# Profiling Side-Channel Attacks on Dilithium: A Small Bit-Fiddling Leak Breaks It All

Soundes Marzougui[1†], Vincent Ulitzsch[2†], Mehdi Tibouchi[3] and Jean-Pierre Seifert[4]

[1] Technical University Berlin, Berlin, Germany, soundes.marzougui@tu-berlin.de
[2] Technical University Berlin, Berlin, Germany, v.ulitzsch@campus.tu-berlin.de
[3] NTT Secure Platform Laboratories, Tokyo, Japan, mehdi.tibouchi.br@hco.ntt.co.jp
[4] Technical University Berlin, Berlin, Germany, Jean-Pierre.Seifert@external.telekom.de

**Abstract.** We present an end-to-end (equivalent) key recovery attack on the Dilithium lattice-based signature scheme, one of the top contenders in the NIST postquantum cryptography competition. The attack is based on a small side-channel leakage we identified in a bit unpacking procedure inside Dilithium signature generation. We then combine machine-learning based profiling with various algorithmic techniques, including least squares regression and integer linear programming, in order to leverage this small leakage into essentially full key recovery: we manage to recover, from a moderate number of side-channel traces, enough information to sign arbitrary messages. We confirm the practicality of our technique using concrete experiments against the ARM Cortext-M4 implementation of Dilithium, and verify that our attack is robust to real-world conditions such as noisy power measurements. This attack appears difficult to protect against reliably without strong side-channel countermeasures such as masking of the entire signing algorithm, and underscores the necessity of implementing such countermeasures despite their known high cost.

**Keywords:** Dilithium · Lattice-based cryptography · Machine learning · Profiling attacks · Side-channel analysis · Integer linear programming

## 1 Introduction

The plausible advent of general-purpose quantum computers in the coming decades is a mounting threat to currently deployed public-key cryptography, and particularly digital signatures such as RSA signatures and ECDSA, since those schemes are broken by Shor's algorithm [Sho97]. It therefore appears increasingly important to design and implement quantum-resistant cryptographic schemes that are suitable for real-world deployment, whether in terms of security, performance or practicality of implementation. This is the aim of the ongoing NIST-led process to evaluate and standardize post-quantum primitives for public-key encryption, key encapsulation and signatures [Cen20].

Post-quantum digital signatures can be based on a variety of cryptographic assumptions, ranging from collision-resistant hash functions to the hardness of solving large multivariate quadratic systems or finding isogenies between supersingular elliptic curves. Prominently among those are assumptions related to Euclidean lattices: two of the three finalist digital signatures in the NIST competition, Dilithium and Falcon, are lattice-based schemes. Interestingly, those two schemes are instances of the two different design frameworks for lattice-based signatures: the hash-and-sign approach for Falcon and the Fiat–Shamir with aborts paradigm for Dilithium.

† These authors contributed equally to this work.

Hash-and-sign signatures based on lattices have a longer history, starting with GGH and NTRUSign [GGH97, HHGP+03], but the early constructions were quickly found to be insecure [Ngu99, NR06], as each released signature would leak a small amount of information of the signing trapdoor. The first successful construction dates back to a seminal paper of Gentry, Peikert and Vaikuntanathan (GPV) [GPV08], who showed that a careful use of lattice Gaussian sampling could eliminate the statistical leaks that plagued earlier attempts. While the GPV scheme itself was not very practical, later developments improved the efficiency of lattice-based hash-and-sign signatures [MP12, DLP14, CGM19, EFG+21], and Falcon is in particular quite fast and compact. However, like almost all schemes in this family, it retains the crucial reliance on lattice Gaussian sampling for its security. This makes it tricky to implement, validate, and protect against side-channels.

In contrast, the Fiat–Shamir with aborts paradigm, introduced by Lyubashevsky in [Lyu09], tends to give rise to simpler schemes, that have a structure similar to traditional Fiat–Shamir schemes like Schnorr signatures, but with a twist to deal with the fact that there is no uniform distribution on a lattice. Typically, those schemes only rely one one-dimensional discrete Gaussian sampling (as is the case for [DDLL13]; this is already considerably simpler than the lattice Gaussian sampling used in hash-and-sign signatures) or no Gaussian sampling at all (as is the case for [GLP12]). Dilithium, in particular, belongs to the second category, and the submission document states that eliminating the reliance on Gaussian sampling is a deliberate design choice aiming at a greater ease of *secure implementation*, particularly against side-channel attacks [LDK+20]. Indeed, Gaussian sampling-based schemes have been the target of numerous devastating side-channel attacks.

Side-channel resilience has also been a point of focus of the NIST standardization process from the get go. Indeed, the original call for proposals states that

> *Schemes that can be made resistant to side-channel attacks at minimal cost are more desirable than those whose performance is severely hampered by any attempt to resist side-channel attacks.*

More recently, the latest PQC summary document (NISTIR 8309) notes that

> *NIST hopes to see more and better data for performance in the third round. This performance data will hopefully include implementations that protect against side-channel attacks, such as timing attacks, power monitoring attacks, fault attacks, etc.*

It is thus of particular importance to understand to what extent and at what cost the finalists, including Dilithium, can be protected against side-channel attacks.

The reference implementation of Dilithium claims security against timing attacks, and as mentioned earlier, the scheme has been designed with particular consideration for side-channel resilience, but it does not necessarily offer protection against more powerful attacks like differential power analysis. In fact, some amount of side-channel leakage has been demonstrated in certain parts of the Dilithium signing algorithm. They have not been turned into a complete end-to-end attack so far, however, so implementers may be reluctant to embrace strong side-channel countermeasures like masking.

Indeed, while Migliore et al. [MGTF19] have described how Dilithium could be masked at any order, and hence protected against arbitrary-order power analysis attacks, this countermeasures comes at considerable cost. In fact, Migliore et al. restrict their practical evaluation to the masking of a simplified version of Dilithium, instantiated with a power-of-two modulus instead of the original prime modulus (which makes the countermeasure much less costly), and even so, they find that the first-order masked signature scheme is around five times slower than the unmasked scheme.

**Contributions.** In this work, we present a profiling-based power analysis attack on the Cortex-M4 implementation of Dilithium achieving essentially full key recovery. To the best of our knowledge, this constitutes the first end-to-end attack on this signature scheme.

To describe our approach in a bit more details, we need to recall how Dilithium signature generation proceeds at a high level. It is roughly similarly to Schnorr signatures in cyclic group $\mathbb{G} = \langle g \rangle$ of order $p$, which work by sampling some uniform random value $y$ modulo $p$, hashing the message $\mu$ together with $g^y$ to obtain $c = H(\mu, g^y)$, and returning the signature as $(z, c)$ with $z = y + cs$ (where $s$ is the private key and $a = g^s$ the public key). Verification is then carried out by checking whether $c = H(\mu, g^z/a^s)$. Accordingly, Dilithium proceeds as follows:

1. sample $\mathbf{y}$ as a random module element with uniform coefficients in some interval;
2. hash the message $\mu$ together with $\mathbf{Ay}$ ($\mathbf{A}$ a public matrix) to obtain $c = H(\mu, \mathbf{Ay})$, a ring element with sparse 0/1 coefficients;
3. let $\mathbf{z} = \mathbf{y} + c\mathbf{s}_1$ (where $\mathbf{s}_1$ is the first part of the secret key);
4. return the signature as $(\mathbf{z}, c)$ provided that a certain rejection condition is met (mostly stating that $\mathbf{z}$ has its coefficients in a small interval) and reject otherwise.

The starting point of our attack is identifying a power side-channel leakage in the first step: the generation of the random $\mathbf{y}$, which is carried out by constructing a certain random string, and expanding it into a module element (a vector of polynomials) using bit fiddling operations. Then, machine-learning assisted profiling lets us predict with relatively high accuracy whether a given polynomial coefficient in one of the components of $\mathbf{y}$ is zero or not. If we could do so with perfect accuracy, then the recovery of $\mathbf{s}_1$ would be a simple matter of linear algebra: indeed, for each zero coefficient $\mathbf{y}_i$ identified in this way, we get the relation $\mathbf{z}_i = 0 + (c\mathbf{s}_1)_i$, and since both $\mathbf{z}_i$ and $c$ are known, this gives a linear relation on the coefficients of $\mathbf{s}_1$. Collecting sufficiently many of those reveals $\mathbf{s}_1$ in full.

However, the machine learning predictor is not perfectly accurate in practice, so two challenges naturally present themselves. On the one hand, we want to improve the accuracy of the predictor as much as possible, which is mainly a matter of improving the profiling phase of the attack (mainly by adjusting the hyperparameters of our ML model, and by carefully choosing the learning data). On the other hand, we need to solve a *noisy* linear system involving the secret key while minimizing the number of required traces. We consider several possible approaches to do so, and find that expressing the problem as an integer linear program offers the best results in practice for our particular setting.

We validate our attack by mounting it on the full Cortex-M4 implementation of Dilithium in a realistic attack setting, by capturing actual, noisy power traces and using them to train and apply our ML model, and then recovery the full vector $\mathbf{s}_1$. Although $\mathbf{s}_1$ is not the entire secret key, it has been shown in previous work [RJH+18, GBP18] that knowing $\mathbf{s}_1$ suffices to sign essentially arbitrary messages, so its recovery is essentially as good as full key recovery.

As an aside, we note that Dilithium has two variants that differ in the way the vector $\mathbf{y}$ is generated: one is deterministic (in the spirit of EdDSA) and the other is probabilistic (aiming at thwarting certain classes of implementation and fault attacks, in the spirit of signatures with hedged randomness [AOTZ20]). However, our attack applies to both variants, since the bit-fiddling step we target is common to both of them. Moreover, our attack extracts information about the vector $y$ per signature, from a single trace. Thus, while sampling a different vector $y$ per execution does indeed thwart attacks that rely on aggregating traces, the randomization does not yield additional protection in our case.

All in all, this paper demonstrates a new power side-channel leakage in Dilithium, and leverages into an ML-assisted profiling attack that achieves essentially full key recovery. The attack relies on a novel, non trivial algorithmic decoding step based on integer linear programming, and is validated against the Cortex-M4 implementation of Dilithium.

**Related work.** Side-channel attacks were introduced by Kocher et al. [KJJ99] and are considered one of the main threat against cryptographic algorithms deployed in embedded systems, and have also been demonstrated on larger devices like smartphones and desktop computers, as well as remotely over the internet. In a side-channel attack, an attacker does not exploit mathematical weaknesses or invalid behavior of an implementation but uses physical information to reveal secret data. The attacker measures physical information (for instance, power consumption, electromagnetic emanations, elapsed time, etc.) during the execution of cryptographic operations in order to gain information about sensitive internal data, including secret keys.

Although no end-to-end side-channel analysis of Dilithium has been presented so far to the best of our knowledge, several earlier works did demonstrate potential side-channel leakage in various parts of the signing algorithm.

In [FDK20], P. Fournaris et al. present a correlation-based differential side-channel analysis of the polynomial multiplication operation $c \cdot \mathbf{s}_1$ during signature generation. The analysis considers various multiplication algorithms (schoolbook multiplication, NTT or sparse multiplier), and argues that mounting a correlation power analysis is *in principle* feasible with all of them. The authors also verify that the required leakage is detectable on power traces of Dilithium signature generation implemented on a commercial off-the-shelf embedded system board. However, they do not mount the full actual attack, evaluate the number of traces it would actually require in practice, or quantify the sensitivity to measurement noise.

A more detailed theoretical analysis along the same lines in carried out by Ravi et al. [RJH+18], although only for schoolbook multiplication and sparse multiplication. The paper discusses the feasibility of key recovery in several idealized leakage models, and does some extent of evaluation of the effect of leakage noise. However, they again do not attempt to actually mount an attack on an actual implementation. Moreover, the NTT, which is used in the actual implementation of Dilithium, is not considered.

Aside from polynomial-multiplication-related vulnerabilities, several points of interest are identified by Migliore et al. [MGTF19] as leaking sensitive values, particularly in the rounding functions (low and high bits calculations), and the rejection sampling executed during signature generation. Using test vector leakage assessment on a Cortex-M3 implementation of Dilithium, they show that these functions do present detectable leakage, and can thus, again *in principle*, be exploited by differential power analysis. Again, however, they do not mount an actual attack (and only use those leakage points to evaluate the side-channel resistance of the *masked* implementation presented in the paper.

In a different direction, Groot Bruinderink et al. [GBP18] present a fault injection attack on Dilithium which aims at creating a nonce-reuse scenario. Groot Bruinderink et al. demonstrate that single random faults in the Dilithium signature can lead to nonce reuse. They perform their attack on an ARM Cortex-M4 microcontroller and prove that the success probability for all fault scenarios can reach 91%.

An interesting feature of [RJH+18] and [GBP18] is that they only consider the recovery of the main secret key element $\mathbf{s}_1$, and therefore discuss how signing is possible using just that knowledge. The same techniques can be used in our setting as well.

**Organization of the paper.** The remainder of this paper is organized in six sections. In Sec. 2, we give preliminaries on the Dilithium scheme and the use of machine-learning classifier in our side-channel attack. In Sec. 3, we give an overview on the attack and the attacker's model. Likewise, we analyze the weakness of Dilithium against machine-learning power side-channel analysis in Sec. 4. In Sec. 5, we present a detailed mathematical description of the attack strategy and its resistance against noisy measurements. Then, we describe our experimental setup for the profiling and the attack phases in Sec. 6 and present the results of our attack. We conclude the paper with Sec. 7 where we discuss possible countermeasures against our attack.

## 2 Background

### 2.1 Notation

For any integer $q$, the ring $\mathbb{Z}_q$ is represented by the set $[-q/2, q/2) \cap \mathbb{Z}$. We denote $\mathbb{Z}_q[X] = (\mathbb{Z}/q\mathbb{Z}[X])$ as the set of polynomials with integer coefficients modulo $q$. We define $R_q = \mathbb{Z}_q[X]/(X^n + 1)$, the ring of polynomials with integer coefficients modulo $q$, reduced by the cyclotomic polynomial $X^n + 1$. For any $\alpha < q$, $R_\alpha$ refers to the set of polynomials with coefficients in $\left[-\frac{\alpha}{2}, +\frac{\alpha}{2}\right]$. By default, we use the $L^\infty$-norm: $\|\mathbf{v}\|_\infty = \max_i |v_i|$.

By $vec(\cdot)$ we denote the function which maps a vector of polynomials in the ring $R_q$ to the vector obtained by concatenating all coefficients of the respective polynomials.

In the following expressions, all polynomial operations are performed in $R_q$. Bold lowercase letters represent vectors with coefficients in $R$ or $R_q$ . Bold uppercase letters are matrices. Similarly, vector coefficients are represented by roman lowercase letters.

### 2.2 Lattices

A lattice $\Lambda$ is a discrete subgroup of $\mathbb{R}^n$ such that given $m \leq n$ are linearly independent vectors $\mathbf{b}_1, \ldots, \mathbf{b}_m \in \mathbb{R}^n$, the lattice $\Lambda = \Lambda(\mathbf{b}_1, \ldots, \mathbf{b}_m)$ is the set of all integer linear combinations of the $\mathbf{b}_i$'s, i.e.,

$$\Lambda(\mathbf{b_1}, ..., \mathbf{b_m}) = \left\{ \sum_{i=1}^{m} x_i \mathbf{b_i} \;\middle|\; x_i \in \mathbb{Z} \right\},$$

where $\mathbf{b}_1, \ldots, \mathbf{b}_m$ is the basis of $\Lambda$ and $m$ is the rank. In this paper, we consider full-rank lattices, i.e., with $m = n$. An integer lattice is a lattice for which the basis vectors are in $\mathbb{Z}^n$. Usually, we consider elements modulo $q$, i.e., the basis vectors and coefficients, are taken from $\mathbb{Z}_q$.

### 2.3 Learning With Errors

The Learning with Errors problem (LWE), a generalization of the classic Learning Parity with Noise problem (LPN), was introduced by Oded Regev [BDK+20].

**Definition.** *Let $n$, $q$ be positive integers, and $\chi$ be a distribution over $\mathbb{Z}$ . For $\mathbf{s} \in \mathbb{Z}_q^n$, the LWE distribution $A_{s,\chi}$ is the distribution over $\mathbb{Z}_q^n \times \mathbb{Z}_q$ obtained by choosing $\mathbf{a} \in \mathbb{Z}_q^n$ uniformly at random and an integer error $e \in \mathbb{Z}$ from $\chi$. The distribution outputs the pair $(\mathbf{a}, \langle \mathbf{a}, \mathbf{s} \rangle + e \mod q) \in \mathbb{Z}_q^n \times \mathbb{Z}q$.*

There are two important computational LWE problems:

- The *search problem* is to recover the secret $\mathbf{s} \in \mathbb{Z}_q^n$ given a certain number of samples are drawn from the LWE distribution $A_{\mathbf{s},\chi}$.
- The *decision problem* is to distinguish a certain number of samples drawn from the LWE distribution from uniformly random samples.

### 2.4 Dilithium Description

The Dilithium signature scheme is based on the Fiat-Shamir with aborts structure [Lyu09]. It can also be seen as a variant of the Bai-Galbraith scheme (BG) [BG14]. Dilithium is one of the most promising post-quantum signatures submitted to the NIST competition

[Cen20]. At the moment of writing this paper, Dilithium has reached the third round of the NIST competition.

Dilithium depends on parameters $q$, $n$, $k$, $l$, $\eta$, $d$, $\gamma_1$, $\gamma_2$, $\beta$, $w$, and $\tau$. Details about constraints and recommended values for these parameters are provided in the specifications [BDK+20]. In this paper, we point the reader to the following set of recommended parameters: ($q = 8380417 = 2^{23} - 2^{13} + 1$, $n = 256$, $k = 4$, $l = 4$, $\eta = 2$, $d = 13$, $\gamma_1 = 2^{17}$, $\gamma_2 = q - 1/88$, $\beta = 78$, $w = 80$, $\tau = 39$). In addition, the scheme also uses:

– $H$: a collision-resistant hash function

– $ExpandMask$ : A function used to deterministically generate the randomness of the signature scheme, maps a seed $\rho'$ and a nonce $\kappa$ to $S^l_{\gamma_1}$

– $ExpandA$: A function that maps a uniform seed $\{0,1\}^{256}$ to a matrix $\mathbf{A} \in R_q^{k\times l}$

– $CRH$: A collision resistant hash function

For complete details of these functions, please refer to [BDK+20]. The scheme is known to perform well in terms of its key size (i.e., Dilithium-II has a public key of 1,312 bytes and a signature of 2,420 bytes) and the signing process speed (i.e., its signing process takes 251,144 cycles and its signature verification takes 72,633 cycles on a Skylake CPU, AVX implementation) [BDK+20].

Dilithium performs well because of its different techniques. First, Dilithium is instantiated with Module-LWE. Module-LWE deals with matrix of “small” polynomials instead of a unique one as in Ring-LWE. Module-LWE addresses the limitation of R-LWE: the size of polynomials increases with security. For Module, only the number of rows and columns impacts security, not the size of polynomials—which can be set the same for all instantiations (256 coefficients for Dilithium).

Another optimization employed by Dilithium is the key compression mechanism to reduce public key size. The compression is done in two different ways. First, the sampling of $\mathbf{A}$ is produced with an XOF function (Extendable Output Function), which generates a (deterministic) pseudo-random string from a small seed. Therefore, the public key contains the seed instead of the polynomial $\mathbf{A}$. Another compression is a per-coefficients truncation (or rounding) associated with a correcting code mechanism to guess truncated bits.

In the following section, we give a description of the key generation, signing, and verification processes of the Dilithium scheme.

**Key Generation.** The key generation algorithm is presented in Alg. 1. First, it generates a uniform seed $\rho$. Then, the function $ExpandA$ maps a uniform seed to a matrix $\mathbf{A}$. Given that $\mathbf{s}_1$ and $\mathbf{s}_2$ are two secret random vectors, each coefficient of these vectors is an element of the polynomial ring $R_q$ and is of small size at most $\eta$ (See Tab.1 in the Dilithium specification [BDK+20]). Next, the public key $pk$ is computed as $\mathbf{t} = \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2$ (Alg. 1, Line 5). Note here that only the first $d$ bits of $\mathbf{t}$ are public. This rounding technique yields to a public key size reduction.

**Signing.** The authors of Dilithium specified a non-deterministic signing algorithm. This was added to avoid recent side-channel attacks that exploit determinism [BDK+20]. The signing process is described in Alg. 2. It starts with generating a vector of polynomials $\mathbf{y}$ with coefficients less than a defined constant $\gamma_1$ [BDK+20] (Alg. 2, Line 6). The signer retrieves the highest-order bits of $\mathbf{Ay}$ and computes $\mathbf{w}$. Precisely, each coefficient $w_i$ of $\mathbf{Ay}$ is written in the form $w_i = w_{1,i}.2\gamma_2 + \mathbf{w}_{0,i}$, where $|w_{0,i}| \leq \gamma_2$; $\mathbf{w_0}$, $\mathbf{w}_1$ are the vectors of coefficients $w_{0,i}$ and $w_{1,i}$ respectively. Then, a challenge $c$ is generated as the hash of the message and $\mathbf{w_1}$ (Alg. 2, Line 9). The potential signature is then calculated as $\mathbf{z} = \mathbf{y} + c\mathbf{s}_1$, where $c$ is sampled with the function $B_\tau$. This function generates an element having $\tau$

**Algorithm 1** *Key generation*

```
1: $\zeta \leftarrow \{0,1\}^{256}$
2: $(\rho, \varsigma, K) \in \{0,1\}^{256\times 3} := \mathrm{H}(\varsigma)$
3: $(\mathbf{s}_1, \mathbf{s}_2) \in S_\eta^l \times S_\eta^k := \mathrm{H}(\varsigma)$
4: $\mathbf{A} \in R_q^{k\times l} := ExpandA(\rho)$
5: $\mathbf{t} = \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2$
6: $(\mathbf{t}_1, \mathbf{t}_0) := Power2Round_q(\mathbf{t}, d)$
7: $tr \in \{0,1\}^{384} := CRH(\rho \parallel \mathbf{t}_1)$
8: return $(pk = (\rho, \mathbf{t}_1), sk = (\rho, K, tr, \mathbf{s}_1, \mathbf{s}_2, \mathbf{t}_0))$
```

**Algorithm 2** Signature generation

```
1: $\mathbf{A} \in R_q^{k\times l} := ExpandA(\rho)$
2: $\mu \in \{0,1\}^{384} := CRH(tr \parallel M)$
3: $\kappa := 0, (\mathbf{z}, \mathbf{h}) := \perp$
4: $\rho' \in \{0,1\}^{384} := CRH(K \parallel \mu)$ (or $\rho' \leftarrow \{0,1\}^{384}$ for randomized signing)
5: while $(\mathbf{z}, \mathbf{h}) := \perp$ do
6:     $\mathbf{y} \in \widetilde{S}_{\gamma_1}^l := ExpandMask(\rho', \kappa)$
7:     $\mathbf{w} := \mathbf{A}\mathbf{y}$
8:     $\mathbf{w_1} := HighBits_q(\mathbf{w}, 2\gamma_2)$
9:     $c \in B_\tau := \mathrm{H}(\mu \parallel \mathbf{w_1})$
10:    $\mathbf{z} := \mathbf{y} + c\mathbf{s_1}$
11:    $r_0 := LowBits_q(\mathbf{w} - c\mathbf{s_2}\ , 2\gamma_2)$
12:    if $\|\mathbf{z}\|_\infty \geq \gamma_1 - \beta$ and $\|\mathbf{r_0}\|_\infty \geq \gamma_2 - \beta$ then
13:        $(\mathbf{z}, \mathbf{h} := \perp)$
14:    else
15:        $\mathbf{h} := MakeHint_q(-c\mathbf{t_0}\ , \mathbf{w} - c\mathbf{s_2} + c\mathbf{t_0}\ , 2\gamma_2)$
16:        if $\|c\mathbf{t_0}\| \geq \gamma_2$ or the # of 1's in $\mathbf{h}$ is greater than $\mathbf{w}$ then
17:            $(\mathbf{z}, \mathbf{h}) := \perp$
18:        end if
19:    end if
20:    $\kappa := \kappa + l$
21: end while
22: return $\sigma = (\mathbf{z}, \mathbf{h}, c)$
```

coefficients with values either -1 or 1, and the rest, 0. A rejection condition is applied to the signature $\mathbf{z}$ in order to avoid dependency on the security key. The parameter $\beta$ is set to be the maximum possible coefficient of $c\mathbf{s}_i$. Since $c$ has a defined number of non-zero elements and the maximum coefficients in $\mathbf{s}_i$ is $\eta$, the absolute value of each coefficient in $c\mathbf{s}_i$ is less than or equal to $\beta = \tau \cdot \eta$. If any of the $\mathbf{z}$ coefficients is larger than $\gamma_1 - \beta$, then a rejection occurs and the signing process restarts (Alg. 2, Line 6). In the same manner, the restart also occurs if the low-order coefficients of $\mathbf{A}\mathbf{z} - c\mathbf{t}$ is greater than $\gamma_2 - \beta$. The rejection probability as explained in [BDK+20] is low (between 4 and 7 per signature). The $MakeHint_q$ procedure (Alg. 2, Line 15) produces hints to help guessing the shrunk bits of the public key.

In the deterministic version of Dilithium, a seed is added to the secret key and is used together with the message to produce the randomness $\mathbf{y}$ (Alg. 2, Line 6).

**Verification.** The verification algorithm is described in Alg. 3. The verifier computes the high-order bits of $\mathbf{A}\mathbf{z} - c\mathbf{t}$, and accepts if all the coefficients of $\mathbf{z}$ are less than $\gamma_1 - \beta$

**Algorithm 3** Signature verification

```
1: $\mathbf{A} \in \mathbf{R}_q^{k \times l} := ExpandA(\rho)$
2: $\mu \in \{0,1\}^{384} := CRH(CRH(\rho \| \mathbf{t}_1) \| \; M)$
3: $\mathbf{w}_1' := UseHint_q(\mathbf{h},\ \mathbf{Az} - c\mathbf{t}_1.2^d,\ 2\gamma_2)$
4: if $\|\mathbf{z}\|_\infty < \gamma_1 - \beta$ and $c = H(\mu \; \|\mathbf{w}_1')$ and # of 1's in $\mathbf{h}$ is$\leq \omega$ then
5:     return 1
6: else
7:     return 0
8: end if
```

provided that $c$ is the hash of the message and $\mathbf{w_1'}$. A valid signature should satisfy:

$$HighBits_q(\mathbf{Ay}, 2\gamma_2) = HighBits_q(\mathbf{Ay} - c\mathbf{s_2}, 2\gamma_2).$$

We know that $\|LowBits_q(\mathbf{Ay} - c\mathbf{s}_2)\|_\infty \leq \gamma_2 - \beta$ and $c\mathbf{s}_2$ coefficients are smaller than $\beta$. Then, adding $c\mathbf{s}_2$ will not cause carries by increasing the low-order coefficient to have magnitude of at least $\gamma_2$.

## 2.5 Universal Forgery Assuming Partial Secret-Key Knowledge

In what follows, we present the universal forgery attack on Dilithium assuming the knowledge of partial secret key $\mathbf{s_1}$. In doing so, we acknowledge the work of Ravi et al. in [RJH+18].

We assume an attacker has the knowledge of $\mathbf{s_1}$. Our goal is to generate a valid signature of a message. In the deterministic version of Dilithium, $K$ is public and is used to deterministically generate randomness $\mathbf{y}$ (Alg. 2, Line 6). The attacker proceeds from line 6 to 10 (Alg. 2) by choosing $\mathbf{y}$, uniformly at random, from $S^l_{\gamma_1 - 1}$ in line 6 (Alg. 2) and computing the signature $\mathbf{z}$ using the partial knowledge of $\mathbf{s_1}$ (Alg. 2, Line 10). In the signature verification (Alg. 3), the attacker requires the knowledge of $\mathbf{w_1}$. It is proven that $P[\mathbf{w_1} = HighBits_q(\mathbf{w} - \mathbf{cs_2})]$ is very close to 1 [BDK+20].

Note that $\mathbf{w} - \mathbf{cs_2} = (-\mathbf{ct_0}) + (\mathbf{w} - \mathbf{cs_2} + \mathbf{ct_0})$. We write $\mathbf{u} = -\mathbf{ct_0}$ and $\mathbf{r} = \mathbf{w} - \mathbf{cs_2} + \mathbf{ct_0}$. As we know that $P[\|\mathbf{u}\|_\infty \leq \gamma_2] \approx 1$, which means $\| -\mathbf{ct_0}\|_\infty \leq \|\mathbf{t_0}\|_\infty < 2^d < \gamma_2$. Hence, the attacker can compute $\mathbf{w_1}$ as:

$$UseHint_q(MakeHint_q(\mathbf{u}, \mathbf{r}, 2\gamma_2)) = HighBits_q(\mathbf{u}{+}\mathbf{r}, 2\gamma_2) = HighBits_q(\mathbf{w}{-}\mathbf{cs_2}, 2\gamma_2) = \mathbf{w_1}$$

The attacker needs to compute the hint matrix $\mathbf{h} = MakeHint_q(\mathbf{u}, \mathbf{r}, 2\gamma_2)$ without the knowledge of $\mathbf{t_0}$ ($\mathbf{u} = -\mathbf{ct_0}$). This was done by Ravi et al. in [RJH+18]. The authors of [RJH+18] showed that the function $UseHint_q(\mathbf{u}, \mathbf{r}, \alpha)$ can be inverted to produce the correct hint only if $\|\mathbf{u}\|_\infty \leq \alpha/2$. Therefore, in order to compute the hint $\mathbf{h}$, the attacker has access to $HighBits_q(\mathbf{u} + \mathbf{r}, 2\gamma_2) = HighBits_q(\mathbf{w} - c\mathbf{s}_2)$, where $\mathbf{r} = \mathbf{w} - c\mathbf{s}_2 + c\mathbf{t}_0$ and can easily be computed as $\mathbf{w} - c\mathbf{s}_2 + c\mathbf{t}_0 = \mathbf{Az} - c\mathbf{t}_1 \cdot 2^d$. Another condition to satisfy is: $\|LowBits_q(\mathbf{w} - c\mathbf{s}_2, 2\gamma_2)\|_\infty \leq \gamma_2 - \beta$ which the attacker cannot, as he is unknown to the value of $\mathbf{s}_2$. However, it has been proven in the Dilithium specification [BDK+20] that $P[\|LowBit_q(\mathbf{w} - c\mathbf{s}_2, 2\gamma_2)\|_\infty \leq \gamma_2 - \beta]$ is very close to 1.

Bruinderink. et. al. [GBP18] also present a method to achieve signature forgery after recovering the secret key $\mathbf{s}_1$. The method leverages the fact that $\mathbf{u} = \mathbf{t}_0 - \mathbf{s}_2$, which can be computed as $u = \mathbf{u} = \mathbf{As}_1 - \mathbf{t}_1 \cdot 2^d = \mathbf{t}_0 - \mathbf{s}_2$, approximately matches $\mathbf{t}_0$, given $\mathbf{s}_2$'s small coefficients. Thus, we can compute a hint $\mathbf{h}$ that will be accepted with probability using $\mathbf{u}$ only and instead of $\mathbf{t}_0$ and $\mathbf{s}_2$.

### 2.6 Machine-Learning Model for Profiling Side-Channel Analysis

The main idea behind profiling techniques is that side-channel measurements follow an unknown distribution that can only be approximated by an assumed statistical distribution for the leakage. Among all, template attack is known as the first and best-known method for profiling attacks, where an attacker assumes that the leakage follows a multi-variate Gaussian distribution [BL12].

The advent of machine learning techniques resulted into more profiling attacks later. These techniques allow the attacker to train a classifier to learn automatically from the profiling set of the statistics of the unknown leakage distribution. Thus, one advantage of machine learning over template attacks is that the profiling model is learned without any assumption about the statistical distribution of the leakage.

Profiling side-channel attacks are performed in two phases: profiling and attack. The profiling phase can be achieved by creating a template [APSQ06, CPM+18], or training a model e.g., artificial neural networks such as Multilayer Perceptron (MLP), Convolutional Neural Networks (CNN), among others [BFD20, KPH+18, MPP16, SKL+20]. When using an artificial neural network, the profiling phase requires network training to learn the target device leakage for all possible values of the sensitive variable. In this paper, we use MLP models as a methodology to achieve the profiling, which consist of: the input layer, at least one hidden layer, and the output layer.

The input layer directly receives the data, whereas the output layer creates the required output. The layers in between are known as hidden layers where the intermediate computation occurs. In the training phase, the hidden layers enhance the ability of MLP classifiers to learn a non-linear function $f : X \to Y$ by training on data sets $X$ and $Y$. The set $X$ represents the traces captured from the profiling device while $Y$ is the label according to the selected leakage model such as the Hamming weight or value of the desired variable. MLP models are composed of multiple layers of perceptrons. The perceptron passes the input into a non-linear activation function and produces an output. Next, in the attack phase, the trained classifier is used to classify the captured traces from the victim's device and predict the sensitive information.

## 3 Overview of the Attack

### 3.1 Main Idea

We present a profiling side-channel attack targeting the leakage in the implementation of the bit-unpacking function. The bit-unpacking function is executed multiple times during the signing process. It converts a byte buffer into the vector $\mathbf{y}$, an intermediate value computed and used during the signing process.

We adopt a profiling attack composed of two phases: the profiling phase and the attack phase. During the profiling phase, we execute the signing process with random input messages on a device and collect the sub-traces corresponding to the power usage during the execution of the bit-unpacking function. We label the sub-traces with the sensitive internal data that will be leaked (i.e., zero and non-zero coefficients) and we train our classifiers. In a real-world scenario, an attacker would need access to a cloned device with an architecture identical to the device under attack and can be controlled by the attacker to facilitate the collection of training data for the classifiers. Likewise, in the attack phase, by observing the power traces of the signature generation on Device B, the trained classifier is used to predict the sensitive internal data generated during the signing algorithm with high accuracy. Together with known challenge vector elements (i.e., $\mathbf{A}$, $\mathbf{z}$, and $c$, as described in Sec. 2), we map the (potentially wrong) predictions obtained into a system of linear equations. We use the Least Squares Method (LSM) to get a solution candidate. Then, we uncover the secret key $\mathbf{s}_1$ by solving an Integer Linear Program (ILP).

We follow the alternative signing procedure proposed in [RJH+18, GBP18] to generate a valid signature for arbitrary messages with only the secret key polynomial $\mathbf{s}_1$.

### 3.2 Attacker Model

The goal of this attack is to reconstruct the first part of the secret key of Dilithium. We conduct a profiling machine-learning-based side-channel attack and assume the following:

- In the profiling phase, the attacker has access to a similar cloned device (Device A) as the victim's device. Thus, the attacker can measure the power trace of the entire signing process.
- There are no additional conditions on the secret key and on the messages used in the profiling and attack phases.
- In the attack phase, we assume that the attacker can trigger several signatures on the device under attack (Device B), where the same secret key is used for all signatures.
- There are no conditions on the determinism of the Dilithium-targeted version. Our attack can be mounted against both the deterministic and non-deterministic versions of Dilithium.

## 4 Power Side-Channels Leakage in Dilithium

We enclosed the code and the data required for the profiling and attack phases along with the paper during the submission. It will be published if the paper is accepted.

### 4.1 Identifying Leaking Points

The leaking point we identified is a function used when generating the vector $\mathbf{y}$ (Alg. 2, Line 6). In the reference implementation of Dilithium [BDK+], the initial randomness seed $\rho'$ is extended through an Extensible Output Function (XOF) such as SHAKE-256 or AES-256. The resulting bit-string is then viewed as a byte string and unpacked into $\ell$ polynomials, where each polynomial is unpacked separately. To unpack a bit-string into a polynomial, the byte-string is transformed into a positive number from $\{0, \ldots, 2\gamma_1\}$. Then, we subtract this number from $\gamma_1$. List. 1 lists the C code of the bit unpacking for one polynomial as found in the Dilithium reference implementation [BDK+]. Note that in the $i^{th}$ iteration of the $N/4$ iterations, four coefficients are unpacked namely the $(i)^{th}$, $(i+1)^{th}$, $(i+2)^{th}$, and $(i+3)^{th}$ coefficients.

```
void polyz_unpack(poly *r, const uint8_t *a) {
  unsigned int i;
  for(i = 0; i < N/4; ++i) {
    r->coeffs[4*i+0]  = a[9*i+0];
    r->coeffs[4*i+0] |= (uint32_t)a[9*i+1] << 8;
    r->coeffs[4*i+0] |= (uint32_t)a[9*i+2] << 16;
    r->coeffs[4*i+0] &= 0x3FFFF;

    r->coeffs[4*i+1]  = a[9*i+2] >> 2;
    r->coeffs[4*i+1] |= (uint32_t)a[9*i+3] << 6;
    r->coeffs[4*i+1] |= (uint32_t)a[9*i+4] << 14;
    r->coeffs[4*i+1] &= 0x3FFFF;

    r->coeffs[4*i+2]  = a[9*i+4] >> 4;
    r->coeffs[4*i+2] |= (uint32_t)a[9*i+5] << 4;
    r->coeffs[4*i+2] |= (uint32_t)a[9*i+6] << 12;
    r->coeffs[4*i+2] &= 0x3FFFF;

    r->coeffs[4*i+3]  = a[9*i+6] >> 6;
    r->coeffs[4*i+3] |= (uint32_t)a[9*i+7] << 2;
    r->coeffs[4*i+3] |= (uint32_t)a[9*i+8] << 10;
    r->coeffs[4*i+3] &= 0x3FFFF;

    r->coeffs[4*i+0] = GAMMA1 - r->coeffs[4*i+0];
    r->coeffs[4*i+1] = GAMMA1 - r->coeffs[4*i+1];
    r->coeffs[4*i+2] = GAMMA1 - r->coeffs[4*i+2];
    r->coeffs[4*i+3] = GAMMA1 - r->coeffs[4*i+3];
  }
}
```

Listing 1: C Implementation of the bit-unpacking function as in the Dilithium reference implementation [BDK+]

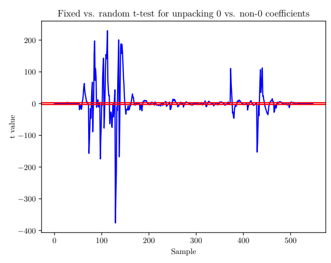
To test whether this function leaks information about the coefficients of the generated polynomial through the power traces of its execution, we leverage Welch's t-test. We restrict our measurement setup to collect a power trace of one iteration of the loop executed in `polyz_unpack` function. This iteration $i$ unpacks the $(i)^{th}$, $(i+1)^{th}$, $(i+2)^{th}$, and $(i+3)^{th}$ coefficients. We now want to test which of the generated coefficients the power trace leaks information on: $(i)^{th}$, $(i+1)^{th}$, $(i+2)^{th}$, or $(i+3)^{th}$. For each of the four coefficients, we follow a *fixed-vs-random* approach to identify if the power trace leaks:

1. We collect multiple power traces where the respective coefficient is unpacked to zero. We denote the set of traces so collected by $L_A$.

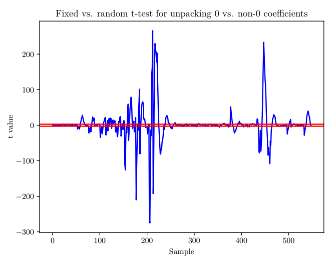
2. We collect multiple power traces where the respective coefficient is unpacked to non-zero and denote the set of traces by $L_B$.

3. For each sample in the collected traces, we then perform Welch's t-test to identify if the distribution from the traces in $L_A$ in that sample is different from $L_B$.

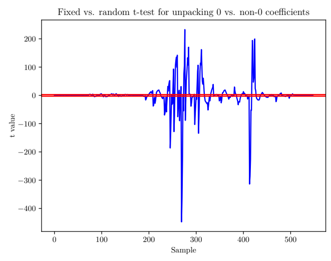
The results of our t-test evaluations are depicted in Fig. 1. For each coefficient, we can see clear peaks in the t-test values that exceed the necessary value for a $t$-critical value for a confidence level of $p = 0.05$. This is a strong indicator that the power traces indeed leak information about whether a certain coefficient is zero or non-zero.



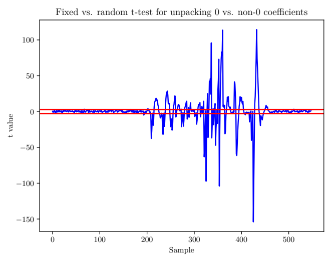
(a) $t$-values for the $(i+0)^{th}$ coefficient



(b) $t$-values for the $(i+1)^{th}$ coefficient



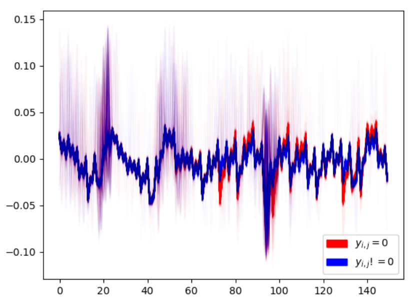
(c) $t$-values for the $(i+2)^{th}$ coefficient



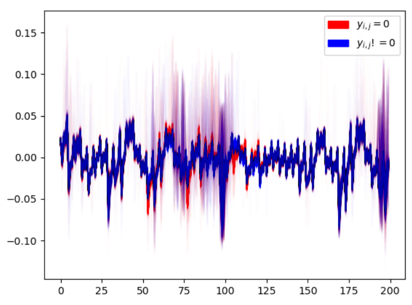
(d) $t$-values for the $(i+3)^{th}$ coefficient

Figure 1: $t$-values for a fixed vs. random $t$-test of the `unpack` function (using 21692 traces). The red line indicates the positive and negative $t$-critical value at a confidence value of $p = 0.05$.

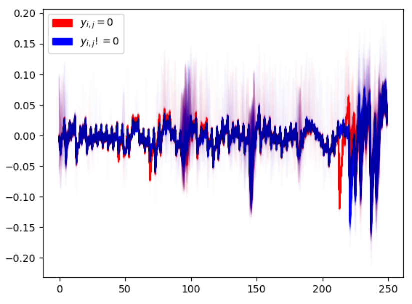
**Profiling Phase.** To prepare the profiling, we executed the signing process with random input messages. The bit-unpacking function is called multiple times during the signing algorithm, when generating the vector $\mathbf{y}$ (Alg. 2, Line 6). It converts a byte buffer into a polynomial of $N$ coefficients. The unpacking routine occurs in $N/4$ iterations. In every iteration $i$, four coefficients are unpacked i.e., the $(i)^{th}$, $(i+1)^{th}$, $(i+2)^{th}$, and $(i+3)^{th}$ coefficients. We noticed that the unpacking of every coefficient necessitates the execution of different instructions. For that reason, we trained four different neural networks, each using 548 samples as input, corresponding to the execution of the unpacking function in List. 1. While we failed to predict the exact value of the unpacked coefficients from the obtained power traces, each one of these traces is susceptible to reveal whether the $(i)^{th}$, $(i+1)^{th}$, $(i+2)^{th}$, and $(i+3)^{th}$ coefficients are zero, which is sufficient to mount our attack.
Fig. 2 illustrate multiple executions of the unpacking routine to unpack the $(i)^{th}$, $(i+1)^{th}$, $(i+2)^{th}$, and $(i+3)^{th}$ coefficients respectively. The blue lines indicate the unpacking of the non-zero coefficients while the red lines, that of the zero coefficients. In order to visualize the difference in the graph when the unpacking result is zero or non-zero, the power consumption traces are overlapped.
Although the difference is clear, an attacker cannot deduce the value of the unpacked coefficient from the power consumption trace with unaided eyes. Therefore, we train different machine-learning classifiers to predict the unpacked coefficient.



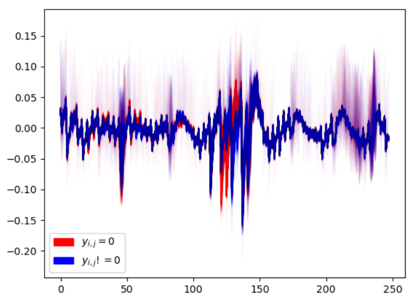
(a)



(b)



(c)



(d)

Figure 2: Overlapped power consumption traces of the bit-unpacking of the polynomial $\mathbf{y}$; the blue lines illustrate when the unpacked coefficient is zero and the red lines illustrate when the unpacked coefficient is non-zero; traces (a), (b), (c), and (d) corresponds to the unpacking of the the $(i)^{th}$, $(i+1)^{th}$, $(i+2)^{th}$, and $(i+3)^{th}$ coefficient, respectively.

With the prepared traces and their corresponding labels, we are able to build for each classifier a list of examples $(x, y) \in \mathbb{R}^t \times Y$, anticipating that $x$ leaks information about $y$. The power traces $x \in \mathbb{R}^t$ act as the *features* and the unpacked coefficients $y \in Y$ as the *labels*. The list of these (noisy) examples $(x, y)$ is split into training, validation, and test set of the Multi-Layer Perceptron (MLP) machine learning model.

As the choice of model and training parameters can have a decisive influence on the prediction accuracy of the resulting model, our hyper-parameters for each classifier were chosen by the hyperparameter optimization technique HYPERBAND [LJD+16]. HYPERBAND is a principled early-stoppping algorithm for hyperparameter optimization, that adaptively allocates a pre-defined resource, e.g., iterations, data samples or number of features, to randomly sampled configurations [LJD+16]. We present in Tab. 1 the MLP architecture used for the four classifiers in order to predict respectively the four unpacked coefficients of the polynomial $\mathbf{y}$ in each iteration $i$ of the unpacking routine. An overview of the obtained accuracy, precision, recall, and specificity of each classifier is shown in Tab. 2.

Table 1: MLP architecture of the classifiers 0, 1, 2, and 3 used for the predictions of the $(i)^{th}$, $(i+1)^{th}$, $(i+2)^{th}$, and $(i+3)^{th}$ coefficients respectively; $i < 256/4$

| | Layer Type | (Input, output) shape | # Parameters |
|---|---|---|---|
| Classifier 0 | Dense | (548, 385) | 211365 |
| | Dropout | (385, 385) | 0 |
| | Dense | (385, 17) | 6562 |
| | Dropout | (17, 17) | 0 |
| | Dense | (17, 97) | 1746 |
| | Dropout | (97, 97) | 0 |
| | Dense | (97, 1) | 98 |
| Classifier 1 | Dense | (548, 385) | 211365 |
| | Dropout | (385, 385) | 0 |
| | Dense | (385, 17) | 6562 |
| | Dropout | (17, 17) | 0 |
| | Dense | (17, 97) | 1746 |
| | Dropout | (97, 97) | 0 |
| | Dense | (97, 1) | 98 |
| Classifier 2 | Dense | (548, 465) | 255285 |
| | Dropout | (465, 465) | 0 |
| | Dense | (465, 257) | 119762 |
| | Dropout | (257, 257) | 0 |
| | Dense | (257, 1) | 258 |
| Classifier 3 | Dense | (548, 385) | 211365 |
| | Dropout | (385, 385) | 0 |
| | Dense | (385, 17) | 6562 |
| | Dropout | (17, 17) | 0 |
| | Dense | (17, 97) | 1746 |
| | Dropout | (97, 97) | 0 |
| | Dense | (97, 1) | 98 |

**Attack Phase.** In the attack phase, we sign a number of uniform, randomly chosen, attacker-known messages. Then, we collect the trace snippets of the bit-unpacking function. We store the obtained traces along with all public information about the signature process (i.e., $\mathbf{A}$, $\mathbf{z}$, and $c$). With the recorded power trace snippets, the attacker is able to use

Table 2: Accuracy, Precision, Recall, and Specificity of the classifiers 0, 1, 2, and 3 used for the predictions of the $(i)^{th}$, $(i+1)^{th}$, $(i+2)^{th}$, and $(i+3)^{th}$ coefficients respectively; $i < 256/4$

| | Classifier 0 | Classifier 1 | Classifier 2 | Classifier 3 |
|---|---|---|---|---|
| Accuracy | 0.999 | 0.999 | 0.999 | 0.999 |
| Precision | 0.999 | 0.997 | 0.999 | 0.998 |
| Recall | 1.0 | 0.999 | 0.999 | 0.996 |
| Specificity | 0.999 | 0.999 | 0.998 | 0.999 |

the four trained classifiers in the profiling phase to obtain a prediction of the unpacked coefficients i.e., the $(i)^{th}$, $(i+1)^{th}$, $(i+2)^{th}$, and $(i+3)^{th}$ coefficients of the polynomial $\mathbf{y}$, with precision as determined on the test set during training stage. In Sec. 5, we explain how to use the leaking information to recover the first part of the secret key $\mathbf{s}_1$.

## 5 Secret Key Retrieval

The source code of the attack is enclosed in the submission of this paper and will be made publicly available.

Throughout this section, we utilize the indices $i$, $j$ to refer to the $j^{th}$ coefficient of the $i^{th}$ polynomial. For NIST 2-security level, $i \leq 256$ and $j \leq 4$. We denote $m$ to refer to the $m^{th}$ signature. $M$ is the total number of needed signatures to perform the attack.

After collecting the power traces of approximately hundred thousand runs of the signing process, we identify the relevant snippets. Then, we use the trained classifiers of Sec. 4 to predict for each coefficient of the polynomials in $\mathbf{y}$ whether it is zero.

The key retrieval proceeds in four steps. First, we define conditional equations in order to minimize the false-positive predictions. In the second step, we map the filtered predictions of the $y_{i,j}$ into a system of linear equations. Then, we use the Least Squares Method (LSM) to get a solution candidate $\hat{\mathbf{s}}_1$ from this noisy equation system. This step is followed by solving an Integer Linear Program (ILP) which computes the correct secret key $\mathbf{s}_1$ by leveraging the solution candidate $\hat{\mathbf{s}}_1$. We provide a formal algorithmic description of the attack procedure in Alg. 4.

### 5.1 Step 1: Predicting which Error Polynomial Coefficients are Zero

Even though the classifiers presented in Sec. 4 provide high accuracy, it is unlikely that the system of linear equations we want to solve in Sec. 5.3 and Sec. 5.4 has an exact solution. As we cannot predict accurately whether $y_{i,j} = 0$, the system is likely not to have a perfect solution. To accommodate for this, we want to define conditions on the $\mathbf{z}$ values to filter the false-positive predictions. Note that for our purposes, we prefer false-negatives (failing to predict that a certain coefficient is zero) to false-positives (predicting a coefficient $y_{i,j}$ to be zero when it is actually not). False-negatives only require us to obtain more signatures and the respective power traces, while false-positives introduce additional noise and could affect the success of our attack.

We observe that $|(cs_1)_{i,j}| \leq \beta$, where $\beta = \tau \cdot \eta$. Assuming the value $y_{i,j} = 0$, $|z_{i,j}| = |y_{i,j} + (cs_1)_{i,j}| \leq \beta$ as well. We can thus dismiss the possibility that a certain coefficient $y_{i,j} = 0$ if the corresponding coefficient

$$|z_{i,j}| \geq \beta \tag{1}$$

To bias our predictions towards false-negatives, we make use of the fact that each coefficient $(cs_1)_{i,j}$ can be approximated by a normal distribution: recall that $c$ is a vector of coefficients

**Algorithm 4** Dilithium secret key retrieval

```
Input A list of $M$ signatures $(\mathbf{z}, \mathbf{c})$ and the respective power traces $T$
Output The secret key $\mathbf{s}_1$
1: $L \leftarrow []$
2: for $m = 1 \ldots M$ do
3:   for $i = 1 \ldots k$ do
4:     for $j = 1 \ldots \ell$ do
5:       if $|z_{i,j}| \leq \frac{(2\cdot\eta+1)^2-1}{12} \cdot \tau$ then
6:         $\hat{y}^m_{i,j} \leftarrow \text{classifier}(T_{y^m_{i,j}})$
7:         if classifier outputs $\hat{y}^m_{i,j} = 0$ then
8:           append $(m, i, j)$ to $L$
9:         end if
10:      end if
11:    end for
12:  end for
13: end for
14: for $j = 1 \ldots \ell$ do
15:   for each prediction made on the $j^{th}$polynomial, collect the respective challenge
      polynomials into matrix $\mathbf{C}$
16:   solve least squares $\mathbf{z} = \mathbf{Cs} + \mathbf{e}$ to obtain candidate solution $\hat{\mathbf{s}}$
17:   obtain correct secret key polynomial $\mathbf{s}$ by solving the integer linear program described
      in Sec. 5.4
18:   $(\mathbf{s}_1)_j = \mathbf{s}$
19: end for
20: return the secret key $\mathbf{s}_1$
```

in the set $\{-1, 0, 1\}$ and thus, each coefficient in the polynomial $cs_1$ can be viewed as the sum of $\tau$ i.i.d. random variables uniformly distributed over the range $[-\eta, \ldots, \eta] \cap \mathbb{Z}$, where $\tau$ is the weight of $c$ (the number of non-zero coefficient in polynomial $c$).

By a central limit theorem argument, this sum is close to a normal distribution with mean 0 and variance $\sigma^2 = \frac{(2\cdot\eta)^2-1}{12\cdot\tau}$. It follows that if a given coefficient $y_{i,j} = 0$, then

$$|z_{i,j}| \leq 2 \cdot \sigma \tag{2}$$

with very high probability. Given the conditions in Eqn. 1 and 2, we only invoke the machine-learning classifier to predict whether $y_{i,j} = 0$ if $|z_{i,j}| \leq 2 \cdot \sigma$. Otherwise, if $|z_{i,j}|$ is larger than that, we assume that $y_{i,j} \neq 0$. These conditions allow a reduced number of false positives for a minor increase in false-negative.

## 5.2 Step 2: Mapping the Predictions Into a Set of Linear Equations

In this section, we build a system of linear equations, which will be used to retrieve the secret key in Sec. 5.3 and Sec. 5.4. Let's assume that we traced $M$ signatures in Sec. 5.1. We denote $y^m$ and $z^m$ as the error polynomial and the resulting signature value for the $m^{th}$ signature, respectively. From Sec. 5.1, we obtain a list $L$ of triples $(m, i, j)$, where each triple represents the assumption that $y^m_{i,j} = 0$. Given $\mathbf{z}^m = \mathbf{y}^m + \mathbf{c}^m\mathbf{s}_1$, we create a system of equations from this list in the following way:

$$z^m_{i,j} = y^m_{i,j} + (cs_1)_{i,j}$$

For each prediction $\hat{y}^m_{i,j} = 0$, we have:

$$z^m_{i,j} = (c^m s_1)_{i,j}$$

Factoring in the erroneous predictions from our classifier, we have a set of equations:

$$z^m_{i,j} = (c^m s_1)_{i,j} + e \tag{3}$$

where $e$ is zero if the classifier correctly predicted the coefficient $y^m_{i,j}$ and $e \neq 0$ otherwise. Thus, we want to obtain the secret key $\mathbf{s}_1$ from a set of equations $\mathbf{z} = \mathbf{C}\mathbf{s_1} + \mathbf{e}$, where $\mathbf{C} \in \mathbb{Z}^{|L| \times n}$ is derived from the challenge polynomials $c$, $\mathbf{z}$ contains the signature coefficients $z^m_{i,j}$, and $\mathbf{e}$ is a vector of error coefficients.

Assuming that we correctly predicted $y_{i,j} = 0$, $e$ is zero for most of the equations. It also holds that $||\mathbf{e}||_\infty \leq 2\sigma + \beta$. (Recall in Eqn. 2, we dismissed observations where $|z_{i,j}| \geq 2\sigma$ and we know that $|(cs_1)_{i,j}| \leq \beta$, thus if $|y_{i,j}| > 2\sigma + \beta$, then $|z_{i,j}| \geq 2\sigma$).

Before we turn our attention to retrieving the secret key $\mathbf{s}_1$ from this set of equations, let us first observe that we can split up the given problem into $\ell$ separate sets of equations, one for each polynomial in the vector $\mathbf{s}_1$. Note that

$$c\mathbf{s}_1 = \begin{pmatrix} c \cdot (s_1)_1 \\ c \cdot (s_1)_2 \\ \vdots \\ c \cdot (s_1)_\ell \end{pmatrix}$$

i.e., to obtain $c\mathbf{s}_1$ we multiply $c$ with each polynomial in the vector $\mathbf{s}_1$ independently. $\mathbf{s}_1 \in S^\ell_\eta$ consists of $\ell$ polynomials, each with $N = 256$ coefficients, while $c \in B_\tau$ is exactly one polynomial with $\tau$ non-zeroes coefficients. For a given leak $y^m_{i,j}$ at polynomial $i$, coefficient $j$, we observe that the equation $z^m_{i,j} = y^m_{i,j} + (c^m s_1)$ is only influenced by the $i^{th}$ polynomial $(s_1)_i$. As a result, we can create $\ell$ independent equation systems (one for each polynomial in $\mathbf{s}_1$) and solve for each polynomial of $\mathbf{s}_1$ separately. This will reduce the computation complexity in the following steps.

The rest of this section describes how to obtain each polynomial in $\mathbf{s}_1$ from a given set of equations. For ease of notation, we denote the polynomial we are currently solving for as $\mathbf{s} = (\mathbf{s_1})_i$.

## 5.3 Step 3: Obtaining a Solution Candidate From a Set of Linear Equations

In Sec. 5.3 and 5.4, we intend to obtain a secret key polynomial $\mathbf{s}_1 \in S^\ell_\eta$ from the set of equations (as in Eqn. 3), where $e$ is zero for most of the equations. In Sec. 5.3, we will first obtain a key candidate $\hat{\mathbf{s}}_1 \in \mathbb{R}^n$ close to the correct secret key polynomial $\mathbf{s}_1$.

Since $\|c\mathbf{s} + \mathbf{e}\|_\infty < q$, there are no modular reductions involved in the given equations. We can thus view the problem of obtaining a secret key polynomial $\hat{\mathbf{s}}$ from a system of linear equations $\mathbf{z} = \mathbf{C}\mathbf{s} + \mathbf{e}$ as an LWE without modular reduction problem. Using an approach described by [BDE+18], we obtain a solution candidate $\hat{\mathbf{s}} \in \mathbb{R}^n$ by employing the least-squares method.

The least-squares method computes $\hat{\mathbf{s}} \in \mathbb{R}^n$ as the vector minimizing the squared euclidean norm: $\|\mathbf{C}\hat{\mathbf{s}} - \mathbf{z}\|^2_2$. We calculate $\hat{\mathbf{s}}$ by employing the closed-form solution formula for least squares:

$$\hat{\mathbf{s}} = (\mathbf{C}^T\mathbf{C})^{-1} \cdot \mathbf{C} \cdot \mathbf{z}$$

This solution candidate converges to a correct solution [BDE+18]. Given enough equations, it holds that $\lfloor \hat{\mathbf{s}}_i \rceil = \mathbf{s}_i$ for all $i \in \{1, \dots, n\}$. Even with fewer equations, the solution $\lfloor \hat{\mathbf{s}} \rceil$ is usually close to the correct solution $\mathbf{s}$ i.e., most of the coefficients are correct and some are wrong by $\pm 1$.

More precisely, we leverage our solution candidate $\hat{\mathbf{s}}$ to identify the correct secret key polynomial $\mathbf{s}$ by making use of the following observation. Given enough equations, the following should hold for each coefficient in the least-squares solution $\hat{\mathbf{s}}$:

1. Either rounding up or down should yield the correct solution
$$\lfloor \hat{s}_j \rfloor = (s)_j \text{ or } \lceil \hat{s}_j \rceil = (s)_j$$
2. For coefficients $j$, where $\hat{s}_j$ is close to an integer, the coefficient candidate should be correct. Formally, define the function:
$$\text{dist\_nint}(x) = \min\big(x - \lfloor x \rfloor, 1 - (x - \lfloor x \rfloor)\big)$$
If $\text{dist\_nint}(\hat{s_1}_j) \approxeq 0$ then $\lfloor \hat{s}_j \rceil = (s)_j$

## 5.4 Step 4: Solving an Integer Linear Program Leveraging the Solution Candidate

Note that for most of the equations in our equation system $e = 0$, assuming that our classifier is correct in most instances. As a result, identifying the correct secret key polynomial $\mathbf{s}$ from the set of noisy equations amounts to identifying the secret key polynomial that maximizes the number of fulfilled equations.

We formulate an Integer Linear Program (ILP) to solve this problem, factoring in the solution candidate $\hat{\mathbf{s}}$ obtained from the least-squares method.

$$\begin{aligned}
&\text{maximize} && \sum_{l=1}^{|L|} x_l \\
&\text{subject to} && \\
& && \mathbf{z}_l - \mathbf{C}_l\mathbf{s} \leq K \cdot (1 - x_l) && \forall l \in \{1, \ldots, |L|\} && (1)\\
& && \mathbf{z}_l - \mathbf{C}_l\mathbf{s} \geq -K \cdot (1 - x_l) && \forall l \in \{1, \ldots, |L|\} && (2)\\
& && (s)_i \in \{\lfloor \hat{s}_i \rfloor, \lceil \hat{s}_i \rceil\} && \forall i \in \{1, \ldots, n\} && (3)\\
& && (s)_i = \lfloor (s)_i \rceil && \forall i \in \{1, \ldots, N \mid \text{dist\_nint}(\hat{s}_i) \leq 0.01\} && (4)\\
& && x_l \in \{0, 1\} && \forall l \in \{1, \ldots, |L|\} && (5)
\end{aligned}$$

where constraints (1) and (2) ensure that if $x_m = 1$, $\mathbf{z}_l - \mathbf{C}_l\mathbf{s} = 0$ (This is canonically known as the big-$M$ method, we choose $K$ as the maximum possible distance between $\mathbf{z}_l$ and $\mathbf{C}_l\mathbf{s}$) and constraint (3) and (4) factor in the information from our solution candidate $\hat{\mathbf{s}}$ obtained by least-squares. Note that constraints (3) and (4) are optional, and can be removed to trade performance for a higher success chance of the attack. We run two ILP-solvers in parallel (one with constraints (3) and (4), and one without) until we have solution $\mathbf{s}$ that satisfies at least $(1 - \epsilon) \cdot N$ equations, where $\epsilon$ is our assumed false-positive rate (the percentage of equations that we assume to be wrong). The obtained solution should match the secret key polynomial of the secret key $\mathbf{s}$.

We perform step Sec. 5.3 and 5.4 $l$ times, for each polynomial equation independently. This then yields a secret key candidate $\mathbf{s}_1$. We can then perform universal forgery as described in Sec. 2.5.

## 5.5 Alternative Attack Strategies

We also explored two alternative attack strategies in addition to the ILP approach: the *LWE with side information* technique by Dachman-Soled et al. [DDGR20], and *the ternary LWE attacks* by Kirshanova and May [KM21].

**LWE with Side Information.** The "LWE with side information" technique provides a framework to integrate additional information about an LWE problem in the form of so-called hints. To this end, a given LWE instance is transformed into a Distorted Bounded

Distance Decoding (DBDD) instance, which allows to keep track of a distribution of the secret vector. Hints can, for example, alter the secret distribution in a way that potentially makes the problem easier. After providing enough hints, it might be feasible to recover the LWE secret through lattice-reduction attacks. We explored the option of integrating the information obtained through the leaking implementation as hints until we can recover the secret through a lattice-reduction step. However, the Sage toolkit provided by the authors of [DDGR20] is not yet optimized enough to integrate enough approximate hints in a reasonable time. So, we were unable to verify this method in practice.

**Meeting Ternary LWE Keys.** Kirshanova and May [KM21] propose a MITM attack that can recover a ternary secret key $\mathbf{s} \in \{0, \pm 1\}^n$ from an LWE instance $\mathbf{A}\mathbf{s} = \mathbf{b} + \mathbf{e} \mod q$, where $e \in \{0, \pm 1\}$ in asymptotic complexity $S^{0.25}$, and $S$ is the size of the search space we need to consider for $s$. We explored the option of leveraging this MITM attack to correct the wrong coefficients in candidate secret key polynomial $\hat{\mathbf{s}}$ as obtained from least-squares. However, we could not test our ideas in practice, since there are no implementations of the MITM attack available yet.

The ILP approach proved to be the most practical as far as our parameters are concerned. However, the techniques above can provide tighter asymptotic complexity estimates, and when properly optimized, could make it possible to reduce the number of traces needed to mount the attack. We plan to explore them further in future work.

## 6 Experimental Setup and Results

### 6.1 Experimental Setup

Usually, power traces are collected by observing the power usage of the device during the entire time of signature generation. However, in this work, we propose an analysis approach that only runs sections of the Dilithium signature and collects only the relevant power trace snippets. This approach enables us to use low-cost power analysis hardware and increases the efficiency of the profiling phase of the attack significantly. Due to the constant-time nature of Dilithium, the power trace snippets as recorded by our setup could easily be derived from the full power traces of the signature process. Hence, the usage of power trace snippets does not reveal any additional information to the attacker.

**Data Pre-processing.** To facilitate our low-cost approach, we conduct the data collection for both profiling and attack in a phase we call *data pre-processing.* During this phase, we ran the Dilithium implementation [BDK+20] on an x86 Ubuntu 20.04 server machine. In the second stage, we used the profiling and attack devices to rerun sections of Dilithium code susceptible to leakage (i.e., the bit-unpacking function called multiple times during the signing process) and record power traces for the profiling and attack phases.

For the profiling, we signed a number of uniform and randomly chosen messages using random, individual keys. For the attack phase, the key pair generation was invoked to generate the key under attack and a number of uniform, randomly chosen, and attacker-known messages were signed. For both profiling and attack, we collected the internal inputs and outputs of the function susceptible to leakage of sensitive data (i.e., the unpacking function), and stored them along with all public information about the signature process. With this prepared data, we were able to rerun and analyze the parts of the code susceptible to leakage, individually.

**Workbench.** In our experiment, we considered two identical Cortex M4 CPUs equipped with two STM32F4 microcontrollers, named Device A and B. Device A will be used for profiling, while Device B is the device under attack. Our target is the bit-unpacking function `polyz_unpack()` which is called multiple times during the signing process.

As the ChipWhisperer Lite is limited to 24,400 samples per recorded trace and cannot be used to record the entire power trace of the Dilithium signature process, we used the data prepared in the pre-processing phase as explained above. During recording, the ChipWhisperer and the microcontroller both ran on the same 7,372,800 Hz clock. The sampling rate of the ADC was set to 4 samples/cycle with 10-bit resolution and a 45 dB low-noise gain filter. Collecting and storing all relevant traces was coordinated using a Python script.

**Compilation.** The Dilithium source code [BDK+] was provided as a portable C implementation, which makes it suitable for compilation to different architectures.

The original benchmarking was done using the SUPERCOP benchmarking framework [BL] on an Intel Core-i7 6600U (Skylake) CPU [BDK+20]. We compiled Dilithium using the gcc-arm cross-compiler *arm-none-eabi-gcc 9.2.1* on an Intel(R) Xeon(R) CPU E7-4870 running Ubuntu 20.04 and the default SUPERCOP [BL] compiler options (*-O3 -fomit-frame-pointer -fwrapv*), with the necessary changes to cross-compile it for the two used devices. We used the SCIP Optimization suite [GAB+20] to solve the ILPs.

## 6.2 Experimental Results

We evaluate our attack in two settings: The first is a theoretical evaluation, where we run the Dilithium signature algorithm and simulate erroneous classifiers. A practical evaluation based on the measurement setup described in Sec. 6.1 gives the results for the experiment run on the ChipWhisperer.

**Theoretical Evaluation.** Alg. 5 describes the theoretical evaluation framework. We report that the number of signatures required to retrieve the secret key $\mathbf{s}_1$ is approximately 10 minutes for NIST 2-security level. Fig. 4 shows the number of equations needed for key retrieval as function of the success rate.

It is worth mentioning that by applying the two conditions in Eqn. 1 and 2 we already reduced number of false positives. Then, we assumed that we achieved a false-positive and true-positive rate of 0.981. We noticed that we needed at least 789,965 signatures and 4,524 equations to retrieve the secret key $\mathbf{s}_1$.

We believe that a higher rate of false negatives will result in an increasing number of signatures needed to retrieve the secret key. On the other hand, a higher rate of false positive has a dramatic influence on the success of Alg. 4.

To investigate the impact of the parameter choices, Table 3 evaluates whether the described attack was successful in at least 1 in 20 trials for different parameters. The results show that a lower true positive rate (=lower false negative) can be compensated for by collecting more signatures, while even a slightly higher false positive rate quickly becomes prohibitive. Attacking NIST security level 3 was only feasible with a very low false positive rate, while attacking NIST security Level 5 can be done via increasing the number of collected signatures.

**Practical Evaluation.** We conducted a practical evaluation of our attack against the Dilithium signature scheme variant having a security equivalent to the NIST 2-security level. The evaluation follows the setup described in Sec. 6.1, using the machine classifiers described in Sec. 2. We were able to recover the secret key $\mathbf{s}_1$ by tracing the `polyz_unpack()` function for $756,589$ signatures, out of which we extracted $2,015$ equations. 24 of those equations were wrong, amounting to a noise level of $\approx 1.19\%$. We emphasize that our method can also handle higher noise levels, as can be seen from the theoretical evaluation.

**Algorithm 5** Simulate noise in predictions from an assumed side-channel on $\mathbf{y}$

**Input** The number of signatures $M$ to try, an assumed true and false positive rate of the machine-learning classifier equal to 0.981. An equation threshold $T$.

**Output** Whether $\mathbf{s_1}$ could be recovered in under 10 minutes computation time.

```
1:  Generate $M$ signatures
2:  for $m = 1 \ldots M$ do
3:    for  $i = 1 \ldots k$ do
4:      for $j = 1 \ldots \ell$ do
5:        if $|z_{i,j}^m| \leq \frac{(2\cdot\eta+1)^2-1}{12} \cdot \tau$ then
6:          if $y_{i,j}^m = 0$ then
7:            predict that $y_{i,j}^m = 0$ with probability true-positive rate
8:          else
9:            Predict that $y_{i,j}^m = 0$ with probability false positive rate
10:         end if
11:       end if
12:       if Collected more than $T$ equations then
13:         Use Alg. 4 to try to recover the secret key polynomial $\mathbf{s_1}$ from predictions
14:       end if
15:     end for
16:   end for
17: end for
18: Use Alg. 4 to try to recover the secret key polynomial $\mathbf{s_1}$ from predictions
```

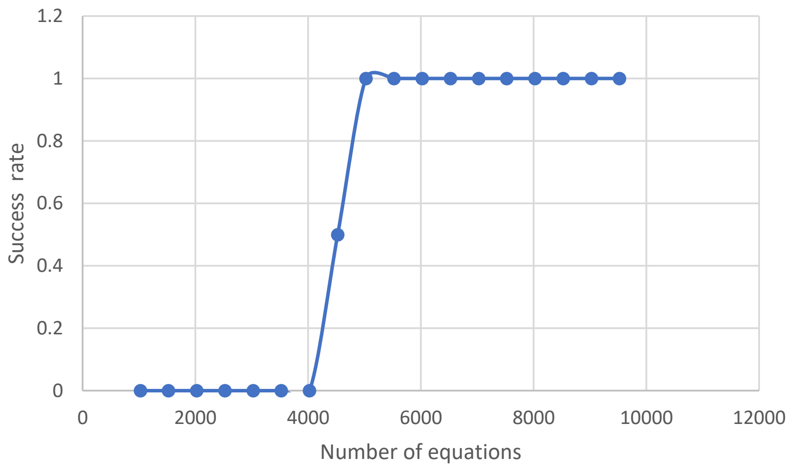


Figure 4: Theoretical evaluation as described in Alg. 5 for NIST 2-security level, a false positive and true-positive rate of 0.981. The number of equations describes the number of zero coefficients predicted. The success rate describes the percentage of 30 repeated experiments that resulted in successful key recovery.

## 7 Conclusion and Possible Countermeasures

In this paper, we presented a profiling power side-channel attack on the Dilithium signature scheme. Using a leak in a bit-unpacking function, we leverage machine-learning to recover noisy information about the vector $\mathbf{y}$. Together with known elements (i.e., the signature $\mathbf{z}$

Table 3: Evaluation on the influence of different parameters on the success of the key recovery Alg. 4, evaluated using the method described in Alg. 5. Each parameter set was tested for twenty trials. If there was at least one success, we report the number of equations in a successful run, otherwise we report the number of equations in any one of the twenty runs.

| Security Level | True Negative Rate | True Positive Rate | Number Of Signatures | Number of Total Equations | Number Of Wrong Equations | At Least One Success |
|---|---|---|---|---|---|---|
| 2 | 0.99 | 0.99 | 650000 | 3204 | 796 | Yes |
| 2 | 0.99 | 0.9 | 750000 | 3315 | 860 | Yes |
| 2 | 0.99 | 0.9 | 650000 | 2893 | 767 | No |
| 2 | 0.99 | 0.8 | 650000 | 2719 | 812 | No |
| 2 | 0.99 | 0.8 | 2000000 | 8451 | 2506 | Yes |
| 2 | 0.985 | 1 | 800000 | 4480 | 1484 | Yes |
| 2 | 0.981 | 0.981 | 800000 | 4393 | 1729 | Yes |
| 2 | 0.97 | 1 | 4000000 | 30811 | 12465 | No |
| 3 | 0.985 | 1 | 800000 | 1570 | 755 | No |
| 3 | 0.985 | 1 | 2000000 | 1570 | 755 | No |
| 3 | 0.985 | 1 | 4000000 | 7750 | 3734 | No |
| 3 | 0.995 | 1 | 3500000 | 4792 | 1120 | Yes |
| 5 | 0.985 | 1 | 800000 | 2233 | 889 | No |
| 5 | 0.985 | 1 | 3500000 | 9321 | 3631 | Yes |

and the message digest $c$), this small leak suffices to achieve the equivalent of key recovery with a moderate number of signatures.

Defending against our attack requires dedicated countermeasures against power side channel attacks. Masking Dilithium, as described by Migliore et. al. [MGTF19], constitutes such a countermeasure. The proposed masking scheme for Dilithium is based on Boolean and arithmetic masking. Each sensitive variable is split into $t + 1$ shares, where $t$ is the so-called masking order. Every operation that acts on the sensitive information is reformulated to act on each of the shares independently instead. Breaking a fully masked Dilithium implementation with the attack described in this paper would require to deduce the value of all shares of $\mathbf{y}$. Power consumption is inherently noisy, implying the potential for erroneous classifications on each share. As a result, the probability to correctly deduce the value of a coefficient $\mathbf{y}_{i,j}$ decreases exponentially with the number of shares.

However, the masking countermeasure described in [MGTF19] induces a performance overhead. Migliore et. al. [MGTF19] measure that first-order masking already slows down signature creation by a factor of five. Arguably, the estimates given by [MGTF19] are a lower bound, as the authors replaced the real modulus by a modulus that is a power of 2, which boosts performance of the countermeasure. The performance loss could impede the adoption of such countermeasure. On the other hand, considering that our attack only requires noisy information about the hamming weight of the coefficients of $\mathbf{y}$, more efficient countermeasure are non-trivial to design. For example, restricting the masking to just $\mathbf{y}$ is not sufficient. As soon as $\mathbf{y}$ would be unmasked, an attacker could again retrieve all information necessary for the attack.

We highlight that it was thus far unclear whether a dedicated countermeasure against power side-channels is needed for Dilithium at all: No end-to-end power side-channel attack against Dilithium had been demonstrated so far. Naturally, this could be taken as an indicator that Dilithium's countermeasures against side-channels attacks are sufficient for preventing power side-channel attacks also, especially when implementing the randomized version of Dilthium. This would allow implementations to skip masking in order to not suffer from its performance impact. Our results show that this is not the case. Consequently, our work calls attention to an urgent need of further defensive as well as offensive research: It is paramount for the implementation security of Dilithium to continue to explore possible countermeasures against power side-channel attacks in order to identify countermeasure that have only a minor impact on performance. In addition to that, offensive research is needed to uncover potential bypasses against existing countermeasures.This will ensure that protective measures taken to prevent power side-channel attacks are indeed sufficient.

## References

[AOTZ20] Diego F. Aranha, Claudio Orlandi, Akira Takahashi, and Greg Zaverucha. Security of hedged Fiat-Shamir signatures under fault attacks. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 644–674. Springer, Heidelberg, May 2020.

[APSQ06] C. Archambeau, E. Peeters, F. X. Standaert, and J. J. Quisquater. Template attacks in principal subspaces. In Louis Goubin and Mitsuru Matsui, editors, *Cryptographic Hardware and Embedded Systems - CHES 2006*, pages 1–14, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[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.

[BDK$^+$] Shi Bai, Leo Ducas, Eike Kiltz, Tancrede Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehle. Dilithium reference implementation. `https://github.com/pq-crystals/dilithium`. [Online; accessed 20-December-2020].

[BDK$^+$20] Shi Bai, Leo Ducas, Eike Kiltz, Tancrede Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehle. Dilithium official website. `https://pq-crystals.org/dilithium/index.shtml`, 2020. [Online; accessed 20-December-2020].

[BFD20] Martin Brisfors, Sebastian Forsmark, and Elena Dubrova. How deep learning helps compromising USIM. In Pierre-Yvan Liardet and Nele Mentens, editors, *Smart Card Research and Advanced Applications - 19th International Conference, CARDIS 2020, Virtual Event, November 18-19, 2020, Revised Selected Papers*, volume 12609 of *Lecture Notes in Computer Science*, pages 135–150. Springer, 2020.

[BG14] Shi Bai and Steven D. Galbraith. An improved compression technique for signatures based on learning with errors. In Josh Benaloh, editor, *CT-RSA 2014*, volume 8366 of *LNCS*, pages 28–47. Springer, Heidelberg, February 2014.

[BL] Daniel J. Bernstein and Tanja Lange. eBACS: ECRYPT Benchmarking of Cryptographic Systems. `https://bench.cr.yp.to/`. [Online; accessed 20-December-2020].

[BL12] Timo Bartkewitz and Kerstin Lemke-Rust. Efficient template attacks based on probabilistic multi-class support vector machines. In Stefan Mangard, editor, *Smart Card Research and Advanced Applications - 11th International Conference, CARDIS 2012, Graz, Austria, November 28-30, 2012, Revised Selected Papers*, volume 7771 of *Lecture Notes in Computer Science*, pages 263–276. Springer, 2012.

[Cen20] NIST Information Technology Laboratory Computer Security Resource Center. NIST Standardization round. https://csrc.nist.gov/projects/post-quantum-cryptography, 2020. [Online; accessed 15-Ju ne-2021].

[CGM19] Yilei Chen, Nicholas Genise, and Pratyay Mukherjee. Approximate trapdoors for lattices and smaller hash-and-sign signatures. In Steven D. Galbraith and Shiho Moriai, editors, *ASIACRYPT 2019, Part III*, volume 11923 of *LNCS*, pages 3–32. Springer, Heidelberg, December 2019.

[CPM$^+$18] Giovanni Camurati, Sebastian Poeplau, Marius Muench, Tom Hayes, and Aurélien Francillon. Screaming channels: When electromagnetic side channels meet radio transceivers. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 163–177. ACM Press, October 2018.

[DDGR20] Dana Dachman-Soled, Léo Ducas, Huijing Gong, and Mélissa Rossi. LWE with side information: Attacks and concrete security estimation. Cryptology ePrint Archive, Report 2020/292, 2020. https://eprint.iacr.org/2020/292.

[DDLL13] Léo Ducas, Alain Durmus, Tancrède Lepoint, and Vadim Lyubashevsky. Lattice signatures and bimodal gaussians. In *Annual Cryptology Conference*, pages 40–56. Springer, 2013.

[DLP14] Léo Ducas, Vadim Lyubashevsky, and Thomas Prest. Efficient identity-based encryption over NTRU lattices. In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT 2014, Part II*, volume 8874 of *LNCS*, pages 22–41. Springer, Heidelberg, December 2014.

[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*, 2021.

[FDK20] Apostolos P. Fournaris, Charis Dimopoulos, and Odysseas Koufopavlou. Profiling dilithium digital signature traces for correlation differential side channel attacks. In Alex Orailoglu, Matthias Jung, and Marc Reichenbach, editors, *Embedded Computer Systems: Architectures, Modeling, and Simulation*, pages 281–294, Cham, 2020. Springer International Publishing.

[GAB$^+$20] Gerald Gamrath, Daniel Anderson, Ksenia Bestuzheva, Wei-Kun Chen, Leon Eifler, Maxime Gasse, Patrick Gemander, Ambros Gleixner, Leona Gottwald, Katrin Halbig, Gregor Hendel, Christopher Hojny, Thorsten Koch, Pierre Le Bodic, Stephen J. Maher, Frederic Matter, Matthias Miltenberger, Erik Mühmer, Benjamin Müller, Marc E. Pfetsch, Franziska Schlösser, Felipe Serrano, Yuji Shinano, Christine Tawfik, Stefan Vigerske, Fabian Wegscheider, Dieter Weninger, and Jakob Witzig. The SCIP Optimization Suite 7.0. Technical report, Optimization Online, March 2020.

[GBP18] Leon Groot Bruinderink and Peter Pessl. Differential fault attacks on deterministic lattice signatures. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2018(3):21–43, Aug. 2018.

[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.

[GLP12] Tim Güneysu, Vadim Lyubashevsky, and Thomas Pöppelmann. Practical lattice-based cryptography: A signature scheme for embedded systems. In Emmanuel Prouff and Patrick Schaumont, editors, *CHES 2012*, volume 7428 of *LNCS*, pages 530–547. Springer, Heidelberg, September 2012.

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

[HHGP$^+$03] Jeffrey Hoffstein, Nick Howgrave-Graham, Jill Pipher, Joseph H Silverman, and William Whyte. Ntrusign: Digital signatures using the ntru lattice. In *Cryptographers' track at the RSA conference*, pages 122–140. Springer, 2003.

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 388–397. Springer, Heidelberg, August 1999.

[KM21] Elena Kirshanova and Alexander May. How to find ternary lwe keys using locality sensitive hashing. Cryptology ePrint Archive, Report 2021/1255, 2021. https://ia.cr/2021/1255.

[KPH$^+$18] Jaehun Kim, Stjepan Picek, Annelie Heuser, Shivam Bhasin, and Alan Hanjalic. Make some noise: Unleashing the power of convolutional neural networks for profiled side-channel analysis. Cryptology ePrint Archive, Report 2018/1023, 2018. https://eprint.iacr.org/2018/1023.

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

[LJD$^+$16] Lisha Li, Kevin G. Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Efficient hyperparameter optimization and infinitely many armed bandits. *CoRR*, abs/1603.06560, 2016.

[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, Heidelberg, December 2009.

[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.

[MP12] Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 700–718. Springer, Heidelberg, April 2012.

[MPP16] Houssem Maghrebi, Thibault Portigliatti, and Emmanuel Prouff. Breaking cryptographic implementations using deep learning techniques. Cryptology ePrint Archive, Report 2016/921, 2016. https://eprint.iacr.org/2016/921.

[Ngu99] Phong Q. Nguyen. Cryptanalysis of the Goldreich-Goldwasser-Halevi cryptosystem from Crypto'97. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 288–304. Springer, Heidelberg, August 1999.

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

[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.

[Sho97] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput.*, 26(5):1484–1509, October 1997.

[SKL+20] Bo-Yeon Sim, Jihoon Kwon, Joohee Lee, Il-Ju Kim, Tae-Ho Lee, Jaeseung Han, Hyojin Yoon, Jihoon Cho, and Dong-Guk Han. Single-trace attacks on message encoding in lattice-based kems. *IEEE Access*, 8:183175–183191, 2020.