Uncompressing Dilithium's public key

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. To be competitive with other signature schemes, the MLWE instance $(\mathbf{A}, \mathbf{t})$ on which Dilithium is based is compressed: the least significant bits of $\mathbf{t}$, which are denoted $\mathbf{t}_0$, are considered part of the secret key. Knowing $\mathbf{t_0}$ does not provide any information about the other data in the secret key, but it does allow the construction of much more efficient side-channel attacks. Yet to the best of our knowledge, there is no kown way to recover $\mathbf{t_0}$ from Dilithium signatures. In this work, we show that each Dilithium signature leaks information on $\mathbf{t_0}$, then we construct an attack that retrieves the vector $\mathbf{t_0}$ from Dilithium signatures. Experimentally, for Dilithium-2, 4 000 000 signatures and 2 hours are sufficient to recover $\mathbf{t}_0$ on a desktop computer.

1 Introduction

Dilithium. Following NIST's Post-Quantum Cryptography competition, the Dilithium signature scheme [BDK+21] has been selected as one of the winners under the name ML-DSA. It belongs to the family of lattice-based signature schemes, and is an application of the Fiat-Shamir with abort [Lyu09] to the Module Learning-With-Errors (MLWE) problem. In general, the public key of such a scheme is an (M)LWE instance, which is to say a (matrix, vector) pair $(\mathbf{A}, \mathbf{t})$ such that there exists two "small" *secret* vectors $\mathbf{s}_1$ and $\mathbf{s}_2$ that satisfy $\mathbf{t} = \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2$. One of the main selling points of Dilithium is its compressed public key. Indeed, the vector $\mathbf{t}$ is split coefficient-wise into a high and a low part, respectively denoted by $\mathbf{t}_1$ and $\mathbf{t}_0$. The Dilithium public key finally corresponds to the seed that was used to generate the matrix $\mathbf{A}$, along with $\mathbf{t}_1$, while $\mathbf{t}_0$ is considered to be part of the secret key. The generation of the signature and its verification must be adapted so that the verifier is able to verify the signature without knowledge of $\mathbf{t}_0$.

On the status of $\mathbf{t_0}$. As stated previously, $\mathbf{t}_0$ is considered as a part of the secret key. However, the security proof of Dilitium [BDK+21] assumes that the

whole vector $\mathbf{t}$ is given in the public key, which means that this compression is not intended as a security measure. Moreover, Lyubashevsky mentions $\mathbf{t}_0$ in a conference given in 2022 [Lyu22]: "$\mathbf{t}_0$ are not given but they are not secret, some informations is leaked with every signature. The security proof assumes that $\mathbf{t}_0$ is public.". This is further emphasized by the NIST draft standard for ML-DSA [NIS23], which states that $\mathbf{t}_0$ "can be reconstructed from a small number of signatures and, therefore, need not be regarded as secret". Unfortunately, it seems that this claim has never been formally studied. While this may seem benign due to the fact that the formal security of Dilithium does not rely on the secrecy of $\mathbf{t}_0$, it seems that its knowledge can be useful in the context of side-channel attacks.

Some papers assume that $\mathbf{t}_0$ is known to the attacker: in [RRB+19] we can read "note that the security analysis of DILITHIUM is done with the assumption that the whole of $\mathbf{t}$ is declared as the public key. In addition to this, some information about $\mathbf{t}_0$ is leaked with every published signature and thus the whole of $\mathbf{t}$ can be reconstructed by just observing several signatures generated using the same secret key", but unfortunately again there is no argument or proof.

Others articles remain conservative and study their attacks in both cases: with or without the knowledge of $\mathbf{t}_0$. For example, in [EAB+23a] the authors state that: "the knowledge of $\mathbf{t}_0$ is not required for the MLWE to RLWE reduction part of our attack [...]. However, it has an impact on the resulting security of the RLWE problem making it harder to solve".

There are even papers that explicitly ask for a clarification of the role of $\mathbf{t}_0$ in the side-channel literature on Dilithium. In particular in [WNGD23]: " the main contribution of this paper is highlighting the possibility of recovering the complete secret vector $\mathbf{s}_1$ from a single trace with a non-negligible probability (9% in our experiments) in the case when $\mathbf{t}_0$ is known. None of the previous attacks on Dilithium can recover the full s1 from fewer than 100 traces. Our results demonstrate the necessity of protecting the secret key of Dilithium from single-trace attacks. They also prompt a reassessment of the role of t0 in the security of Dilithium implementations."

Finally, at least one paper considers that it is unrealistic to assume that a "real" attacker can find $\mathbf{t}_0$ in a side-channel setting. In [RJH+18] we read:" Thus, it might indeed be possible that the whole of $\mathbf{t}$ leaks as part of the signature and observations of sufficiently many signatures might lead to the recovery of the complete LWE instance, $\mathbf{t}$. But again, we expect the number of signatures and the computational effort to be very high, which cannot be expected in a practical SCA setting".

In conclusion, there is no consensus on the role of $\mathbf{t}_0$ in the case of side-channel attacks. As side-channel protection is costly, especially in embedded environments, where industrial constraints are tight, it would be tragic if certain products were not protected against attacks deemed unrealistic because they required knowledge of $\mathbf{t}_0$.

Our contribution. In this article, we study the possibility of recovering $\mathbf{t}_0$ from signatures corresponding to arbitrary messages. In more details, from each signature, we extract inequalities on the coefficients of $\mathbf{t}_0$, until we get a system of linear non-equalities that admit $\mathbf{t}_0$ as its only solution. In order to solve the system, we rely on Linear Programming. The key takeways of our work are the following:

1. A few millions of signatures are needed to reliably recover the value of $\mathbf{t}_0$. Indeed, in our experiments, even using a million signatures, the system admits a huge number of solutions.
2. As a consequence, this results in a very large system of inequalities, which is computationally heavy to solve. We built a more efficient approach that builds a sequence of filtered systems of inequalities that have an increasingly smaller number of solutions until $\mathbf{t}_0$ becomes the unique solution.

Overall, our method is relatively simple yet non-trivial, and allows us to find $\mathbf{t}_0$ in all our experiments, each time in less than 2 hours on a desktop computer. Our result shows two points: the fact that not knowing $\mathbf{t}_0$ does not strengthen Dilithium's security (which is not really surprising) but, more importantly, it shows that $\mathbf{t}_0$ can be found quickly in practice. Therefore assuming that it could be known by a physical attacker is a reasonable assumption.

Outline. This paper is organized as follows. In Section 2, we redefine the basic notions about Dilithium and recall some results from linear programming which will be useful in the rest of this article. In Section 3, we define and motivate the problem we will solve in the rest of the article. In Section 4, we propose an approach based on linear programming tools and results. Finally, Section 5 presents the experimental results obtained for this new attack and a brief discussion of our results.

2 Preliminary requirements

In this section we begin by briefly introducing the notations and main functions used in Dilithium. For a detailed description of Dilitihum, the reader is referred to [BDK+21]. We then review the main definitions and results of linear programming, which will be used in the next section.

2.1 Notations, hints and inequalities

Definition 1 *Let α be an even (resp. odd) integer. We define $r' := r \bmod^{\pm}(\alpha)$ 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 $r'' := r \bmod^{+}(\alpha)$ the unique $0 \leq r'' < \alpha$ such that $r'' = r \mod (\alpha)$.*

Definition 2 *We define $\phi_n = x^n + 1$ with n a power of 2 and q a prime, and introduce the following rings:*

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

Notation 1 *For an integer $l \in \mathbb{N}^*$ and for an element $\mathbf{t}_0 \in \mathcal{R}^l$, we will note $\mathbf{t}_0 = (\mathbf{t}_0^{[1]}, ..., \mathbf{t}_0^{[l]}) \in \mathcal{R}^l$ and $\mathbf{t}_{0,i}^{[j]}$ will be the i-th coefficient of the polynomial $\mathbf{t}_0^{[j]}$.*

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

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

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

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

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

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

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

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

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

Dilithium is a signature scheme based on structured lattices, we will therefore manipulate matrices and vectors of $\mathcal{R}$ or $\mathcal{R}_q$, with the values of n and q fixed at $n = 256$ and $q = 2^{23} - 2^{13} + 1 = 8\,380\,417$ regardless of the security level. In addition, to reduce the size of the public key and to generate the signature Dilithium uses algorithms that splits elements in $\mathbb{Z}_q$. The first and most natural way is to use bit decomposition: for $r \in \mathbb{Z}_q$ and $d \in \mathbb{N}^*$, $r = r_1 2^d + r_0$ where $r_0 = r \bmod^{\pm} 2^d$ and $r_1 = (r - r_0)/2^d$. This is done with the algorithm $\texttt{Power2Round}_q$ defined in Algorithm 1 and is used to reduce the size of the public key.

Since the public key is not "completely" known to the verifier, the signer must add "hints" to the signature to allow its verification. Given $r \in \mathbb{Z}_q$ and a small element $z \in \mathbb{Z}_q$, the verifier must calculate the most significant bits of $z + r$ without knowing z. To do this, the authors have decided to use a slightly different split: for an even α divisor of $q - 1$ and $r \in \mathbb{Z}_q$ they 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. With this simple tweak we can calculate the most significant bits of $r+z$, only with the knowledge of z and a hint bit $h \in \{0, 1\}$. In Algorithm 1 we give the description of the algorithms and recall in Lemma 1 the main property used.

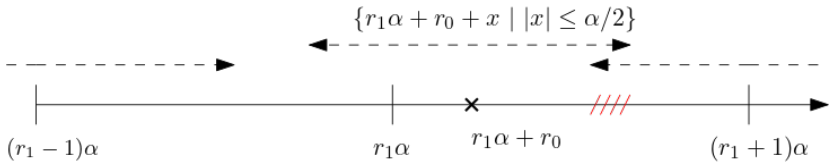

Fig. 1. carry caused by x

Algorithm 1 `Supporting algorithms for Dilithium`

$\texttt{Power2Round}_q(r, d)$:
1: $r = r \bmod^+ q$
2: $r_0 = r \bmod^\pm 2^d$
3: **return** $(r - r_0)/2^d, r_0)$

$\texttt{Decompose}_q(r, \alpha)$:
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)

$\texttt{HighBits}_q(r, \alpha)$:
1: $(r_1, r_0) = \texttt{Decompose}_q(r, \alpha)$
2: **return** r_1

$\texttt{LowBits}_q(r, \alpha)$:
1: $(r_1, r_0) = \texttt{Decompose}_q(r, \alpha)$
2: **return** r_0

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

$\texttt{UseHint}_q(h, r, \alpha)$:
1: $m = (q - 1)/\alpha$
2: $(r_1, r_0) = \texttt{Decompose}_q(r, \alpha)$
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:*

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

2.2 Algorithm description

Key Generation: The key generation algorithm is described in Algorithm 2. Dilithium is based on the Module-LWE problem, a variant of the LWE problem introduced by Regev in [Reg05], which we will not recall here. From some seeds, $\mathbf{A} \in \mathcal{R}_q^{k \times l}$ and $\mathbf{s}_1 \in S_\eta^l$ and $\mathbf{s}_2 \in S_\eta^k$ are generated and then $\mathbf{t} = \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2$ is computed. Two optimisations are made to the public key, which is traditionally

$(\mathbf{A}, \mathbf{t})$, to reduce its size. The first optimisation, the most natural, consists of transmitting only the seed used to generate the matrix $\mathbf{A}$. For the second optimisation, only $\mathbf{t}_1$ (the high part of $\mathbf{t}$ computed with $\texttt{Power2Round}_q$) is considered to be part of the public key. This reduces the size of the public key by half at the cost of adding a few bits at the time of signing, so that the verifier does not need the knowledge of $\mathbf{t}_0$.

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: $\mathbf{A} \in \mathcal{R}_q^{k \times l} := \texttt{ExpandA}(\rho)$
4: $(\mathbf{s}_1, \mathbf{s}_2) \in S_\eta^l \times S_\eta^k := \texttt{ExpandS}(\rho')$
5: $\mathbf{t} := \mathbf{A}\,\mathbf{s}_1 + \mathbf{s}_2$
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: The signature algorithm is described in Algorithm 3. The signer derives a masking vector $\mathbf{y} \in \mathcal{R}_q^l$, from which it calculates $\mathbf{w}_1$, the most significant bits of $\mathbf{w} := \mathbf{A}\,\mathbf{y}$ and then a challenge $c \in \mathcal{R}$ which is a sparse polynomial whose coefficients are in $\{-1, 0, 1\}$. It then calculates $\mathbf{z} := \mathbf{y} + c\,\mathbf{s}_1$, the main part of the signature, which verifies the following equation, used for verification:

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

The verifier then checks that $\mathbf{z}$ does not give information about the secret key; if it does, it starts again by drawing another masking vector. Once $\mathbf{z}$ has passed the tests, we have the following equation:

$$\mathbf{w}_1 = \texttt{HighBits}_q(\mathbf{A}\mathbf{y} - c\mathbf{s}_2,\, 2\,\gamma_2) = \texttt{HighBits}_q(\mathbf{A}\mathbf{z} - c\mathbf{t},\, 2\,\gamma_2).$$

Since $\mathbf{t}_0$ is not known, anyone attempting to verify the signature cannot directly compute $\texttt{HighBits}_q(\mathbf{A}\mathbf{z} - c\mathbf{t},\, 2\,\gamma_2)$. Using the method described in subsection 2.1, the signer adds $\mathbf{h} = \texttt{MakeHint}_q(-c\mathbf{t}_0, \mathbf{A}\mathbf{y} - c\mathbf{s}_2 + c\mathbf{t}_0, 2\gamma_2)$ to allow the verifier to calculate $\texttt{HighBits}_q(\mathbf{A}\mathbf{z} - c\mathbf{t},\, 2\,\gamma_2)$, without the knowledge of $\mathbf{t}_0$. Finally, the signature is composed of the challenge c, $\mathbf{z}$, and the hint vector $\mathbf{h}$.

Algorithm 3 `Sig`

Require: sk, M
Ensure: $\sigma = (\tilde{c}, \mathbf{z}, \mathbf{h})$
$\mathbf{A} \in \mathcal{R}_q^{k \times l} := \texttt{ExpandA}(\rho)$
$\mu \in \{0,1\}^{512} := \mathrm{H}(tr \,||\, M)$
$\kappa := 0$, $(\mathbf{z}, \mathbf{h}) := \perp$
$\rho' \in \{0,1\}^{512} := \mathrm{H}(K \,||\, \mu)$
while $(\mathbf{z}, \mathbf{h}) = \perp$ **do**
  $\mathbf{y} \in \tilde{S}_{\gamma_1}^l := \texttt{ExpandMask}(\rho', \kappa)$
  $\mathbf{w} := \mathbf{A}\,\mathbf{y}$
  $\mathbf{w}_1 = \texttt{HighBits}_q(\mathbf{w}, 2\,\gamma_2)$
  $\tilde{c} \in \{0,1\}^{256} := \mathrm{H}(\mu \,||\, \mathbf{w}_1)$
  $c \in B_\tau := \texttt{SampleInBall}(\tilde{c})$
  $\mathbf{z} := \mathbf{y} + c\,\mathbf{s}_1$
  $\mathbf{r}_0 := \texttt{LowBits}_q(\mathbf{w} - c\mathbf{s}_2, 2\,\gamma_2)$
  if $\|\mathbf{z}\|_\infty \geq \gamma_1 - \beta$ or $\|\mathbf{r}_0\|_\infty \geq \gamma_2 - \beta$ **then**
    $(\mathbf{z}, \mathbf{h}) := \perp$
  else
    $\mathbf{h} := \texttt{MakeHint}_q(-c\mathbf{t}_0, \mathbf{w} - c\mathbf{s}_2 + c\mathbf{t}_0, 2\gamma_2)$
    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: The verification algorithm is described in Algorithm 4. To verify the signature, it is sufficient to reconstruct the matrix $\mathbf{A}$ and the polynomial c on which the signer has commited. Using the vector $\mathbf{h}$ of the signature, we can recalculate $\mathbf{w}_1 = \texttt{HighBits}_q(\mathbf{A}\mathbf{z} - c\mathbf{t},\, 2\,\gamma_2)$, by using $\texttt{UseHint}_q$. Finally, the signature will be accepted if it is possible to reconstruct the correct c from $\mathbf{w}_1$ and if $\mathbf{z}$ meets the security conditions imposed during signature generation.

Algorithm 4 `Ver`

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

Remark 1 *As shown above, in the formal definition of Dilithium,* $\mathbf{t_0}$ *is considered to be secret data even though it reveals nothing about the other polynomials of the secret key. Therefore, in a side channel attack, an attacker cannot use knowledge of* $\mathbf{t}_0$ *to attack a Dilithium implementation. Despite this, a large proportion of papers on side-channel or fault-based attacks against Dilithium [BVC+23] [RRB+18] [RRB+18] [EAB+23b] make the assumption that* $\mathbf{t}_0$ *is known. In the rest of the paper we will show that* $\mathbf{t_0}$ *can indeed be considered as part of the public key, since it can be reconstructed from Dilithium signatures.*

2.3 An overview of Polyhedral Theory

A polyhedron is a set of points verifying a finite number of inequalities, in other words: an instersection of a finite number of half-spaces. We are interested in this geometrical object because in Section 3 we will show that by querying Dilithium signatures generated under the same secret key, we will collect inequalities on the coefficients of the polynomial vector $\mathbf{t}_0$. $\mathbf{t}_0$ will therefore be in a bounded polyhedron, traditionally called a polytope. Obtaining information about this polytope will allow us to find $\mathbf{t}_0$ in Section 4. We refer to [NW88] for general definitions and unproven propositions.

Definition 4 *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 5 *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 6 *Let* $P \subset \mathbb{R}^n$ *be a polytope, we call the diameter of* P *and we note* $diam(P)$ *the quantity:*

$$diam(P) = \max_{p_1, p_2 \in P} ||p_1 - p_2||_\infty$$

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 2 *The definition of diameter and dimension are of particular interest to us because they provide an estimation on the number of elements in a polytope. In our case, we are going to collect inequalities verified by* $\mathbf{t}_0$*, so we will obtain a polytope containing* $\mathbf{t}_0$*. Estimating the dimension and diameter of this polytope allows us to obtain an estimation on the coefficients of* $\mathbf{t}_0$.

2.4 The basics of Linear Programming

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 $m \times 1$ matrix. 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:

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

Remark 3 *Let P be a polytope described by a set of inequalities. Trivialy, finding an $x \in P$ (i.e a point that satisfies all the inequalities that form the description of P) is an (LP) problem, as it can be solved by maximizing any function on P.*

Proposition 1 *Let $P = \{x \in \mathbb{R}^n \mid Ax \leq b\}$ be a polytope, upper-bounding the dimension of P or calculating the diameter of P are two (LP) problems.*

Proof. For $i \in \{1, \ldots, n\}$, by solving the following two (LP) problems:

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

Fig. 2. The 2×256 (IP) problems related to P.

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 P. By solving the same (LP) problems we can also estimate the diameter of P.

Notation 3 *Let $P \subset \mathbb{R}^n$ be a polytope. The procedure for calculating a point by minimizing the null function on P is denoted* `lp_guess`*, and we denote* `calculate_diam` *the procedure which consists in computing the diameter of P.*

3 Problem definition and existing solutions

In the rest of the paper, we study the case of an attacker who tries to recover $\mathbf{t}_0$ based on knowledge of $pk = (\rho, \mathbf{t}_1)$ and a certain number of Dilithium signatures $\{\sigma_i\}_{i \in I}$ signed under the corresponding secret key $sk = (\rho, K, tr, \mathbf{s}_1, \mathbf{s}_2, \mathbf{t}_0)$. In this section we show that, with a very high probability, each Dilithium signature provides information on the coefficients of $\mathbf{t}_0$, in the form of one (or more) inequalities on its coefficients. Naturally, we will try to exploit this leakage of information by using linear programming theory to propose a solution.

3.1 Getting inequalities on $\mathbf{t_0}$ from Dilithium's signatures

Assumption 1 *With overwhelming probability, for $\sigma = (\tilde{c}, \mathbf{z}, \mathbf{h})$ a signature of* `Sig` *the polynomial vector $\mathbf{h}$ has at least one non-zero coefficient.*

Proposition 2 *Let $\sigma = (\tilde{c}, \mathbf{z}, \mathbf{h})$ be a signature of* `Sig`*, under Assumption 1 there exists at least one $j \in \{1, ..., k\}$ and one $i \in \{0, ..., 255\}$ such that:*

- *if $\mathbf{h}_i^{[j]} = 1$ and $\texttt{LowBits}_q(\mathbf{Az} - c\mathbf{t}_1 \cdot 2^d, 2\,\gamma_2)_i^{[j]} > 0$:*

$$(-c\mathbf{t}_0)_i^{[j]} \geq \gamma_2 + \beta + 1 - \texttt{LowBits}_q(\mathbf{Az} - c\mathbf{t}_1 \cdot 2^d, 2\,\gamma_2)_i^{[j]} \geq 0.$$

- *if $\mathbf{h}_i^{[j]} = 1$ and $\texttt{LowBits}_q(\mathbf{Az} - c\mathbf{t}_1 \cdot 2^d, 2\,\gamma_2)_i^{[j]} < 0$:*

$$(-c\mathbf{t}_0)_i^{[j]} \leq -(\gamma_2 + \beta + 1) - \texttt{LowBits}_q(\mathbf{Az} - c\mathbf{t}_1 \cdot 2^d, 2\,\gamma_2)_i^{[j]} \leq 0.$$

In any case, if others $j \in \{1, ..., k\}$ and $i \in \{0, ..., 255\}$ verifiy $\mathbf{h}_i^{[j]} = 0$, then:

$$|(-c\mathbf{t}_0)_i^{[j]} + \texttt{LowBits}_q(\mathbf{Az} - c\mathbf{t}_1 \cdot 2^d, 2\,\gamma_2)_i^{[j]}| \leq (\gamma_2 - \beta + 1).$$

Remark 4 *The integers i and j are given by the values at 1 in the coefficients of the polynomial vector $\mathbf{h}$, which is part of the signature. In addition, $\mathbf{A}$ and $\mathbf{t}_1$ are publicly known and $c, \mathbf{z}$ belong to the signature, so the attacker can calculate $\texttt{LowBits}_q(\mathbf{Az} - c\mathbf{t}_1 \cdot 2^d, 2\,\gamma_2)$. Since γ_2 and β are known parameters, an attacker can calculate the bound of the inequation obtained on $\mathbf{t}_0$. For all zero coefficients of $\mathbf{h}$, we obtain two inequalities on $\mathbf{t}_0$. Although they are numerous, in our experiments we figured out that they provide less information about $\mathbf{t}_0$ than those where $\mathbf{h}_i^{[j]} = 1$, which we will briefly illustrate at the end of the paper. If we had decided to take them into account, we would have recovered at least 500 inequations per signature, which would quickly have become unmanageable. This is why we chose to focus on the inequalities given by the non-zero hints.*

Proof. Let $\sigma = (\tilde{c}, \mathbf{z}, \mathbf{h})$ be a signature of `Sig`, we have:

$$\mathbf{h} := \texttt{MakeHint}_q(-c\mathbf{t}_0, \mathbf{Az} - c\mathbf{t}_1 \cdot 2^d, 2\gamma_2) \text{ and } \mathbf{Az} - c\mathbf{t} = \mathbf{Az} - c\mathbf{t}_1 \cdot 2^d - c\mathbf{t}_0$$

Let $j \in \{1, ..., k\}$ and $i \in \{0, ..., 255\}$ be such that $\mathbf{h}_i^{[j]} = 1$. If $\texttt{LowBits}_q(\mathbf{Az} - c\mathbf{t}_1 \cdot 2^d, 2\,\gamma_2) \geq 0$, then adding $(-c\mathbf{t}_0)_i^{[j]}$ creates a carry in the most significant bits of $(\mathbf{Az} - c\mathbf{t}_1 \cdot 2^d)_i^{[j]}$. Formally, one has

$$\begin{aligned}(\texttt{HighBits}_q(\mathbf{Az} - c\mathbf{t}_1 \cdot 2^d, 2\gamma_2))_i^{[j]} &= (\texttt{HighBits}_q(\mathbf{Az} - c\mathbf{t}_1 \cdot 2^d - c\mathbf{t}_0, 2\gamma_2))_i^{[j]} + 1 \\ &= (\texttt{HighBits}_q(\mathbf{Az} - c\mathbf{t}, 2\gamma_2))_i^{[j]} + 1 \\ &= (\mathbf{w}_1')_i^{[j]} + 1.\end{aligned}$$

Furthermore, since the Dilithium signature is correct, we have:

$$|\mathtt{LowBits}_q(\mathbf{Az} - c\mathbf{t}, 2\gamma_2))_i^{[j]}| = |\mathtt{LowBits}_q(\mathbf{Ay} - c\mathbf{s}_2, 2\gamma_2))_i^{[j]}| < \gamma_2 - \beta.$$

So if $\mathbf{h}_i^{[j]} = 1$ and $\mathtt{LowBits}_q(\mathbf{Az} - c\mathbf{t}_1 \cdot 2^d, 2\,\gamma_2)_i^{[j]} \geq 0$ we have:

$$(-c\mathbf{t}_0)_i^{[j]} \geq \gamma_2 + \beta + 1 - \mathtt{LowBits}_q(\mathbf{Az} - c\mathbf{t}, 2\gamma_2))_i^{[j]} \geq 0.$$

The reasoning above is summarised in Figure 3. In red the impossible values of $(-c\mathbf{t}_0)_i^{[j]}$ according to the value of $\mathbf{h}_i^{[j]}$. In purple, the impossible values of $(-c\mathbf{t}_0)_i^{[j]}$ according to the generation of the signature.

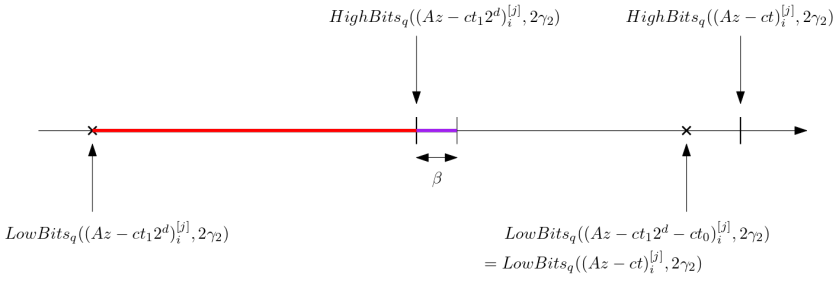

Fig. 3. Idea to obtain inequalities on $\mathbf{t_0}$.

The same arguments can be used to show the second and third inequality.

To measure the frequency with which we obtain an inequation on the coefficient of $\mathbf{t}_0$, we collected 10 000 signatures for an equal number of random messages for 10 random keys, for the three security level of Dilithium. The practical results are summarised in Table 1 below.

NIST Level	II	III	V
Average inequation obtained	62.1	38.1	56.8

Table 1. Average number of inequalities per signature, over 10 000 signatures, for different security levels.

Remark 5 *To visualise the information obtained, we can study the following problem: an attacker knows all the coefficients of* $\mathbf{t}_0^{[1]}$ *except the first two,* $\mathbf{t}_{0,0}^{[1]}$ *and* $\mathbf{t}_{0,1}^{[1]}$*. He queries signatures and obtains inequalities on the two missing coefficients, which can be represented as a point in the set of solutions to the inequalities it has collected.*

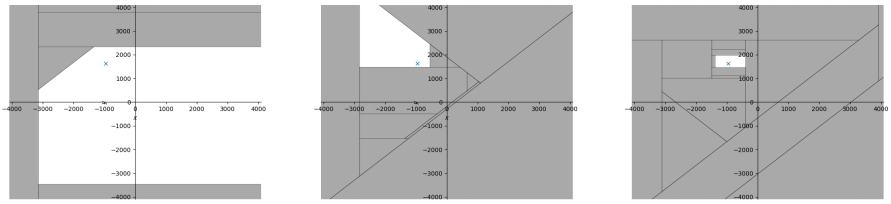

Fig. 4. Polytope containing $(\mathbf{t}_{0,0}^{[1]}, \mathbf{t}_{0,1}^{[1]})$ for $10, 50$ and 100 inequalities.

In Figure 3, the two coefficients the attacker is seeking are $(-961, 1631)$ *and the white part represents the polytope of solutions for* $10, 50$ *and* 100 *collected inequalities. As can be seen in the third image, we have an increasingly complex algebraic description (several dozen inequations, most of which are not useful) of a simple geometric object (a polytope with* 5 *faces).*

4 An attack methodology

The natural approach is to recover enough inequalities on $\mathbf{t}_0$ to form a system (LP) for which it is the unique solution. As we shall see later, this "naive" solution is not possible in our case.

Building the (LP) system After collecting enough signatures, we will have multiple inequalities on the k polynomials of $\mathbf{t}_0$ independently, so we can split the problem into k smaller ones, one for each polynomial of the vector $\mathbf{t}_0$. Again we explain the methodology for a single polynomial of the vector $\mathbf{t}_0 = (\mathbf{t}_0^{[1]}, ..., \mathbf{t}_0^{[k]})$. We select a signature that gives an inequation on $\mathbf{t}_0^{[1]}$. Let $\sigma = (\tilde{c}, \mathbf{z}, \mathbf{h})$ be such a signature, with i such that $\mathbf{h}_i^{[1]} = 1$. Assuming $\texttt{LowBits}_q(\mathbf{Az} - c\mathbf{t}_1 \cdot 2^d, 2\,\gamma_2)_i^{[1]} > 0$, one has

$$(-c\mathbf{t}_0)_i^{[1]} \geq \gamma_2 + \beta + 1 - \texttt{LowBits}_q(\mathbf{Az} - c\mathbf{t}_1 \cdot 2^d, 2\,\gamma_2)_i^{[1]}, \tag{1}$$

$$\sum_{j=0}^{n-1} \mathbf{t}_{0,j}^{[1]}(-cx^j)_1 \geq \gamma_2 + \beta + 1 - \texttt{LowBits}_q(\mathbf{Az} - c\mathbf{t}_1 \cdot 2^d, 2\,\gamma_2)_i^{[1]}. \tag{2}$$

Since the polynomial c is known, σ gives an inequality on the coefficients of $\mathbf{t}_0^{[1]}$. The case of $\texttt{LowBits}_q(\mathbf{Az} - c\mathbf{t}_1 \cdot 2^d, 2\,\gamma_2) < 0$ is treated in the same way. Thus, with these signatures, we can construct two matrices A_+ and A_- and two vectors b_+ and b_- such that $\mathbf{t}_0^{[1]} \in \{x \in \{\gamma_1 + 1, \ldots, \gamma_1\}^n \mid A_+x \geq b_+ \text{ and } A_-x \leq b_-\}$. Each row of one of these matrices representing an inequality collected on $\mathbf{t}_0^{[1]}$. In particular, if we collect enough inequalities for $\mathbf{t}_0^{[1]}$ to be the only solution, we can find $\mathbf{t}_0^{[1]}$ by solving the following (LP) problem of dimension $n = 256$:

$$
\begin{aligned}
&\text{maximize } 0\\
&\text{subject to } A_+ x \geq b_+\\
&\qquad\qquad A_- x \leq b_-\\
&\qquad\qquad x \in [-\gamma_1 + 1, \gamma_1]^n
\end{aligned}
$$

Fig. 5. The (LP) problem related to $\mathbf{t}_0^{[1]}$.

Results using the naive approach. Unless a huge number of inequalities is collected, as can be seen in the Table 2, $\mathbf{t}_0^{[1]}$ will never be the only solution. Nevertheless, we can assume that we know $\mathbf{t}_0$, in order to estimate the size of the polytope containing $\mathbf{t}_0$. In the table below, the attack time takes into account the time required to generate the signatures and the time required to solve the (LP) system associated with $\mathbf{t}_0^{[1]}$, and $\tilde{\mathbf{t}_0}$ denote the polynomial obtained by minimizing the null function on the polytope defined by the inequalities collected. Finally, attack times and average sizes have been calculated for 10 randomly generated keys.

Number of signatures	Number of inequalities	$\|\|\mathbf{t}_0^{[1]} - \tilde{\mathbf{t}_0}\|\|_\infty$	Attack time
1 000	15 511	343	0h0m43s
10 000	158 529	49	0h11m32s
50 000	817 231	8	5h12m17s

Table 2. Attack times and size of the (LP) system on $\mathbf{t}_0^{[1]}$.

Although the attack does not work, it yields interesting results. The method provides us with a point close to $\mathbf{t}_0^{[1]}$ (because $\mathbf{t}_0^{[1]}$ is in the polytope constructed by the (LP) system by definition). Unfortunately, it is not possible to increase the number of inequations endlessly, as the calculation time depends polynomially on the number of inequations.

Remark 6 *If we denote P the polytope obtained on $\mathbf{t}_0^{[1]}$ with a large number of inequations, most of the inequations we collect are not "useful" in the sense that P remains unchanged whether the inequation is taken into account or not. This is illustrated in Figure 3: with* 100 *inequations collected, only* 5 *of them are actually useful in describing the polytope of solutions. By collecting lots of inequations, we get an increasingly complex algebraic description (a growing set of inequations) of a simple geometric object: a polytope with a few faces that approximates $\mathbf{t}_0^{[1]}$. We need a way of selecting "useful" and "useless" inequations to reduce the complexity of solving the (LP) problem associated with $\mathbf{t}_0^{[1]}$.*

4.1 Useful inequalities

In this subsection we assume that we know $\tilde{\mathbf{t}_0} \in \mathcal{R}_q$ and C such that $||\tilde{\mathbf{t}_0} - \mathbf{t}_0^{[1]}||_\infty \leq C$. In other words, $\mathbf{t}_0^{[1]} \in B_\infty(\tilde{\mathbf{t}_0}, C)$. Given an inequation on $\mathbf{t}_0$, we want to determine efficiently if the intersection between $B_\infty(\tilde{\mathbf{t}_0}, C)$ and the set of solutions of the inequation is non-trivial.

Definition 8 *Let $\tilde{\mathbf{t}_0} \in \mathcal{R}_q^k$ and $C \in \mathbb{R}_+$. We say that an inequation on $\mathbf{t}_0^{[1]}$ of the form $\{a^T x - b \geq 0\}$ (resp. $\{a^T x - b \leq 0\}$) is useful according to $\tilde{\mathbf{t}_0}$ and C if and only if:*

$$B_\infty(\tilde{\mathbf{t}_0}, C) \not\subset \{x \in \mathbb{R}^n \mid a^T x - b \geq 0 \} \text{ (resp. } a^T x - b \leq 0)$$

Remark 7 *This definition is very natural and can be illustrated with the following drawing.*

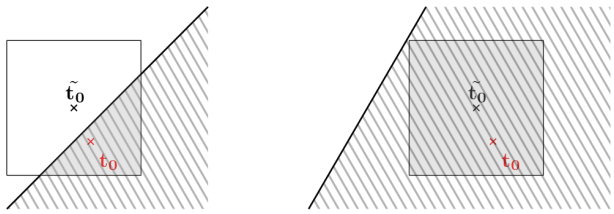

Fig. 6. On the left, a useful inequation. On the right a useless inequation.

Proposition 3 *An inequation on $\mathbf{t}_0^{[1]}$ of the form $\{a^T x - b \geq 0\}$ is useful according to $\tilde{\mathbf{t}_0}$ and C if and only if:*

$$a^T \tilde{\mathbf{t}_0} - C||a^T||_\infty^* < b.$$

An inequation on $\mathbf{t}_0^{[1]}$ of the form $\{a^T x - b \leq 0\}$ is useful according to $\tilde{\mathbf{t}_0}$ and C if and only if:

$$a^T \tilde{\mathbf{t}_0} + C||a^T||_\infty^* > b,$$

where $||.||_\infty^*$ denote the operator norm.

Proof. Let $f : x \mapsto a^T x - b$, then:

$$B_\infty(\tilde{\mathbf{t}_0}, C) \subset \{x \in \mathbb{R}^n \mid f(x) \geq 0\} \iff \inf_{||u||_\infty \leq 1} (f(\tilde{\mathbf{t}_0} + Cu)) \geq 0.$$

In addition,

$$\inf_{||u||_\infty \leq 1} (a^T(\tilde{\mathbf{t}_0} + Cu) - b) = a^T\tilde{\mathbf{t}_0} - C \sup_{||u||_\infty \leq 1} (a^T(-u)) - b$$
$$= a^T\tilde{\mathbf{t}_0} - C||a^T||_\infty^* - b.$$

This concludes the first relation, and the same reasoning can be used to deduce the second result.

Remark 8 *Proposition 3 allows us to calculate efficiently whether an inequation on* $\mathbf{t}_0$ *is useful or not according to* $\tilde{\mathbf{t}_0}$ *and* C*. Finally, it is important to note that the definitions and propositions stated here remain when the infinite norm is replaced by another norm, even if these formulations are not useful for us.*

Notation 4 *We will note* `generate_useful_ineq`$(\delta, \tilde{\mathbf{t}_0}, C)$ *the procedure for generating* δ *useful inequalities according to* $\tilde{\mathbf{t}_0}$ *and* C*.*

4.2 Formal description of the attack

With the different tools we need now defined, the main idea behind the attack strategy can be summed up in one sentence: 'Collect, guess, filter, repeat.' More precisely we will:

- Collect inequalities to obtain a polytope P_0. By definition, $\mathbf{t}_0^{[1]} \in P_0$.
- Calculate (or estimate heuristically) the diameter of P_0 to obtain C and $\tilde{\mathbf{t}_0}$ such that $\mathbf{t}_0^{[1]} \in B_\infty(\tilde{\mathbf{t}_0}, C)$.
- Collect useful inequalities according to $\tilde{\mathbf{t}_0}$ and C to obtain P_1 and by construction $\mathbf{t}_0^{[1]} \in P_1$.
- Repeat until the polytope P verifies $diam(P) \leq 1/2$, in which case $\mathbf{t}_0$ as been recovered.

Algorithm 5 `Recovering` $\mathbf{t}_0^{[1]}$

Ensure: $\mathbf{t}_0^{[1]}$
Require: An inequation step δ

$\tilde{\mathbf{t}_0} = 0$
$C, diam = \gamma_1$
$P = \{-\gamma_1 + 1 \leq x_i \leq \gamma_1\}_{i=1,\ldots,256}$
while $C \geq 1$ **do**
    $\Delta = \delta$
    while $diam > C/2$ **do**
        $P =$ `generate_useful_ineq`$(\Delta, \tilde{\mathbf{t}_0}, C)$
        $diam =$ `calculate_diam`(P)
        $\Delta = 2 \times \Delta$
    $C = C/2$
    $\tilde{\mathbf{t}_0} =$ `round(lp_guess(P))`
return $\tilde{\mathbf{t}_0}$

Proposition 4 *For any $\delta \in \mathbb{N}^*$, Algorithm 5 terminates in a finite number of steps, giving $\mathbf{t}_0^{[1]}$.*

Proof. Let us note $(diam_i)$ the sequence formed by the diameters generated at each step of the of the Algorithm 5. For a sufficiently large Δ, we will always have $diam_i \leq C/2^i$. Thus $(diam_i)$ is a strictly decreasing sequence, so from a certain rank j we have $diam_j < C$, which ensures that the algorithm finishes in a finite number of steps. Finally, at each stage of the algorithm, by choice of C, $\mathbf{t}_0^{[1]} \in B_\infty($`lp_guess`$(P), C)$, so that at the last stage we have $C < 1/2$ and therefore `round(lp_guess`$(P)) = \mathbf{t}_0^{[1]}$.

Remark 9 *Algorithm 5 is useful because it can be proved that it systematically finds $\mathbf{t}_0^{[1]}$. Unfortunately, in practice it is too complex to be used, as each call to the function* `calculate_diam` *requires the solution of 2×256 (LP) problems, each potentially containing several hundred thousand inequalities. Rather than calculating the size of the polytope containing $\mathbf{t}_0^{[1]}$ at each step, we estimate the number of inequations needed to make the size of the corresponding polytope small enough, without having to calculate it explicitly. This is formally described in Algorithm 6.*

Algorithm 6 `Recovering` $\mathbf{t}_0^{[1]}$ `heuristically`

Ensure: A candidate for $\mathbf{t}_0^{[1]}$
Require: An inequation step sequence $(\delta_i)_{i\in\{1,\dots,\log_2(\gamma_1)\}}$.

$\tilde{\mathbf{t}_0} = 0$
$C = \gamma_1$
$P = \{-\gamma_1 + 1 \leq x_i \leq \gamma_1\}_{i=1,\dots,256}$
while $C \geq 1$ **do**
    $P =$ `generate_useful_ineq`$(\delta_i, \tilde{\mathbf{t}_0}, C)$
    $C = C/2$
    $i = i + 1$
    $\tilde{\mathbf{t}_0} =$ `round(lp_guess(P))`
return $\tilde{\mathbf{t}_0}$

Remark 10 *By choosing (δ_i) correctly, this sequence will coincides with the one that would have been given by Algorithm 5. Therefore there exists a sequence (δ_i) of steps such that Algorithm 6 gives $\mathbf{t}_0^{[1]}$. However, if the sequence is not chosen carefully it may be that $\mathbf{t}_0 \notin B_\infty(\tilde{\mathbf{t}_0}, C)$ at some step of Algorithm 6. As Figure 10 shows, if at any stage of the algorithm we have an $\tilde{\mathbf{t}_0}$ and C such that $\mathbf{t}_0^{[1]} \notin B_\infty(\tilde{\mathbf{t}_0}, C)$, when collecting useful inequations according to $\tilde{\mathbf{t}_0}$ and C can lead to a point which deviates from $\mathbf{t}_0^{[1]}$, or even worse: an (LP) system without solution.*

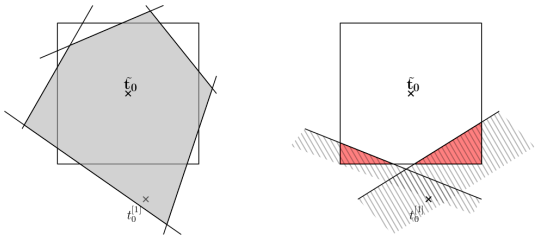

Fig. 7. (LP) system for a poorly choosen (δ_i).

5 Experimental results

For now, we have focused our results on finding $\mathbf{t}_0^{[1]}$ (the first polynomial of $\mathbf{t}_0$) for the first 10 keys in the KAT file. We have used the reference implementation of Dilithium [DKL+22]to generate signatures and to solve the (LP) problems linked to $\mathbf{t}_0^{[1]}$ we have decided to use lp_solve [MB04], a free linear programming solver in C. Our attack method applies independently of lp_solve and any other solver could have been chosen. All the tests and results presented in this section

were carried out on a laptop computer equipped with an Intel(R) Core(TM) i7-10850H 2.70GHz CPU.

Attack results: After several heuristic tests, we decided to choose $(\delta_i) = (50\,000)$. In other words, at each step we collect 50 000 inequalities on the coefficients of $\mathbf{t}_0^{[1]}$ before solving the associated (LP) problem. Table 4 shows the results obtained for the first 10 KAT keys.

Signatures	inequalities selected	Recovery probability	Average time	Median time
3 957 304	$14 \times 50\,000$	1	1h 39min 57sec	1h 29min 54sec

Table 3. Average results of the attack on $\mathbf{t}_0^{[1]}$

Remark 11 *At each stage of the Algorithm 6, we are increasingly selective about the inequalities we keep, so we need more and more signatures to obtain the* 50 000 *inequalities we require.*

If we had sought to find $\mathbf{t}_0^{[1]}$ from a single (LP) system, it would have contained $16 \times 4\,000\,000 = 64\,000\,000$ signatures, resulting in a huge (LP) system which is much more costly to solve than our sequence of small systems. Indeed, thanks to our natural definitions of "useful" inequations, we were able to find an equivalent representation of the polytope containing $\mathbf{t}_0^{[1]}$ with only 50 000 inequations. For greater clarity, we detail the results of the attack calculation for the first key of the KAT in Table 4. For this table only, the knowledge of $\mathbf{t}_0$ was used to illustrate the correctness of Algorithm 6. In Table 4, Time includes time to generate signatures as well as time to solve the problem (LP). With our choice of parameters, this attack time is largely dominated by signature collection time.

Attack improvements and discussions: There are several ways to optimize our attack. We tested to take into account the inequalities obtained by the null coefficients of $\mathbf{h}$. While it leads to a worse guess, once we start to filter inequations, it allows us to reduce the required signatures needed to find $\mathbf{t}_0$. To compare it with the method we have chosen to present, we have detailed its results in Table 5 for the key 0 of the KAT. Another way to reduce the number of signatures required would be to store the signatures generated at each stage of the algorithm, since the same inequation can be useful for different C. Both optimizations should reduce the numbers of queries by a factor 4.

For the moment, only the first polynomial of $\mathbf{t}_0$, $\mathbf{t}_0^{[1]}$, has been found, but since the inequations are evenly distributed between the different polynomials of $\mathbf{t}_0$, finding the whole vector will not require any more signatures. We have also chosen to focus on Dilithium level 2 security, but the theory presented here remains unchanged, and we expect the attack to work in the same way. We will

present the results discussed here in more details in an extended version of the article.

Round	C_i	Inequalities selected	Signatures	$\|\|\mathbf{t}_0^{[1]} - \tilde{\mathbf{t}_0}\|\|_\infty$	Time
1	4096	50 000	3 3302	149	3m55s
2	2048	50 000	3 254	164	3m54s
3	1024	50 000	3 275	147	3m34s
4	512	50 000	3 619	135	3m32s
5	256	50 000	5 169	131	3m38s
6	128	50 000	8 899	48	3m30s
7	64	50 000	16 449	29	3m31s
8	32	50 000	31 777	15	3m45s
9	16	50 000	62 507	11	3m49s
10	8	50 000	126 060	4	5m8s
11	4	50 000	249 139	2	6m20s
12	2	50 000	501 050	2	9m21s
13	1	50 000	981 021	1	15m7s
14	0.5	50 000	1 971 864	0	26m12s
Total	-	$14 \times 50\,000$	3 967 385	-	1h35m16s

Table 4. Detailed results of the attack on the first KAT key.

Round	C_i	Inequalities selected	Signatures	Inequalities-0	Inequalities-1	$\|\|\mathbf{t}_0^{[1]} - \tilde{\mathbf{t}_0}\|\|_\infty$	Time
1	4096	50 000	117	48 394	1 913	1 273	4m0s
2	2048	50 000	233	46 440	3 809	519	3m11s
3	1024	50 000	468	43 067	7 685	237	23m59s
4	512	50 000	937	37 146	13 688	142	3m8s
5	256	50 000	1 874	32 002	18 593	76	32m55s
6	128	50 000	3 741	28 844	21 641	33	2m58s
7	64	50 000	7 518	27 144	23 538	19	3m0s
8	32	50 000	14 992	26 208	24 134	11	3m26s
9	16	50 000	29 951	25 658	24 737	5	3m37s
10	8	50 000	59 919	25 543	24 655	3	4m13s
11	4	50 000	120 361	25 530	24 970	2	5m0s
12	2	50 000	242 273	25 522	25 018	1	7m57s
13	1	50 000	485 922	25 321	25 344	1	12m21s
14	0.5	50 000	988 074	25 053	25 742	0	21m7s
Total	-	$14 \times 50\,000$	1 956 380	441 872	265 467	-	1h19m52s

Table 5. Detailed results with the alternative method (using inequalities from zero and non-zero hints), on the first KAT key.

References

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.

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.

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.

EAB+23a. 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. *IACR TCHES*, 2023(4):262–286, 2023.

EAB+23b. 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.

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

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

Lyu22. Vadim Lyubashevsky. Nist conference. NIST website, 2022. https://www.nist.gov/video/fourth-pqc-standardization-conference-virtual-day-1-part-1.

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

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.

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

Reg05. Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *Proceedings of the Thirty-Seventh Annual ACM Symposium on Theory of Computing*, STOC '05, page 84–93, New York, NY, USA, 2005. Association for Computing Machinery.

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, Paper 2018/821, 2018. https://eprint.iacr.org/2018/821.

RRB+18. 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. Cryptology ePrint Archive, Paper 2018/211, 2018. https://eprint.iacr.org/2018/211.

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, *COSADE 2019*, volume 11421 of *LNCS*, pages 232–250. Springer, Heidelberg, April 2019.

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.