From Decoding Leakage to Full Key Recovery: A Single-Trace Attack on FALCON

Merve Karabulut[1], Panasayya Yalla[2], Reza Azarderakhsh[1] and Mehran Mozaffari Kermani[3]

[1]**Florida Atlantic University, Boca Raton, FL, USA**
[2]**Keysight Technologies, USA**
[3]**University of South Florida, Tampa, FL, USA**

Abstract

FALCON is one of the post-quantum signature schemes selected by NIST. Its compact keys make it attractive for embedded devices, which are exposed to side-channel attacks. Such attacks on FALCON have so far targeted its Gaussian sampler, its floating-point arithmetic, and its number-theoretic transform. We instead attack a step that runs before any of these and has gone unexamined: the routine that decodes the secret key at the start of every signing. We show that this decoding leaks each secret coefficient through two intermediate values it forms. One arises in the decoding itself, the other in the modular conversion that follows. A profiled template attack on these two points recovers each coefficient of the secret polynomials from a single power trace with 98.32% accuracy. Each prediction also carries a confidence score. A single trace does not recover every coefficient, so we add a key-recovery step. It ranks the predictions by confidence and solves the public key relation $g = h \cdot f$ for the rest, using only linear algebra and no lattice reduction. We evaluate the attack on an off-the-shelf ARM Cortex-M4 running FALCON-1024 at its nominal clock. The attack recovers the complete secret key for all 1000 tested keys, requiring 11.04 seconds on average. The decoding routine is shared across FALCON implementations, and the countermeasures proposed for the sampler and the arithmetic do not cover it. It must therefore be protected before FALCON is deployed.

Keywords: Post-Quantum Cryptography, FALCON, Side-Channel Analysis, Template Attack, Key Recovery

1. Introduction

Quantum computing poses a significant threat to classical public key cryptosystems whose security relies on the hardness of integer factorization [47] and discrete logarithm problems [12, 27]. Shor's algorithm demonstrates that these problems can be solved in polynomial time on a quantum computer [50]. In response, the National Institute of Standards and Technology (NIST) launched the Post-Quantum Cryptography (PQC) standardization process to develop cryptographic algorithms resistant to quantum attacks [34]. This effort resulted in the selection of ML-KEM (Module-Lattice-Based Key-Encapsulation Mechanism) [35] and HQC (Hamming Quasi-Cyclic) [33] for key encapsulation schemes. For digital signatures, ML-DSA (Module-Lattice-Based Digital Signature Algorithm) [36], FALCON [14], and SLH-DSA (Stateless Hash-Based Digital Signature Algorithm) [37] are selected. Among these signature standards, FALCON offers the most compact combined key and signature size, making it especially attractive for embedded and hardware platforms that operate under strict resource constraints. NIST is standardizing FALCON as FN-DSA [38]. Since the final FN-DSA standard was not yet available when this work was conducted, we refer to the scheme by its original name, FALCON, throughout this paper.

FALCON is a lattice-based digital signature scheme that follows the Gentry-Peikert-Vaikuntanathan (GPV) hash-and-sign framework [14]. It is derived from NTRU (Nth Degree Truncated Polynomial Ring Units), and its security relies on the hardness of finding short vectors in lattices [14]. While FALCON's security is strong against both classical and quantum adversaries, its implementations may remain vulnerable to side-channel attacks, which exploit physical characteristics to extract secret information [5, 9, 16, 17, 24, 26, 28–30, 32, 40, 44–46, 48, 49, 53, 54]. The most powerful of these attacks are single-trace attacks, which recover secret information from a single execution of the algorithm [39]. Because they require only one trace, leakage from any secret-dependent operation may be sufficient to compromise the key. Protecting individual components is therefore insufficient if secret-dependent leakage remains elsewhere in the implementation. Given its suitability for resource-constrained embedded devices, FALCON is a particularly attractive target for this class of attack.

FALCON has been the subject of several single-trace side-channel attacks, which target different components of its implementation: the floating-point conversions [26, 28], the discrete Gaussian sampler [45, 46], and the number-theoretic transform [44]. However, each of these attacks faces practical limitations. The attacks on the Gaussian sampler operate during key generation [45, 46], which typically runs in a controlled environment and is therefore difficult for an attacker to observe. The attack on the number-theoretic transform [44] targets signing, but to recover the full key, it relies on a lattice-reduction step, a computationally intensive procedure whose cost grows as more coefficients remain unrecoverable from the trace. The attacks on the floating-point conversions [26, 28] also target signing, but countermeasures for this exact arithmetic have been proposed [8, 25]. Since these attacks were evaluated only against unprotected implementations, it remains unknown whether they succeed once a countermeasure is applied. These limitations leave a gap in the study of practical single-trace attacks on a component of FALCON that has not yet been studied.

In this paper, we present a single-trace side-channel attack on FALCON that targets an earlier, previously unexamined step: the routine that decodes the secret key before it is used. This routine runs at the start of every signing operation, and no countermeasure has been proposed for it. Our attack is the first to target secret key decoding of FALCON and therefore exposes a vulnerability that is orthogonal to those found in the Gaussian sampler and the signing arithmetic [9, 26, 28, 44–46, 54].

Our attack exploits power leakage during secret key decoding to recover coefficients of the secret polynomials f and g. It rests on three steps. First, we show that the attacker does not need to recover every coefficient from the side-channel measurement. A full key has thousands of coefficients, so a few of them can be recovered incorrectly, and the attacker cannot tell which ones. The attacker can, however, tell which predictions are confident. The public key h relates the secret polynomials through $g = h \cdot f$, so the coefficients predicted with high confidence turn this relation into equations for the remaining ones. Solving these equations over a finite field completes the key without lattice reduction. Second, we examine the decoding routine of the reference implementation [21] and identify the operations that process each secret coefficient. A single operation is not enough, because different coefficient values produce the same leakage there. We therefore combine two operations that leak the same coefficient in different ways. Third, we build a profiled attack on these two operations and measure its per-coefficient accuracy on an ARM Cortex-M4. The public key relation then corrects the remaining errors and yields the full key. The contributions of this paper are as follows:

- We present the first side-channel attack on FALCON's secret key decoding, the routine that unpacks the secret polynomials from their packed form. In the evaluated implementation [21], this decoding runs on each signing invocation. To our knowledge, this step has not been targeted before, and unlike the sampler and floating-point operations attacked in prior work, it has no proposed countermeasure.
- We turn this leakage into a practical attack. At the decoding leakage point, multiple coefficient values produce identical leakage, making them indistinguishable. We overcome this by combining

Table 1: Parameter sets for the two security levels of FALCON [14].

Version	n	q	σ	$\lfloor \beta^2 \rfloor$	f, g encoding width in bits	Security
FALCON-512	512	12289	165.74	34 034 726	6	I
FALCON-1024	1024	12289	168.39	70 265 242	5	V

two leakage points, the decoding routine and the modular reduction that immediately follows it. Together, these two points recover coefficients of the secret polynomials f and g with 98.32% accuracy from a single signing execution.

- We introduce a key-recovery method that corrects the errors of the side channel using nothing but the FALCON's public key. The public key $h = gf^{-1} \bmod q$ satisfies for the secret polynomials f and g. Our method ranks the predicted coefficients by template confidence and solves this equation for the remaining ones. This corrects the errors left by the side channel and removes the need for lattice reduction.
- We demonstrate the practicality of the attack on the `pqm4` implementation of FALCON-1024[1] running on an STM32F4 device. From a single power trace, we recover coefficients of f and g and reconstruct the complete secret key using the public key. The attack recovers the full FALCON-1024 secret key in 11.04 seconds on average and succeeds on all 1000 randomly generated keys tested in our experiments.

Our results show that the secret key decoding of FALCON leaks enough information in a single power trace to recover the full key. This leakage is independent of the sampler and the floating-point arithmetic that existing countermeasures protect, so those countermeasures do not prevent it. Implementations of FALCON intended for embedded devices should therefore treat the key decoding routine as sensitive and protect it.

2. Background

In this section, we provide the background necessary to understand our attack. We first introduce the notation used throughout the paper, then describe the FALCON digital signature scheme, including key generation, signing, and verification. We conclude by defining the adversary model assumed in our attack, including the attacker's capabilities and access to the target device.

2.1 Notation

Let n and q be two integers, where $n \in \{512, 1024\}$ and $q = 12289$. We define R_q as the polynomial ring $\mathbb{Z}_q[x]/(\phi)$, where $\phi = x^n + 1$. The Euclidean norm $\|x\|$ denotes the square root of the sum of the squared coefficients of a polynomial x. Matrices are represented by bold capital letters, such as $\mathbf{B} \in R_q^{2\times 2}$. Vectors are represented by bold lowercase letters, such as $\mathbf{t}$. Polynomials in the FFT domain are denoted with a hat (e.g., $\hat{c}$, where $\hat{c} = \mathrm{FFT}(c)$). This notation is transitive, so $\hat{\mathbf{B}}$ represents each polynomial in $\mathbf{B}$ being individually transformed into the FFT domain. The symbol "$\odot$" denotes pointwise multiplication, and "$\perp$" indicates an invalid or undefined value. The superscript "$*$" denotes the adjoint of a matrix.

2.2 FALCON Digital Signature Algorithm

To understand the source of the leakage exploited in our attack, it is necessary to examine the FALCON digital signature scheme. FALCON is a lattice-based digital signature scheme selected by NIST for PQC standardization [38]. Its security is based on the hardness of the NTRU problem and follows the

[1] https://github.com/mupq/pqm4/tree/a24bb4b

Algorithm 1: FALCON: Key Generation, Signing, and Verification [14].

KeyGen()
- $(f, g, F, G) \leftarrow \text{NTRUGen}(\phi, q)$
- $h \leftarrow g \cdot f^{-1} \bmod q$
- $sk \leftarrow \text{skEncode}(f, g, F)$
- $pk \leftarrow \text{pkEncode}(h)$
- **return** (pk, sk)

Sign(sk, M)
- $(f, g, F) \leftarrow \text{skDecode}(sk)$
- $G \leftarrow g \cdot F \cdot f^{-1} \bmod q$ ▷,reconstruct G from $fG - gF = q$
- $\mathbf{B} \leftarrow \begin{bmatrix} g & -f \\ G & -F \end{bmatrix}$
- $\hat{\mathbf{B}} \leftarrow \text{FFT}(\mathbf{B})$
- $T \leftarrow \text{ffLDL}^*(\hat{\mathbf{B}}\hat{\mathbf{B}}^*)$
- $r \leftarrow \{0,1\}^{320}$
- $c \leftarrow \text{HashToPoint}(r \| M)$
- $\mathbf{t} \leftarrow \left(-\frac{1}{q}\text{FFT}(c) \odot \text{FFT}(F), \frac{1}{q}\text{FFT}(c) \odot \text{FFT}(f)\right)$
- $s \leftarrow \perp$
- **while** $s = \perp$ **do**
  - $\mathbf{z} \leftarrow \text{ffSampling}_n(\mathbf{t}, T)$
  - $\mathbf{s}' \leftarrow (\mathbf{t} - \mathbf{z}) \cdot \hat{\mathbf{B}}$
  - **if** $\|\mathbf{s}'\|^2 > \lfloor \beta^2 \rfloor$ **then**
    - $s \leftarrow \perp$
  - **else**
    - $(s_1, s_2) \leftarrow \text{invFFT}(\mathbf{s}')$
    - $s \leftarrow \text{Compress}(s_2)$
- **return** $\sigma = (r, s)$

Verify$(pk, M, \sigma = (r, s))$
- $h \leftarrow \text{pkDecode}(pk)$
- $c \leftarrow \text{HashToPoint}(r \| M)$
- $s_2 \leftarrow \text{Decompress}(s)$
- $s_1 \leftarrow c - s_2 \cdot h \bmod q$
- **return** $\|(s_1, s_2)\|^2 \leq \lfloor \beta^2 \rfloor$

GPV hash-and-sign framework over structured lattices. The scheme operates over the polynomial ring $R_q = \mathbb{Z}_q[x]/(x^n + 1)$ with modulus $q = 12289$ [14].

FALCON has two variants, FALCON-512 and FALCON-1024 as shown in the Table 1. The modulus q and the form of the ring polynomial $\phi = x^n + 1$ are the same in both, while the ring degree n doubles from 512 to 1024. The degree sets the security level: FALCON-512 meets NIST level I and FALCON-1024 meets level V, the highest level defined by NIST. The larger degree raises the signing bound $\lfloor \beta^2 \rfloor$ and the Gaussian width σ, and the public key, secret key, and signature all grow with n. In this study, we target FALCON-1024, which provides the strongest security guarantees within the scheme.

The FALCON scheme includes key generation, signature generation, and signature verification, as shown in Algorithm 1. Key generation begins with NTRUGen to sample two short polynomials f and g [14]. These must satisfy three conditions. First, f must be invertible modulo q, so that the public key $h = gf^{-1} \bmod q$ can be formed. Second, the basis must be short enough to meet the Gram-Schmidt norm bound $\gamma \leq 1.17\sqrt{q}$ that FALCON requires for signing. Third, every coefficient of f and g must fit the packed secret-key encoding, which stores each coefficient in 5 bits over the range $[-15, 15]$ for FALCON-1024 and in 6 bits over $[-31, 31]$ for FALCON-512, as listed in Table 1. From f and g, two further polynomials F and G are computed such that f, g, F, G jointly satisfy the NTRU equation $fG - gF = q \pmod{\phi}$. The public key is computed as $h = gf^{-1} \bmod q$ and published for use during

signature verification. The secret polynomials f, g, and F are packed into a compact encoded form through skEncode, which becomes the secret key, with each coefficient of F stored in 8 bits in both variants. The polynomial G is not stored directly, since it can be recomputed from f, g, and F [14].

The signature generation procedure begins by decoding the secret key through the skDecode operation to reconstruct the secret polynomials (f, g, F). The polynomial G is not stored in the secret key; it is recomputed here from f, g, and F through the NTRU equation $fG - gF = q$, which yields $G = g \cdot F \cdot f^{-1} \bmod q$. These four polynomials (f, g, F, G) are used to construct the secret lattice basis $\mathbf{B} = \left[\begin{smallmatrix} g & -f \\ G & -F \end{smallmatrix}\right]$, which is transformed into the Fourier domain. There, it is factored using ffLDL*. The LDL decomposition is a standard matrix factorization that splits a matrix into a lower-triangular part and a diagonal part. ffLDL* is simply this factorization carried out in the FFT domain. This factorization produces the Falcon tree T, a data structure used later during Gaussian sampling to draw a short vector efficiently. A random salt r is concatenated with the message M and mapped to a polynomial c via HashToPoint, binding the eventual signature to both the message and a fresh source of randomness. FALCON then repeatedly invokes ffSampling to draw a candidate short vector, accepting it only once it falls within the norm bound $\lfloor \beta^2 \rfloor$ required for a valid signature. The accepted vector is split into the polynomial pair (s_1, s_2) and compressed to produce the final signature $\sigma = (r, s)$ [14].

The signature verification procedure uses the public key h, the message M, and the signature $\sigma = (r, s)$ to check the signature's validity. The verifier first recomputes the challenge polynomial $c = \text{HashToPoint}(r \| M)$ and decompresses the signature component s to recover s_2. It then reconstructs the companion value $s_1 = c - s_2 h \bmod q$ using only public information, exploiting the same relation between h, s_1, and s_2 that the signer satisfies during signing. Finally, the verifier checks whether the resulting vector (s_1, s_2) satisfies the norm bound $\|(s_1, s_2)\|^2 \leq \lfloor \beta^2 \rfloor$, accepting the signature if the condition holds and rejecting it otherwise [14].

2.3 Adversary Model

We adopt the standard threat model for single-trace side-channel attacks [1, 13, 18, 20, 22, 26, 28, 41–46, 51, 52]. The adversary's goal is to recover the secret key that a target device uses for FALCON signing. It has physical access to the device and can measure its power consumption while the device signs. Following standard profiled side-channel analysis assumptions, the adversary knows the FALCON algorithm, the target software implementation, and all public parameters. It also knows the public key h, which is non-secret information by definition [14].

The attack proceeds in two phases: a profiling phase and an attack phase. In the profiling phase, the adversary has access to a clone device identical to the target and full control over its secret key and inputs. This allows the adversary to collect power traces under known secret values and build leakage templates offline. In the attack phase, the adversary no longer controls or knows the secret key and is restricted to a single power trace captured from the target device during signing. The adversary uses the templates built during profiling to recover the secret key from this single trace.

3. Core Principle of the Attack

This section presents the principle on which our attack rests. We first describe the secret polynomials we target and why extracting them enables recovering the full secret key of FALCON. We then show that the public key of FALCON allows the attacker to complete the key from a subset of their coefficients and to discard the mispredicted ones.

3.1 The Targeted Operation and Rationale

The secret key of FALCON consists of the four polynomials f, g, F, and G. Together they form the secret basis $\mathbf{B}$ used in line 10 of Algorithm 1, and they are related by the NTRU equation

$$fG - gF = q \pmod{\phi}. \tag{1}$$

Key generation samples f and g first and then derives F and G from them with the public, deterministic NTRUSolve procedure [14]. Because F and G are recomputed from f and g by this same procedure, they carry no information of their own, and recovering f and g is enough to recover the entire signing key. Given the recovered f and g, the adversary runs NTRUSolve, obtains a short pair (F, G) satisfying Equation (1), and assembles a secret basis $\mathbf{B}$ for the victim's public key. This basis does not need to be equal to the victim's, yet it signs and verifies exactly like it.

To understand why the reconstructed basis is sufficient for forgery, consider the verification procedure of FALCON [14], reproduced in Algorithm 1. The verifier recomputes the challenge c, reconstructs $s_1 = c - s_2 h \bmod q$ from the public key h, and accepts the signature only when the vector (s_1, s_2) is short enough, that is, when $\|(s_1, s_2)\|^2 \leq \lfloor \beta^2 \rfloor$. This test depends only on h and the norm bound $\lfloor \beta^2 \rfloor$, and not on which secret basis was used to sign. The pair (F, G) that the adversary obtains from NTRUSolve is short and satisfies the NTRU equation for the same h, so the basis it forms signs within the same bound, so its signatures pass verification exactly as the victim's do, and no verifier can tell the two bases apart. The security of FALCON's secret key therefore rests on f and g alone.

A side-channel can reveal f and g only where the implementation processes their coefficients, which first happens during decoding. FALCON stores its secret key in a packed form to keep it compact. The signing procedure cannot operate on this form. As shown in Algorithm 1, it therefore begins by calling skDecode, which unpacks the secret polynomials f, g, and F into their integer coefficients before any other computation takes place. The decoding processes every coefficient of f and g one after the other, so a single power trace of this step contains the processing of the entire secret polynomials f and g. The adversary does not need to recover F from the trace, even though skDecode also unpacks it.

Signing is also the phase an adversary can observe in practice. It runs every time the device signs a message, so the decoding is exposed for the lifetime of the key. In the evaluated implementation, it decodes the secret key on each signing invocation. Key generation, by contrast, runs once and in a controlled environment, and the attacks on it [45, 46] assume the adversary is present at that single moment. The decoding also asks little of the adversary. It does not depend on the message or on the signing randomness, so no control over the device input and no knowledge of the resulting signature are required.

3.2 Completing the Key from Partial Coefficient Recovery

Besides the NTRU equation, the public key gives a second relation between f and g. Key generation computes $h = g f^{-1} \bmod q$ and publishes h. Equivalently, $g = h \cdot f$ in R_q, where h is known and f and g are secret. At the coefficient level, this product is a system of linear equations:

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

This gives n equations in the $2n$ unknown coefficients of f and g. The system is under-determined, so the public key alone cannot recover the secret key. This raises the question of how many coefficients the side-channel must supply before Equation (2) determines the rest.

Suppose the attacker already knows d_f coefficients of f and d_g coefficients of g and treats them as

```
fndsa_trim_i8_decode(uint8_t *sk, int8_t *poly)
 1  for i = 0 to needed - 1 do
 2      acc = (acc << 8) | sk[i]
 3      acc_len = acc_len + 8
 4      while acc_len >= nbits do
 5          acc_len = acc_len - nbits
 6          uint32_t c = (acc >> acc_len) & mask1
 7          c = c | -(c & mask2)
 8          poly[u++] = (int8_t)c
 9      end while
10  end for
```

Figure 1: The `fndsa_trim_i8_decode` routine. Coefficients extracted from the packed secret key `sk` are sign-extended and written to the output polynomial `poly`. The highlighted assignment in line 8 is the first leakage point targeted in our attack [21].

trusted. In Equation (2), each unknown coefficient multiplies one column of the system's matrix. Fixing a trusted coefficient moves its column to the known side and leaves one unknown column fewer. The remaining coefficients are determined uniquely when these leftover columns are linearly independent, so that none of them is a combination of the others. Independent columns cannot outnumber the n rows, which gives

$$d_f + d_g \geq n, \tag{3}$$

the same threshold used in [28]. Therefore, the attack needs only n of the $2n$ coefficients, half of the key. Reaching this count leaves two conditions to meet, and the attack enforces both. First, the trusted coefficients it uses are correct. Second, it chooses them so that the leftover columns are independent, pinning down every remaining unknown. If a column is still a combination of the others, the attack adds further trusted coefficients, each removing one more column, until the rest are independent. Because $q = 12289$ is prime, the values modulo q form a field, so the whole system is solved by ordinary linear algebra, with no lattice reduction at any point.

The attack also makes sure the trusted coefficients are correct by testing the candidate each subset produces. A wrong coefficient drives the solved values far from any real key, and two checks catch it at once. In FALCON-1024, the true coefficients of f and g lie in $[-15, 15]$, so a candidate with any value outside this range is rejected outright. A candidate that survives is then matched against the public key, where the solved f and g must satisfy $g \equiv hf \pmod{q}$ in every coordinate with no residual. A candidate that fails either check is dropped, and the attack moves to a different subset. Candidates passing these filters undergo further validation. We check the invertibility of f, reconstruct a suitable short pair (F, G), verify the integer NTRU equation, and generate a signature that verifies under the target public key.

4. The Proposed Attack

In this section, we analyze how the secret key decoding is implemented on the target design and how it processes each coefficient of f and g. We then expose two leakage points, each revealing the coefficient value and together resolving what either leaves ambiguous. Finally, we show how a profiled template attack turns these two points into the complete secret key, recovered from a single power trace.

4.1 Identifying the Leakage Sources

The target implementation performs the decoding in `trim_i8_decode`, whose structure is shown in Figure 1 . The function takes the packed secret key `sk` and writes the recovered coefficients into the output

```
fndsa_trim_i8_decode:
 1  stmdb  sp!, {r4,r5,r6,r7,r8,r9,sl,lr}
 2  mov.w  ip, #1
 3  subs   r4, r3, #1
 4  lsl.w  r5, r3, r0
 5  lsl.w  r8, ip, r3
 6  lsl.w  ip, ip, r4                        ; ip = ip << r4
 7  movs   r4, #0
 8  lsrs   r0, r5, #3
 9  add.w  r8, r8, #-1                       ; r8 = r8 - 1
10  add.w  r5, r1, r5, lsr #3
11  mov    lr, r4
12  mov    r6, r4
13  rsb    r9, ip, #0                       ; r9 = -ip
14 .Louter: cmp r1, r5
15      bne.n  .Lwork
16      ldmia.w sp!, {r4,r5,r6,r7,r8,r9,sl,pc}
17 .Lwork: ldrb r7, [r1]
18      adds   r4, #8
19      orr.w  lr, r7, lr, lsl #8             ; lr = (lr << 8) | r7
20      add    r6, r2
21 .Lemit: cmp r4, r3
22      bcs.n  .Lextract
23      adds   r1, #1
24      subs   r6, r6, r2
25      b.n    .Louter
26 .Lextract: subs r4, r4, r3
27      lsr.w  sl, lr, r4                     ; sl = lr >> r4
28      and.w  sl, sl, r8                     ; sl = sl & r8
29      and.w  r7, sl, ip
30      negs   r7, r7
31      orr.w  r7, r7, sl                     ; r7 = sl | -(sl & ip)
32      cmp    r7, r9
33      beq.n  .Lreject
34      strb.w r7, [r6], #1                   ; store r7  --> Point 1
35      b.n    .Lemit
```

Figure 2: Cortex-M4 assembly of `trim_i8_decode` on the m4f target. The coefficient is extracted with `mask1` in line 28. The `negs` and `orr` instructions in lines 30 and 31 form the sign-extended coefficient in register `r7`, which is stored to memory by the `strb` instruction in line 34 [21].

array `poly`, which holds the coefficients of f and g as signed bytes. In FALCON-1024 each coefficient occupies `nbits`=5 bits, and the coefficients are packed back to back without regard to byte boundaries, so the routine rebuilds them with a bit accumulator. The outer loop, in lines 2 and 3, reads `sk` one byte at a time, shifting each byte into the accumulator `acc` and increasing the bit count `acc_len` by eight. Once `acc_len` reaches `nbits`, the inner loop of lines 4 to 9 emits one coefficient. Line 6 shifts `acc` down and applies `mask1 = 0x1F` to keep the low five bits, giving an unsigned value `c` between 0 and 31. Line 7 sign-extends `c` with `mask2 = 0x10`, which isolates the sign bit of the five-bit field; when that bit is set, the high bits of `c` are filled with ones to form the negative value. Line 8 writes `c` to `poly` as a signed byte. The routine repeats this until every coefficient is decoded, one at a time, so the adversary can target each in turn.

To see how the sign-extended coefficient is processed on the device, we compile `trim_i8_decode` with arm-none-eabi-gcc for the Cortex-M4 core of the m4f target. The corresponding assembly is shown in Figure 2. The routine keeps a bit accumulator in a register and refills it one byte at a time: line 17 loads the next byte of the packed key, and line 19 appends it to the accumulator with a shift and an `orr`. Once at least five bits are available, line 27 shifts the accumulator down to the next coefficient and line 28 masks it with `mask1`, the constant `0x1F`, leaving the unsigned five-bit value between 0 and 31 in a register. The sign extension follows at lines 29 to 31. Line 29 isolates the sign bit by masking the value with

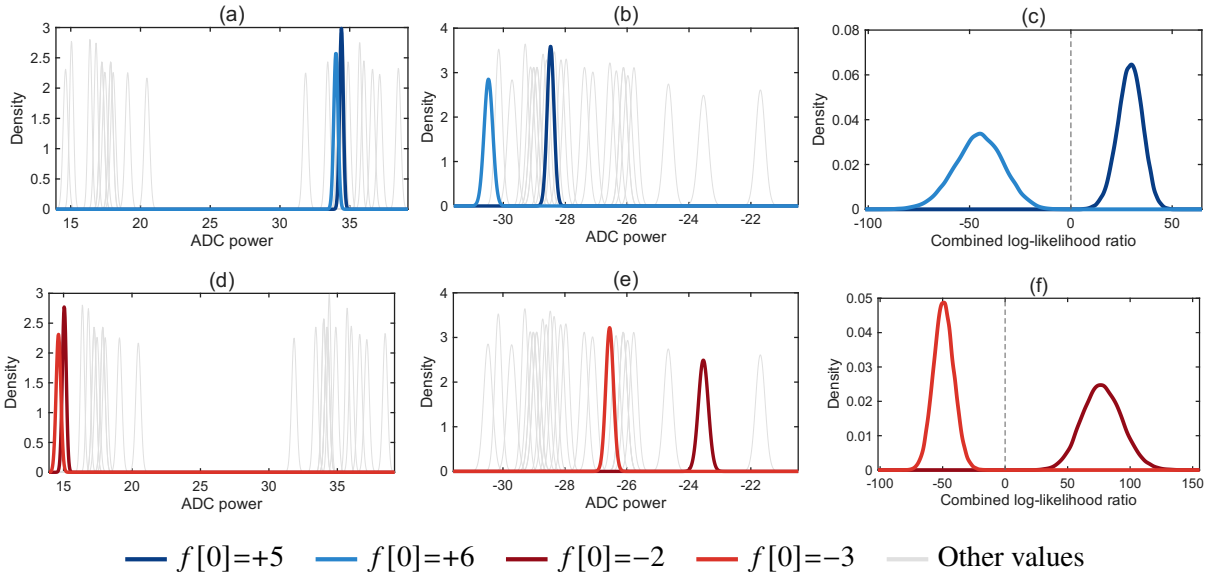

Figure 3: Distributions of the leakage at the two points and of the combined log-likelihood score, for a coefficient taking different values. The Point 1 and Point 2 panels show the ADC power measurement at the decoding and modular-conversion points, while the combined panels show the log-likelihood score that the attack forms from both points, which is largest for the value that best matches the measured leakage. The highlighted pairs $(+5, +6)$ and $(-2, -3)$ are competing candidate values for one coefficient that share a Hamming weight and cannot be told apart at Point 1. The first row shows $(+5, +6)$ in (a), (b), and (c), and the second row shows $(-2, -3)$ in (d), (e), and (f), each in the order Point 1, Point 2, combined log-likelihood score.

`0x10`, line 30 negates the result, and line 31 `orr`s it back onto the value. When the sign bit is set, this extends the sign through the high bits and forms the negative coefficient; when it is clear, the value is left unchanged. After a comparison that discards the one forbidden value, line 34 writes the coefficient to the output polynomial with the `strb` store, as a signed eight-bit byte.

Several instructions involve the coefficient value; therefore, we target the one that exhibits the strongest leakage and provides the clearest separation between candidate values. The mask and sign-extension instructions keep the coefficient in registers and operate on intermediate representations of it. In contrast, the `strb` instruction at line 34 writes the final signed coefficient to memory, causing the value to be transferred across the data bus. Memory stores are among the strongest data-dependent operations on this core [10], and our leakage analysis confirms that the dominant sample in the decoding window coincides with this instruction. Because the device's power consumption is proportional to the Hamming weight of the transferred data [7], the store reveals the Hamming weight of the sign-extended coefficient. We therefore use this store as our first leakage source, denoted Point 1.

The value stored at line 34 also reveals the sign of the coefficient. Figures 3(a) and 3(d) show the power distributions measured at Point 1 for different coefficient values. The distributions form two distinct groups: negative coefficients on the left and positive coefficients on the right, with no overlap between them. This separation comes from the two's-complement representation of the stored byte. For example, $+5$ is stored as 00000101_2, whereas -5 is stored as 11111011_2. Because negative values contain more set bits, they have a larger Hamming weight. This difference is reflected in the measured leakage, producing a clear distinction between positive and negative coefficients. As a result, the sign of each coefficient can be identified directly from Point 1. Distinguishing magnitudes within the same sign is more difficult. Values with the same Hamming weight generate nearly identical leakage. For example, $+5$ (00000101_2) and $+6$ (00000110_2) both contain two set bits, making them indistinguishable at Point 1. Resolving such collisions requires a second leakage point.

That second operation is the modular conversion that follows the decoding. The coefficients of f

```
fndsa_mqpoly_small_to_int:
 1  push   {r4,r5,r6,r7}
 2  movs   r3, #1
 3  lsls   r3, r0
 4  movw   r0, #12289
 5  movt   r0, #12289              ; r0 = 0x3001_3001
 6  mov.w  r7, #65537              ; r7 = 0x0001_0001
 7 .L1: ldr.w  r5, [r1], #4        ; r5 = 4 packed coefficients
 8      sxtb16 r4, r5              ; r4 = [ c2 | c0 ]
 9      sadd16 r6, r4, r0          ; r6 = r4 + r0  --> Point 2
10      ssub16 ip, r4, r7
11      sel    r4, r4, r6
12      sxtb16 r5, r5, ror #8      ; r5 = [ c3 | c1 ]
13      sadd16 r6, r5, r0          ; r6 = r5 + r0
14      ssub16 ip, r5, r7
15      sel    r5, r5, r6
16      pkhtb  r6, r5, r4, asr #16
17      pkhbt  r5, r4, r5, lsl #16
18      strd   r5, r6, [r2], #8    ; store r5, r6
19      subs   r3, #4
20      bne.n  .L1
21      pop    {r4,r5,r6,r7}
22      bx     lr
```

Figure 4: Cortex-M4 assembly of `mqpoly_small_to_int` on the m4f target. Two coefficients are sign-extended with `sxtb16` in line 8, and the packed `sadd16` instruction adds q to both at once in line 9, forming the sum c+q. The second pair is handled in lines 12–15, and the four converted coefficients are repacked and stored in lines 16–18 [21].

and g are converted into their internal representation modulo q by the function `mqpoly_small_to_int`. Unlike the decoding routine, this operation is not compiled from C for the Cortex-M4 target. Instead, it is implemented as the optimized assembly routine shown in Figure 4, which uses the Cortex-M4 packed arithmetic instructions to process multiple coefficients in parallel. Each loop iteration handles four signed coefficients. The load at line 7 fetches all four at once into a single 32-bit register. These coefficients are then split into two pairs. The `sxtb16` instruction at line 8 sign-extends two coefficients into the upper and lower halves of register `r4`, producing `[c2 | c0]`. A constant register already holds two copies of the modulus $q = 12289$, one in each 16-bit half. The packed addition at line 9 therefore computes `[c2+q | c0+q]` in a single instruction. The subsequent `ssub16` and `sel` instructions operate on both halves independently and choose either `c` or `c+q` for each coefficient, thereby converting them into representatives in $[0, q)$. Lines 12–15 repeat the same sequence for the remaining pair, `[c3 | c1]`. Finally, lines 16–18 repack the four converted coefficients and store them to memory. Thus, throughout the modular conversion, coefficients are processed in pairs, with a single packed arithmetic instruction acting on two coefficients simultaneously.

Because the addition at line 9 is packed, the register it produces holds the sums `c+q` of two coefficients at once, so its leakage depends on both. We take this sum as our second leakage source, Point 2, in preference to the representative written by the final store. The reason is that `c+q` is formed for every coefficient, positive ones included, whereas the final store keeps a positive coefficient unchanged and would only repeat for it what Point 1 already shows. Adding q changes the bit pattern of every coefficient. Recall that +5 and +6 cannot be separated at Point 1. After the addition they become $q + 5 = 12294$ and $q + 6 = 12295$. Their binary representations,

$$12294 = 0011000000000110_2, \qquad 12295 = 0011000000000111_2,$$

have Hamming weights 4 and 5, respectively. The two coefficients can therefore be distinguished at Point 2 even though Point 1 cannot separate them. We target this addition for its separability rather than

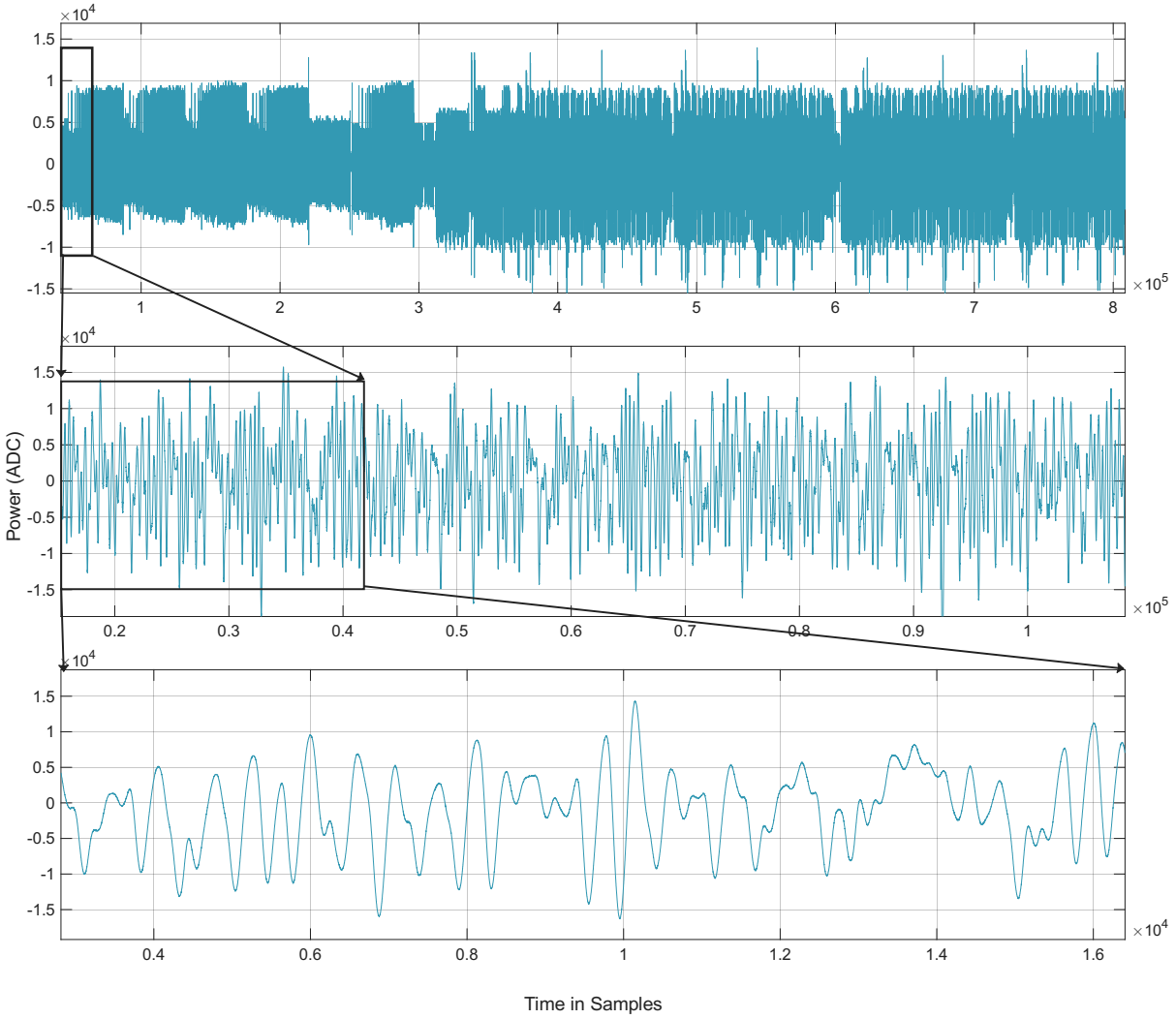

Figure 5: Power trace of a single FALCON signing execution. The top panel shows the complete power trace captured during one FALCON-1024 signature generation operation. The highlighted region corresponds to the decoding of the secret polynomial f, which is performed before the main signing computations. The middle panel zooms into this decoding region and reveals a sequence of highly regular operations. The bottom panel provides a further magnified view, showing the per-coefficient processing pattern in which each coefficient is decoded within its own short, repeating window. These windows form the basis for the alignment procedure used by the attack and enable the extraction of coefficient-specific leakage from a single trace.

its leakage strength. As a short arithmetic step it leaks less than a store, but it separates the values that Point 1 leaves ambiguous, which is what the second point must do. Each coefficient of f and g is therefore processed twice, once as its sign-extended value at Point 1 and once as the sum `c+q` at Point 2.

Although the Hamming weight example explains why the two leakage points differ in their ability to separate coefficient values, the attack itself does not rely on a Hamming weight model. It uses the leakage measured on the device rather than a Hamming weight prediction, so values that would appear identical under such a model can still be distinguished in practice. Values that are ambiguous at Point 1 become more separable when Point 2 is added, as confirmed by the measured traces in Section 4.4. The two points do not need to separate every value perfectly. The key-recovery procedure in Section 4.5 uses only a subset of coefficients and verifies each candidate, allowing the few remaining ambiguities to be corrected during recovery.

4.2 Pinpointing the Regions of Interest

Before the two points can be used, the processing of each coefficient must be located in the trace. The secret key decoding is easy to find, as it is the first operation of signing and runs before the floating-point computations that fill the rest of the trace. The top panel of Figure 5 shows the power trace of a complete FALCON-1024 signing, with the decoding region at its start. The middle panel zooms into the decoding of f, and the bottom panel shows several iterations of the decoding loop, one per coefficient. These iterations have the same shape, because the loop runs the same instructions for every coefficient. We use this regularity to cut the decoding region into one sub-trace per coefficient and align the sub-traces with one another, using an established technique [2] that matches them by their maximum and minimum power draw. Alignment is the only processing we apply, and each aligned sub-trace holds the Point 1 sample of one coefficient at the same offset.

The modular conversion is located and aligned in the same way, with one complication. Its Cortex-M4 routine is packed and adds the modulus to two coefficients at once with each `sadd16`, so the leakage at Point 2 reflects a pair of coefficients rather than a single one. Locating a coefficient is still straightforward, because the routine visits the coefficients in the same order as the decoding, so coefficient i lies at a known offset. The difficulty is that the Point 2 sample at that offset mixes coefficient i with the coefficient packed alongside it in the same `sadd16`. We do not assume the neighbor is known. Instead, we marginalize over it. During profiling the neighboring coefficient takes many values across traces, so the Point 2 template estimated for a given value of coefficient i already averages over the neighbor's contribution. The neighbor adds a term that varies from trace to trace but not with the value of coefficient i. It widens the template yet leaves the per-value means separated, so it enters as nuisance variability rather than a quantity that must be fixed in advance. The neighbor is thus accounted for from the attack and profiling traces themselves, never taken from the public key. Targeting this packed, two-coefficient leakage follows earlier attacks on packed Cortex-M4 arithmetic [4, 19]. Each coefficient then has one Point 1 sample and one Point 2 sample.

4.3 Constructing the Profiling Model

We now construct a profiled model to recover the secret coefficients, in two steps. The first identifies the points of interest, and the second builds the templates. Each per-coefficient sub-trace from Section 4.2 contains many samples, but only a few of them carry information about the coefficient; the rest reflect unrelated computation and measurement noise. The first step finds those few samples. We group the profiling traces by the value of the coefficient being processed, so that each possible value forms one leakage class. For a class i of N_i traces, we first compute, at each sample t, the class mean and variance,

$$m_i(t) = \frac{1}{N_i}\sum_{j=1}^{N_i} x_{i,j}(t), \qquad \sigma_i^2(t) = \frac{1}{N_i}\sum_{j=1}^{N_i}\big(x_{i,j}(t) - m_i(t)\big)^2, \tag{4}$$

where $x_{i,j}(t)$ is sample t of the j-th trace in class i. A sample is useful when these class means are far apart relative to the within-class spread. The Sum Of Squared T-differences (SOST) metric [15] captures this by forming the Welch t-statistic between every pair of classes at sample t, squaring it, and summing over all pairs,

$$\mathrm{SOST}(t) = \sum_{i<j}\left(\frac{m_i(t) - m_j(t)}{\sqrt{\sigma_i^2(t)/N_i + \sigma_j^2(t)/N_j}}\right)^2. \tag{5}$$

A single term is large when the two classes have well-separated means and little spread of their own, so $\mathrm{SOST}(t)$ is large only at samples where the coefficient value leaves a clear mark on the power. We evaluate it across the sub-trace and take the samples at its peaks as the points of interest.

Figure 6 shows the resulting SOST curve around the two operations targeted in our attack. The

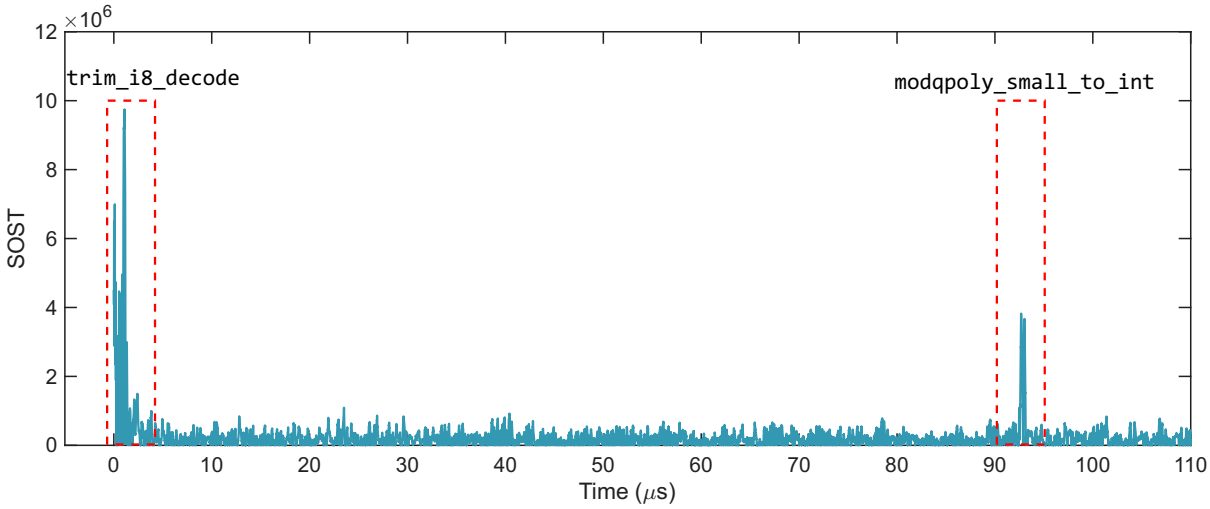

Figure 6: SOST over the trace region containing the two targeted operations. The dominant peak is the sign extension in `trim_i8_decode`, which forms Point 1. The second peak is the modular addition in `mqpoly_small_to_int`, which forms Point 2.

dominant peak corresponds to the sign extension of Point 1. The modular addition of Point 2 produces a second, clearly smaller peak, so this operation leaks less than the sign extension, as expected for a short arithmetic step rather than a store. Although its peak is small, we retain Point 2, because it separates coefficient values that Point 1 alone cannot. The samples at these two peaks are the points of interest, and they are the samples shown in Figure 3.

The second step builds a Gaussian template for every possible coefficient value [7]. During profiling, we collect traces under known secret keys and group them by the value of the coefficient being processed. For each value, we estimate the mean and covariance of its samples at the two points of interest, which together describe how the device leaks when the coefficient takes that value. Because the leakage behaves the same way at every coefficient position, the templates are estimated once and then applied unchanged to every coefficient of f and g.

4.4 Recovering the Coefficients from a Single Trace

Given these templates, we recover the coefficients from a single trace. The trace is divided into per-coefficient windows as described in Section 4.2, so that each coefficient contributes one leakage sample at Point 1 and one at Point 2, which we denote x_1 and x_2. To identify the coefficient, we ask which of its possible values is most consistent with this pair of samples.

Each candidate value v is described by the templates built during profiling, one for each point. The template at point p is a univariate Gaussian with mean $\mu_{v,p}$ and variance $\sigma^2_{v,p}$, and it summarizes how the device leaks when the coefficient equals v at that point. Evaluating its density $\mathcal{N}(x_p; \mu_{v,p}, \sigma^2_{v,p})$ at the observed sample x_p gives the likelihood of that sample under value v. The density is large when x_p lies close to the mean $\mu_{v,p}$, where the device usually leaks for this value, and small when x_p lies far from it, so each sample votes for the values whose templates it fits and against those it does not.

A single point rarely settles the value on its own, so we combine the two. The Point 2 likelihood incorporates the paired-coefficient dependency through the marginalization described in Section 4.2, so each point enters as a per-value density for coefficient i alone. We then combine the Point 1 and Point 2 samples under a conditional independence model, which approximates the likelihood of the pair for a candidate value v as the product of the corresponding per-point densities. This conditional independence is a modeling approximation rather than an exact property of the device. We work with the logarithm of this product, which is numerically stable and turns it into a sum. The resulting combined log-likelihood

$\mathcal{L}(v)$ is

$$\mathcal{L}(v) = \log \mathcal{N}\left(x_1; \mu_{v,1}, \sigma^2_{v,1}\right) + \log \mathcal{N}\left(x_2; \mu_{v,2}, \sigma^2_{v,2}\right). \tag{6}$$

The first term measures how well the Point 1 sample matches value v, and the second measures the Point 2 sample, so $\mathcal{L}(v)$ gathers the evidence from both points into a single number. A high $\mathcal{L}(v)$ means the measured leakage matches value v well at Point 1 and at Point 2 at the same time. Because the logarithm is monotonic, the value that maximizes $\mathcal{L}(v)$ is the most likely value overall, and we take it as the prediction for the coefficient,

$$\hat{v} = \arg\max_v \mathcal{L}(v).$$

Since the two points enter the sum together, a value that one point cannot separate from its neighbors is resolved by the other, which is what raises the accuracy above either point alone. This combined log-likelihood is the quantity plotted in the third column of Figure 3.

Figure 3 illustrates this behavior for two representative coefficient pairs, $(+5, +6)$ and $(-2, -3)$, while all other coefficient values are shown in gray. Each panel plots the distribution of a leakage quantity across many traces for the two values in the pair. At Point 1, both values in a pair produce nearly the same power consumption, causing their distributions to overlap almost completely. As a result, the two values cannot be distinguished using Point 1 alone, as shown in panels (a) and (d). At Point 2, however, the addition of q changes the integer representation and shifts the two values to different power levels. This separates their distributions, as shown in panels (b) and (e), providing information that was unavailable at Point 1. To determine the coefficient value, we combine the evidence from both points using the log-likelihood score in Equation (6). For each trace, the difference between the two candidates' combined log-likelihoods is computed. As shown in panels (c) and (f), this combined log-likelihood ratio places the two values on opposite sides of the zero decision boundary, so every trace is classified correctly.

Even so, a prediction can still be wrong, because a few value pairs are not fully separable and single-trace noise can tip a close decision. We therefore attach a confidence score to each prediction, equal to the gap between the highest and the second-highest log-likelihood,

$$\mathcal{L}(\hat{v}) - \max_{v \neq \hat{v}} \mathcal{L}(v).$$

This gap measures how far the chosen value $\hat{v}$ outscores its closest competitor. A large gap means one value matches the leakage far better than any other, so the prediction is almost certainly correct. A small gap means two values match the leakage almost equally well, which is exactly the case in which the prediction is likely to be wrong.

4.5 Completing the Key

We use the confidence scores obtained during profiling to complete the key. All $2n$ coefficient predictions are ranked by confidence, and the $d \geq n$ most reliable predictions are selected as a trusted set, leaving only a small number of low-confidence coefficients unresolved. These trusted coefficients are then passed to the completion procedure of Section 3.2, which uses the public key relation to solve for the remaining coefficients and verifies each candidate solution. Any candidate that contains an incorrect prediction fails the verification step and is discarded in favor of the next candidate, requiring only a single linear solve per attempt. As a result, the recovery process remains efficient and avoids lattice reduction entirely. This procedure reconstructs the full secret key even though the side-channel attack recovers only probabilistic coefficient estimates rather than exact coefficient values.

5. Experimental Results

In this section, we evaluate the attack on a real target. We first present the measurement setup and the FALCON-1024 implementation under attack. We then report the per-coefficient success rate of the

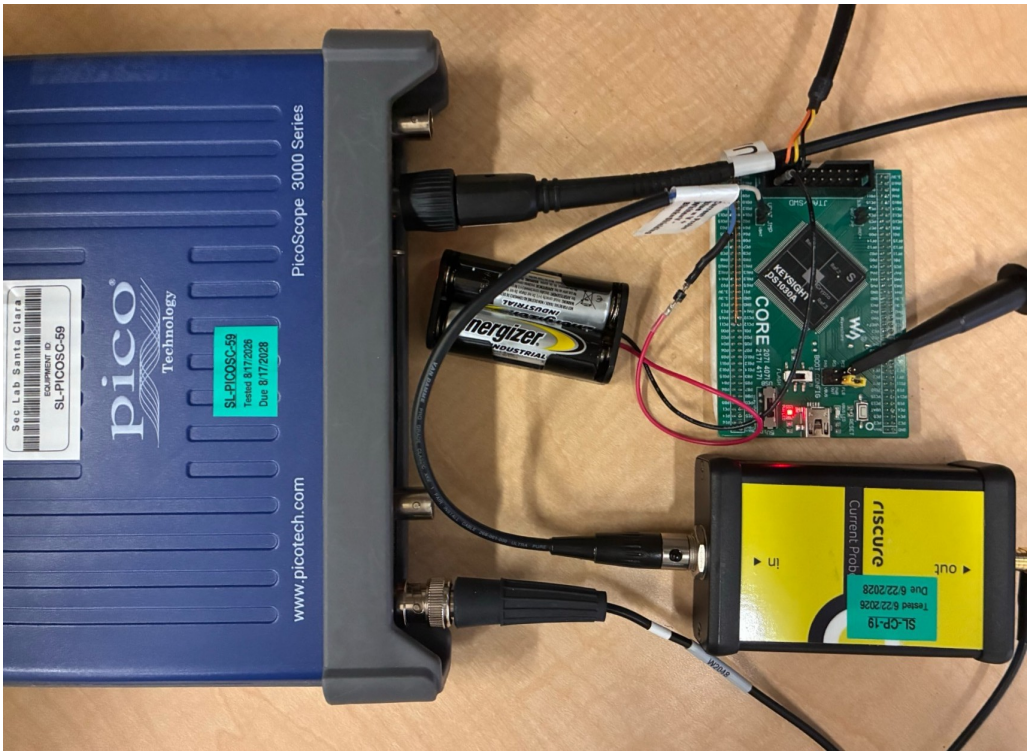

Figure 7: Measurement setup. The target is a Pinata board with an STM32F417 ARM Cortex-M4 core running FALCON-1024. The board is powered by two AA batteries. A current probe placed in series with the supply line measures the power consumption, which is captured by a PicoScope 3000-series oscilloscope.

single-trace attack, and show that the completion step turns it into full recovery of the secret key, together with the time the recovery takes.

5.1 Attack Environment and Equipment

Figure 7 shows the experimental setup. The target device is a Pinata development board featuring an ARM Cortex-M4 processor running at 168 MHz. We executed the `m4f` implementation of FALCON-1024 from the `pqm4`project [21], compiled with `arm-none-eabi-gcc` 12.2.1 using the `-Os` optimization flag. To reduce supply noise, the board was powered by two AA batteries instead of a regulated bench supply. Power consumption was measured using a current probe inserted in series with the supply line and digitized by a PicoScope 3000-series oscilloscope at 1 GS/s.

In our experiments, a GPIO trigger marked the start of the signing routine and synchronized the oscilloscope with the target device. The trigger serves only as a convenient synchronization aid and does not reveal any secret information. Since decoding is always performed at the beginning of the signing operation, an attacker can identify the corresponding region directly from the power trace without relying on the trigger. Each trace contains one complete signing operation. The attack uses only the secret key decoding region, which contains one leakage window for each coefficient of f and g. Template construction and key recovery are performed offline on a laptop equipped with an Intel Core Ultra 5 235U processor and 16 GB of RAM.

5.2 Quantifying the Success Rate

We first evaluate how accurately the attack recovers a single coefficient. A profiling observation is the leakage window of one coefficient, extracted from a signing trace captured under a known key and grouped by the value that coefficient takes. We collect 800 observations for each of the 31 possible values, 24,800 in total, and build the templates from them. We then evaluate on 3100 further observations, 100 for each value, extracted from signing traces captured under fresh keys disjoint from the profiling set. Over this

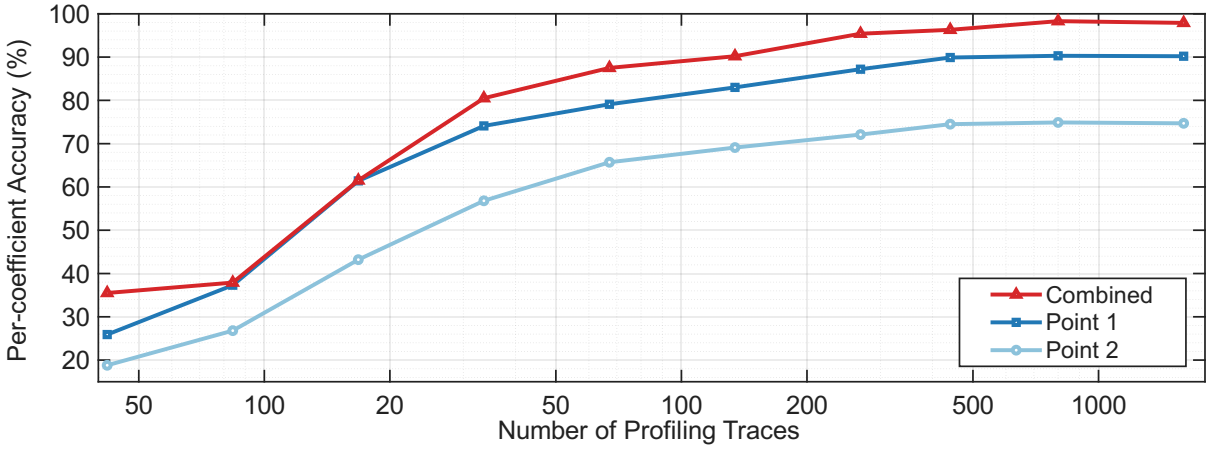

Figure 8: Per-coefficient success rate against the number of profiling observations per coefficient value, for Point 1 alone, Point 2 alone, and their combination. Each curve rises as more profiling observations are used, and the combined score stays above either single point across the range.

Table 2: Per-coefficient success rate, expected number of wrong coefficients per key, probability of an error-free read, and full key-recovery rate for FALCON-1024.

Metric	Value
Per-coefficient success rate	98.32%
Expected wrong coefficients per key	34.41
Probability of an error-free read	8.52×10^{-16}
Full-key recovery rate	100%

test set, the correct coefficient value is recovered in 98.32% of the cases. Figure 8 reports this accuracy as a function of the number of profiling observations per value, for Point 1 alone, Point 2 alone, and their combination. Individually, Point 1 and Point 2 each carry only partial information about the coefficient value and classify less accurately than their combination. Combining the two raises the success rate to 98.32%, because values that are hard to separate at one point are usually separated by the other. We used 800 observations per value for the reported results. Figure 8 shows the accuracy at the profiling budgets we evaluated and does not establish a minimum budget required for the attack.

Although the per-coefficient accuracy reaches 98.32%, it is still insufficient for direct key-recovery. For FALCON-1024, the two targeted secret polynomials, f and g, each contain $n = 1024$ coefficients, giving a total of $2n = 2048$ coefficients. Thus, an average coefficient error rate of 1.68% translates to roughly 34 incorrect predictions per key. Assuming independent coefficient errors, the probability of recovering an entire key without a single error is only 8.52×10^{-16}, making direct key extraction from the trace infeasible. The completion step bridges this gap. It ranks all $2n$ coefficient predictions by confidence, trusts the most reliable ones, and uses the public key relation to solve for the remaining coefficients. We evaluate the complete attack on 1000 independently generated keys, separate from those used during profiling and testing. For each key, we capture a single physical power trace during signing and apply the trained templates to extract coefficient predictions and their corresponding confidence scores. These predictions are then passed to the key-completion procedure to reconstruct the remaining coefficients. As shown in Table 2, the attack successfully recovers the full secret key from a single power trace for all 1000 evaluated keys, achieving a success rate of 100% by correcting prediction errors rather than eliminating them.

Table 3 reports the computational cost of this completion process, as measured on the laptop described in Section 5.1. In the best case, a key is recovered after a single linear solve and requires only 0.485 s,

Table 3: Key-completion cost over 1000 FALCON-1024 keys.

	Best	Average	Worst
Linear solves per key	1	22.78	197
Recovery time (s)	0.485	11.04	95.54

which occurs when the initial trusted set contains no incorrect predictions. Across all 1000 keys, recovery requires an average of 22.78 linear solves and 11.04 s. Additional solves are needed when a highly ranked but incorrect prediction enters the trusted set and must be replaced after candidate verification fails. In the worst case, a key required 197 solves and 95.54 s before the correct trusted set was identified. Nevertheless, each attempt involves only a linear solve rather than a lattice-reduction step, keeping even the worst-case recovery time below two minutes.

6. Discussion

In this section, we discuss where our attack applies beyond the evaluated target, the countermeasures that could stop it, the effect of the measurement conditions on its success rate, and the limitations of our evaluation.

6.1 Applicability to Other Implementations

Our evaluation targets the m4f build of FALCON-1024, but the vulnerability is not specific to it. The decoding routine `trim_i8_decode` is plain C and is identical in the reference implementation and the m4f build, so Point 1 arises in both implementations. The modular conversion differs between them: the reference C negates each coefficient, while the m4f build adds the modulus with the packed `sadd16` instruction [21]. Each of these two routines still produces a value that depends on the coefficient in both implementations as well. Only the exact intermediate, and hence the templates built for it, changes from one to the other.

The attack also carries over to FALCON-512, with a trade-off. FALCON-512 stores each coefficient in 6 bits over the range $[-31, 31]$, so the classifier must separate 63 values rather than 31. More values are harder to tell apart and measuring it would require profiling the 63 classes on that target, which we have not done. The completion step, on the other hand, is lighter: a FALCON-512 key has 1024 coefficients instead of 2048, so the linear system it solves is half the size. We therefore expect the attack to apply to FALCON-512, but we report a measured success rate only for FALCON-1024.

6.2 Calibration Factors of the Experiments

The noise of the platform increases with operating frequency. Prior single-trace attacks on lattice-based schemes reduced the target clock frequency to improve the signal-to-noise ratio [23]. In contrast, we evaluated FALCON at its nominal frequency of 168 MHz. To reduce measurement noise, the board was powered by batteries and monitored with a current probe. As a result, our setup more closely reflects a realistic deployment scenario than previous works.

The completion procedure provides robustness against occasional coefficient misclassifications. Since only n of the $2n$ coefficient predictions must be trusted, the attack can tolerate some prediction errors while still recovering the full key. In practice, lower accuracy increases the number of completion attempts rather than causing immediate failure. We evaluated the attack only at the target's nominal clock frequency and leave an investigation under noisier measurement conditions to future work.

6.3 Limitations of Our Evaluation

Our evaluation shares the limitations common to template attacks. The profiling and attack traces are collected from the same device, whereas a practical cross-device attack must cope with variations between the profiling and target devices [11]. Addressing these variations may require more sophisticated profiling

techniques, potentially involving machine-learning-based models [3,6,31]. This limitation is not unique to our work and is shared by prior single-trace attacks on lattice-based schemes [28,41,42,44]. Developing reliable cross-device attacks therefore remains an important open problem.

The attack also relies on a substantial profiling phase. We collect 24,800 profiling observations, 800 for each of the 31 possible coefficient values. Figure 8 reports the per-coefficient accuracy at the profiling budgets we evaluated. This budget was enough for the results we report, and we do not present it as a minimum requirement. An attacker with fewer profiling observations would obtain a lower classification accuracy. The completion procedure provides some robustness against such errors, because a reduced coefficient accuracy mainly increases the number of completion attempts before the key is recovered rather than causing an immediate failure.

6.4 Countermeasures

Our attack targets the unmasked secret key decoding of the evaluated implementation. Masking the later signing computations does not protect leakage that occurs during decoding, so the protection must also cover the loading and decoding of the secret key. The masked FALCON implementations proposed so far protect the sampler and the floating-point arithmetic and leave the decoding unprotected [8, 25]. The single-trace property of our attack is a measurement requirement, not a mechanism for defeating masking.

7. Conclusion

We presented the first single-trace side-channel attack on FALCON's secret key decoding, the step that unpacks the secret polynomials before any other computation. The decoding leaks each coefficient of f and g through the Hamming weight of two intermediate values. A profiled attack reads these coefficients with 98.32% accuracy, and a completion step based on the public-key relation recovers the full key with linear algebra alone, without lattice reduction. In our experiments it recovered all 1000 FALCON-1024 keys, each from a single trace, in 11.04 seconds on average on an off-the-shelf ARM Cortex-M4. The vulnerability is orthogonal to the prior single-trace attacks on FALCON, and the countermeasures proposed for the sampler and the floating-point arithmetic do not cover the decoding. A protected implementation must therefore extend its countermeasures to the decoding itself.

References

[1] Amund Askeland and Sondre Rønjom. A side-channel assisted attack on NTRU. Cryptology ePrint Archive, Paper 2021/790, 2021.

[2] Aydin Aysu, Youssef Tobah, Mohit Tiwari, Andreas Gerstlauer, and Michael Orshansky. Horizontal side-channel vulnerabilities of post-quantum key exchange protocols. In *2018 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 81–88, 2018.

[3] Ryad Benadjila, Emmanuel Prouff, Rémi Strullu, Eleonora Cagli, and Cécile Dumas. Deep learning for side-channel analysis and introduction to the ASCAD database. *Journal of Cryptographic Engineering*, 10(2):163–188, 2020.

[4] Estuardo Alpirez Bock, Gustavo Banegas, Chris Brzuska, Łukasz Chmielewski, Kirthivaasan Puniamurthy, and Milan Šorf. Breaking DPA-protected Kyber via the pair-pointwise multiplication. In *Applied Cryptography and Network Security (ACNS 2024)*, Lecture Notes in Computer Science. Springer, 2024. Cryptology ePrint Archive, Paper 2023/551.

[5] Marcus Brinkmann, Nicolai Kraus, and Alexander May. Halfspace learning for lattice signature key recovery from signs. Cryptology ePrint Archive, Paper 2026/1366, 2026.

[6] 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.

[7] 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.

[8] 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.

[9] Keng-Yu Chen, Ming Qing Ching, Jiun-Peng Chen, and Bo-Yin Yang. When masking multiplication isn't enough: Exploiting floating-point leakage in falcon's pre-image computation. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2026(2):686–708, 2026.

[10] Zhaohui Chen, Emre Karabulut, Aydin Aysu, Yuan Ma, and Jiwu Jing. An efficient non-profiled side-channel attack on the CRYSTALS-dilithium post-quantum signature. In *IEEE International Conference on Computer Design (ICCD)*, pages 583–590, 2021.

[11] Omar Choudary and Markus G. Kuhn. Template attacks on different devices. In Emmanuel Prouff, editor, *Constructive Side-Channel Analysis and Secure Design*, pages 179–198, Cham, 2014. Springer International Publishing.

[12] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[13] Pierre-Alain Fouque, Damien Marion, Quyen Nguyen, and Alexandre Wallet. Avengers assemble! supervised learning meets lattice reduction: A single power trace attack against CRYSTALS-kyber key generation. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2025(4):409–436, 2025.

[14] Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Prest, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. FALCON: Fast-Fourier lattice-based compact signatures over NTRU, specification v1.2. Technical Report v1.2, École normale supérieure, Paris, October 2020.

[15] Benedikt Gierlichs, Kerstin Lemke-Rust, and Christof Paar. Templates vs. stochastic methods. In Louis Goubin and Mitsuru Matsui, editors, *Cryptographic Hardware and Embedded Systems - CHES 2006*, pages 15–29, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[16] Morgane Guerreau, Ange Martinelli, Thomas Ricosset, and Mélissa Rossi. The hidden parallelepiped is back again: Power analysis attacks on falcon. Cryptology ePrint Archive, Paper 2022/057, 2022.

[17] Morgane Guerreau and Mélissa Rossi. A not so discrete sampler: Power analysis attacks on HAWK signature scheme. Cryptology ePrint Archive, Paper 2024/1248, 2024.

[18] Jaeseung Han, Taeho Lee, Jihoon Kwon, Joohee Lee, Il-Ju Kim, Jihoon Cho, Dong-Guk Han, and Bo-Yeon Sim. Single-trace attack on NIST round 3 candidate dilithium using machine learning-based profiling. *IEEE Access*, 9:166283–166292, 2021.

[19] Fan Huang, Xiaolin Duan, Chengcong Hu, Mengce Zheng, and Honggang Hu. MulLeak: Exploiting multiply instruction leakage to attack the stack-optimized Kyber implementation on Cortex-M4. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2025(2):23–68, 2025.

[20] Vahid Jahandideh. Too small to hide: Single-trace key recovery from ML-KEM key generation. Cryptology ePrint Archive, Paper 2026/2137, 2026.

[21] Matthias J. Kannwischer et al. pqm4: Post-quantum cryptography for the arm cortex-m4. https://github.com/mupq/pqm4. Accessed: October 2026.

[22] Matthias J. Kannwischer, Peter Pessl, and Robert Primas. Single-trace attacks on keccak. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020(3):243–268, 2020.

[23] Emre Karabulut, Erdem Alkim, and Aydin Aysu. Single-trace side-channel attacks on omega-small polynomial sampling: with applications to NTRU, NTRU prime, and CRYSTALS-dilithium. Cryptology ePrint Archive, Paper 2022/494, 2022.

[24] Emre Karabulut and Aydin Aysu. Falcon down: Breaking falcon post-quantum signature scheme through side-channel attacks. Cryptology ePrint Archive, Paper 2021/772, 2021.

[25] Emre Karabulut and Aydin Aysu. Masking Falcon's floating-point multiplication in hardware. *IACR Transactionnd Embedded Systems*, 2024(4):483–508, 2024.

[26] GyuSang Kim, JeongHwan Lee, Myeonghoon Lee, Seokhie Hong, and HeeSeok Kim. Secret key recovery of FALCON using simple power analysis in conditional calculator. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2026(2):953–975, 2026.

[27] Neal Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, 48(177):203–209, 1987.

[28] Kang Li, Shouran Ma, Haochen Dou, and Qian Guo. One fell swoop: A single-trace key-recovery attack on the falcon signing algorithm. Cryptology ePrint Archive, Paper 2025/2159, 2025.

[29] Xiuhan Lin, Mehdi Tibouchi, and Yang Yu. Swing the lure: How to cheaply mitigate sign leakage in falcon. Cryptology ePrint Archive, Paper 2026/2170, 2026.

[30] 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. Cryptology ePrint Archive, Paper 2025/351, 2025.

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

[32] Sarah McCarthy, James Howe, Neil Smyth, Seamus Brannigan, and Máire O'Neill. BEARZ attack FALCON: Implementation attacks with countermeasures on the FALCON signature scheme. Cryptology ePrint Archive, Paper 2019/478, 2019.

[33] Carlos Aguilar Melchor, Nicolas Aragon, Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Jurjen Bos, Jean-Christophe Deneuville, Arnaud Dion, Philippe Gaborit, Jérôme Lacan, Edoardo Persichetti, Jean-Marc Robert, Pascal Véron, and Gilles Zémor. Hamming quasi-cyclic (hqc): Fourth round specification. Technical report, NIST Post-Quantum Cryptography Standardization Project, February 2025. Fourth Round Version, updated on 19 February 2025.

[34] Dustin Moody. Post-quantum cryptography standardization: Announcement and outline of NIST's call for submissions. In *International Conference on Post-Quantum Cryptography (PQCrypto)*, 2016.

[35] National Institute of Standards and Technology. FIPS 203: Module-lattice-based key-encapsulation mechanism standard. Technical report, U.S. Department of Commerce, Aug. 2024. Accessed: 2025-06-08.

[36] National Institute of Standards and Technology. FIPS 204: Module-lattice-based digital signature standard. Technical report, U.S. Department of Commerce, Aug. 2024. Accessed: 2024-11-16.

[37] National Institute of Standards and Technology. FIPS 205: Stateless hash-based digital signature standard. Technical report, U.S. Department of Commerce, Aug. 2024. Accessed: 2025-06-08.

[38] National Institute of Standards and Technology. Module-Lattice-Based Digital Signature Standard (ML-DSA). FIPS PUB 206, 2024.

[39] Kalle Ngo, Elena Dubrova, Qian Guo, and Thomas Johansson. A side-channel attack on a masked IND-CCA secure saber KEM implementation. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(4):676–707, 2021.

[40] Jaesang Noh, Hyunseo Choi, Dongwoo Han, and Dong-Joon Shin. Efficient error-tolerant side-channel attacks on GPV signatures based on ordinary least squares regression. Cryptology ePrint Archive, Paper 2024/2043, 2024.

[41] Peter Pessl and Robert Primas. More practical single-trace attacks on the number theoretic transform. Cryptology ePrint Archive, Paper 2019/795, 2019.

[42] Robert Primas, Peter Pessl, and Stefan Mangard. Single-trace side-channel attacks on masked lattice-based encryption. Cryptology ePrint Archive, Paper 2017/594, 2017.

[43] 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, Paper 2024/512, 2024.

[44] Jiahao Qiu and Aydin Aysu. Ntt-peel: Bit shift side-channel in falcon's number theoretic transform. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2026(2):709–735, 2026.

[45] Jinyi Qiu. Single trace side-channel vulnerabilities discovery using statistical leakage simulator. Cryptology ePrint Archive, Paper 2025/319, 2025.

[46] Jinyi Qiu and Aydin Aysu. Shift snare: Uncovering secret keys in falcon via single-trace analysis, 2025.

[47] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, Feb. 1978.

[48] Vladimir Sarde and Nicolas Debande. Improvement of side-channel attacks on mitaka. In *CASCADE*, 2025.

[49] Honglin Shao, Yuejun Liu, Mingyao Shao, Wei Cheng, and Yongbin Zhou. NTRU with hints: Secret key recovery under partial leakage on NTRU-based signatures. Cryptology ePrint Archive, Paper 2025/1229, 2025.

[50] P. W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 124–134, 1994.

[51] Yiteng Sun, Zhuo Huang, Yan Zhuang, Shuo Sun, Xinyu Li, Yu Yu, and Weijia Wang. Efficient soft analytical side-channel attacks on large-scale cryptographic computations. Cryptology ePrint Archive, Paper 2026/1811, 2026.

[52] Ruize Wang, Kalle Ngo, Joel Gärtner, and Elena Dubrova. Unpacking needs protection: A single-trace secret key recovery attack on dilithium. *IACR Communications in Cryptology*, 1(3), 2024.

[53] Shiduo Zhang, Xiuhan Lin, Yang Yu, and Weijia Wang. Improved power analysis attacks on falcon. Cryptology ePrint Archive, Paper 2023/224, 2023.

[54] Yuanyuan Zhou, Weijia Wang, Yiteng Sun, and Yu Yu. Every signing leaks: Breaking falcon via floating-point conversion leakage. Cryptology ePrint Archive, Paper 2026/2124, 2026.