The Hidden Parallelepiped Is Back Again: Power Analysis Attacks on Falcon

Morgane Guerreau[1,2], Ange Martinelli[2], Thomas Ricosset[1] and Mélissa Rossi[2]

[1] Thales, Gennevilliers, France
morgane.guerreau@gmail.com
thomas.ricosset@thalesgroup.com
[2] ANSSI, Paris, France
ange.martinelli@ssi.gouv.fr
melissa.rossi@ssi.gouv.fr

Abstract. Falcon is a very efficient and compact lattice-based signature finalist of the NIST's Post-Quantum standardization campaign. This work assesses Falcon's side-channel resistance by analyzing two vulnerabilities, namely the pre-image computation and the trapdoor sampling. The first attack is an improvement of Karabulut and Aysu (DAC 2021). It overcomes several difficulties inherent to the structure of the stored key like the Fourier representation and directly recovers the key with a limited number of traces and a reduced complexity. The main part of this paper is dedicated to our second attack: we show that a simple power analysis during the signature execution could provide the exact value of the output of a subroutine called the base sampler. This intermediate value does not directly lead to the secret and we had to adapt the so-called hidden parallelepiped attack initially introduced by Nguyen and Regev in Eurocrypt 2006 and reused by Ducas and Nguyen in Asiacrypt 2012. We extensively quantify the resources for our attacks and experimentally demonstrate them with Falcon's reference implementation on the ELMO simulator (McCann, Oswald and Whitnall USENIX 2017) and on a ChipWhisperer Lite with STM32F3 target (ARM Cortex M4).
While the success of these attacks may be unsurprising because the reference implementation does not claim any side-channel protection, these new attacks highlight the need for side-channel protection for one of the three finalists of NIST's standardization campaign by pointing out the vulnerable parts and quantifying the resources of the attacks.

Keywords: Power Analysis · Lattices · Falcon Signature Scheme · Hidden Parallelepiped Attack

1 Introduction

With the upcoming menace of quantum computers, post-quantum algorithms are subject to extensive analysis in order to gain assurance in their security. Falcon [PFH+20] is one of the three signature finalists of NIST call for standardization. It is a very competitive scheme with high efficiency both in terms of speed and signature size. This feature makes it the natural option for quantum-safe embedded systems. Such systems are the target of so-called physical attacks, that make use of additional information given through a *side-channel*. Those can be the time of execution, the power consumption or the electromagnetic emanations of the chip during the execution of the algorithm, or even the behavior of the algorithm in the presence of perturbations and errors injections. These attacks are nowadays the major threat to any cryptographic embedded device.

An exploitation of physical data can be of two types: either it makes use of statistical tools to correlate key guesses to information extracted from several leakage traces (unified under the name *Correlation Power Analysis* or CPA) or it merely determines the execution flow of an algorithm based on recognizable patterns in a single power trace (historically called *Simple Power Analysis* or SPA). The strength of the latter is the possibility to recover information from a single trace, but it is easily thwarted by educated implementation, while the former needs many more traces but requires more complex and costly countermeasures.

While the cryptanalytic security of Falcon and its underlying mathematical problem is well studied through best known attacks [PFH+20, (Section 2.5.1)] and security proofs of the underlying GPV framework in both the random oracle model (ROM) [GPV08] and the quantum random oracle model (QROM) [CD20], its physical security can still be considered as unknown territory.

Falcon is a hash-and-sign signature scheme based on the GPV framework [GPV08] and working over NTRU lattices [HHP+03]. These aspects make it both a very efficient signature scheme and a complex one to implement. This complexity comes through two necessary features: the use of floating point arithmetics and the need for Gaussian sampling. Protecting its implementation while preserving its efficiency is even more a challenge. The NIST emphasized the need for side-channel analysis on the finalists and the evaluation of their protection both in terms of security and efficiency. This work aims at doing this analysis in the case of Falcon.

While the Gaussian sampling algorithms used in Falcon were subject to timing attacks [BHLY16, EFGT17, PBY17, FKT+20] and countermeasures were designed [HPRR20] accordingly, concrete power attacks threatening Falcon implementations have only been performed lately in [KA21]. Moreover, this attack targets a subroutine of the algorithm and focuses on the recovery of values encoded in floating points. In particular, no attack of our knowledge targets the Gaussian sampling and the criticality of side-channel protections on this block is, up to now, an open question. More precisely, while a side-channel leakage would obviously occur during an unprotected Gaussian sampling, exploiting it to mount an effective key recovery is not straightforward. Moreover, the potential cost of such attack remains unknown.

Contributions In this article, we give an extensive analysis of side-channel vulnerabilities of Falcon. This analysis goes through two contributions:

- In Section 3, we substantially improve the attack of [KA21] by diving deeper into the specifications of Falcon. In particular, we exploit the fact that the polynomial coefficients are integers to lower the complexity of the attack.The two correlation analysis targetting 26 and 27 bis of the mantissa in [KA21] can be replaced by a single one targeting only 8 bits. Moreover, we use the redundancy of the computation in complex numbers products to divide the needed number of traces by two, leading to a full key recovery on ChipWhisperer with as few as 5 000 traces.

- Our main contribution is presented in Section 4. We propose the first practical power analysis of the Gaussian sampling algorithm in the context of Falcon. We show that one can recover partial information on the samples via simple power analysis and perform a variant of the deformed parallelepiped attack introduced in [DN12] leading to a complete recovery of the secret basis. The attack is quite demanding in terms of computation ressources and measurements but it leads to a key recovery in $\approx 10^6$ traces or less if the attacker is willing to perform a lattice reduction step. We backup our analysis with practical codes running both on material devices and simulated traces for easier reproducibility. Comparatively, the first attack leads to a much more efficient key recovery. Nevertheless, this second attack allows to assess

a new vulnerability that could not be traditionally protected with masking as it leverages a Gaussian sampling step. At last, we outline the different possibilities in terms of countermeasures and provide a modification of the code that significantly lowers the leakage at no additional cost.

As a side contribution, we show in Appendix B that previous conjectured attack from [FKT+20] cannot be achieved with the actual precision provided by computers.

Related work Side-channel attacks on lattice-based signature schemes start to have a relatively long history initiated with a series of attacks [BHLY16, PBY17, EFGT17, BDE+18] on BLISS signature scheme [DDLL13]. Later, NIST's signature finalists were successfully analyzed through various types of side-channel attacks (timing attacks, fault attack or electromagnetic analysis (EMA)), see [FKT+20, RJH+19, BP18, RJH+18, MHS+19] as examples. Falcon is definitely the least studied lattice-based signature scheme in terms of side-channel attacks. Indeed, besides a fault attack [MHS+19], only one timing attack on the Gaussian sampling [FKT+20] and one EMA of the floating point product [KA21] have been attempted. While the former was efficiently thwarted by asynchronous implementation [HPRR20], no efficient countermeasure have been proposed against EMA or power analysis for Falcon. As a comparison, the family of Fiat-Shamir with aborts lattice-based signatures, where the signature finalist Crystals-Dilithium belongs, has been protected against side-channels with masking techniques [BBE+18, GR19, MGTF19] that cannot be easyly applied to Falcon implementations.

Paper organization In the next section, we will recall the necessary context and explicit the notations used in the sequel. Section 3 will focus on the floating point arithmetic and the improvements of [KA21]. In Section 4, we exhibit a practical attack on the Gaussian sampler and show how to exploit this leakage in a full key recovery.

Code The source code of our attacks and experiments on Falcon's reference implementation can be found on this git repository: github.com/mguerrea/FalconPowerAnalysis. It has been made openly accessible online for easy reproducibility.

2 Preliminaries

2.1 GPV

Notations We use bold lowercase letters for vectors and bold uppercase letters for matrices. For a matrix $\mathbf{B}$, we note $\mathbf{b_i}$ the i-th row of $\mathbf{B}$. We denote by $\|\mathbf{v}\|$ the Euclidean norm L_2 of $\mathbf{v}$ and by $\langle \mathbf{u}, \mathbf{v} \rangle$ its associated inner product for two vectors $\mathbf{u}$ and $\mathbf{v}$. Two vectors $\mathbf{u}$ and $\mathbf{v}$ are orthogonal if their inner product is 0, and similarly we say that two matrices $\mathbf{A}$ and $\mathbf{B}$ are orthogonal if their product is the zero matrix (or equivalently if their rows are pairwise orthogonal).

Gram-Schmidt Orthogonalization Let n, m be integer parameters. Let $\mathbf{B} \in \mathbb{R}^{n \times m}$ be a full-rank matrix. We call Gram-Schmidt Orthogonalization (GSO) of $\mathbf{B}$ the unique matrix $\tilde{\mathbf{B}} = (\tilde{\mathbf{b}}_0, \ldots, \tilde{\mathbf{b}}_{n-1}) \in \mathbb{R}^{n \times m}$ such that $\mathbf{B} = \mathbf{L}\tilde{\mathbf{B}} \in \mathbb{R}^{n \times m}$, where $\mathbf{L}$ is lower triangular and the $\tilde{\mathbf{b}}_i$ are pairwise orthogonal. For any positive definite matrix, its LDL* decomposition is the product $\mathbf{L} \cdot \mathbf{D} \cdot \mathbf{L}^*$ where $\mathbf{L}$ is lower triangular and $\mathbf{D}$ is diagonal. The notation $\mathbf{L}^*$ denotes the transpose conjugate of $\mathbf{L}$. Note the following equivalence, for any $\mathbf{B}, \mathbf{L}, \tilde{\mathbf{B}} \in \mathbb{R}^{n \times m}$:

$\mathbf{L}\cdot\tilde{\mathbf{B}}$ is the GSO decomposition of $\mathbf{B} \iff \mathbf{L}\cdot(\tilde{\mathbf{B}}\tilde{\mathbf{B}}^*)\cdot\mathbf{L}^*$ is the LDL* decomposition of $\tilde{\mathbf{B}}\tilde{\mathbf{B}}^*$.

Lattices A lattice is a discrete subgroup of $\mathbb{R}^m$. For a basis $\mathbf{B} = (\mathbf{b_0}, ..., \mathbf{b_{n-1}}) \in \mathbb{R}^{n \times m}$, we note $\Lambda(\mathbf{B})$ and call lattice generated by $\mathbf{B}$ the set of vectors:

$$\left\{ \sum_{i=1}^{n} x_i \mathbf{b_i} \mid x_i \in \mathbb{Z} \right\}$$

A lattice will be noted Λ or $\Lambda(\mathbf{B})$ when a basis $\mathbf{B}$ will be explicitly provided. Given a lattice Λ generated by $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$, we define the lattice orthogonal to Λ modulo q as:

$$\Lambda_q^{\perp} := \left\{ \mathbf{v} \in \mathbb{Z}^m \mid \mathbf{A}\mathbf{v}^t = \mathbf{0} \bmod q \right\}.$$

Gaussian Distributions For $\sigma \in \mathbb{R}$ with $\sigma > 0$ and any $\mathbf{c} \in \mathbb{R}^n$, we call the Gaussian function centered at $\mathbf{c}$ of standard deviation σ the function defined over $\mathbb{R}^n$ as $\rho_{\sigma,\mathbf{c}}(\mathbf{x}) := \exp\left(-\frac{\|\mathbf{x}-\mathbf{c}\|^2}{2\sigma^2}\right)$. We call discrete Gaussian distribution over Λ of standard deviation σ and center $\mathbf{c}$, the distribution defined for all $\mathbf{z} \in \Lambda$ by $D_{\Lambda,\sigma,\mathbf{c}}(\mathbf{z}) = \rho_{\sigma,\mathbf{c}}(\mathbf{z}) / \left(\sum_{\mathbf{x} \in \Lambda} \rho_{\sigma,\mathbf{c}}(\mathbf{x})\right)$.

Fourier Transform For n a power of two, let ω be the n-th complex square root of unity, we recall the Fourier transform:

$$\begin{aligned} \text{FFT}: \quad \mathbb{Z}[x] &\to \mathbb{C}[x] \\ f = (f_0, ..., f_{n-1}) &\mapsto \hat{f} = (\hat{f}_0, ..., \hat{f}_{n-1}) \\ &\quad \text{where } \hat{f}_j := \sum_{k=0}^{n} f_k \cdot \omega^{jk}, \end{aligned}$$

and its inverse:

$$\begin{aligned} \text{FFT}^{-1}: \quad \mathbb{C}[x] &\to \mathbb{Z}[x] \\ \hat{f} = (\hat{f}_0, ..., \hat{f}_{n-1}) &\mapsto f = (f_0, ..., f_{n-1}) \\ &\quad \text{where } f_j := \frac{1}{n} \sum_{k=0}^{n} \hat{f}_k \cdot \omega^{-jk}. \end{aligned}$$

For a polynomial f, we denote by $\hat{f}$ its Fourier representation. We extend the FFT and its inverse to matrices and vectors by component-wise application, and for a matrix $\mathbf{B}$ we note $\tilde{\mathbf{B}}$ its Fourier representation.

GPV framework The GPV framework [GPV08] is a hash-and-sign protocol. The key pair consists in two lattice basis $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ and $\mathbf{B} \in \mathbb{Z}_q^{m \times m}$, where $\mathbf{A}$ is a public basis generating a lattice Λ and $\mathbf{B}$ is a private basis generating the corresponding lattice $\Lambda_q^{\perp}$. The matrix $\mathbf{B}$ should contain small, almost orthogonal vectors, while $\mathbf{A}$ should be indistinguishable from a uniformly generated matrix such as $\mathbf{A} \cdot \mathbf{B}^t = \mathbf{0} \bmod q$. The signature generation for a message $m \in \{0,1\}^*$ proceeds as follows:

- First, the signer computes $H(m)$ a hash of its message, with $H : \{0,1\}^* \to \mathbb{Z}_q^n$ modeled as a random oracle.
- Next, they find a pre-image $\mathbf{c} \in \mathbb{Z}_q^m$ such that $\mathbf{A}\mathbf{c}^t = H(m)$. Because there is no requirement on the size of the coefficients of $\mathbf{c}$, this step can be achieved with linear algebra and the sole knowledge of the public key $\mathbf{A}$.
- Finally, the signer uses a trapdoor sampler to find a vector $\mathbf{v} \in \Lambda$ close in L_2 norm to $\mathbf{c}$. The secret basis $\mathbf{B}$ is the necessary trapdoor. The signature $\mathbf{s}$ is the difference between $\mathbf{v}$ and the pre-image $\mathbf{c}$, $\mathbf{s} := \mathbf{v} - \mathbf{c}$.

To verify that $\mathbf{s}$ is a valid signature, one has to check that $\mathbf{As}^t = H(m) \bmod q$ and that $\|\mathbf{s}\|$ is small. This is indeed the case: we have $\mathbf{As^t} = \mathbf{A}(\mathbf{v}-\mathbf{c})^t = \mathbf{Av}^t - \mathbf{Ac}^t = H(m) \bmod q$ because $\mathbf{Av}^t = 0 \bmod q$ since the lattices defined by $\mathbf{A}$ and $\mathbf{B}$ are orthogonal modulo q. Furthermore, since $\mathbf{v}$ is close to $\mathbf{c}$, their difference is small and so is $\|\mathbf{s}\|$.

2.2 Hidden Parallelepiped Problem

Hidden Parallelepiped Problem. In [NR06], Nguyen and Regev present an attack against the signature schemes NTRUSign [HHP+03] and GGH [GGH97], introducing the so-called Hidden Parallelepiped Problem (HPP). The vulnerability of these two schemes comes from a correlation between the distribution of the signatures and the secret key. With a few thousand signatures, the authors could recover the underlying private key. As this attack will be revisited in our paper, we outline its principle.

Assume that in the GPV framework, the last step proceeds as follows. To find a vector $\mathbf{v} \in \Lambda$ close to $\mathbf{c}$, the signer uses the private matrix $\mathbf{B}$ to perform the round-off algorithm [Bab85, Bab86]. In other words, $\mathbf{v}$ is defined as $\lfloor \mathbf{cB}^{-1} \rceil \mathbf{B}$. This procedure ensures that $\mathbf{v}$ is deterministically defined as a closest vector of $\mathbf{c}$ in Λ. Thus, $\mathbf{v}-\mathbf{c}$ belongs in the fundamental parallelepiped $\mathcal{P}(\mathbf{B}) := \{\mathbf{xB}, \mathbf{x} \in [-1/2, 1/2]^n\}$ where n is the number of rows of $\mathbf{B}$.

Problem 1 (The Hidden Parallelepiped Problem)**.** *Let $\mathbf{b_0}, \ldots, \mathbf{b_{n-1}} \in \mathbb{R}^n$ be n linearly independent vectors and let $\mathcal{P}(\mathbf{B}) = \{\sum_{i=0}^{n-1} x_i \mathbf{b_i}, x_i \in [-1, 1]\}$ be the parallelepiped spanned by $\mathbf{B} := (\mathbf{b_0}, \ldots, \mathbf{b_{n-1}})$. Given a sequence of poly(n) independent samples drawn uniformly at random in $\mathcal{P}(\mathbf{B})$, find a good approximation of the $\pm\mathbf{b_i}$'s.*

Let U be the set of samples drawn uniformly at random in $\mathcal{P}(\mathbf{B})$, Nguyen and Regev show in [NR06] that the $\pm\mathbf{b_i}$'s are local minimums of the experimental fourth moment function defined as:

$$\mathrm{mom}_{\mathbf{V},4} : \mathbf{w} \mapsto \mathrm{Exp}_{\mathbf{u}\in U}[\langle \mathbf{u}, \mathbf{w} \rangle^4],$$

where $\mathrm{Exp}_{\mathbf{u}\in U}[\cdot]$ denotes the experimental expectation over the set of samples. The authors then propose an algorithm solving the HPP by gradient descent. Since NTRUSign and GGH signatures are in the parallelepiped spanned by their private bases, solving HPP results in breaking NTRUSign and GGH.

Deformed Parallelepiped Problem. In [DN12], Ducas and Nguyen extend the attack of [NR06] on protected instances. In particular, they attack several NTRUSign instances applying the "IEEET-IT perturbation". This perturbation consists in adding a small perturbation $\delta \in [-1, 1]^n$ during the round-off algorithm before the multiplication by the secret basis. The vector $\mathbf{v}$ becomes $(\lfloor \mathbf{cB}^{-1} \rceil + \delta)\mathbf{B}$.

Such perturbation is said to be *partial* in the sense that a set of fixed coordinate indexes are ignored by the perturbation in all samples. Ducas and Nguyen show that, if the perturbation is partial, then the attack of [NR06] can be applied to recover some $\pm\mathbf{b_i}$'s. In particular, if the i-th coordinate is ignored by the perturbation δ, then the [NR06] attack can successfully retrieve $\pm\mathbf{b}_i$.

2.3 Falcon signature scheme

We describe here the high level idea behind Falcon. Several parts of the specification are omitted as they are not necessary for understanding our work, we refer to the NIST submission of Falcon [PFH+20] for a complete description. Let us first fix the necessary notations and parameters. Let $q \in \mathbb{N}^*$ and n a power of two be two public parameters of the scheme. In practice, q is set to 12289, and n is either 512 (Falcon-512) or 1024 (Falcon-1024).

We define $\mathcal{R} := \mathbb{Z}_q[x]/(x^n + 1)$. The elements $F \in \mathcal{R}$ will be equivalently represented as polynomials e.g. $\sum_{i=0}^{n-1} F_i \cdot x^i$ or vectors of coefficients e.g. $(F_0, F_1, ..., F_{n-1})$. We denote by F^* the adjoint of F, defined by $F^* := F(x^{-1}) = F_0 + \sum_{i=1}^{n-1} -F_{n-i}x^i$. We extend the definition to matrices: for a matrix $\mathbf{A}$, $\mathbf{A}^*$ denotes the entry-wise adjoint of the transpose of $\mathbf{A}$.

Falcon is an instantiation of the GPV framework with NTRU lattices. More precisely, the public and private keys are defined as follows. We first draw two private polynomials f and g in $\mathcal{R}$ with small coefficients. In practice, the coefficients of the polynomials f and g are generated following a discrete Gaussian distribution with center 0 and standard deviation $\sigma = 1.17\sqrt{q/2n}$. Then, we compute two (uniquely defined) additional private polynomials $F, G \in \mathcal{R}$ such that

$$fG - gF = q \bmod (x^n + 1).$$

We refer to [PP19] for more detailed information about this computation. Finally, a public polynomial $h \in \mathcal{R}$ is derived such that $h = g \cdot f^{-1} \bmod q$. Recovering the secret polynomials from the knowledge of h corresponds to breaking the so-called NTRU problem.

From these polynomials, the public basis $\mathbf{A} \in \mathcal{R}^2$ and the private basis $\mathbf{B} \in \mathcal{R}^{2\times 2}$ as defined above in the GPV framework are defined as follows.

$$\mathbf{A} := \begin{pmatrix} 1 & h^* \end{pmatrix}, \ \mathbf{B} := \begin{pmatrix} g & -f \\ G & -F \end{pmatrix}.$$

Note that these bases – and thus their associated lattices – are orthogonal modulo q, i. e. $\mathbf{B} \cdot \mathbf{A}^* = 0 \bmod q$.

The public key of Falcon's signature scheme is defined as $pk := h$. For efficiency reasons, the secret key is not directly defined as $\mathbf{B}$ but as $sk := (\hat{\mathbf{B}}, T)$ where $\hat{\mathbf{B}}$ is the entry-wise Fourier representation of the private basis $\mathbf{B}$, and T is a binary tree representing the Gram-Schmidt Orthogonalization of $\mathbf{B}$.

The signature algorithm follows GPV framework and is presented in Algorithm 1. Compared to the initial GPV framework, this procedure has been refined in several ways. First, a salt $\mathbf{r}$ has been added in the hashing. Secondly, the target is computed with the private key instead of the public matrix, this modification is needed to make the GPV framework compatible with the trapdoor sampler used in Falcon. Thirdly, many intermediate values are represented in the Fourier domain to comply with the trapdoor requirements (see later in Section 2.3.1). We refer to the specifications for the verification algorithm and the correctness of this signature scheme.

In this paper, the term "pre-image computation" refers to the step 3 of algorithm 1 and the term "trapdoor sampler" corresponds to the step 5 of algorithm 1.

2.3.1 Trapdoor sampler

In reaction to the hidden parallelepiped attack outlined in Section 2.2, the design of Falcon ensures that the distribution of the signatures is independent of the secret basis. The solution lies in the introduction of a non-deterministic sampler called `ffSampling`, based on the Fast Fourier Nearest Plane algorithm [DP16].

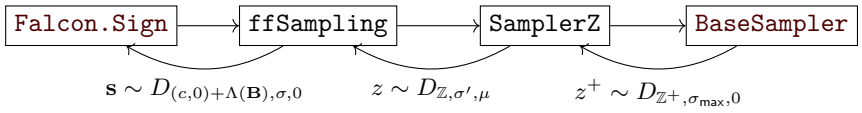


Figure 1: Falcon's signature flowchart

Algorithm 1: FALCON.Sign(m, sk)

Input : A message m, a secret key $sk = (\hat{\mathbf{B}}, T)$
Output : A signature sig

$r \xleftarrow{\$} \{0,1\}^{320}$ uniformly
$c \leftarrow \texttt{HashToPoint}(r||m, q, n)$ $\quad c \in \mathcal{R}$
$\hat{\mathbf{t}} \leftarrow (\hat{c}, 0) \cdot \hat{\mathbf{B}}^{-1}$ • pre-image computation
do
  $\hat{\mathbf{v}} \leftarrow \texttt{ffSampling}(\hat{\mathbf{t}}, T)$ • trapdoor sampler
  $\hat{\mathbf{s}} \leftarrow (\hat{\mathbf{t}} - \hat{\mathbf{v}}) \cdot \hat{\mathbf{B}}$
while $\|\mathbf{s}\|^2 > \lfloor 2.42 \cdot n \cdot \sigma^2 \rfloor$ $\quad \sigma := \frac{1.17}{\pi\sqrt{2}} \cdot \sqrt{q \cdot \log\Big(4n(1 + 2^{32} \cdot \sqrt{n/4})\Big)}$;
return $sig := (r, \mathbf{s})$

The inputs of `ffSampling` are a binary tree T representing the GSO of the private basis and a pre-image $\mathbf{t}$. It outputs a vector $\mathbf{v} \in \Lambda$ such that $(\mathbf{t} - \mathbf{v}) \cdot \mathbf{B} \sim D_{(c,0)+\Lambda(\mathbf{B}),\sigma,0}$. Note that the coordinates of $\mathbf{v}$ are not sampled linearly as `ffSampling` is a recursive algorithm with a tree traversal.

As a subroutine, it uses a discrete Gaussian sampler `SamplerZ` with varying mean μ and standard deviation σ'. The standard deviations σ' are derived from the Gram-Schmidt orthogonalization $\hat{\mathbf{B}}$ of the private basis $\mathbf{B}$.

This `SamplerZ` calls another sampler called `BaseSampler`, whose distribution is statistically close to a half Gaussian distribution centered on 0 with a fixed standard deviation. This standard deviation is a parameter of the scheme and is denoted $\sigma_{\max}$. In practice, for FALCON-512 and FALCON-1024, $\sigma_{\max} = 1.8205$. The closeness of the output of `BaseSampler` to the ideal half-Gaussian is ensured with Rényi divergence arguments (see [HPRR20] for more details).

The `BaseSampler` is presented in algorithm 2 where we denote by $[\![P]\!]$ the function that, for any logical proposition P, returns 1 if P is true, and 0 if P is false. We also call RCDT a reverse cumulative distribution table generated as a parameter of FALCON.

Algorithm 2: `BaseSampler` ()

Output : An integer $z^+ \sim D_{\mathbb{Z}^+,\sigma_{\max},0}$

$u \leftarrow \texttt{UniformBits}(72)$
$z^+ \leftarrow 0$
for $i \leftarrow 0$ **to** 17 **do**
  $z^+ \leftarrow z^+ + [\![u < \mathrm{RCDT}[i]]\!]$
end
return z^+

Floating Point Arithmetic Since the polynomials considered are in Fourier representation, their coefficients are complex numbers. Their real and imaginary part are stored as float numbers in the "double precision" encoding. In double representation, each number is encoded over 64 bits split into three elements: the first bit, denoted s, represents the sign, the 11 following bits represent the exponent denoted e, and the remaining 52 bits represent the mantissa denoted m. These elements are such that the intermediate value is defined as $(-1)^s \cdot 2^{e-1023} \cdot m$.

In Falcon, multiplication between two numbers in double precision is done in three steps: the mantissas of the operands are multiplied, then the exponents of the operands are added, and finally the signs of the operands are xored together.

3 Improvement of the pre-image attack

The first leak one can encounter in Falcon's signature scheme (algorithm 1) targets the pre-image computation in step 3. Indeed, this step consists in a multiplication of a known value, namely $(\hat{c}, 0)$, and the secret key $\hat{\mathbf{B}}^{-1}$. Note that the known value cannot be chosen by the entry-wise as it is salted with a random salt $\mathbf{r}$ generated at step 1. The computation $(\hat{c}, 0) \cdot \hat{\mathbf{B}}^{-1}$, implies the computation of $\hat{c} \cdot \hat{f}$ where f is defined as part of the secret key in Section 2.3. We focus on this particular multiplication to recover f. Indeed, the whole private key can be recomputed from the public key and f.

On a high level, the success of this first attack comes with no surprise as it targets a simple product between key material and known values on an unprotected device. However, the specificity of this attacks comes from the type of the intermediate variables targeted. In fact, performing a side-channel on complex numbers that correspond to the Fourier representation of the variables encoded in double precision was not straightforward and required some precision analysis. Our attack can be seen as an improvement of [KA21] in terms of methodology. We emphasize on the methodology as it is not relevant to compare our works in terms of absolute number of traces, as our experimental set-ups were different.

Even though this section is not our main contribution, we believe that this attack can be of interest for anyone looking to perform side-channel attacks on polynomial multiplication with Fast Fourier transformation, or more generally on floating points.

3.1 The general idea of [KA21]

The authors of [KA21] propose a general attack targeting a multiplication between two numbers in double representation, which recovers one of the operand with complete precision (i.e. 53 bits) if the other one is known and give application to attack Falcon. As [KA21] outlined, the sign, exponent and mantissa can be retrieved separately as they are computed separately in the global multiplication. In [KA21] the recovery of the sole mantissa requires six correlation analysis: three on each half of the mantissa, targeting two multiplications and one addition. Our improvements are threefold:

- We show that only the 6 MSB of the mantissa are required to derive the key, which can be recovered with a single CPA on the higher half of the mantissa.
- We use the redundancy in complex product to halve the number of required traces.
- We present a simple way to reduce noise by exploiting *similar* inputs.

3.2 Lowering the complexity of exhaustive search

In the specific case of Falcon's signature generation, and in the more general case of a multiplication between two integral polynomials in FFT representation, one does not need perfect precision. In a nutshell, it is not necessary to recover the entire mantissa coefficients in order to derive exactly the polynomial after applying the inverse Fourier transformation.

Thanks to the $\mathbb{R}$-linearity of FFT^{-1}, the relative error is conserved through FFT^{-1}. Indeed, let $(\hat{f}'_j) \in \mathbb{C}^n$ and $(\hat{f}_j) \in \mathbb{C}^n$ be the Fourier representation of two polynomials $f \in \mathbb{Z}[x]$ and $f' \in \mathbb{Z}[x]$. Assume that $\hat{f}'_j = \hat{f}_j + \epsilon \hat{f}_j$ with $\epsilon \geq 0$; then after applying FFT^{-1} we obtain $f'_j = f_j + \epsilon f_j$.

Lemma 1 (Partial knowledge of the coefficients)**.** *Let $\hat{f}$ be the Fourier representation of an unknown polynomial $f = \sum_{j=0}^{n-1} f_j \cdot x^j \in \mathbb{Z}[x]$ and let $p \in \mathbb{N}$. Assume that f is defined such that $|f_j| < 2^p$ for all $0 \leq j \leq n-1$. Then the knowledge for all the $\hat{f}_j$ of (1) their sign, (2) their exponent and (3) the $p+1$ most significant bits of their mantissa, is enough to derive f exactly.*

Proof. We fix $0 \leq j \leq n-1$. Assume that the sign, denoted s_j, the exponent, denoted e_j, and the $p+1$ most significant bits of the mantissa of $\hat{f}_j$ are known. Let m_j be the exact mantissa of $\hat{f}_j$ and let m'_j be the approximation of m_j where the bits outside the $p+1$ most significant bits are set to 0. Let $\hat{f}'_j := (-1)^{s_j} \cdot 2^{e_j - 1023} \cdot m'_j$. Then, the relative error between $\hat{f}_j$ and $\hat{f}_j{}'$ is defined as $\epsilon := |(\hat{f}_j - \hat{f}'_j)/\hat{f}_j| = |m'_j - m_j|/|m_j| \leq 2^{-p-1}$. Then, by $\mathbb{R}$-linearity of FFT^{-1},

$$|f_j - f'_j| = \epsilon \cdot |f_j| < 2^{-p-1} \cdot 2^p = 0.5.$$

As the coefficients f_j lie in $\mathbb{Z}$, one can recover the correct coefficients by rounding. □

In Falcon-512 and Falcon-1024, the coefficients of f lie in $[-2^5, 2^5]$. By Lemma 1, besides the sign and the exponent, only the 6 most significant bits of the mantissa are necessary to recover each coefficient of f. Thus, our attack recovers the mantissa by a single 8-bits CPA taking some margin. The computational complexity of our attack is thus bounded by the recovery of the exponent, that is an 11-bit CPA.

3.3 Halving the number of required traces

The multiplication between $\hat{c}$, and the secret key f is a succession of multiplications between complex numbers. A multiplication between two coefficients will thus involve two elements, one element being the real part and the other being the imaginary part. More precisely, the complete pattern will be formed of two multiplications, one subtraction, then again two multiplications, and finally one addition: `MulMulSubMulMulAdd`. The first and second multiplications of the pattern are enough to recover a coefficient f_i, as they involve the imaginary part and the real part of the operands to compute the real part of the result. However, we can also exploit the third and fourth multiplications, as they too involve the real part and the imaginary part of the secret coefficient f_i, this time to compute the imaginary part of the result. We can group together the sub-traces of the `Mul` patterns that involve the same element of f_i as shown in Table 1. This allows us to divide the required number of acquisitions by two while having the same number of actual leaking traces.

Table 1: Utilization of the real and imaginary parts of the operands

	$\mathrm{Re}(\hat{f}_i)$	$\mathrm{Im}(\hat{f}_i)$
$\mathrm{Re}(\hat{c}_i)$	1st `Mul`	3rd `Mul`
$\mathrm{Im}(\hat{c}_i)$	4th `Mul`	2nd `Mul`

3.4 Mitigating the noise

A classical way to deal with the noise in power analysis is to produce several traces for the same values and to average them. This is not possible in our model because the message is salted with a random $\mathbf{r}$ salt generated at step 1 of algorithm 1. Interestingly, we can nevertheless reuse the results of Section 3.2 to create groups of traces for *similar* values of $\hat{c}$ and average them to mitigate noise. More precisely, c_i and a c'_i are said *k-similar* if they share the same sign, exponent, and the k most significant bits of their mantissas.

This number k is a trade-off between the need for precision and the objective of combining several traces for similar value. Experimentally, we observed good results by setting[1] k to 10.

3.5 Experimental results

In this paper, we experimented our attacks on both simulated traces and real acquisitions. Our simulated traces were generated using the ELMO simulator [MOW17], which emulates consumption power of an ARM Cortex M0 processor and produces noise-free traces. The tool reproduces the three stages pipeline of a M0 processor though, which means that the algorithmic noise is taken into account. The actual power traces were generated from a ChipWhisperer Lite with STM32F3 target (ARM Cortex M4) [OC14]. Table 2 gives the number of traces needed to recover the full key in these different setups.

Table 2: Number of traces needed to perform a key recovery with power analysis

	Number of Traces
ELMO	2 000
ChipWhisperer	5 000

We recall that comparing the number of traces with [KA21] is not relevant, as the measurement setups are quite different. However, for a fair comparison, we have implemented the state of the art and we show in Figure 2 that the exploitation of all `Mul` patterns allows to halve the number of traces. Furthermore, while we have no theoretical claim concerning the benefits of Section 3.4's improvement, as experimentally shown on Figure 2, it greatly diminishes the number of traces for a given probability of success. As a side benefit, this trick also reduces the computation time of the CPA, as the averaging of the traces results in less data to analyze. Finally, we recall that our attack has a better computational complexity than [KA21].

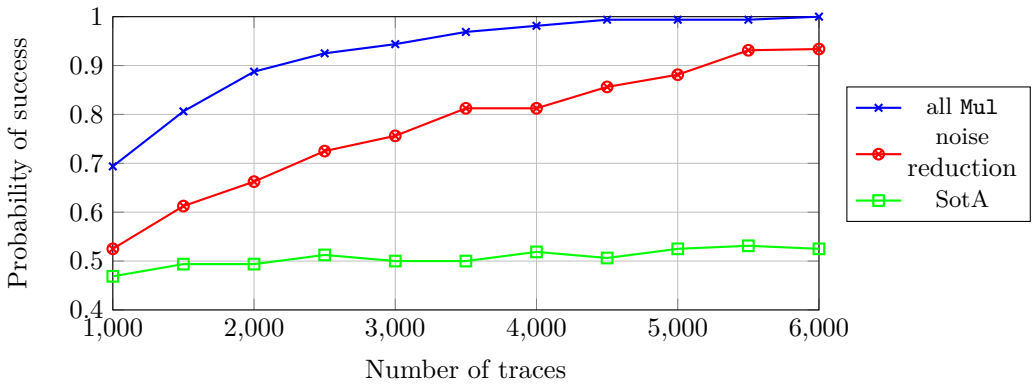


Figure 2: Probability of recovering one coefficient as a function of the number of traces measured with a ChipWhisperer (data for 10×16 coefficients). We plot the attack with no improvement in green (State of the Art [KA21]). The attack with noise reduction (sec. 3.4) is plotted in red. The blue plot corresponds to the attack with noise reduction and exploitation of all `Mul` (sec. 3.3).

[1] One could be surprised by such a high value because in that case two 10-similar values share $1 + 11 + 10 = 22$ bits. But actually, there are less than 14 possible values for the exponent, as c lies in $\mathbb{Z}_q[x]$ with $q = 12289$.

4 Unravelling the Hidden Parallelepiped with side-channel information

Falcon's Gaussian sampling is a complex part of the signature algorithm. Contrary to the recent signature scheme based on the Fiat–Shamir with aborts paradigm, this Gaussian sampling cannot easily be removed from the design. It is a very sensitive operation as each sample depends on the secret key. However it is also a complex target and only one timing attack was attempted on it [FKT+20], without successfully recovering the key. We provide further information about the possibility of this attack in Appendix B. In this section, we propose the very first side-channel key recovery of Falcon's Gaussian sampling. We can summarize our attack steps as follows.

- Step 1: In Section 4.1, we apply a single-trace attack on the `BaseSampler` to recover some knowledge on the z_i. The information consists in knowing if $z_i \in \{0, 1\}$ or not[2].
- Step 2: In Section 4.2, we apply a variant of the HPP solver on several sets of filtered signatures according to the knowledge gained in Step 1.
- Step 3: In Section 4.3, we finalize the key recovery with lattice reduction.

Finally, we analyze the attack and its performance in Section 4.4.

4.1 Step 1: Side-channel analysis on the `BaseSampler`

As outlined in Section 2.3.1, Falcon's trapdoor sampler relies on a `BaseSampler` to sample integers. This sampler, presented in algorithm 2, is constant-time, but previous work by [KH18] have shown that table-based samplers are vulnerable to Simple Power Analysis (SPA). This is also the case for the `BaseSampler`. Moreover, we show in the sequel that we only need to classify whether the samples z^+ are equal to 0 or not, which removes the constraint to retrieve the exact value of z^+. The goal of this section is to explicit how to make this classification.

The comparison $[\![u < \mathrm{RCDT}[i]]\!]$ at line 4 of `BaseSampler` is implemented as a subtraction of the 72-bit variables u and $\mathrm{RCDT}[i]$. More precisely, in Falcon's official implementation, this subtraction is split in three successive subtractions between two 24-bit unsigned integer stored in 32-bit registers – the 8 most significant bits thus set as 0. When the result of this subtraction is negative, the register *underflows*, setting its 8 most significant bits to 1. If the result is positive, the 8 most significant bits remain unchanged. Thus, the Hamming weight of the two possible results differs by 8. This causes a significant difference in the power consumption, allowing the distinction of the result of the comparison and the increment of z^+ (or the absence thereof). This theoretically allows the side-channel attacker to determine how many times the comparison results in an increment by computing the number of high (respectively low) consumptions, and thus the actual value of the sample. Figure 3 illustrates the variations in power consumption during the execution of the comparisons.

[2] $z_i \in \{0, 1\}$ if and only if the corresponding call to `BaseSampler` returns $z^+ = 0$.

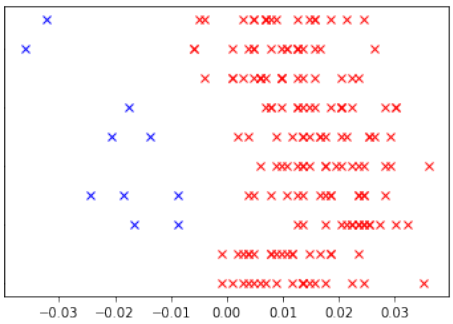


Figure 3: Power variation during the execution of the substraction. Each line is a different execution. Blue ticks stand for incrementation of z^+, red ones for absence of it.

Similarly to [KH18], we apply a low-pass filter to reduce the noise, and determine an experimental threshold to distinguish between the iterations containing incrementations of z^+. As the case where $z^+ = 0$ is the only one relevant for our attack, it is possible to adapt the threshold to identify only the executions of `BaseSampler` with no incrementation at all. For the next steps, we assume that all the $z^+ = 0$ are clearly identified. Note that after the execution of `BaseSampler`, a random sign bit is generated. Thus, $z^+ = 0$ becomes $z_i = 0$ or $z_i = 1$, and since the sign bit is not relevant in our attack we will consider both cases.

4.2 Step 2: application of the HPP

4.2.1 Filtering the signatures though side-channel analysis

Now that we have exposed physical leakage within the `BaseSampler`, we propose an attack exploiting the leaked samples to recover the FALCON's secret key.
The `ffSampling` algorithm works in an equivalent way than the Klein-GPV algorithm [Kle00, GPV08]. Given a target vector $\mathbf{t}_n := \mathbf{t}$, the Klein-GPV algorithm proceeds as follows on $\mathbf{v}_n \leftarrow \mathbf{0}$ for $i = n-1, \ldots, 0$:

1. Let $x_i \leftarrow \langle \mathbf{t}_i, \tilde{\mathbf{b}}_i \rangle / \left\|\tilde{\mathbf{b}}_i\right\|^2$, implying that $x_i \tilde{\mathbf{b}}_i + \text{span}(\mathbf{b}_0, \ldots, \mathbf{b}_{i-1})$ is close to $\mathbf{t}$.
2. Draw $z_i \sim D_{\mathbb{Z}, \sigma', x_i - \lfloor x_i \rceil}$ with $\sigma' := \sigma / \left\|\tilde{\mathbf{b}}_i\right\|$, and let $k_i \leftarrow \lfloor x_i \rceil + z_i$. As illustrated in Figure 4, this is the only step that differs from the nearest plane algorithm for which $k_i \leftarrow \lfloor x_i \rceil$.
3. Let $\mathbf{t}_i \leftarrow \mathbf{t}_{i+1} - k_i \mathbf{b}_i$ and let $\mathbf{v}_i \leftarrow \mathbf{v}_{i+1} + k_i \mathbf{b}_i$.

Then $\mathbf{v} := \mathbf{v}_0 \sim D_{(c,0)+\Lambda(\mathbf{B}), \sigma, 0}$ is returned.

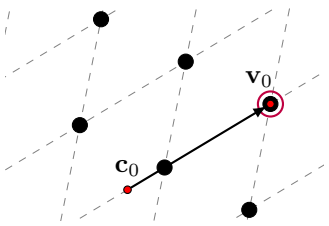
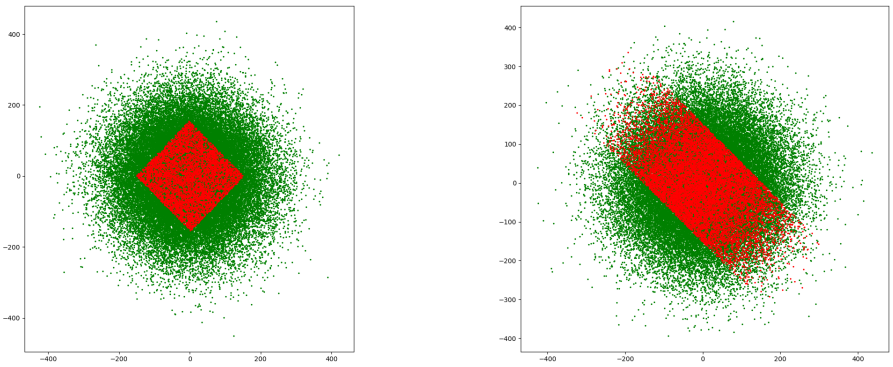


Figure 4: $\mathbf{c}_0 = \mathbf{v}_1 + x_0 \mathbf{b}_0$ is shifted to $\mathbf{v}_0 = \mathbf{v}_1 + k_0 \mathbf{b}_0$ rather than to the closest point

Although the vector $\mathbf{z} = (z_0, \ldots, z_{n-1})$ is correlated to the secret, its knowledge though side-channel analysis (as presented in Section 4.1) does not directly lead to $\mathbf{B}$, f, g F, G



(a) Signatures where $\mathbf{z} \in \{0,1\}^2$ are in red

(b) Signatures where $z_0 \in \{0,1\}$ are in red

Figure 5: Simplified 2D representation of the distributions of FALCON's signatures with two different filterings.

or even T. Note that one cannot revert the translation as the $\mathbf{t}_i$'s are unknown from the attacker.

However, the knowledge of $\mathbf{z}$ removes in some sense the probabilistic part of the sampling and thus an adaptation of the hidden parallelepiped attack as presented in Section 2.2 seems natural. To illustrate this, in Figure 5, we print some selected generated signatures, according to the value of $\mathbf{z}$.

On the left, we represent in green a representation in 2 dimensions of the distribution of $100,000$ FALCON signatures. We also represent in red the signatures where $\mathbf{z} \in \{0,1\}^2$. One can see that the parallelepiped is fully disclosed. However, since $n \in \{512, 1024\}$, it is not possible to filter the signature to force $\mathbf{z} \in \{0,1\}^n$ in these dimensions.

Nevertheless, when only one z_i is forced to 0 or 1 for all signatures, i.e., when signatures are filtered such as the corresponding output of algorithm 2 is 0 as presented on the right in Figure 5, the parallelepiped is not fully disclosed, it is more a deformed parallelepiped. But, one direction is preserved, we will see in Section 4.2.2 that it is enough to trigger an attack.

Number of filtered signatures For *each* index $i \in [0, n-1]$, we will apply a different filter keeping the traces such that $z_i \in [0,1]$ and discarding the other signature traces. Note that, even if one trace is discarded for an index, it can be kept for another index. Thus, all the traces are eventually used multiple times. Let us compute the number of discarded signatures for one index. Coefficients of f and g follow a centered discrete Gaussian whose standard deviation depends only on n and q, therefore the probability of $\mathbf{z}$ having one coordinate in $\{0,1\}$ is almost fixed by the parameters of the scheme. Indeed, FALCON parameters imply that $1.2915 < \sigma' < 1.8502$, thus $\mathbb{P}[z_i \in \{0,1\}] \approx \mathrm{erf}(\frac{\sqrt{2}}{2\sigma'}) \in [0.4111, 0.5613]$ for all $i \in [0, n-1]$. Thus, for each index, we keep between 41% and 56% of the signatures.

4.2.2 Applying the HPP

We aim at applying the HPP solver from [NR06, DN12] to the signatures fulfilling the condition $z_0 \in \{0,1\}$. Informally, the HPP solver searches for a point $\mathbf{w}$ on the n-dimensional unit sphere which minimizes the collinearity between $\mathbf{w}$ and all vectors in the sample pool. In our case, coordinates associated to the private basis are Gaussian in a quite large interval, except for coordinates where z_i is forced to 0 or 1 which are in a smaller interval.

Taking advantage of this truncated Gaussian, the HPP solver returns an approximation of the corresponding vector in the private basis. This situation is close but not directly equivalent to the deformed parallelepiped case as presented in Section 2.2. There are several differences between our signatures and the signatures used in [NR06, DN12] to run their attack.

1. First, the signature schemes attacked in [NR06, DN12] are using the round-off algorithm [Bab85, Bab86], while FALCON's trapdoor sampler is based on the fast Fourier nearest plane algorithm [DP16]. The first consequence is that the hidden parallelepiped that we can disclosed is not the private basis $\mathbf{B}$, but its GSO $\tilde{\mathbf{B}}$. Thus, the HPP solver cannot recover rows of $\mathbf{B}$ but rather rows of $\tilde{\mathbf{B}}$.

2. A second consequence of the use of a randomized variant of the nearest plane algorithm is the distribution of the signatures $\mathbf{s} := (\mathbf{t}-\mathbf{v})\cdot\mathbf{B}$. Let $x_i := \langle \mathbf{t}_i, \tilde{\mathbf{b}}_i \rangle / \left\|\tilde{\mathbf{b}}_i\right\|^2$, from [GPV08, (Lemma 4.4)] and [DP16, (Lemma 3)]:

$$\mathbf{s} = \sum_{i\in[n]} y_i \cdot \tilde{\mathbf{b}}_i \text{ where } y_i = z_i - x_i + \lfloor x_i \rfloor.$$

   Given that $z_i \in \{0,1\}$, we have $y_i \in (-1,1]$, but unlike in randomized round-off algorithms, in our case the distribution of y_i is not uniform over $(-1,1]$. There is no trivial way to describe the probability distribution of the y_i's given that $z_i \in \{0,1\}$, but it can be seen as the sum of two distributions, one being a uniform distribution over $(-1,1]$, and the other being a sum of small uniform distributions centered at 0 which seems "Gaussian". Since directions of Gaussian vectors are uniform at random from the unit sphere, these vectors have no impact on the minimums of the fourth moments considered in the gradient descent, but they add "useless" samples to take into account in the moment and gradient computation and thus increase the number of required signatures.

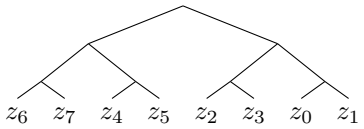


Figure 6: Sampling order of the z_i in dimension 8

3. Another consequence of the `ffSampling` algorithm is that the order in which coordinates are sampled is not the same order as the rows of $\tilde{\mathbf{B}}$, i.e. filtering the signatures with the condition $z_0 \in \{0,1\}$ will not provide information on $\tilde{\mathbf{b}}_0$. The sampling order, called "bit reversal order" is illustrated on Figure 6. For dimension 8, the z_i are then reordered as follows:

$$z_6 z_4 z_7 z_5 \; z_2 z_0 z_3 z_1$$

   Here, if one wants to recover $\tilde{\mathbf{b}}_0$, a filter according to $z_6 \in \{0,1\}$ is necessary. More generally, to recover the first row of $\tilde{\mathbf{B}}$ in dimension n we have to apply the filtering $z_{n-2} \in \{0,1\}$. If the z_i were sampled with a classical binary tree traversal order from left to right, the final order of the signature's coordinates would be merely the bit reversal order of i on $n/2$ bits. Let $(b_0, b_1, \ldots, b_{\ell-1})_2$ be the binary representation of an integer $k = \sum_{i=0}^{\ell-1} b_i \cdot 2^i$ the bit reversal of k on ℓ bits is $\mathsf{br}_\ell((b_0, b_1, \ldots, b_{\ell-1})_2) = (b_{\ell-1}, \ldots, b_1, b_0)_2$.

4.2.3 Correcting the noisy vectors

Since $\tilde{\mathbf{b}}_0 = \mathbf{b}_0 = (g_0, \ldots, g_{n-1}, -f_0, \ldots, -f_{n-1})$, this row is the most interesting to recover, as it allows to directly recover the private key. However, our attack only recovers approximations for $\mathbf{b}_0$, even if the approximation gets better with the number of signatures. If we consider only $\mathbf{b}_0$, we are able to obtain a good approximation of (f, g), but we can achieve better results by taking into account other rows.

The GSO of $\mathbf{B}$ is computed in a recursive way by a $\mathbf{LDL}^*$ decomposition. The $\mathbf{L}$ matrix is stored in a binary tree as it is computed. The leaves of this tree corresponds to the $\|\tilde{\mathbf{b}}_i\|$, with i following a bit reversal order on $2n$ bits. We have $\tilde{\mathbf{b}}_0 = \mathbf{b}_0$, and the closer we get to the center of the tree the more different the $\tilde{\mathbf{b}}_i$ are from the $\mathbf{b}_i$. However, we observed that the $\tilde{\mathbf{b}}_i$ corresponding to the four left-most leaves of the tree are still very close to the $\mathbf{b}_i$, that is very close to $(g, -f)$ up to a multiplication by a power of x (rotation of the row).

We know that $\tilde{\mathbf{b}}_n = (\frac{qf^*}{ff^*+gg^*}, \frac{qg^*}{ff^*+gg^*})$. Experimentally, we see that $\tilde{\mathbf{b}}_n$ is in fact very close to (f^*, g^*). This row corresponds to the right-most leave of the $\mathbf{LDL}^*$ tree. Similarly to the previous observation, the four right-most leaves of the tree are also very close to (f^*, g^*), up to a multiplication by a power of x. Thus, by computing an average vector from the 8 vectors previously described, we can improve our approximation of f, g, and experimentally we were able to divide by two the standard deviation of the error.

4.3 Step 3: Recovering the secret key from approximate (f, g)

The recovered vector from the HPP solver, denoted (f', g'), is an approximation of (f, g). From obtaining the correct key from (f', g'), a lattice reduction phase may be necessary. To estimate the remaining work for the full key recovery, we use the Leaky LWE/NTRU estimator tool introduced in [DDGR20] (see [DSDGR21] for the implementation). This tool is a variant of a tool introduced by Albrecht et al. [ACD+18]. The Leaky-LWE/NTRU framework uses the a posteriori distribution of (f, g) and the parameters n and q to embed the NTRU instance into a Distorted Bounded Distance Decoding instance. The a posteriori distribution of (f, g) is modeled as a multivariate Gaussian centered in (f', g') with a standard deviation provided experimentally.

We wrote a simple SageMath script to call the Leaky LWE / NTRU tool. For our very low values of the experimental standard deviation, we had to increase the precision of the intermediate computations inside the tool. The tool returns the estimated BKZ block-size β. This value gives an idea of the remaining work though the so-called cost models (See [ACD+18] for more details). In a nutshell, most models suggest a security increase of one bit every 2 to 4 increase of β. Note that the block size for a key recovery on Falcon-512 is estimated around $\beta = 480$.

4.4 Results

Experimental results for Step 1 Concerning the determination of the value of z^+, we achieved perfect results (100% accurate classification) using traces generated with ELMO without any extra noise. On the ChipWhisperer, achieving perfect accuracy was challenging. Fortunately, we only need perfect accuracy when determining $z^+ = 0$, the other values of z^+ are of no interest for our attack. Thus, we can strengthen the classification parameters such that no trace is wrongly classified as $z^+ = 0$ if it corresponds to $z^+ \neq 0$ (called "false positive"). This will induce some loss: some ambiguous traces corresponding to $z^+ = 0$ may not be classified as $z^+ = 0$ and they are not used for the attack. On the ChipWhisperer, the percentage of traces that are correctly classified is around 94%. This little loss should

slightly decrease the number of filtered signatures computed in Section 4.2.1. To avoid this loss, it is possible to simply generate more traces to compensate for the ambiguous discarded ones. One can note that better experimental results (100% accuracy) were achieved by [KH18] using more precise material (LeCroy HDO6104A oscilloscope at a sampling rate of 250 M/s), adding to the belief that the attack is actually practical.

Table 3: Percentage of traces that are correctly classified. We adapted the classification parameters such that there are no "false positives", i.e. no traces wrongly classified as $z^+ = 0$.

	Percentage of traces accurately classified
ELMO	100%
ChipWhisperer	94.2%
More efficient setup [KH18]	100%

Note that an ambiguous trace is always discarded for a specific index only, and can be used for another if the classification algorithm is successful (e.g. z^+ may be ambiguous for z_0 but not for z_1), which means that since we are targetting 8 different indexes all generated traces are likely to be used at least once.

Remark 1. For the rest of the experimental results, we assume a 100% accurate classification as obtained with ELMO or with [KH18]. The slight classification loss of the ChipWhisperer measurement would imply a small increase of the number of traces (+6%) but given the orders of magnitude for the number of traces, this increase does not change the final estimations.

Implementation of the Step 2 First of all, we remark that the algorithm proposed by [NR06, DN12] to solve HPP is highly parallelizable. Thus, we implemented a fully parallelized version, which mainly consists in launching a gradient descent on every CPU and waiting for the first one to succeed. We abort the descents if they take too long, ensuring that they do not fall into false positives, i.e. local minimums different from the rows of $\tilde{\mathbf{B}}$. Aborted gradient descents are launched again with a new random starting point on the unit sphere.

The signatures are generated with Falcon's official implementation assuming that the intermediate value z^+ is recovered though side-channels (see Section 4.1). Next, the parallelized HPP solver is run eight times on the signatures, each time with a different filtering to recover a different row of $\tilde{\mathbf{B}}$ as described in Section 4.2.3.

We improved the search in the following way. When we get a first approximation of a $\mathbf{b}_i$, the seven remaining ones are almost free: we can use the first found vector as a starting point for the gradient descent, resulting in very fast convergence (few dozens of iterations). The only costly step that remains is the computation of the Gram matrix.

We focus our experimental measurements on Falcon-512 parameters. We generated a database of 10 millions of signatures and the associated values of the z_i's. This pre-computation took one week of computations on a standard computer. The obtained database is around 40 Gigabytes. This database allowed us to extensively analyze the resources of the attack. We performed our parallel HPP attack on a server with 80 cores and were able to finish the gradient descent of the HPP within a few days or weeks depending on the number of traces. As detailed in [NR06, DN12], the convergence rate of the gradient descent is not precisely known and we cannot give accurate time estimation for a complete run.

Direct key recovery When the number of traces is high enough, there is no need for lattice reduction. For instance, we were able to correct our noisy vector and fully recover

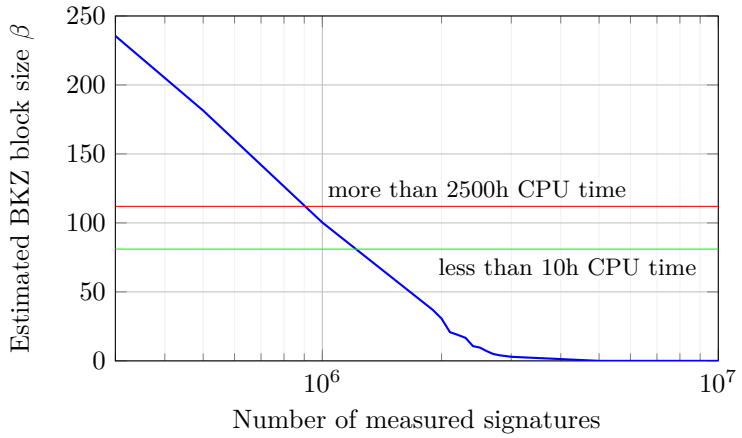


Figure 7: Work/Measurement trade-off for Falcon-512. We show the estimated block size β as a function of the generated signatures. By convention, $\beta = 0$ corresponds to the case where the secret key is exactly recovered with a simple rounding without need of lattice reduction. Recall that without attack, the necessary block size for Falcon-512 is estimated around $\beta = 480$. CPU time is taken from [ADH+19].

the key by mere rounding with 10 millions of signature measurements. Indeed, we obtain a standard deviation of the error $\sigma_{(f',g')-(f,g)} = 0.08$. The probability to get, for each of the 1024 coefficients of (f, g), an absolute error less than 0.5 is greater than 0.9999. This probability is lesser with only 5 millions traces but still overwhelming (greater than 0.99).

Step 3: Work/Measurement trade-off Instead of a direct recovery, an attacker may want to trade measurement for computation time and use Section 4.3 for finalizing the attack. In Figure 7, we provide the estimated BKZ block size as a function of the number of signature measurements. The graph is obtained from 16 attack measurements with a number of signatures belonging in $[300000, 10000000]$. From Figure 7, one can notice that 1.5 million traces would leave a realistic remaining computation for a standard computer. Thus, a trade-off for a powerful attacker would currently be at 1 million traces (based on [ADH+19]). More precisely, CPU time would be around 1000 hours with 1 million traces, and on [ADH+19] set-up this would result in less than 24 hours of wall-clock time. Attacks with less than 1 million traces are currently very costly and would probably require weeks of computation on powerful machines, but the induced theoretical loss of security can still raise concerns.

For generating the data in a minimal time, we used the following tweak: once a successful attack is obtained with a certain number of signatures e.g. 10000000, it is possible to speed-up the next experiments with different number of traces using the previously found vectors as a starting point for the gradient descent. The recovered value is still the correct one as it corresponds to a local minimum for the new number of traces.

Experiments on other n We also performed our attack on modified instances of Falcon with smaller n (32, 64, 128 and 256), several experiments are provided in appendix in Table 4. One interesting fact is that the attack behaves better when the n increases. Indeed, the standard deviation of the error between (f', g') and (f, g) decreases with the dimension. This may be explained by the fact that the norm $\|(f, g)\|$ is bounded by $1.17\sqrt{q}$ and is very close to this value for every n. The direct consequence is that, when n increases, the values of the coefficients of f, g decrease, easing the recovery. This behavior is favorable for

the attacker, since the complexity of correcting noisy vectors increases with the dimension.

For Falcon-1024, we did not generate a database of signature measurements and thus we cannot provide experimental data, though [DN12] claimed the number of required signatures to be polynomial in the dimension. We believe with confidence that, the behavior of the attack will be similar to a shift to the right of the data for Falcon-512 on Figure 7.

4.5 Countermeasure

In a nutshell, the demonstrated attack leverages a particular information on z^+ obtained by side-channel analysis on the `BaseSampler`. The information is binary: either the 8 most significant bits of the register are set to 0 or they are all set to 1. In the first case, z^+ is incremented and in the second case it is not, leading to the knowledge of $z^+ = 0$ or $z^+ \neq 0$. This binary information is crucial to transform a practically impossible attack into a (quite costly but) doable attack by filtered HPP resolution (see Section 4.2.1).

In this section, we provide a simple trick to practically lower the Hamming weight gap and thus mitigate the attack. Let us focus on the point of entry of the side-channel attack: the comparison $[\![u < \mathrm{RCDT}[i]]\!]$ at line 4 of `BaseSampler`. As previously stated in Subsection 4.1, this comparison is implemented in Falcon's official implementation as three successive subtractions between two 24-bit unsigned integer stored in 32-bit registers. Of these three successive subtractions, the precise point of entry of the attack is the last one: the subtraction of the 24 MSB of u, denoted $\overline{u}$, and the 24 MSB of $\mathrm{RCDT}[i]$, denoted $\overline{\mathrm{RCDT}[i]}$. We now propose to modify this last comparison as presented in algorithm 3.

Algorithm 3: Proposed countermeasure to mitigate our attack. This pseudocode corresponds to the last comparison during the computation of $[\![u < \mathrm{RCDT}[i]]\!]$ at line 4 of `BaseSampler`

Input : Two 24-bit variables $\overline{u}$ and $\overline{\mathrm{RCDT}[i]}$ stored in 32 bit registers.
a bit c carrying the result of the comparison of both least significant registers (corresponding to the 48 least significant bits of $u - \mathrm{RCDT}[i]$).
Output : 1 if $\overline{\mathrm{RCDT}[i]} + c > \overline{u}$ and 0 is $\overline{u} \leq \overline{\mathrm{RCDT}[i]} + c$

1 $b \leftarrow$ `0xffffff`
2 $b := b - \overline{u} + \overline{\mathrm{RCDT}[i]} + c$
3 **return** $b \gg 24$

Proposition 1. *In the proposed modification, the Hamming weight difference of the internal register between output* 1 *and output* 0 *is at most* 1*, contrary to* 8 *in the original solution. Thus, assuming that the leakage is correlated to Hamming weight gap, the recovery of information becomes significantly more difficult.*

Indeed, in algorithm 3, the underflow is replaced by an "overflow". This overflow is not a physical one, as the 24-bits values will overflow on 25 bits in 32 bits register. The consequence is that, in case of "overflow", of the 8 MSB, only one will be set to 1, the 25-th bit. Thus, the two different outputs of the subtraction will only have an average difference of Hamming weight of 1. This is to put in perspective with the difference of 8 in the actual algorithm. Theoretically, the leakage is thus divided by at least 8, and experimentally we observed that it is no longer possible to classify the traces.

The presented trick consists in a small modification of the C code and does not have any noticeable impact on the performance, while provable masking (e.g. [EFG+21]) would have a cost of $O(Td^{3/2})$ with d the masking order and T the size of the RCDT. However, it is important to note that our proposition obviously provides a weaker assurance compared to a provable masking in the ISW model [ISW03]. For a theoretical security assurance, we

would recommend to apply a provable masked implementations of RCDT-based Gaussian samplers as introduced in [GR19, BBE+19], or more recently in [EFG+21]. Applying one of these countermeasures would prevent our single trace analysis but the cost could be prohibitive in certain applications. In some cases, our trick could possibly be considered as a sufficient mitigation accounting the cost of our attack in terms or resources and measurements.

5 Conclusion

In this article we have exhibited two leaking operations of Falcon and performed practical side-channel attacks on both leakages. While the former leakage was already known and our attacks is no more than an improvement of the existing one, the latter leakage was unexploited until now. We have shown that an attack using the hidden parallelepiped from [NR06, DN12] can be achieved using side-channel information to select good executions, that is execution for which one dimension of the parallelepided is unchanged.

The resulting attack is practical and was performed on Falcon-512 reference implementation using traces generated with the ELMO simulator for Arm Cortex-M0 processor and ChipWhisperer-Lite for Arm Cortex-M4 processor. Since all publicly available Falcon implementations, except the AVX2-optimized, are using the same `BaseSampler` as targeted in this paper and in [KH18], our attack applies to all these implementations in a similar way, including *clean* PQClean[3] and pqm4[4] implementations.

While the existence of these attacks may be unsurprising because the authors of Falcon do not claim any side-channel protection, these new attacks highlight the need for side-channel protection by pointing out the vulnerable parts and quantifying the resources of the attacks.

Future works Several other side-channel attacks could be performed on Falcon with a more powerful threat model. Concerning the trapdoor sampler, we only performed SPA on the `BaseSampler`. A template attack could be performed on the `SamplerZ`, targeting an operation involving the returned value of `BaseSampler`, as there are only 18 possible values. If the attack of [FKT+20] went to be adapted in a way that accepts approximate outputs, one could use the values sampled by the `BaseSampler` to compute the standard deviations and recover the key.

[3] https://github.com/PQClean/PQClean
[4] https://github.com/mupq/pqm4

A Application of Section 4's attack on smaller n

Table 4: We represent the standard deviation of the error vector $(f' - f, g' - g)$ for various number of signatures and parameter n. In addition, we show the necessary block size β of BKZ in the form $\sigma_{(f'-f,g'-g)} : \beta$.

number of signatures \ n	32	64	128	256
50 000	0.98 : 2	0.84 : 2		
100 000		0.66 : 2		
150 000		0.62 : 2		
200 000			0.75 : 9	
250 000			0.73 : 9	
300 000			0.71 : 8	
500 000			0.63 : 7	0.32 : 50
1 000 000				0.22 : 4

B SCA on the `SamplerZ`: about the applicability of [FKT+20]

In [FKT+20], Fouque *et al.* proposed a theoretical attack against Falcon, exploiting the `SamplerZ` which was then vulnerable to timing attacks. The algorithm performs rejection sampling, and the standard deviations used could be leaked through time measurements. The standard deviations are sensitive data, because they are the Gram-Schmidt norms of the private basis $\mathbf{B}$. The authors of [FKT+20] propose an algorithm `TowerRecovery` to recover the private key from these Gram-Schmidt norms.

Thanks to the work of [HPRR20], `SamplerZ` is now isochronous and no longer vulnerable to timing attacks. However, we have seen previously in Section 4.1 that its subroutine `BaseSampler` is vulnerable to power analysis, so that an attacker can retrieve all the values returned by this algorithm. With enough samples from `BaseSampler`, we can compute the standard deviations with arbitrary precision and the attack from [FKT+20] could be considered again.

The authors of [FKT+20] claimed that they had no practical results in the case of Falcon because their algorithm requires exact inputs while side-channel attacks only provide approximate values. However, we show in Section B that the needed precision on the standard deviations makes the attack unfeasible even with a powerful side-channel attacker.

Lemma 2. *Successfully running `TowerRecovery` on Falcon-512 requires more than 6 000 bits of precision for the values of the λ_i.*

Let λ'_i the approximate value of the standard deviation λ_i such that $\lambda_i = \lambda_i + \epsilon_i \lambda_i$. The expected input of the `TowerRecovery` algorithm are the m_i such that $m_i = \prod_{j=0}^{i} \lambda_j$. We have

$$\begin{aligned} m'_1 &= \lambda'_0 \cdot \lambda'_1 \\ &= (\lambda_0 + \epsilon_0\lambda_0)(\lambda_1 + \epsilon_1\lambda_1) \\ &= \lambda_0\lambda_1 + \epsilon_0\lambda_0\lambda_1 + \epsilon_1\lambda_0\lambda_1 + \epsilon_0\epsilon_1\lambda_0\lambda_1 \\ &= m_1 + (\epsilon_0 + \epsilon_1 + \epsilon_0\epsilon_1)m_1 \\ m'_n &= m_n + (\sum_{i=0}^{n} \epsilon_i)m_n + O(\epsilon^2)m_n. \end{aligned}$$

The m_i are supposed to be integers while the λ_i are reals. Thus if we only have approximate values of λ_i, we can recover the correct m_i by rounding only if $|m'_i - m_i| < 0.5 = 2^{-1}$, which requires

$$m_n \sum_{i=0}^{n} \epsilon_i < 0.5. \tag{1}$$

However,the λ_i are of the same magnitude of $q = 12289 \approx 2^{13.5}$, and thus the m_i are of order $2^{13.5n}$. From 1 we can deduce that $\sum_{i=0}^{n} \epsilon_i < 2^{-1-13.5n}$.

If we consider that all the errors e_i are approximately equal to an error ϵ which is the error $\epsilon = 2^{-p}$ due to the precision p of the λ_i, we have $\epsilon < \frac{2^{-1-13.5n}}{n}$. In the case $n = 512$, we finally have $\epsilon < 2^{-1-13.5\cdot 512-9}$ which implies $p > 6000$. Since the standard deviations are only stored in double precision (i.e. with 53 bits of precision) in FALCON's implementation, this attack cannot succeed no matter how these values are retrieved.

References

[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, Heidelberg, September 2018.

[ADH+19] Martin R. Albrecht, Léo Ducas, Gottfried Herold, Elena Kirshanova, Eamonn W. Postlethwaite, and Marc Stevens. The general sieve kernel and new records in lattice reduction. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part II*, volume 11477 of *LNCS*, pages 717–746. Springer, Heidelberg, May 2019.

[Bab85] László Babai. On lovász' lattice reduction and the nearest lattice point problem (shortened version). In Kurt Mehlhorn, editor, *STACS 85, 2nd Symposium of Theoretical Aspects of Computer Science, Saarbrücken, Germany, January 3-5, 1985, Proceedings*, volume 182 of *Lecture Notes in Computer Science*, pages 13–20. Springer, 1985.

[Bab86] László Babai. On lovász' lattice reduction and the nearest lattice point problem. *Comb.*, 6(1):1–13, 1986.

[BBE+18] Gilles Barthe, Sonia Belaïd, Thomas Espitau, Pierre-Alain Fouque, Benjamin Grégoire, Mélissa Rossi, and Mehdi Tibouchi. Masking the GLP lattice-based signature scheme at any order. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 354–384. Springer, Heidelberg, April / May 2018.

[BBE+19] Gilles Barthe, Sonia Belaïd, Thomas Espitau, Pierre-Alain Fouque, Mélissa Rossi, and Mehdi Tibouchi. GALACTICS: Gaussian sampling for lattice-based constant- time implementation of cryptographic signatures, revisited. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 2147–2164. ACM Press, November 2019.

[BDE+18] Jonathan Bootle, Claire Delaplace, Thomas Espitau, Pierre-Alain Fouque, and Mehdi Tibouchi. LWE without modular reduction and improved side-channel attacks against BLISS. In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018, Part I*, volume 11272 of *LNCS*, pages 494–524. Springer, Heidelberg, December 2018.

[BHLY16] Leon Groot Bruinderink, Andreas Hülsing, Tanja Lange, and Yuval Yarom. Flush, gauss, and reload - A cache attack on the BLISS lattice-based signature scheme. In Benedikt Gierlichs and Axel Y. Poschmann, editors, *CHES 2016*, volume 9813 of *LNCS*, pages 323–345. Springer, Heidelberg, August 2016.

[BP18] Leon Groot Bruinderink and Peter Pessl. Differential fault attacks on deterministic lattice signatures. *IACR TCHES*, 2018(3):21–43, 2018. https://tches.iacr.org/index.php/TCHES/article/view/7267.

[CD20] André Chailloux and Thomas Debris-Alazard. Tight and optimal reductions for signatures based on average trapdoor preimage sampleable functions and applications to code-based signatures. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *PKC 2020, Part II*, volume 12111 of *LNCS*, pages 453–479. Springer, Heidelberg, May 2020.

[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, Heidelberg, August 2020.

[DDLL13] Léo Ducas, Alain Durmus, Tancrède Lepoint, and Vadim Lyubashevsky. Lattice signatures and bimodal Gaussians. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 40–56. Springer, Heidelberg, August 2013.

[DN12] Léo Ducas and Phong Q. Nguyen. Learning a zonotope and more: Cryptanalysis of NTRUSign countermeasures. In Xiaoyun Wang and Kazue Sako, editors, *ASIACRYPT 2012*, volume 7658 of *LNCS*, pages 433–450. Springer, Heidelberg, December 2012.

[DP16] Léo Ducas and Thomas Prest. Fast fourier orthogonalization. In *Proceedings of the ACM on International Symposium on Symbolic and Algebraic Computation*, ISSAC '16, page 191–198, New York, NY, USA, 2016. Association for Computing Machinery.

[DSDGR21] Dana Dachman-Soled, Léo Ducas, Huijing Gong, and Mélissa Rossi. Lwe with side information: Attacks and concrete security estimation. Implementation, accessed in August 2021. https://github.com/lducas/leaky-LWE-Estimator.

[EFG+21] 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. Cryptology ePrint Archive, Report 2021/1486, 2021. https://ia.cr/2021/1486.

[EFGT17] Thomas Espitau, Pierre-Alain Fouque, Benoît Gérard, and Mehdi Tibouchi. Side-channel attacks on BLISS lattice-based signatures: Exploiting branch tracing against strongSwan and electromagnetic emanations in microcontrollers. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 1857–1874. ACM Press, October / November 2017.

[FKT+20] Pierre-Alain Fouque, Paul Kirchner, Mehdi Tibouchi, Alexandre Wallet, and Yang Yu. Key recovery from Gram-Schmidt norm leakage in hash-and-sign signatures over NTRU lattices. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part III*, volume 12107 of *LNCS*, pages 34–63. Springer, Heidelberg, May 2020.

[GGH97] Oded Goldreich, Shafi Goldwasser, and Shai Halevi. Public-key cryptosystems from lattice reduction problems. In Burton S. Kaliski Jr., editor, *CRYPTO'97*, volume 1294 of *LNCS*, pages 112–131. Springer, Heidelberg, August 1997.

[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.

[GR19] François Gérard and Mélissa Rossi. An efficient and provable masked implementation of qtesla. In Sonia Belaïd and Tim Güneysu, editors, *Smart Card Research and Advanced Applications - 18th International Conference, CARDIS 2019, Prague, Czech Republic, November 11-13, 2019, Revised Selected Papers*, volume 11833 of *Lecture Notes in Computer Science*, pages 74–91. Springer, 2019.

[HHP+03] Jeffrey Hoffstein, Nick Howgrave-Graham, Jill Pipher, Joseph H. Silverman, and William Whyte. NTRUSIGN: Digital signatures using the NTRU lattice. In Marc Joye, editor, *CT-RSA 2003*, volume 2612 of *LNCS*, pages 122–140. Springer, Heidelberg, April 2003.

[HPRR20] James Howe, Thomas Prest, Thomas Ricosset, and Mélissa Rossi. Isochronous gaussian sampling: From inception to implementation. In Jintai Ding and Jean-Pierre Tillich, editors, *Post-Quantum Cryptography - 11th International Conference, PQCrypto 2020*, pages 53–71. Springer, Heidelberg, 2020.

[ISW03] Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 463–481. Springer, Heidelberg, August 2003.

[KA21] Emre Karabulut and Aydin Aysu. Falcon down: Breaking falcon post-quantum signature scheme through side-channel attacks. Cryptology ePrint Archive, Report 2021/772, 2021. https://ia.cr/2021/772.

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

[Kle00] Philip N. Klein. Finding the closest lattice vector when it's unusually close. In David B. Shmoys, editor, *11th SODA*, pages 937–941. ACM-SIAM, January 2000.

[MGTF19] Vincent Migliore, Benoît Gérard, Mehdi Tibouchi, and Pierre-Alain Fouque. Masking Dilithium - efficient implementation and side-channel evaluation. In

Robert H. Deng, Valérie Gauthier-Umaña, Martín Ochoa, and Moti Yung, editors, *ACNS 19*, volume 11464 of *LNCS*, pages 344–362. Springer, Heidelberg, June 2019.

[MHS+19] 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, Report 2019/478, 2019. https://eprint.iacr.org/2019/478.

[MOW17] David McCann, Elisabeth Oswald, and Carolyn Whitnall. Towards practical tools for side channel aware software engineering: 'grey box' modelling for instruction leakages. In Engin Kirda and Thomas Ristenpart, editors, *USENIX Security 2017*, pages 199–216. USENIX Association, August 2017.

[NR06] Phong Q. Nguyen and Oded Regev. Learning a parallelepiped: Cryptanalysis of GGH and NTRU signatures. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 271–288. Springer, Heidelberg, May / June 2006.

[OC14] Colin O'Flynn and Zhizhang (David) Chen. Chipwhisperer: An open-source platform for hardware embedded security research. In Emmanuel Prouff, editor, *Constructive Side-Channel Analysis and Secure Design - 5th International Workshop, COSADE 2014, Paris, France, April 13-15, 2014. Revised Selected Papers*, volume 8622 of *Lecture Notes in Computer Science*, pages 243–260. Springer, 2014.

[PBY17] Peter Pessl, Leon Groot Bruinderink, and Yuval Yarom. To BLISS-B or not to be: Attacking strongSwan's implementation of post-quantum signatures. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 1843–1855. ACM Press, October / November 2017.

[PFH+20] 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, 2020. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions.

[PP19] Thomas Pornin and Thomas Prest. More efficient algorithms for the NTRU key generation using the field norm. In Dongdai Lin and Kazue Sako, editors, *PKC 2019, Part II*, volume 11443 of *LNCS*, pages 504–533. Springer, Heidelberg, April 2019.

[RJH+18] Prasanna Ravi, Mahabir Prasad Jhanwar, James Howe, Anupam Chattopadhyay, and Shivam Bhasin. Side-channel assisted existential forgery attack on Dilithium - A NIST PQC candidate. Cryptology ePrint Archive, Report 2018/821, 2018. https://eprint.iacr.org/2018/821.

[RJH+19] Prasanna Ravi, Mahabir Prasad Jhanwar, James Howe, Anupam Chattopadhyay, and Shivam Bhasin. Exploiting determinism in lattice-based signatures: Practical fault attacks on pqm4 implementations of NIST candidates. In Steven D. Galbraith, Giovanni Russello, Willy Susilo, Dieter Gollmann, Engin Kirda, and Zhenkai Liang, editors, *ASIACCS 19*, pages 427–440. ACM Press, July 2019.