Finding a polytope: A practical fault attack against Dilithium

Paco Azevedo-Oliveira[1,2], Andersson Calle Viera[1,3], Benoît Cogliati[1], and Louis Goubin[2]

[1] Thales DIS, France
paco.azevedo-oliveira@thalesgroup.com
andersson.calle-viera@thalesgroup.com
benoit-michel.cogliati@thalesgroup.com
[2] Laboratoire de Mathématiques de Versailles, UVSQ, CNRS, Université Paris-Saclay, 78035 Versailles, France
louis.goubin@uvsq.fr
[3] Sorbonne Université, CNRS, Inria, LIP6, F-75005 Paris, France

Abstract. In Dilithium, the rejection sampling step is crucial for the proof of security and correctness of the scheme. However, to our knowledge, there is no attack in the literature that takes advantage of an attacker knowing rejected signatures. The aim of this paper is to create a practical black-box attack against Dilithium with a weakened rejection sampling. We succeed in showing that an adversary with enough rejected signatures can recover Dilithium's secret key in less than half an hour on a desktop computer. There is one possible application for this result: by physically preventing one of the rejection sampling tests from happening, we obtain two fault attacks against Dilithium.

1 Introduction

In July 2022, the National Institute of Standards and Technology (NIST) selected CRYSTALS-Dilithium, also known as Dilithium, as a new post-quantum digital signature scheme. It is being standardized under the name ML-DSA [NIS23], and the National Security Agency (NSA) has included it in the Commercial National Security Algorithm (CNSA 2.0) suite for national security systems [NSA22]. Moreover, due to its relative efficiency compared to other post-quantum schemes, Dilithium is recommended for computing quantum-secure signatures in most use cases.

From a theoretical point of view, Dilithium benefits from security proofs supporting its Strong existential Unforgeability under Chosen Message Attack (SUF-CMA) in the classical and quantum random oracle models [BDK+21]. To complement this cryptanalytic approach, it is necessary to investigate the security of embedded implementations. The security of Dilithium against Side-Channel Attacks (SCA) and Fault Attacks (FA) thus needs to be carefully assessed.

Following this direction, many papers have already been published about physical attacks [BBK16],[BVC+23] [BP18] [CKA+21] [KLH+20] [MUTS22] [RJH+18] [EAB+23] [BAE+24] [WNGD23] [KPLG24] against Dilithium, see also this survey [RCDB22].

In the present paper, we use Linear Programming (LP) [NW88] in several attack scenarios. This kind of technique has already been used in [MUTS22] [UMB+23] to mount a fault attack and recover the $\mathbf{s}_1$ value of Dilithium from noisy values, and hence the whole secret key. Note that, in the (very) rare papers using linear programming, this technique has, up to now, been seen as a complement to a computationally heavy profiling phase.

Our LP-based technique is first applied to fault attacks against Dilithium. Up to now, all the developed strategies have consisted of faulting either the NTT-based computations, the nonce $\mathbf{y}$ used to sign, or additions/multiplications with secret polynomials. However, none of the previous attacks have targeted the tests used in the rejection sampling mechanism. This is the core of the fault attack we elaborate in the present paper.

For each attack described here, we consider two versions of the Dilithium signature algorithm. The first is the official specification published in the FIPS (draft) standard [NIS23]. The second one corresponds to the alternative way (described in Section 5.1 of [BDK+21]) to perform the validity checks on $\mathbf{r_0}$ and to compute $\mathbf{h}$, which corresponds to the reference implementation [DKL+].

It is essential to analyze the impact of the attacks on these two versions of Dilithium. Indeed, even if they are functionally equivalent, this is no longer true when we consider perturbations of the tests involved in the rejection sampling mechanism or when we have access to internal values in the context of multi-party computation.

In all the attacks we describe in the present paper, the parameter $\mathbf{t}_0$ of Dilithium plays a particular role that has been debated in the literature. In the draft of the FIPS ML-DSA standard [NIS23], $\mathbf{t}_0$ is officially considered as part of the secret key. However, as indicated in the same document: "The vector $\mathbf{t}$ is compressed in the actual public key by dropping the d least significant bits from each coefficient, thus producing the polynomial vector $\mathbf{t}_1$. This compression is an optimization for performance, not security. The low order bits of t can be reconstructed from a small number of signatures and, therefore, need not be regarded as secret." Note that the EUF-CMA security proof provided by the authors of Dilithium [BDK+21] makes the same assumption and considers that $\mathbf{t}$ is public. In the same spirit, Schwabe writes [Sch19]: "$\mathbf{t}_0$ is not part of the secret key, but actually a public value (taken into account in the security analysis). The reason not to make it part of the public key is that it's not needed for verification so we can have smaller public keys." Furthermore, a recent eprint paper [AOCVCG24], would seem to show that $\mathbf{t}_0$ can be reconstructed with 4 000 000 Dilithium signatures signed under the same secret key. Therefore, in

the rest of the paper, we will assume –as in most of the literature about the security of Dilithium– that $\mathbf{t}_0$ is a public parameter.

Our Contributions. In this paper, we describe several kinds of attacks that take advantage of potential weaknesses in the implementations of the tests involved in the rejection sampling mechanism. Our attacks require from 1 million to 4 millions signatures, depending on the security level of Dilithium that is targeted.

The first kind of attack depends on fault injections that allow the attacker to skip the *second* test (on $\mathbf{r}_0$) involved in the rejection sampling mechanism. Two scenarios are considered:

- If the target is the official specification of Dilithium [LDK+22], knowing that the faulty signatures are not valid (they are not accepted by the verification algorithm) allows to recover the secret value $\mathbf{s}_2$. It is well known that $\mathbf{s}_1$ can then be deduced, assuming the attacker knows the $\mathbf{t}_0$ parameter.
- If the target is the official reference implementation of Dilithium [DKL+22], based on an alternative way of performing the validity checks on $\mathbf{r_0}$ and computing $\mathbf{h}$, this time the obtained (faulty) signatures *do* pass the verification phase, as well as normal (non-faulty) signatures. However, the knowledge of $\mathbf{t}_0$ enables to detect the faulty signatures, leading to a recovery of $\mathbf{s}_2$, then $\mathbf{s}_1$.

In both cases, the attack uses Linear Programming (LP) tools, and experiments show that it is very efficient on usual Dilithium parameters (typically less than 30 minutes).

The second kind of attack is also based on fault injections that are used to skip the *first* test (on $\mathbf{z}$) in the rejection sampling mechanism. We describe a method that is similar to the previous attacks and also makes use of (LP) tools. This attack (which remains the same for all versions of Dilithium) allows to recover $\mathbf{s}_1$ (and thus enables to forge arbitrary many signatures). It is very efficient and does not even require the knowledge of the $\mathbf{t}_0$ parameter.

All these attacks illustrate the power of LP-based methods to recover secret information from faulty/rejected signatures. This can be applied in several attack models (fault attacks, side-channels, white-box, multi-party computation) and bring new arguments supporting the need for protecting not only the values manipulated during the Dilithium signature computation but also the very execution of the tests during the rejection sampling phase.

Outline. This paper is organized as follows. In Section 2, we review the necessary knowledge on Dilithium and linear programming. In Section 3, we define an attack scenario and an associated problem, then show how this problem can be reformulated in terms of integer optimization. In Section 4, we discuss the difference between the algorithm specification, proposed as a standard implementation, and the reference implementation, which does not follow this specification. We explain how to adapt the attack on the reference implementation.

In Section 5, we propose a method for solving the problem defined in Section 3. In Section 6, we present our practical results obtained by attacking Dilithium-2, Dilithium-3, and Dilithium-5. Finally, in Section 7 we discuss the results obtained, their limitations and their implications in our opinion.

2 Background and notations

This section recalls the definitions and results already known, which will be useful throughout the rest of the paper. To pose the problem, we recall the notations used in Dilithium and briefly explain how the algorithm works. To reformulate the problem, we give the basic definitions of polytope theory, and finally, to solve the problem, we provide the main linear programming results.

2.1 Lattices

Definition 1 ***(Modular reductions)***
Let α an even integer (resp. odd), we define $r' := r \mod^{\pm}(\alpha)$ the unique $-\frac{\alpha}{2} < r' \leq \frac{\alpha}{2}$ (resp. $-\frac{\alpha-1}{2} \leq r' \leq \frac{\alpha-1}{2}$) such that $r' = r \mod (\alpha)$. We will speak of centered reduction modulo α. We define $r'' := r \mod^{+}(\alpha)$ the unique $0 \leq r'' < \alpha$ such that $r'' = r \mod (\alpha)$.

Definition 2 ***(Cyclotomic ring)***
We define $\phi_n = x^n + 1$ with n a power of 2. This is a cyclotomic polynomial (One can show that ϕ_n is the $2n$-th cyclotomic polynomial.) In particular, it is irreducible over $\mathbb{Q}$.

For q a prime, we define:

$$\mathcal{Q} := \mathbb{Q}[x]/(\phi_n) \ , \ \mathcal{R} := \mathbb{Z}[x]/(\phi_n) \text{ and } \mathcal{R}_q := \mathbb{Z}_q[x]/(\phi_n).$$

Definition 3 *For $w \in \mathbb{Z}_q$:*

$$||w||_\infty := |w \mod^{\pm} (q)|.$$

For $\boldsymbol{w} = \sum w_i x^i \in \mathcal{R}$:

$$||\boldsymbol{w}||_\infty := \max ||w_i \mod^{\pm}(q)||_\infty \text{ and } ||\boldsymbol{w}|| := \Big(\sum ||w_i||_\infty^2 \Big)^{1/2}$$

and for $\mathbf{w} = (\mathbf{w}^{[1]}, ..., \mathbf{w}^{[k]}) \in \mathcal{R}^k$,

$$||\mathbf{w}||_\infty := \max ||\mathbf{w}^{[i]}||_\infty \text{ and } ||\mathbf{w}|| := \Big(\sum ||\mathbf{w}^{[i]}||^2 \Big)^{1/2}.$$

Finally, we define two sets $S_\eta, \tilde{S}_\eta \subset \mathcal{R}$:

$$S_\eta := \{\boldsymbol{w} \in \mathcal{R} \mid ||\boldsymbol{w}||_\infty \leq \eta\} \quad \text{and} \quad \tilde{S}_\eta := \{\boldsymbol{w} \mod^{\pm} (2\eta) \mid \boldsymbol{w} \in \mathcal{R}\}.$$

Notation 1 *For an element $\mathbf{w}_1 \in \mathcal{R}^l$ we will note $\mathbf{w}_1 = (\mathbf{w}_1^{[1]}, ..., \mathbf{w}_1^{[l]}) \in \mathcal{R}^l$ and $\mathbf{w}_{1,i}^{[j]}$ will be the $i-th$ coefficient of the polynomial $\mathbf{w}_1^{[j]}$.*

Notation 2 *We will note $[\![$ statement $]\!]$ the boolean operator wich evaluates to 1 if statement is true, and to 0 otherwise.*

2.2 Dilithium, hints and inequalities

Dilithium uses $\mathcal{R}_q^{k\times l}$ with k, l varying according to the level of security required and with n and q chosen as:

$$n = 256, \quad q = 2^{23} - 2^{13} + 1 = 8\,380\,417$$

Dilithium uses algorithms that split elements in $\mathbb{Z}_q$. Informally speaking, for an even α divisor of $q-1$, $r \in \mathbb{Z}_q$ one can define $r = r_1\alpha + r_0$ with $r_0 = r \bmod^{\pm}(\alpha)$ and $r_1 = (r - r_0)/\alpha$. We will call r_1 the most significant bits of r and r_0 the least significant bits of r. As shown in Figure 1, for $z \in \mathbb{Z}_q$ such that $|z| \leq \alpha/2$, adding z to r can increase or decrease the most significant bits of r by ± 1. The aim is to be able to calculate the most significant bits of an addition between $r \in \mathbb{Z}_q$ and a small element $z \in \mathbb{Z}_q$, without having to store z. To do this, an algorithm which generates a one bit hint h is used. In Algorithm 1 we give the description of the algorithms and recall in Lemma 1 the main property used.

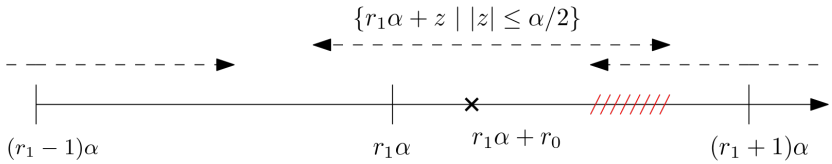

Fig. 1. carry caused by x

Algorithm 1 `Supporting algorithms for Dilithium`

`Decompose`$_q(r, \alpha)$:

1: $r = r \bmod^{+} q$
2: $r_0 = r \bmod^{\pm} \alpha$
3: **if** $r - r_0 = q - 1$ **then** $r_1 = 0$ $r_0 = r_0 - 1$
4: **else** $r_1 = (r - r_0)/\alpha$
5: **return** (r_1, r_0)

`HighBits`$_q(r, \alpha)$:

1: $(r_1, r_0) =$ `Decompose`$_q(r, \alpha)$
2: **return** r_1

`LowBits`$_q(r, \alpha)$:

1: $(r_1, r_0) =$ `Decompose`$_q(r, \alpha)$
2: **return** r_0

`MakeHint`$_q(z, r, \alpha)$:

1: $r_1 =$ `HighBits`$_q(r, \alpha)$
2: $v_1 =$ `HighBits`$_q(r + z, \alpha)$
3: **return** $[\![r_1 \neq v_1]\!]$

`UseHint`$_q(h, r, \alpha)$:

1: $m = (q - 1)/\alpha$
2: $(r_1, r_0) =$ `Decompose`$_q(r, \alpha)$
3: **if** $h = 1$ and $r_0 > 0$ **then return** $(r_1 + 1) \bmod^{+} m$
4: **if** $h = 1$ and $r_0 \leq 0$ **then return** $(r_1 - 1) \bmod^{+} m$
5: **return** r_1

Lemma 1 *[LDK+22] Let q and α be two positive integers such that $q > 2\alpha,\ q \equiv 1 \mod (\alpha)$ and α even. Let $\mathbf{r}$ and $\mathbf{z}$ be two vectors of $\mathcal{R}_q$ such that $||\mathbf{z}||_\infty \leq \alpha/2$ and let $\mathbf{h}, \mathbf{h}'$ be bit vectors. So the algorithms $\texttt{HighBits}_q$, $\texttt{MakeHint}_q$, $\texttt{UseHint}_q$ satisfy the properties:*

$$\texttt{UseHint}_q(\texttt{MakeHint}_q(\mathbf{z}, \mathbf{r}, \alpha), \mathbf{r}, \alpha) = \texttt{HighBits}_q(\mathbf{r} + \mathbf{z}, \alpha).$$

Remark 1 *Dilithium uses these algorithms to reduce the size of the public key: it splits a part of the public key $\mathbf{t}$ into two using an algorithm named* Power2Round defined in the specification of Dilithium [BDK+21] *and makes public only $\mathbf{t}_1$, the most significant bits of $\mathbf{t}$. In return, the signer adds a few hint bits to the signature to enable the signature to be verified without knowledge of the least significant bits of $\mathbf{t}$.*

2.3 Algorithm description

The remainder of this section gives a quick overview of Dilithium, for an exhaustive description of the functions used see [BDK+21].

Key Generation: The key generation consists of sampling two short vectors of polynomials $\mathbf{s}_1$ and $\mathbf{s}_2$ with a public matrix $\mathbf{A}$. We then calculate $\mathbf{t} = \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2$, which will also become public. To reduce the size of the public key, only the most significant bits of $\mathbf{t}$, $\mathbf{t}_1$, are part of the public key. For the same reason, we keep only the seed ρ used to generate $\mathbf{A}$. The vectors of small polynomials $\mathbf{s}_1$ and $\mathbf{s}_2$ remain secret. The key generation algorithm is described in Algorithm 2.

Algorithm 2 KeyGen

Ensure: (pk, sk)
1: $\zeta \leftarrow \{0,1\}^{256}$
2: $(\rho, \rho', K) \in \{0,1\}^{256} \times \{0,1\}^{512} \times \{0,1\}^{256} := \mathrm{H}(\zeta)$
3: $\mathbf{A} \in \mathcal{R}_q^{k \times l} := \texttt{ExpandA}(\rho)$ ▷ $\mathbf{A}$ is generated and stored in NTT as $\hat{\mathbf{A}}$
4: $(\mathbf{s}_1, \mathbf{s}_2) \in S_\eta^l \times S_\eta^k := \texttt{ExpandS}(\rho')$
5: $\mathbf{t} := \mathbf{A}\,\mathbf{s}_1 + \mathbf{s}_2$ ▷ Compute $\mathbf{A}\mathbf{s}_1$ as $\texttt{NTT}^{-1}(\hat{\mathbf{A}} \cdot \texttt{NTT}(\mathbf{s}_1))$
6: $(\mathbf{t}_1, \mathbf{t}_0) := \texttt{Power2Round}_q(\mathbf{t}, d)$
7: $tr \in \{0,1\}^{256} := \mathrm{H}(\rho \,||\, \mathbf{t}_1)$
8: **return** $pk = (\rho, \mathbf{t}_1)$, $sk = (\rho, K, tr, \mathbf{s}_1, \mathbf{s}_2, \mathbf{t}_0)$

Signature: Dilithium is based on the "Fiat-Shamir with aborts" framework: a signature is generated and accepted if it meets certain conditions; if it does not, the process is repeated until a valid signature is obtained. The signer draws a random polynomial vector $\mathbf{y} \in \tilde{S}_{\gamma_1}^l$. Then from $\texttt{HighBits}_q(\mathbf{A}\mathbf{y}, 2\,\gamma_2)$ using a

hash function, it creates a challenge c. Then, it calculates $\mathbf{z} := \mathbf{y} + c\,\mathbf{s}_1$ the definition of $\mathbf{z}$ and $\mathbf{t}$ gives:

$$\texttt{HighBits}_q(\mathbf{A}\mathbf{z} - c\mathbf{t},\, 2\,\gamma_2) = \texttt{HighBits}_q(\mathbf{A}\mathbf{y} - c\mathbf{s}_2,\, 2\,\gamma_2).$$

Furthermore, $\mathbf{y}$ is chosen such that:

$$\texttt{HighBits}_q(\mathbf{A}\mathbf{y} - c\mathbf{s}_2,\, 2\,\gamma_2) = \texttt{HighBits}_q(\mathbf{A}\mathbf{y},\, 2\,\gamma_2)$$

As $\mathbf{t}_0$ is not public, the signer adds a vector of hint $\mathbf{h} = \texttt{MakeHint}_q(-c\mathbf{t}_0, \mathbf{A}\mathbf{y} - c\mathbf{s}_2 + c\mathbf{t}_0, 2\gamma_2)$ to enable the verifier to calculate $\texttt{HighBits}_q(\mathbf{A}\mathbf{z} - c\mathbf{t},\, 2\,\gamma_2)$ and then recalculate c without knowledge of $\mathbf{t}_0$. The signature algorithm is described in Algorithm 3.

Algorithm 3 `Sig`

Require: sk, M
Ensure: $\sigma = (\tilde{c}, \mathbf{z}, \mathbf{h})$

$\mathbf{A} \in \mathcal{R}_q^{k\times l} := \texttt{ExpandA}(\rho)$ ▷ $\mathbf{A}$ is generated and stored in NTT as $\hat{\mathbf{A}}$
$\mu \in \{0,1\}^{512} := \mathrm{H}(tr \,||\, M)$
$\kappa := 0$, $(\mathbf{z}, \mathbf{h}) := \perp$
$\rho' \in \{0,1\}^{512} := \mathrm{H}(K \,||\, \mu)$
while $(\mathbf{z}, \mathbf{h}) = \perp$ **do** ▷ Pre-compute $\hat{s}_1 := \texttt{NTT}(\mathbf{s}_1)$, $\hat{s}_2 := \texttt{NTT}(\mathbf{s}_2)$ and $\hat{\mathbf{t}}_0 := \texttt{NTT}(\mathbf{t}_0)$
  $\mathbf{y} \in \tilde{S}_{\gamma_1}^l := \texttt{ExpandMask}(\rho', \kappa)$
  $\mathbf{w} := \mathbf{A}\,\mathbf{y}$ ▷ $\mathbf{w} := \texttt{NTT}^{-1}(\hat{\mathbf{A}} \cdot \texttt{NTT}(\mathbf{y}))$
  $\mathbf{w}_1 = \texttt{HighBits}_q(\mathbf{w},\, 2\,\gamma_2)$
  $\tilde{c} \in \{0,1\}^{256} := \mathrm{H}(\mu \,||\, \mathbf{w}_1)$
  $c \in B_\tau := \texttt{SampleInBall}(\tilde{c})$ ▷ Store c in `NTT` representation as $\hat{c} = \texttt{NTT}(c)$
  $\mathbf{z} := \mathbf{y} + c\,\mathbf{s}_1$ ▷ Compute $c\mathbf{s}_1$ as $\texttt{NTT}^{-1}(\hat{c} \cdot \hat{s}_1)$
  $\mathbf{r}_0 := \texttt{LowBits}_q(\mathbf{w} - c\mathbf{s}_2,\, 2\,\gamma_2)$ ▷ Compute $c\mathbf{s}_2$ as $\texttt{NTT}^{-1}(\hat{c} \cdot \hat{s}_2)$
  if $\|\mathbf{z}\|_\infty \geq \gamma_1 - \beta$ or $\|\mathbf{r}_0\|_\infty \geq \gamma_2 - \beta$ **then**
    $(\mathbf{z}, \mathbf{h}) := \perp$
  else
    $\mathbf{h} := \texttt{MakeHint}_q(-c\mathbf{t}_0, \mathbf{w} - c\mathbf{s}_2 + c\mathbf{t}_0, 2\gamma_2)$ ▷ Compute $c\,\mathbf{t}_0$ as $\texttt{NTT}^{-1}(\hat{c} \cdot \hat{\mathbf{t}}_0)$
    if $||c\,\mathbf{t}_0||_\infty \geq \gamma_2$ or $|\mathbf{h}|_{\mathbf{h}_j=1} > \omega$ **then**
      $(\mathbf{z}, \mathbf{h}) := \perp$
  $\kappa := \kappa + l$
return $\sigma = (\tilde{c}, \mathbf{z}, \mathbf{h})$

Verification: From the signature, c is recalculated. The verifier then uses the hints to recalculate $\mathbf{w}_1'$ the value to which the signer has committed. If the

commitment is correct and $\mathbf{z}$ meets other conditions, the signature is accepted. Otherwise it is rejected. The verification algorithm is described in Algorithm 4.

Algorithm 4 `Ver`

Require: pk, σ

1: $\mathbf{A} \in \mathcal{R}_q^{k \times l} := \texttt{ExpandA}(\rho)$
2: $\mu \in \{0,1\}^{512} := \mathrm{H}(\mathrm{H}(\rho \,||\, \mathbf{t}_1) \,||\, M)$
3: $c := \texttt{SampleInBall}(\tilde{c})$
4: $\mathbf{w}_1' := \texttt{UseHint}_q(\mathbf{h}, \mathbf{Az} - c\mathbf{t}_1 \cdot 2^d, 2\gamma_2)$
5: **return** $[\![||\mathbf{z}||_\infty < \gamma_1 - \beta]\!]$ and $[\![\tilde{c} = \mathrm{H}(\mu \,||\, \mathbf{w}_1')]\!]$ and $[\![|\mathbf{h}|_{\mathbf{h}_j = 1} \leq \omega]\!]$

2.4 The basis of Polyhedral Theory

In the rest of the paper, we will show that some rejected signatures of Dilithium provide inequalities on the coefficients of $\mathbf{s}_2$. Therefore $\mathbf{s}_2$ is one of the solutions to a set of inequalities. We are going to show that this type of set has certain properties, and we will need to define a natural notion of dimension. As this dimension is linked to the number of points in the set, we will need a practical way of estimating it. We would like to point out that the general definitions and unproven propositions come from [NW88].

Definition 4 *A set of points $x_1, ..., x_k \in \mathbb{R}^n$ is affinely independent if the unique solution of $\sum_{i=1}^k \alpha_i x_i = 0$, $\sum_{i=1}^k \alpha_i = 0$ is $\alpha_i = 0$ for $i = 1, ..., k$.*

Remark 2 *When dealing with linear inequalities it is often more appropriate to use the concept of affine independence, linear independence implies affine independence, but the converse is not true.*

Definition 5 *A polyhedron $P \subset \mathbb{R}^n$ is the set of points that satisfy a finite number of linear inequalities, $P = \{x \in \mathbb{R}^n : \ Ax \leq b\}$ where (A, b) is a $m \times (n+1)$ matrix.*

Definition 6 *A polyhedron $P \subset \mathbb{R}^n$ is bounded if there exists an $w \in \mathbb{R}_+$ such that $P \subset \{x \in \mathbb{R}^n : \ -w \leq x_j \leq w \text{ for } j = 1, ..., n\}$. A bounded polyhedron is called a polytope.*

Definition 7 *A polyhedron P is of dimension k, denoted by $dim(P) = k$, if the maximum number of affinely independent points in P is $k + 1$.*

Remark 3 *It is essential to calculate the dimension of a polytope formed by a set of inequalities efficiently, as the dimension of the polytope gives us a upper-bound on the number of its elements. More importantly, if we collect enough inequalities so that the dimension of the associated polytope becomes 0, we know that $\mathbf{s}_2$ will be the only solution to this set of inequalities.*

Definition 8 *We note a^i the $i-th$ row of A. Let $M = \{1, 2, ..., m\}$, $M^= = \{i \in M : \ a^i x = b_i \text{ for all } x \in P\}$ and let $M^\leq = M \backslash M^=$. Let $(A^=, b^=)$ and $(A^\leq, b^\leq)$ be the corresponding rows of (A, b). We refer to the equality and inequality sets of representation (A, b) of P, that is:*

$$P = \{x \in \mathbb{R}^n : \ A^= = b^=, \ A^\leq x \leq b^\leq\}.$$

Proposition 1 *If $P \subset \mathbb{R}^n$, then $dim(P) + rank(A^=, b^=) = n$.*

Remark 4 *Unfortunately, finding the matrix $(A^=, b^=)$ corresponding to a polyhedron P can be computationally expensive. Instead, we will use the following proposition, which is more appropriate in our case.*

Proposition 2 *Let $P \subset \mathbb{R}^n$ be a polyhedron, let $I = \{i \in M : \ \exists w_i \in \mathbb{R}, \forall x \in P, \ x_i = w_i\}$ then:*

$$dim(P) \leq n - card(I).$$

Proof. There exist real numbers $(w_i)_{i \in I}$ such that $P \subset \tilde{P} = \{x \in \mathbb{R}^n : \ \forall i \in I, \ x_i = w_i\}$ and according to the previous proposition: $dim(\tilde{P}) = n - card(I)$.

2.5 The basis of Integer Programming

In the previous section we looked at the properties of polytopes, which represent sets of solutions to inequalities. Even if we collect enough inequalities for $\mathbf{s}_2$ to be the only solution, we still need to find $\mathbf{s}_2$ efficiently. The aim of this section is to study the basics of integer linear programming, this will enable us to efficiently find a solution to a set of inequalities, in other words: To find a point on the polytope containing $\mathbf{s}_2$. The general linear programming problem is to find:

$$z_{LP} = \max\{cx : \ Ax \leq b, \ x \in \mathbb{R}_+\}$$

where A is a $m \times n$ matrix and c, b are $m \times 1$ matrices. This problem is well defined is the sense that if it is feasible and does not have unbounded optimal values, then it has an optimal solution. In the rest of this paper, we will note (LP) and write it in the following form:

$$\begin{aligned} &\text{maximize } cx \\ &\text{subject to } Ax \leq b \\ &\qquad\qquad x \in \mathbb{R} \end{aligned}$$

One can also define the integer programming problem, noted (IP):

$$\begin{aligned} &\text{maximize } cx \\ &\text{subject to } Ax \leq b \\ &\qquad\qquad x \in \mathbb{Z} \end{aligned}$$

Integer programming is a harder problem than linear programming, linear programming algorithms are very often used as a subroutine in integer programming algorithms to obtain upper bounds on the value of the integer program. Exact resolution algorithms exist and we believe it is important from a theoretical point of view to recall the following theorems:

Theorem 1 *[NW88] For a fixed n there is a polynomial algorithm for the integer programming problem (IP).*

Theorem 2 *[NW88] For a fixed m there is a polynomial algorithm for the integer programming problem (IP).*

For a fixed m, the degree of the polynomial by which the running time of the algorithm in [NW88] is bounded as an exponential function of n. Therefore it does not achieve the performance required to solve certain problems. Instead, a wide range of approximate solvers have been developed which provide much more efficient results. In this paper, we use a free solver called lpsolve [MB04], which uses heuristic methods that are very efficient in practice.

Remark 5 *In our case, we are trying to find $\mathbf{s}_2$ from a number of inequalities on its coefficients. If we collect enough inequalities on the coefficients of $\mathbf{s}_2$, solving an (IP) or (LP) problem will give the same result because $\mathbf{s}_2$ will be the only solution (integer or not). Since solving a (LP) problem is much more efficient, in the rest of the document we will focus on (LP) problems related to $\mathbf{s}_2$.*

3 Problem definition and reformulation

We study the case of an attacker who retrieves rejected signatures. More precisely, we want to mount a practical attack against Dilithium without the first or second condition from the line 13. The attack methodology we used is independent of the condition. However, attacking Dilithium without the second condition in line 13 of Algorithm 3 is less straightforward because we first need to retrieve $\mathbf{w}_1$ to exploit such signatures. In this paper, we have chosen to focus on this attack. Nonetheless, sub-section 6.4 briefly explains how to transpose the attack on Dilithium without the first condition and gives the obtained experimental results. Formally, we define another signature algorithm called `F-Sig` in Algorithm 5, and we demonstrate the existence of a practical attack against it.

Remark 6 *As Dilithium's proof of security does not use the knowledge of $\mathbf{t}_0$, most of the literature considers it public data. Recently, a paper published on eprint [AOCVCG24] appears to prove that $\mathbf{t}_0$ can indeed be reconstructed from a reasonable amount of Dilithium signatures. From now on and in the rest of the paper, we will assume that $\mathbf{t}_0$ is public.*

Remark 7 *The verification algorithm will not always accept the signature generated by* `F-Sig`*. Moreover, we know in advance that the Dilithium security proof does not apply here, as we have deliberately removed a security check. In the event of a physical attack that would skip this condition, the security of Dilithium would fall back to* `F-Sig`*.*

Algorithm 5 `F-Sig`

Require: sk, M
Ensure: $\sigma = (\tilde{c}, \mathbf{z}, \mathbf{h})$

$\mathbf{A} \in \mathcal{R}_q^{k \times l} := \texttt{ExpandA}(\rho)$ ▷ $\mathbf{A}$ is generated and stored in NTT as $\hat{\mathbf{A}}$
$\mu \in \{0,1\}^{512} := \mathrm{H}(tr \,||\, M)$
$\kappa := 0$, $(\mathbf{z}, \mathbf{h}) := \perp$
$\rho' \in \{0,1\}^{512} := \mathrm{H}(K \,||\, \mu)$
while $(\mathbf{z}, \mathbf{h}) = \perp$ **do** ▷ Pre-compute $\hat{s}_1 := \texttt{NTT}(\mathbf{s}_1)$, $\hat{s}_2 := \texttt{NTT}(\mathbf{s}_2)$ and $\hat{\mathbf{t}}_0 := \texttt{NTT}(\mathbf{t}_0)$
  $\mathbf{y} \in \tilde{S}_{\gamma_1}^l := \texttt{ExpandMask}(\rho', \kappa)$
  $\mathbf{w} := \mathbf{A}\,\mathbf{y}$ ▷ $\mathbf{w} := \texttt{NTT}^{-1}(\hat{\mathbf{A}} \cdot \texttt{NTT}(\mathbf{y}))$
  $\mathbf{w}_1 = \texttt{HighBits}_q(\mathbf{w}, 2\,\gamma_2)$
  $\tilde{c} \in \{0,1\}^{256} := \mathrm{H}(\mu \,||\, \mathbf{w}_1)$
  $c \in B_\tau := \texttt{SampleInBall}(\tilde{c})$ ▷ Store c in `NTT` representation as $\hat{c} = \texttt{NTT}(c)$
  $\mathbf{z} := \mathbf{y} + c\,\mathbf{s}_1$ ▷ Compute $c\mathbf{s}_1$ as $\texttt{NTT}^{-1}(\hat{c} \cdot \hat{s}_1)$
  $\mathbf{r}_0 := \texttt{LowBits}_q(\mathbf{w} - c\mathbf{s}_2, 2\,\gamma_2)$ ▷ Compute $c\mathbf{s}_2$ as $\texttt{NTT}^{-1}(\hat{c} \cdot \hat{s}_2)$
  if $\|\mathbf{z}\|_\infty \geq \gamma_1 - \beta$ **then**
    $(\mathbf{z}, \mathbf{h}) := \perp$
  else
    $\mathbf{h} := \texttt{MakeHint}_q(-c\mathbf{t}_0, \mathbf{w} - c\mathbf{s}_2 + c\mathbf{t}_0, 2\gamma_2)$ ▷ Compute $c\,\mathbf{t}_0$ as $\texttt{NTT}^{-1}(\hat{c} \cdot \hat{\mathbf{t}}_0)$
    if $||c\,\mathbf{t}_0||_\infty \geq \gamma_2$ or $|\mathbf{h}|_{\mathbf{h}_j=1} > \omega$ **then**
      $(\mathbf{z}, \mathbf{h}) := \perp$
  $\kappa := \kappa + l$
return $\sigma = (\tilde{c}, \mathbf{z}, \mathbf{h})$

3.1 Turning it into a linear programming problem

The aim of this part is to prove (under hypotheses verified in practice) that a non-negligible proportion of signatures of `F-Sig` provide inequalities on the coefficients of $\mathbf{s}_2$. Thus, finding $\mathbf{s}_2$ from a set of signatures of `F-Sig` is equivalent to finding $\mathbf{s}_2$ among the points of a polytope defined by a set of inequalities. If we collect enough inequations so that the dimension of the polytope containing $\mathbf{s}_2$ is 0, we can find it using linear programming. The first step is to show the following: from a signature $\sigma = (\tilde{c}, \mathbf{z}, \mathbf{h})$ of `F-Sig` we can find the polynomial vector $\mathbf{w}_1$ used in the signature. To do this, we need a hypothesis that will be verified in practice, through simulations.

Assumption 1 *With overwhelming probability, for a signature of* `F-Sig` *the polynomial vector* $\mathbf{w}_1 - \mathbf{w}_1'$ *has at most one non-zero coefficient, which will be* 1 *or* -1.

Proposition 3 *Under Assumption 1, if* σ *is a signature of* `F-Sig`*, with overwhelming probability we can find* $\mathbf{w}_1$ *by knowing* $\mathbf{w}_1'$.

Proof. If the signature is accepted by the verification, we get directly $\mathbf{w}_1 = \mathbf{w}'_1$. If the signature is rejected, we carry out an exhaustive search on the coefficients of $\mathbf{w}_1$ (because we know that $c = \mathrm{H}(\mu \,||\, \mathbf{w}_1)$). According to Assumption 1, we will have at most $2 \times k \times 256$ values to test.

Proposition 4 *For any $\sigma = (\tilde{c}, \mathbf{z}, \mathbf{h})$ signature of* `F-Sig` *that is not accepted by the verification algorithm, there exists a unique $j \in \{1, ..., k\}$ and a unique $i \in \{0, ..., 255\}$ such that:*

- *if $(\mathbf{w}_1 - \mathbf{w}'_1)_i^{[j]} = 1$:*
$$(c\mathbf{s}_2)_i^{[j]} \geq \gamma_2 - \mathbf{r}_{0,i}^{[j]} \geq 0,$$
- *if $(\mathbf{w}_1 - \mathbf{w}'_1)_i^{[j]} = -1$:*
$$(c\mathbf{s}_2)_i^{[j]} \leq -\gamma_2 - \mathbf{r}_{0,i}^{[j]} \leq 0.$$

Proof. Let $\sigma = (\tilde{c}, \mathbf{z}, \mathbf{h})$ be a rejected signature. We have $\mathbf{w}_1 \neq \mathbf{w}'_1$. If we assume that the non-zero coefficient of $\mathbf{w}_1 - \mathbf{w}'_1$ is 1, according to Assumption 1, there exists a unique $j \in \{1, ..., k\}$ and a unique $i \in \{0, ..., 255\}$ such that:

$$\mathtt{HighBits}_q((\mathbf{w})_i^{[j]}, 2\,\gamma_2) = \mathtt{HighBits}_q((\mathbf{w}')_i^{[j]},\, 2\,\gamma_2) + 1.$$

Thus, one has

$$\mathtt{HighBits}_q(\mathbf{A}\mathbf{y}, 2\,\gamma_2)_i^{[j]} = \mathtt{HighBits}_q(\mathbf{A}\mathbf{y} - c\mathbf{s}_2,\, 2\,\gamma_2)_i^{[j]} + 1,$$

since $\mathbf{r}_0 = \mathtt{LowBits}_q(\mathbf{A}\mathbf{y} - c\mathbf{s}_2, 2\,\gamma_2)_i^{[j]}$, we have:

$$(c\mathbf{s}_2)_i^{[j]} \geq \gamma_2 - \mathbf{r}_{0,i}^{[j]} \geq 0.$$

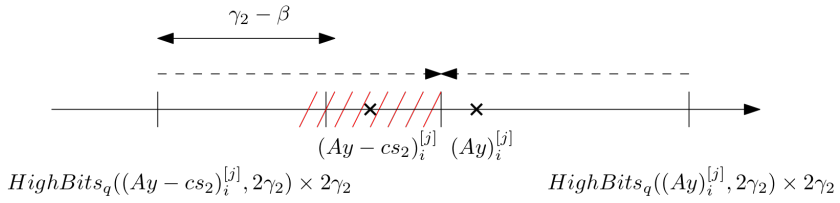

Fig. 2. In red, impossible values for $(c\mathbf{s}_2)_i^{[j]}$

The same arguments can be used to show the second inequality when the non-zero coefficient of $\mathbf{w}_1 - \mathbf{w}'_1$ is -1.

Remark 8 *Since $\sigma = (\tilde{c}, \mathbf{z}, \mathbf{h})$, $\mathbf{A}$ and $\mathbf{t}_0$ are known, $\mathbf{r}_0$ can be calculated, using the relation $\mathbf{A}\mathbf{z} - c\mathbf{t} = \mathbf{A}\mathbf{y} - c\mathbf{s}_2$. Therefore, each signature not accepted by the*

verification algorithm `Ver` *will provide an inequality verified by certain coefficients of the polynomial vector* $\mathbf{s}_2$*. We are going to use the linear programming theory, firstly to estimate how many inequalities it would take for* $\mathbf{s}_2$ *to be the only solution, and secondly to find this solution efficiently. As [BP18,RJH+18] shows, recovering* $\mathbf{t}_0$ *and* $\mathbf{s}_2$ *allows to forge arbitrary signatures, and to an equivalent key recovery.*

Building the (*LP*) system After collecting enough signatures, we will have multiple inequalities on the k polynomials of $\mathbf{s}_2$ independently, so we can split the problem into k smaller ones, one for each polynomial of the vector $\mathbf{s}_2$. For the sake of clarity, let us explain the methodology for a single polynomial of the vector $\mathbf{s}_2 = (\mathbf{s}_2^{[1]}, ..., \mathbf{s}_2^{[k]})$. We select a signature that gives an inequation on $\mathbf{s}_2^{[1]}$. Let $\sigma = (\tilde{c}, \mathbf{z}, \mathbf{h})$ be such a signature, we have:

If $(\mathbf{w}_1 - \mathbf{w}'_1)_i^{[1]} = 1$:

$$(c\mathbf{s}_2)_i^{[1]} \geq \gamma_2 - \mathbf{r}_{0,i}^{[1]} \tag{1}$$

$$\sum_{j=0}^{n-1} s_{2,j}^{[1]}(cx^j)_i \geq \gamma_2 - \mathbf{r}_{0,i}^{[1]} \tag{2}$$

Since the polynomial c is known, σ gives us an inequality on the coefficients of $\mathbf{s}_2^{[1]}$. The case of $(\mathbf{w}_1 - \mathbf{w}'_1)_i^{[1]} = -1$ is treated in the same way. Thus, with its rejected signatures, we can construct two matrices A_+ and A_- and two vectors b_+ and b_- such that $\mathbf{s}_2^{[1]} \in \{x \in [-\eta, \eta]^n \mid A_+x \geq b_+ \text{ and } A_-x \leq b_-\}$. Each row of one of these matrices representing an inequality collected on $\mathbf{s}_2^{[1]}$. In particular, if we collect enough inequalities for $\mathbf{s}_2^{[1]}$ to be the only solution, we can find $\mathbf{s}_2^{[1]}$ by solving the following (LP) problem of dimension $n = 256$:

$$\begin{array}{ll} \text{maximize} & 0 \\ \text{subject to} & A_+x \geq b_+ \\ & A_-x \leq b_- \\ & x \in [-\eta, \eta]^n \end{array}$$

Fig. 3. The (LP) problem related to $\mathbf{s}_2^{[1]}$.

4 Differences between specification algorithm and reference implementation

For Dilithium, the reference implementation noted $\mathtt{Sig}_{\mathtt{Ref}}$ uses an alternative way of decomposing and calculating hints, to avoid calling the $\mathtt{Decompose}_q$ function

three times. This method, which we will note $\texttt{MakeHint\_ref}_q$, is detailed in the Dilithium specification [DKL+22], in Section 5.1. Remark that this alternative method to compute the hints is no longer equivalent if we remove the condition on $\tilde{\mathbf{r_0}}$. Rather than re-describing an attack by detailing the entire procedure, we explain only the main ideas for transforming our attack into an attack against $\texttt{Sig}_{\texttt{Ref}}$. Algorithm 6 gives the pseudo code of $\texttt{Sig}_{\texttt{Ref}}$.

Remark 9 *To avoid copying the* $\texttt{Sig}_{\texttt{Ref}}$ *and* $\texttt{F-Sig}_{\texttt{Ref}}$ *algorithms, which are similar, we only write* $\texttt{Sig}_{\texttt{Ref}}$ *.* $\texttt{F-Sig}_{\texttt{Ref}}$ *is obtained by removing the condition* $\|\tilde{\mathbf{r}}_0\|_\infty \geq \gamma_2 - \beta$ *on line* 13*. Even if* $\texttt{Sig}_{\texttt{Ref}}$ *and* $\texttt{Sig}$ *work in the same way, our attack on* $\texttt{F-Sig}_{\texttt{Ref}}$ *will be more difficult to detect, as all the signatures produced will be accepted by the verification algorithm.*

Algorithm 6 $\texttt{Sig}_{\text{Ref}}$

Require: sk, M
Ensure: $\sigma = (\tilde{c}, \mathbf{z}, \mathbf{h})$

1: $\mathbf{A} \in R_q^{k \times l} := \texttt{ExpandA}(\rho)$ ▷ $\mathbf{A}$ is generated and stored in NTT as $\hat{\mathbf{A}}$
2: $\mu \in \{0,1\}^{512} := \mathrm{H}(tr \,||\, M)$
3: $\kappa := 0$, $(\mathbf{z}, \mathbf{h}) := \perp$
4: $\rho' \in \{0,1\}^{512} := \mathrm{H}(K \,||\, \mu)$
5: **while** $(\mathbf{z}, \mathbf{h}) = \perp$ **do** ▷ Pre-compute $\hat{s}_1 := \texttt{NTT}(\mathbf{s}_1)$, $\hat{s}_2 := \texttt{NTT}(\mathbf{s}_2)$ and $\hat{\mathbf{t}}_0 := \texttt{NTT}(\mathbf{t}_0)$
6: $\quad \mathbf{y} \in \tilde{S}_{\gamma_1}^l := \texttt{ExpandMask}(\rho', \kappa)$
7: $\quad \mathbf{w} := \mathbf{A}\,\mathbf{y}$ ▷ $\mathbf{w} := \texttt{NTT}^{-1}(\hat{\mathbf{A}} \cdot \texttt{NTT}(\mathbf{y}))$
8: $\quad (\mathbf{w}_1, \mathbf{w}_0) = \texttt{Decompose}_q(\mathbf{w}, 2\,\gamma_2)$
9: $\quad \tilde{c} \in \{0,1\}^{256} := \mathrm{H}(\mu \,||\, \mathbf{w}_1)$
10: $\quad c \in B_\tau := \texttt{SampleInBall}(\tilde{c})$ ▷ Store c in NTT representation as $\hat{c} = \texttt{NTT}(c)$
11: $\quad \mathbf{z} := \mathbf{y} + c\,\mathbf{s}_1$ ▷ Compute $c\mathbf{s}_1$ as $\texttt{NTT}^{-1}(\hat{c} \cdot \hat{s}_1)$
12: $\quad \tilde{\mathbf{r}}_0 := \mathbf{w}_0 - c\mathbf{s}_2$ ▷ Compute $c\mathbf{s}_2$ as $\texttt{NTT}^{-1}(\hat{c} \cdot \hat{s}_2)$
13: $\quad$ **if** $\|\mathbf{z}\|_\infty \geq \gamma_1 - \beta$ or $\|\tilde{\mathbf{r}}_0\|_\infty \geq \gamma_2 - \beta$ **then**
14: $\quad\quad (\mathbf{z}, \mathbf{h}) := \perp$
15: $\quad$ **else**
16: $\quad\quad \mathbf{h} := \texttt{MakeHint\_ref}_q(\mathbf{w}_1, \mathbf{w}_0 - c\mathbf{s}_2 + c\mathbf{t}_0, 2\gamma_2)$
17: $\quad\quad$ **if** $||c\,\mathbf{t}_0||_\infty \geq \gamma_2$ or $|\mathbf{h}|_{\mathbf{h}_j=1} > \omega$ **then**
18: $\quad\quad\quad (\mathbf{z}, \mathbf{h}) := \perp$
19: $\quad \kappa := \kappa + l$
20: **return** $\sigma = (\tilde{c}, \mathbf{z}, \mathbf{h})$

Assumption 2 *The signature made by* $\texttt{F-Sig}_{\texttt{Ref}}$ *will always be accepted by the Dilithium verification algorithm* $\texttt{Ver}$*.*

Proposition 5 *Under Assumption 2, let* $\sigma = (\tilde{c}, \mathbf{z}, \mathbf{h})$ *be a signature of* $\texttt{F-Sig}_{\texttt{Ref}}$*, then either* $\mathbf{w}_1 = \texttt{HighBits}_q(\mathbf{Az} - c\mathbf{t}, 2\gamma_2)$ *or there exists at least one* $j \in \{1, ..., k\}$ *and at least one* $i \in \{0, ..., 255\}$ *such that:*

- *if* $(\mathbf{w}'_1 - \texttt{HighBits}_q(\mathbf{Az} - c\mathbf{t}, 2\gamma_2))_i^{[j]}$ *is positive:*

$$(c\mathbf{s}_2)_i^{[j]} \geq \gamma_2 - \boldsymbol{r}_{0,i}^{[j]} \geq 0,$$

- *if* $(\mathbf{w}'_1 - \texttt{HighBits}_q(\mathbf{Az} - c\mathbf{t}, 2\gamma_2))_i^{[j]}$ *is negative:*

$$(c\mathbf{s}_2)_i^{[j]} \leq -\gamma_2 - \boldsymbol{r}_{0,i}^{[j]} \leq 0.$$

Proof. Let $\sigma = (\tilde{c}, \mathbf{z}, \mathbf{h})$ be a signature of $\texttt{F-Sig}_{\texttt{Ref}}$which verifies Assumption 2. Since the signature is validated by the verification algorithm, $\mathbf{w}_1 = \mathbf{w}'_1$. Lets assume that for this σ, $\mathbf{w}_1 \neq \texttt{HighBits}_q(\mathbf{Az} - c\mathbf{t}, 2\gamma_2)$. We have $\mathbf{Az} - c\mathbf{t} = \mathbf{Ay} - c\mathbf{s}_2$ so $\mathbf{w}_1 \neq \texttt{HighBits}_q(\mathbf{Ay} - c\mathbf{s}_2, 2\gamma_2)$. There exists a $j \in \{1, ..., k\}$ and a $i \in \{0, ..., 255\}$ such that:

$$\texttt{HighBits}_q((\mathbf{Ay})_i^{[j]}, 2\,\gamma_2) \neq \texttt{HighBits}_q((\mathbf{Ay} - c\mathbf{s}_2)_i^{[j]},\, 2\,\gamma_2).$$

If $(\mathbf{w}'_1 - \texttt{HighBits}_q(\mathbf{Az} - c\mathbf{t}, 2\gamma_2))_i^{[j]}$ is positive, then because $\|c\mathbf{s}_2\|_\infty \leq \beta$:

$$\texttt{HighBits}_q((\mathbf{Ay})_i^{[j]}, 2\,\gamma_2) = \texttt{HighBits}_q((\mathbf{Ay} - c\mathbf{s}_2)_i^{[j]},\, 2\,\gamma_2) + 1,$$

and:

$$(c\mathbf{s}_2)_i^{[j]} \geq \gamma_2 - \texttt{LowBits}_q((\mathbf{Ay} - c\mathbf{s}_2)_i^{[j]},\, 2\,\gamma_2) \geq 0.$$

But by definition, $\mathbf{r}_0 := \texttt{LowBits}_q(\mathbf{Ay} - c\mathbf{s}_2,\, 2\,\gamma_2)$, finally:

$$(c\mathbf{s}_2)_i^{[j]} \geq \gamma_2 - \mathbf{r}_{0,i}^{[j]} \geq 0.$$

The same arguments can be used to show the second inequality when $(\mathbf{w}'_1 - \texttt{HighBits}_q(\mathbf{Az} - c\mathbf{t}, 2\gamma_2))_i^{[j]}$ is negative.

5 Finding a polytope

The aim is to find the $k \times 256$ coefficients of the polynomial vector $\mathbf{s}_2$. But as explained in Section 3, we can find the coefficients of each polynomial of $\mathbf{s}_2$ separately. In the following, we will only study how to find the first polynomial of $\mathbf{s}_2$. By signing messages, we will obtain inequalities verified by $\mathbf{s}_2^{[1]}$. Once we have obtained enough inequalities, $\mathbf{s}_2^{[1]}$ is the only solution, and we can find it by solving a (LP) problem with a arbitrary objective function, such as the null function. In other words, we want enough inequalities so that the dimension of the associated polytope $P \subset [-\eta, \ldots, \eta]^{256}$ is 0. By the same procedure, we will find all the polynomials in $\mathbf{s}_2$, in practice in Section 6. To estimate the

dimension of the polytope P, we will use proposition 2 proved in Section 2. For $i \in \{1, \ldots, n\}$, by solving the following two (LP) problems:

$$\begin{array}{ll} \text{minimize} & x_i \\ \text{subject to} & A_+ x \geq b_+ \\ & A_- x \leq b_- \\ & x \in [-\eta, \eta]^n \end{array} \qquad \begin{array}{ll} \text{maximize} & x_i \\ \text{subject to} & A_+ x \geq b_+ \\ & A_- x \leq b_- \\ & x \in [-\eta, \eta]^n \end{array}$$

Fig. 4. The 2×256 (LP) problems related to $\mathbf{s}_2^{[1]}$.

We can calculate $\text{card}(\{i \in \{1, \ldots, n\} : \exists w_i \in \mathbb{R}, \forall x \in P,\ x_i = w_i\})$ and therefore upper-bound the dimension of the polytope containing $\mathbf{s}_2^{[1]}$.

Remark 10 *To estimate $dim(P)$ we need to solve 2×256 (LP) problems (2 for each coordinate function). Some of these problems were costly because the solution time depends on the function chosen and can soar when the number of inequations is insufficient. Despite efforts, we were unable to produce statistics using exactly this method. In the following section, we described a slightly modified method to estimate the number of inequalities required for the dimension of the polytope containing $\mathbf{s}_2^{[1]}$ to become 0.*

5.1 Evolution of polytopes dimensions

Our goal is to provide an empirical justification for the number of inequalities required to find $\mathbf{s}_2^{[1]}$. We will do what an attacker might do: we choose random keys and simulate not having access to the j first coefficients of $\mathbf{s}_2^{[1]}$. By collecting rejected signatures, we will obtain inequalities on the "missing" $\mathbf{s}_2^{[1]}$coefficients. By doing this, we reduce ourselves to a polytope of lower maximum dimension (of maximum dimension j). The corresponding $2 \times j$ (LP) problems will be less costly to solve. By solving these problems for increasing values of j, we can try to guess the number of inequalities required when no coefficients are known. In this subsection only, we assume that some coefficients of $\mathbf{s}_2^{[1]}$ are known. We sign messages with `F-Sig` to obtain inequalities on the missing coefficients of $\mathbf{s}_2^{[1]}$. Table 1 summarizes the obtained results, for Dilithium-2 [4]

[4] For each time and probability of success, this is an average over 100 randoms keys.

Unknown coefficients	32	64	128	256
Nb tests	100	100	100	-
Inequalities	323	1306	3917	10 445 (predicted)
Polytopes dimensions	0	0	0	-
Attack time	1.36 s	17.4s	227.3s	-

Table 1. Evolution of the dimension as a function of the unknowns

The number of inequations appears to be linear on the number of unknowns. Based on the results, we conjecture that on average 10 000 inequalities will be sufficient to guarantee that $\mathbf{s}_2^{[1]}$ is the unique solution to the associated (LP) problem. Hence an opponent who does not know any coefficient of $\mathbf{s}_2^{[1]}$ will need 10 000 inequalities on average to find it.

Remark 11 *Note that even if the theory remains unchanged, these practical results are highly dependent on the size of the secret coefficients* $\mathbf{s}_2$*, and therefore on the security level of Dilithium. So, the expected number of inequalities needed to recover the* 256 *coefficients is likely to change with the security level.*

6 Experimental results

The purpose of this section is to evaluate the usability of Proposition 4 and 5 in practice. We tested the key recovery methodology for both versions of Dilithium, `Sig` and $\texttt{Sig}_{\texttt{Ref}}$, as well as the three different security levels, Dilithium-2, Dilithium-3, and Dilithium-5.

Experimental Setup We use the C reference implementation of Dilithium from [DKL$^+$22] as well as a modified version that follows the specification [LDK$^+$22]. We adapt both of them to get implementations of `F-Sig` and $\texttt{F-Sig}_{\texttt{Ref}}$, as stated in Algorithm 5 and Algorithm 6. We use the resulting signatures in a Sage script that allows us to formulate the (LP) problems for a given secret key. The (LP) solving is done using the lpsolve library from Python. The tests were done on a laptop equipped with an Intel(R) Core(TM) i7-10850H 2.70GHz CPU. All the materials used to collect the signatures and perform the attack are available at `https://github.com/anders1901/Polytope_attack`. For our study, we focus on finding all polynomials of $\mathbf{s}_2$. In our evaluations attack time means the time taken to find $\mathbf{s}_2$ once the inequations have been extracted from the signatures generated.

Remark 12 *In a fault attack scenario, various fault injection techniques, such as clock and voltage glitches, laser, and electromagnetic pulse injection can lead to the skipping of an instruction [DRPR19,MDP*$^+$*20,CPHR21]. In the context of our attack, we are interested in skipping the call to the* `polyveck_checknorm` *function, which allows us to output signatures without checking the norm of*

$\mathbf{z}$ *or* $\mathbf{r}_0$*. Our attack applies to both the deterministic and randomized versions of Dilithium. However, targeting the randomized version may require a more powerful attacker model. Indeed, injecting faults into the randomized version generally involves taking into account the rejection sampling step of the signature algorithm without prior analysis of the signature execution trace.*

Remark 13 *For both the specification and implementation of Dilithium we focus on retrieving the 100 keys produced by the KAT from [DKL+22].*

6.1 Attack on Dilithium's specification

In the previous section, we conjectured that approximately 10 445 inequalities per polynomial of $\mathbf{s}_2$ are needed to determine its $k \times 256$ coefficients. Therefore, the primary goal is to determine the number of signatures of `F-Sig` required to collect the given number of inequalities.

Assumption 1 in practice To measure the frequency with which Assumption 1 was verified, we collected 1 250 000 signatures for an equal number of random messages for each of the 100 keys obtained from the KAT files. Of the total 1 250 000 signatures for each key, more than half have at least one coefficient among the $k \times n$ exceeding the bound $\gamma_2 - \beta$. On average within this subset of 717 448 signatures, 46 459 do not pass signature verification, indicating potential exploitability under Assumption 1. This assumption states that the vast majority of these 46 459 signatures are likely to have no more than one coefficient where $\mathbf{w}_1$ and $\mathbf{w}'_1$ differ by a magnitude of 1 or -1. We tested for this on the set of 46 459 signatures and the experimental results are summarized in Table 2.

signatures	1 coefficient changed	2 or more coefficients changed	inequalities/polynomial
1 250 000	45 584	874	11 085

Table 2. Average Inequalities collected for 46 459 signatures over 100 keys

From these results we conclude that around 3.6% of the 1 250 000 signatures can be exploited. The vast majority of rejected signatures will provide an inequation on one of the coefficients of a polynomial of $\mathbf{s}_2$, and can be used in the formulation of the (LP) problem. In the worst case, if the hypothesis is not verified, in other words if $\mathbf{w}_1 - \mathbf{w}'_1 \neq \pm 1$, the attacker will be unable to exploit the signature produced by `F-Sig`, as he cannot find the inequation verified by $\mathbf{s}_2$. He simply discards this signature and proceeds with the next one.

Attack results After collecting enough inequalities for each of the k polynomials, we expect to recover the entirety of the $\mathbf{s}_2$ vector based on the analysis

in Section 5. For this, we formulated the k (LP) problem for $\mathbf{s}_2$, as depicted in Fig. 3. In order to be able to produce statistics in a reasonable amount of time, we set the solver resolution time to 30 minutes maximum. Table 3 summarizes the results obtained.

Signatures	Average inequalities	Success probability	Average time	Median Time
1 250 000	11 085	0.99	277.53s	180.00s

Table 3. Average results of the attack on `F-Sig`

We conclude that our attack is very efficient. Moreover, when the key is not found, it is systematically because the solver was unable to solve the system of inequalities in the given time. By increasing the limit we have set to more than 30 minutes, in about 2 and a half hours of calculation we were able to find the missing key.

6.2 Attack on Dilithium's implementation

As stated in Section 4, the attack can easily be mounted for signatures produced by the reference implementation, by using Proposition 5. For completeness, we detail the results obtained for the reference implementation of Dilithium-2.

Assumption 2 in practice Just like in the previous sub-section, we collected 1 250 000 signatures for random messages for each of the 100 keys obtained from the KAT files. Among the 1 250 000 signatures collected for each key, in average 717 448 have at least one coefficient among the $k \times n$ exceeding the bound $\gamma_2 - \beta$. But, this time, all the signatures of the algorithm $\texttt{F-Sig}_{\texttt{Ref}}$ are accepted by the verification algorithm `Ver`, as stated in Assumption 2.

signatures	1 coefficient changed	2 or more coefficients changed	inequalities/polynomial
1 250 000	45 578	875	11 083

Table 4. Average Inequalities collected over 100 keys

Attack results Here also we formulate the (LP) problem as in Fig. 3 but with the inequalities from Proposition 5. Using the same methodology as for sub-section 6.1, we tried to recover the 100 keys from the KAT files. The results are summarized in Table 5.

Signatures	Average inequalities	Sucess probability	Average time	Median time
1 250 000	11 083	0.98	261.79s	148.79s

Table 5. Average results of the attack on $\texttt{F-Sig}_{\text{Ref}}$

Once again, we can see that the attack is very effective and works as the attack described for Dilithium's specification. Note also that when the keys are not recovered, it is always due to solver timeouts. Therefore, we can assume that increasing the solver's limit to more than 30 minutes would allow us to recover the missed keys.

6.3 Attack on Dilithium-3 and Dilithium-5

Since the theory presented above does not change according to the security level of Dilithium, we give the results obtained for the same attack against Dilithium-3 and Dilithium-5. The relevant parameters are summarized in Table 6. Note that the Dilithium specification and its reference implementation are also functionally equivalent. The only change in the attack is the condition to collect the inequality, not the inequality itself. Therefore, as a proof of concept, we decided to focus on evaluating the sensitivity of Dilithium's reference implementation. For this subsection, the evaluation is done only on the first 10 of KAT files. Finally, to confirm that we could eventually have a success rate of 100%, we decided not to set a timeout for the solver.

Security level	2	3	5
(k, l)	$(4, 4)$	$(6, 5)$	$(8, 7)$
γ_1	2^{17}	2^{19}	2^{19}
γ_2	$(q-1)/88$	$(q-1)/32$	$(q-1)/32$
η	2	4	2
τ	39	49	60

Table 6. Settings for different security levels of Dilithium

The primary difference between the different security levels is the dimension of the module, parameterized by k and l. Specifically, for $\mathbf{s}_2$, the relevant dimension is $k = 6$ for Dilithium-3 and $k = 8$ for Dilithium-5. Consequently, due to the increased size of the vector, a larger number of signatures is required to ensure the minimum number of inequalities needed to initiate the attack. Additionally, another difference is the size of η, which is larger for Dilithium-3 compared to both Dilithium-2 and Dilithium-5. This change requires either a greater number of inequalities or an extended solver runtime to recover the coefficients of $\mathbf{s}_2$ for Dilithium-3.

For Dilithium-3: Based on the statistics we ran on the dimension of the polytope for the parameters in Table 6, we estimated that we could run our attack with 3 000 000 signatures collected (i.e., about 18 000 inequalities). In practice, since no timeout was set, and to keep the solver's runtime to no more than twice that of Dilithium-2, we set the number of signatures to 3 500 000 (i.e., about 22 000 of inequalities). It provided a balanced trade-off between the number of signatures to collect and the solver's runtime. Table 7 summarizes the results obtained.

Signatures	Average inequalities	Success probability	Average time	Median time
3 500 000	22 020	1	1 239.36s	767.69s

Table 7. Average results of the attack for Dilithium-3

For Dilithium-5: Once again, using the statistics made on the dimension of the polytope, we estimate that the same number of inequalities as for Dilithium-2 will need to be collected. However, because there is twice as much polynomials in the vector $\mathbf{s}_2$, we will need to acquire at least twice more signatures. To be conservative, we collected 4 000 000 signatures. The results obtained are summarized in Table 8.

Signatures	Average inequalities	Success probability	Average time	Median time
4 000 000	15 348	1	186.78s	177.59s

Table 8. Average results of the attack for Dilithium-5

6.4 Impact on the norm check of z

If we remove the condition on $\mathbf{z}$ line 13 of the algorithm `Sig`, we can obtain vectors of polynomials $\mathbf{z}$ which satisfy $\|\mathbf{z}\|_\infty \geq \gamma_1$. According to the definition of $\mathbf{y}$, such a $\mathbf{z}$ provides an inequality on one of the coefficients of $\mathbf{s}_1$. We can exploit these inequalities in the same way as in Section 3 to find $\mathbf{s}_1$. We believe that it is easy for an attentive reader to use these inequalities to find $\mathbf{s}_1$ using the same method as described in this paper. Nevertheless, we explicit the proposition that allows us to obtain the inequalities:

Proposition 6 *Let $\sigma = (\tilde{c}, \mathbf{z}, \mathbf{h})$ be a rejected signature of* `Sig` *such that $\|\mathbf{z}\|_\infty \geq \gamma_1$, then there exists $j \in \{1, ..., l\}$ and $i \in \{0, ..., 255\}$ such that:*

- *if $\mathbf{z}_i^{[j]} \geq \gamma_1$:*

$$(c\mathbf{s}_1)_i^{[j]} \geq \mathbf{z}_i^{[j]} - \gamma_1 \geq 0,$$

– *if* $\mathbf{z}_i^{[j]} \leq -\gamma_1$*:*

$$(c\mathbf{s}_1)_i^{[j]} \leq \mathbf{z}_i^{[j]} + \gamma_1 \leq 0.$$

Proof. If $\|\mathbf{z}\|_\infty \geq \gamma_1$, then there exists $j \in \{1, ..., l\}$ and $i \in \{0, ..., 255\}$ such that $|\mathbf{z}_i^{[j]}| \geq \gamma_1$. Lets assume that $\mathbf{z}_i^{[j]} \geq \gamma_1$. We have $\mathbf{z}_i^{[j]} = \mathbf{y}_i^{[j]} + (c\mathbf{s}_1)_i^{[j]}$ and by definition of $\mathbf{y}$, $|\mathbf{y}_i^{[j]}| \leq \gamma_1$. Thus,

$$(c\mathbf{s}_1)_i^{[j]} \geq \mathbf{z}_i^{[j]} - \gamma_1 \geq 0.$$

The same arguments can be used to show the second inequality when $\mathbf{z}_i^{[j]} \leq -\gamma_1$.

As a proof of concept, we ran this attack on $\mathbf{s}_1$ for the first 10 keys in the KAT files, for Dilithium-2. Table 9 summarizes the results obtained.

Signatures	Average inequalities	Success probability	Average time	Median time
2 000 000	13 584	1	51.94s	49.76s

Table 9. Average results of the attack for Dilithium-2

Remark 14 *In the Dilithium implementation, the way the signature is packed does not allow us to apply the attack directly, as we have to invert the* `polyz_pack` *function in order to find the coefficient of* $\mathbf{z}$ *which provides an inequality on the coefficients of* $\mathbf{s}_1$*. For our proof of concept, we simply attacked the Dilithium specification, without trying to invert this function.*

7 Conclusion and discussion

In this paper, we created an attack on Dilithium with weakened rejection sampling, using linear programming tools. Since Dilithium's rejection sampling ensures that the scheme is zero knowledge, the existence of such an attack is not surprising. On the other hand, we think it is surprising that this attack is so effective, requiring just a few million signatures and a few minutes of computation on a modern computer. The main use of this result is that it reformulates as a fault-based attack against Dilithium. We have tested this attack against the official Dilithium implementation with simulated faults, as well as another modified version that strictly follows the main specification, for the three security levels of each implementation.

With regard to the feasibility of the attack, it requires between 1 and 4 million signatures, this amount is considered significant but realistic in the side-channel literature. Regarding the solving method, any lp solver can solve the systems provided by the obtained signatures. In this work we have chosen to use lpsolve, a MILP solver, even though in reality it is only used to solve (LP) problems.

The main reason being the solver's performance, we tested the same key recovery with a more generic (and less optimised) lp solver (the one provided by scipy in python) and the solution times were up to 16 times slower. However, as the installation of lp solve can be rather complex, in the artifact we propose a solving with scipy and with lp solve. Finally, to be conservative we have reformulated the problems in the (IP) form for the Dilithium-2 reference implementation, and unsurprisingly we kept the same results with roughly identical computation times.

There are two main consequences of our results. Firstly, we show that rejection sampling is essential for the practical safety of Dilithium: the tests must be protected and not just the values manipulated during the test. Secondly, and perhaps more importantly, the reference implementation behaves differently. Faulty signatures will be accepted by the verification algorithm, which makes fault detection more delicate in restricted environments (for example, verifying signatures before outputting them will not be sufficient).

Acknowledgments

This research was funded in part by the France 2030 program under grant agreement No. ANR-22-PETQ-0008 PQ-TLS and by the ANRT (Association nationale de la recherche et de la technologie).

References

AOCVCG24. Paco Azevedo Oliveira, Andersson Calle Viera, Benoît Cogliati, and Louis Goubin. Uncompressing dilithium's public key. Cryptology ePrint Archive, Paper 2024/1373, 2024.

BAE+24. Olivier Bronchain, Melissa Azouaoui, Mohamed ElGhamrawy, Joost Renes, and Tobias Schneider. Exploiting small-norm polynomial multiplication with physical attacks: Application to crystals-dilithium. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2024(2):359–383, Mar. 2024.

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.

BDK+21. Shi Bai, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. Algorithm specifications and supporting documentation (version 3.1), 2021. `https://pq-crystals.org/dilithium/data/dilithium-specification-round3-20210208.pdf`.

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

BVC+23. Alexandre Berzati, Andersson Calle Viera, Maya Chartouny, Steven Madec, Damien Vergnaud, and David Vigilant. Exploiting intermediate

value leakage in dilithium: A template-based approach. *IACR TCHES*, 2023(4):188–210, 2023.

CKA+21. Zhaohui Chen, Emre Karabulut, Aydin Aysu, Yuan Ma, and Jiwu Jing. An efficient non-profiled side-channel attack on the crystals-dilithium post-quantum signature. In *2021 IEEE 39th International Conference on Computer Design (ICCD)*, pages 583–590, 2021.

CPHR21. Ludovic Claudepierre, Pierre-Yves Péneau, Damien Hardy, and Erven Rohou. Traitor: A low-cost evaluation platform for multifault injection. In *Proceedings of the 2021 International Symposium on Advanced Security on Software and Systems*, ASSS '21, page 51–56, New York, NY, USA, 2021. Association for Computing Machinery.

DKL+. Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, Gregor Seiler, Peter Schwabe, and Damien Stehlé. Official reference implementation of the dilithium signature scheme. `https://github.com/pq-crystals/dilithium/`.

DKL+22. Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, Gregor Seiler, Peter Schwabe, and Damien Stehlé. PQ-CRYSTALS, Dilithium. `https://github.com/pq-crystals/dilithium`, 2022. GitHub repository. Accessed: 2022-12-15.

DRPR19. Jean-Max Dutertre, Timothé 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*, pages 221–237, Cham, 2019. Springer International Publishing.

EAB+23. Mohamed ElGhamrawy, Melissa Azouaoui, Olivier Bronchain, Joost Renes, Tobias Schneider, Markus Schönauer, Okan Seker, and Christine van Vredendaal. From mlwe to rlwe: A differential fault attack on randomized & deterministic dilithium. Cryptology ePrint Archive, Paper 2023/1074, 2023. `https://eprint.iacr.org/2023/1074`.

KLH+20. Il-Ju Kim, Tae-Ho Lee, Jaeseung Han, Bo-Yeon Sim, and Dong-Guk Han. Novel single-trace ML profiling attacks on NIST 3 round candidate dilithium. Cryptology ePrint Archive, Report 2020/1383, 2020. `https://eprint.iacr.org/2020/1383`.

KPLG24. Elisabeth Krahmer, Peter Pessl, Georg Land, and Tim Güneysu. Correction fault attacks on randomized crystals-dilithium. Cryptology ePrint Archive, Paper 2024/138, 2024. `https://eprint.iacr.org/2024/138`.

LDK+22. Vadim Lyubashevsky, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Peter Schwabe, Gregor Seiler, Damien Stehlé, and Shi Bai. CRYSTALS-DILITHIUM. Technical report, National Institute of Standards and Technology, 2022. available at `https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022`.

MB04. Peter Notebaert Michel Berkelaar, Kjell Eikland. lp solve. `https://lpsolve.sourceforge.net/5.5`, 2004. Open source (Mixed-Integer) Linear Programming system.

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 *2020 15th Design & Technology of Integrated Systems in Nanoscale Era (DTIS)*, pages 1–7, 2020.

MUTS22. Soundes Marzougui, Vincent Ulitzsch, Mehdi Tibouchi, and Jean-Pierre Seifert. Profiling side-channel attacks on Dilithium: A small bit-fiddling

leak breaks it all. Cryptology ePrint Archive, Report 2022/106, 2022. https://eprint.iacr.org/2022/106.

NIS23. NIST. Fips 204 (draft): Module-lattice-based digital signature standard. Federal Inf. Process. Stds. (NIST FIPS), National Institute of Standards and Technology, Gaithersburg, MD, 2023. https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.204.ipd.pdf.

NSA22. NSA. Announcing the commercial national security algorithm suite 2.0. National Security Agency, U.S Department of Defense, 2022. https://media.defense.gov/2022/Sep/07/2003071834/-1/-1/0/CSA_CNSA_2.0_ALGORITHMS_.PDF.

NW88. George L. Nemhauser and Laurence A. Wolsey. Integer and combinatorial optimization. In *Wiley interscience series in discrete mathematics and optimization*, 1988.

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.

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.

Sch19. Peter Schwabe. Twitter, 2019. https://twitter.com/cryptojedi/status/1192375176438128641.

UMB+23. Vincent Quentin Ulitzsch, Soundes Marzougui, Alexis Bagia, Mehdi Tibouchi, and Jean-Pierre Seifert. Loop aborts strike back: Defeating fault countermeasures in lattice signatures with ILP. *IACR TCHES*, 2023(4):367–392, 2023.

WNGD23. Ruize Wang, Kalle Ngo, Joel Gärtner, and Elena Dubrova. Single-trace side-channel attacks on crystals-dilithium: Myth or reality? Cryptology ePrint Archive, Paper 2023/1931, 2023. https://eprint.iacr.org/2023/1931.