# Differential Fault Attacks on Deterministic Lattice Signatures

**Leon Groot Bruinderink[1], Peter Pessl[2]**
**[1]Technische Universiteit Eindhoven, [2]Graz University of Technology**

CHES 2018, September 10

- Signatures and nonce reuse...
  - solution: determinism, $n = \text{Hash}(M, K)$
  - EdDSA, Deterministic ECDSA

- Signatures and nonce reuse...
  - solution: determinism, $n = \text{Hash}(M, K)$
  - EdDSA, Deterministic ECDSA

- Signatures and differential fault attacks...
  - sign $M$ twice and fault after Hash
  - nonce-reuse for “different” messages [PSS$^+$17, ABF$^+$18]

- Signatures and nonce reuse...
  - solution: determinism, $n = \text{Hash}(M, K)$
  - EdDSA, Deterministic ECDSA
- Signatures and differential fault attacks...
  - sign $M$ twice and fault after Hash
  - nonce-reuse for “different” messages [PSS$^+$17, ABF$^+$18]
- ...but what about lattices?

# Our Contribution

- Extend DFA to deterministic lattice signatures
  - Dilithium, qTESLA$^{(*)}$

- Closer look at peculiarities of lattices
  - rejection-sampling technique
  - key compression
  - efficient exploitation of partial nonce reuse

# Dilithium Lattice Signatures

- Module-LWE assumption with base ring $\mathcal{R}_q = \mathbb{Z}_q[x]/(x^{256} + 1)$
- Key generation
  - "small" keys $(\mathbf{s}_1, \mathbf{s}_2) \in \mathcal{R}_q^l \times \mathcal{R}_q^k$
  - random public $\mathbf{A} \in \mathcal{R}_q^{k \times \ell}$
  - public key $\mathbf{t} = \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2$
- Determinism to protect against bad randomness

# Dilithium - Framework

**Input:** Message $M$, private key $sk = (\mathbf{A}, K, \mathbf{s}_1, \mathbf{s}_2)$

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

  $\mathbf{y} := \text{DeterministicSample}(K || M || \kappa{+}{+})$

  $\mathbf{w} := \mathbf{Ay}$; $\mathbf{w}_1 := \text{HighBits}(\mathbf{w})$

  $c := \text{H}(M || \mathbf{w}_1)$

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

  **if** $\|\mathbf{z}\|_\infty \geq \gamma_1 - \beta$ or $\text{Reject}(\mathbf{w}, c\mathbf{s}_2)$ or … **then** $\mathbf{z} := \perp$

**return** $(\mathbf{z}, c)$

# Dilithium - Framework

**Input:** Message $M$, private key $sk = (\mathbf{A}, K, \mathbf{s}_1, \mathbf{s}_2)$

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

2: $\quad \mathbf{y} := \text{DeterministicSample}(K||M||\kappa++)$ $\triangleright$ same $M$, same $\mathbf{y} \rightarrow$ nonce reuse?

3: $\quad \mathbf{w} := \mathbf{A}\mathbf{y}$; $\mathbf{w}_1 := \text{HighBits}(\mathbf{w})$

4: $\quad c := \text{H}(M||\mathbf{w}_1)$

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

6: $\quad$ **if** $||\mathbf{z}||_\infty \geq \gamma_1 - \beta$ or $\text{Reject}(\mathbf{w}, c\mathbf{s}_2)$ or ... **then** $\mathbf{z} := \perp$

7: **return** $(\mathbf{z}, c)$

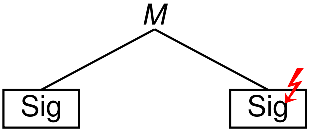
# Dilithium - Framework

**Input:** Message $M$, private key $sk = (\mathbf{A}, K, \mathbf{s}_1, \mathbf{s}_2)$

1: **while** $\mathbf{z} = \perp$ **do**
2: $\quad \mathbf{y} := \text{DeterministicSample}(K||M||\kappa++)$ ▷ same $M$, same $\mathbf{y} \rightarrow$ nonce reuse?
3: $\quad \mathbf{w} := \mathbf{Ay}$; $\mathbf{w}_1 := \text{HighBits}(\mathbf{w})$
4: $\quad c := \text{H}(M||\mathbf{w}_1)$
5: $\quad \mathbf{z} := \mathbf{y} + c\mathbf{s}_1$ ▷ only if we can change something else!
6: $\quad$ **if** $\|\mathbf{z}\|_\infty \geq \gamma_1 - \beta$ or $\text{Reject}(\mathbf{w}, c\mathbf{s}_2)$ or ... **then** $\mathbf{z} := \perp$
7: **return** $(\mathbf{z}, c)$

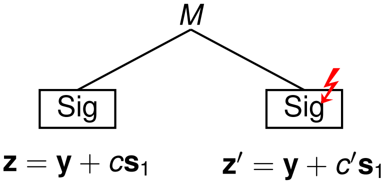
# Attack Intuition

- Sign same $M$ twice
  1. no fault: $(\mathbf{z}, c)$
  2. fault s.t. same $\mathbf{y}$ but different $c$: $(\mathbf{z}', c')$



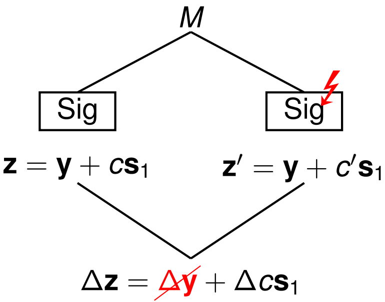
# Attack Intuition

- Sign same $M$ twice
  1. no fault: $(\mathbf{z}, c)$
  2. fault s.t. same $\mathbf{y}$ but different $c$: $(\mathbf{z}', c')$



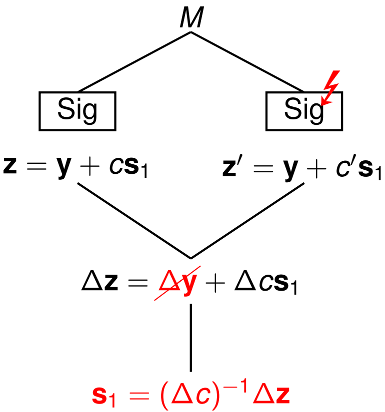
# Attack Intuition

- Sign same $M$ twice
    1. no fault: $(\mathbf{z}, c)$
    2. fault s.t. same $\mathbf{y}$ but different $c$: $(\mathbf{z}', c')$



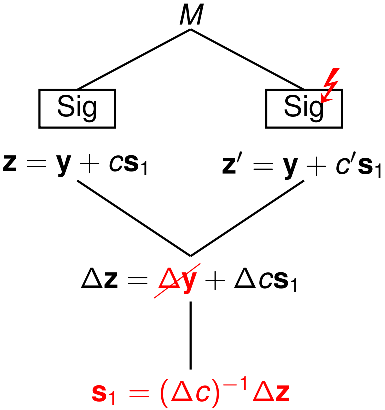
# Attack Intuition

- Sign same $M$ twice
  1. no fault: $(\mathbf{z}, c)$
  2. fault s.t. same $\mathbf{y}$ but different $c$: $(\mathbf{z}', c')$



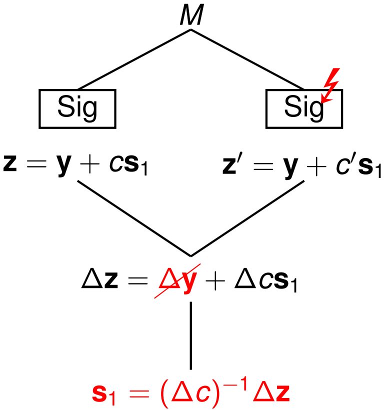
# Attack Intuition

- Sign same $M$ twice
  1. no fault: $(\mathbf{z}, c)$
  2. fault s.t. same $\mathbf{y}$ but different $c$: $(\mathbf{z}', c')$
- Problem with $\mathbf{s}_2$: key compression!
  - attacker can forge with $\mathbf{s}_1$ only



# Attack Intuition

- Sign same $M$ twice
  1. no fault: $(\mathbf{z}, c)$
  2. fault s.t. same $\mathbf{y}$ but different $c$: $(\mathbf{z}', c')$
- Problem with $\mathbf{s}_2$: key compression!
  - attacker can forge with $\mathbf{s}_1$ only
- . . . wasn't there something else?



# Rejection hurts...

- $\mathbf{y} := \text{DeterministicSample}(K||M||\kappa{+}{+})$
  - "rejection counter" $\kappa$ for fresh **y** in iterations
  - same **y** requires same $\kappa$
  - key recovery successful $\iff$ accept in same iteration

# Rejection hurts. . .

- $\mathbf{y} := \text{DeterministicSample}(K||M||\kappa{+}{+})$
  - “rejection counter” $\kappa$ for fresh **y** in iterations
  - same **y** requires same $\kappa$
  - key recovery successful $\iff$ accept in same iteration
- Faults influence intermediates used in rejection conditions
  - fault position determines success probability

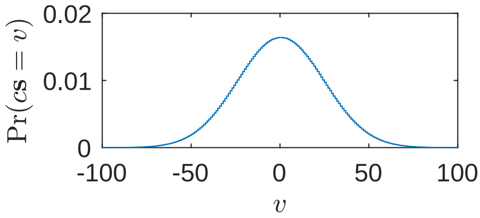
# Rejection hurts...

- **y** $:=$ DeterministicSample($K||M||\kappa{+}{+}$)
  - "rejection counter" $\kappa$ for fresh **y** in iterations
  - same **y** requires same $\kappa$
  - key recovery successful $\iff$ accept in same iteration
- Faults influence intermediates used in rejection conditions
  - fault position determines success probability
- 5 fault scenarios (concrete positions)

# Faulting the Hash: fH

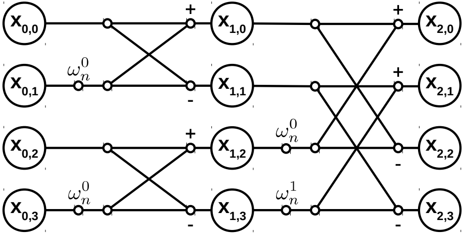
- Target: $c := \mathrm{H}(M||\mathbf{w}_1)$
- Observation: $\|\mathbf{y}\| \gg \|c\mathbf{s}_1\|$
  - $\mathbf{z} = \mathbf{y} + c\mathbf{s}_1 \approx \mathbf{y}$
  - in other words $\mathbf{z} \approx \mathbf{z}'$
- Success probability: 91%

$y \in [\pm 523\,776]$



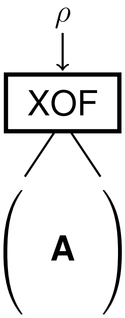
# Faulting Polynomial Multiplication: fW

- Target: $\mathbf{w} := \mathbf{Ay}$
  - $c := \mathrm{H}(M \| \mathrm{HighBits}(\mathbf{w}))$
- Multiplication using NTT
  - fault early $\rightarrow$ many output coefficients affected
- Success probability: $25 - 90\%$
  - but a “larger” target



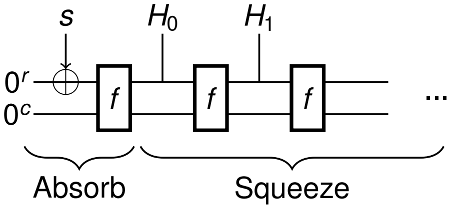
# Faulting the Public Key: $fA_E$, $fA_\rho$

- Target: loading of **A**
  - $c := H(M \| \text{HighBits}(\mathbf{Ay}))$
- **A** generated from seed $\rho$
  - attack $\rho$ or expansion
- Success probability: $25 - 54\%$
  - but potentially permanent



# Faulting the Nonce: fY

- Target: $\mathbf{y} := \text{DeterministicSample}(K||M||\kappa{+}{+})$
  - $c := \text{H}(M||\text{HighBits}(\mathbf{Ay}))$
- but then no nonce-reuse anymore...
  - target partial reuse: $\mathbf{y} \approx \mathbf{y}'$
- Sampling uses SHAKE XOF
  - fault Keccak-$f$ application in squeeze
  - previous XOF output unchanged



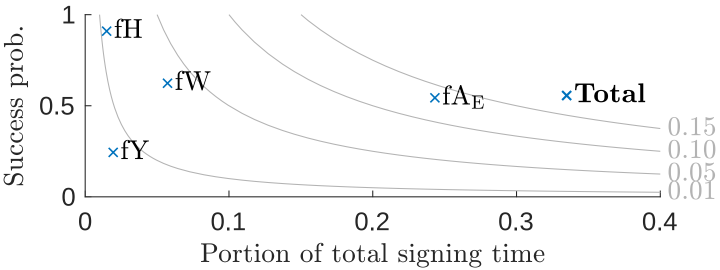
# Efficient Exploitation of Partial Reuse

- First $n - v$ coefficients are 0
  - brute-force still infeasible
- Key recovery as a lattice problem
  - $\mathbf{t} = (\Delta c)^{-1}\Delta\mathbf{z} = (\Delta c)^{-1}\Delta\mathbf{y} + \mathbf{s}_1$
  - vector close to $\mathbf{t}$ in the lattice generated by $(\Delta c)^{-1}$
- Can fault last 2 (from 5) Keccak-*f* permutations
  - recovery runtime: $< 1$ minute
  - success probability: 24%

$$\Delta\mathbf{y} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ \Delta y_v \\ \Delta y_{v+1} \\ \vdots \\ \Delta y_{n-1} \end{pmatrix}$$

# Experimental Verification

- Clock glitches @ ARM Cortex M4
  - single random fault



# Countermeasures

| | $fA_\rho$ | $fA_E$ | fY | fW | fH | Runtime |
|---|---|---|---|---|---|---|
| Double computation | ✗ | ✓ | ✓ | ✓ | ✓ | +100% |
| Verification-after-sign | ✓ | ✓ | ✗ | ✓ | ✓ | +25% |
| Additional randomness | ✓ | ✓ | ✓ | ✓ | ✓ | +0% |

# Countermeasures

| | $fA_\rho$ | $fA_E$ | fY | fW | fH | Runtime |
|---|---|---|---|---|---|---|
| Double computation | ✗ | ✓ | ✓ | ✓ | ✓ | $+100\%$ |
| Verification-after-sign | ✓ | ✓ | ✗ | ✓ | ✓ | $+25\%$ |
| Additional randomness | ✓ | ✓ | ✓ | ✓ | ✓ | $+0\%$ |

- Additional randomness: $\mathbf{y} := \text{DeterministicSample}(K||M||\kappa{+}{+}||r)$
  - protects against faults, bad randomness, and DPA-recovery of $K$ [SHS16]
  - qTESLA: recent update, countermeasure now mandatory [BAA+17]
  - Dilithium: not compatible with proof… [KLS18]

# Differential Fault Attacks on Deterministic Lattice Signatures

**Leon Groot Bruinderink[1], Peter Pessl[2]**
**[1]Technische Universiteit Eindhoven, [2]Graz University of Technology**

CHES 2018, September 10

# Bibliography I

[ABF+18] Christopher Ambrose, Joppe W. Bos, Björn Fay, Marc Joye, Manfred Lochter, and Bruce Murray. Differential Attacks on Deterministic Signatures. In *CT-RSA*, volume 10808 of *LNCS*, pages 339–353. Springer, 2018.

[BAA+17] Nina Bindel, Sedat Akleylek, Erdem Alkim, Paulo S. L. M. Barreto, Johannes Buchmann, Edward Eaton, Gus Gutoski, Juliane Krämer, Patrick Longa, Harun Polat, Jefferson E. Ricardini, and Gustavo Zanon. qTESLA. Submission to the NIST Post-Quantum Cryptography Standardization [NIS], 2017. `https://qtesla.org`.

[KLS18] Eike Kiltz, Vadim Lyubashevsky, and Christian Schaffner. A Concrete Treatment of Fiat-Shamir Signatures in the Quantum Random-Oracle Model. In *EUROCRYPT (3)*, volume 10822 of *LNCS*, pages 552–586. Springer, 2018.

[NIS] NIST. Post-Quantum Cryptography Standardization. `https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization`.

[PSS+17] Damian Poddebniak, Juraj Somorovsky, Sebastian Schinzel, Manfred Lochter, and Paul Rösler. Attacking Deterministic Signature Schemes using Fault Attacks. Cryptology ePrint Archive, Report 2017/1014, 2017.

[SHS16] Hermann Seuschek, Johann Heyszl, and Fabrizio De Santis. A Cautionary Note: Side-Channel Leakage Implications of Deterministic Signature Schemes. In *CS2@HiPEAC*, pages 7–12. ACM, 2016.