A Not So Discrete Sampler: Power Analysis Attacks on HAWK signature scheme

Morgane Guerreau[1*] and Mélissa Rossi[2]

[1] CryptoNext Security, Paris, France
morgane.guerreau@cryptonext-security.com
[2] ANSSI, Paris, France
melissa.rossi@ssi.gouv.fr

Abstract. HAWK is a lattice-based signature scheme candidate to the fourth call of the NIST's Post-Quantum standardization campaign. Considered as a cousin of Falcon (one of the future NIST post-quantum standards) one can wonder whether HAWK shares the same drawbacks as Falcon in terms of side-channel attacks. Indeed, Falcon signature algorithm and particularly its Gaussian sampler, has shown to be highly vulnerable to power-analysis attacks. Besides, efficiently protecting Falcon's signature algorithm against these attacks seems a very challenging task.
This work presents the first power analysis leakage review on HAWK signature scheme: it extensively assesses the vulnerabilities of a central and sensitive brick of the scheme, the discrete Gaussian sampler. Knowing the output x of the sampler for a given signature leads to linear information about the private key of the scheme.
This paper includes several demonstrations of simple power analysis attacks targeting this sample x with various attacker strengths, all of them performed on the reference implementation on a ChipWhisperer Lite with STM32F3 target (ARM Cortex M4). We report being able to perform key recoveries with very low (to no) offline resources. As this reference implementation of HAWK is not claimed to be protected against side-channel attacks, the existence of such attacks is not surprising, but they still concretely warn about the use of this unprotected signature on physical devices.
To finish, we propose an ad-hoc technique to lower down the leakage on the identified vulnerability points. These modifications prevent our attacks on our platform and come with essentially no cost in terms of performance. It could be seen as a temporary solution and encourages more analysis on proven side-channel protection of HAWK like masking.

Keywords: Side-channel attack · HAWK signature scheme · Residual complexity

1 Introduction

Structured lattice-based signature schemes are promising solutions for post-quantum protocols. The study of their performance and security has accelerated in the last decade with the NIST call for post-quantum algorithms and the second NIST 2023 call that specifically aims at signature schemes. These schemes can be divided into two families: the signatures following the Hash-and-Sign framework [GPV08] and the ones using the Fiat–Shamir transform [Lyu09, Lyu12]. Now, concrete schemes in both families are available and among them, two future NIST post-quantum standards: Falcon [PFH+22] and Crystals-Dilithium [LDK+22]. Other candidate signature schemes may join these two finalists in the future as the standardization campaign is still ongoing.

*Work done while the author was at Thales – Gennevilliers, France.

While still slightly moving, the black-box security analysis of existing lattice-based signature algorithms has been theoretically documented with many convenient evaluation tools e.g. [ADPS16, ACD+18]. The grey-box or side-channel security analysis of these schemes has been part of a second step analysis. But, seven years after NIST's first round opening, the side-channel resistance of the current future standards starts to be well furnished. For Crystals-Dilithium, we can cite the following recent attack examples [MUTS22, EAB+23, BVC+23]. For Falcon, the main target of the side-channel attacks has been its core and critical part: the Gaussian sampler. This block has shown to be vulnerable to several types of side-channel attacks. A theoretical timing attack [FKT+20] has highlighted the need for efficient countermeasures which are now implemented in the reference implementation of Falcon [PFH+22]. Concerning the resistance against other types of side-channel attacks such as power analysis, several key recoveries were published recently [KH18, GMRR22, ZLYW23]. While provable countermeasures like masking are costly but achievable for Crystals-Dilithium [ABC+23, CGTZ23], efficient provable countermeasures are very challenging to implement for Falcon. The Gaussian sampler is again the difficult part. An attempt to modify this sampler to make it easier to protect [EFG+22] has been invalidated [Pre23]. Hence, efficiently protecting Falcon against physical attack remains an open issue, preventing the use of this signature for physical applications.

Hawk [BBD+23] is a new post-quantum signature scheme that has been submitted to the second NIST call for post-quantum signatures. It presents similarities with Falcon. The main (and non-negligible) difference is the use of a new structured lattice-based problem: the Lattice Isomorphism Problem (LIP). This problem has been introduced in [HR14, Dv22] and its use as an underlying problem for designing public key cryptography is relatively new [DPPv22]. Hawk is a signature scheme based on [DPPv22] with a module variant of LIP. Such a new problem allows to use a quadratic form as a public key and leads to very interesting performance results, bypassing the already very compact and efficient Falcon. On the other hand, the black-box security of Hawk's underlying problem may still be fluctuating due to its relative youth compared to other lattice-based problems. Hence, more theoretical analysis on this problem is still needed and several papers will probably argument on the security (or vulnerability) of this problem in the future. On the side-channel analysis side, whether Hawk shares the same drawbacks as Falcon in terms of physical vulnerabilities is a natural question. We think that anticipating side-channel leakages on new schemes is good practice[1].

Inherited from the Hash-and-Sign structure shared with Falcon, Hawk uses a discrete Gaussian sampler. As for Falcon, it is expectedly the most complex and critical step of the signature scheme in terms of side-channel leakage. The goal of this sampler is to derive a small sample $\mathbf{x}$ that will be multiplied by the inverse of the private matrix $\mathbf{B}$ in order to provide a signature. More technical details will be proposed later on in the paper.

Contributions. The goal of the paper is to open the sampler box, analyze the reference code and assess all the possible side-channel vulnerabilities of this sampler leading to key recoveries. We hope that this work will encourage more analysis on proven side-channel protections of Hawk. More precisely, we propose the following contributions:

- We first describe several power-analysis attacks targeting the output sample $\mathbf{x}$ with various attacker strengths, all of them performed on the reference implementation on a ChipWhisperer Lite with STM32F3 target (ARM Cortex M4). Our power measurement traces and attack code are available online for easy reproducibility. We

[1] This could even be a lesson learned from the analysis of Crystals-Dilithium and Falcon. Since the focus on the side-channel security arrived after a long period of black-box analysis, modifying the schemes to protect them against physical attacks was somehow prevented. This partly explains why some other side-channel friendly variant schemes like Raccoon [dPEK+23] emerged in the second NIST call.

detail all the leakage points in the sampling algorithm and propose three levels of attack. The first attack is a simple power analysis (SPA), meaning that it uses only one trace. It also does not use any template (*i.e.* it does not assume a learning phase on traces with known secret key). This attack analyzes the most leaky instructions. It is able to deduce some information about $\mathbf{x}$ like the parity and sign of its coefficients but does not directly lead to a key recovery. In our second attack, we go a bit deeper in the analysis, still with only one trace and without templates, this analysis allows to recover almost all $\mathbf{x}$. This can lead to a complete key recovery with a small residual offline work (e.g. 2^{55} remaining elementary operations for Hawk-1024). Finally, we present a stronger profiled simple power analysis with a light template, leading to a full knowledge of $\mathbf{x}$ and thus a direct key recovery.
The success of such attacks is of course not groundbreaking as the authors of Hawk do not claim a protection of their implementation against these power analysis attacks. However, we report and detail a large amount points of leakage and very low resource adversaries. Hence, it can be seen as a concrete warning on the use of Hawk for physical devices, and it is a motivation for side-channel protection of such instructions.

- In a second step, we discuss the possibility of going further and introducing a generic way of assessing the performance of a side-channel attack on $\mathbf{x}$ even when less information is recovered. And, we propose a direct brute-force oriented, key-recovery method when a lot of information about $\mathbf{x}$ is recovered.

- Finally, we propose an ad-hoc technique to lower down the leakage on identified vulnerability points. These simple modifications prevent all of our attacks on our platform. It comes with almost no cost in terms of performance. While it is not a provable solution, it could be seen as a temporary fix.

Acknowledgments. We warmly thank Gaspard Billaud and Hugo Béguinet for their contributions to this work, and the anonymous reviewers for their suggestions and corrections that helped to improve the quality of this paper.

Structure of the paper. In Section 2, we outline the necessary linear algebra notions and a simplified description of Hawk signature scheme. In Section 3, we extensively detail the specific targeted instructions of the discrete Gaussian sampler and our different single-trace side-channel attacks with their success results. In Section 4, we present an open problem related to generic key recovery when partial information is recovered on the sampled value. In Section 5, we detail a particular key recovery brute force-based technique that does not use lattice reduction for the particular case when the sample is almost entirely recovered. This key recovery is detailed for one of our attacks in Section 3 for Hawk-1024. Finally, we present our code modification techniques to avoid obvious leakages in Section 6.

2 Preliminaries

2.1 Algebra

Notations. Vectors will be denoted with bold lowercase letters and matrices with bold uppercase letters. We denote by $\mathbf{b}_i$ the i-th row of a matrix $\mathbf{B}$. We denote by $\|\mathbf{v}\|$ the Euclidean norm L_2 of a vector $\mathbf{v}$ and by $\langle \mathbf{u}, \mathbf{v} \rangle$ its associated inner product with a second vector $\mathbf{u}$.

Number fields. Let n be a power of two, $K_n = \mathbb{Q}[X]/(X^n + 1)$ a cyclotomic number field and its ring of integers $R_n = \mathbb{Z}[X]/(X^n + 1)$. For a polynomial $f \in K_n$, let $f = f_0 + f_1 X + \cdots + f_{n-1}X^{n-1}$ and define a bijection that maps it to the column vector of its coefficients $\text{vec} : K_n \to \mathbb{Q}^n, f \to (f_0, \ldots, f_{n-1})$. The elements of K_n and R_N will be equivalently represented as polynomials or vectors of coefficients.

Negacyclic matrix. We call the negacyclic matrix of $f \in K_n$ the matrix

$$\text{nrot}(f) := (\text{vec}(f), \text{vec}(Xf) \ldots, \text{vec}(X^{n-1}f)) = \begin{pmatrix} f_0 & \cdots & f_{n-1} \\ \vdots & \ddots & \vdots \\ -f_{n-1} & \cdots & f_0 \end{pmatrix}$$

Hermitian adjoint and Gram matrix. We denote the Hermitian adjoint of an element $f \in K_n$ by $f^* = f_0 - \sum_{i=1}^{n-1} f_i X^i$. We extend the notations to elements $\mathbf{B} \in K_n^{r\times r}$: $\mathbf{B}^*$ denotes the entry-wise adjoint of $\mathbf{B}^{\mathrm{T}}$, the transpose of $\mathbf{B}$. For any element $\mathbf{B} \in K_n^{r\times r}$, we call the Gram matrix of $\mathbf{B}$ the matrix $\mathbf{Q} = \mathbf{B}^*\mathbf{B}$.

Unimodular matrix. Given $n \in \mathbb{N}$, $\mathbf{U} \in \mathbb{Z}^{n\times n}$ is an unimodular matrix if $\det(\mathbf{U}) = \pm 1$.

Q-inner product and Q-norm. For a matrix $\mathbf{B} \in K_n^{r\times r}$, let $\mathbf{Q} = \mathbf{B}^*\mathbf{B}$. We define the $\mathbf{Q}$-inner product as $\langle \cdot, \cdot \rangle_{\mathbf{Q}} : K_n^r \times K_n^r \to \mathbb{Q}, (\mathbf{f}, \mathbf{g}) \to \frac{1}{n}\text{Tr}(\mathbf{f}^*\mathbf{Q}\mathbf{g})$ with $\text{Tr}(\cdot)$ being the algebraic trace over the field extension $K_n/\mathbb{Q}$. Similarly, we define the $\mathbf{Q}$-norm of an element $\mathbf{f} \in K_n^r$ as $\|\mathbf{f}\|_{\mathbf{Q}} = \sqrt{\langle \mathbf{f}, \mathbf{f} \rangle_{\mathbf{Q}}}$. In particular, we have the following identity: $\|\mathbf{B} \cdot \mathbf{f}\| = \|\mathbf{f}\|_{\mathbf{Q}}$.

2.2 HAWK signature scheme

We describe here the high-level idea behind Hawk. For further details on the signature scheme, we refer to the official specification [BBD+23].

Hawk keypair. Let $n \in \{256, 512, 1024\}$ be a fixed parameter. The Hawk private key consists in four small polynomials $f, g, F, G \in R_n$ which satisfy the equation $fG - gF = 1 \bmod X^n + 1$. The polynomials f and g are sampled according to a centered binomial distribution with $\eta = n/128$ and are used to find F and G to verify the previous equation. Lastly, the keypair (sk, pk) is defined as $sk \coloneqq \mathbf{B}$ and $pk \coloneqq \mathbf{Q}$. The matrices $\mathbf{B}$ and $\mathbf{Q}$ are computed as

$$\mathbf{B} := \begin{pmatrix} f & F \\ g & G \end{pmatrix}, \mathbf{Q} := \mathbf{B}^* \cdot \mathbf{B}$$

and by construction $\mathbf{B}$ is unimodular while $\mathbf{Q}$ is a Gram matrix. Note that both are blockwise negacyclic matrices (*i.e.* $\mathbf{B}, \mathbf{Q} \in R_n^{2\times 2}$).

Signature overview. At a very high level, Hawk signature algorithm can be summarized as follows:

- Hash the message to a vector $\mathbf{h} \in R_n^2$.
- Compute a binary target $\mathbf{t} = \mathbf{B}\mathbf{h} \mod 2$ using the private key $\mathbf{B}$.
- Sample a vector $\mathbf{x}$ from a distribution that approximates a Gaussian on $2\mathbb{Z}^{2n} + \mathbf{t}$ with a standard deviation of $2\sigma_{\text{sign}}$ where σ_{sign} is a fixed parameter. Hence, $\|\mathbf{x}\|$ is small.
- Compute $\mathbf{w} = \mathbf{B}^{-1}\mathbf{x}$ and then the signature $\mathbf{s} = \frac{1}{2}(\mathbf{h} - \mathbf{w})$.

To verify a signature, $\mathbf{w}$ is recomputed from public data ($\mathbf{s}$ and $\mathbf{h}$) and $\|\mathbf{w}\|_{\mathbf{Q}} = \|\mathbf{x}\|$ is checked to be short enough. Here, we want to emphasize that, although $\mathbf{x}$ is a secret data, its L_2-norm is public, which in itself is not a vulnerability.

Discrete sampling. To sample a short vector $\mathbf{x}$, the signature generation uses a discrete Gaussian sampler algorithm called `SamplerSign`. The center is the target $\mathbf{t}$ and the standard deviation is fixed for each variant of Hawk. Each coefficient $\mathbf{x}[i]$ is sampled separately, with center being the corresponding $\mathbf{t}[i]$. The algorithm uses two Reverse Cumulative Distribution Tables (RCDTs), depending on the value of the current $\mathbf{t}[i]$ (either 0 or 1). A uniform 78-bit integer is generated and compared to each entry of both RCDTs. A counter is incremented if the random integer is smaller than the current entry in the corresponding RCDT. Once the integer is greater than a table entry it will be greater than all the following table entries, but both tables are fully parsed to avoid timing attacks. In the end, a sign bit is drawn uniformly and applied to each $\mathbf{x}[i]$.

3 Side-channel analysis of the discrete sampler

The central building block of Hawk signature generation consists in the discrete Gaussian sampler `SamplerSign` (called in the third step of the signature presented in Section 2). For efficiency purpose, this sampler relies on two distribution tables. Such duplication is quite unusual for table-based samplers. However, it has no impact from a side-channel attack point of view, since a simple power analysis allows us to target indifferently one or two consecutive table look-ups.

Remark 1. By looking at this high level description of the Hawk signature algorithm, the second step could also appear as a straightforward target for a side-channel attack. However, the secret polynomials composing $\mathbf{B}$ are reduced modulo 2 before the actual multiplication with $\mathbf{h}$. Hence, one could only recover very limited information by targeting this operation.

The goal of this section is to open the `SamplerSign` box in order to extract as much information as possible.

Targeted sensitive value. Two intermediate values are involved in the sampler: the center $\mathbf{t}$ and the sample $\mathbf{x}$ which is also the output of the sampler. Both values depend on the secret matrix $\mathbf{B}$ with linear equations. On the one hand, $\mathbf{t} = \mathbf{Bh} \mod 2$ where $\mathbf{h}$ can be publicly recomputed. On the other hand, $\mathbf{x} = \mathbf{Bw}$ where $\mathbf{w}$ can also be recomputed from $\mathbf{s}$ and $\mathbf{h}$. Let us note that the first linear equation is a particular case of the second: since $\mathbf{x} \in 2\mathbb{Z}^{2n} + \mathbf{t}$, $\mathbf{x} \equiv \mathbf{t} = \mathbf{Bh} \mod 2$. Thus, we will concentrate our efforts on recovering as much as we can from the vector $\mathbf{x}$.

In a perfectly favorable case where the full exact value of $\mathbf{x}$ can be recovered with a side-channel analysis, the secret matrix $\mathbf{B}$ can be deduced from two signatures and their associated samples $\mathbf{x}$ and $\mathbf{x}'$ with high probability (given than $\mathbf{w}$ and $\mathbf{w}'$ are linearly independent). The vectors $\mathbf{x}$ and $\mathbf{x}'$ being in R_n^2, we separate their coordinates in R_n by noting $\mathbf{x} = (x_0, x_1)$ and $\mathbf{x}' = (x'_0, x'_1)$. Hence, we can write

$$\begin{pmatrix} f & F \\ g & G \end{pmatrix} = \begin{pmatrix} x_0 & x'_0 \\ x_1 & x'_1 \end{pmatrix} \cdot \begin{pmatrix} w_0 & w'_0 \\ w_1 & w'_1 \end{pmatrix}^{-1}. \tag{1}$$

In practice, such a favorable case is unlikely, and the recovered information may only concern the sign or parity of the sample $\mathbf{x}$. In this section, we will detail the leaking instructions inside the sampling algorithm and extract as much information as possible with different single trace analysis techniques (profiled or non-profiled).

```
Inputs:  lab_hi, tab_lo (RCDTs), v (index), t (target), hi, lo (randomness)
Ouputs:  x[v] (sample for index v), sn (squared norm of x)

uint32_t neg = -(uint32_t)(lo >> 63);                                  △
uint32_t pbit = (t[v >> 3] >> (v & 7)) & 1;
uint64_t p_odd = -(uint64_t)pbit;                                      ◯
uint32_t p_oddw = (uint32_t)p_odd;
uint32_t r = 0;
for (size_t i = 0; i < hi_len; i += 2) {
    uint64_t tlo0 = tab_lo[i + 0];
    uint64_t tlo1 = tab_lo[i + 1];
    uint64_t tlo = tlo0 ^ (p_odd & (tlo0 ^ tlo1));                     ◯
    uint32_t cc = (uint32_t)((lo - tlo) >> 63);                        □
    uint32_t thi0 = tab_hi[i + 0];
    uint32_t thi1 = tab_hi[i + 1];
    uint32_t thi = thi0 ^ (p_oddw & (thi0 ^ thi1));                    ◯
    r += (hi - thi - cc) >> 31;                                        □
}
uint32_t hinz = (hi - 1) >> 31;
for (size_t i = hi_len; i < lo_len; i += 2) {
    uint64_t tlo0 = tab_lo[i + 0];
    uint64_t tlo1 = tab_lo[i + 1];
    uint64_t tlo = tlo0 ^ (p_odd & (tlo0 ^ tlo1));                     ◯
    uint32_t cc = (uint32_t)((lo - tlo) >> 63);                        □
    r += hinz & cc;                                                    □
}
r = (r << 1) - p_oddw;                                                 ◯
r = (r ^ neg) - neg;                                                   △
x[v] = (int8_t)*(int32_t *)&r;
sn += r * r;                                                           □ △
```

Listing 1: *Extract of* `SamplerSign`[2]*, with leakage of the sign (△), parity (◯) and value (□) of* $\mathbf{x}$*. Inputs and outputs are written here to provide some context.*

Targeted code. Listing 1 shows the targeted code, which samples only one coefficient. It is extracted from the `SamplerSign` in the reference implementation, which loops over this code in order to sample all the coefficients in one call. In Line 4, the sign is sampled and stored in a register `neg`. In Line 6, the parity is generated and stored in a register `p_odd`. Next, the code contains two for-loops where a variable `r` is either incremented by 0 or 1 depending on the result of the comparison between a random number `lo` and the corresponding entry in the reverse cumulative distribution table. The final steps in Lines 27, 28 and 30 respectively correspond to the application of the parity, the sign and the computation of the norm. Lines where leakage has been observed have been highlighted, and will be referred to in the following sections.

Experimental setup. The power traces have been recorded with a ChipWhisperer Lite with STM32F3 target (ARM Cortex M4). As the length of recordable power traces is limited by our acquisition device, we only record the sampling of one coefficient at a time, as shown in the Listing 1. Thus, we store one power trace by coefficient. One can easily generalize the attack for all the coefficients by measuring one long power trace running for the whole execution of the `SamplerSign` and divide it into one trace per coefficient, as the coefficients are sampled sequentially in the original code.

Generation of our trace database. In order to assess the accuracy of our simple power analysis (SPA) attacks, 10 000 power traces have been generated (*i.e.* 10 000 coefficients). Such a large number allows to make various accuracy tests for our classifiers. The coefficients have been generated according to the parameters of Hawk-1024. We chose the highest security level to be able to assess as well the theoretical success of the attack on Hawk-256 and Hawk-512 by ignoring the large coefficients. The trace database and attack scripts are available online[3].

Figures. All the figures representing the power consumption always show 20 power traces. Figures can be regenerated from the traces with the provided code for better readability.

Global analysis of an iteration. In Figure 1, we plot twenty power measurements of the execution of Listing 1, each trace corresponding to the sampling of one coefficient. In this figure, starting from sample 93, one can observe 5 times the same repeating pattern corresponding to the traversal of the first distribution table. From sample 838, there is another pattern repeating 8 times, corresponding to the traversal of the second distribution table. Those patterns are really helpful to match quite precisely the power consumption to the corresponding instructions in a non-profiled setting. In the next section, we detail the leakages corresponding to specific points of leakages (also called points of interest). In other words, we identify the samples (in the x-axis of Figure 1) where information about $\mathbf{x}$ can be retrieved.

3.1 A first non-profiled power analysis

3.1.1 Sign recovery

Computation of the sign in Line 4 of Listing 1. The sign of the sampled coefficient is computed at the very beginning of the loop at the first line of Listing 1. The register holding the value of `neg` will be either filled with zeros or with ones, hence having a Hamming weight of either 0 or 32. This is a first point of leakage where the sign can

[2] From https://github.com/hawk-sign/dev/blob/main/src/hawk_sign.c, function `sig_gauss`, commit ac3a98c3107ea030cc18fb2afef7f5655c588138

[3] https://github.com/mguerrea/HawkPowerAnalysis/

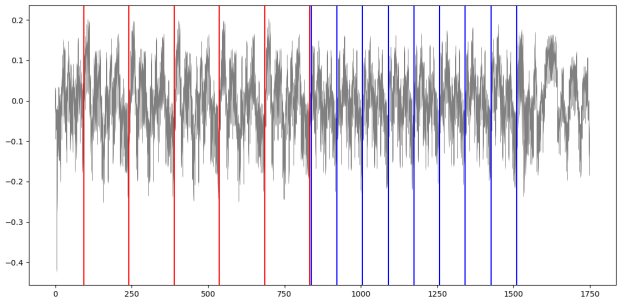
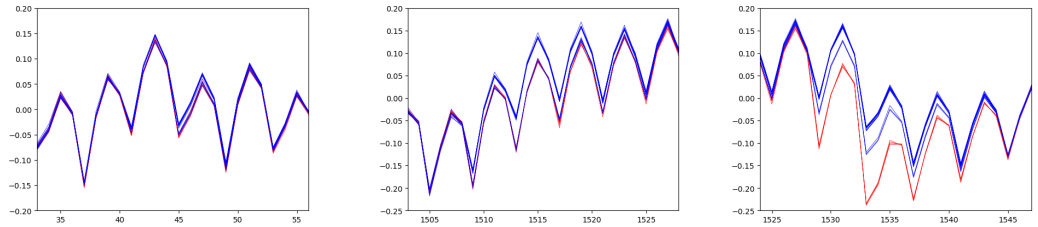


Figure 1: *Power consumption during execution of Listing 1 (20 power traces). Red and blue lines correspond respectively to the iterations of the first (line* 6*) and second (line* 17*) for-loops.*



(a) *Power consumption during the computation of the sign (Line 4).*

(b) *Power consumption during the application of the sign (Line 28).*

(c) *Power consumption during the norm computation (Line 30).*

Figure 2: *Blue (resp. red) traces correspond to coefficients with a positive (resp. negative) sign.*

be retrieved by analyzing the power traces. In Figure 2a, the power consumption of the sign computation for various traces is plotted. It is possible to see a slight difference in consumption between two groups of traces, which does not exactly correspond to the sign of the coefficients. This is due to the fact that, as for all other values, a sign is also drawn for the zeros. Thus, the zero coefficients that one could consider as positive ones, are actually classified alternatively as positive or negative. Hence, some blue traces are in the lower group of traces in Figure 2a, corresponding to the "negative zeros".

Application of the sign in Line 28 of Listing 1. The sign is applied at the end of the sampler, at line 28 on listing 1. In the corresponding Figure 2b, the power traces are very clearly divided into two groups, due to very high leakage in the operation. The leakage is so significant that it can be seen with bare eyes. A simple threshold computed with the mean of the traces is enough to split them into two distinct subsets. Here again, as all other values, the zeros also get a sign applied to them. If the sign is negative, then the zero will be classified as a negative number, because its Hamming weight will greatly increase during the operation. If the sign is positive, then the zero will be classified as a positive number, as its Hamming weight will remain at 0 during the whole operation. Hence, here again our "negative zeros" are in the lower group of traces in Figure 2b.

Computation of the norm in Line 30 of Listing 1. The operation immediately following the sign application is the norm computation. The traces are shown in Figure 2c and will be analyzed further in Section 3.2 but we can already also clearly distinguish the sign.

Using this leakage, we obtain two different but related pieces of information.

- First, we can easily split the power traces into two sets to distinguish between positive and negative coefficients.
- Next, contrary to the sign application analyzed above, all the zeros behave as positive integers, as they indeed have a Hamming value of 0 after the sign application. This allows for the second extraction, as if a coefficient has been classified as negative with the analysis of the sign application and then positive with the norm computation, it should be a zero. Hence, we can clearly identify all the zero coefficients that have previously been classified as negative integers (the so-called "negative zeros") by taking an intersection of two classifications. The number of such zeros is around half of the zero coefficients as there is a probability of exactly 0.5 to get a negative sign.

Recovered information. The sign can be recovered with 100% success rate over our 10 000-coefficient database. Any of the three points of leakage highlighted above is sufficient to recover the drawn sign, but only the norm computation allows us to correctly classify the zeros as positive coefficients. Additionally, the combination of this last point of leakage (Line 30) with the first (Line 4) or second (Line 28) classification allows the identification of half of the coefficients equal to 0 with perfect accuracy. The number of recovered zero coefficients is around 7% for Hawk-1024 of the total number of coefficients, and would be around 8% for Hawk-512 and 9% for Hawk-256. Those percentages correspond to half of the probability of 0 in the distribution of $\mathbf{x}$.

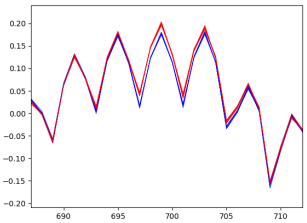
3.1.2 Parity recovery

Computation of the parity in Line 6 of Listing 1. The parity of the sampled coefficient is computed in Line 6. Power traces are not shown here but lead to the exact same analysis as for the sign and allow the recovery of the parity.

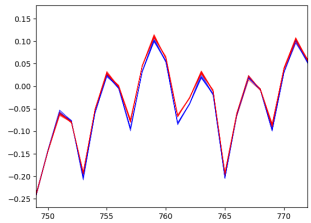
Application of the parity in Line 27 of Listing 1. The application of the parity (Line 27) does not seem as leaky at the application of the sign and is not clearly visible on the power traces. However, the parity is used during the traversal of the two distribution tables to choose which table to use. This means that there are a lot of points of interest. By looking at the code, there should be two points for each iteration of the first table traversal (Lines 12 and 16 of Listing 1) and one point for each iteration of the second table traversal (Line 23 of Listing 1), giving a total of 18 points of potential leakage.

This is confirmed on the power traces, once again visible with bare eyes. We plot the power consumption of Line 12 of Listing 1 in Figure 3a and Line 16 of Listing 1 in Figure 3b. In Figures 3a and 3b, the clear partitioning of the traces into two sets reveals leakage between samples 693 and 705, and between 757 and 765, both in the same iteration of the first table's traversal. The leakage is higher on the first column access because the parity is stored in a 64 bits register, leading to Hamming weight being either 64 or 0, whereas the parity is stored in a 32 bits register for the second access, leading to Hamming weight being either 32 or 0. Leakage from the second table's traversal at Line 23 is not shown here but is very similar to the two other points of interest.

Recovered information. The parity can be recovered with 100% accuracy over our 10 000-coefficient database. The leakage induced by any of the Lines 12, 16, 23 is enough to recover the parity, however the leakage from Line 12 is more significant and would probably lead to better results than the two other lines in a scenario with more noise.



(a) *Power consumption during first column access in the first for-loop (Line 12)*



(b) *Power consumption during second column access in the first for-loop (Line 16)*

Figure 3: *Red (resp. blue) traces correspond to even (resp. odd) coefficients.*

3.2 Looking deeper in the power traces

During each iteration of the two for-loops, the variable `r` is either incremented by 0 or 1 depending on the result of the comparison between a random 72-bits number and the corresponding entry in the table. As the table is filled with reverse cumulative distribution, once the random number is less than a table entry it will be inferior to all the following entries.

Analysis of the first and second iterations. Note that if `r` is not incremented in the first iterations, then it will not be incremented later on and will remain 0 until eventually a sign and a parity is applied to it. Hence, much information can be deduced from the leakage of the first two iterations. While the leakage is less obvious than the one caused by the sign and the parity, we used a simple one-dimensional clustering methods (k-means) to achieve 100% accuracy. In a nutshell, thanks to the first two iterations of lines 13 and 17 of Listing 1, it is possible to obtain the value of all the coefficients in $[-3, 3]$ with 100% accuracy on our database.

Third iteration. Further analysis is required to recover information on larger coefficients. We can extend this attack on the third execution of lines 13 and 17. We report being able to obtain the following bit of information by analysing the leakage of the increment of `r`:

(Case A) either `r` has not been incremented for sure,

(Case B) or the classifier does not know.

Indeed, if `r` has not been incremented (Case A), this means that $\mathbf{x}[i] \in [-5, 5]$, since we are on the third iteration of the loop. As all the coefficients in $[-3, 3]$ were previously identified, we can correctly retrieve the information $\mathbf{x}[i] \in \{-5, -4, 4, 5\}$ if a coefficient is assigned to Case A and was not previously found in a former classification. In the second case (Case B), nothing can be deduced on the value of $\mathbf{x}[i]$. In fact, a large number of coefficients fall in this second case. For example, almost all the 0 are assigned to Case B, thus we simply disregard the Case B coefficients. The key point is that Case A do not have any false positives and the partial classification it provides is thus reliable.

In our 10 000-coefficient database, we had 1449 coefficients belonging to $\{-5, -4, 4, 5\}$, and only 31 of them were classified in Case B. Besides, all the coefficients equal to ± 5 are correctly classified in Case A. So, if a value (not previously classified as in $[-3, 3]$) is classified in Case A and it is odd, one can conclude that it is a ± 5. And if a value (not previously classified as in $[-3, 3]$) is classified in Case A and it is even, one can conclude that it is a ± 4. However, we recall that some ± 4 are assigned to Case B and are thus not recovered.

Computation of the norm. It is possible to distinguish even larger coefficients by looking at other points of interest like the computation of the norm. Recall that by looking at Line 30 of listing 1 and particularly samples 1529 to 1541, it is possible to recover the sign of the coefficient (See Section 3.1). However, more can be exploited in the power traces. Indeed, as it can be seen in Figure 4, the traces appear to be divided in not only 2 distinct sets but 3 distinct sets, two of them corresponding to positive coefficients while the last one corresponding to negative coefficients.

By looking at the sampled coefficients, we remark that the group of traces in the middle corresponds to positive coefficients with their 3rd LSB set to 1. Hence, it is possible to partition the values according to the identified set during the norm computation.

Let us define these three sets. We generated 10 000 coefficients and none of them were outside the interval $[-6, 9]$, which is expected with very high probability due to the standard deviation used in the sampler. In the sequel, we will assume than 9 is a maximum value for a coefficient of $\mathbf{x}$ and -6 a minimum value. In other words, we tolerate a very small failure probability (even smaller for low security levels of Hawk).
With this assumption, we define the following partitioning for our traces, also visible in Figure 4:

- $S_1 = \{0, 1, 2, 3, 8, 9\}$ (top in blue)
- $S_2 = \{4, 5, 6, 7\}$ (middle in yellow)
- $S_3 = \{-1, -2, -3, -4, -5, -6\}$ (bottom in red)

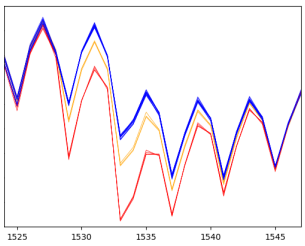


Figure 4: *Power consumption during the norm computation (Line 30). Top traces correspond to S_1, middle traces to S_2, and bottom traces to S_3.*

To wrap up, a finer analysis of the power consumption of Line 30 of listing 1 allows to obtain the j such that $\mathbf{x}[i] \in S_j$. As we were able to retrieve all coefficients between -3 and 3 with the table comparison, it follows that we can identify the coefficients equal to 8 and 9. Indeed, if a value that was not classified before is assigned in S_1, then it is a 8 if it is even and a 9 if it is odd. The values 8 and 9 correspond to a total of 82 for 10 000 coefficients.

At this point, in a total of 10 000 coefficients, 258 of them are still not classified, belonging to the following subset: $\{-4, 4, 7, -6, 6\}$. Note that we do not have any false positives, all coefficients that have been previously classified have been with 100% accuracy. The only odd possible value is 7 so its classification is straightforward. The only unknown now concerns the 220 unclassified even coefficients, that could be equal to ± 4 or ± 6.

Table 1: Number of retrieved coefficients in each subset for 10 000 coefficients. There is no false positives in classified coefficients. In the unrecoverable coefficients, we list the coefficients that were assigned to Case B during the analysis of the table comparison and for which the computation of the norm could not help deducing any information about their value.

Subset	Number of retrieved coefficients	Number of unrecoverable coefficients
$\{-1, 0, 1\}$	4308	0
$\{-3, -2, 2, 3\}$	3934	0
$\{-4, 4\}$	941	31
$\{-5, 5\}$	208	0
$\{-6, 6\}$	0	189
$\{-7, 7\}$	38	0
$\{-9, -8, 8, 9\}$	82	0
Total	9780	220

Table 2: Recovered information from the different analysis. parity (◯), sign (△), value (□)

	◯	△	□ for HAWK-1024
Non-profiled 1 (Sec. 3.1)	100%	100%	7%
Non-profiled 2 (Sec. 3.2)	100%	100%	98%
Profiled (Sec. 3.3)	100%	100%	100%

Conclusion. Combining the information gained from the table comparison and the norm computation (and the sign and parity obtained in Section 3.1), we report being able to recover the exact value of a large fraction of the coefficients of the sample $\mathbf{x}$, we refer to Table 1 for more details. The final percentage of recovered coefficients is around 98% for HAWK-1024, with no false positives, and only two possible values left for the unknown coefficients. For other variants of HAWK, the accuracy is expected to be even higher as the standard deviation of the sampler is smaller, leading to smaller coefficients which are easier to classify.

3.3 A profiled power analysis

In this section, we assess the feasibility of a straightforward template on the whole algorithm to conclude our extensive leakage analysis.

Template generation. The template was generated with 10 000 coefficients, which would correspond to approximately 5 traces in a realistic setup for HAWK-1024, as each signature requires the sampling of 2048 coefficients. Points of interest (PoIs) were classically determined with a Sum of Differences (SoD) method. The best results were obtained with 30 points of interest, so that each iteration of the two loops was represented by at least two PoIs. We represent the sum of differences used to build the template in Figure 5. We did not use the norm computation to build our template, because the leakage of the sign value overrides any other information in this operation.

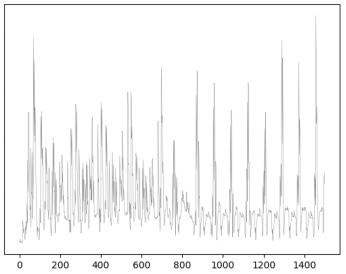


Figure 5: *Sum of Differences used to build the template*

Accuracy of the attack. To assess the accuracy of our template, we generated a set of 1 000 new coefficients. The classification accuracy was of 100%, which would lead to a straightforward key recovery from two different signatures by computing a matrix inversion as described in Equation 1.

3.4 Discussion

As shown in Table 2, the three attacks presented above allow to recover progressively a lot of information about the sample $\mathbf{x}$. The substantial leakage comes from the many different vulnerable points identified in the reference code.

Such a leakage is of course unsurprising given that the targeted reference implementation is unprotected. Table 2 provides a very favorable example that cannot represent realistic attacks where the target device could be more physically protected. Although, in a general setting, given the very large amount of points of leakage, one can assume that some information may still be recovered. For example, one can assume that it is possible to recover x parity bits, y signs and z exact values of coefficients (as shown in Section 3.2, some values like the zeros are very leaky). It is important to show how a partial information can impact the security of the scheme and which are the ways to proceed to a key recovery.

In the next section, we introduce an open problem related to the general impact of partial or complete leakage of the sign, parity and value of $\mathbf{x}$ on the lattice reduction key recovery attack.

4 Impact of partial information on x on the lattice reduction key recovery attack

Overview of the key recovery. Secret key recovery in lattice-based cryptography often resorts in finding a short vector by applying lattice reduction techniques (mostly BKZ [Kan83]). It implies using BKZ on the lattice whose Gram matrix is the public key $\mathbf{Q} = \mathbf{B}^* \cdot \mathbf{B}$ where $\mathbf{B}$ is the secret key. This technique is used by the authors of Hawk to estimate the security of the scheme as well as to highlight necessary security parameters.

Given the partial hints that may be obtained through side-channel attacks, one can hope to improve these attacks to lower the security of the scheme.

This security estimation is performed using the notion of block size β, a crucial parameter of the algorithm BKZ which is directly related to its complexity. Since hints directly impact the volume of a lattice as well as potentially its dimension, it directly impacts the value of β and thus the underlying complexity to find a short vector. We use this to practically lower the security of HAWK given side-channel information.

For the rest of the section, let us denote by Λ the structured lattice of dimension $2n$ whose Gram matrix corresponds to the public key $\mathbf{Q} = \mathbf{B}^* \cdot \mathbf{B}$. It can be constructed from existing tools like [ACD+18, DDGR20, BBD+23].

Search for unusually short vectors The authors of Hawk note that if the lattice reduction of Λ is successful enough, it may provide another basis $\mathbf{V}$ of Λ that is unimodular, meaning $\mathbf{V} \in GL_{2n}(\mathbb{Z})$. In particular, it can provide a basis $\mathbf{V}$ whose vectors in $\mathbb{Z}^{2n}$ are of norm 1, If this is the case, while this $\mathbf{V} \in GL_{2n}(\mathbb{Z})$ may be different from $\mathbf{B}$, it allows anyone to sign as if the matrix $\mathbf{B}$ was used. Indeed, it suffices to use the unstructured version of the signature with $\mathbf{V}$ instead of $\mathbf{B}$. We refer to [BBD+23, Section 5.1] for more details about this cryptanalysis.

4.1 Key recovery complexity evaluation

Finding a vector of norm 1 in Λ consists in applying BKZ algorithm with a block size β that is large enough. The choice of the block size of BKZ offers a trade-off between the quality of the output and the complexity: higher block sizes lead to shorter vectors at the expense of the runtime. The complexity of such an algorithm as function of the block size has been very well documented in the literature [GN08, CN11, ADPS16, AGVW17]. In particular, the link between the blocksize β of BKZ, the size of the target short vector, denoted here $\|\mathbf{s}\|$, and the lattice Λ is as follows:

$$\sqrt{\beta/\dim(\Lambda)} \cdot \|\mathbf{s}\| \leq \delta_\beta^{2\beta-\dim(\Lambda)-1} \cdot \mathrm{Vol}(\Lambda)^{1/\dim(\Lambda)} \tag{2}$$

where δ_β is the root-Hermite-Factor of BKZ-β. For small β's, the root-Hermite-Factor can be tabulated and for $\beta \geq 50$, it is predictable using the Gaussian Heuristic [CN11] $\delta_\beta = \left((\pi\beta)^{\frac{1}{\beta}} \cdot \frac{\beta}{2\pi e}\right)^{1/(2\beta-2)}$.

To estimate the complexity of BKZ, the authors of Hawk use a technique inspired from [DDGR20]. A first step in the analysis consists in scaling the lattice by a factor $\sqrt{2n}$.

Table 3: Type of side-channel information

	Gained knowledge	Type of hint
sign △	$\mathbf{x}_0[i] \geq 0$ or ≤ 0	Inequality hint or approximate hint
parity ◯	$\mathbf{x}_0[i] \bmod 2 = 0$ or 1	Modular hint
value □	value of $\mathbf{x}_0[i]$	Perfect hint

In other words, Λ becomes $\sqrt{2n} \cdot \Lambda$. Hence, in Equation 2, we can then apply the following:

$$\begin{cases} \dim(\Lambda) = 2n \\ \mathrm{Vol}(\Lambda) = \sqrt{2n^{2n} \det(\mathbf{Q})} = 2n^n & (\det(\mathbf{Q}) = 1 \text{ because } \mathbf{B} \text{ is unimodular}) \\ \|\mathbf{s}\| = \sigma_{\mathrm{krsec}} & (\text{given as a parameter}) \end{cases} \tag{3}$$

The scaling by $\sqrt{2n}$ allows to move the dependency in n into the volume instead of the size of the short vector $\|\mathbf{s}\|$ whose size should normally be $\sigma_{\mathrm{krsec}}\sqrt{2n}$.

The complexity estimation script is provided in the HAWK submission (accessible at https://github.com/hawk-sign/aux/blob/main/code). In particular, the function `predict_beta_and_prev_sd` roughly computes the minimum β such that Equation 2 including Equation 3 becomes true. The computation is optimized by using logarithms to avoid large numbers in the volume and a probabilistic technique (see [BBD+23] for more details).
Such a β provides a good estimate of the complexity of BKZ but converting it into a precise bit-security is delicate as such conversions are based on heuristic (still moving) models. For this study, the key notion is the *relative difference* between the black-box security and the residual security with the knowledge gained by side-channel information. So, we can stay with β as a complexity metric and refer for example to [ACD+18] for a summary of existing cost conversion models. We could recompute the block size β with this script for the different versions of HAWK and get the same values as in [BBD+23, Table 7].

Remark 2 (Variants of the security estimation). In addition to this presented estimation technique inspired from [DDGR20], the authors of HAWK also use the GSA intersect method [AGVW17] which provides slightly better results. The authors of HAWK also use the dimensions for free optimization [Duc18], denoted "d4f" in the specification.

4.2 Types of side information

The i-th coefficient of the polynomial $\mathbf{x}$ corresponds to the i-th coefficient of $\mathbf{Bw}$ which is the zero-th coefficient of $\mathbf{B}\tilde{\mathbf{w}}$ where $\tilde{\mathbf{w}}$ is a (known) rotation of $\mathbf{w}$. Hence, any information about a coefficient of the polynomial $\mathbf{x_0}$ provides linear information on the secret vector $\begin{pmatrix} f & F \end{pmatrix}$. The linear information may be of three types (as shown in section 3.1): sign △, parity ◯ and exact value □.

In a previous version of this paper, these hints were integrated in a similar way as [DDGR20] by modifying the volume and dimension of the lattice. However, the authors later figured that this technique was incorrect as the adaptation of the [DDGR20] framework to Gram matrices is actually more complex. The matrix $\mathbf{Q}$ represents an isomorphism class of lattices containing the lattice of basis $\mathbf{B}$ but also all its rotations. Hence, there is no simple way of integrating side information on $\mathbf{B}$ into the the matrix $\mathbf{Q}$. Therefore, this issue is left as an open problem. Ideally, one should be able to use the hints to reduce the dimension or lowering down the coefficients of $\mathbf{Q}$ in order to more easily recover the unusually short vectors.

5 Key recovery without lattice reduction when most of $\mathbf{x}$ is recovered

In this section, we present a full key recovery that leverages the information on both $\mathbf{x_0}$ and $\mathbf{x_1}$ and that does not use a lattice reduction key recovery technique. The attack is detailed for Hawk-1024 where the length of $\mathbf{x}$ is 2048 and the lattice reduction attack remains prohibitive. The case study will be the side-channel attack presented in Section 3.1 that can correctly retrieve 98% of the $x[i]$, leaving an average of only 46 unknown coefficients in $\{-6, -4, 4, 6\}$. In this section, we propose a simple way to retrieve the missing coefficients, leading to a key recovery in reasonable computation time.

Computation of the coefficients distribution. As highlighted in Section 2.2, we can recover $\|\mathbf{x}\|$ from public data and

$$\|\mathbf{x}\|^2 = x[0]^2 + \cdots + x[n-1]^2. \tag{4}$$

By subtracting the squares of the $x[i]$ that we already know to the square of its norm and with a reordering, we can simplify the equation to

$$\alpha = x[0]^2 + \cdots + x[j]^2 \tag{5}$$

with j being the number of unknown coefficients in $\mathbf{x}$ and α is known. As pointed earlier, the unknown coefficients $x[i]$ can only take two absolute values: 4 or 6. We can thus rewrite the equation as $\alpha = n_4 \cdot 16 + n_6 \cdot 36$, with $n_d = \#\{x[i] \mid x[i] = \pm d\}$ and $n_4 + n_6 = j$. This is a simple linear system with two equations and two variables, allowing us to retrieve n_4 and n_6.

Recovery of the private key. The number of possible combinations of the unknown $x[i]$ is given by the binomial coefficient, the worst case being $\binom{j}{j/2} = \binom{46}{23} < 2^{30}$. The distribution between coefficients whose absolute value is equal to 4 or 6 is however not uniform, and a more realistic bound is given by $\binom{46}{7} < 2^{18}$. We recall that a signature $\mathbf{s}$ is computed as follows: $\mathbf{s} = \frac{1}{2}(\mathbf{h} - \mathbf{w})$, with $\mathbf{Bw} = \mathbf{x}$. As shown by Equation 1, we need to recover two different $\mathbf{x}$ and $\mathbf{x}'$ to attempt the recovery of the private basis $\mathbf{B}$ which amounts to around 2^{36} different combinations.

We recall that because of the algebraic structure used in Hawk, all matrices are block-wise negacyclic, and the inversion of a matrix consists on operations on a few polynomials, which can be optimized by various methods (for example with NTT with a small modulus). As we are working with 2x2 matrices, the total number of polynomial multiplications to naively compute Equation 1 is $14 \approx 2^4$, each of them with a complexity of approximately 2^{15} thanks to NTT. This amounts to a total complexity for the key recovery (with a very large margin) of 2^{55} in terms of elementary operations.

6 Reducing the leakage with simple modifications

6.1 Our modifications

In a similar manner as [GMRR22], we propose very light adaptations to the discrete sampler to greatly reduce the leakage induced by the operations. Please note that these adaptations are in no way a provable countermeasure but they are some almost free implementation tricks to avoid straightforward leakages. The modified sampler is presented in Listing 2.

```
1     uint32_t neg = -(uint32_t)(lo >> 63);
2     uint8_t pbit = (t[v >> 3] >> (v & 7)) & 1;
3     uint8_t r0 = 0, r1 = 0;                                          ◯
4     for (size_t i = 0; i < hi_len; i += 2) {
5         uint64_t tlo0 = tab_lo[i + 0];
6         uint64_t tlo1 = tab_lo[i + 1];
7         uint64_t tmp0 = 0x7FFFFFFFFFFFFFFF;                          □
8         uint64_t tmp1 = 0x7FFFFFFFFFFFFFFF;                          □
9         uint32_t cc0 = (uint32_t)((tmp0 - lo + tlo0) >> 63);         □ ◯
10        uint32_t cc1 = (uint32_t)((tmp1 - lo + tlo1) >> 63);         □ ◯
11        uint32_t thi0 = tab_hi[i + 0];
12        uint32_t thi1 = tab_hi[i + 1];
13        tmp0 = 0x7FFF;                                               □
14        tmp1 = 0x7FFF;                                               □
15        r0 += (tmp0 - hi + thi0 + cc0) >> 15;                        □ ◯
16        r1 += (tmp1 - hi + thi1 + cc1) >> 15;                        □ ◯
17    }
18    uint32_t hinz = (hi - 1) >> 31;
19    for (size_t i = hi_len; i < lo_len; i += 2) {
20        uint64_t tlo0 = tab_lo[i + 0];
21        uint64_t tlo1 = tab_lo[i + 1];
22        uint64_t tmp0 = 0x7FFFFFFFFFFFFFFF;                          □
23        uint64_t tmp1 = 0x7FFFFFFFFFFFFFFF;                          □
24        uint32_t cc0 = (uint32_t)((tmp0 - lo + tlo0) >> 63);         □ ◯
25        uint32_t cc1 = (uint32_t)((tmp1 - lo + tlo1) >> 63);         □ ◯
26        r0 += hinz & cc0;                                            ◯
27        r1 += hinz & cc1;                                            ◯
28    }
29    uint8_t r = MUL5((1 - pbit), r0) + MUL5(pbit, r1);               ◯
30    r = (r << 1) | pbit;                                             ◯
31    x[v] = (int8_t)r;
32    sn += (uint32_t)(r * r)
33    x[v] = (x[v] ^ neg) - neg;
34
```

Listing 2: `SamplerSign` with modified lines to mitigate leakage of parity (◯) and value (□) of **x** highlighted. Norm computation and sign application in Lines 32 and 33 are also permuted.

Parity. To prevent the parity from leaking in each iteration of both loops, we recommend sampling into two registers r_0 and r_1 during the tables traversal, and only applying the parity at the very end just before the norm computation, as it was designed in the pseudocode of the HAWK specification. This however implies an overhead as some of the operations need now to be done twice, but on the other hand we remove two XOR and one AND, mitigating the global overhead.

To choose between r_0 and r_1 at the end of the algorithm, we can use the same technique as the one proposed in the original code to choose from one of the two tables:

```
r = r0 ^ (p_odd & (r0 ^ r1));
```

However, even if we reduce the size of the registers to 8 bits, this operation is still very leaky as the Hamming weight of p_odd is either 8 or 0. We can tackle this by using only the one bit stored in p_bit and performing a multiplication:

```
r = (1 - pbit) * r0 + pbit * r1;
```

Multiplications require more CPU cycles than bit-wise operations, but as this operation is only performed once by coefficient the overhead is reasonable. However, multiplication is not constant-time on all the CPUs, and in particular it can be faster if one of the operands is zero or a power of two. To mitigate this, we propose a simple trick inspired by BearSSL[4] and tailored for our small values:

```
#define MUL5(x,y)    ((uint16_t)((x) | (uint16_t)0b11000000) \
                     * (uint16_t)((y) | (uint16_t)0b110000000) \
                     & (uint16_t)0b111111)
```

Contrary to BearSSL, we also manage the case when one of the operands is a power of two as this will necessarily happen.

Comparison with RCDT. We propose a straightforward application of [GMRR22] on the last operation of the first loop, corresponding to line 17. Two variables of 15 bits are subtracted into a 32-bit register, leading to the 17 MSB of the result being either all 1 or all 0 and thus a Hamming weight difference between positive and negative results of at least 17. By initializing the register to $2^{15} - 1$ and inverting the sign of the operands, we can transform the physical underflow into a logical overflow, causing an average Hamming weight difference of only 1.

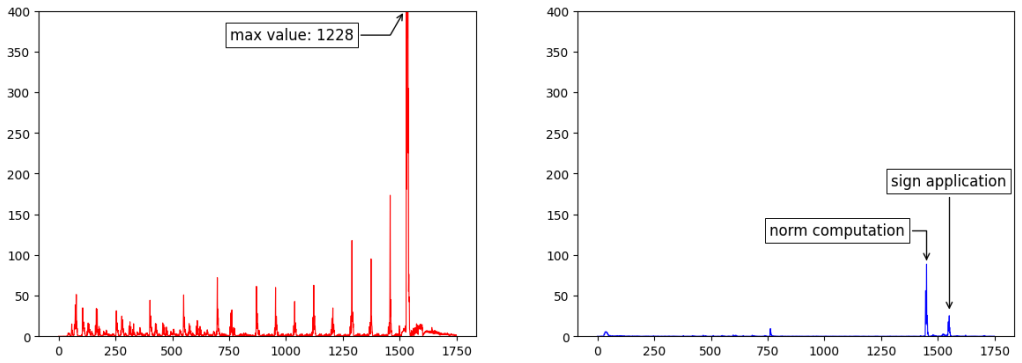
Reducing the leakage for the norm computation. As a very simple additional modification, we permuted the computation of the norm and the application of the sign in Lines 32 and 33. Functionally, this does not change the result as the norm does not depend on the sign of the coefficients. This allows to reduce the leakage as the sign induces very high Hamming weight differences during the multiplication.

Conclusion. We have presented simple modifications that remove many leakage spots without affecting the performance. Indeed, with the modifications, some sensitive values are no longer used in each loop iteration, and the intensity of the leakage for the spots corresponding to the parity is highly reduced as we limit the Hamming weight difference to at most 1 compared to 17 initially. In the next section, we focus our analysis on the effect of our techniques on the leakage.

6.2 Experimental validation

Since the attack targets a sampler, we are in a simple power analysis context (as $\mathbf{x}$ is fresh each time it is sampled). Hence, it is not possible to apply commonly known statistical techniques like Test Vector Leakage Assessments (TVLA) because this test measures the leakage according to a fixed secret and random input (which does not work in our SPA setting). To our knowledge, the most relevant way of validating a countermeasure against an SPA is to compute the signal-to-noise ratio (SNR) reduction of the modified code. A common rule is to validate the design if the SNR is below a certain threshold, preventing SPA attacks for a certain level of noise of the device.

[4] https://www.bearssl.org/ctmul.html



(a) *SNR on the original code (Listing 1)* **(b)** *SNR on our modified code (Listing 2)*

Figure 6: *SNR before and after modification of the code. Remaining spot of leakage in figure 6b correspond to computation of the norm and application of the sign.*

Reduction of the leakage. To estimate the reduction of the leakage, we have plotted the SNR with respect to the value of $\mathbf{x}$ of each sample point of the sampler trace. The impact of our modifications is presented in Figures 6a and 6b. One can see that the leakage is greatly reduced. More precisely, during the core operations of the sampler, i.e. during the tables traversal (before sample 1450), the SNR shows that the leakage is almost unnoticeable. The trick of removing the sign from the norm computation is also highly diminishing the SNR. The SNR of the spike corresponding to the norm is going from 1228 to 89. The sign is still slightly leaking as shown in the graph of Figure 6b.

Performance. The impact of our modifications on the performance is also unnoticeable. As illustrated by the graphs 6a and 6b, a similar number of samples is needed to capture the targeted function before and after the application of our modifications. Overall, we did not notice any change of the number of cycles in the whole signature process.

Residual leakage. The leakage cannot be totally mitigated with our techniques as they are not provable generic masking techniques with fresh randomness. As outlined, Lines 32 and 33 of Listing 2 correspond to the computation of the norm and the sign after the tables traversal. While we permuted them to lower down the leakage, there are no simple tweaks to totally mitigate the leakage: for the sign, it would require modifying the encoding and lead to incompatibility with the rest of the signature, for the norm computation, the multiplication operation leaks the absolute value of $\mathbf{r}$. Therefore, while highly reduced, these two operations still leak some information about $\mathbf{x}$.
Note that since they consist in usual multiplication, XOR and subtraction operations, they could be protected with existing masking techniques with two shares for example. The algorithms (called gadgets) for performing these masked tasks can be found in [BBD+16]. The norm computation can be performed with a SecMult gadget for instance. The cost of such generic countermeasure is certainly a bit more expensive than our ad-hoc modifications but it would allow to efficiently remove the residual leakage in a device with very low noise. As this would require a new implementation and a masking proof, this is left as interesting future work.

Applying our attacks with the modified code. Traces generated after the application of our modifications are also available on the repository. Our non-profiled attacks from Section 3.1 and Section 3.2 can no longer recover the key as even the 0's and 1's cannot be distinguished. Regarding our profiled attack from Section 3.3, we applied it again and

Table 4: Recovered information with different attacks on 1000 coefficients.

	Recovered values of $\mathbf{x}$ for Hawk-1024
Non-Profiled (Sec. 3.1/ 3.2) on the original code	7% and 98%
Non-Profiled (Sec. 3.1/ 3.2) on the modified code	0% and 0%
Profiled (Sec. 3.3) on the original code	100%
Profiled (Sec. 3.3) on the modified code	78% (*at unknown positions*)

store the average percentage of correctly classified coefficients with and without leakage mitigation on 1000 coefficients in Table 4. Contrarily to Table 2, the percentage of the last line leads to a *much weaker knowledge* as the positions of the correctly classified coefficients of $\mathbf{x}$ are unknown. Hence, a heavy brute-force step is necessary to guess the correct positions and this classification cannot work with the key recovery complexity analysis presented in Section 4. All in all, recovering the key from such information seems very challenging.

7 Conclusion

In this paper, we have provided a review of many vulnerable instructions in Hawk's discrete Gaussian sampler and a diversity of ways to use them to recover the private key. In terms of vulnerability, we would be tempted to conclude that side-channel assisted key recoveries are more accessible for Hawk than for Falcon due to the many points of leakage in the Gaussian sampler. The next research step would clearly be to apply side-channel countermeasures to Hawk. On the one hand, protecting Hawk could be technically easier than protecting Falcon because Hawk is free from floating points. But on the other hand, due to the many sensitive instructions inside the sampler, the open issue will rely on minimizing the performance overhead. We hope that our work can provide a basis for more evolved attacks later on if practical protections are one day implemented.

References

[ABC+23] Melissa Azouaoui, Olivier Bronchain, Gaëtan Cassiers, Clément Hoffmann, Yulia Kuzovkova, Joost Renes, Tobias Schneider, Markus Schönauer, François-Xavier Standaert, and Christine van Vredendaal. Protecting Dilithium against leakage revisited sensitivity analysis and improved implementations. *IACR TCHES*, 2023(4):58–79, 2023.

[ACD+18] Martin R. Albrecht, Benjamin R. Curtis, Amit Deo, Alex Davidson, Rachel Player, Eamonn W. Postlethwaite, Fernando Virdia, and Thomas Wunderer. Estimate all the LWE, NTRU schemes! In Dario Catalano and Roberto De Prisco, editors, *SCN 18*, volume 11035 of *LNCS*, pages 351–367. Springer, Cham, September 2018.

[ADPS16] Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. Post-quantum key exchange - A new hope. In Thorsten Holz and Stefan Savage, editors, *USENIX Security 2016*, pages 327–343. USENIX Association, August 2016.

[AGVW17] Martin R. Albrecht, Florian Göpfert, Fernando Virdia, and Thomas Wunderer. Revisiting the expected cost of solving uSVP and applications to LWE. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part I*, volume 10624 of *LNCS*, pages 297–322. Springer, Cham, December 2017.

[BBD+16] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, Pierre-Yves Strub, and Rébecca Zucchini. Strong non-interference and type-directed higher-order masking. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 116–129. ACM Press, October 2016.

[BBD+23] Joppe W. Bos, Olivier Bronchain, Léo Ducas, Serge Fehr, Yu-Hsuan Huang, Thomas Pornin, Eamonn W. Postlethwaite, Thomas Prest, Ludo N. Pulles, and Wessel van Woerden. HAWK. Technical report, National Institute of Standards and Technology, 2023. available at https://csrc.nist.gov/Projects/pqc-dig-sig/round-1-additional-signatures.

[BVC+23] Alexandre Berzati, Andersson Calle Viera, Maya Chartouny, Steven Madec, Damien Vergnaud, and David Vigilant. Exploiting intermediate value leakage in Dilithium: A template-based approach. *IACR TCHES*, 2023(4):188–210, 2023.

[CGTZ23] Jean-Sébastien Coron, François Gérard, Matthias Trannoy, and Rina Zeitoun. Improved gadgets for the high-order masking of Dilithium. *IACR TCHES*, 2023(4):110–145, 2023.

[CN11] Yuanmi Chen and Phong Q. Nguyen. BKZ 2.0: Better lattice security estimates. In Dong Hoon Lee and Xiaoyun Wang, editors, *ASIACRYPT 2011*, volume 7073 of *LNCS*, pages 1–20. Springer, Berlin, Heidelberg, December 2011.

[DDGR20] Dana Dachman-Soled, Léo Ducas, Huijing Gong, and Mélissa Rossi. LWE with side information: Attacks and concrete security estimation. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part II*, volume 12171 of *LNCS*, pages 329–358. Springer, Cham, August 2020.

[dPEK+23] Rafael del Pino, Thomas Espitau, Shuichi Katsumata, Mary Maller, Fabrice Mouhartem, Thomas Prest, Mélissa Rossi, and Markku-Juhani Saarinen. Raccoon signature scheme. *Technical Report, National Institute of Standards and Technology,*, 2023. available at https://csrc.nist.gov/csrc/media/Projects/pqc-dig-sig/documents/round-1/spec-files/raccoon-spec-web.pdf.

[DPPv22] Léo Ducas, Eamonn W. Postlethwaite, Ludo N. Pulles, and Wessel P. J. van Woerden. Hawk: Module LIP makes lattice signatures fast, compact and simple. In Shweta Agrawal and Dongdai Lin, editors, *ASIACRYPT 2022, Part IV*, volume 13794 of *LNCS*, pages 65–94. Springer, Cham, December 2022.

[Duc18] Léo Ducas. Shortest vector from lattice sieving: A few dimensions for free. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part I*, volume 10820 of *LNCS*, pages 125–145. Springer, Cham, April / May 2018.

[Dv22] Léo Ducas and Wessel P. J. van Woerden. On the lattice isomorphism problem, quadratic forms, remarkable lattices, and cryptography. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part III*, volume 13277 of *LNCS*, pages 643–673. Springer, Cham, May / June 2022.

[EAB+23] Mohamed ElGhamrawy, Melissa Azouaoui, Olivier Bronchain, Joost Renes, Tobias Schneider, Markus Schönauer, Okan Seker, and Christine van Vredendaal. From MLWE to RLWE: A differential fault attack on randomized &

deterministic dilithium. Cryptology ePrint Archive, Paper 2023/1074, 2023. https://eprint.iacr.org/2023/1074.

[EFG+22] Thomas Espitau, Pierre-Alain Fouque, François Gérard, Mélissa Rossi, Akira Takahashi, Mehdi Tibouchi, Alexandre Wallet, and Yang Yu. Mitaka: A simpler, parallelizable, maskable variant of falcon. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part III*, volume 13277 of *LNCS*, pages 222–253. Springer, Cham, May / June 2022.

[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 Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part III*, volume 12107 of *LNCS*, pages 34–63. Springer, Cham, May 2020.

[GMRR22] Morgane Guerreau, Ange Martinelli, Thomas Ricosset, and Mélissa Rossi. The hidden parallelepiped is back again: Power analysis attacks on falcon. *IACR TCHES*, 2022(3):141–164, 2022.

[GN08] Nicolas Gama and Phong Q. Nguyen. Predicting lattice reduction. In Nigel P. Smart, editor, *EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 31–51. Springer, Berlin, Heidelberg, April 2008.

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In Richard E. Ladner and Cynthia Dwork, editors, *40th ACM STOC*, pages 197–206. ACM Press, May 2008.

[HR14] Ishay Haviv and Oded Regev. On the lattice isomorphism problem. In Chandra Chekuri, editor, *25th SODA*, pages 391–404. ACM-SIAM, January 2014.

[Kan83] Ravi Kannan. Improved algorithms for integer programming and related lattice problems. In *15th ACM STOC*, pages 193–206. ACM Press, April 1983.

[KH18] Suhri Kim and Seokhie Hong. Single Trace Analysis on Constant Time CDT Sampler and Its Countermeasure. *Applied Sciences*, 8(10), 2018.

[LDK+22] Vadim Lyubashevsky, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Peter Schwabe, Gregor Seiler, Damien Stehlé, and Shi 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.

[Lyu09] Vadim Lyubashevsky. Fiat-Shamir with aborts: Applications to lattice and factoring-based signatures. In Mitsuru Matsui, editor, *ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 598–616. Springer, Berlin, Heidelberg, December 2009.

[Lyu12] Vadim Lyubashevsky. Lattice signatures without trapdoors. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 738–755. Springer, Berlin, Heidelberg, April 2012.

[MUTS22] Soundes Marzougui, Vincent Ulitzsch, Mehdi Tibouchi, and Jean-Pierre Seifert. Profiling side-channel attacks on Dilithium: A small bit-fiddling leak breaks it all. Cryptology ePrint Archive, Report 2022/106, 2022.

[PFH+22] Thomas Prest, Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. FALCON. Technical report, National Institute of Standards and Technology, 2022. available at https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022.

[Pre23] Thomas Prest. A key-recovery attack against mitaka in the t-probing model. In Alexandra Boldyreva and Vladimir Kolesnikov, editors, *PKC 2023, Part I*, volume 13940 of *LNCS*, pages 205–220. Springer, Cham, May 2023.

[ZLYW23] Shiduo Zhang, Xiuhan Lin, Yang Yu, and Weijia Wang. Improved power analysis attacks on falcon. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part IV*, volume 14007 of *LNCS*, pages 565–595. Springer, Cham, April 2023.