NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

Exploiting Determinism in Lattice-based Signatures - Practical Fault Attacks on pqm4 Implementations of NIST candidates

Prasanna Ravi, Mahabir Prasad Jhanwar, James Howe, Anupam Chattopadhyay, Shivam Bhasin

ACM AsiaCCS-2019
10th July 2019

Table of Contents

Table of Contents

Context

- Huge money in quantum computing is being invested by computer industry giants like Google, IBM, Intel and other companies like D-Wave, IonQ.
- The most powerful universal gate quantum computer: 160 physical qbits from IonQ.
- How many qubits do we need to break RSA-2048??

Context

- Huge money in quantum computing is being invested by computer industry giants like Google, IBM, Intel and other companies like D-Wave, IonQ.
- The most powerful universal gate quantum computer: 160 physical qbits from IonQ.
- How many qubits do we need to break RSA-2048?? 4096 logical qubits $\leftarrow$ Millions of physical qubits

Context

- Huge money in quantum computing is being invested by computer industry giants like Google, IBM, Intel and other companies like D-Wave, IonQ.
- The most powerful universal gate quantum computer: 160 physical qbits from IonQ.
- How many qubits do we need to break RSA-2048?? 4096 logical qubits $\leftarrow$ Millions of physical qubits
- NIST process for standardization of Post-Quantum Cryptography (PQC) is underway.

Context

- Huge money in quantum computing is being invested by computer industry giants like Google, IBM, Intel and other companies like D-Wave, IonQ.
- The most powerful universal gate quantum computer: 160 physical qbits from IonQ.
- How many qubits do we need to break RSA-2048?? 4096 logical qubits $\leftarrow$ Millions of physical qubits
- NIST process for standardization of Post-Quantum Cryptography (PQC) is underway.
- Started in December 2017, 3-5 years analysis period, followed by 2 years for draft standards.

NIST PQC Call

Type	Signatures	KEM/Encryption	Overall
Lattice-based	5	23	28
Code-based	3	17	20
Multivariate	8	2	10
Hash-based	3	0	3
Isogeny-based	0	1	1
Others	2	5	7
Total	21	48	69

NIST PQC Call

Type	Signatures	KEM/Encryption	Overall
Lattice-based	3	9	12
Code-based	0	7	7
Multivariate	4	0	4
Hash-based	2	-	2
Isogeny-based	0	1	1
Others	0	0	0
Total	9	17	26

NIST PQC Call

Type	Signatures	KEM/Encryption	Overall
Lattice-based	2	5	7
Code-based	0	3	3
Multivariate	2	0	2
Hash-based	2	0	2
Isogeny-based	0	1	1
Others	0	0	0
Total	6	9	15

This Work

- Practical fault attacks against two *deterministic* lattice-based signature schemes, Dilithium and qTESLA
- Demonstration of practicality of *skip-addition* fault attacks proposed by Bindel *et al.* [1] through exploitation determinism in lattice-based signature schemes.
- Signature forgery algorithm for Dilithium using retrieved part of the secret key.
- Experimental validation through Electromagnetic fault injection on implementations taken from the *pqm4*, open-source benchmarking and testing framework for PQC schemes on the ARM Cortex-M4 microcontroller.

This Work

- We show that two well known countermeasures known to protect against *skip-addition* fault attacks can be defeated. This was also made possible owing to the *deterministic* nature of Dilithium.
- We also propose a *zero-cost* mitigation approach against our attack which exponentially increases attacker's complexity.

Table of Contents

Lattice-based Cryptography

- Based on hard problems over geometric structures called "lattices" in n-dimensional space
- Shortest Vector Problem (SVP), Closes Vector Problem (CVP), Bounded Distance Decoding (BDD) problem
- **Unique** and **Strong** Security Guarantees: Average case hard problems are as hard as worst-case instances of hard problems in lattices
- **Good** Efficiency Guarantees: Computations over polynomials in rings
- This makes lattice-based cryptographic schemes, one of the leading candidates in the ongoing NIST standardization process for post-quantum cryptography.

Learning With Errors Problem (LWE)

- Well known Average case problem based on which multiple lattice-based schemes are built.
- Let $\mathbf{A} \in \mathbb{Z}_q^{n \times n}$ and $\mathbf{S}, \mathbf{E} \in \mathbb{Z}_q^n \leftarrow D_\sigma$
- $\mathbf{T} = (\mathbf{A} \times \mathbf{S} + \mathbf{E}) \in \mathbb{Z}_q^n$
- **Search LWE**: Given several pairs $(\mathbf{A}, \mathbf{T})$, find $\mathbf{S}$.
- **Decisional LWE**: Distinguish between valid LWE pairs $(\mathbf{A}, \mathbf{T})$ from uniformly random samples in $(\mathbb{Z}_q^{n \times n} \times \mathbb{Z}_q^n)$.
- More efficient variants of LWE known as Ring-LWE and Module (LWE) which involve computation over polynomials in rings.
- Ring-LWE: $R_q = \mathbb{Z}_q[X]/(X^n + 1)$ with $\mathbf{A}, \mathbf{S}, \mathbf{E} \in \mathbf{R}_q$.
- Module-LWE: $R_q^{k \times l} = (\mathbb{Z}_q[X]/(X^n + 1))^{k \times l}$ with $\mathbf{A} \in \mathbf{R}_q^{k \times \ell}$, $\mathbf{S} \in \mathbf{R}_q^{\ell}$, $\mathbf{E} \in \mathbf{R}_q^{k}$.

Fault Injection Attacks

- Intentional manipulation to cause physical disturbance which corrupts the behavior of device that runs cryptographic implementations.
- Attacker analyzes faulty outputs to derive relation with the secret key.
- Intentional Faults can be induced through multiple techniques:
  - Clock glitch, Voltage glitch
  - Underpowering/Temperature
  - Optical/Laser Fault Injection
  - Electromagnetic Fault Injection

Electromagnetic Fault Injection

- Injection of high voltage and short electromagnetic pulses through micro-probes directly onto the chip.
- Loops inside the chip act as antennas and cause the EM pulses to create additional "Eddy-currents".
- These additional currents cause unexpected changes to the normal behaviour of the device.
- **Advantages:**
  - Low-cost
  - Non/Semi-invasive approach.
  - Localized Effect $\rightarrow$ Controllability

Dilithium Signature Scheme

- Dilithium is a lattice-based signature scheme based on the hardness of MLWE and MSIS problems.
- Computation over modules in $R_q^{k \times l}$ (Matrices/Vectors of polynomials).
- Base Ring: $\mathbf{R}_q = \mathbb{Z}_q[X]/(X^n + 1)$ with n = 256 and q = $2^{23} - 2^{13} + 1$.
- Built upon the "Fiat-Shamir with Aborts" framework.
- Signing operation is iterative and repeated until signatures satisfy a certain condition.

Dilithium Signature Scheme

procedure DILITHIUM.KEYGEN()

  $\rho, \rho' \leftarrow \{0,1\}^{256}$, $K \leftarrow \{0,1\}^{256}$, $N := 0$

For i from 0 to $\ell - 1$

  $\mathbf{s}_1[i] := Sample(PRF(\rho', N))$

  $N := N + 1$

EndFor

For i from 0 to $k - 1$

  $\mathbf{s}_2[i] := Sample(PRF(\rho', N))$

  $N := N + 1$

EndFor

  $\mathbf{A} \sim R_q^{k \times \ell} := \mathsf{ExpandA}(\rho)$

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

  $\mathbf{t}_1 := \mathsf{Power2Round}_q(\mathbf{t}, d)$

  $tr \in \{0,1\}^{384} := \mathsf{CRH}(\rho || \mathbf{t}_1)$

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

end procedure

Dilithium Signature Scheme

procedure DILITHIUM.SIGN(sk, M)
  $\mathbf{A} \in R_q^{k\times\ell} :=$ ExpandA(ρ)
  $\mu =$ CRH(tr$\|M$)
  $\kappa = 0, (\mathbf{z}, \mathbf{h}) = \perp$
While($(\mathbf{z}, \mathbf{h}) = \perp$)
  $\rho = (K\|\mu)$ ($\rho \leftarrow \{0,1\}^{384}$ for the randomized variant)
  $\mathbf{y} \in S_{\gamma_1-1}^{\ell} :=$ ExpandMask($\rho\|\kappa$)
  $\mathbf{w} = \mathbf{A} \cdot \mathbf{y}$
  $\mathbf{w}_1 = \mathsf{HB}_q(\mathbf{w}, 2\gamma_2)$
  $\mathbf{c} \in B_{60} = H(\mu\|\mathbf{w}_1)$
  $\mathbf{z} = \mathbf{y} + \mathbf{c} \cdot \mathbf{s}_1$
  $(\mathbf{r}_1, \mathbf{r}_0) := \mathsf{D}_q(\mathbf{w} - \mathbf{c} \cdot \mathbf{s}_2, 2\gamma_2)$
If($\|\mathbf{z}\|_\infty \geq \gamma_1 - \beta$ or $\|\mathbf{r}_0\|_\infty \geq \gamma_2 - \beta$ or $\mathbf{r}_1 \neq \mathbf{w}_1$)
  $(\mathbf{z}, \mathbf{h}) = \perp$
Else
  $\mathbf{h} = \mathsf{MH}_q(-\mathbf{c} \cdot \mathbf{t}_0, \mathbf{w} - \mathbf{c} \cdot \mathbf{s}_2 + \mathbf{c} \cdot \mathbf{t}_0, 2\gamma_2)$
If($\|\mathbf{c} \cdot \mathbf{t}_0\|_\infty \geq \gamma_2$ or wt($\mathbf{h}$) $> \omega$)
  $(\mathbf{z}, \mathbf{h}) = \perp$
EndIf $\kappa = \kappa + 1$
EndWhile
  Return $\sigma = (\mathbf{z}, \mathbf{h}, \mathbf{c})$
end procedure

Dilithium Signature Scheme

procedure DILITHIUM.SIGN(sk, M)
    $\mathbf{A} \in R_q^{k \times \ell} :=$ ExpandA(ρ)
    $\mu =$ CRH(tr$\|M$)
    $\kappa = 0, (\mathbf{z}, \mathbf{h}) = \perp$
While($(\mathbf{z}, \mathbf{h}) = \perp$)
    $\rho = (K\|\mu)$ ($\rho \leftarrow \{0,1\}^{384}$ for the randomized variant)
    $\mathbf{y} \in S_{\gamma_1 - 1}^{\ell} :=$ ExpandMask($\rho\|\kappa$)
    $\mathbf{w} = \mathbf{A} \cdot \mathbf{y}$
    $\mathbf{w}_1 = \mathsf{HB}_q(\mathbf{w}, 2\gamma_2)$
    $\mathbf{c} \in B_{60} = H(\mu\|\mathbf{w}_1)$
    $\mathbf{z} = \mathbf{y} + \mathbf{c} \cdot \mathbf{s}_1$
    $(\mathbf{r}_1, \mathbf{r}_0) := \mathsf{D}_q(\mathbf{w} - \mathbf{c} \cdot \mathbf{s}_2, 2\gamma_2)$
If($\|\mathbf{z}\|_\infty \geq \gamma_1 - \beta$ or $\|\mathbf{r}_0\|_\infty \geq \gamma_2 - \beta$ or $\mathbf{r}_1 \neq \mathbf{w}_1$)
    $(\mathbf{z}, \mathbf{h}) = \perp$
Else
    $\mathbf{h} = \mathsf{MH}_q(-\mathbf{c} \cdot \mathbf{t}_0, \mathbf{w} - \mathbf{c} \cdot \mathbf{s}_2 + \mathbf{c} \cdot \mathbf{t}_0, 2\gamma_2)$
If($\|\mathbf{c} \cdot \mathbf{t}_0\|_\infty \geq \gamma_2$ or wt($\mathbf{h}$) $> \omega$)
    $(\mathbf{z}, \mathbf{h}) = \perp$
EndIf $\kappa = \kappa + 1$
EndWhile
    Return $\sigma = (\mathbf{z}, \mathbf{h}, \mathbf{c})$
end procedure

Dilithium Signature Scheme

procedure DILITHIUM.SIGN(sk, M)
   $\mathbf{A} \in R_q^{k\times\ell} :=$ ExpandA(ρ)
   $\mu =$ CRH$(\text{tr}\|M)$
   $\kappa = 0, (\mathbf{z}, \mathbf{h}) = \perp$
While$((\mathbf{z}, \mathbf{h}) = \perp)$
   $\rho = (K\|\mu)$ $(\rho \leftarrow \{0,1\}^{384}$ for the randomized variant$)$
   $\mathbf{y} \in S_{\gamma_1-1}^{\ell} :=$ ExpandMask$(\rho\|\kappa)$
   $\mathbf{w} = \mathbf{A}\cdot\mathbf{y}$
   $\mathbf{w}_1 = \text{HB}_q(\mathbf{w}, 2\gamma_2)$
   $\mathbf{c} \in B_{60} = H(\mu\|\mathbf{w}_1)$
   $\mathbf{z} = \mathbf{y} + \mathbf{c}\cdot\mathbf{s}_1$
   $(\mathbf{r}_1, \mathbf{r}_0) := \text{D}_q(\mathbf{w} - \mathbf{c}\cdot\mathbf{s}_2, 2\gamma_2)$
If$(\|\mathbf{z}\|_\infty \geq \gamma_1 - \beta$ or $\|\mathbf{r}_0\|_\infty \geq \gamma_2 - \beta$ or $\mathbf{r}_1 \neq \mathbf{w}_1)$
   $(\mathbf{z}, \mathbf{h}) = \perp$
Else
   $\mathbf{h} = \text{MH}_q(-\mathbf{c}\cdot\mathbf{t}_0, \mathbf{w} - \mathbf{c}\cdot\mathbf{s}_2 + \mathbf{c}\cdot\mathbf{t}_0, 2\gamma_2)$
If$(\|\mathbf{c}\cdot\mathbf{t}_0\|_\infty \geq \gamma_2$ or wt$(\mathbf{h}) > \omega)$
   $(\mathbf{z}, \mathbf{h}) = \perp$
EndIf $\kappa = \kappa + 1$
EndWhile
   Return $\sigma = (\mathbf{z}, \mathbf{h}, \mathbf{c})$
end procedure

Table of Contents

Adversary Model

- Attacker has complete physical access to the device.
- Trigger the device into computing signatures for the message of the attacker's choice.
- Attacker should have access to the computed signatures.

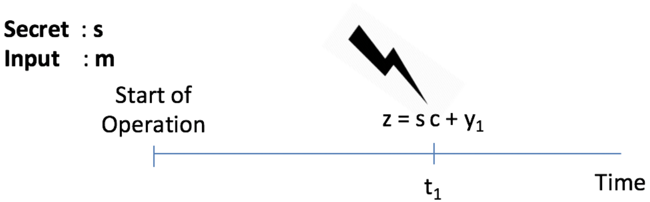
Skip-Addition attacks on lattice-based signature schemes

- Generation of the primary signature component $\mathbf{z}$ has been the main target of most reported fault attacks [1, 2].
- $\mathsf{z_{gen}}$: $\mathbf{z} = \mathbf{s} \cdot \mathbf{c} + \mathbf{y} \in R_q$
- Bindel *et al.*[1] proposed the first targeted *Skip-Addition* attacks on a number of lattice-based signature schemes following the same framework.
- Base Idea: Skip the final addition in $\mathsf{z_{gen}}$ to obtain the following faulty output:
- $\hat{\mathsf{z}}_{\mathsf{gen}}$: $\hat{\mathbf{z}} = \mathbf{s} \cdot \mathbf{c} \in R_q$
- $\hat{\mathsf{z}}_{\mathsf{gen}}$: System of N linear equations with N "unknowns" - Solve for coefficients of $\mathbf{s}$ using Gaussian Elimination

Problems with the *Skip-Addition* attack

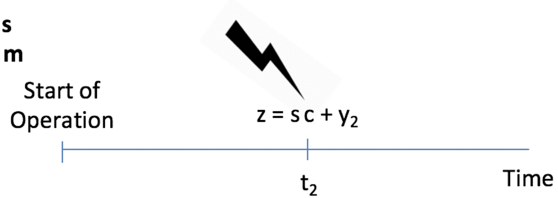
- Requires to skip addition corresponding to all coefficients of $\mathbf{z}$ (numbering in the hundreds).
- Requires several hundreds of precisely targeted faults within single run of the signing procedure.
- From a practical perspective: *Totally infeasible*.
- *Attacker synchronization* very difficult if not impossible in case of probabilistic schemes.
- The ephemeral nonce $\mathbf{y}$ changes for every run of the signing procedure.

Problems with the *Skip-Addition* attack



Problems with the *Skip-Addition* attack

Secret : s
Input : m



Problems with the *Skip-Addition* attack

Problems with the *Skip-Addition* attack

Problems with the *Skip-Addition* attack

- If the faulted computation resulted in $\mathbf{z} = \mathbf{y}$, then the attack does not work on probabilistic signature schemes.
- This was proposed as a potential countermeasure against *Skip-Addition* fault attacks [1].
- Three Problems:
  - Large Number of Faults
  - Attacker Synchronization
  - Simple Countermeasure
- We will show that *determinism* can be exploited to easily defeat the above problems to perform practical fault attacks on Dilithium and qTESLA.

Main Attack Idea

- Fault individual coefficients of $\mathbf{z}$ **one at a time** and aggregate information over **multiple** faulty signatures to obtain the secret key $\mathbf{s}$.
- We consider two cases based on the order of operands in the addition operation in $\mathsf{z_{gen}}$.
- Case-1:

$$\mathbf{z} = \mathbf{s} \cdot \mathbf{c}$$

$$\mathbf{z} = \mathbf{z} + \mathbf{y}$$

Main Attack Idea

- Fault individual coefficients of $\mathbf{z}$ **one at a time** and aggregate information over **multiple** faulty signatures to obtain the secret key $\mathbf{s}$.
- We consider two cases based on the order of operands in the addition operation in $\mathsf{z_{gen}}$.
- Case-1:

$$\mathbf{z} = \mathbf{s} \cdot \mathbf{c}$$

$$\mathbf{z} = \mathbf{z} + \mathbf{y}$$

Main Attack Idea

- Fault individual coefficients of $\mathbf{z}$ **one at a time** and aggregate information over **multiple** faulty signatures to obtain the secret key $\mathbf{s}$.
- We consider two cases based on the order of operands in the addition operation in $\mathsf{z}_{\mathsf{gen}}$.
- Case-1:

$$\mathbf{z} = \mathbf{s} \cdot \mathbf{c}$$

$$\mathbf{z} = \mathbf{z} + \mathbf{y}$$

- $(\hat{\mathbf{z}})_t = (\mathbf{s} \cdot \mathbf{c})_t$ for $t \in \{0, N-1\}$

Main Attack Idea

- Fault individual coefficients of $\mathbf{z}$ **one at a time** and aggregate information over **multiple** faulty signatures to obtain the secret key $\mathbf{s}$.
- We consider two cases based on the order of operands in the addition operation in $\mathsf{z}_{\mathsf{gen}}$.
- Case-1:

$$\mathbf{z} = \mathbf{s} \cdot \mathbf{c}$$

$$\mathbf{z} = \mathbf{z} + \mathbf{y}$$

- $(\hat{\mathbf{z}})_t = (\mathbf{s} \cdot \mathbf{c})_t$ for $t \in \{0, N-1\}$

Main Attack Idea

- Fault individual coefficients of $\mathbf{z}$ **one at a time** and aggregate information over **multiple** faulty signatures to obtain the secret key $\mathbf{s}$.
- We consider two cases based on the order of operands in the addition operation in $\mathsf{z_{gen}}$.
- Case-1:

$$\mathbf{z} = \mathbf{s} \cdot \mathbf{c}$$

$$\mathbf{z} = \mathbf{z} + \mathbf{y}$$

- $(\hat{\mathbf{z}})_t = (\mathbf{s} \cdot \mathbf{c})_t$ for $t \in \{0, N-1\}$

Main Attack Idea

- Fault individual coefficients of $\mathbf{z}$ **one at a time** and aggregate information over **multiple** faulty signatures to obtain the secret key $\mathbf{s}$.
- We consider two cases based on the order of operands in the addition operation in $\mathsf{z}_{\mathsf{gen}}$.
- Case-1:

$$\mathbf{z} = \mathbf{s} \cdot \mathbf{c}$$

$$\mathbf{z} = \mathbf{z} + \mathbf{y}$$

- $(\hat{\mathbf{z}})_t = (\mathbf{s} \cdot \mathbf{c})_t$ for $t \in \{0, N-1\}$

Main Attack Idea

- Fault individual coefficients of $\mathbf{z}$ **one at a time** and aggregate information over **multiple** faulty signatures to obtain the secret key $\mathbf{s}$.
- We consider two cases based on the order of operands in the addition operation in $\mathsf{z}_{\mathsf{gen}}$.
- Case-1:

$$\mathbf{z} = \mathbf{s} \cdot \mathbf{c}$$
$$\mathbf{z} = \mathbf{z} + \mathbf{y}$$

- $(\hat{\mathbf{z}})_t = (\mathbf{s} \cdot \mathbf{c})_t$ for $t \in \{0, N-1\}$

Main Attack Idea

- The attacker similarly faults the addition operation corresponding to the other $N-1$ coefficients to obtain all the coefficients of $\mathbf{s}\cdot\mathbf{c}$.
- Since $\mathbf{c}$ is known, $\mathbf{s}$ can be recovered through Gaussian elimination.

Main Attack Idea

- Case-2:

$$\mathbf{z} = \mathbf{y}$$
$$\mathbf{z} = \mathbf{z} + \mathbf{s} \cdot \mathbf{c}$$

- $(\hat{\mathbf{z}})_t = (\mathbf{y})_t$ for $t \in \{0, N-1\}$
- But, since the attacker has access to the correct coefficient of $\mathbf{z}$ (i.e) $\mathbf{z}_t = (\mathbf{y})_t + (\mathbf{s} \cdot \mathbf{c})_t$, he can compute $(\mathbf{s} \cdot \mathbf{c})_t$ as follows:
- $(\mathbf{s} \cdot \mathbf{c})_t = (\mathbf{z})_t - (\hat{\mathbf{z}})_t$
- The attacker similarly faults the addition operation corresponding to the other $N-1$ coefficients to obtain all the coefficients of $\mathbf{s} \cdot \mathbf{c}$.
- Since $\mathbf{c}$ is known, $\mathbf{s}$ can be recovered through Gaussian elimination.

How Determinism helps the fault attacker?

- Attacking individual coefficients is possible because the time instance of operation remains the same, given the same inputs.
- Determinism also allows to compare the correct and faulty outputs corresponding to the same inputs.

Forging signatures for Dilithium

- Through the fault attack, we can recover $\mathbf{s}_1$.
- Moreover, there are other components of the secret key since $sk = (\mathbf{s}_1, \mathbf{s}_2, K, tr, \mathbf{t}_0)$.
- The scheme does not prove knowledge of K, tr and hence attacker can use random K, tr.
- Retrieval of $\mathbf{s}_1$ enables us to create $(\mathbf{z}, \mathbf{c})$.
- But, the attacker still needs to construct the hint vector $\mathbf{h}$ since $\sigma = (\mathbf{z}, \mathbf{h}, \mathbf{c})$.
- Thus, the attacker has to bypass use of $\mathbf{s}_2, \mathbf{t}_0$.

Forging signatures for Dilithium

- We also found that the scheme also does not really prove knowledge of $\mathbf{s}_2$ since the public key is a rounded off version of the LWE instance.
- We were able to reverse-calculate the remaining hint vector $\mathbf{h}$ just based on the knowledge of $\mathbf{s}_1$.
- The $\mathbf{s}_1$ component of the secret key is the most crucial with respect of security of Dilithium signature scheme.

Table of Contents

Experimental Validation on ARM Cortex-M4

- We target reference implementations of Dilithium and qTESLA from the *pqm4* benchmarking framework for PQC candidates on the ARM Cortex-M4 microcontroller.
- Implementations were ported to the STM32F4DISCOVERY board (DUT) housing the STM32F407 microcontroller.
- Clock Frequency: 24 MHz.
- We used Electromagnetic Fault Injection (EMFI) to induce transient faults into the device.

Experimental Setup

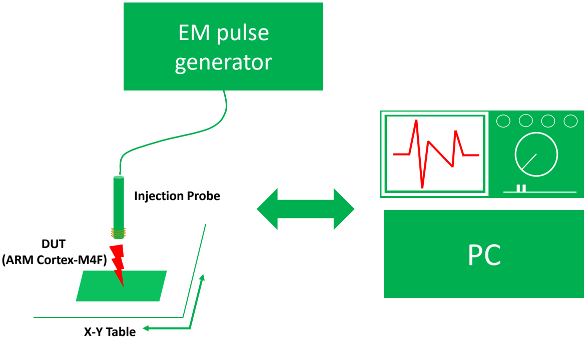


Figure: Description of our EMFI setup

Experimental Setup

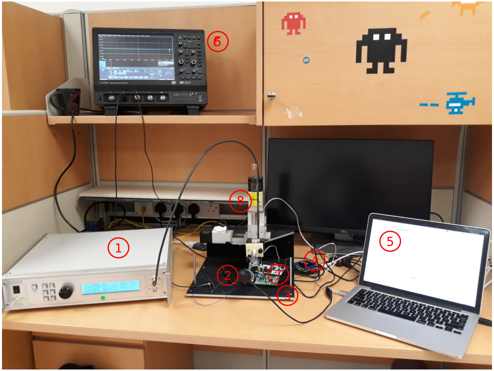
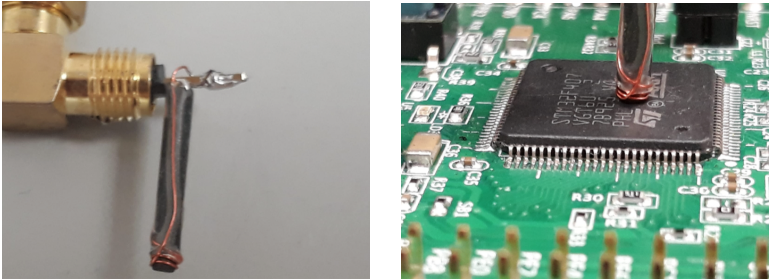


Figure: (1) EM Pulse Generator (2) USB-Microscope (3) STM32M4F Discovery Board (DUT) (4) Arudino based Relay Shield (5) Controller Laptop (6) Oscilloscope (7) EM Pulse Injector (8) XYZ Motorized Table

Experimental Setup



(a) (b)

Figure: (a) Hand-made probe used for our EMFI setup (b) Probe placed over the DUT

Analysis of implementation for Fault Vulnerability

- Precise identification of instruction to be targeted and the required fault model.
- We consider three different variants of the z_{gen} operation.
  - Variant-1: Adding $\mathbf{y}$ to $\mathbf{z} = \mathbf{s} \cdot \mathbf{c}$
  - Variant-2: Adding $\mathbf{s} \cdot \mathbf{c}$ to $\mathbf{z} = \mathbf{y}$
  - Variant-3: Prevent overwriting the result onto either $\mathbf{y}$ or $\mathbf{s} \cdot \mathbf{c}$
- While the first two variants are based on the order of the operands, the third variant is based on writing the result of the addition to a new variable.

Variant-1: Adding $\mathbf{y}$ to $\mathbf{z} = \mathbf{s} \cdot \mathbf{c}$

```
/* Sampling y */
for(i = 0; i < L; ++i)
  poly_unif_gamma1m1(y.vec+i,key,nonce++);
/* Computing NTT(y) */
yhat = y;
polyvecl_ntt(&yhat);
/* Computing NTT(c) */
chat = c;
poly_ntt(&chat);
/* Computing product sc */
for(i = 0; i < L; ++i)
{
  poly_ptwise_imont(z.vec+i,&chat,s1.vec+i);
  poly_intt_mont(z.vec+i);
}
/* Last addition to generate z */
/* (y added to sc) */
polyvecl_add(&z,&y,&z);
```

Variant-1: Adding $\mathbf{y}$ to $\mathbf{z} = \mathbf{s} \cdot \mathbf{c}$

```
LDR.W   r3,[r4,#4]!
LDR.W   r1,[r2,#4]!
CMP     r4,r5
ADD     r3, r1
/* Target store operation */
STR.W   r3, [r0, #4]!
```

- `LDR.W r3,[r4,#4]!` → Load $z_t = sc_t$
- `LDR.W r1,[r2,#4]!` → Load y_t
- `ADD r3, r1` → Add z_t, y_t
- `STR.W r3, [r0, #4]!` → Store result in z_t

Variant-1: Adding $\mathbf{y}$ to $\mathbf{z} = \mathbf{s} \cdot \mathbf{c}$

```
LDR.W   r3,[r4,#4]!
LDR.W   r1,[r2,#4]!
CMP     r4,r5
ADD     r3, r1
/* Target store operation */
STR.W   r3, [r0, #4]!
```

Load $z_t = sc_t$

Load y_t

Add z_t, y_t

Store result in z_t

Variant-2: Adding $\mathbf{s} \cdot \mathbf{c}$ to $\mathbf{z} = \mathbf{y}$

```
/* Sampling y in z */
for(i = 0; i < L; ++i)
  poly_unif_gamma1m1(z.vec+i,key,nonce++);
/* Computing NTT(y) */
yhat = z;
polyvecl_ntt(&yhat);
/* Computing NTT(c) */
chat = c;
poly_ntt(&chat);
/* Computing product sc */
for(i = 0; i < L; ++i)
{
  poly_ptwise_imont(sc.vec+i,&chat,s1.vec+i);
  poly_intt_mont(sc.vec+i);
}
/* Last addition to generate z */
/* (sc added to y) */
polyvecl_add(&z,&sc,&z);
```

Variant-2: Adding $\mathbf{s} \cdot \mathbf{c}$ to $\mathbf{z} = \mathbf{y}$

`LDR.W r3,[r4,#4]!`	Load $z_t = y_t$
`LDR.W r1,[r2,#4]!`	Load sc_t
`CMP r4,r5`	
`ADD r3, r1`	Add z_t, sc_t
`/* Target store operation */`	
`STR.W r3, [r0, #4]!`	Store result in z_t

Variant-2: Adding $\mathbf{s} \cdot \mathbf{c}$ to $\mathbf{z} = \mathbf{y}$

```
LDR.W   r3,[r4,#4]!
LDR.W   r1,[r2,#4]!
CMP     r4,r5
ADD     r3, r1
/* Target store operation */
STR.W   r3, [r0, #4]!
```

Load $z_t = y_t$

Load sc_t

Add z_t, sc_t

Store result in z_t

Variant-3: Prevent overwriting the result onto either $\mathbf{y}$ or $\mathbf{s} \cdot \mathbf{c}$

```
/* Sampling y */
for(i = 0; i < L; ++i)
  poly_unif_gamma1m1(y.vec+i,key,nonce++);
/* Computing NTT(y) */
yhat = y;
polyvecl_ntt(&yhat);
/* Computing NTT(c) */
chat = c;
poly_ntt(&chat);
/* Computing product sc */
/* Result stored in ztemp */
for(i = 0; i < L; ++i)
{
  poly_ptwise_imont(ztemp.vec+i,&chat,s1.vec+i);
  poly_intt_mont(ztemp.vec+i);
}
    /* Last addition to generate z */
    /* Result stored in new variable z */
    polyvecl_add(&z,&y,&ztemp);
```

Variant-3: Prevent overwriting the result onto either $\mathbf{y}$ or $\mathbf{s} \cdot \mathbf{c}$

`LDR.W r3,[r4,#4]!`	Load $z_t = y_t$
`LDR.W r1,[r2,#4]!`	Load sc_t
`CMP r4,r5`	
`ADD r3, r1`	Add z_t, sc_t
`/* Target store operation */`	
`STR.W r3, [r0, #4]!`	Store result in $ztemp_t$

Variant-3: Prevent overwriting the result onto either $\mathbf{y}$ or $\mathbf{s} \cdot \mathbf{c}$

```
LDR.W   r3,[r4,#4]!
LDR.W   r1,[r2,#4]!
CMP     r4,r5
ADD     r3, r1
/* Target store operation */
STR.W   r3, [r0, #4]!
```

Load $z_t = y_t$

Load sc_t

Add z_t, sc_t

Store result in $ztemp_t$

Systematic Approach towards Targeted Fault Injection

- Our attack requires to inject targeted faults at specific instructions.
- How do we identify the time instance to fault?
- We use the EM/power side-channel and exploit determinism in computations to precisely identify the time-instance to inject fault.
- EM measurements are observed from the same DUT using a near field probe and processed using a digital oscilloscope.

Results on ARM Cortex-M4

- **Required Fault:**
  - Variant-1&2: *Skip-Store* fault
  - Variant-3: *Skip-Add* fault
- Profiled the ARM device to identify a fault sensitive region - Area on top of the "A" of the ARM logo of the STM32M4F07 microcontroller.
- Achieved fault repeatability of almost 100% at the identified location for effectively skipping the store instruction.
- Voltage:150V-200V, Pulse Width = 12ns, Rise-Time = 2 ns.

Results on ARM Cortex-M4

- But, we were not able to achieve faults to precisely skip only the add instruction with the current setup.
- But, a more powerful attacker with enhanced fault injection capabilities can possibly mount an attack on the Variant-3 implementation as well.

Table of Contents

Zero-Cost Mitigation

- **Generic Countermeasures**: Double computation, Verification-after-Sign.
- Remove determinism from signatures by randomly sampling the nonce $\mathbf{y}$.
- Number Theoretic Transform used to perform polynomial multiplication

$$\mathbf{z} = \mathsf{INTT}(\mathsf{NTT}(\mathbf{s}_1) * \mathsf{NTT}(\mathbf{c})) + \mathbf{y}$$

- **Observation:** Our target addition operation is the last operation operating over $\mathbf{z}$.

Zero-Cost Mitigation

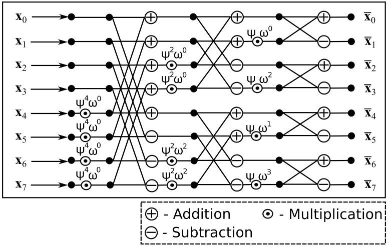
- Fault in a single coefficient does not cause enough perturbation to the $\mathbf{z}$ output for it to be rejected by the signing procedure.
- We compute $\mathbf{z}$ such that the addition operation is *pushed deeper* into the computation.
- **Idea**: Perform the Addition in the NTT domain.

$$\mathbf{z} = \mathsf{INTT}(\mathsf{NTT}(\mathbf{s}_1) * \mathsf{NTT}(\mathbf{c}) + \mathsf{NTT}(\mathbf{y}))$$

- The INTT operation performed after the faulty addition operation, propagates the fault to all the coefficients of $\mathbf{z}$.

Zero-Cost Mitigation

- Structure of INTT operation:



- Coefficients of the faulty $\mathbf{z}$ are uniformly distributes in $[0, q-1]$ while they are expected to be present in the interval $[0, \gamma_1 - 1]$.
- Thus, faulted signatures would always be rejected with very high probability!

Zero-Cost Mitigation

- It would take 20 years to actually build the same system of equations to recover $\mathbf{s}_1$ as opposed to just 621 seconds in case of the unprotected implementation.
- We use the NTT as a fault propagation mechanism which enables to reject faulty signatures.

Table of Contents

Conclusion

- Practical *Skip-Addition* fault attacks against two *deterministic* lattice-based signature schemes, Dilithium and qTESLA.
- Signature forgery algorithm for Dilithium using retrieved part of the secret key.
- Experimental validation through Electromagnetic fault injection on implementations taken from the *pqm4*, open-source benchmarking and testing framework for PQC schemes on the ARM Cortex-M4 microcontroller.
- We show that two well known countermeasures known to protect against *skip-addition* fault attacks can be defeated. This was also made possible owing to the *deterministic* nature of Dilithium.

Conclusion

- We also propose a *zero-cost* mitigation approach using the Number Theoretic Transform (NTT) as an in-built fault propagation mechanism with lattice-based signature schemes.

Thank you!

Any questions?

Nina Bindel, Johannes Buchmann, and Juliane Krämer.
Lattice-based signature schemes and their sensitivity to fault attacks.
In *Fault Diagnosis and Tolerance in Cryptography (FDTC), 2016 Workshop on*, pages 63–77. IEEE, 2016.

Thomas Espitau, Pierre-Alain Fouque, Benoit Gerard, and Mehdi Tibouchi.
Loop-abort faults on lattice-based signatures and key exchange protocols.
IEEE Transactions on Computers, 2018.