Fault Attacks Sensitivity of Public Parameters in the Dilithium Verification

Andersson Calle Viera[1,2], Alexandre Berzati[1],
Karine Heydemann[1,2]
CARDIS 2023, 15 november 2023

[1] Thales DIS, France
[2] Sorbonne Université, France

Outline

Outline

Introduction

PQC: Cryptosystems **resistant** to quantum computers are being standardized

NIST: Draft specification of **ML-DSA** derived from Version 3.1 of **Dilithium**

Importance: Soon to be implemented **securely** in many **different use cases**

Introduction

PQC: Cryptosystems **resistant** to quantum computers are being standardized

NIST: Draft specification of **ML-DSA** derived from Version 3.1 of **Dilithium**

Importance: Soon to be implemented **securely** in many **different use cases**

Introduction

PQC: Cryptosystems **resistant** to quantum computers are being standardized

NIST: Draft specification of **ML-DSA** derived from Version 3.1 of **Dilithium**

Importance: Soon to be implemented **securely** in many **different use cases**

Introduction

PQC: Cryptosystems **resistant** to quantum computers are being standardized

NIST: Draft specification of **ML-DSA** derived from Version 3.1 of **Dilithium**

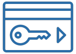
Importance: Soon to be implemented **securely** in many **different use cases**

Introduction

PQC: Cryptosystems **resistant** to quantum computers are being standardized

NIST: Draft specification of **ML-DSA** derived from Version 3.1 of **Dilithium**

Importance: Soon to be implemented **securely** in many **different use cases**



Introduction

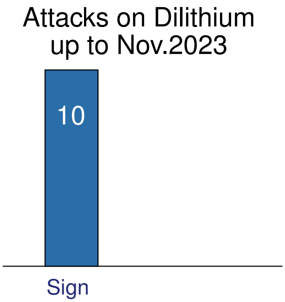
PQC: Cryptosystems **resistant** to quantum computers are being standardized

NIST: Draft specification of **ML-DSA** derived from Version 3.1 of **Dilithium**

Importance: Soon to be implemented **securely** in many **different use cases**

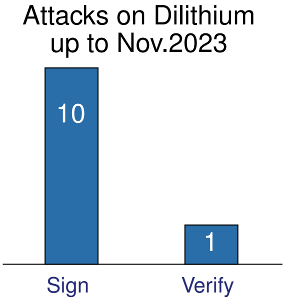
Introduction

- PQC: Cryptosystems **resistant** to quantum computers are being standardized
- NIST: Draft specification of **ML-DSA** derived from Version 3.1 of **Dilithium**
- Importance: Soon to be implemented **securely** in many **different use cases**



Introduction

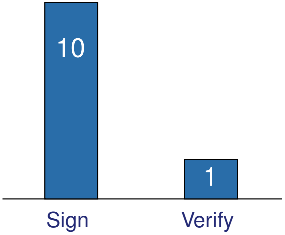
- PQC: Cryptosystems **resistant** to quantum computers are being standardized
- NIST: Draft specification of **ML-DSA** derived from Version 3.1 of **Dilithium**
- Importance: Soon to be implemented **securely** in many **different use cases**



Introduction

- PQC: Cryptosystems **resistant** to quantum computers are being standardized
- NIST: Draft specification of **ML-DSA** derived from Version 3.1 of **Dilithium**
- Importance: Soon to be implemented **securely** in many **different use cases**

Attacks on Dilithium
up to Nov.2023



- Motivation: It is considered less important to secure public parameters than private ones

Dilithium

- Public key signature algorithm, based on hard problems on Lattices
  - M-LWE
  - M-SIS
- No known efficient algorithm, classical or quantum,
  can solve these problems in less than exponential time

Dilithium

- Public key signature algorithm, based on hard problems on Lattices → M-LWE, M-SIS
- No known efficient algorithm, classical or quantum, can solve these problems in less than exponential time
- Three security levels: Dilithium-2, Dilithium-3, Dilithium-5
- Two versions: deterministic and randomized

Dilithium

- Public key signature algorithm, based on hard problems on Lattices — M-LWE, M-SIS
- No known efficient algorithm, classical or quantum, can solve these problems in less than exponential time
- Three security levels: Dilithium-2, Dilithium-3, Dilithium-5
- Two versions: deterministic and randomized
- Quotient Ring $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^n + 1)$ where $n = 2^8$ and $q = 2^{23} - 2^{13} + 1$
  - Most of the time we work with vectors of k or l elements in $\mathcal{R}_q$
  - Polynomial multiplication using the Number Theoretic Transform (NTT)

KeyGen:

1 $A \in \mathcal{R}_q^{k \times l}$

2 $(s_1, s_2) \in S_\eta^l \times S_\eta^k$

3 $t = A\, s_1 + s_2 \in \mathcal{R}_q^k$

4 $(t_1, t_0) = \texttt{Power2Round}(t, d)$

5 return $pk = (A, t_1), sk = (A, s_1, s_2, t_0, pk)$

KeyGen:

1 $A \in \mathcal{R}_q^{k \times l}$

2 $(s_1, s_2) \in S_\eta^l \times S_\eta^k$

3 $t = A s_1 + s_2 \in \mathcal{R}_q^k$

4 $(t_1, t_0) = \texttt{Power2Round}(t, d)$

5 return $pk = (A, t_1), sk = (A, s_1, s_2, t_0, pk)$

KeyGen:

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

$(s_1, s_2) \in S_\eta^l \times S_\eta^k$

$t = A\, s_1 + s_2 \in \mathcal{R}_q^k$

$(t_1, t_0) = \texttt{Power2Round}(t, d)$

return $pk = (A, t_1)$, $sk = (A, s_1, s_2, t_0, pk)$

KeyGen:

1 $A \in \mathcal{R}_q^{k \times l}$

2 $(s_1, s_2) \in S_\eta^l \times S_\eta^k$

3 $t = A\, s_1 + s_2 \in \mathcal{R}_q^k$

4 $(t_1, t_0) = \texttt{Power2Round}(t, d)$

5 return $pk = (A, t_1)$, $sk = (A, s_1, s_2, t_0, pk)$

$t_{0,0}$	$t_{0,1}$	$\cdots$	$t_{0,n-2}$	$t_{0,n-1}$
$t_{1,0}$	$t_{1,1}$	$\cdots$	$t_{1,n-2}$	$t_{1,n-1}$
		$\cdots$		
$t_{k-2,0}$	$t_{k-2,1}$	$\cdots$	$t_{k-2,n-2}$	$t_{k-2,n-1}$
$t_{k-1,0}$	$t_{k-1,1}$	$\cdots$	$t_{k-1,n-2}$	$t_{k-1,n-1}$

KeyGen:

1. $A \in \mathcal{R}_q^{k \times l}$
2. $(s_1, s_2) \in S_\eta^l \times S_\eta^k$
3. $t = A s_1 + s_2 \in \mathcal{R}_q^k$
4. $(t_1, t_0) = \texttt{Power2Round}(t, d)$
5. return $pk = (A, t_1), sk = (A, s_1, s_2, t_0, pk)$

$t_{0,0}$	$t_{0,1}$	$\cdots$	$t_{0,n-2}$	$t_{0,n-1}$
$t_{1,0}$	$t_{1,1}$	$\cdots$	$t_{1,n-2}$	$t_{1,n-1}$
		$\cdots$		
$t_{k-2,0}$	$t_{k-2,1}$	$\cdots$	$t_{k-2,n-2}$	$t_{k-2,n-1}$
$t_{k-1,0}$	$t_{k-1,1}$	$\cdots$	$t_{k-1,n-2}$	$t_{k-1,n-1}$

KeyGen:

1 $A \in \mathcal{R}_q^{k \times l}$

2 $(s_1, s_2) \in S_\eta^l \times S_\eta^k$

3 $t = A\, s_1 + s_2 \in \mathcal{R}_q^k$

4 $(t_1, t_0) = \texttt{Power2Round}(t, d)$

5 return $pk = (A, t_1)$, $sk = (A, s_1, s_2, t_0, pk)$

$t_{0,0}$	$t_{0,1}$	$\cdots$	$t_{0,n-2}$	$t_{0,n-1}$
$t_{1,0}$	$t_{1,1}$	$\cdots$	$t_{1,n-2}$	$t_{1,n-1}$
		$\cdots$		
$t_{k-2,0}$	$t_{k-2,1}$	$\cdots$	$t_{k-2,n-2}$	$t_{k-2,n-1}$
$t_{k-1,0}$	$t_{k-1,1}$	$\cdots$	$t_{k-1,n-2}$	$t_{k-1,n-1}$

KeyGen:

1 $A \in \mathcal{R}_q^{k \times l}$

2 $(s_1, s_2) \in S_\eta^l \times S_\eta^k$

3 $t = A\, s_1 + s_2 \in \mathcal{R}_q^k$

4 $(t_1, t_0) = \texttt{Power2Round}(t, d)$

5 return $pk = (A, t_1)$, $sk = (A, s_1, s_2, t_0, pk)$

$t_{0,0}$	$t_{0,1}$	$\cdots$	$t_{0,n-2}$	$t_{0,n-1}$
$t_{1,0}$	$t_{1,1}$	$\cdots$	$t_{1,n-2}$	$t_{1,n-1}$
		$\cdots$		
$t_{k-2,0}$	$t_{k-2,1}$	$\cdots$	$t_{k-2,n-2}$	$t_{k-2,n-1}$
$t_{k-1,0}$	$t_{k-1,1}$	$\cdots$	$t_{k-1,n-2}$	$t_{k-1,n-1}$

Sign(M, $sk = (A,\ s_1,\ s_2,\ t_0,\ pk)$):

```
(z, h) = ⊥
while (z, h) = ⊥ do
    y ∈ S̃^l_{γ1}
    w = A y
    w1 = HighBits(w)
    c ∈ B_τ = H(pk || M || w1)
    z = y + c s1
    r0 = LowBits(w − c s2)
    if ||z||_∞ ≥ γ1 − β or ||r0||_∞ ≥ γ2 − β, then (z, h) = ⊥
    else
        h = MakeHint(−c t0, w − c s2 + c t0)
        if ||c t0||_∞ ≥ γ2, then (z, h) = ⊥
return σ = (c, z, h)
```

Sign(M, $sk = (A, s_1, s_2, t_0, pk)$):

$(z, h) = \perp$
while $(z, h) = \perp$ do
  $y \in \tilde{S}^l_{\gamma_1}$
  $w = A\, y$
  $w_1 = \texttt{HighBits}(w)$
  $c \in B_\tau = \texttt{H}(pk \,||\, M \,||\, w_1)$
  $z = y + c\, s_1$
  $r_0 = \texttt{LowBits}(w - c\, s_2)$
  if $||z||_\infty \geq \gamma_1 - \beta$ or $||r_0||_\infty \geq \gamma_2 - \beta$, then $(z, h) = \perp$
  else
    $h = \texttt{MakeHint}(-c\, t_0, w - c\, s_2 + c\, t_0)$
    if $||c\, t_0||_\infty \geq \gamma_2$, then $(z, h) = \perp$
return $\sigma = (c, z, h)$

Sign(M, $sk = (A,\ s_1,\ s_2,\ t_0,\ pk)$):

$(z, h) = \perp$

while $(z, h) = \perp$ do

  $y \in \tilde{S}^{l}_{\gamma_1}$

  $w = A\,y$

  $w_1 = \texttt{HighBits}(w)$

  $c \in B_\tau = \mathtt{H}(pk \,||\, M \,||\, w_1)$

  $z = y + c\, s_1$

  $r_0 = \texttt{LowBits}(w - c\, s_2)$

  if $||z||_\infty \geq \gamma_1 - \beta$ or $||r_0||_\infty \geq \gamma_2 - \beta$, then $(z, h) = \perp$

  else

    $h = \texttt{MakeHint}(-c\, t_0,\ w - c\, s_2 + c\, t_0)$

    if $\|c\, t_0\|_\infty \geq \gamma_2$, then $(z, h) = \perp$

return $\sigma = (c,\ z,\ h)$

Sign(M, $sk = (A,\ s_1,\ s_2,\ t_0,\ pk)$):

$(z, h) = \perp$

while $(z, h) = \perp$ do

$\quad y \in \tilde{S}^{l}_{\gamma_1}$

$\quad w = A\,y$

$\quad w_1 = \texttt{HighBits}(w)$

$\quad c \in B_\tau = \texttt{H}(pk\,||\,M\,||\,w_1)$

$\quad z = y + c\,s_1$

$\quad r_0 = \texttt{LowBits}(w - c\,s_2)$

$\quad$ if $||z||_\infty \geq \gamma_1 - \beta$ or $||r_0||_\infty \geq \gamma_2 - \beta$, then $(z, h) = \perp$

$\quad$ else

$\quad\quad h = \texttt{MakeHint}(-c\,t_0,\ w - c\,s_2 + c\,t_0)$

$\quad\quad$ if $||c\,t_0||_\infty \geq \gamma_2$, then $(z, h) = \perp$

return $\sigma = (c,\ z,\ h)$

Sign$(M, sk = (A, s_1, s_2, t_0, pk))$:

$(z, h) = \perp$

while $(z, h) = \perp$ do

  $y \in \tilde{S}^l_{\gamma_1}$

  $w = A\, y$

  $w_1 = \texttt{HighBits}(w)$

  $c \in B_\tau = \mathtt{H}(pk \,||\, M \,||\, w_1)$

  $z = y + c\, s_1$

  $r_0 = \texttt{LowBits}(w - c\, s_2)$

  if $||z||_\infty \geq \gamma_1 - \beta$ or $||r_0||_\infty \geq \gamma_2 - \beta$, then $(z, h) = \perp$

  else

    $h = \texttt{MakeHint}(-c\, t_0,\ w - c\, s_2 + c\, t_0)$

    if $\|c\, t_0\|_\infty \geq \gamma_2$, then $(z, h) = \perp$

return $\sigma = (c,\ z,\ h)$

Verify($pk = (\rho, t_1), M, \sigma = (c, z, h)$):

1 $w_1' = \texttt{UseHint}(h, A z - c\, t_1 2^d)$

Verify($pk = (\rho, t_1), M, \sigma = (c, z, h)$):

$A\,z - c\,t_1 2^d$

1 $w_1' = \texttt{UseHint}(h, \boxed{A\,z - c\,t_1 2^d})$

Verify($pk = (\rho, t_1), M, \sigma = (c, z, h)$):

$$A\,z - c\,t_1 2^d = A\,\overbrace{(y + c\,s_1)}^{z} - c\,\overbrace{(A\,s_1 + s_2 - t_0)}^{t_1 2^d}$$

1 $w_1' = \texttt{UseHint}(h, \boxed{A\,z - c\,t_1 2^d})$

Verify($pk = (\rho, t_1), M, \sigma = (c, z, h)$):

$$\begin{aligned} A\,z - c\,t_1 2^d &= A\,\overbrace{(y + c\,s_1)}^{z} - c\,\overbrace{(A\,s_1 + s_2 - t_0)}^{t_1 2^d} \\ &= \underbrace{A\,y}_{w} - c s_2 + c t_0 \\ &= w - c s_2 + c t_0 \end{aligned}$$

1 $w'_1 = \texttt{UseHint}(h, \boxed{A\,z - c\,t_1 2^d})$

Verify($pk = (\rho, t_1), M, \sigma = (c, z, h)$):

$$\begin{aligned} A z - c\, t_1 2^d &= A \overbrace{(y + c\, s_1)}^{z} - c \overbrace{(A\, s_1 + s_2 - t_0)}^{t_1 2^d} \\ &= \underbrace{A\, y}_{w} - c s_2 + c t_0 \\ &= w - c s_2 + c t_0 \end{aligned}$$

Lemma 1.1 [1] $\implies$ $\texttt{UseHint}(h, w - c s_2 + c\, t_0) = \texttt{HighBits}(w - c s_2)$

1 $w_1' = \texttt{UseHint}(h, \boxed{A z - c\, t_1 2^d})$

[1] S. Bai, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, D. Stehlé, CRYSTALS - Dilithium: Digital Signatures from Module Lattices

Verify($pk = (\rho, t_1), M, \sigma = (c, z, h)$):

$$A\,z - c\,t_1 2^d = A\,\overbrace{(y + c\,s_1)}^{z} - c\,\overbrace{(A\,s_1 + s_2 - t_0)}^{t_1 2^d}$$
$$= \underbrace{A\,y}_{w} - c s_2 + c t_0$$
$$= w - c s_2 + c t_0$$

Lemma 1.1 [1] $\Longrightarrow$ $\texttt{UseHint}(h, w - c s_2 + c\,t_0) = \texttt{HighBits}(w - c s_2)$

Lemma 2 [1] $\Longrightarrow$ $\texttt{HighBits}(w - c s_2) = \underbrace{\texttt{HighBits}_q(w)}_{}$ $= w_1$

1 $w'_1 = \texttt{UseHint}(h, \boxed{A\,z - c\,t_1 2^d})$

[1] S. Bai, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, D. Stehlé, CRYSTALS - Dilithium: Digital Signatures from Module Lattices

Verify$(pk = (\rho, t_1), M, \sigma = (c, z, h))$:

$$
\begin{aligned}
A\,z - c\,t_1 2^d &= A\,\overbrace{(y + c\,s_1)}^{z} - c\,\overbrace{(A\,s_1 + s_2 - t_0)}^{t_1 2^d} \\
&= \underbrace{A\,y}_{w} - c s_2 + c t_0 \\
&= w - c s_2 + c t_0
\end{aligned}
$$

Lemma 1.1 [1] $\Longrightarrow$ $\texttt{UseHint}(h, w - c s_2 + c\,t_0) = \texttt{HighBits}(w - c s_2)$

Lemma 2 [1] $\Longrightarrow$ $\texttt{HighBits}(w - c s_2) = \underbrace{\texttt{HighBits}_q(w)}_{} = w_1$

1 $w_1' = \texttt{UseHint}(h, \boxed{A\,z - c\,t_1 2^d})$

2 if $||z||_\infty < \gamma_1 - \beta$ and $c = \mathtt{H}(pk\,||\,M\,||\,w_1')$ and $\#$ 1's in $h \leq \omega$

3 return *True*

4 else

5 return *False*

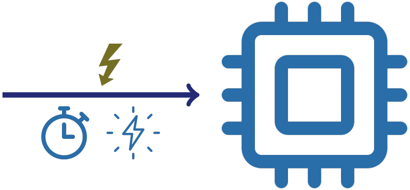
[1] S. Bai, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, D. Stehlé, CRYSTALS - Dilithium: Digital Signatures from Module Lattices

Fault Models

- Fault Attacks on signature algorithms: retrieve secrets/**verify false signatures**

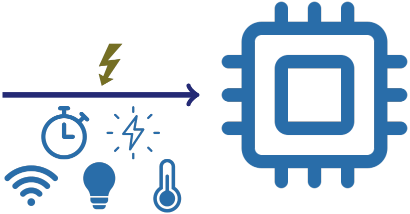
Fault Models

- Fault Attacks on signature algorithms: retrieve secrets/**verify false signatures**



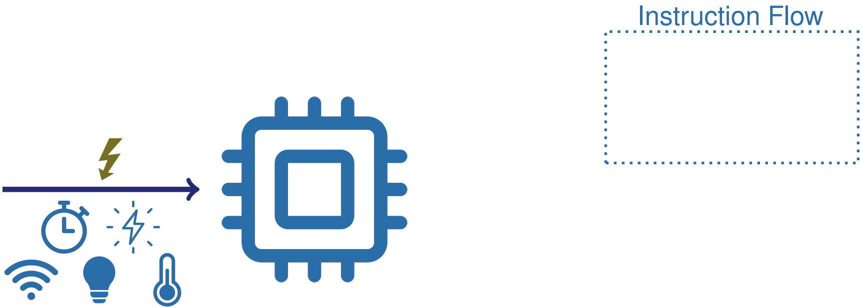
Fault Models

- Fault Attacks on signature algorithms: retrieve secrets/**verify false signatures**



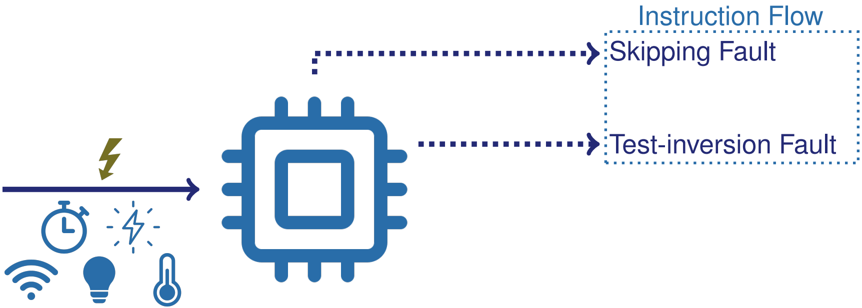
Fault Models

- Fault Attacks on signature algorithms: retrieve secrets/**verify false signatures**



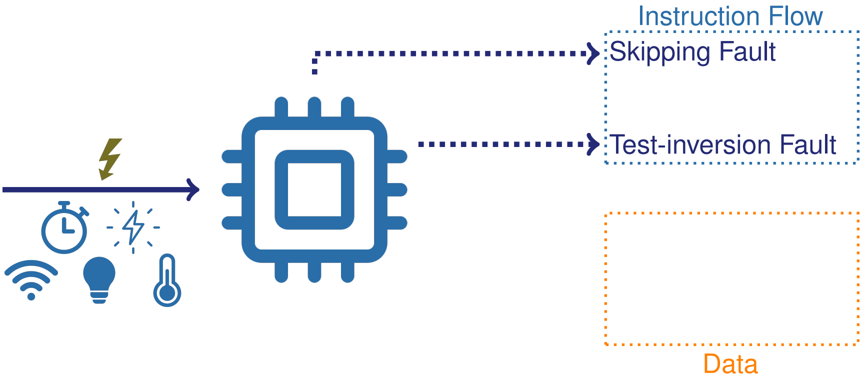
Fault Models

- Fault Attacks on signature algorithms: retrieve secrets/**verify false signatures**



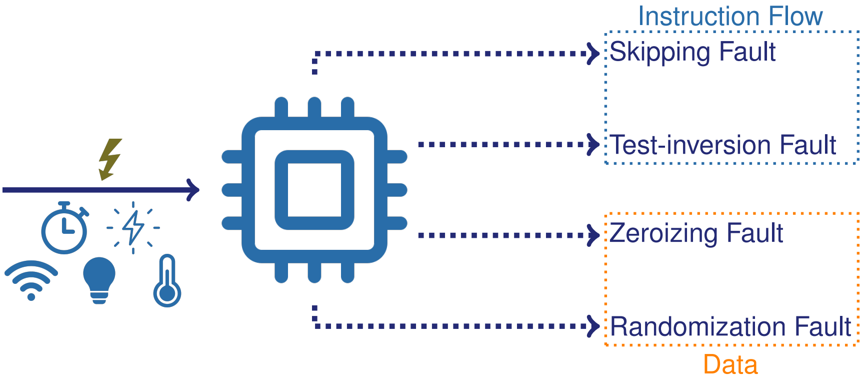
Fault Models

- Fault Attacks on signature algorithms: retrieve secrets/**verify false signatures**



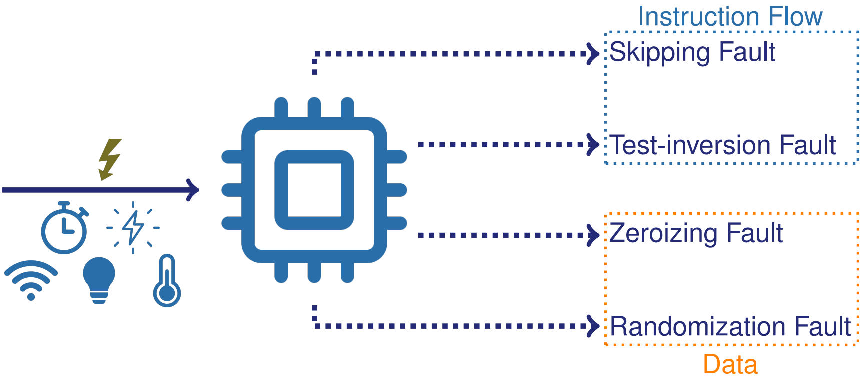
Fault Models

- Fault Attacks on signature algorithms: retrieve secrets/**verify false signatures**



Fault Models

- Fault Attacks on signature algorithms: retrieve secrets/**verify false signatures**



- Here, we only consider the **type** and **number** of fault observation

Outline

How to make accept a false signature?

Goal: Make accept false signatures by Verify with faults injected

Verification **checks** are the **most sensitive** and usually **hardened**: 3 checks $\approx$ 3 faults

How to make accept a false signature?

Goal: Make accept false signatures by Verify with faults injected

Verification **checks** are the **most sensitive** and usually **hardened**: 3 checks $\approx$ 3 faults

Other sensitive **locations** requiring possibly **less faults** to inject?

How to make accept a false signature?

Goal: Make accept false signatures by Verify with faults injected

Verification **checks** are the **most sensitive** and usually **hardened**: 3 checks $\approx$ 3 faults

Other sensitive **locations** requiring possibly **less faults** to inject?

1 $||z||_\infty < \gamma_1 - \beta$

How to make accept a false signature?

Goal: Make accept false signatures by Verify with faults injected

Verification **checks** are the **most sensitive** and usually **hardened**: 3 checks $\approx$ 3 faults

Other sensitive **locations** requiring possibly **less faults** to inject?

1 Choose random z such that $||z||_\infty < \gamma_1 - \beta$

1 $||z||_\infty < \gamma_1 - \beta$

How to make accept a false signature?

Goal: Make accept false signatures by Verify with faults injected

Verification **checks** are the **most sensitive** and usually **hardened**: 3 checks $\approx$ 3 faults

Other sensitive **locations** requiring possibly **less faults** to inject?

1 Choose random z such that $||z||_\infty < \gamma_1 - \beta$

1 $||z||_\infty < \gamma_1 - \beta$

2 $c = \mathtt{H}(pk \,||\, M \,||\, w_1')$

How to make accept a false signature?

Goal: Make accept false signatures by Verify with faults injected

Verification **checks** are the **most sensitive** and usually **hardened**: 3 checks $\approx$ 3 faults

Other sensitive **locations** requiring possibly **less faults** to inject?

1 Choose random z such that $||z||_\infty < \gamma_1 - \beta$

1 $||z||_\infty < \gamma_1 - \beta$

2 $\texttt{HighBits}(Az - ct_1 2^d) = \texttt{HighBits}(Az)$

How to make accept a false signature?

Goal: Make accept false signatures by Verify with faults injected

Verification **checks** are the **most sensitive** and usually **hardened**: 3 checks $\approx$ 3 faults

Other sensitive **locations** requiring possibly **less faults** to inject?

1 Choose random z such that $||z||_\infty < \gamma_1 - \beta$

2 Assure that $ct_1 2^d$ doesn't affect the high bits of Az

1 $||z||_\infty < \gamma_1 - \beta$

2 $\texttt{HighBits}(Az - ct_1 2^d) = \texttt{HighBits}(Az)$

How to make accept a false signature?

Goal: Make accept false signatures by Verify with faults injected

Verification **checks** are the **most sensitive** and usually **hardened**: 3 checks $\approx$ 3 faults

Other sensitive **locations** requiring possibly **less faults** to inject?

1 Choose random z such that $||z||_\infty < \gamma_1 - \beta$

2 Assure that $ct_1 2^d$ doesn't affect the high bits of Az

1 $||z||_\infty < \gamma_1 - \beta$

2 $\texttt{HighBits}(Az - ct_1 2^d) = \texttt{HighBits}(Az)$

3 $\#$ 1's in $h \leq \omega$

How to make accept a false signature?

Goal: Make accept false signatures by Verify with faults injected

Verification **checks** are the **most sensitive** and usually **hardened**: 3 checks $\approx$ 3 faults

Other sensitive **locations** requiring possibly **less faults** to inject?

1 Choose random z such that $||z||_\infty < \gamma_1 - \beta$
2 Assure that $ct_1 2^d$ doesn't affect the high bits of Az
3 Compute h with $\#\, 1$'s in $h \leq \omega$ accordingly

1 $||z||_\infty < \gamma_1 - \beta$
2 $\texttt{HighBits}(Az - ct_1 2^d) = \texttt{HighBits}(Az)$
3 $\#\, 1$'s in $h \leq \omega$

How to make accept a false signature?

1 Choose random z such that $||z||_\infty < \gamma_1 - \beta$

2 Assure that $ct_1 2^d$ doesn't affect the high bits of Az

3 Compute h with $\#$ 1's in $h \leq \omega$ accordingly

1 $||z||_\infty < \gamma_1 - \beta$

2 $\texttt{HighBits}(Az - ct_1 2^d) = \texttt{HighBits}(Az)$

3 $\#$ 1's in $h \leq \omega$

How to make accept a false signature?

1 Choose random z such that $||z||_\infty < \gamma_1 - \beta$
2 Assure that $ct_1 2^d$ doesn't affect the high bits of Az
3 Compute h with $\#$ 1's in $h \leq \omega$ accordingly

1 $||z||_\infty < \gamma_1 - \beta$
2 $\texttt{HighBits}(Az - ct_1 2^d) = \texttt{HighBits}(Az)$
3 $\#$ 1's in $h \leq \omega$

Proposition 1

Let $z \in R_q^l$ be a random vector with $||z||_\infty < \gamma_1 - \beta$.
If at least one of the following conditions is satisfied:

P1. $\|c\, t_1\, 2^d\|_\infty \leq 0$

P2. $\|c\, t_1\, 2^d\|_\infty \leq \beta$ and $\|\texttt{LowBits}(A\, z - c\, t_1\, 2^d)\|_\infty < \gamma_2 - \beta$

P3. $\|c\, t_1\, 2^d\|_\infty \leq \gamma_2$ and $h = \texttt{MakeHint}(c\, t_1\, 2^d, A\, z - c\, t_1\, 2^d)$

Then, $\texttt{HighBits}(A\, z - c\, t_1\, 2^d) = \texttt{HighBits}(A\, z)$.

$$\|c\, t_1 2^d\|_\infty \leq 0 \implies A\, z - c\, t_1\, 2^d = A\, z$$

How to make accept a false signature?

1 Choose random z such that $||z||_\infty < \gamma_1 - \beta$
2 Assure that $ct_1 2^d$ doesn't affect the high bits of Az
3 Compute h with $\#$ 1's in $h \le \omega$ accordingly

1 $||z||_\infty < \gamma_1 - \beta$
2 $\texttt{HighBits}(Az - ct_1 2^d) = \texttt{HighBits}(Az)$
3 $\#$ 1's in $h \le \omega$

Proposition 1

Let $z \in R_q^l$ be a random vector with $||z||_\infty < \gamma_1 - \beta$.
If at least one of the following conditions is satisfied:

P1. $\|c\, t_1\, 2^d\|_\infty \le 0$

P2. $\|c\, t_1\, 2^d\|_\infty \le \beta$ and $\|\texttt{LowBits}(A\, z - c\, t_1\, 2^d)\|_\infty < \gamma_2 - \beta$

P3. $\|c\, t_1\, 2^d\|_\infty \le \gamma_2$ and $h = \texttt{MakeHint}(c\, t_1\, 2^d, A\, z - c\, t_1\, 2^d)$

Then, $\texttt{HighBits}(A\, z - c\, t_1\, 2^d) = \texttt{HighBits}(A\, z)$.

$\|c\, t_1 2^d\|_\infty \le \beta \implies \texttt{HighBits}(A\, z - c\, t_1\, 2^d) = \texttt{HighBits}(A\, z)$ (Lemma 2 in [1])

How to make accept a false signature?

1 Choose random z such that $||z||_\infty < \gamma_1 - \beta$
2 Assure that $ct_1 2^d$ doesn't affect the high bits of Az
3 Compute h with $\#$ 1's in $h \leq \omega$ accordingly

1 $||z||_\infty < \gamma_1 - \beta$
2 $\texttt{HighBits}(Az - ct_1 2^d) = \texttt{HighBits}(Az)$
3 $\#$ 1's in $h \leq \omega$

Proposition 1

Let $z \in R_q^l$ be a random vector with $||z||_\infty < \gamma_1 - \beta$.
If at least one of the following conditions is satisfied:

P1. $\|c\, t_1\, 2^d\|_\infty \leq 0$

P2. $\|c\, t_1\, 2^d\|_\infty \leq \beta$ and $\|\texttt{LowBits}(A\, z - c\, t_1\, 2^d)\|_\infty < \gamma_2 - \beta$

P3. $\|c\, t_1\, 2^d\|_\infty \leq \gamma_2$ and $h = \texttt{MakeHint}(c\, t_1\, 2^d, A\, z - c\, t_1\, 2^d)$

Then, $\texttt{HighBits}(A\, z - c\, t_1\, 2^d) = \texttt{HighBits}(A\, z)$.

$\|c\, t_1 2^d\|_\infty \leq \gamma_2 \Rightarrow \texttt{HighBits}(A\, z - c\, t_1\, 2^d) = \texttt{HighBits}(A\, z)$ (Lemma 1.1 in [1])

How to make accept a false signature?

1 Choose random z such that $||z||_\infty < \gamma_1 - \beta$

2 Assure that $ct_1 2^d$ doesn't affect the high bits of Az

3 Compute h with $\#$ 1's in $h \leq \omega$ accordingly

1 $||z||_\infty < \gamma_1 - \beta$

2 $\texttt{HighBits}(Az - ct_1 2^d) = \texttt{HighBits}(Az)$

3 $\#$ 1's in $h \leq \omega$

Proposition 1

Let $z \in R_q^l$ be a random vector with $||z||_\infty < \gamma_1 - \beta$.
If at least one of the following conditions is satisfied:

P1. $\|c\,t_1\,2^d\|_\infty \leq 0$

P2. $\|c\,t_1\,2^d\|_\infty \leq \beta$ and $\|\texttt{LowBits}(A\,z - c\,t_1\,2^d)\|_\infty < \gamma_2 - \beta$

P3. $\|c\,t_1\,2^d\|_\infty \leq \gamma_2$ and $h = \texttt{MakeHint}(c\,t_1\,2^d, A\,z - c\,t_1\,2^d)$

Then, $\texttt{HighBits}(A\,z - c\,t_1\,2^d) = \texttt{HighBits}(A\,z)$.

Problem: $\|c\,t_1 2^d\|_\infty$ is too big to use Proposition 1

How to make accept a false signature?

1 Choose random z such that $||z||_\infty < \gamma_1 - \beta$
2 Assure that $ct_1 2^d$ doesn't affect the high bits of Az
3 Compute h with $\#\,1$'s in $h \leq \omega$ accordingly

1 $||z||_\infty < \gamma_1 - \beta$
2 $\texttt{HighBits}(Az - ct_1 2^d) = \texttt{HighBits}(Az)$
3 $\#\,1$'s in $h \leq \omega$

Proposition 1

Let $z \in R_q^l$ be a random vector with $||z||_\infty < \gamma_1 - \beta$.
If at least one of the following conditions is satisfied:

P1. $\|c\,t_1\,2^d\|_\infty \leq 0$

P2. $\|c\,t_1\,2^d\|_\infty \leq \beta$ and $\|\texttt{LowBits}(A\,z - c\,t_1\,2^d)\|_\infty < \gamma_2 - \beta$

P3. $\|c\,t_1\,2^d\|_\infty \leq \gamma_2$ and $h = \texttt{MakeHint}(c\,t_1\,2^d, A\,z - c\,t_1\,2^d)$

Then, $\texttt{HighBits}(A\,z - c\,t_1\,2^d) = \texttt{HighBits}(A\,z)$.

Problem: $\|c\,t_1 2^d\|_\infty$ is too big to use Proposition 1
Solution: Inject Faults such as to be in P1, P2, or P3

Where to target?

$$1 \quad w_1' = \texttt{UseHint}(h, A\,z - c\,t_1 2^d)$$

Where to target?

$$1 \quad w_1' = \texttt{UseHint}(h, Az - c\,t_1 2^d)$$

Scenario 1: Sampling of c

- Direct use of P1

Where to target?

$$1 \; w_1' = \texttt{UseHint}(h, Az - c t_1 2^d)$$

Scenario 1: Sampling of c

- Direct use of P1

Scenario 2: Shift by d

- Direct use of P3

Where to target?

Scenario 3: Subtraction
- Direct use of P1

$$1 \quad w_1' = \texttt{UseHint}(h, Az - ct_1 2^d)$$

Scenario 1: Sampling of c
- Direct use of P1

Scenario 2: Shift by d
- Direct use of P3

Where to target?

Dilithium Verify code snippet from PQClean [2]

```c
if (siglen != CRYPTO_BYTES)
    return -1;

unpack_pk(rho, &t1, pk);
if (unpack_sig(c, &z, &h, sig))
    return -1;
if (polyvecl_chknorm(&z, GAMMA1 - BETA))
    return -1;

/* Compute CRH(H(rho, t1), msg) */
shake256(mu, SEEDBYTES, pk, CRYPTO_PUBLICKEYBYTES);
shake256_init(&state);
shake256_absorb(&state, mu, SEEDBYTES);
shake256_absorb(&state, m, mlen);
shake256_finalize(&state);
shake256_squeeze(mu, CRHBYTES, &state);

/* Matrix-vector multiplication; Az - c2^dt1 */
poly_challenge(&cp, c);
polyvec_matrix_expand(mat, rho);

polyvecl_ntt(&z);
polyvec_matrix_pointwise_montgomery(&w1, mat, &z);

poly_ntt(&cp);
polyveck_shiftl(&t1);
polyveck_ntt(&t1);
polyveck_pointwise_poly_montgomery(&t1, &cp, &t1);

polyveck_sub(&w1, &w1, &t1);
polyveck_reduce(&w1);
polyveck_invntt_tomont(&w1);

/* Reconstruct w1 */
polyveck_caddq(&w1);
polyveck_use_hint(&w1, &w1, &h);
polyveck_pack_w1(buf, &w1);

/* Call random oracle and verify challenge */
shake256_init(&state);
shake256_absorb(&state, mu, CRHBYTES);
shake256_absorb(&state, buf, K*POLYW1_PACKEDBYTES);
shake256_finalize(&state);
shake256_squeeze(c2, SEEDBYTES, &state);
for (i = 0; i < SEEDBYTES; ++i) {
    if (c[i] != c2[i]) {
        return -1;
    }
}
return 0;
```

Highlighting potential sensitive operations

```
if (siglen != CRYPTO_BYTES)
    return -1;
unpack_pk(rho, &t1, pk);
if (unpack_sig(c, &z, &h, sig))
    return -1;
if (polyvecl_chknorm(&z, GAMMA1 - BETA))
    return -1;
/* Compute mu=CRH(H(rho, t1), msg) to sample c2 */
                          ...
/* Matrix-vector multiplication; Az - c2^dt1 */
poly_challenge(&cp, c);
polyvec_matrix_expand(mat, rho);
polyvecl_ntt(&z);
polyvec_matrix_pointwise_montgomery(&w1, mat, &z);
poly_ntt(&cp);
polyveck_shiftl(&t1);
polyveck_ntt(&t1);
polyveck_pointwise_poly_montgomery(&t1, &cp, &t1);
polyveck_sub(&w1, &w1, &t1);
polyveck_reduce(&w1);
polyveck_invntt_tomont(&w1);
/* Reconstruct w1 */
polyveck_caddq(&w1);
polyveck_use_hint(&w1, &w1, &h);
```

Highlighting potential sensitive operations

```c
if (siglen != CRYPTO_BYTES)
    return -1;
unpack_pk(rho, &t1, pk);
if (unpack_sig(c, &z, &h, sig))
    return -1;
if (polyvecl_chknorm(&z, GAMMA1 - BETA))
    return -1;
/* Compute mu=CRH(H(rho, t1), msg) to sample c2 */
                        ...
/* Matrix-vector multiplication; Az - c2^dt1 */
poly_challenge(&cp, c);
polyvec_matrix_expand(mat, rho);
polyvecl_ntt(&z);
polyvec_matrix_pointwise_montgomery(&w1, mat, &z);
poly_ntt(&cp);
polyveck_shiftl(&t1);
polyveck_ntt(&t1);
polyveck_pointwise_poly_montgomery(&t1, &cp, &t1);
polyveck_sub(&w1, &w1, &t1);
polyveck_reduce(&w1);
polyveck_invntt_tomont(&w1);
/* Reconstruct w1 */
polyveck_caddq(&w1);
polyveck_use_hint(&w1, &w1, &h);
```

Scenario 1: Sampling of c

- for loop inside: skipping/test-inversion/zeroizing
- Direct use of P1

Highlighting potential sensitive operations

```c
if (siglen != CRYPTO_BYTES)
    return -1;
unpack_pk(rho, &t1, pk);
if (unpack_sig(c, &z, &h, sig))
    return -1;
if (polyvecl_chknorm(&z, GAMMA1 - BETA))
    return -1;
/* Compute mu=CRH(H(rho, t1), msg) to sample c2 */
...
/* Matrix-vector multiplication; Az - c2^dt1 */
poly_challenge(&cp, c);
polyvec_matrix_expand(mat, rho);
polyvecl_ntt(&z);
polyvec_matrix_pointwise_montgomery(&w1, mat, &z);
poly_ntt(&cp);
polyveck_shiftl(&t1);
polyveck_ntt(&t1);
polyveck_pointwise_poly_montgomery(&t1, &cp, &t1);
polyveck_sub(&w1, &w1, &t1);
polyveck_reduce(&w1);
polyveck_invntt_tomont(&w1);
/* Reconstruct w1 */
polyveck_caddq(&w1);
polyveck_use_hint(&w1, &w1, &h);
```

Scenario 1: Sampling of c

- for loop inside: skipping/test-inversion/zeroizing
- Direct use of P1

Scenario 2: Shift by d

- `polyveck_shiftl` function call: skipping
- `poly_shiftl` function call: skipping
- constant d: zeroizing
- Direct use of P3

Highlighting potential sensitive operations

```
if (siglen != CRYPTO_BYTES)
    return -1;
unpack_pk(rho, &t1, pk);
if (unpack_sig(c, &z, &h, sig))
    return -1;
if (polyvecl_chknorm(&z, GAMMA1 - BETA))
    return -1;
/* Compute mu=CRH(H(rho, t1), msg) to sample c2 */
                        ...
/* Matrix-vector multiplication; Az - c2^dt1 */
poly_challenge(&cp, c);
polyvec_matrix_expand(mat, rho);
polyvecl_ntt(&z);
polyvec_matrix_pointwise_montgomery(&w1, mat, &z);
poly_ntt(&cp);
polyveck_shiftl(&t1);
polyveck_ntt(&t1);
polyveck_pointwise_poly_montgomery(&t1, &cp, &t1);
polyveck_sub(&w1, &w1, &t1);
polyveck_reduce(&w1);
polyveck_invntt_tomont(&w1);
/* Reconstruct w1 */
polyveck_caddq(&w1);
polyveck_use_hint(&w1, &w1, &h);
```

Scenario 1: Sampling of c

- for loop inside: skipping/test-inversion/zeroizing
- Direct use of P1

Scenario 2: Shift by d

- `polyveck_shiftl` function call: skipping
- `poly_shiftl` function call: skipping
- constant d: zeroizing
- Direct use of P3

Scenario 3: Subtraction

- `polyveck_sub` function call: skipping
- `poly_sub` function call: skipping
- Direct use of P1

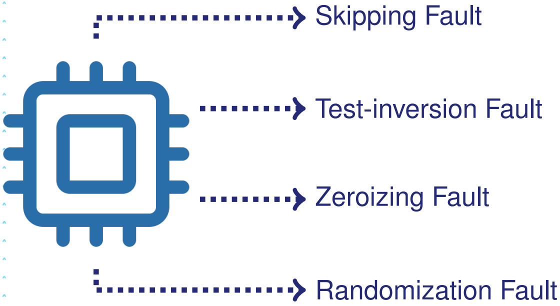
OPEN

And then?

- Every condition used $\leftrightarrow$ Algorithm to forge signatures (given the corresponding faults)
- Verified in Python with simulated faults (modified versions of Dilithium)

And then?

Every condition used ↔ Algorithm to forge signatures (given the corresponding faults)
Verified in Python with simulated faults (modified versions of Dilithium)



Scenario 1: Sampling of c

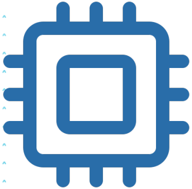
Scenario 2: Shift by d

Scenario 3: Subtraction

And then?

Every condition used $\leftrightarrow$ Algorithm to forge signatures (given the corresponding faults)
Verified in Python with simulated faults (modified versions of Dilithium)

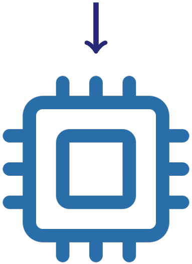
Skipping Fault



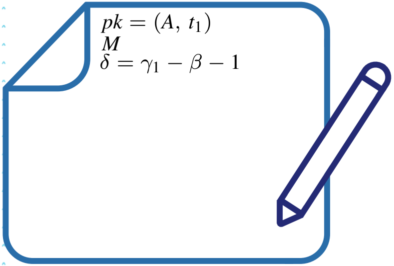
Scenario 3: Subtraction

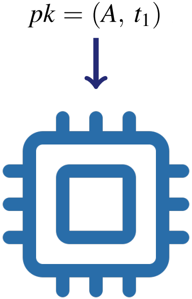
And then?

$$pk = (A,\ t_1)$$

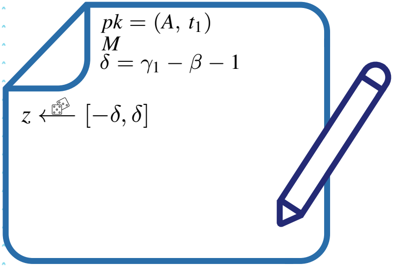


And then?

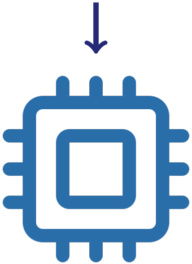




And then?

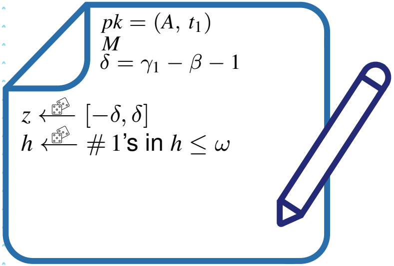


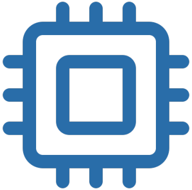
$pk = (A, t_1)$



And then?

$pk = (A,\ t_1)$



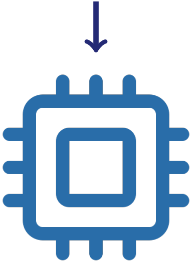


And then?

$pk = (A,\ t_1)$
M
$\delta = \gamma_1 - \beta - 1$

$z \xleftarrow{\$} [-\delta, \delta]$
$h \xleftarrow{\$}$ $\#$ 1's in $h \leq \omega$
$w_1 = \texttt{UseHint}(h, Az)$

$pk = (A,\ t_1)$



And then?

$pk = (A,\ t_1)$

M

$\delta = \gamma_1 - \beta - 1$

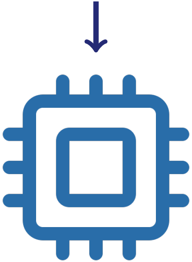
$z \leftarrow [-\delta, \delta]$

$h \leftarrow \#$ 1's in $h \leq \omega$

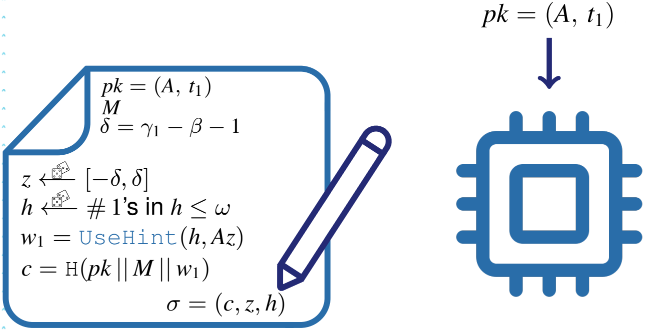
$w_1 = \texttt{UseHint}(h, Az)$

$c = \mathtt{H}(pk \,||\, M \,||\, w_1)$

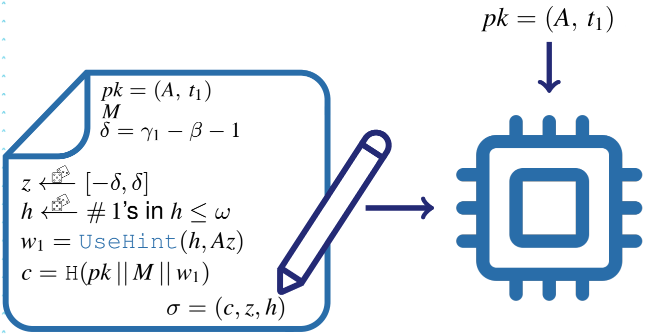
$pk = (A,\ t_1)$



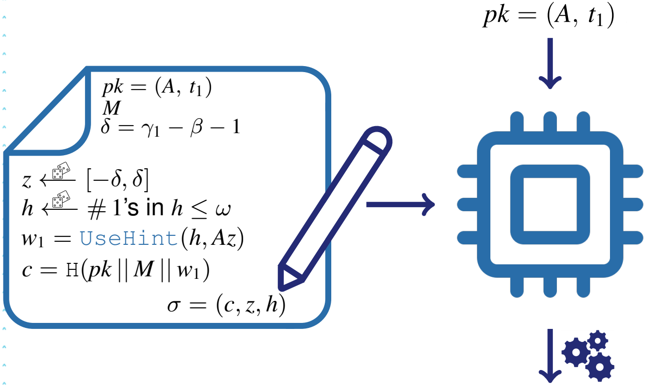
And then?


$pk = (A,\ t_1)$
$pk = (A,\ t_1)$
M
$\delta = \gamma_1 - \beta - 1$
$z \leftarrow [-\delta, \delta]$
$h \leftarrow \#$ 1's in $h \leq \omega$
$w_1 = \texttt{UseHint}(h, Az)$
$c = \mathtt{H}(pk\,||\,M\,||\,w_1)$
$\sigma = (c, z, h)$

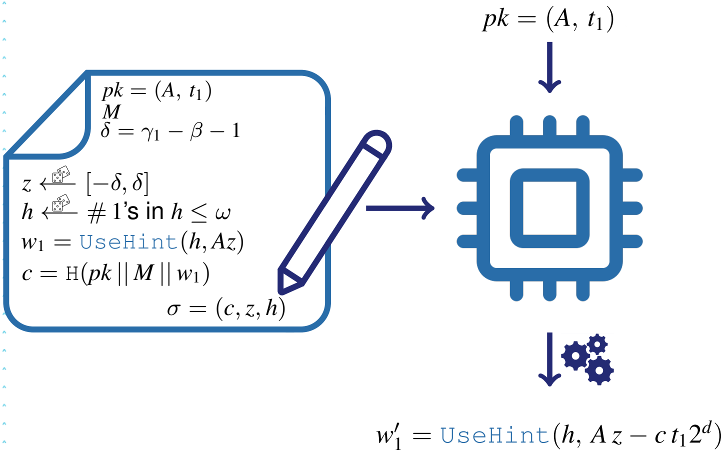
And then?


$pk = (A,\ t_1)$
$pk = (A,\ t_1)$
M
$\delta = \gamma_1 - \beta - 1$
$z \leftarrow [-\delta, \delta]$
$h \leftarrow$ # 1's in $h \leq \omega$
$w_1 = \texttt{UseHint}(h, Az)$
$c = \mathtt{H}(pk \,||\, M \,||\, w_1)$
$\sigma = (c, z, h)$

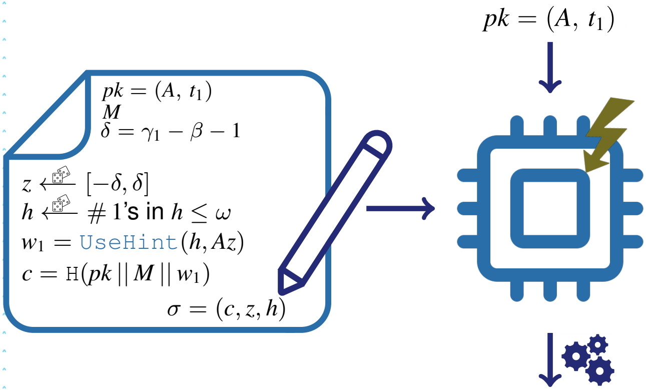
And then?


$pk = (A,\ t_1)$
$pk = (A,\ t_1)$
M
$\delta = \gamma_1 - \beta - 1$
$z \xleftarrow{} [-\delta, \delta]$
$h \xleftarrow{}$ # 1's in $h \leq \omega$
$w_1 = \texttt{UseHint}(h, Az)$
$c = \mathtt{H}(pk \,||\, M \,||\, w_1)$
$\sigma = (c, z, h)$

And then?


$pk = (A, t_1)$
M
$\delta = \gamma_1 - \beta - 1$
$z \xleftarrow{} [-\delta, \delta]$
$h \xleftarrow{}$ # 1's in $h \leq \omega$
$w_1 = \texttt{UseHint}(h, Az)$
$c = \mathtt{H}(pk \,||\, M \,||\, w_1)$
$\sigma = (c, z, h)$
$pk = (A, t_1)$
$w'_1 = \texttt{UseHint}(h, A\,z - c\,t_1 2^d)$

And then?



$$w'_1 = \texttt{UseHint}(h,\ A\,z \qquad\qquad)$$

And then?

$pk = (A,\ t_1)$

M

$\delta = \gamma_1 - \beta - 1$

$z \xleftarrow{\$} [-\delta, \delta]$

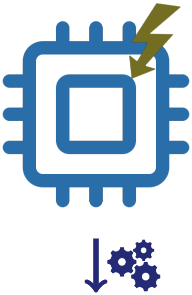
$h \xleftarrow{\$}$ # 1's in $h \leq \omega$

$w_1 = \texttt{UseHint}(h, Az)$

$c = \mathtt{H}(pk \,||\, M \,||\, w_1)$

$\sigma = (c, z, h)$

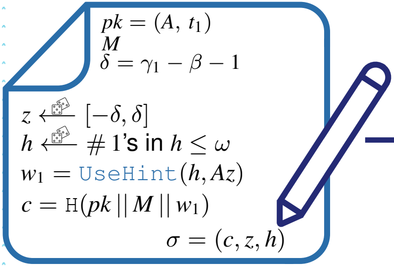
$pk = (A,\ t_1)$



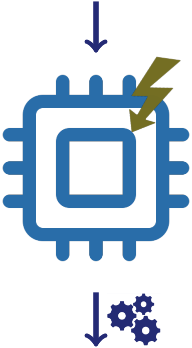
$w_1' = \texttt{UseHint}(h,\ A\,z \qquad\qquad)$

if $||z||_\infty < \gamma_1 - \beta$

And then?



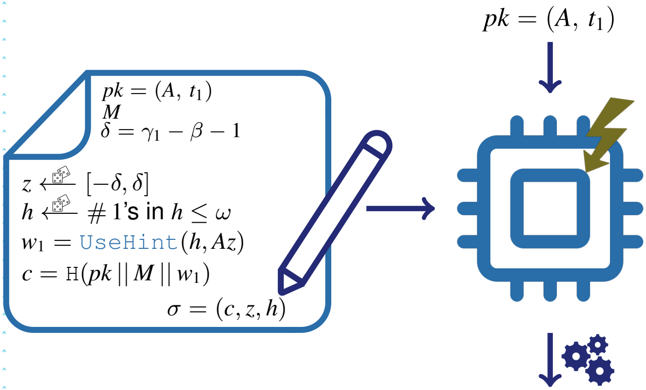
$pk = (A,\ t_1)$



$w'_1 = \mathtt{UseHint}(h,\ A\,z \qquad\qquad)$

if $||z||_\infty < \gamma_1 - \beta$

And then?

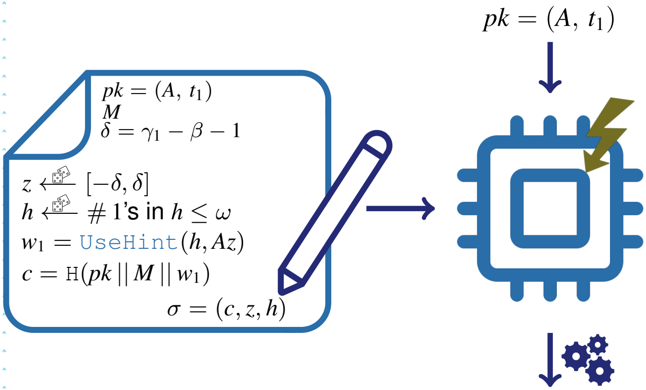


$w_1' = \texttt{UseHint}(h,\ A\,z \qquad\qquad)$

if $||z||_\infty < \gamma_1 - \beta$ and $c = \mathtt{H}(pk \,||\, M \,||\, w_1')$

✓

And then?

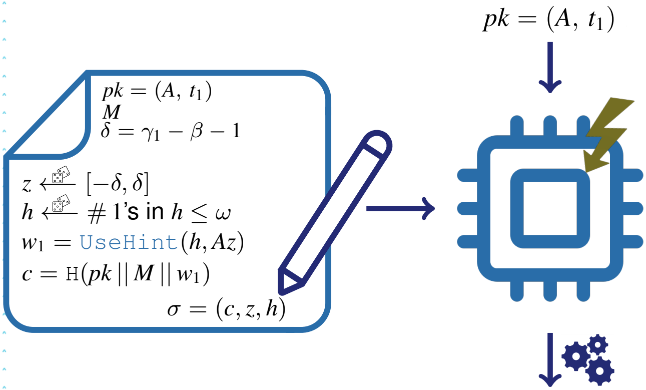


$$w'_1 = \texttt{UseHint}(h, A\,z \qquad\qquad)$$

if $||z||_\infty < \gamma_1 - \beta$ and $c = \mathtt{H}(pk \,||\, M \,||\, w'_1)$

✓ ✓

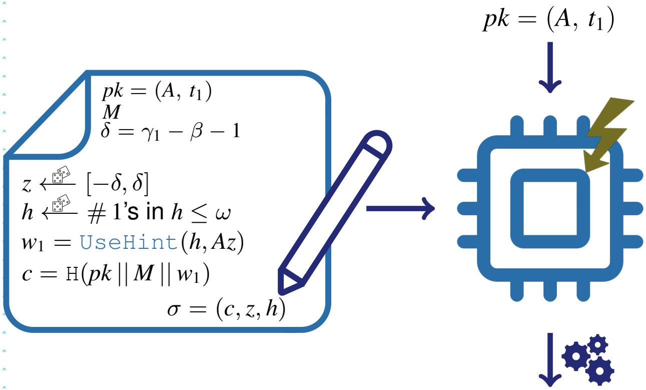
And then?



$w'_1 = \texttt{UseHint}(h,\ A\,z\qquad\quad)$

if $||z||_\infty < \gamma_1 - \beta$ ✓ and $c = \mathrm{H}(pk \,||\, M \,||\, w'_1)$ ✓ and # 1's in $h \leq \omega$

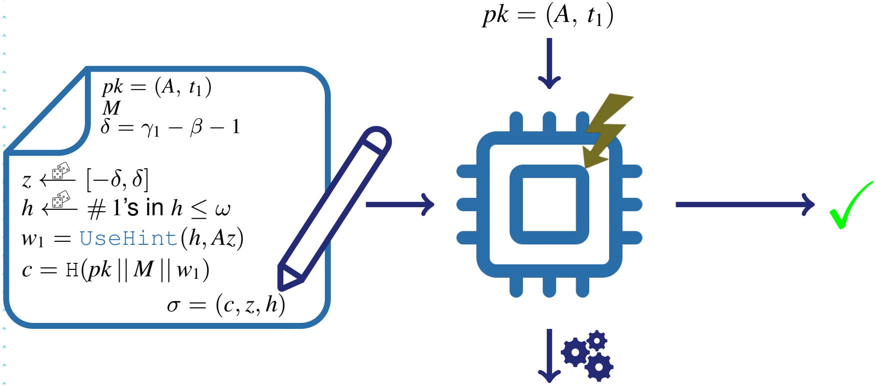
And then?



$$w'_1 = \texttt{UseHint}(h,\ A\,z \qquad\qquad)$$

if $||z||_\infty < \gamma_1 - \beta$ ✓ and $c = \mathrm{H}(pk \,||\, M \,||\, w'_1)$ ✓ and # 1's in $h \leq \omega$ ✓

And then?



$$w_1' = \texttt{UseHint}(h,\ A\,z \qquad\qquad)$$

if $||z||_\infty < \gamma_1 - \beta$ ✓ and $c = \mathrm{H}(pk \,||\, M \,||\, w_1')$ ✓ and $\#$ 1's in $h \leq \omega$ ✓

Outline

Countermeasures

- Don't store the result of the subtraction in the same location as the left operand
- Conditions from Proposition 1 based on the idea to make $ct_1 2^d$ "smaller"
- Idea: Make sure it is not...

Countermeasures

- Don't store the result of the subtraction in the same location as the left operand
- Conditions from Proposition 1 based on the idea to make $ct_1 2^d$ "smaller"
- Idea: Make sure it is not...

Versions		Skipping	Test-Inv	Randomization	Zeroizing	Countermeasures
Scenario 1	for	✓	✓	-	✓	Distribution Check, Norm Check
	`TAU`	-	-	✓	✓	
Scenario 2	polyvec for	✓	✓	-	✓	Distribution Check, Norm Check, Verify d, Split d
	poly for	✓	✓	-	✓	
	d	✓	-	✓	✓	
Scenario 3	polyvec for	✓	✓	-	✓	Alternative implementation
	poly for	✓	✓	-	✓	
	function call	✓	-	-	✓	

Table: Vulnerable locations of Verify and the corresponding fault models and countermeasures (✓: easy exploitation, ✓: possible exploitable, -: not applicable)

Outline

Conclusion

To sum up:

Conclusion

To sum up:

- Make sure that $ct_1 2^d$ is not small in practice
- Otherwise false signatures can be verified
- Simple countermeasures to make Verify intrisically resistant

Conclusion

To sum up:

- Make sure that $ct_1 2^d$ is not small in practice
- Otherwise false signatures can be verified
- Simple countermeasures to make Verify intrisically resistant

- Is it possible to exploit P2?
- Are there more operations vulnerable?
- What about in practice (faults analyzed, countermeasures proposed)?

Conclusion

To sum up:

- Make sure that $ct_1 2^d$ is not small in practice
- Otherwise false signatures can be verified
- Simple countermeasures to make Verify intrisically resistant

- Is it possible to exploit P2?
- Are there more operations vulnerable?
- What about in practice (faults analyzed, countermeasures proposed)?

Thank you

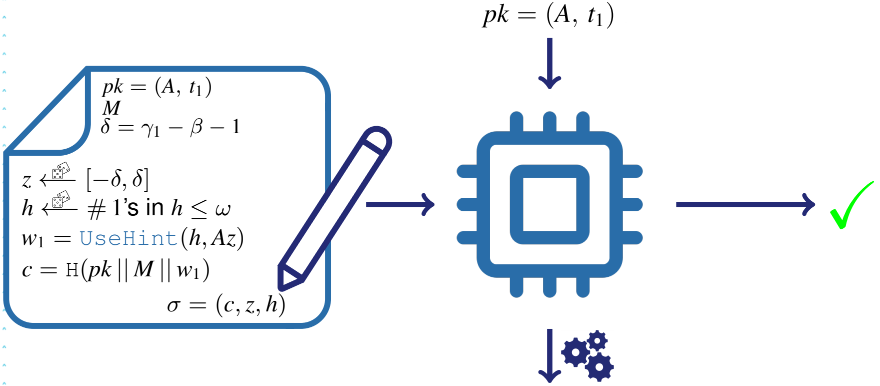
Questions?

Bibliography

[1] S. Bai, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, D. Stehlé, *CRYSTALS - Dilithium: Digital Signatures from Module Lattices*.

[2] M.J. Kannwischer, P. Schwabe, D. Stebila, T. Wiggers, *Improving Software Quality in Cryptography Standardization Projects*.

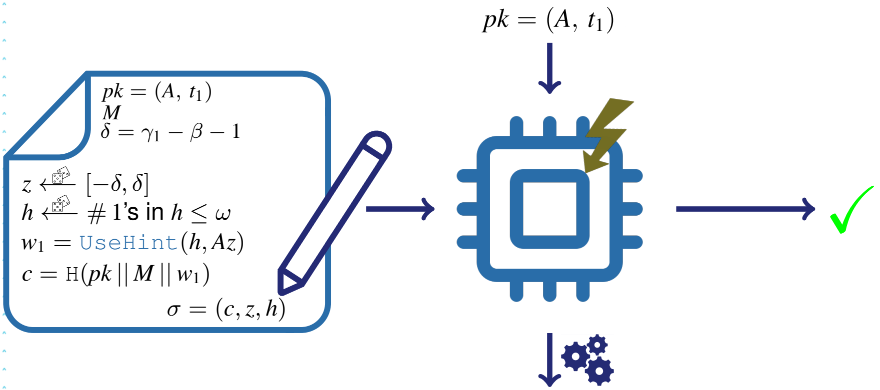
Example for Scenario 1



$$w_1' = \texttt{UseHint}(h,\ A\,z - c\,t_1 2^d)$$

if $||z||_\infty < \gamma_1 - \beta$ ✓ and $c = \mathtt{H}(pk \,||\, M \,||\, w_1')$ ✓ and # 1's in $h \leq \omega$ ✓

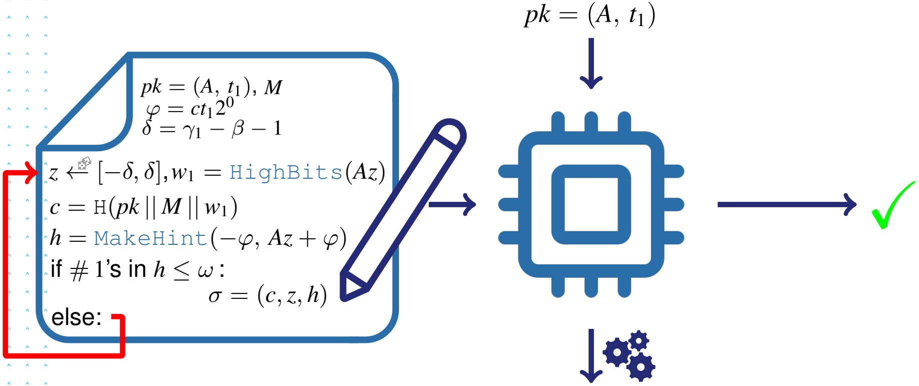
Example for Scenario 1



$w'_1 = \texttt{UseHint}(h,\ A\,z - 0\,t_1 2^d)$

if $||z||_\infty < \gamma_1 - \beta$ ✓ and $c = \mathtt{H}(pk \,||\, M \,||\, w'_1)$ ✓ and $\#$ 1's in $h \leq \omega$ ✓

Example for Scenario 2



$$w_1' = \texttt{UseHint}(h,\ A\,z - c\,t_1 2^d)$$

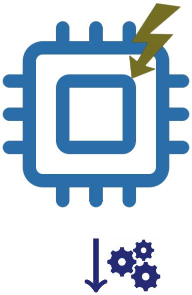
if $||z||_\infty < \gamma_1 - \beta$ ✓ and $c = \mathtt{H}(pk \,||\, M \,||\, w_1')$ ✓ and $\#$ 1's in $h \leq \omega$ ✓

Example for Scenario 2

$pk = (A, t_1)$

$pk = (A, t_1), M$
$\varphi = c t_1 2^0$
$\delta = \gamma_1 - \beta - 1$

$z \leftarrow [-\delta, \delta], w_1 = \texttt{HighBits}(Az)$
$c = \mathtt{H}(pk \,||\, M \,||\, w_1)$
$h = \texttt{MakeHint}(-\varphi, Az + \varphi)$
if $\#$ 1's in $h \leq \omega$:
$\sigma = (c, z, h)$
else:



✓

$w_1' = \texttt{UseHint}(h, Az - c\, t_1 2^0)$

if $||z||_\infty < \gamma_1 - \beta$ and $c = \mathtt{H}(pk \,||\, M \,||\, w_1')$ and $\#$ 1's in $h \leq \omega$
✓ ✓ ✓