From MLWE to RLWE: A Differential Fault Attack on Randomized & Deterministic Dilithium

Mohamed ElGhamrawy[1,2], Melissa Azouaoui[1], Olivier Bronchain[3], Joost Renes[4], Tobias Schneider[5], Markus Schönauer[5], Okan Seker[1] and Christine van Vredendaal[4]

[1] NXP Semiconductors, Hamburg, Germany
[2] Hamburg University of Applied Sciences, Hamburg, Germany
[3] NXP Semiconductors, Leuven, Belgium
[4] NXP Semiconductors, Eindhoven, the Netherlands
[5] NXP Semiconductors, Gratkorn, Austria

Abstract. The post-quantum digital signature scheme CRYSTALS-Dilithium has been recently selected by the NIST for standardization. Implementing CRYSTALS-Dilithium, and other post-quantum cryptography schemes, on embedded devices raises a new set of challenges, including ones related to performance in terms of speed and memory requirements, but also related to side-channel and fault injection attacks security. In this work, we investigated the latter and describe a differential fault attack on the randomized and deterministic versions of CRYSTALS-Dilithium. Notably, the attack requires a few instructions skips and is able to reduce the MLWE problem that Dilithium is based on to a smaller RLWE problem which can be practically solved with lattice reduction techniques. Accordingly, we demonstrated key recoveries using hints extracted on the secret keys from the same faulted signatures using the LWE with side-information framework introduced by Dachman-Soled et al. at CRYPTO'20. As a final contribution, we proposed algorithmic countermeasures against this attack and in particular showed that the second one can be parameterized to only induce a negligible overhead over the signature generation.

Keywords: Post-Quantum Cryptography · Differential Fault Attacks · Dilithium · Lattice Reduction

1 Introduction

Current digital security infrastructures heavily rely on secure and efficient cryptographic primitives, including digital signatures which are based on asymmetric/public-key cryptography. However, schemes based on classic public-key cryptography, like RSA and ECC, are at risk of being broken once a relevant quantum computer is realized. This threat has accelerated the research into Post-Quantum Cryptography (PQC) schemes: cryptographic algorithms which are still secure even against an adversary with access to a quantum computer. After a few years since the call for proposals for new public-key cryptography standards by the National Institute of Standards and Technology (NIST) [Nat], on July 5th, 2022, the NIST has selected two primary algorithms to standardize: CRYSTALS-Kyber for key establishment and CRYSTALS-Dilithium for digital signatures. In addition, the signature schemes FALCON and SPHINCS+ will also be standardized. Notably, CRYSTALS-Dilithium [DKL+21] (which we refer to as Dilithium for conciseness in the rest of this paper) is recommended for embedded use cases due to its relative efficiency compared to other PQC schemes.

In the embedded context implementing PQC schemes raises a few challenges, including ones related to large keys, signatures or ciphertexts, memory usage and notably resistance to implementation attacks, which include both side-channel attacks and Fault Attacks (FA). Dilithium has received relatively significant attention when it comes to FA compared to other PQC digital signature schemes. Precisely, safe error [BMR21], zeroing/skipping [PV06, EFGT16] and nonce reuse attacks [RRB+19] have been applied to Dilithium's embedded implementations. Recently, a new kind of attack, so-called *signature correction*, has been introduced by Islam et al. [IMS+22] and applied to Dilithium. However, so far and up to our knowledge, Differential Fault Attacks (DFA) have been limited to the deterministic version of Dilithium [BP18]. We refer to the work of Ravi et al. [RCDB22] for a detailed survey of both side-channel and fault attacks on lattice-based PQC schemes.

Similarly to classic public-key cryptography schemes, implementation attacks on lattice-based PQC schemes can be enhanced with lattice reduction attacks. At CRYPTO 2020, Dachman-Soled et al. [DDGR20] proposed a framework for the cryptanalysis of lattice-based schemes with side-information. This side-information is also referred to as hints and corresponds to information that an adversary can extract about secret keys after a side-channel or fault attack. Dachman-Soled et al. additionally provided a Sage toolkit for both estimating the difficulty of and performing lattice cryptanalysis with hints.

Contributions. In this work, we extend the previous state of the art and fill the gap by introducing a new DFA targeting MLWE schemes, which applies to both the randomized and deterministic versions of Dilithium. This attack exploits the MLWE structure of Dilithium and the way the masking or randomness vector $\mathbf{y}$ is sampled. Our main contributions are the following:

- First, we describe the attack by highlighting in Subsection 3.1 the location of the instruction to skip. We then explain its impact on the security of Dilithium in Subsection 3.2. Considering the abort property of Dilithium, we describe attack strategies for the randomized and deterministic versions. We estimated that on average 11.3, 18.8 and 22.2 fault injections are needed for levels II, III and V, respectively, in the randomized case. In the deterministic case, less fault injections are required: on average 7.3, 13 and 15.8 for levels II, III and V, respectively. With the collected faulty signatures, we are able to reduce the security of Dilithium from its standard MLWE problem to a much simpler RLWE one. Accordingly, we show in Section 4 that this RLWE problem can be practically solved with lattice reduction.

- We then demonstrate in Subsection 5.1 how the same faulty signatures can be used to extract hints (as defined in [DDGR20]) about Dilithium's secret key. We then use this same framework to integrate these hints into a lattice reduction attack to significantly improve its efficiency. Our results are shown in Subsection 5.2 and Subsection 5.3. One interesting conclusion is that Dilithium level V seems to be the most vulnerable to the attack. Eventually, key recoveries using lattice reduction take on average a few hours to recover the full secret key on our particular setup.

- Finally, in Section 6 we propose algorithmic countermeasures to prevent the attack presented in this work. We also analyze their performance impact and notably show that the main countermeasure we suggest only incurs a negligible overhead compared to the already costly signature generation. This comes at the trade-off of marginally reducing the fault detection probability for variations of our attack.

2 Background

2.1 Polynomial ring notations

All ring arithmetic operations in the paper are over the polynomial ring $R = \mathbb{Z}_q[X]/(X^n+1)$. We denote a polynomial with regular lower-case letters, e.g., $p \in R$, a vector of polynomials with bold lower-case letters, e.g., $\mathbf{a} \in R^k$ and a matrix of polynomials with bold upper-case letters, e.g., $\mathbf{A} \in R^{k \times k}$. In addition, we denote the i-th coefficient of p as $p_{[i]}$, the i-th coefficient of the j-th polynomial in $\mathbf{a}$ as $\mathbf{a}_{[j,i]}$, and the i-th coefficient of the (k,j)-th polynomial in $\mathbf{A}$ as $\mathbf{A}_{[k,j,i]}$. In cases where the lowest index is omitted, we are referring to the complete j-th and (k,j)-th polynomial of $\mathbf{a}$ and $\mathbf{A}$, respectively.

For z, $\alpha \in \mathbb{Z}$ we write $z \bmod^{\pm} \alpha$ to mean the unique integer z' in $]-\frac{\alpha}{2}, \frac{\alpha}{2}]$ with $z \equiv z' \bmod \alpha$ if α is even (resp., odd). The notation $\mathbf{z} \bmod^{\pm} \alpha$ implies that all the coefficients in $\mathbf{z}$ are given $\bmod^{\pm} \alpha$. With this, the following norms on $\mathbb{Z}_q$, R and R^k are defined:

$$\|z\|_\infty = |z \bmod^{\pm} q| \qquad \|p\|_\infty = \max_i \|p_{[i]}\|_\infty \qquad \|\mathbf{w}\|_\infty = \max_i \|\mathbf{w}_{[i]}\|_\infty$$

with $z \in \mathbb{Z}_q$ and $p \in R$. We use the notation $x \leftarrow \chi$ whenever we assign a uniformly random element of a set χ to a variable x. The symbol $||$ is used for the concatenation of two bit strings or two vectors/matrices.

2.2 Learning with errors

Learning with Errors (LWE) was first introduced by Regev [Reg05] and later expanded to polynomial rings by Lyubashevsky, Peikert and Regev [LPR10] to Ring Learning with Errors (RLWE). A Module Learning with Errors (MLWE) problem is obtained by setting the polynomial ring dimension of the RLWE problem to a dimension greater than one, thereby relying on multiple polynomial ring elements in the same instance. RLWE instances can be expressed in LWE form by representing the polynomial ring multiplication as a matrix vector product as recalled in [LZS+21, section 2]. In the following, we focus on the Search (as opposed to Decision) variants of these problems.

Search LWE problem. Let $\mathbf{A} \in \mathbb{Z}_q^{m \times n}$ and χ_e be a fixed distribution over $\mathbb{Z}$. The problem of recovering a secret $\mathbf{s} \in \mathbb{Z}_q^n$ given samples of the form: $(\mathbf{A}, \mathbf{t} = \mathbf{A} \cdot \mathbf{s} + \mathbf{e})$ with $\mathbf{e} \leftarrow \chi_e^m$ is known as the Search-LWE problem.

Methods of solving Search-LWE. There exists two main methods to solve a Search-LWE instance by lattice reduction: the primal-uSVP attack and dual attack [AGVW17]. The primal attack uses either Kannan's embedding [Kan87] or the Bai-Galbraith embedding [BG14] to construct an integer embedding lattice to solve the unique Shortest Vector Problem (uSVP). Using Kannan's embedding, recovering $\mathbf{s}$ and $\mathbf{e}$ given $\mathbf{t} = \mathbf{A} \cdot \mathbf{s} + \mathbf{e}$ where $\mathbf{t}, \mathbf{e} \in \mathbb{Z}_q^m$ and $\mathbf{s} \in \mathbb{Z}_q^n$ is as difficult as recovering the unique shortest non-zero vector $\mathbf{v} \in \mathbb{Z}^{m+n+1}$ from the embedding lattice $\mathbf{\Lambda}$ in Equation 1 with embedding parameter c and $\|\mathbf{v}\| \approx \sigma\sqrt{n+m}$ [ADPS16].

$$\mathbf{\Lambda} = \begin{bmatrix} q\mathbf{I}_m & 0 & 0 \\ \mathbf{A}^T & -\mathbf{I}_n & 0 \\ \mathbf{t} & 0 & c \end{bmatrix} \tag{1}$$

On the other hand, the dual attack solves Decision-LWE via reduction to the Short Integer Solution (SIS) problem, which in turn is reduced to finding short vectors in a lattice embedding [Ajt99].

There exists two main sets of algorithms for finding short vectors in lattices: enumeration and sieving. Enumeration algorithms perform an (exhaustive) search for an integer linear combination of the basis vectors, with well-known examples being LLL (Lenstra–Lenstra–Lovász) [LLL82] and BKZ (Block Korkine-Zolotarev) [Kor77, Sch87]. LLL can only find an approximation of shortest vector in polynomial time, and as such, is most commonly used as a pre-processing step in other lattice reduction algorithms. The BKZ-β algorithm repeatedly calls an enumeration SVP oracle for finding shortest vectors in dimension or block size β. The dimension of the underlying SVP Oracle, β, is the most widely used measure for cryptographic security of lattice-based cryptography as the time complexity of the BKZ algorithm is exponential in β. First introduced in [AKS01], lattice sieving algorithms find the shortest vector in a lattice by repeatedly computing linear combinations of vectors with the aim of producing shorter vectors. Whereas other algorithms require polynomial memory, lattice sieving algorithms have non-polynomial space complexity and typically require large magnitudes of memory.

The remainder of this work centers on utilizing the BKZ algorithm, which has been enhanced through the inclusion of extreme pruning techniques as outlined in [CN11] and commonly referred to as *BKZ 2.0*. Multiple open-source implementations of the BKZ algorithm exist, most notably FPLLL [dt21] (which is used in [DDGR20]) and NTL [Sho21].

2.3 CRYSTALS-Dilithium

Dilithium is a digital signature scheme based on the MLWE and MSIS (Module Short Integer Solution) problems [DKL+21]. Table 1 provides the Dilithium parameters for different NIST security levels[1].

Table 1: Dilithium parameters.

NIST Security level	II	III	V
q (modulus)	$2^{23} - 2^{13} + 1$	$2^{23} - 2^{13} + 1$	$2^{23} - 2^{13} + 1$
d (number of dropped bits from $\mathbf{t}$)	13	13	13
τ (# of $\pm$ 1's in c)	39	49	60
γ_1 ($\mathbf{y}$ coefficient range)	2^{17}	2^{19}	2^{19}
γ_2 (low order rounding range)	$(q\text{-}1)/88$	$(q\text{-}1)/32$	$(q\text{-}1)/32$
(k, ℓ) (dimensions of $\mathbf{A}$)	(4,4)	(6,5)	(8,7)
η (secret key range)	2	4	2
β $(= \tau \cdot \eta)$	78	196	120
ω (max. # 1's in $\mathbf{h}$)	80	55	75
average number of signing iterations	4.25	5.1	3.85

Signature generation. We describe the signature generation in Algorithm 1. For more details, e.g., regarding key generation or signature verification, we refer to [DKL+21]. First, the message M is hashed into a bit string μ. For deterministic signing, μ is hashed together with K to produce a seed ρ'. For the randomized version ρ' is generated randomly. This seed and a rejection counter κ (initially set to 0) are inputs to `ExpandMask` to sample the secret polynomial $\mathbf{y}$. Then, $\mathbf{w} = \mathbf{Ay}$ is decomposed into $\mathbf{w}_1$ and $\mathbf{w}_0$. The challenge $\tilde{c}$ is the hash of $\mu \| \mathbf{w}_1$. Next, $\tilde{c}$ is converted into a polynomial c that contains exactly τ coefficients set to ± 1 and the others set to zero. The vectors $\mathbf{z}$ and $\tilde{\mathbf{r}}$ are then computed from c, $\mathbf{y}$ and $\mathbf{w_0}$. For both security and correctness, two checks are performed:

$$\|\mathbf{z}\|_\infty < \gamma_1 - \beta, \qquad \|\tilde{\mathbf{r}}\|_\infty < \gamma_2 - \beta,$$

[1]Throughout this paper, when referring to Dilithium, we refer to version 3.1 of Dilithium released on 08/02/2021 for round 3 of the NIST PQC competition.

If any of the two checks do not pass, κ is incremented and the process starts over (from sampling a new $\mathbf{y}$). Otherwise, a hint $\mathbf{h}$ is computed, which is needed in the verification to account for the public key compression. Two more checks are performed on $c\mathbf{t}_0$ and $\mathbf{h}$. Again, if these checks do not pass the signature is rejected, κ is incremented and the process starts over. Otherwise, the signature $\sigma = (\tilde{c}, \mathbf{z}, \mathbf{h})$ is returned.

Algorithm 1 Sign(sk, M).

$\mathbf{A} = \texttt{ExpandA}(\rho)$
$\mu = \texttt{H}(tr \| M)$ ▷ $\mu \in \{0,1\}^{512}$
$\kappa = 0, (\mathbf{z}, \mathbf{h}) = \perp$
$\rho' = \texttt{H}(K \| \mu)$ (or $\rho' \xleftarrow{\$} \{0,1\}^{512}$ for randomized signing) ▷ $\rho' \in \{0,1\}^{512}$
while $(\mathbf{z}, \mathbf{h}) = \perp$ **do**
  $\mathbf{y} = \texttt{ExpandMask}(\rho', \kappa)$ ▷ $\mathbf{y} \in \tilde{S}^{\ell}_{\gamma_1}$
  $\mathbf{w} = \mathbf{A}\mathbf{y}$
  $(\mathbf{w_0}, \mathbf{w_1}) = \texttt{Decompose}(\mathbf{w}, 2\gamma_2)$
  $\tilde{c} = \texttt{H}(\mu \| \mathbf{w_1})$ ▷ $\tilde{c} \in \{0,1\}^{256}$
  $c = \texttt{SampleInBall}(\tilde{c})$ ▷ $c \in B_\tau$
  $\mathbf{z} = \mathbf{y} + c\mathbf{s_1}$
  $\tilde{\mathbf{r}} = \mathbf{w_0} - c\mathbf{s_2}$
  if $\|\mathbf{z}\|_\infty \geq \gamma_1 - \beta$ or $\|\tilde{\mathbf{r}}\|_\infty \geq \gamma_2 - \beta$ **then** $(\mathbf{z}, \mathbf{h}) = \perp$
  else
    $\mathbf{h} = \texttt{MakeHint}(\tilde{\mathbf{r}}, c, \mathbf{t_0}, \mathbf{w_1}, \gamma_2)$
    if $\|c\mathbf{t_0}\|_\infty \geq \gamma_2$ or the # of 1's in $\mathbf{h}$ is greater than ω **then** $(\mathbf{z}, \mathbf{h}) = \perp$
  $\kappa = \kappa + l$
return $\sigma = (\tilde{c}, \mathbf{z}, \mathbf{h})$

2.4 Fault injection attacks on Dilithium

Fault attacks against the signature generation of Dilithium have been the subject of several works in the literature with the majority exploiting the computation of the signature $\mathbf{z} = \mathbf{y} + c\mathbf{s}_1$. The ability to potentially establish a linear relation between the long-term secret key component $\mathbf{s}_1$ and public values, $\mathbf{z}$ and c has made this step a promising attack vector for a variety of adversaries [BBK16, BP18, RJH+19, IMS+22]. In addition, it has been shown that recovering $\mathbf{s}_1$ is sufficient to achieve existential forgery for Dilithium [BBK16, RJH+19].

In the following, we provide a short survey of fault attacks on Dilithium's signature generation algorithm and distinguish between the ones that only apply to the deterministic variant and the ones that apply to both deterministic and randomized Dilithium.

Deterministic and randomized Dilithium. Bettale et al. [BMR21] investigate the application of safe error attacks to PQC schemes. In the Dilithium case, the main targets are the secret vectors $\mathbf{s_1}$ and $\mathbf{s_2}$, since they have small coefficient values ($\in \{-\eta, \ldots, \eta\}$). The main objective of the attack is to fault each of the coefficients of $\mathbf{s_1}$ or $\mathbf{s_2}$ to zero and find faulted signatures which do not differ from unfaulted signatures, i.e., the secret key coefficient is indeed zero. However, knowing the zero coefficients in the secrets does not allow for straightforward key recovery but rather reduces the complexity of the underlying MLWE problem. Loop abort faults were first proposed by Page and Vercauteren [PV06]. Their use against lattice-based cryptosystems was first documented in [EFGT16] in an attack to target the sampling of $\mathbf{y}$ (`ExpandMask`), to generate masks of low degree by skipping the sampling of some coefficients in $\mathbf{y}$. The non-sampled coefficients of $\mathbf{y}$ are then assumed to be zero and, with enough (faulty) signatures, a solvable system of equations between $\mathbf{z}$ and $c\mathbf{s}_1$ can be established. The so-called nonce reuse attacks target sampling

in a variety of lattice-based schemes and have been demonstrated in [RRB+19]. This previous work applies to the Dilithium key generation and the main idea is to output faulty keys by not incrementing the nonce used to generate the secret key vectors $\mathbf{s}_1$ and $\mathbf{s}_2$, thus outputting weak keys. Islam et al. [IMS+22] propose a fault injection attack against both variants of Dilithium. The main idea behind this attack is that a single bit flip in the secret polynomials will result in (a limited number of) specific fault patterns in the faulty signature. These patterns can be recovered by exhaustively correcting the faulty signature and testing with the public key if it verifies. The number of recovered secret key bits depends on the number of injected bit flips and the distribution of the key coefficients.

Deterministic Dilithium only. The deterministic variant is naturally more vulnerable to DFA than the randomized one. The first DFA against the signing procedure of deterministic Dilithium was demonstrated by Bruinderink and Pessl [BP18], who showed that 65% of the execution of a deterministic Dilithium signature is vulnerable. Their work requires that an attacker can sign the same message M multiple times using the deterministic variant of Dilithium; first, to obtain a proper signature $\sigma = (c, \mathbf{z}, \mathbf{h})$ without any fault injected and then to produce a faulted signature σ' to obtain a faulty $\mathbf{z}'$ and c'. Using $\mathbf{z}$, c, $\mathbf{z}'$, c', the attacker can recover $\mathbf{s}_1$ by computing $\mathbf{s}_1 = \Delta c^{-1}\Delta\mathbf{z}$ where $\Delta c = c - c'$ and $\Delta\mathbf{z} = \mathbf{z} - \mathbf{z}'$. However, the faulted $\mathbf{z}'$ and c' must come from the same signing attempt (the same value of the counter κ) as the proper σ for the masks $\mathbf{y}$ to be equal. Naturally, this attack or more generally standard DFA does not apply to randomized Dilithium since an adversary cannot observe or fault two signatures for the same message with the same value of $\mathbf{y}$.

3 Attack description & practical consideration

At a high level, the attack proposed in this work leverages the possibility for an adversary to force (part of) polynomials in the vector of polynomials $\mathbf{y}$ to be equal during signature generation, i.e., $\mathbf{y}_{[i]} = \mathbf{y}_{[j]}$ for $i \neq j$. From the observation of this faulty signature $\mathbf{z}$, the adversary can compute the difference between signature polynomials $\Delta\mathbf{z} = \mathbf{z}_{[i]} - \mathbf{z}_{[j]} = c(\mathbf{s_1}[i] - \mathbf{s_1}[j])$ for $i \neq j$. With $\ell - 1$ such independent differences the underlying lattice problem of Dilithium is weakened, leading to possible key recovery. Namely, we demonstrate how the resulting faults can be used to perform a lattice attack. We first perform concrete attacks against the version of Dilithium considered in proofs (with $\mathbf{t}$ being public). In such a case, we are able to recover keys from all Dilithium parameter sets. Then, we discuss the applicability to concrete instantiations of Dilithium where only $\mathbf{t_1}$ (the upper bits of $\mathbf{t}$) is public. In such a case, the inserted faults lead to a significant drop in estimated security for all the parameter sets, but we report successful key recovery only against level V parameter set due to limitations in available computing resources.

In the rest of this section, we first describe how an instruction skip can be leveraged to insert such faults in Dilithium implementations. We also provide the number of faults needed when considering realistic adversarial capabilities, namely one instruction skip per signature. Especially, we discuss the applicability to randomized and deterministic versions of Dilithium. Second, we put forward how the attack turns Dilithium's MLWE problem into a RLWE one. In Section 4, we discuss the resulting estimated security and concrete attack results (when $\mathbf{t}$ is public) when the plain RLWE problem needs to be solved with Dilithium MLWE parameters. In Section 5, we put forward how partial secret key enumeration can be performed prior to the lattice reduction attack and then used into a lattice reduction framework, further reducing the attack runtime.

```
1  void polyvecl_uniform_gamma1(polyvecl *y, const uint8_t rhoprime[CRHBYTES
     ],    uint16_t kappa){
2    unsigned int i;
3    for(i = 0; i < L; ++i)
4    poly_uniform_gamma1(&y->vec[i], seed, L*kappa + i);
5  }
```

(a) C source code

```
1  push        { r3, r4, r5, r6, r7, lr }
2  mov         r5,r0
3  mov         r6,r1
4  uxth        r4,r2
5  add.w       r7,r0,#0x1000
6  loop:
7  mov         r2,r4
8  mov         r0,r5
9  mov         r1,r6
10 add.w       r5,r5,#0x400
11 bl          pqcrystals_dilithium2_ref_poly_uniform_gamma1
12 adds        r4, #0x1
13 cmp         r5,r7
14 uxth        r4,r4
15 bne         loop
16 pop         { r3, r4, r5, r6, r7, pc }
```

(b) Disassembled binary

Figure 1: Reference software for $\mathbf{y}$ sampling. *(a)* C source code. *(b)* Disassembled source code compiled with `arm-none-eabi-gcc` v10.3.1 with the following compiler flags: `-mcpu=cortex-m4 -mfloat-abi=hard -mfpu=fpv4-sp-d16 -O3 -fomit-frame-pointer`. Red corresponds to the targeted counter increment.

3.1 Vulnerability description

Weakness identification. As mentioned above, the goal of the adversary is to force (part of) polynomials in the vector $\mathbf{y}$ to be equal and build equations of the form $\Delta\mathbf{z} = \mathbf{z}_{[i]} - \mathbf{z}_{[j]}$ for $i \neq j$. Hence, a natural sweet spot for such a fault is the generation of these polynomials. Indeed, each polynomial $\mathbf{y}_{[i]}$ in $\mathbf{y}$ is derived from the `ExpandMask` function such that $\mathbf{y}_{[i]} \leftarrow$ `ExpandMask`$(\rho', \kappa * \ell + i)$ where κ is the rejection counter. The `ExpandMask` function referred to as `polyvecl_uniform_gamma1` in the reference implementation[2] is reported in Figure 1. This function generates all the ℓ polynomials of the vector $\mathbf{y}$ by repeatedly calling the function `poly_uniform_gamma1` with the same seed ρ' but incrementing the nonce with $\kappa * \ell + i$. Our new attack is based on skipping the instruction highlighted in red in the two listings. That is, by skipping the increment of the nonce in line 12 in Figure 1b, an attacker can force the reuse of the same nonce for the generation of two consecutive polynomials in $\mathbf{y}$. For instance, skipping the increment instruction during the first iteration of the loop enables to force $\mathbf{y}_{[0]} = \mathbf{y}_{[1]}$. Eventually, we note that repeating this fault can lead to forcing equivalence of two polynomials in $\mathbf{y}$ for multiple independent signatures which can be exploited as detailed in Subsection 3.2. We do not report experiments for injecting such a fault, however it has been shown in recent literature that the single instruction skip fault model is highly plausible to implement in practice with high precision and reproducibility [DRPR19, MDP+20, BFP19].

Next, we analyze aspects of the attack and strategies that affect the number of fault injections needed. Based on these previous aspects, we then provide the total number of instruction skips required.

[2] https://github.com/pq-crystals/dilithium/blob/master/ref/polyvec.c

Identifying successful fault injection. The attack relies on collecting $\ell - 1$ correct and independent equations of the form $\mathbf{z}[i] - \mathbf{z}[j] = c(\mathbf{s_1}[i] - \mathbf{s_1}[j])$ for $i \neq j$ from the returned signatures. We notice that an attacker can easily detect whether they have successfully injected a fault in the last signing attempt by observing the difference $\mathbf{z}_{[0]} - \mathbf{z}_{[1]}$. If indeed, $\mathbf{y}_{[0]} = \mathbf{y}_{[1]}$ then $\mathbf{z}_{[0]} - \mathbf{z}_{[1]} = c(\mathbf{s_1}_{[0]} - \mathbf{s_1}_{[1]})$ and we know from the distribution of the challenge polynomial and the secret vector that $c(\mathbf{s_1}_{[0]} - \mathbf{s_1}_{[1]}) \in [-2\tau\eta, 2\tau\eta]$. It is very unlikely with overwhelming probability that all coefficients of $\mathbf{z}_{[0]} - \mathbf{z}_{[1]}$ are in that small interval.[3] Fault attacks usually identify correctly inserted faults by comparing valid with (possibly) faulty signatures. Hence, the randomized version of Dilithium does not enable such fault identification by the adversary, However, we put forward that the previously described property enables an adversary to identify successful fault injection for both deterministic and randomized versions of Dilithium for arbitrary messages.

The attack requires faulty outputs, however the Dilithium signature generation usually involves a few attempts. Every time a signature is rejected, the counter κ is incremented and a new signature is generated. Accordingly, an attacker has no knowledge of the final signing iteration in order to inject a fault in its execution. The same holds even for faulty deterministic signatures, since the fault affects $\mathbf{y}$ and hence $\mathbf{z}, \mathbf{w}, c$ and $\tilde{\mathbf{r}}$, which then affect the rejection likelihood. Next, we discuss attack strategies for randomized and deterministic signing to deal with Dilithium's abort property and estimate the number of faults needed for the attack.

Attack strategy for randomized signatures. Recall that an attacker can identify successful fault injections by analyzing the returned signatures, as described previously. If it is not possible for an attacker to target the last iteration because of randomized signing or the fact that the fault changes the rejection behavior, they can target the first one. Based on Dilithium's aborts probability, the first iteration is the most likely to return a signature and succeeds with probabilities $p \approx 0.23$, $p \approx 0.20$ and $p \approx 0.26$ for levels II, III and V. Interestingly, the fault described earlier decreases the probability that a signature is rejected. We estimated by sampling a large number of signatures the probability of accepting a signature at a faulted first iteration and found that this probability increases to $p \approx 0.26$, $p \approx 0.21$ and $p \approx 0.27$ for levels II, III and V, respectively.[4] By targeting only the first iteration and taking into account the impact of the fault on the rejection probability, an attacker requires on average $1/p$ fault injections to acquire one faulty signature. This has been confirmed by sampling a large numbers of signatures and corresponds to ≈ 3.8, 4.7 and 3.7 fault injections for levels II, III and V. Eventually, to acquire the $\ell - 1$ faulty signatures needed to carry out the rest of the attack, on average 11.3, 18.8 and 22.2 fault injections are needed for levels II, III and V, respectively.

Improved attack strategy for deterministic signatures. In the deterministic case, an attacker can determine the expected final signing iteration since signing the same message multiple times always results in the same signature and the same number of iterations. For the randomized case, we proposed to always target the first iteration since it maximizes the probability of accepting a signature. For the deterministic case, this probability is maximized by faulting the final iteration. However, this probability is not 1, since faulting the generation of $\mathbf{y}$ also affects $\mathbf{w}, c$ and $\tilde{\mathbf{r}}$. Hence, the fault could lead to a signature being rejected despite being previously accepted when no fault was injected. We estimated by sampling that the probability of accepting a signature after the fault at the previously determined accepted iteration is $p \approx 0.4$, $p \approx 0.31$ and $p \approx 0.38$ for levels II, III and V,

[3] This occurs with probability $(\frac{2\tau\eta+1}{2\gamma_1})^n$ assuming the coefficients of $\mathbf{z}$ are uniform and independent.

[4] This probability can also be approximated by using Equation 5 from [DKL+21], by applying the new effective ℓ, which is the number of distinct polynomials in $\mathbf{y}$ after the fault. This however does not take into account the rejection probability on $c\mathbf{t_0}$ or $\mathbf{h}$.

respectively. These probabilities are naturally higher than the ones determined previously for the randomized case when always targeting the first signing iteration. Accordingly, in the deterministic case an attacker requires on average 2.4, 3.2 and 2.6 fault injections for levels II, III and V to acquire one faulty signature. We estimated that on average 7.3, 13 and 15.8 fault injections are needed for levels II, III and V, respectively. In addition, as opposed to the randomized case, valid signatures are needed to determine the expected final signing attempt.

Overview of number of fault injections required. In the following, we provide an overview of the number of fault injections required for the attack when skipping one instruction per signature. The attack can also be carried out with $\ell - 1$ faults in a single signature, however this can be quite challenging in practice. Recall that the attack requires $\ell - 1$ independent equations of the form $\Delta\mathbf{z} = c(\mathbf{s_1}_{[i]} - \mathbf{s_1}_{[j]})$ for $i \neq j$. Collecting these equations requires skipping one instruction in the last signing iteration for $\ell - 1$ signatures. We have estimated for both deterministic and randomized Dilithium, following the previously described strategies, the number of fault injections needed for an attacker to acquire such $\ell - 1$ faulty signatures. The results are shown in Figure 2 where the x-axis corresponds to the total number of fault injections and the y-axis to the probability of acquiring n_σ faulty signatures such that $n_\sigma \geq \ell - 1$. To illustrate how to interpret this figure, we provide a few examples. For instance, for randomized Dilithium II, with 10 fault injections, an attacker acquires the $\ell - 1$ required faulty signatures with probability 0.5. Eventually and for all security levels, with approximately 40 fault injections for randomized Dilithium and approximately 25 fault injections for deterministic Dilithium, the attacker will most likely succeed in acquiring the $\ell - 1$ faulty signatures.

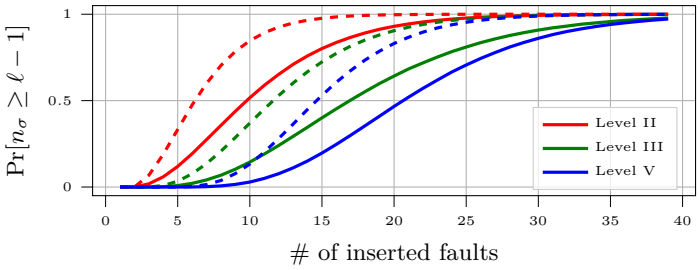


Figure 2: Probability that the number of released faulted signatures n_σ is at least $\ell - 1$ after inserting a single instruction skip per independent execution of $\mathrm{Sign}(sk, M)$, hence enabling full secret key recovery. Continuous lines stand for randomized Dilithium. Dashed lines stand for deterministic Dilithium.

3.2 From MLWE to RLWE

For simplicity, in the rest of this work, we assume that an adversary was able to inject $(\ell - 1)$ instruction skips during the final signing iteration such that all the polynomials of $\mathbf{y}$ are equal, i.e., $\forall i \in \{0, 1, \ldots, \ell - 1\}, \mathbf{y}_{[i]} = y$. Accordingly, such a faulty signature is of the form:

$$\begin{bmatrix} \mathbf{z}_{[0]} \\ \mathbf{z}_{[1]} \\ \vdots \\ \mathbf{z}_{[\ell-1]} \end{bmatrix} = \begin{bmatrix} y \\ y \\ \vdots \\ y \end{bmatrix} + c \cdot \begin{bmatrix} \mathbf{s_1}_{[0]} \\ \mathbf{s_1}_{[1]} \\ \vdots \\ \mathbf{s_1}_{[\ell-1]} \end{bmatrix} \tag{2}$$

From this, we express the $\ell - 1$ differences with the first polynomial in $\mathbf{s_1}$ and the following ones as:

$$\begin{bmatrix} \mathbf{s_1}_{[0]} - \mathbf{s_1}_{[1]} \\ \mathbf{s_1}_{[0]} - \mathbf{s_1}_{[2]} \\ \vdots \\ \mathbf{s_1}_{[0]} - \mathbf{s_1}_{[\ell-1]} \end{bmatrix} = \begin{bmatrix} c^{-1}(\mathbf{z}_{[0]} - \mathbf{z}_{[1]}) \\ c^{-1}(\mathbf{z}_{[0]} - \mathbf{z}_{[2]}) \\ \vdots \\ c^{-1}(\mathbf{z}_{[0]} - \mathbf{z}_{[\ell-1]}) \end{bmatrix} = \begin{bmatrix} \boldsymbol{\lambda}_{[0]} \\ \boldsymbol{\lambda}_{[1]} \\ \vdots \\ \boldsymbol{\lambda}_{[\ell-2]} \end{bmatrix} = \boldsymbol{\lambda} \tag{3}$$

where the vector of polynomials $\boldsymbol{\lambda}$ is constructed from the faulty signatures. Indeed, the challenge polynomial c is most likely invertible thanks to the Dilithium polynomial ring[5] and is part of the signature. Concretely, Equation 3 illustrates that recovering one polynomial of $\mathbf{s_1}$ enables the recovery of the full secret vector $\mathbf{s_1}$. Indeed, this linear system has $\ell - 1$ independent equations with ℓ unknowns. Similar equations can be derived from multiple faulty signatures by simply taking into account the different $\mathbf{z}$s and cs. Which pairwise differences of the polynomials of $\mathbf{s_1}$ an attacker gets do not matter, as long as they obtain $\ell - 1$ independent ones, which is the case when skipping the increment in the `ExpandMask` function as detailed in the previous section.

Next, we put forward how one secret polynomial can be recovered by the adversary. For simplicity, we only describe the methodology for $\mathbf{s_1}_{[0]}$. The same strategy can be used to recover any $\mathbf{s_1}_{[i]}$ and from Equation 3 recovering all the remaining $(\ell - 1)$ polynomials of $\mathbf{s_1}$. Concretely, we observe that the MLWE instance of Dilithium $\mathbf{t} = \mathbf{A} \cdot \mathbf{s_1} + \mathbf{s_2}$ can be expressed by exploiting the linear system of equations in Equation 3 as:

$$\begin{bmatrix} \mathbf{t}_{[0]} \\ \mathbf{t}_{[1]} \\ \vdots \\ \mathbf{t}_{[k-1]} \end{bmatrix} = \mathbf{A} \cdot \begin{bmatrix} \mathbf{s_1}_{[0]} \\ \mathbf{s_1}_{[0]} - \boldsymbol{\lambda}_{[0]} \\ \vdots \\ \mathbf{s_1}_{[0]} - \boldsymbol{\lambda}_{[\ell-2]} \end{bmatrix} + \begin{bmatrix} \mathbf{s_2}_{[0]} \\ \mathbf{s_2}_{[1]} \\ \vdots \\ \mathbf{s_2}_{[k-1]} \end{bmatrix} = \mathbf{A} \cdot \begin{bmatrix} \mathbf{s_1}_{[0]} \\ \mathbf{s_1}_{[0]} \\ \vdots \\ \mathbf{s_1}_{[0]} \end{bmatrix} - \mathbf{A} \cdot \begin{bmatrix} 0 \\ \boldsymbol{\lambda}_{[0]} \\ \vdots \\ \boldsymbol{\lambda}_{[\ell-2]} \end{bmatrix} + \begin{bmatrix} \mathbf{s_2}_{[0]} \\ \mathbf{s_2}_{[1]} \\ \vdots \\ \mathbf{s_2}_{[k-1]} \end{bmatrix} \tag{4}$$

Expanding the matrix multiplication, the previous equality is equivalent to:

$$\begin{bmatrix} \mathbf{t}_{[0]} \\ \mathbf{t}_{[1]} \\ \vdots \\ \mathbf{t}_{[k-1]} \end{bmatrix} + \begin{bmatrix} \sum_{j=1}^{\ell-1} \mathbf{A}_{[0,j]} \cdot \boldsymbol{\lambda}_{[j-1]} \\ \sum_{j=1}^{\ell-1} \mathbf{A}_{[1,j]} \cdot \boldsymbol{\lambda}_{[j-1]} \\ \vdots \\ \sum_{j=1}^{\ell-1} \mathbf{A}_{[k-1,j]} \cdot \boldsymbol{\lambda}_{[j-1]} \end{bmatrix} = \begin{bmatrix} \sum_{j=0}^{\ell-1} \mathbf{A}_{[0,j]} \cdot \mathbf{s_1}_{[0]} \\ \sum_{j=0}^{\ell-1} \mathbf{A}_{[1,j]} \cdot \mathbf{s_1}_{[0]} \\ \vdots \\ \sum_{j=0}^{\ell-1} \mathbf{A}_{[k-1,j]} \cdot \mathbf{s_1}_{[0]} \end{bmatrix} + \begin{bmatrix} \mathbf{s_2}_{[0]} \\ \mathbf{s_2}_{[1]} \\ \vdots \\ \mathbf{s_2}_{[k-1]} \end{bmatrix} \tag{5}$$

where each of the rows is a RLWE problem. For instance, the first row is:

$$\mathbf{t}_{[0]} + \sum_{j=1}^{\ell-1} \mathbf{A}_{[0,j]} \cdot \boldsymbol{\lambda}_{[j-1]} = \sum_{j=0}^{\ell-1} \mathbf{A}_{[0,j]} \cdot \mathbf{s_1}_{[0]} + \mathbf{s_2}_{[0]} \tag{6}$$

where the left part of the equation is known to the adversary thanks to the standard Dilithium MLWE instance and public key (giving $\mathbf{t}$ and $\mathbf{A}$) and the $\boldsymbol{\lambda}$ from the faulty signatures. The right part of the equation depends on two single secret polynomials $\mathbf{s_1}_{[0]}$ and $\mathbf{s_2}_{[0]}$. Therefore, our attack reduces the Dilithium MLWE instance of dimension (k, ℓ) to a $(1, 1)$ instance (hence a RLWE instance) using the same polynomial ring with dimension $n = 256$. Hence, the resulting RLWE has a reduced security compared to the original MLWE problem. In Section 4.1, we explore the feasibility of recovering the full key by solving this reduced complexity RLWE problem through lattice reduction.

[5] If c is not invertible, then the equations can be expressed as a matrix vector product thanks to the polynomial ring structure or the attack can simply be repeated for another signature until c is invertible.

A note on key compression. Above, we described how the attack decreases the security of Dilithium's underlying MLWE problem. This attack assumes that Dilithium's public key contains $\mathbf{t} = \mathbf{A} \cdot \mathbf{s_1} + \mathbf{s_2}$, as considered in the security proofs. However, to reduce the public key size, the complete $\mathbf{t}$ is decomposed into a high part $\mathbf{t_1}$ which is part of public key and a low part $\mathbf{t_0}$ which is part of the secret key with the relation $\mathbf{t} = \mathbf{t_1} \cdot 2^d + \mathbf{t_0}$. In previous attacks, since the knowledge of $\mathbf{t_0}$ is typically required to recover the secret vector $\mathbf{s}_2$, it has been shown that recovering $\mathbf{s}_1$ only is sufficient to achieve existential forgery for Dilithium [BBK16, RJH+19]. In our attack, the knowledge of $\mathbf{t_0}$ is not required for the MLWE to RLWE reduction part of our attack as $\mathbf{t_0}$ can be embedded into the additive noise vector (moving $\mathbf{t_0}$ to the right part of Equation 6). However, it has an impact on the resulting security of the RLWE problem making it harder to solve. As it is unclear if $\mathbf{t_0}$ must be considered secret or public (it has been hinted that $\mathbf{t_0}$ can be recovered from enough signatures in [Lyu22, RJH+18, RRB+19]), we take a worst-case approach for the rest of this work. If not specified in the following sections, the full $\mathbf{t}$ is assumed to be public. In Subsection 5.3, we derive the impact of fully secret $\mathbf{t_0}$ on the complexity of the (reduced) RLWE instance. We hope that this approach gives a complete view to the reader about the applicability of the attack to Dilithium.

Eventually, once the missing polynomial $\mathbf{s_1}_{[0]}$ is obtained thanks to a lattice reduction attack, the complete $\mathbf{s_1}$ can be recovered thanks to the linear system of equations. The secret polynomial $\mathbf{s_2}$ can be obtained thanks to Equation 5 when $\mathbf{t}$ is known. In such a case, the relations put forward in Appendix A can also be used. If only $\mathbf{t_1}$ is known, the adversary can leverage the results from [BBK16, RJH+19] to forge valid signatures.

4 Impact on estimated security & key recovery

In the previous section, we showed that faulty signatures enable to reduce the MLWE instance of Dilithium to a RLWE one. In the following, we evaluate the complexity of solving the RLWE problem in Equation 5 using a lattice reduction. Concretely, we first estimate the security parameter β with various tools from the literature, both for MLWE and RLWE. This enables to quantify the security reduction. Second, we solve concrete RLWE instances and report the runtime. The results presented in this section are summarized in Table 2. Later in Section 5, we will highlight how side-information can be used to decrease even more the security of the RLWE instance [DDGR20].

In the following, we denote by *pre-attack* the unfaulted case, i.e., the standard hardness of Dilithium's MLWE problem. Accordingly, *post-attack* stands for the case where the adversary inserted the required faults resulting in an easier RLWE problem. In both cases, $\mathbf{t}$ is considered public.

4.1 Post-attack hardness estimates

In order to estimate the hardness of MLWE and RLWE instances, we use the estimator in [DDGR20]. This estimator has been selected as it allows easily integrating side-information (see Section 5), perform attacks and estimate the BKZ β. In addition, it was observed by the authors to be accurate for small dimension lattices which is the case in our scenario (we compare various estimators in Appendix B and observe the same behavior).

The results are provided in Table 2 for the three Dilithium NIST security levels. The top half of the table corresponds to the standard Dilithium security prior to the attack and the bottom half of the table to the reduced security after the attack by exploiting the reduction from MLWE to RLWE resulting from the induced faults. The dimensions for

[6] https://github.com/pq-crystals/security-estimates

[7] Produced using the "DBDD_predict" variant of the framework. Results are rounded up to next integer as estimator outputs BKZ block-size estimations with 2 decimal places for finer security estimations.

Table 2: Summary of estimated hardness reduction from the fault attack and runtime of resulting RLWE solving. Both are obtained using the tools provided in [DDGR20][7]. For concrete solving, we provide minimum, average and maximum values observed over 10 random faulted signatures with random keys (in the form min/avg/max).

NIST Security Level	II	III	V
LWE Hardness (pre-attack)			
Dimension $(n \cdot k, n \cdot \ell)$	(1024, 1024)	(1536, 1280)	(2048, 1792)
Secret key range η	2	4	2
Estimated β	434	641	890
LWE Hardness (post-attack)			
LWE dimension (n, n)	(256, 256)	(256, 256)	(256, 256)
Estimated β	62	68	62
Actual β	57/59/61	-	57/59/61
Runtime (hh:mm)	21:40/40:05/54:50	-	21:40/40:05/54:50
Success Rate	100%	-	100%

the respective problems are also provided. From Table 2, it is clear that after the fault attack the baseline security is significantly decreased. For illustration, the estimated BKZ block-size β is reduced from 434 to 62 for Dilithium level II, from 641 to 68 for level III and from 890 to 62 for level III. Interestingly, we note that for level II and V, the resulting RLWE is similar as the resulting dimensions are the same as well as the noise range η. The only difference is the number of faults that need to be injected to reduce the MLWE down to RLWE. For level III, the noise range is slightly larger, hence the estimated β is slightly higher.

4.2 Solving RLWE instances

For all versions, the significant hardness reduction leads to much smaller LWE dimensions and BKZ block-sizes, and hence to practical lattice reduction attacks. For this purpose, we use the toolbox provided by Dachman-Soled et al. [DDGR20]. The Search-LWE problem is solved via primal-uSVP and Kannan's embedding (as mentioned in Subsection 2.2). In terms of setup, we use 16 cores of an Intel Xeon processor running at 2.75 GHz.

The results of the lattice reduction attacks are shown in Table 2. In particular, we provide the minimum, average and maximum BKZ block-size β, along with the minimum, average and maximum runtime observed over 10 lattice reductions for random keys for Dilithium level II. The results for levels II and V are identical since the LWE instances are equivalent as discussed above. Over our 10 experiments with level II and V parameters, we were always able to recover the polynomial $\mathbf{s_1}_{[0]}$ in Equation 6 with an average runtime of 40 hours. The rest of the secret key being trivially obtained from Equation 3. Interestingly, we observe that the actual β is close to the estimated one. The actual β is on average equal to 59 and was estimated to be 62, confirming that the estimations of [DDGR20] are accurate in this case. We did not perform the attack for level III since the attack time would be prohibitive with our current setup (estimated BKZ block-size β for level III is 68). However, from Table 2 it is clear that since the block-size for level III is only marginally higher than the one for levels II and V, the lattice reduction should still be practical, potentially with lattice sieving.

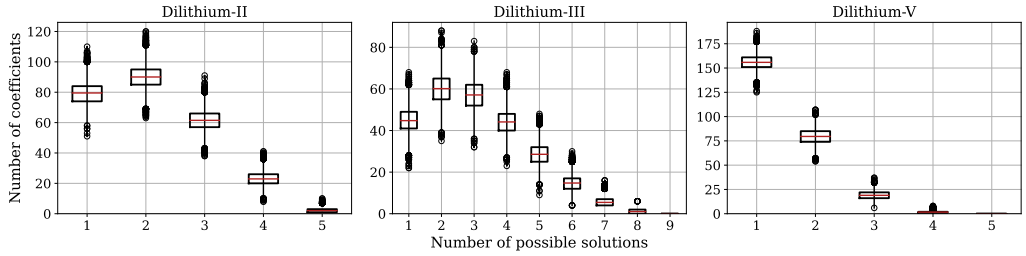


Figure 3: Frequency of the number of remaining possibilities after enumeration.

5 Improving attacks with side-information

In the previous sections, we showed that it is possible to recover the secret signing key of Dilithium in very few signatures using a combined fault injection and lattice reduction attack. The lattice reduction attack succeeds without using any additional information and instead only relying on the faulty signatures.

In the following, we first show in Subsection 5.1 how additional knowledge on the secret key, referred to as side-information (or hints) in [DDGR20], can be recovered from the faulty signatures. This additional knowledge can be plugged into to the solver to further reduce the complexity of the previous lattice reductions. Subsequently, in Subsection 5.2 we provide security estimates and the runtime of the improved attack. These results are summarized in Table 3. In Subsection 5.3, we explore the possibility to recover the secret without the full knowledge of $\mathbf{t}$, and instead only the high part $\mathbf{t_1}$ that is shared as part of the public key after key compression. Finally, in Subsection 5.4, we compare these results with other published attacks.

5.1 Side-information from partial enumeration

In addition to reducing the number of dimensions of the MLWE problem, the faults can also be used to gain additional information on each of the coefficients $\mathbf{s_1}_{[0,j]}$ in $\mathbf{s_1}_{[0]}$ independently. Indeed, for each of these coefficients, some value can never be compatible with the linear system of equations in Equation 3. Concretely, we first observe that these values are uniformly distributed on a small range $[-\eta, \eta]$ with $\eta \in \{2, 4\}$. Then, the linear system in Equation 3 can be expressed for each of the n coefficients independently. As a result, it is possible to narrow down the possible values for each $\mathbf{s_1}_{[0,j]}$ independently. This can be efficiently achieved by enumerating over all possibilities for the ℓ coefficients $\mathbf{s_1}_{[:,j]}$ at a particular index j. The plausible values of $\mathbf{s_1}_{[0,j]}$ are then obtained by listing all the $\mathbf{s_1}_{[:,j]}$ leading to a valid solution of Equation 3. Overall, this process involves n enumerations over $(2\eta + 1)^\ell$ possibilities.

The results of such an enumeration are plotted in Figure 3 for Dilithium with NIST security levels II, III and V, from left to right. The enumeration is performed by simulating our attack for 10,000 random Dilithium keys for each level. On all the plots the x-axes correspond to the number of possibilities remaining after enumeration for a single coefficient index $\mathbf{s_1}_{[0,j]}$, i.e., the number of possible solutions or the size of the solution space for a single coefficient after solving the system of equations. This value ranges from 1 (the coefficient is known) to $2\eta + 1$ (no side-information is recovered). The y-axes correspond to the number of coefficients in the polynomial $\mathbf{s_1}_{[0]}$ that are known up to a number of possibilities on the x-axis. Since these results are provided for a set of 10,000 random keys for each plot, the red crosses indicate the average number of coefficients with x remaining possibilities. Solid lines indicate key space reduction within $\pm$ 1.5 the interquartile range from the median, while the hollow circles indicate outliers.

First, we observe from Figure 3 that the number of coefficients fully recovered is relatively high. Specifically, on average we fully recover ≈ 80 coefficients (31%), ≈ 45 (17%) and ≈ 156 (61%) out of the total 256 coefficients for all secret key polynomials, for level II, III and V, respectively. Many other coefficients are also reduced to 2 possibilities (for instance ≈ 90 and ≈ 60 coefficients for level II and III, respectively) and in general to less than $2\eta + 1$ possibilities. No information is recovered on a very small number of coefficients for which the initial $2\eta + 1$ possible values remain. Accordingly, in addition to reducing the MLWE problem to a significantly easier RLWE problem, the attack is also able to recover a significant portion of the secret key coefficients and partial information on the remaining ones.

Interestingly, the key recovery proportion depends on the security level, precisely on the values of the parameters η and ℓ. First and naturally, the key recovery potential is lower for level III since the key space or the solution space for the system of pairwise differences is larger for $\eta = 4$ compared to levels II and V for which $\eta = 2$. Second and surprisingly, the increase in ℓ leads to more coefficients being recovered, as illustrated by the enumeration results for level V. As opposed to standard MLWE security where increasing ℓ results in harder instances, in this case the increase in ℓ negatively impacts security (but still increasing the number of faults needed). This is due to having more equations in the system we use for enumeration and hence to more constraints which reduce the key space.

5.2 Lattice reduction attack with hints

In the following, we use the toolkit from Dachman-Soled et al. [DDGR20] to evaluate the impact of the side-information obtained in the previous section on the hardness of the lattice problem. Indeed, it provides the ability to insert leaked information about secrets and/or errors into lattice reduction attacks and estimate their complexity. The resulting estimates are available in Table 3.

Table 3: Summary of estimated hardness reduction from the fault attack with side-information and runtime of resulting RLWE solving. Both are obtained using the tools provided in [DDGR20][7]. We provide minimum, average and maximum values observed over 10 random faulted signatures with random keys (in the form min/avg/max). The total runtime is the sum of the BKZ runtime and the hint integration.

NIST Security Level	II	III	V
LWE Hardness (pre-attack)			
Lattice dimension	2049	2817	3841
Secret key range η	2	4	2
Estimated β	434	641	890
LWE Hardness (post-attack with side-information)			
Lattice dimension	426/438/450	449/465/472	267/340/359
Estimated β	4/15/27	35/41/43	2/2/2
Actual β	21/29/35	41/48/54	2/2/2
BKZ Runtime (hh:mm)	01:01/01:53/02:54	03:44/09:55/19:01	00:03/00:07/00:13
Total Runtime (hh:mm)	13:08/15:10/19:38	11:04/17:22/24:06	20:38/23:12/25:50
Success Rate	100%	100%	100%

Integration of perfect hints. As mentioned above, some secret key coefficients can be determined uniquely simply thanks to enumeration. This knowledge can be integrated as *Perfect Hints* in the [DDGR20, Definition 23] framework and provides the strongest form

of side information. In addition, the framework can be used to implicitly integrate *Short Vector Hints* [DDGR20, Definition 30]. Concretely, all the prefect hints are first integrated, and the short vector hint is then integrated (its computation is integrated into the toolkit).

In addition, the framework of Dachman-Soled et al. considers so-called modular and approximate hints. These correspond in the context of our attack to key coefficients that are not uniquely determined with enumeration and their integration could lead to improved attacks. We leave the study of these kinds of hints for future works since in the context of our attack perfect and short vector hints are sufficient to significantly reduce the complexity of the RLWE problem and lead to practical key recovery. In the following, we refer to both perfect and short vector hints as simply hints.

Estimated & concrete security. The results of the lattice reduction attacks using hints are provided in Table 3. In this table, both estimates and concrete key recovery results are averaged over 10 experiments as the number of hints may vary for different secret keys as illustrated in Figure 3. We again provide the minimum, average and maximum BKZ block-size β, along with the minimum, average and maximum runtime. Concretely, our experiments illustrate that the integration of hints significantly reduces the hardness of the lattice problem. First comparing with Table 2, the lattice dimension decreases from 513 for all NIST security levels without hints down to 438, 465 and 340 on average for level II, III and V, respectively. Similarly, it reduces the block-size β. As an example for level II, the actual (resp. estimated) β is on average decreased from 59 (resp., 62) down to 29 (resp., 15)[8]. Interestingly, the improvement offered by the integration of hints is larger for level V, as the number of available perfect hints is higher compared to level II and III (see Figure 3).

Next, the runtime of the attack is also significantly reduced. With our resources, from $\approx$ 40 hours on average to $\approx$ 15 hours for level II and $\approx$ 23 hours for level V. The attack is also practical for level III, with an average runtime of $\approx$ 17 hours. In particular, we note that with the large number of hints we are able to extract, the total runtime of the attack is actually dominated by the hint integration. Despite the long hint integration runtime, our results show that integrating short vector hints significantly decreased the dimension of the instance and significantly improved lattice reduction time. As previously mentioned, this suggests that for practical attacks it might be possible to reach some kind of trade-off between the number of hints integrated and the remaining BKZ complexity to minimize the total runtime. This question is left for future work.

5.3 Lattice reduction attack with hints without knowledge of $\mathbf{t_0}$

A key feature of Dilithium is public key compression. That is, the key generation algorithm only publicly outputs $\mathbf{t}_1$ such that $\mathbf{t}_1 2^d + \mathbf{t}_0 = \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2$. In the previous sections, we assumed that the attacker has knowledge of $\mathbf{t_0}$, the lower 13 bits of the public key. While $\mathbf{t_0}$ is not assumed to be secret and can potentially be recovered from enough signatures as hinted in [Lyu22, RJH+18, RRB+19], we still explore the possibility for key recovery without knowledge of $\mathbf{t_0}$. For this purpose, we essentially include $\mathbf{t}_0$ as part of the error, which was only $\mathbf{s_2}$ in the previous sections. This increases the support of the error distribution from $[-\eta, \eta]$ for $\mathbf{s_2}$ to $[-\eta - 2^{d-1}, \eta + 2^{d-1}]$ for $\mathbf{s_2} - \mathbf{t_0}$. For key recovery, we apply the same previously described lattice reduction methodology to recover $\mathbf{s}_1$. Unlike Section 5 where lattice reduction will also recover $\mathbf{s}_2$, this method will only yield $\mathbf{s}_2 - \mathbf{t}_0$ and will not result in a straight-forward recovery of the private key component $\mathbf{s}_2$. However, $\mathbf{s}_1$ is sufficient to achieve existential forgery as shown in [RJH+18].

The lattice reduction estimates using hints but without knowledge of $\mathbf{t_0}$ are given in Table 4. As expected, since the variance of the error distribution is larger, the attacks are

[8]The discrepancies between actual and estimated small block-sizes are discussed in [DDGR20, page 12].

more difficult than with the knowledge of $\mathbf{t_0}$. However, we can still observe a significant reduction of the security level. It reduces on average from $\beta = 434$ to $\beta = 94$ for level II, and from $\beta = 641$ to $\beta = 136$ for level III. Notably for level V, since more perfect hints can be recovered on the secret, security is reduced from $\beta = 890$ to $\beta = 49$. Such a block-size is practical on our setup as illustrated in Table 3. Indeed, successful attacks are reported for $\beta = 48$ and higher dimensions.

Table 4: Summary of estimated hardness reduction (assuming $\mathbf{t_0}$ is secret) from the fault attack with side-information and runtime of resulting RLWE solving. Both are obtained using the tools provided in [DDGR20][7]. We provide minimum, average and maximum values observed over 10 random faulted signatures with random keys (in the form min/avg/max).

NIST Security Level	II	III	V
LWE Hardness (pre-attack)			
Lattice dimension	2049	2817	3841
Secret key range η	2	4	2
Estimated β	434	641	890
LWE Hardness (post-attack with side-information)			
Lattice dimension	403/433/462	445/468/491	325/357/388
Estimated β	73/94/117	114/136/159	28/49/65

5.4 Comparison to related attacks

In this section we compare our attack to state-of-the-art fault injection attacks against Dilithium. For consistency and to simplify comparison in future works we extend the table provided by Ravi et al. [RCDB22] to include our attack. The columns Attack_Vector and Countermeasure have been removed for conciseness, since instruction skipping faults can be achieved by different means and countermeasures against our attack are discussed in Section 6. For the full table we refer the reader to [RCDB22, Table 3]. Notably, the attack characteristic column in our case essentially captures the difference between deterministic and randomized signing. This characteristic is defined in [RCDB22], and namely with the ability to communicate with the target device (Communicate_DUT_IO) an attacker can request signatures for specific messages. In the deterministic case this is useful since it allows an attacker to sign the same message twice or multiple times and hence identify the expected final signing iteration. In the randomized case, this is not possible and it usually suffices to observe the returned signature (Observe_DUT_IO).

Table 5: Extension of [RCDB22, Table 3] with our attack. The number of executions is given on average for all Dilithium security levels (in the form level II/level III/level V).

Attack	**Attack Characteristic**			
	DUT_IO_Access	Targeted_Or_Not	Num_Faults	Num_Executions
This work	Communicate_DUT_IO	Targeted_Fault	1	7.3 / 13 / 15.8 †
	Observe_DUT_IO	Targeted_Fault	1	11.3 / 18.8 / 22.2

† additional valid signatures are required in the deterministic case to determine the expected final signing iteration and is not accounted for in the average number of faulted signatures.

6 Re-computation and norm-based countermeasures

In the following, we present and analyze two countermeasures to protect Dilithium against the attack described earlier in this paper, when an attacker can induce one fault per signature. This is motivated by the fact that verification after signing does not prevent the attack, since despite the fault, the returned signature is still valid. Indeed, the whole signature stems from the pseudo randomly generated $\mathbf{y}$ and the message, so generating a faulty $\mathbf{y}$ at the beginning of a signing attempt is virtually the same as signing with a different random $\mathbf{y}$. Interestingly however, on memory-constrained devices $\mathbf{y}$ is typically generated twice since it is used at two different stages of the signing algorithm [GKS21, BRS22]. Naturally, faulting only one out of the two generations of $\mathbf{y}$ leads to a faulty signature and hence the attack would be detected by verification after signing. In the following, we focus on implementations which do not regenerate $\mathbf{y}$.

We first discuss a simple trick related to re-computation based detection for Fiat-Shamir with aborts signatures. We then also present a more efficient countermeasure in comparison to re-computation which protects specifically against the attack presented in this work. Note that ensuring the control flow integrity of the signature generation can prevent the attack, by making sure that the increment of the nonce cannot be skipped, still in this section we suggest and discuss efficient algorithmic countermeasures.

6.1 Re-computation for Fiat-Shamir with aborts

This section is based on the two following observations. First, almost all known fault attacks on Dilithium (safe error and ineffective fault attacks excluded) require access to the returned signature. Second, one particular property of Fiat-Shamir with aborts signature schemes such as Dilithium is that signatures are checked before being released. In the case of Dilithium, if the norm checks on $\mathbf{z}$ and $\tilde{\mathbf{r}}$ do not pass, the signature generation is aborted and repeated starting from the generation of $\mathbf{y}$ with an incremented counter κ. On average, Dilithium requires 3 to 5 attempts depending on the parameter set as recalled in Table 1, but even 10 or 20 attempts could be observed with non-negligible probability before accepting a signature[9].

A full re-computation of a signature generation leads to a 100% overhead. Based on the previous observations, we propose to significantly reduce this overhead by only re-computing the last signing attempt, hence ensuring that the released final signature is not faulty or detecting any faults in the final signature and not releasing it. An extension of this proposal is to use lightweight and more efficient countermeasures for all the signing attempts and use stronger but potentially more expensive countermeasures during the re-computation of the last attempt. In the following, we simply focus on re-computing the last signing attempt to estimate the benefit of this strategy. These estimations are given in Table 6, where for instance when 5 signing attempts are required (which is close to the average number of signing attempts for Dilithium III) to generate a signature, re-computing only the last attempt induces only a 20% overhead instead of a 100% when re-computing all signing attempts.

Table 6: Comparison between re-computing every signing attempt and re-computing only the last valid signing attempt. Values represent the overhead on top of the regular signing, which is constant at 100% when re-computation is applied to all attempts.

# signing attempts	1	2	3	4	5
Re-computing signature generation	100%	100%	100%	100%	100%
Re-computing only valid attempt	100%	50%	33%	25%	20%

[9]The number of signing attempts follows a geometric distribution with parameter $p \approx 0.23$, $p \approx 0.20$ or $p \approx 0.26$.

6.2 Norm-based fault detection countermeasure

In this section, we propose an efficient countermeasure meant to protect against the attack presented in this paper. We first explain the idea behind the countermeasure, then provide an algorithm detailing it and finally analyze it, in particular with respect to false positives. This countermeasure uses the same trick described previously, i.e., it is only performed for the final valid signature, however, it is also more efficient since it avoids re-computing a whole signing attempt.

Notations and reminders. For simplicity, we will use $\Delta\mathbf{z}, \Delta\mathbf{y}, \Delta\mathbf{s_1}, c\Delta\mathbf{s_1}$ to denote differences of the form $\mathbf{z}_{[i,j]} - \mathbf{z}_{[i',j]}, \mathbf{y}_{[i,j]} - \mathbf{y}_{[i',j]}, \mathbf{s_1}_{[i,j]} - \mathbf{s_1}_{[i',j]}, c(\mathbf{s_1}_{[i]} - \mathbf{s_1}_{[i']})_{[j]}$, respectively. Recall that the secret vector $\mathbf{s_1}$ is sampled similarly to $\mathbf{y}$ but from the much smaller range $[-\eta, \eta]$ with $\eta \in \{2, 4\}$. The polynomial c consists of $n = 256$ coefficients, τ of which are ± 1 ($\tau \in \{39, 49, 60\}$), the rest are zero. So the range of possible values for a single coefficient in $c\mathbf{s_1}$ is $[-\tau\eta, \tau\eta]$, and for $c\Delta\mathbf{s_1}$ it is $[-2\tau\eta, 2\tau\eta]$.

Idea behind the fault detection. Our countermeasure is based on the observation that a fault can be detected by analyzing the differences between the coefficients of the polynomials of $\mathbf{z}$, which we already hinted to in Subsection 3.1. Indeed, if $\mathbf{y}_{[i,j]} = \mathbf{y}_{[i',j]}$ then $\Delta\mathbf{z} = c\Delta\mathbf{s_1}$ and we know from the distribution of the challenge polynomial and the secret vector that the coefficients of $c\Delta\mathbf{s_1}$ are $\in [-2\tau\eta, 2\tau\eta]$. The countermeasure consists of analyzing the values of $\Delta\mathbf{z}$'s and counting how many of them are small (i.e., inside the range $[-2\tau\eta, 2\tau\eta]$) and how many are large (i.e., outside this range). On the one hand, if a $\Delta\mathbf{z}$ is outside this range, we can say with confidence that the $\Delta\mathbf{y}$-term is present and no fault was injected (at least for the coefficients in question). On the other hand, since the range of $\Delta\mathbf{y}$ is much larger than that of $c\Delta\mathbf{s_1}$, the probability of an unfaulted $\Delta\mathbf{z}$ lying inside of $[-2\tau\eta, 2\tau\eta]$ by chance is quite small. That is the key observation we use as a criterion for fault detection. One of the main advantages of this method, is that since the check is performed on $\mathbf{z}$, as opposed to checking $\mathbf{y}$, it can be done only once after the final signing attempt.

Norm-based fault detection algorithm. The inputs to the fault detection countermeasure are the vector $\mathbf{z}$ and what we refer to as the strictness parameter N. The strictness parameter N will correspond to the maximum number of small $\Delta\mathbf{z}$'s that we permit for a given signature. Concretely, if there are more than N small $\Delta\mathbf{z}$ values then we assume that a fault was injected, discard the signature and compute a new one. Interestingly, the parameter N additionally dictates what kind of attacks are prevented. By setting $N = 256$ we can prevent attacks which force two full polynomials of $\mathbf{y}$ to be equal. A smaller value of N allows detecting other versions of the attack, e.g., when only a few coefficients of two polynomials of $\mathbf{y}$ are forced to be equal. The smaller N is, the more kind of attacks are detected, but a low value of N also leads to a high False Positive Rate (FPR) since random coefficients can be close to each other and therefore lead to a small $\Delta\mathbf{z}$ value simply by chance. The fault detection is presented in Algorithm 2. In Line 1 we initialize a counter that will count the number of small $\Delta\mathbf{z}$. The for-loops in Lines 2 and 3 iterate over all possible combinations of polynomials in $\mathbf{z}$, the loop in Line 4 iterates over the coefficients. Lines 5 and 6 check whether a $\Delta\mathbf{z}$ is small, and in that case increment the counter. We then check if the maximum number of permitted small $\Delta\mathbf{z}$ is surpassed (Line 7). The output is 1 if we suspect a fault, and 0 if none was detected.

Analysis of the countermeasure. In the following we analyze Algorithm 2 and the probabilities involved, and in particular how the FPR depends on the strictness N. For this we need the probability distributions of $\Delta\mathbf{y}$ and $c\Delta\mathbf{s_1}$. Based on the Dilithium

Algorithm 2 Fault Detection Countermeasure $(\mathbf{z}, N)$

```
counter = 0
for i = 0, ..., ℓ − 2 do
    for i' = i + 1, ..., ℓ − 1 do
        for j = 0, ..., n − 1 do
            if z_[i,j] − z_[i',j] ∈ [−2τη, 2τη] then
                counter += 1
            if counter > N then
                return 1
return 0
```

specifications, we derived the probability mass functions of the former as:

$$P(\Delta\mathbf{y} = x) = \begin{cases} \frac{2\gamma_1 - |x|}{(2\gamma_1)^2}, & x \in (-2\gamma_1, 2\gamma_1), \\ 0, & \text{else}, \end{cases} \tag{7}$$

while the latter can be approximated by a normal distribution $\mathcal{N}(0, \sqrt{\frac{2}{3}\tau\eta(\eta+1)})$ following the Central Limit Theorem. For an unfaulted $\Delta\mathbf{z}$ the probability of lying inside the range $[-2\tau\eta, 2\tau\eta]$ can now be calculated in the following way:

$$P(\Delta\mathbf{z} \in [-2\tau\eta, 2\tau\eta]) = \sum_{x=-\infty}^{\infty} P(\Delta\mathbf{y} = x) \cdot P(c\Delta\mathbf{s_1} \in [-2\tau\eta - x, 2\tau\eta - x]).$$

We will call this probability p. This sum is not infinite, because $P(\Delta\mathbf{y} = x)$ is only non-zero for a finite number of x (see Equation 7). For a given $\mathbf{z}$ we can derive $\ell(\ell-1)/2$ differences of polynomials and because each polynomial has n coefficients, we then get $n \cdot \ell(\ell-1)/2 := n_z$ differences of coefficients $\Delta\mathbf{z}$. We approximate the FPR using a binomial distribution $\mathcal{B}(n_z, p)$[10]. The FPR is the probability that more than N $\Delta\mathbf{z}$'s are small.

$$\text{FPR}(N) = P(\#\Delta\mathbf{z} > N) = 1 - P(\#\Delta\mathbf{z} \leq N) = 1 - \sum_{k=0}^{N} \binom{n_z}{k} p^k (1-p)^{n_z - k}.$$

In an implementation of Dilithium it is important to know how many unfaulted signatures, i.e., ones that passed all Dilithium rejection checks, will have to be computed until one is accepted by the fault detection in Algorithm 2. We want this number to be as close to 1 as possible to avoid rejecting valid signatures due to the false positives in the detection. The probability that $m-1$ signatures are rejected and then the m-th one is accepted is $(1 - \text{FPR})^{m-1} \cdot \text{FPR}$. The expected number of signatures until we accept is then:

$$E_{sig}(N) = \sum_{m=1}^{\infty} m \cdot (1 - \text{FPR}(N))^{m-1} \cdot \text{FPR}(N) = \frac{1}{1 - \text{FPR}(N)}.$$

The above number is not the number of overall signatures created, but rather the number of valid signatures that already passed all rejection requirements of Dilithium itself, and are then passed on to Algorithm 2. In Table 7 we summarize all relevant values for the different parameter sets of Dilithium and provide the FPR and E_{sig} for exemplary values of N. For the strictest fault detection ($N = 0$), the FPR as well as the number of generated signatures E_{sig} are quite high for all three NIST security levels. However, if we increase N

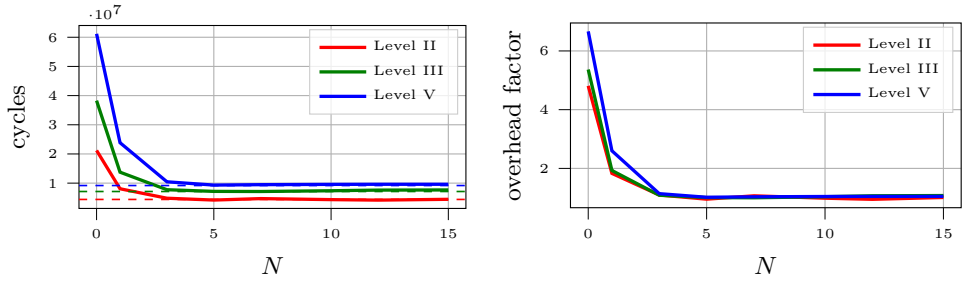
[10] Since the $\Delta\mathbf{z}$ are not independent random variables, we have verified that the approximations given in Table 7 match values derived experimentally for random signatures.

only slightly, which is equivalent to allowing some minimal equality in the polynomials of $\mathbf{y}$ which could be the case simply by chance, both the FPR and E_{sig} drop relatively fast until E_{sig} reaches ≈ 1 for $N = 15$. This value of N still provides sufficient security for all Dilithium parameter sets since it would detect when full polynomials or a significant portion of the polynomials of $\mathbf{y}$ are equal but with a close to zero FPR and negligible impact on the rejection rate.

Table 7: Parameters and FPR of the norm-based fault detection approach for different NIST security levels and strictness parameter N.

NIST security level	II	III	V
n_z	1536	2560	5376
$P(\Delta\mathbf{z} \in [-2\tau\eta, 2\tau\eta])$	0.001190	0.000748	0.000458
FPR$(N = 0)$	0.839	0.853	0.915
$E_{sig}(N = 0)$	6.23	6.79	11.74
FPR$(N = 5)$	0.011	0.014	0.040
$E_{sig}(N = 5)$	1.01	1.01	1.04
FPR$(N = 15)$	1.26×10^{-10}	2.59×10^{-10}	8.54×10^{-9}
$E_{sig}(N = 15)$	~1	~1	~1

Performance evaluation. Figure 4 shows the overhead introduced by the countermeasure as function of the strictness parameter N. The numbers are obtained from the PQM4 reference implementation and the added countermeasure running on the `NUCLEO-L4R5ZI` board. The left side provides the overhead in the number of clock cycles, and the right side the overhead relative to the unprotected signature generation. The dashed lines on the left side correspond to the clock cycles spent on the unprotected signature generation. These results confirm the estimations provided in Table 7[11]. For a very small N (e.g., $N = 0$), the overhead is quite significant since many signing iterations are needed to generate a valid signature that also fulfills the condition set by the countermeasure, but as soon as N is large enough, for instance $N \geq 5$ no additional signing iterations are needed most of the time. As a result, the proposed countermeasure can be parameterized to achieve a negligible overhead.



(a) Performance (b) Relative overhead

Figure 4: Comparison of average runtime of Dilithium signatures with and without the countermeasure as function of the strictness parameter N.

[11]Note that the relative overheads on the right side of the figure when $N = 0$ are not equal to the estimated E_{sig}. This is expected since the public matrix $\mathbf{A}$ is only computed once at the beginning of the signature generation and not for every signing iteration.

7 Conclusion

In this paper, we introduced a new fault attack which applies to both randomized and deterministic versions of Dilithium. In particular, the attack requires a few instruction skips to reduce the MLWE problem Dilithium is based on to a much simpler RLWE problem. The latter can be solved with lattice reduction attacks which we demonstrate and enhance with the integration of side-information or hints extracted from the faulty signatures. As a final contribution we also suggested countermeasures to protect against the presented attack with minimal overheads.

As for perspectives, it is clear that PQC schemes are relatively less investigated than standard asymmetric cryptography based on RSA or ECC that has been practically deployed for decades. This holds true as well with respect to implementation attacks, including side-channel and fault attacks. From our work, we confirm the interest of the first steps taken in [DDGR20] to bind the gap between lattice reduction attacks and fault/side-channel attacks. We believe that using partial (i.e., probabilistic) side-information, as usually obtained from noisy measurements, would be an important step forward into that direction.

A Impact of faults on $\mathbf{s_2}$

It was noted in [ABC+22] that although $\tilde{\mathbf{r}} = \mathbf{w_0} - c\mathbf{s_2} = \mathbf{w} - \alpha\mathbf{w_1} - c\mathbf{s_2}$ is not released as part of the signature it can be computed from the signature and the public key as $\mathbf{Az} - c\mathbf{t} - \alpha\mathbf{w_1}$. In the context of our attack, we first examine how the faulty $\mathbf{y}$ affects $\mathbf{w} = \mathbf{A} \cdot \mathbf{y}$. This is shown in Equation 8 where we denote $\forall i \in \{0, 1, \ldots, k-1\}, \sum_{j=0}^{\ell-1} \mathbf{A}_{[i,j]} = \mathbf{a}_{[i]}$.

$$\begin{bmatrix} \mathbf{w}_{[0]} \\ \mathbf{w}_{[1]} \\ \vdots \\ \mathbf{w}_{[k-1]} \end{bmatrix} = \begin{bmatrix} \mathbf{A}_{[0,0]} & \cdots & \mathbf{A}_{[0,\ell-1]} \\ \mathbf{A}_{[1,0]} & \cdots & \mathbf{A}_{[1,\ell-1]} \\ \vdots & \ddots & \vdots \\ \mathbf{A}_{[k-1,0]} & \cdots & \mathbf{A}_{[k-1,\ell-1]} \end{bmatrix} \cdot \begin{bmatrix} y \\ y \\ \vdots \\ y \end{bmatrix} = y \cdot \begin{bmatrix} \mathbf{a}_{[0]} \\ \mathbf{a}_{[1]} \\ \vdots \\ \mathbf{a}_{[k-1]} \end{bmatrix} \tag{8}$$

Next we compute weighted pairwise differences of the polynomials of $\tilde{\mathbf{r}}$ as follows:

$$\begin{aligned} \mathbf{a}_{[1]}.\tilde{\mathbf{r}}_{[0]} - \mathbf{a}_{[0]}.\tilde{\mathbf{r}}_{[1]} &= \mathbf{a}_{[0]}(\mathbf{w}_{[0]} - \alpha\mathbf{w_1}_{[0]} - c\mathbf{s_2}_{[0]}) - \mathbf{a}_{[1]}(\mathbf{w}_{[1]} - \alpha\mathbf{w_1}_{[1]} - c\mathbf{s_2}_{[1]}) \\ &= \mathbf{a}_{[1]}(\mathbf{a}_{[0]}.y - \alpha\mathbf{w_1}_{[0]} - c\mathbf{s_2}_{[0]}) - \mathbf{a}_{[0]}(\mathbf{a}_{[1]}.y - \alpha\mathbf{w_1}_{[1]} - c\mathbf{s_2}_{[1]}) \\ &= \mathbf{a}_{[0]}(\alpha\mathbf{w_1}_{[1]} + c\mathbf{s_2}_{[1]}) - \mathbf{a}_{[1]}(\alpha\mathbf{w_1}_{[0]} + c\mathbf{s_2}_{[0]}) \end{aligned}$$

And as a result we can write $\mathbf{s_2}_{[1]}$ as function of $\mathbf{s_2}_{[0]}$.

$$\begin{aligned} \mathbf{a}_{[0]}(\alpha\mathbf{w_1}_{[1]} + c\mathbf{s_2}_{[1]}) &= \mathbf{a}_{[1]}.\tilde{\mathbf{r}}_{[0]} - \mathbf{a}_{[0]}.\tilde{\mathbf{r}}_{[1]} + \mathbf{a}_{[1]}(\alpha\mathbf{w_1}_{[0]} + c\mathbf{s_2}_{[0]}) \\ \alpha\mathbf{w_1}_{[1]} + c\mathbf{s_2}_{[1]}) &= \mathbf{a}_{[0]}^{-1}\mathbf{a}_{[1]}(\tilde{\mathbf{r}}_{[0]} + \alpha\mathbf{w_1}_{[0]} + c\mathbf{s_2}_{[0]}) - \tilde{\mathbf{r}}_{[1]} \\ \mathbf{s_2}_{[1]} &= c^{-1}(\mathbf{a}_{[0]}^{-1}\mathbf{a}_{[1]}(\tilde{\mathbf{r}}_{[0]} + \alpha\mathbf{w_1}_{[0]} + c\mathbf{s_2}_{[0]}) - \tilde{\mathbf{r}}_{[1]} - \alpha\mathbf{w_1}_{[1]}) \end{aligned} \tag{9}$$

Eventually, what Equation 9 highlights is that, similarly to $\mathbf{s_1}$, after the attack it is possible to recover all polynomials of $\mathbf{s_2}$ from a single one.

B Comparing LWE estimates for RLWE instance

Table 8: Comparison of LWE estimators pre- and post-attack assuming that $\mathbf{t}$ is public.

NIST Security Level	II	III	V
LWE Hardness of Dilithium (standard)			
$(n \cdot k, n \cdot \ell)$ [LWE dimension]	(1024, 1024)	(1536, 1280)	(2048, 1792)
[DKL+21][6]	433	638	883
"GSA" [APS15]	424	625	883
"GSA-Intersect" [DDGR20][7]	424	627	870
"Probabilistic-Simulation" [DDGR20][7]	434	641	890
LWE Hardness of Dilithium (post-attack)			
(n, n) [LWE dimension]	(256, 256)	(256, 256)	(256, 256)
[DKL+21][6]	50	50	50
"GSA" [APS15]	40	48	40
"GSA-Intersect" [DDGR20][7]	29	46	29
"Probabilistic-Simulation" [DDGR20][7]	62	68	62

References

[ABC+22] Melissa Azouaoui, Olivier Bronchain, Gaëtan Cassiers, Clément Hoffmann, Yulia Kuzovkova, Joost Renes, Markus Schönauer, Tobias Schneider, François-Xavier Standaert, and Christine van Vredendaal. Leveling dilithium against leakage: Revisited sensitivity analysis and improved implementations. *IACR Cryptol. ePrint Arch.*, page 1406, 2022.

[ADPS16] Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. Post-quantum key exchange - A new hope. In Thorsten Holz and Stefan Savage, editors, *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016*, pages 327–343. USENIX Association, 2016.

[AGVW17] Martin R. Albrecht, Florian Göpfert, Fernando Virdia, and Thomas Wunderer. Revisiting the expected cost of solving usvp and applications to LWE. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I*, volume 10624 of *Lecture Notes in Computer Science*, pages 297–322. Springer, 2017.

[Ajt99] Miklós Ajtai. Generating hard instances of the short basis problem. In Jirí Wiedermann, Peter van Emde Boas, and Mogens Nielsen, editors, *Automata, Languages and Programming, 26th International Colloquium, ICALP'99, Prague, Czech Republic, July 11-15, 1999, Proceedings*, volume 1644 of *Lecture Notes in Computer Science*, pages 1–9. Springer, 1999.

[AKS01] Miklós Ajtai, Ravi Kumar, and D. Sivakumar. A sieve algorithm for the shortest lattice vector problem. In Jeffrey Scott Vitter, Paul G. Spirakis, and Mihalis Yannakakis, editors, *Proceedings on 33rd Annual ACM Symposium on Theory of Computing, July 6-8, 2001, Heraklion, Crete, Greece*, pages 601–610. ACM, 2001.

[APS15] Martin R. Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. *J. Math. Cryptol.*, 9(3):169–203, 2015.

[BBK16] Nina Bindel, Johannes Buchmann, and Juliane Krämer. Lattice-based signature schemes and their sensitivity to fault attacks. In *2016 Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2016, Santa Barbara, CA, USA, August 16, 2016*, pages 63–77. IEEE Computer Society, 2016.

[BFP19] Claudio Bozzato, Riccardo Focardi, and Francesco Palmarini. Shaping the glitch: Optimizing voltage fault injection attacks. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(2):199–224, 2019.

[BG14] Shi Bai and Steven D. Galbraith. Lattice decoding attacks on binary LWE. In Willy Susilo and Yi Mu, editors, *Information Security and Privacy - 19th Australasian Conference, ACISP 2014, Wollongong, NSW, Australia, July 7-9, 2014. Proceedings*, volume 8544 of *Lecture Notes in Computer Science*, pages 322–337. Springer, 2014.

[BMR21] Luk Bettale, Simon Montoya, and Guénaël Renault. Safe-error analysis of post-quantum cryptography mechanisms - short paper-. In *18th Workshop on Fault Detection and Tolerance in Cryptography, FDTC 2021, Milan, Italy, September 17, 2021*, pages 39–44. IEEE, 2021.

[BP18] Leon Groot Bruinderink and Peter Pessl. Differential fault attacks on deterministic lattice signatures. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(3):21–43, 2018.

[BRS22] Joppe W. Bos, Joost Renes, and Amber Sprenkels. Dilithium for memory constrained devices. In Lejla Batina and Joan Daemen, editors, *Progress in Cryptology - AFRICACRYPT 2022: 13th International Conference on Cryptology in Africa, AFRICACRYPT 2022, Fes, Morocco, July 18-20, 2022, Proceedings*, Lecture Notes in Computer Science, pages 217–235. Springer Nature Switzerland, 2022.

[CN11] Yuanmi Chen and Phong Q. Nguyen. BKZ 2.0: Better lattice security estimates. In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology - ASIACRYPT 2011 - 17th International Conference on the Theory and Application of Cryptology and Information Security, Seoul, South Korea, December 4-8, 2011. Proceedings*, volume 7073 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2011.

[DDGR20] Dana Dachman-Soled, Léo Ducas, Huijing Gong, and Mélissa Rossi. LWE with side information: Attacks and concrete security estimation. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology - CRYPTO 2020 - 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17-21, 2020, Proceedings, Part II*, volume 12171 of *Lecture Notes in Computer Science*, pages 329–358. Springer, 2020.

[DKL+21] Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. Crystals-dilithium algorithm specifications and supporting documentation (version 3.1). 2021.

[DRPR19] Jean-Max Dutertre, Timothée Riom, Olivier Potin, and Jean-Baptiste Rigaud. Experimental analysis of the laser-induced instruction skip fault model. In Aslan Askarov, René Rydhof Hansen, and Willard Rafnsson, editors, *Secure IT Systems - 24th Nordic Conference, NordSec 2019, Aalborg, Denmark, November 18-20, 2019, Proceedings*, volume 11875 of *Lecture Notes in Computer Science*, pages 221–237. Springer, 2019.

[dt21] The FPLLL development team. fpylll, a Python wraper for the fplll lattice reduction library, Version: 0.5.7. Available at https://github.com/fplll/fpylll, 2021.

[EFGT16] Thomas Espitau, Pierre-Alain Fouque, Benoît Gérard, and Mehdi Tibouchi. Loop-abort faults on lattice-based fiat-shamir and hash-and-sign signatures. In Roberto Avanzi and Howard M. Heys, editors, *Selected Areas in Cryptography - SAC 2016 - 23rd International Conference, St. John's, NL, Canada, August 10-12, 2016, Revised Selected Papers*, volume 10532 of *Lecture Notes in Computer Science*, pages 140–158. Springer, 2016.

[GKS21] Denisa O. C. Greconici, Matthias J. Kannwischer, and Amber Sprenkels. Compact dilithium implementations on cortex-m3 and cortex-m4. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(1):1–24, 2021.

[IMS+22] Saad Islam, Koksal Mus, Richa Singh, Patrick Schaumont, and Berk Sunar. Signature correction attack on dilithium signature scheme. In *7th IEEE European Symposium on Security and Privacy, EuroS&P 2022, Genoa, Italy, June 6-10, 2022*, pages 647–663. IEEE, 2022.

[Kan87] Ravi Kannan. Minkowski's convex body theorem and integer programming. *Math. Oper. Res.*, 12(3):415–440, 1987.

[Kor77] Korkine. Sur les formes quadratiques positives. (zus. mit s. zolotareff). *Mathematische Annalen*, 11:242–292, 1877.

[LLL82] Arjen K. Lenstra, Hendrik W. Lenstra, and László Miklós Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261:515–534, 1982.

[LPR10] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In Henri Gilbert, editor, *Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Monaco / French Riviera, May 30 - June 3, 2010. Proceedings*, volume 6110 of *Lecture Notes in Computer Science*, pages 1–23. Springer, 2010.

[Lyu22] Vadim Lyubashevsky. Crystals-dilithium update. Fourth PQC Standardization Conference, 2022.

[LZS+21] Yuejun Liu, Yongbin Zhou, Shuo Sun, Tianyu Wang, Rui Zhang, and Jingdian Ming. On the security of lattice-based fiat-shamir signatures in the presence of randomness leakage. *IEEE Trans. Inf. Forensics Secur.*, 16:1868–1879, 2021.

[MDP+20] Alexandre Menu, Jean-Max Dutertre, Olivier Potin, Jean-Baptiste Rigaud, and Jean-Luc Danger. Experimental analysis of the electromagnetic instruction skip fault model. In *15th Design & Technology of Integrated Systems in Nanoscale Era, DTIS 2020, Marrakech, Morocco, April 1-3, 2020*, pages 1–7. IEEE, 2020.

[Nat] National Institute of Standards and Technology. Post-quantum cryptography standardization. https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Post-Quantum-Cryptography-Standardization.

[PV06] Dan Page and Frederik Vercauteren. A fault attack on pairing-based cryptography. *IEEE Trans. Computers*, 55(9):1075–1080, 2006.

[RCDB22] Prasanna Ravi, Anupam Chattopadhyay, Jan Pieter D'Anvers, and Anubhab Baksi. Side-channel and fault-injection attacks over lattice-based post-quantum schemes (kyber, dilithium): Survey and new results. Cryptology ePrint Archive, Paper 2022/737, 2022. https://eprint.iacr.org/2022/737.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, May 22-24, 2005*, pages 84–93. ACM, 2005.

[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. *IACR Cryptol. ePrint Arch.*, page 821, 2018.

[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, *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security, AsiaCCS 2019, Auckland, New Zealand, July 09-12, 2019*, pages 427–440. ACM, 2019.

[RRB+19] Prasanna Ravi, Debapriya Basu Roy, Shivam Bhasin, Anupam Chattopadhyay, and Debdeep Mukhopadhyay. Number "not used" once - practical fault attack on pqm4 implementations of NIST candidates. In Ilia Polian and Marc Stöttinger, editors, *Constructive Side-Channel Analysis and Secure Design - 10th International Workshop, COSADE 2019, Darmstadt, Germany, April 3-5, 2019, Proceedings*, volume 11421 of *Lecture Notes in Computer Science*, pages 232–250. Springer, 2019.

[Sch87] Claus-Peter Schnorr. A hierarchy of polynomial time lattice basis reduction algorithms. *Theor. Comput. Sci.*, 53:201–224, 1987.

[Sho21] Victor Shoup. NTL (Number Theory Library) - a library for doing number theory, Version: 11.5.1. Available at https://libntl.org, 2021.