattices. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 34–63. Springer, 2020.

[GMRR22] Morgane Guerreau, Ange Martinelli, Thomas Ricosset, and Mélissa Rossi. The hidden parallelepiped is back again: Power analysis attacks on falcon. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 141–164, 2022.

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *Proceedings of the Fortieth Annual ACM symposium on Theory of Computing*, pages 197–206, 2008.

[HBD+22] A. Hülsing, D. J. Bernstein, C. Dobraunig, M. Eichlseder, S. Fluhrer, S.-L. Gazdag, P. Kampanakis, S. Kölbl, T. Lange, M. M. Lauridsen, F. Mendel, R. Niederhagen, C. Rechberger, J. Rijneveld, P. Schwabe, J.-P. Aumasson, B. Westerbaan, and W. Beullens. Sphincs+. Technical report, National Institute of Standards and Technology, 2022. Available at https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022.

[HPRR20] James Howe, Thomas Prest, Thomas Ricosset, and Mélissa Rossi. Isochronous gaussian sampling: from inception to implementation: With applications to the falcon signature scheme. In *International Conference on Post-Quantum Cryptography*, pages 53–71. Springer, 2020.

[KA21] Emre Karabulut and Aydin Aysu. Falcon down: Breaking falcon post-quantum signature scheme through side-channel attacks. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pages 691–696. IEEE, 2021.

[KA24] Emre Karabulut and Aydin Aysu. Masking falcon's floating-point multiplication in hardware. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2024(4):483–508, 2024.

[KAA21] Emre Karabulut, Erdem Alkim, and Aydin Aysu. Single-trace side-channel attacks on ω-small polynomial sampling: With applications to ntru, ntru prime, and crystals-dilithium. In *2021 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 35–45. IEEE, 2021.

[KH18] Suhri Kim and Seokhie Hong. Single trace analysis on constant time cdt sampler and its countermeasure. *Applied Sciences*, 8(10):1809, 2018.

[KRSS19] Matthias J Kannwischer, Joost Rijneveld, Peter Schwabe, and Ko Stoffelen. pqm4: Testing and benchmarking nist pqc on arm cortex-m4. 2019.

[LDK+22] V. Lyubashevsky, L. Ducas, E. Kiltz, T. Lepoint, P. Schwabe, G. Seiler, D. Stehlé, and S. Bai. Crystals-dilithium. Technical report, National Institute of Standards and Technology, 2022. Available at https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022.

[LZY+25] Xiuhan Lin, Shiduo Zhang, Yang Yu, Weijia Wang, Qidi You, Ximing Xu, and Xiaoyun Wang. Thorough power analysis on falcon gaussian samplers and practical countermeasure. In *IACR International Conference on Public-Key Cryptography*, pages 229–258. Springer, 2025.

[MPP16] Houssem Maghrebi, Thibault Portigliatti, and Emmanuel Prouff. Breaking cryptographic implementations using deep learning techniques. Cryptology ePrint Archive, Paper 2016/921, 2016.

[NR06] Phong Q Nguyen and Oded Regev. Learning a parallelepiped: Cryptanalysis of ggh and ntru signatures. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 271–288. Springer, 2006.

[PFH+22] T. Prest, P.-A. Fouque, J. Hoffstein, P. Kirchner, V. Lyubashevsky, T. Pornin, T. Ricosset, G. Seiler, W. Whyte, and Z. Zhang. Falcon. Technical report, National Institute of Standards and Technology, 2022. Available at https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022.

[Por19] Thomas Pornin. New efficient, constant-time implementations of falcon. *Cryptology ePrint Archive*, 2019.

[PP19a] Peter Pessl and Robert Primas. More practical single-trace attacks on the number theoretic transform. In *International Conference on Cryptology and Information Security in Latin America*, pages 130–149. Springer, 2019.

[PP19b] Thomas Pornin and Thomas Prest. More efficient algorithms for the ntru key generation using the field norm. In *IACR International Workshop on Public Key Cryptography*, pages 504–533. Springer, 2019.

[PPM17] Robert Primas, Peter Pessl, and Stefan Mangard. Single-trace side-channel attacks on masked lattice-based encryption. In *International Conference on Cryptographic Hardware and Embedded Systems*, pages 513–533. Springer, 2017.

[Pra62] Eugene Prange. The use of information sets in decoding cyclic codes. *IRE Transactions on Information Theory*, 8(5):5–9, 1962.

[PSB+18] Emmanuel Prouff, Remi Strullu, Ryad Benadjila, Eleonora Cagli, and Cecile Dumas. Study of deep learning techniques for side-channel analysis and introduction to ASCAD database. Cryptology ePrint Archive, Paper 2018/053, 2018.

[QA25] Jinyi Qiu and Aydin Aysu. Shift snare: Uncovering secret keys in falcon via single-trace analysis. *IACR Cryptol. ePrint Arch.*, 2025.

[QLZ+24] Zehua Qiao, Yuejun Liu, Yongbin Zhou, Yuhan Zhao, and Shuyi Chen. Single trace is all it takes: Efficient side-channel attack on dilithium. *Cryptology ePrint Archive*, 2024.

[VCGS14] Nicolas Veyrat-Charvillon, Benoît Gérard, and François-Xavier Standaert. Soft analytical side-channel attacks. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 282–296. Springer, 2014.

[ZLYW23] Shiduo Zhang, Xiuhan Lin, Yang Yu, and Weijia Wang. Improved power analysis attacks on falcon. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 565–595. Springer, 2023.