Uncompressing Dilithium's public key

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

[1] Thales, France
[2] UVSQ, France
[3] INRIA, LIP6, France

Summary

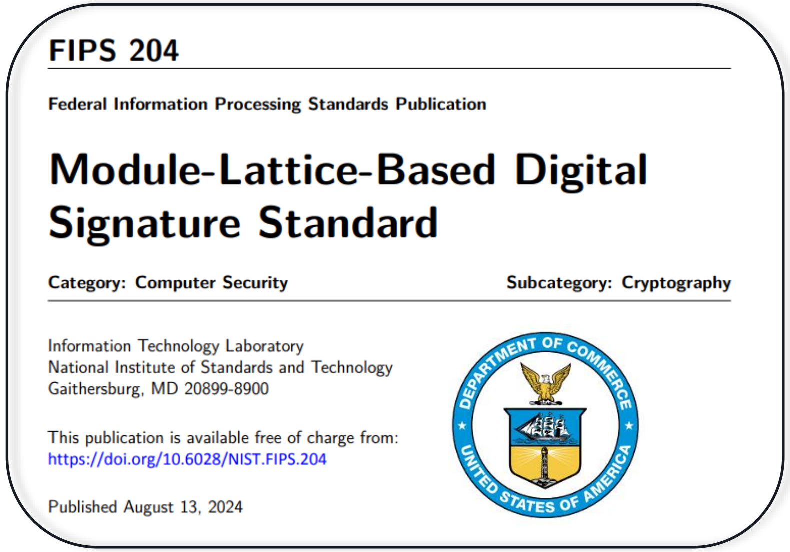
Context

www.thalesgroup.com

Context

Dilithium is a signature algorithm recently standardized by NIST under the name ML-DSA.

Dilithium is recommended for computing quantum-secure signatures in most use cases.



FIPS 204

Federal Information Processing Standards Publication

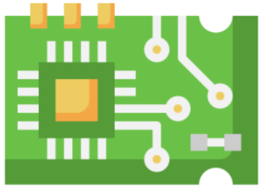
Module-Lattice-Based Digital Signature Standard

Category: Computer Security | Subcategory: Cryptography

Information Technology Laboratory
National Institute of Standards and Technology
Gaithersburg, MD 20899-8900

This publication is available free of charge from:
https://doi.org/10.6028/NIST.FIPS.204

Published August 13, 2024

$m \rightarrow$  $\rightarrow \sigma_m$

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.

Dilithium in details

www.thalesgroup.com

Dilithium in details

Dilithium uses two rings:

$$\mathcal{R} = \mathbb{Z}[x]/(x^n + 1) \qquad \mathcal{R}_q = \mathbb{Z}_q[x]/(x^n + 1)$$

with: $n = 256$ and $q = 8380417$.

Algorithm KeyGen

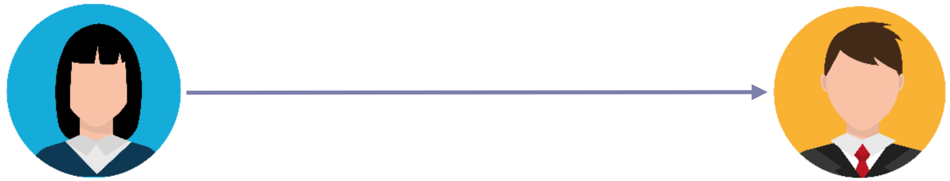
Ensure: (pk, sk)

$\mathbf{A} \leftarrow \mathcal{R}_q^{k \times l}$

$(\mathbf{s}_1, \mathbf{s}_2) \leftarrow S_\eta^l \times S_\eta^k$

$\mathbf{t} := \mathbf{A}\,\mathbf{s}_1 + \mathbf{s}_2$

return $pk = (\mathbf{A}, \mathbf{t}),\ sk = (\mathbf{A}, \mathbf{t}, \mathbf{s}_1, \mathbf{s}_2)$



(A, t, s_1, s_2) (A, t)

Dilithium in details

Dilithium uses two rings:

$$\mathcal{R} = \mathbb{Z}[x]/(x^n + 1) \qquad \mathcal{R}_q = \mathbb{Z}_q[x]/(x^n + 1)$$

with: $n = 256$ and $q = 8380417$.

Algorithm KeyGen

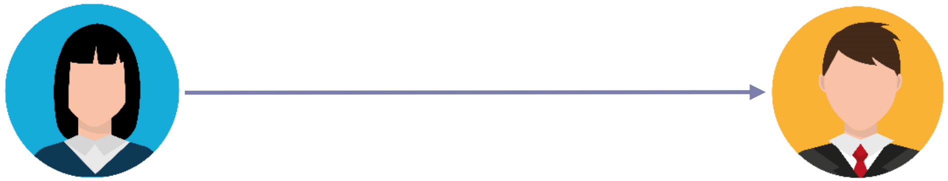
Ensure: (pk, sk)

1: $\mathbf{A} \leftarrow \mathcal{R}_q^{k \times l}$

2: $(\mathbf{s}_1, \mathbf{s}_2) \leftarrow S_\eta^l \times S_\eta^k$

3: $\mathbf{t} := \mathbf{A}\,\mathbf{s}_1 + \mathbf{s}_2$

4: **return** $pk = (\mathbf{A}, \mathbf{t}),\ sk = (\mathbf{A}, \mathbf{t}, \mathbf{s}_1, \mathbf{s}_2)$



(A, t, s_1, s_2) (A, t)

α an even integer which divides $q - 1$ and:

$$r = r_1\alpha + r_0 \; with \; r_0 = r \; mod^{\pm}(\alpha) \; and \; r_1 = \frac{r - r_0}{\alpha}$$

Possible values of r_0: $\left\{-\frac{\alpha}{2} + 1, \dots, 0, \dots, \frac{\alpha}{2}\right\}$

Possible values of $r_1\alpha$: $\{0, \alpha, 2\alpha, \dots, q - 1\}$

One note:

$HighBits_q(r, \alpha) = r_1$ and $LowBits_q(r, \alpha) = r_0$

Dilithium in details

Dilithium uses two rings:

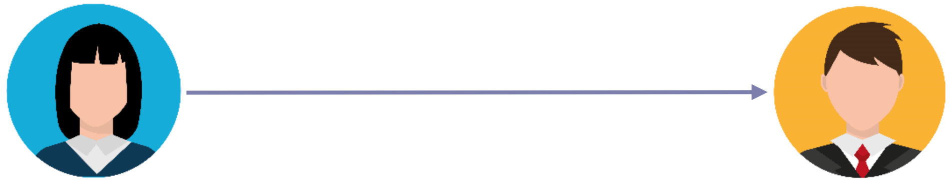
$$\mathcal{R} = \mathbb{Z}[x]/(x^n + 1) \qquad \mathcal{R}_q = \mathbb{Z}_q[x]/(x^n + 1)$$

with: $n = 256$ and $q = 8380417$.

Algorithm KeyGen

Ensure: (pk, sk)

1: $\mathbf{A} \leftarrow \mathcal{R}_q^{k \times l}$
2: $(\mathbf{s}_1, \mathbf{s}_2) \leftarrow S_\eta^l \times S_\eta^k$
3: $\mathbf{t} := \mathbf{A}\,\mathbf{s}_1 + \mathbf{s}_2$
4: **return** $pk = (\mathbf{A}, \mathbf{t})$, $sk = (\mathbf{A}, \mathbf{t}, \mathbf{s}_1, \mathbf{s}_2)$



(A, t, s_1, s_2) (A, t)

$$r = HighBits_q(r, \alpha) \times \alpha + LowBits_q(r, \alpha)$$

$$P = (P_1, \dots, P_l)$$

$$P_i = \sum p_i x^i$$

$$HighBits_q(P_i, \alpha) = \sum HighBits_q(p_i, \alpha) x^i$$

$$HighBits_q(P, \alpha) = \left(HighBits_q(P_1, \alpha), \dots, HighBits_q(P_l, \alpha)\right)$$

Dilithium in details

Algorithm Sig

Require: sk, M

Ensure: $\sigma = (c, \mathbf{z})$

```
z = ⊥
while z = ⊥ do
    y ← S̃^l_{γ1}
    w1 := HighBits(Ay, 2γ2)
    c ∈ B_τ := H(M||w1)
    z := y + c s1
    if ||z||∞ ≥ γ1 − β or LowBits(Ay − cs2, 2γ2)||∞ ≥ γ2 − β then
        z := ⊥
    end if
end while
return σ = (c, z)
```

$(M, \sigma = (c, \boldsymbol{z}))$

(A, t, s_1, s_2) (A, t)

Alice draws a polynomial vector at random:

$$\boldsymbol{y} \in_R \boldsymbol{R}^{\boldsymbol{l}}, \ \left|\left|\boldsymbol{y}\right|\right|_\infty \le \boldsymbol{\gamma_1}.$$

She computes a random challenge that depends on the message:

$$\boldsymbol{c} = \boldsymbol{H}\left(\boldsymbol{M} \,||\, \boldsymbol{HighBits_q}(\boldsymbol{Ay}, 2\boldsymbol{\gamma_2})\right).$$

She provides a response to the challenge:

$$\boldsymbol{z} = \boldsymbol{y} + \boldsymbol{cs_1}.$$

By definition of $\boldsymbol{z}$:

$$\boldsymbol{Az} - \boldsymbol{ct} = \boldsymbol{Ay} - \boldsymbol{cs_2}.$$

$\boldsymbol{z}$ is chosen such that:

Rejection

$$\boldsymbol{HighBits_q}(\boldsymbol{Ay}, 2\boldsymbol{\gamma_2}) = \boldsymbol{HighBits_q}(\boldsymbol{Ay} - \boldsymbol{cs_2}, 2\boldsymbol{\gamma_2}).$$

Dilithium in details

Algorithm Sig

Require: sk, M

Ensure: $\sigma = (c, \mathbf{z})$

$\mathbf{z} = \perp$

while $\mathbf{z} = \perp$ **do**

  $\mathbf{y} \leftarrow \tilde{S}^l_{\gamma_1}$

  $\mathbf{w}_1 := \texttt{HighBits}(\mathbf{A}\mathbf{y}, 2\gamma_2)$

  $c \in B_\tau := H(M || \mathbf{w}_1)$

  $\mathbf{z} := \mathbf{y} + c\,\mathbf{s}_1$

  if $||\mathbf{z}||_\infty \geq \gamma_1 - \beta$ or $\texttt{LowBits}(\mathbf{A}\mathbf{y} - c\mathbf{s}_2, 2\gamma_2)||_\infty \geq \gamma_2 - \beta$ **then**

    $\mathbf{z} := \perp$

  end if

end while

return $\sigma = (c, \mathbf{z})$

$(M, \sigma = (c, \boldsymbol{z}))$

(A, t, s_1, s_2)

(A, t)

By definition of z:

$$Az - ct = Ay - cs_2.$$

Algorithm 1 Ver

$\mathbf{w}_1' := \texttt{HighBits}(\mathbf{A}\mathbf{z} - c\mathbf{t}, 2\gamma_2)$

Accept if $||\mathbf{z}||_\infty \leq \gamma_1 - \beta$ **and** $c = H(M || \mathbf{w}_1')$

Bob can recompute w_1:

$$\begin{aligned} \mathbf{w}_1 &= HighBits_q(Ay, 2\gamma_2) \\ &= HighBits_q(Ay - cs_2, 2\gamma_2) \\ &= HighBits_q(Az - ct, 2\gamma_2) \\ &= w_1' \end{aligned}$$

Dilithium in details

Dilithium's public key is compressed:

$$t = t_1 \times 2^d + t_0.$$

The least significant bits of coefficients of t are not given, verification is no longer possible:

Algorithm Ver
1: $\mathbf{w}'_1 := \texttt{HighBits}(\mathbf{Az} - c\mathbf{t}, 2\gamma_2)$
2: **Accept if** $\|\|\mathbf{z}\|\|_\infty \leq \gamma_1 - \beta$ **and** $c = H(M\|\|\mathbf{w}'_1)$

Bob can only compute:

$$HighBits_q(Az - ct_1\, 2^d, 2\gamma_2) \neq HighBits_q(Az - ct_1\, 2^d - ct_0, 2\gamma_2).$$

Dilithium in details

Dilithium's public key is compressed:

$$t = t_1 \times 2^d + t_0.$$

The least significant bits of coefficients of t are not given, verification is no longer possible:

Algorithm Ver

1: $\mathbf{w}'_1 := \texttt{HighBits}(\mathbf{Az} - c\mathbf{t}, 2\gamma_2)$
2: **Accept if** $||\mathbf{z}||_\infty \leq \gamma_1 - \beta$ **and** $c = H(M||\mathbf{w}'_1)$

Bob can only compute:

$$HighBits_q(Az - ct_1\, 2^d, 2\gamma_2) \neq HighBits_q(Az - ct_1\, 2^d - ct_0, 2\gamma_2).$$

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:*

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

The real Dilithium

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:*

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

Algorithm KeyGen

Ensure: (pk, sk)

1: $\mathbf{A} \leftarrow \mathcal{R}_q^{k \times l}$
2: $(\mathbf{s}_1, \mathbf{s}_2) \leftarrow S_\eta^l \times S_\eta^k$
3: $\mathbf{t} := \mathbf{A}\,\mathbf{s}_1 + \mathbf{s}_2$
4: **return** $pk = (\mathbf{A}, \mathbf{t})$, $sk = (\mathbf{A}, \mathbf{t}, \mathbf{s}_1, \mathbf{s}_2)$

→

Algorithm $\texttt{KeyGen}$

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** $pk = (\rho, \mathbf{t}_1)$, $sk = (\rho, K, tr, \mathbf{s}_1, \mathbf{s}_2, \mathbf{t}_0)$

The real Dilithium

Algorithm Sig

Require: sk, M

Ensure: $\sigma = (c, \mathbf{z})$

$\mathbf{z} = \perp$
while $\mathbf{z} = \perp$ **do**
  $\mathbf{y} \leftarrow \tilde{S}^l_{\gamma_1}$
  $\mathbf{w}_1 := \texttt{HighBits}(\mathbf{A}\mathbf{y}, 2\gamma_2)$
  $c \in B_\tau := H(M || \mathbf{w}_1)$
  $\mathbf{z} := \mathbf{y} + c\,\mathbf{s}_1$
  if $\|\mathbf{z}\|_\infty \geq \gamma_1 - \beta$ or $\texttt{LowBits}(\mathbf{A}\mathbf{y} - c\mathbf{s}_2, 2\gamma_2)||_\infty \geq \gamma_2 - \beta$ **then**
    $\mathbf{z} := \perp$
  end if
end while
return $\sigma = (c, \mathbf{z})$

Algorithm 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**
  $\mathbf{y} \in \tilde{S}^l_{\gamma_1} := \texttt{ExpandMask}(\rho', \kappa)$
  $\mathbf{w} := \mathbf{A}\,\mathbf{y}$
  $\mathbf{w}_1 = \texttt{HighBits}_q(\mathbf{w}, 2\,\gamma_2)$
  $\tilde{c} \in \{0,1\}^{256} := \mathrm{H}(\mu \,||\, \mathbf{w}_1)$
  $c \in B_\tau := \texttt{SampleInBall}(\tilde{c})$
  $\mathbf{z} := \mathbf{y} + c\,\mathbf{s}_1$
  $\mathbf{r}_0 := \texttt{LowBits}_q(\mathbf{w} - c\mathbf{s}_2, 2\,\gamma_2)$
  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 $\sigma = (\tilde{c}, \mathbf{z}, \mathbf{h})$

The real Dilithium

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:*

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

Algorithm 1 Ver

1: $\mathbf{w}_1' := \texttt{HighBits}(\mathbf{Az} - c\mathbf{t}, 2\gamma_2)$

2: **Accept if** $||\mathbf{z}||_\infty \leq \gamma_1 - \beta$ **and** $c = H(M||\mathbf{w}_1')$

→

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 $[\![|\mathbf{h}|_{\mathbf{h}_j = 1} \leq \omega]\!]$

Natural question

www.thalesgroup.com

The real Dilithium: What status for t_0?

Formally, t_0 is part of the private key, but it is a performance optimization. The security proof considers it public, but what about side-channel attacks?

The real Dilithium: What status for t_0?

Formally, t_0 is part of the private key, but it is a performance optimization. The security proof considers it public, but what about side-channel attacks?

NIST

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.

The real Dilithium: What status for t_0?

Formally, t_0 is part of the private key, but it is a performance optimization. The security proof considers it public, but what about side-channel attacks?

NIST

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.

RRB+19

There is a subtle but considerable difference with respect to publicly revealed LWE instances in the Dilithium scheme. The public key reveals only $\mathbf{t}_1$, the d higher order bits of $\mathbf{t}$, while $\mathbf{t}_0$ (the lower order component) is part of the secret key. Even on ensuring nonce-reuse, we would not be able to trivially solve for the secret $\mathbf{s}$ from the faulty public key. But, 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 [1]. Thus it

The real Dilithium: What status for t_0?

Formally, t_0 is part of the private key, but it is a performance optimization. The security proof considers it public, but what about side-channel attacks?

NIST

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.

RRB+19

There is a subtle but considerable difference with respect to publicly revealed LWE instances in the Dilithium scheme. The public key reveals only $\mathbf{t}_1$, the d higher order bits of $\mathbf{t}$, while $\mathbf{t}_0$ (the lower order component) is part of the secret key. Even on ensuring nonce-reuse, we would not be able to trivially solve for the secret $\mathbf{s}$ from the faulty public key. But, 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 [1]. Thus it

References

1. Suppressed for blind review

The real Dilithium: What status for t_0?

Formally, t_0 is part of the private key, but it is a performance optimization. The security proof considers it public, but what about side-channel attacks?

EAB+23

Dilithium [BBK16, RJH+19]. In our attack, the knowledge of $\mathbf{t_0}$ is not required for the MLWE to RLWE reduction part of our attack as $\mathbf{t_0}$ can be embedded into the additive noise vector (moving $\mathbf{t_0}$ to the right part of Equation 6). However, it has an impact on the resulting security of the RLWE problem making it harder to solve. As it is unclear if $\mathbf{t_0}$ must be considered secret or public (it has been hinted that $\mathbf{t_0}$ can be recovered from enough signatures in [Lyu22, RJH+18, RRB+19]), we take a worst-case approach for the rest of this work. If not specified in the following sections, the full $\mathbf{t}$ is assumed to be public. In Subsection 5.3, we derive the impact of fully secret $\mathbf{t_0}$ on the complexity of the (reduced) RLWE instance. We hope that this approach gives a complete view to the reader about the applicability of the attack to Dilithium.

The real Dilithium: What status for t_0?

Formally, t_0 is part of the private key, but it is a performance optimization. The security proof considers it public, but what about side-channel attacks?

EAB+23

Dilithium [BBK16, RJH+19]. In our attack, the knowledge of $\mathbf{t_0}$ is not required for the MLWE to RLWE reduction part of our attack as $\mathbf{t_0}$ can be embedded into the additive noise vector (moving $\mathbf{t_0}$ to the right part of Equation 6). However, it has an impact on the resulting security of the RLWE problem making it harder to solve. As it is unclear if $\mathbf{t_0}$ must be considered secret or public (it has been hinted that $\mathbf{t_0}$ can be recovered from enough signatures in [Lyu22, RJH+18, RRB+19]), we take a worst-case approach for the rest of this work. If not specified in the following sections, the full $\mathbf{t}$ is assumed to be public. In Subsection 5.3, we derive the impact of fully secret $\mathbf{t_0}$ on the complexity of the (reduced) RLWE instance. We hope that this approach gives a complete view to the reader about the applicability of the attack to Dilithium.

RJH+18

Thus, it might indeed be possible that the whole of $\bar{t}$ leaks as part of the signature and observations of sufficiently many signatures might lead to the recovery of the complete LWE instance, $\bar{t}$. 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.

The real Dilithium: What status for t_0?

Formally, t_0 is part of the private key, but it is a performance optimization. The security proof considers it public, but what about side-channel attacks?

EAB+23

Dilithium [BBK16, RJH+19]. In our attack, the knowledge of $\mathbf{t_0}$ is not required for the MLWE to RLWE reduction part of our attack as $\mathbf{t_0}$ can be embedded into the additive noise vector (moving $\mathbf{t_0}$ to the right part of Equation 6). However, it has an impact on the resulting security of the RLWE problem making it harder to solve. As it is unclear if $\mathbf{t_0}$ must be considered secret or public (it has been hinted that $\mathbf{t_0}$ can be recovered from enough signatures in [Lyu22, RJH+18, RRB+19]), we take a worst-case approach for the rest of this work. If not specified in the following sections, the full $\mathbf{t}$ is assumed to be public. In Subsection 5.3, we derive the impact of fully secret $\mathbf{t_0}$ on the complexity of the (reduced) RLWE instance. We hope that this approach gives a complete view to the reader about the applicability of the attack to Dilithium.

WNGD23

ferent from profiling device is non-negligible (9%). The success rate approaches 100% if multiple traces are available for the attack. Our results demonstrate the necessity of protecting the secret key of CRYSTALS-Dilithium from single-trace attacks and call for a reassessment of the role of compression of the public key vector $\mathbf{t}$ in the security of CRYSTALS-Dilithium implementations.

RJH+18

Thus, it might indeed be possible that the whole of $\bar{t}$ leaks as part of the signature and observations of sufficiently many signatures might lead to the recovery of the complete LWE instance, $\bar{t}$. 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.

The real Dilithium: What status for t_0?

Formally, t_0 is part of the private key, but it is a performance optimization. The security proof considers it public, but what about side-channel attacks?

EAB+23

Dilithium [BBK16, RJH+19]. In our attack, the knowledge of $\mathbf{t_0}$ is not required for the MLWE to RLWE reduction part of our attack as $\mathbf{t_0}$ can be embedded into the additive noise vector (moving $\mathbf{t_0}$ to the right part of Equation 6). However, it has an impact on the resulting security of the RLWE problem making it harder to solve. As it is unclear if $\mathbf{t_0}$ must be considered secret or public (it has been hinted that $\mathbf{t_0}$ can be recovered from enough signatures in [Lyu22, RJH+18, RRB+19]), we take a worst-case approach for the rest of this work. If not specified in the following sections, the full $\mathbf{t}$ is assumed to be public. In Subsection 5.3, we derive the impact of fully secret $\mathbf{t_0}$ on the complexity of the (reduced) RLWE instance. We hope that this approach gives a complete view to the reader about the applicability of the attack to Dilithium.

WNGD23

ferent from profiling device is non-negligible (9%). The success rate approaches 100% if multiple traces are available for the attack. Our results demonstrate the necessity of protecting the secret key of CRYSTALS-Dilithium from single-trace attacks and call for a reassessment of the role of compression of the public key vector $\mathbf{t}$ in the security of CRYSTALS-Dilithium implementations.

RJH+18

Thus, it might indeed be possible that the whole of $\bar{t}$ leaks as part of the signature and observations of sufficiently many signatures might lead to the recovery of the complete LWE instance, $\bar{t}$. 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 the context of side-channel attacks, the role of t_0 is unclear

Can t_0 be used or not?

Attack methodology

How to recover t_0?

Each Dilithium signature provides inequalities on the coefficients of t_0.

Proposition 2 *Let* $j \in \{0, \ldots, k-1\}$ *and* $i \in \{0, \ldots, 255\}$ *and* $\sigma = (\tilde{c}, \mathbf{z}, \mathbf{h})$ *be a signature of* $\mathtt{Sig}$.

– *If* $\mathbf{h}_i^{[j]} = 0$:

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

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

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

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

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

How to recover t_0?

Each Dilithium signature provides inequalities on the coefficients of t_0.

Proposition 2 *Let* $j \in \{0, \ldots, k-1\}$ *and* $i \in \{0, \ldots, 255\}$ *and* $\sigma = (\tilde{c}, \mathbf{z}, \mathbf{h})$ *be a signature of* Sig.

- *If* $\mathbf{h}_i^{[j]} = 0$:

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

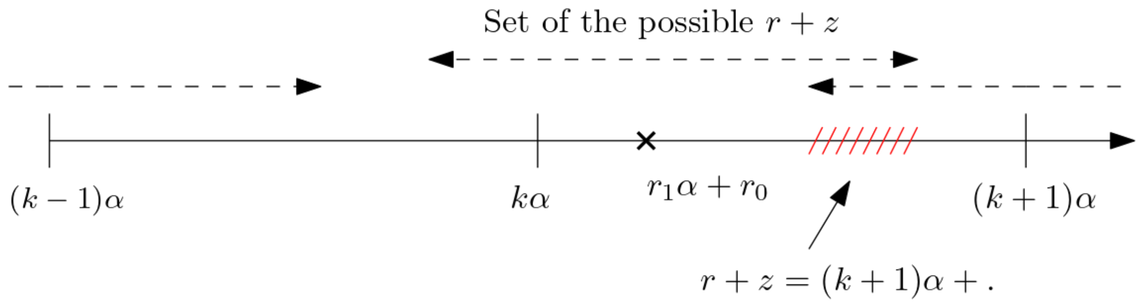
- *If* $\mathbf{h}_i^{[j]} = 1$ *and* $\mathtt{LowBits}_q(\mathbf{Az} - c\mathbf{t}_1 \cdot 2^d, 2\gamma_2)_i^{[j]} > 0$:

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

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

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

Proof (idea) :



The value of h provides information on the size of the polynomial ct_0.

$$HighBits_q(Az - ct_1\, 2^d, 2\gamma_2) \overset{?}{\neq} HighBits_q(Az - ct_1\, 2^d - ct_0, 2\gamma_2)$$

How to recover t_0?

Naive method: We retrieve inequalities and solve them using an LP solver.

Number of signatures	Number of inequalities	$\|\|\mathbf{t}_0^{[0]} - \tilde{\mathbf{t}}_0^{[0]}\|\|_\infty$	Attack time
24	$9\,953 + 389$	$5\,649$	0h0m23s
117	$48\,456 + 1\,915$	$1\,031$	0h3m52s
583	$241\,541 + 9\,378$	247	1h55m47s

Table 4. Attack times and size of the (LP) system on $\mathbf{t}_0$.

How to recover t_0?

Naive method: **We retrieve inequalities and solve them using an LP solver.**

Number of signatures	Number of inequalities	$\|\|\mathbf{t}_0^{[0]} - \tilde{\mathbf{t}}_0^{[0]}\|\|_\infty$	Attack time
24	$9\,953 + 389$	$5\,649$	0h0m23s
117	$48\,456 + 1\,915$	$1\,031$	0h3m52s
583	$241\,541 + 9\,378$	247	1h55m47s

Table 4. Attack times and size of the (LP) system on $\mathbf{t}_0$.

First problem: **There are far too many inequalities per signature.**

NIST Level	II	III	V
Average inequation obtained	$1\,922 + 63$	$2\,996 + 38$	$3\,984 + 56$

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

How to recover t_0?

Naive method: **We retrieve inequalities and solve them using an LP solver.**

Number of signatures	Number of inequalities	$\|\|\mathbf{t}_0^{[0]} - \tilde{\mathbf{t}}_0^{[0]}\|\|_\infty$	Attack time
24	$9\,953 + 389$	$5\,649$	0h0m23s
117	$48\,456 + 1\,915$	$1\,031$	0h3m52s
583	$241\,541 + 9\,378$	247	1h55m47s

Table 4. Attack times and size of the (LP) system on $\mathbf{t}_0$.

Second problem: **Most of the inequalities collected are useless.**

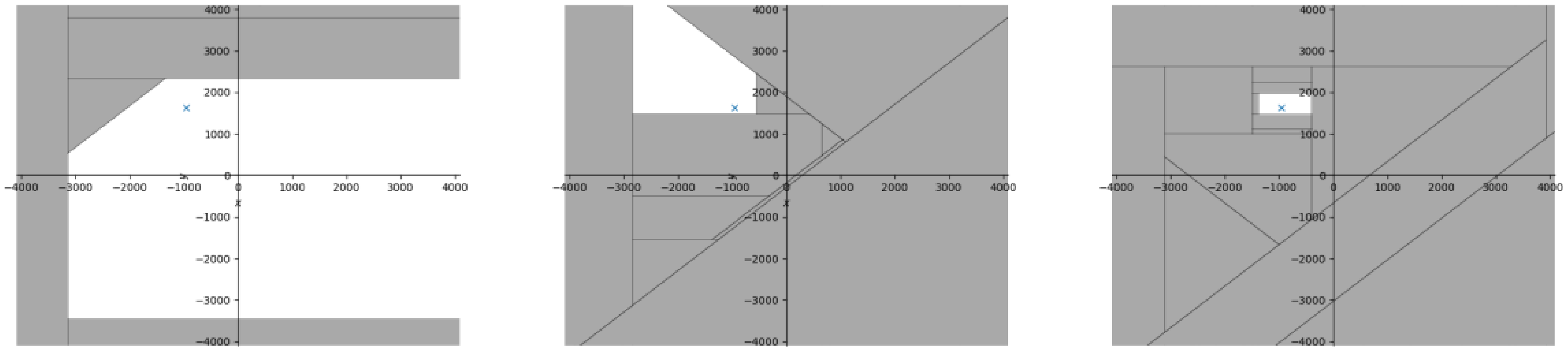


Fig. 4. Polytope containing $(\mathbf{t}_{0,0}^{[0]}, \mathbf{t}_{0,1}^{[0]})$ for 10, 50 and 100 inequalities.

How to recover t_0?

Naive method result:

Number of signatures	Number of inequalities	$\|\|\mathbf{t}_0^{[0]} - \tilde{\mathbf{t}}_0^{[0]}\|\|_\infty$	Attack time
24	9 953 + 389	5 649	0h0m23s
117	48 456 + 1 915	1 031	0h3m52s
583	241 541 + 9 378	247	1h55m47s

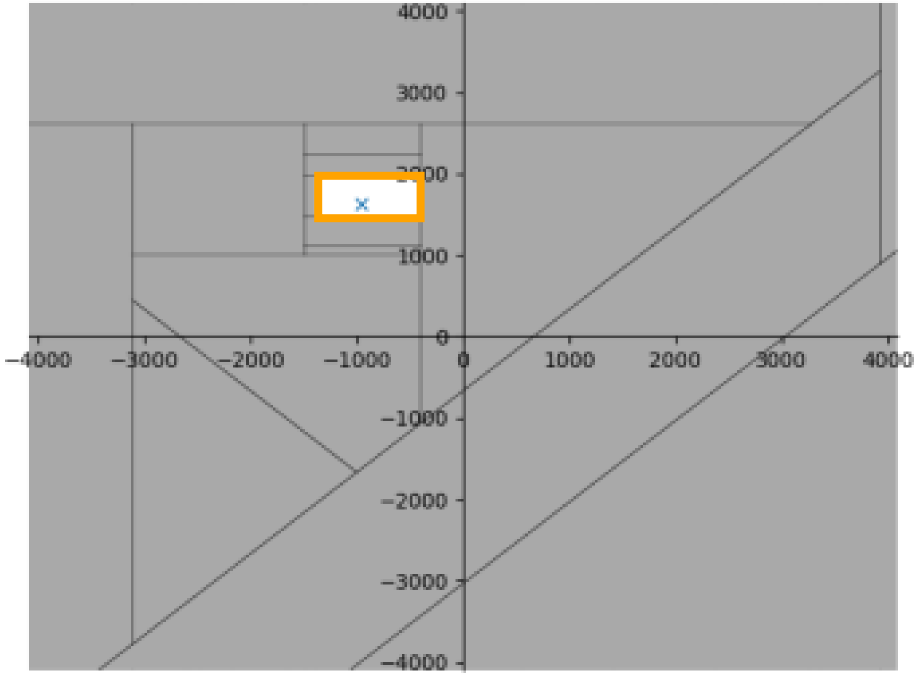
Table 4. Attack times and size of the (LP) system on $\mathbf{t}_0$.

Idea:

We have a complex algebraic representation (a lot of inequalities) of a simple geometric object (a polytope with a few faces).

The complexity of the LP solver depends on the number of inequalities:

We must filter useful inequalities.



How to recover t_0? Filtrations

Assume known C and a polynome $\widetilde{t_0}$ such that $t_0 \in B_\infty(\widetilde{t_0}, C)$.

Definition 11 *Let* $\tilde{\mathbf{t}}_0^{[0]} \in \mathcal{R}_q$ *and* $C \in \mathbb{R}_+$. *We say that an inequation on* $\mathbf{t}_0^{[0]}$ *of the form* $\{\mathbf{a}^\intercal\mathbf{x} - b \geq 0\}$ *(resp.* $\{\mathbf{a}^\intercal\mathbf{x} - b \leq 0\}$*) is useful according to* $\tilde{\mathbf{t}}_0^{[0]}$ *and* C *if and only if:*

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

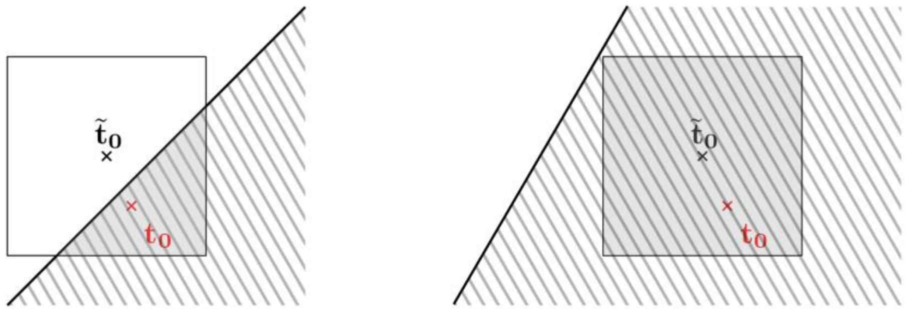


Fig. 6. On the left, a useful inequation. On the right a useless inequation.

How to recover t_0? Filtrations

Assume known C and a polynome $\tilde{t_0}$ such that $t_0 \in B_\infty(\tilde{t_0}, C)$.

Definition 11 *Let* $\tilde{\mathbf{t}}_0^{[0]} \in \mathcal{R}_q$ *and* $C \in \mathbb{R}_+$. *We say that an inequation on* $\mathbf{t}_0^{[0]}$ *of the form* $\{\mathbf{a}^\intercal \mathbf{x} - b \geq 0\}$ *(resp.* $\{\mathbf{a}^\intercal \mathbf{x} - b \leq 0\}$*) is useful according to* $\tilde{\mathbf{t}}_0^{[0]}$ *and* C *if and only if:*

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

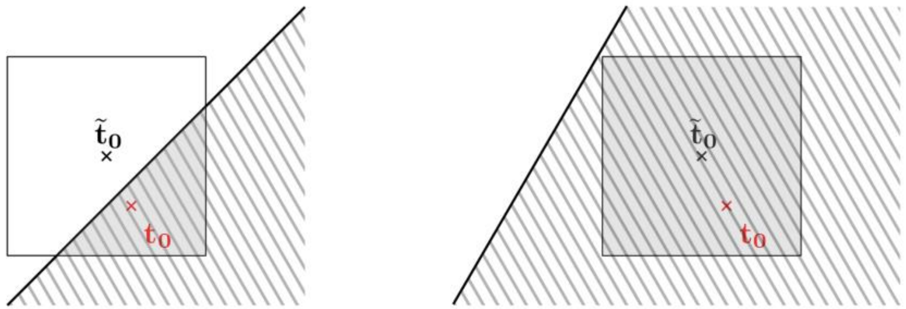


Fig. 6. On the left, a useful inequation. On the right a useless inequation.

It possible to efficently compute if an inequality is useful:

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

$$\mathbf{a}^\intercal \tilde{\mathbf{t}}_0^{[0]} - C||\mathbf{a}^\intercal||_\infty^* < b.$$

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

$$\mathbf{a}^\intercal \tilde{\mathbf{t}}_0^{[0]} + C||\mathbf{a}^\intercal||_\infty^* > b,$$

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

How to recover t_0? Attack idea

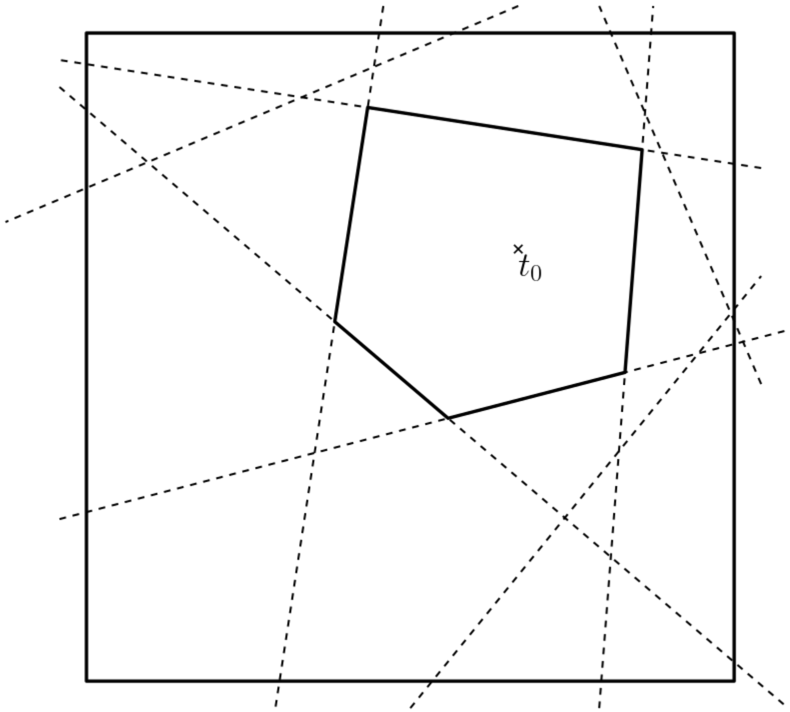
We use the strategy « Collect,guess, filter, repeat. »

Algorithm 6 Recovering $\mathbf{t}_0^{[0]}$ heuristically

Ensure: A candidate for $\mathbf{t}_0^{[0]}$

Require: An inequation step sequence $(\delta_i)_{i\in\{1,\ldots,m\}}$, a radius sequence $C_m < C_{m-1} < \cdots < C_1 = 2^{12}$.

```
t̃_0^[0] = 0
i = 1
P = {−2^12 + 1 ≤ x_i ≤ 2^12}_{i=1,...,256}
while i ≤ m do
    P = generate_useful_ineq(δ_i, t̃_0^[0], C_i)
    i = i + 1
    t̃_0^[0] = round(lp_guess(P))
return t̃_0^[0]
```



How to recover t_0? Attack idea

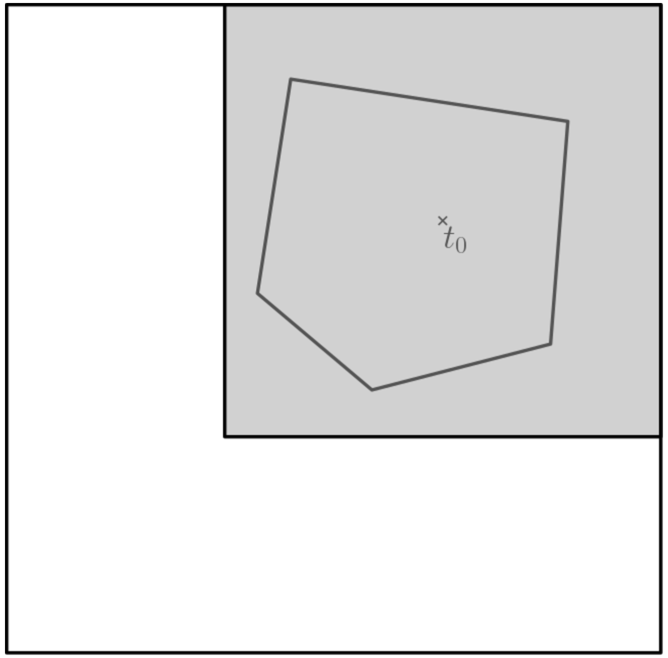
We use the strategy « Collect,guess, filter, repeat. »

Algorithm 6 `Recovering` $\mathbf{t}_0^{[0]}$ `heuristically`

Ensure: A candidate for $\mathbf{t}_0^{[0]}$

Require: An inequation step sequence $(\delta_i)_{i\in\{1,\ldots,m\}}$, a radius sequence $C_m < C_{m-1} < \cdots < C_1 = 2^{12}$.

```
t̃_0^[0] = 0
i = 1
P = {−2^12 + 1 ≤ x_i ≤ 2^12}_{i=1,...,256}
while i ≤ m do
    P = generate_useful_ineq(δ_i, t̃_0^[0], C_i)
    i = i + 1
    t̃_0^[0] = round(lp_guess(P))
return t̃_0^[0]
```



How to recover t_0? Attack idea

We use the strategy « Collect,guess, filter, repeat. »

Algorithm 6 Recovering $\mathbf{t}_0^{[0]}$ heuristically

Ensure: A candidate for $\mathbf{t}_0^{[0]}$

Require: An inequation step sequence $(\delta_i)_{i\in\{1,\ldots,m\}}$, a radius sequence $C_m < C_{m-1} < \cdots < C_1 = 2^{12}$.

1: $\tilde{\mathbf{t}}_0^{[0]} = 0$

2: $i = 1$

3: $P = \{-2^{12}+1 \leq x_i \leq 2^{12}\}_{i=1,\ldots,256}$

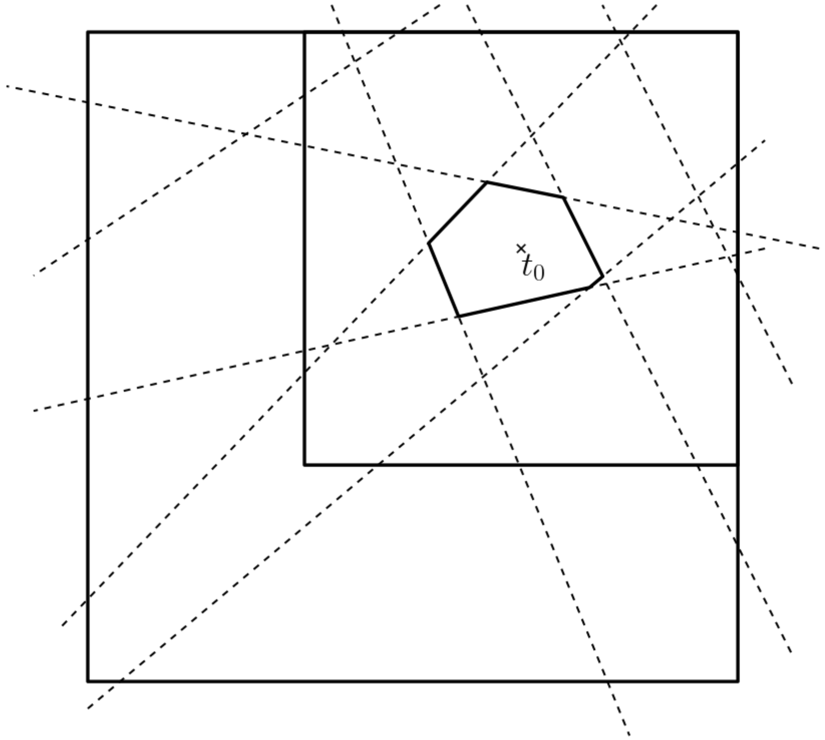
4: **while** $i \leq m$ **do**

5: $\quad P = \texttt{generate\_useful\_ineq}(\delta_i, \tilde{\mathbf{t}}_0^{[0]}, C_i)$

6: $\quad i = i + 1$

7: $\quad \tilde{\mathbf{t}}_0^{[0]} = \texttt{round(lp\_guess(P))}$

8: **return** $\tilde{\mathbf{t}}_0^{[0]}$



How to recover t_0? Attack idea

We use the strategy « Collect,guess, filter, repeat. »

Algorithm 6 `Recovering` $\mathbf{t}_0^{[0]}$ `heuristically`

Ensure: A candidate for $\mathbf{t}_0^{[0]}$

Require: An inequation step sequence $(\delta_i)_{i\in\{1,\dots,m\}}$, a radius sequence $C_m < C_{m-1} < \dots < C_1 = 2^{12}$.

$\tilde{\mathbf{t}}_0^{[0]} = 0$

$i = 1$

$P = \{-2^{12} + 1 \le x_i \le 2^{12}\}_{i=1,\dots,256}$

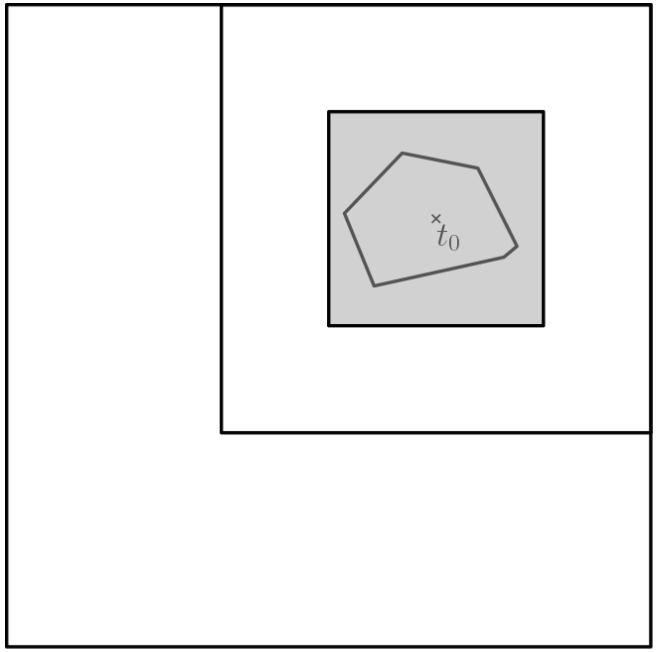
while $i \le m$ **do**

  $P =$ `generate_useful_ineq`$(\delta_i, \tilde{\mathbf{t}}_0^{[0]}, C_i)$

  $i = i + 1$

  $\tilde{\mathbf{t}}_0^{[0]} =$ `round(lp_guess(P))`

return $\tilde{\mathbf{t}}_0^{[0]}$



How to recover t_0? Attack idea

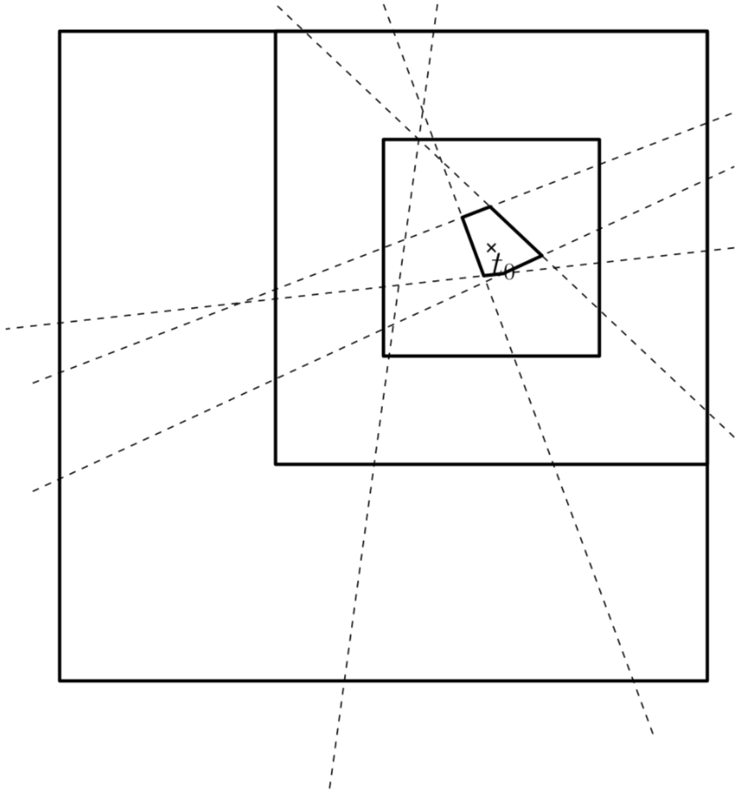
We use the strategy « Collect,guess, filter, repeat. »

Algorithm 6 Recovering $\mathbf{t}_0^{[0]}$ heuristically

Ensure: A candidate for $\mathbf{t}_0^{[0]}$

Require: An inequation step sequence $(\delta_i)_{i\in\{1,\dots,m\}}$, a radius sequence $C_m < C_{m-1} < \dots < C_1 = 2^{12}$.

$\tilde{\mathbf{t}}_0^{[0]} = 0$
$i = 1$
$P = \{-2^{12}+1 \leq x_i \leq 2^{12}\}_{i=1,\dots,256}$
while $i \leq m$ **do**
    $P =$ `generate_useful_ineq`$(\delta_i, \tilde{\mathbf{t}}_0^{[0]}, C_i)$
    $i = i + 1$
    $\tilde{\mathbf{t}}_0^{[0]} =$ `round(lp_guess(P))`
return $\tilde{\mathbf{t}}_0^{[0]}$



How to recover t_0? Attack idea

We use the strategy « Collect,guess, filter, repeat. »

Algorithm 6 Recovering $\mathbf{t}_0^{[0]}$ heuristically

Ensure: A candidate for $\mathbf{t}_0^{[0]}$

Require: An inequation step sequence $(\delta_i)_{i\in\{1,\dots,m\}}$, a radius sequence $C_m < C_{m-1} < \cdots < C_1 = 2^{12}$.

$\tilde{\mathbf{t}}_0^{[0]} = 0$

$i = 1$

$P = \{-2^{12}+1 \leq x_i \leq 2^{12}\}_{i=1,\dots,256}$

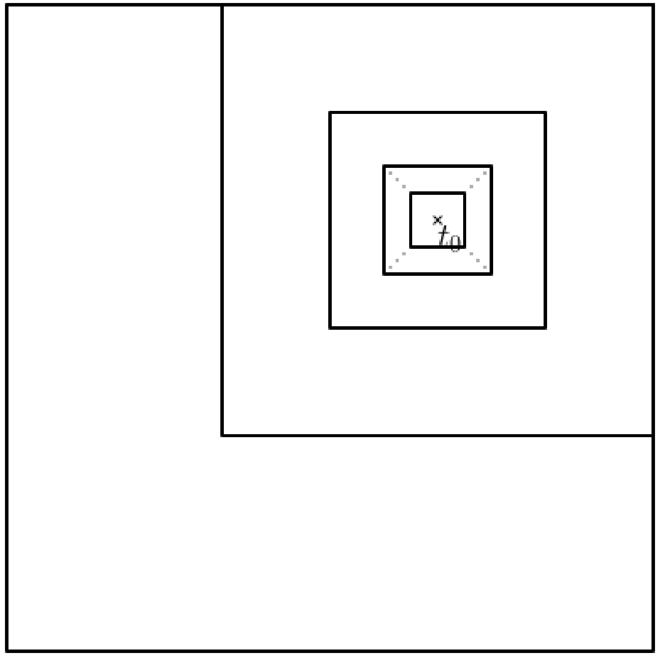
while $i \leq m$ **do**

  $P = \texttt{generate\_useful\_ineq}(\delta_i, \tilde{\mathbf{t}}_0^{[0]}, C_i)$

  $i = i + 1$

  $\tilde{\mathbf{t}}_0^{[0]} = \texttt{round(lp\_guess(P))}$

return $\tilde{\mathbf{t}}_0^{[0]}$



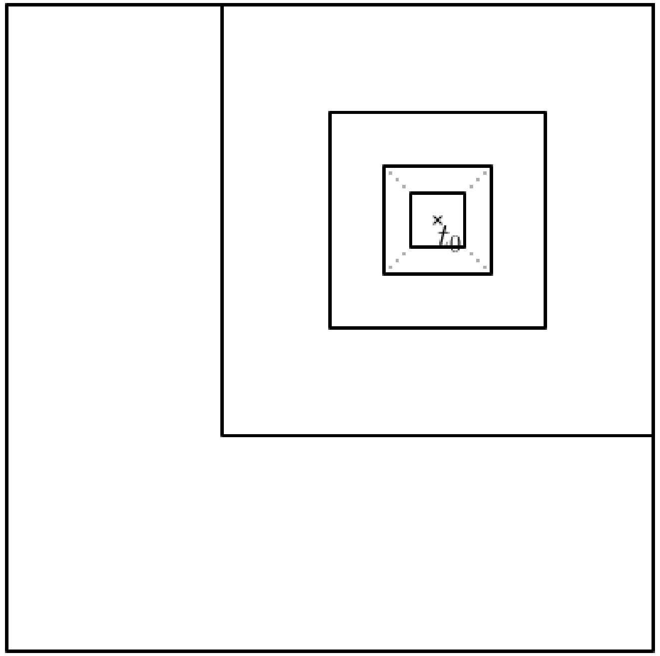
How to recover t_0? Attack idea

We use the strategy « Collect,guess, filter, repeat. »

Algorithm 6 Recovering $\mathbf{t}_0^{[0]}$ heuristically

Ensure: A candidate for $\mathbf{t}_0^{[0]}$
Require: An inequation step sequence $(\delta_i)_{i\in\{1,\dots,m\}}$, a radius sequence $C_m < C_{m-1} < \dots < C_1 = 2^{12}$.

```
t̃_0^[0] = 0
i = 1
P = {−2^12 + 1 ≤ x_i ≤ 2^12}_{i=1,...,256}
while i ≤ m do
    P = generate_useful_ineq(δ_i, t̃_0^[0], C_i)
    i = i + 1
    t̃_0^[0] = round(lp_guess(P))
return t̃_0^[0]
```



How to choose the radius sequence? And the number of inequalities?

Practical results

www.thalesgroup.com

How to recover t_0? Results

We chose $C_i = (4096, 2048, 1024, \dots, 16, 8)$ and a constant number of inequalities equal to $50\,000$. To understand what append, one can suppose t_0 known:

Round	C_i	Signatures	Inequalities selected	$\|\|\mathbf{t}_0 - \tilde{\mathbf{t}}_0\|\|_\infty$	Time
1	4096	117	48 456 + 1 915	1031	4m16s
2	2048	234	46 612 + 3 731	495	4m2s
3	1024	468	43 112 + 7 433	262	3m48s
4	512	937	37 172 + 13 540	135	3m44s
5	256	1 879	32 057 + 18 844	62	3m53s
6	128	3 743	28 787 + 21 863	37	3m53s
7	64	7 485	27 125 + 23 434	19	4m7s
8	32	14 989	26 250 + 23 434	10	4m48s
9	16	30 023	26 055 + 24 700	4	5m27s
10	8	179 515	76 487 + 74 192	0	47m5s
Total	-	179 515	392 113 + 213 853	-	1h25m3s

Table 8. Detailed results of the attack on the first KAT key.

How to recover t_0? Results

We chose $C_i = (4096, 2048, 1024, \ldots, 16, 8)$ and a constant number of inequalities equal to $50\,000$. To understand what append, one can suppose t_0 known:

Round	C_i	Signatures	Inequalities selected	$\|\|\mathbf{t}_0 - \tilde{\mathbf{t}}_0\|\|_\infty$	Time
1	4096	117	48 456 + 1 915	1031	4m16s
2	2048	234	46 612 + 3 731	495	4m2s
3	1024	468	43 112 + 7 433	262	3m48s
4	512	937	37 172 + 13 540	135	3m44s
5	256	1 879	32 057 + 18 844	62	3m53s
6	128	3 743	28 787 + 21 863	37	3m53s
7	64	7 485	27 125 + 23 434	19	4m7s
8	32	14 989	26 250 + 23 434	10	4m48s
9	16	30 023	26 055 + 24 700	4	5m27s
10	8	179 515	76 487 + 74 192	0	47m5s
Total	-	179 515	392 113 + 213 853	-	1h25m3s

Table 8. Detailed results of the attack on the first KAT key.

Without filtration : Each signature gives arround 500 inequalities on each polynmial of t_0.
It would be necessary to solve a problem (LP) of about $100\,000\,000$ inequalities in 256 variables. It is a complicated problem even for modern solvers.

How to recover t_0? Results

Conclusion:

Signatures	inequalities selected	Recovery probability	Average time	Median time
179 354	392 696 + 213 943	1	1h26m53s	1h24m8s

Table 7. Average results of the attack on $\mathbf{t}_0$

- It is possible to recover t_0 from Dilithium signatures, with less than 500 000 signatures for all security levels.
- Using t_0 in attacks is a sound assumption.
- Papers that use t_0 for attacks are realistic, and implementations must be protected against them.

The code is publicly available: GitHub - anders1901/Attack_t0

THALES

Building a future we can all trust

Thank you !

www.thalesgroup.com

References:

[NIST] 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.

[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.

[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. IACR TCHES, 2023(4):262–286, 2023.

[RJH+18]: sanna 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.

[WNDG23]: 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.