Exploiting Determinism in Lattice-based Signatures

Practical Fault Attacks on pqm4 Implementations of NIST candidates

Prasanna Ravi
prasanna.ravi@ntu.edu.sg
Temasek Labs and School of Computer Science and Engineering, NTU Singapore

Mahabir Prasad Jhanwar
mahavir.jhawar@ashoka.edu.in
Department of Computer Science, Ashoka University, Delhi

James Howe*
james.howe@pqshield.com
PQShield, Ltd.
Oxford, UK.

Anupam Chattopadhyay
anupam@ntu.edu.sg
School of Computer Science and Engineering, NTU Singapore

Shivam Bhasin
sbhasin@ntu.edu.sg
Temasek Labs, NTU Singapore

ABSTRACT

In this paper, we analyze the implementation level fault vulnerabilities of deterministic lattice-based signature schemes. In particular, we extend the practicality of *skip-addition* fault attacks through exploitation of determinism in certain variants of Dilithium (Deterministic variant) and qTESLA signature scheme (originally submitted deterministic version), which are two leading candidates for the NIST standardization of post-quantum cryptography. We show that single targeted faults injected in the signing procedure allow to recover an important portion of the secret key. Though faults injected in the signing procedure do not recover all the secret key elements, we propose a novel forgery algorithm that allows the attacker to sign any given message with only the extracted portion of the secret key. We perform experimental validation of our attack using Electromagnetic fault injection on reference implementations taken from the *pqm4* library, a benchmarking and testing framework for post quantum cryptographic implementations for the ARM Cortex-M4 microcontroller. We also show that our attacks break two well known countermeasures known to protect against *skip-addition* fault attacks. We further propose an efficient mitigation strategy against our attack that exponentially increases the attacker's complexity at almost zero increase in computational complexity.

CCS CONCEPTS

• **Security and privacy → Digital signatures**; **Hardware attacks and countermeasures**; **Side-channel analysis and countermeasures**; **Embedded systems security**.

KEYWORDS

Deterministic Lattice Signatures, pqm4, Fault Attack, Lattice-based Cryptography

1 INTRODUCTION

Recently, NIST has called for proposals for standardization of post-quantum cryptographic schemes for public-key encryption, digital signatures, and key establishment protocols [22]. This initiative is partly driven by the onset of the era of practical and scalable quantum computers [4], which has motivated the community to develop cryptographic schemes that are immune to cryptanalytic efforts using quantum algorithms. Amongst the initial candidates from different types of post-quantum cryptography, lattice-based cryptography has emerged as the largest category with 28 of 80 submissions. NIST [24] state that "schemes that can be made resistant to side-channel attacks at minimal cost are more desirable than those whose performance is severely hampered by any attempt to resist side-channel attacks". Thus, it has become essential to the standardization process to analyze the implementation security of these proposals against active and passive side-channel attacks.

Over the years, significant research has been done with respect to development of lattice-based cryptographic algorithms and in particular, initial proposals for lattice-based signature schemes such as GLP [16], BLISS [12] and the Bai-Galbraith (BG) [3] signature schemes attracted a lot of attention from the cryptographic community, owing to its security and efficiency guarantees. One of the noticeable characteristics of the aforementioned schemes is the use of ephemeral nonces during signature generation, similar to elliptic curve signature schemes such as DSA and ECDSA. While the initial lattice-based signature schemes [12, 16] utilized randomly generated nonces, some variants of the current lattice-based signature schemes such as Dilithium [19] and qTESLA [7] which are candidates in the NIST standardization process used deterministic procedures for nonce generation, thus making the signature scheme in itself completely deterministic. Determinism ensures that the signing procedure always outputs a unique signature for a given message. The same transition from non-deterministic to deterministic signature schemes was also previously witnessed in the case of elliptic curve signatures with the onset of deterministic ECDSA [27] and EdDSA [6] signature schemes. But, the determinism property came under heavy scrutiny and subsequently became a target and an enabler for a number of fault and side channel attacks [2, 5].

This naturally leads to critical questions about the introduction of similar implementation-level vulnerabilities due to determinism in lattice-based signature schemes. As the first and only work in exploiting determinism through implementation attacks, Bruinderink and Pessl [11] proposed the first differential style fault attacks on the Dilithium and qTESLA signature schemes. Another pertinent question to be answered is whether the introduced *determinism* can be exploited to increase the practicality or strengthen *existing*

*Most of this research was completed while the author was at the University of Bristol, supported by EPSRC via grant EP/N011635/1.

theoretical implementation attacks against lattice-based signature schemes.

As a first step towards answering this question, we extend the applicability and practicality of *skip-addition* fault attacks to deterministic variants of lattice-based signature schemes, Dilithium and qTESLA[1]. Theoretical *skip-addition* fault attack was first reported by Bindel *et al.* [8] over three non-deterministic lattice-based signature schemes GLP, BLISS and Ring-TESLA, but the attack requires a very high number of precisely targeted faults ($\approx$ 1000 for typical parameters) to be injected within a single run of the signing procedure. Their attack in practice will also be plagued by more pressing practical concerns, such as attacker's synchronization or fault detectability, which have been overlooked. These aspects thus make their attack highly impractical in practice. On the contrary, our proposed attack works with a single-fault injection model, that is only one fault injected in any given run of the algorithm. This single fault model is relaxed both for the equipment and underlying analysis for key recovery. Our attack also works against two well known implementation level countermeasures used to protect against *skip-addition* fault attacks. Thus, the efficacy of our attack has been further enhanced to practical limits through exploitation of determinism in lattice-based signature schemes.

1.1 Contribution

The contribution of this work is as follows:

- We extend the applicability and practicality of *skip-addition* fault attacks to deterministic lattice-based signature schemes. We show that the deterministic nature of Dilithium and qTESLA signature schemes can be exploited to extract the primary component of the secret key with only single targeted faults injected in the signing procedure.
- We further propose an alternate signature forgery algorithm for Dilithium through which one can generate valid signatures with only the extracted primary component of the secret key from the fault attack. Our forgery algorithm involves lesser computations compared to the original signing procedure albeit accompanied with a certain non-negligible failure probability.
- The fault vulnerabilities are validated using electromagnetic fault injection on the ARM Cortex-M4 microcontroller [17]. We performed practical fault attacks on the reference implementations taken from the *pqm4* library, a testing and benchmarking framework for post quantum cryptographic schemes for the ARM Cortex-M4 microcontroller.
- Given that our attack requires to inject precisely targeted faults, we propose a systematic approach towards identifying our target operations to fault through a two-order pattern matching technique over the Electromagnetic Emanation traces collected through the EM side channel.
- We also demonstrate practical attacks against two well known implementation level countermeasures proposed to protect against *skip-addition* fault attacks [8, 9], again made possible owing to the deterministic nature of Dilithium.
- Finally, we propose a *zero-cost* mitigation technique against our *skip-addition* fault attack which exponentially increases the attacker's complexity through simple re-ordering of operations within the signing procedure.

[1]In light of recent attacks against deterministic signature schemes [2, 11], Dilithium recently updated its specifications as part of the second round of the standardization process to allow for a probablistic variant based on the designer's choice, while qTESLA was modified into a probabilistic scheme [23]. While our attack still works against the deterministic variant of Dilithium, it is rendered ineffective on the probabilistic qTESLA signature scheme. We henceforth refer to the original first round submissions of Dilithium [19] and qTESLA [7] for the remainder of this paper and discuss the implications of the aforementioned updates on our attack in Sec.7.

2 PRELIMINARIES

Notation: Let $q \in \mathbb{N}$ be a prime. Elements in ring $\mathbb{Z}$ or $\mathbb{Z}_q$ are denoted by regular font letters viz. $a, b \in \mathbb{Z}$ or $\mathbb{Z}_q$. For an integer r and an even positive integer α, we define centered reduction modulo q denoted as $r \pmod{^{\pm} \alpha}$, to be the unique integer r_0 such that, $r \equiv r_0 \pmod{\alpha}$ and $-\frac{\alpha}{2} < r_0 \leq \frac{\alpha}{2}$. The usual modulo reduction is denoted by $r \pmod{q}$. For a set X, we write $x \xleftarrow{\$} X$ to denote that x is chosen uniformly at random from X. We denote the polynomial ring $\mathbb{Z}_q[X]/\langle X^n + 1\rangle$ as R_q. Polynomials in ring R_q are also represented as equivalent vectors of length n such that $\mathbf{a} \equiv (\mathbf{a}_0, \mathbf{a}_1, \ldots, \mathbf{a}_{n-1})$ for $\mathbf{a}_i \in \mathbb{Z}_q$. For an element $\mathbf{a} \in R_q$, we define $\|\mathbf{a}\|_\infty = \max_{0 \leq i \leq n-1} \|a_i\|_\infty$, where $\|a_i\|_\infty = |a_i \pmod{^{\pm} q}|$. While matrices and vectors with elements in $\mathbb{Z}_q$ are denoted by bold upper case letters ($\mathbf{A} \in \mathbb{Z}_q^n$), polynomials in R_q or matrices and vectors with elements in R_q are denoted using bold lower case letters ($\mathbf{a} \in R_q, \mathbf{b} \in R_q^\ell$). Multiplication of two polynomials $\mathbf{a}, \mathbf{b} \in R_q$ is denoted as $\mathbf{a} {\cdot} \mathbf{b}$ or $\mathbf{ab} \in R_q$. Due to the special structure (cyclotomic nature) of the factor polynomial used in R_q, multiplication can also be alternatively viewed as a matrix-vector multiplication such that $\mathbf{a} \cdot \mathbf{b} = \mathbf{a} \cdot \mathbf{B} = \mathbf{b} \cdot \mathbf{A}$ wherein the columns of the matrices $\mathbf{A}, \mathbf{B} \in \mathbb{Z}_q^{n \times n}$ are anti-cyclic rotations of $\mathbf{a}, \mathbf{b} \in R_q$ respectively. Point-wise multiplication (scalar product) is represented as $\mathbf{a} * \mathbf{b} \in R_q$. For a given $\eta \in \mathbb{N}$, define $S_\eta = \{\boldsymbol{a} \in R_q \mid \|\boldsymbol{a}\|_\infty \leq \eta\}$. We use $\hat{a}$ to denote the faulty value of a given $\mathbf{a} \in R_q$.

Lattice-based Cryptography: Most of the efficient lattice-based cryptographic schemes derive their hardness from two average-case hard problems, known as the Ring-Learning With Errors problem (RLWE) [20] and the Ring-Short Integer Solutions problem (RSIS) [21]. Both the problems reduce to worst-case hard problems over structured ideal lattices. Given a public key $(\mathbf{a}, \mathbf{t}) \in (R_q, R_q)$, an RLWE attacker is asked to find two small polynomials $\mathbf{s}_1, \mathbf{s}_2 \in R_q$ with $\mathbf{s}_1, \mathbf{s}_2 \in S_\eta$ such that $\mathbf{t} = \mathbf{a} \cdot \mathbf{s}_1 + \mathbf{s}_2$. Given m uniformly random elements $\mathbf{a}_i \in R_q$, an MSIS attacker is asked to find out a non-zero vector $\mathbf{z}$ with a small norm $\mathbf{z} \in S_\eta^m$ such that $\sum_i^m \mathbf{a}_i \cdot \mathbf{z}_i = 0 \in R_q$.

The more generalized versions of these problems known as Module-LWE (MLWE) and Module-SIS (MSIS) respectively deal with computations over the space $R_q^{k \times \ell} = \mathbb{Z}_q^{k \times \ell}[X]/(X^n + 1)$ for $k, l > 1$ (as opposed to R_q for their ring variants) and also provide better security guarantees compared to their corresponding ring variants. Any change in the security of a scheme (based on either MLWE or MSIS) only requires changes in the value of the module parameters (k, ℓ) while keeping the underlying structure of the ring fixed, thus warranting very minimal changes from a designer's perspective.

2.1 Dilithium

Dilithium is a deterministic lattice-based signature scheme whose security is based on MLWE and MSIS problems. In particular, security against key-recovery attack under the classical random oracle model is based on the hardness assumption of the MLWE problem; and the security against existential signature forgery is based on the MSIS hardness assumption. The scheme's security against strong signature forgery attack, under the quantum random oracle model, is also discussed by the authors [19].

2.1.1 Rounding Algorithms. This section starts off by briefly describing the various rounding procedures used in Dilithium. The purpose of them is to provide better compression of Dilithium's public-key and signatures. For non-negative integers r, α, the procedure $r \pmod{^{\pm} \alpha}$ outputs a unique integer r_0 in $-\alpha/2 < r_0 \leq \alpha/2$ such that $r \equiv r_0 \pmod{\alpha}$. The procedure Decompose : $\mathsf{D}_q(r, \alpha)$ decomposes r into a pair of integers (r_1, r_0) and the procedures HighBits : $\mathsf{HB}_q(r, \alpha)$ and LowBits : $\mathsf{LB}_q(r, \alpha)$ extract r_1 and r_0 from $\mathsf{D}_q(r, \alpha)$ respectively. The procedure MakeHint : $\mathsf{MH}_q(u, r, \alpha)$ produces a bit $h \in \{0, 1\}$ and subsequently the procedure UseHint : $\mathsf{UH}_q(h, r, \alpha)$ shows how to use the bit h as a hint to derive $\mathsf{HB}_q(u + r, \alpha)$. The relation between the procedures $\mathsf{MH}_q(u, r, \alpha)$ and $\mathsf{UH}_q(h, r, \alpha)$ is stated in the following lemma.

Lemma 2.1 ([19]). *For $r, z \in \mathbb{Z}_q$ with $||z||_\infty \leq \alpha/2$, we have*

$$\mathsf{UH}_q\left(\mathsf{MH}_q(z, r, \alpha), r, \alpha\right) = \mathsf{HB}_q(r + z, \alpha) \quad (1)$$

The exact details of these algorithms are provided in Algorithm 4 in Appendix A. All these algorithms naturally extend to higher abstractions like matrices, vectors, rings and modules by simply performing the same computations over their individual elements.

2.1.2 Description of Dilithium. In the following, we recall the details of the Dilithium signature scheme [19]. The underlying approach of the scheme is based on the "Fiat-Shamir with Aborts" framework [18] while the scheme in itself is an improved variant of the lattice-based signature scheme proposed by Bai and Galbraith[3]. The scheme operates over the base ring R_q with $n, q = (256, 8380417)$ while offering flexibility with the module parameters (k, ℓ) allowing to operate over varying dimensions $(k \times \ell)$ for different security levels. Since all/most of our analysis deals with the signing procedure of Dilithium, we present the details of the signing procedure in Algorithm 1, while we present its respective key-generation and verification procedures in Algorithm 5 in the Appendix A. For more information on the scheme, please refer [19].

Key Generation: The key generation algorithm, KeyGen(), generates the public constant $\mathbf{a} \in R_q^{k \times \ell}$ by expanding a given seed $\rho \leftarrow \{0, 1\}^{256}$ such that $\mathbf{a} = \mathsf{ExpandA}(\rho)$. Next, the secret module $\mathbf{s}_1 \in S_\eta^\ell$ and the error module $\mathbf{s}_2 \in S_\eta^k$ are sampled after which the MLWE instance $\mathbf{t} \in R_q^k$ is computed as $\mathbf{t} = \mathbf{a} \cdot \mathbf{s}_1 + \mathbf{s}_2$. The LWE instance is not directly output as the public key but is decomposed into $\mathbf{t}_0, \mathbf{t}_1$ such that $\mathbf{t}_1 = \mathsf{HB}_q(\mathbf{t}, 2^d)$ and $\mathbf{t}_0 = \mathsf{LB}_q(\mathbf{t}, 2^d)$. Subsequently, $\mathbf{t}_1$ is published as part of the public key while $\mathbf{t}_0$ is kept secret. Subsequently, the published public key is $(\rho, \mathbf{t}_1)$ while the secret key sk is $(\rho, K, tr, \mathbf{s}_1, \mathbf{s}_2, \mathbf{t}_0)$.

Signing: The signing procedure is iterative in nature with a number of conditional checks and it exits with a valid signature only when all the conditional checks are successfully passed. This is done to primarily ensure that the output signatures do not leak the distribution of the secret key. Moreover, these selective rejections in the signing procedure are also performed to ensure 100% correctness of the signature scheme.

Similar to the DSA and ECDSA signature schemes, the most important component of the signing procedure in case of Dilithium (apart from the secret key) is the ephemeral nonce $\mathbf{y} \in R_q^\ell$. Knowledge of a single value of $\mathbf{y}$ or reuse of $\mathbf{y}$ for different messages leads to a trivial break of the signature scheme. Moreover, the method of generation of the ephemeral nonce $\mathbf{y}$ also determines the deterministic nature of the signature scheme. In Dilithium, $\mathbf{y} \in S_\eta^\ell$ is deterministically generated using the ExpandMask function which takes as input, the message μ to be signed, the secret key component K and the iteration count (Line 6 of Sign in Algorithm 1). Further, the product $\mathbf{w} = \mathbf{a} \cdot \mathbf{y} \in R_q^k$ is computed and decomposed into $\mathbf{w}_1$ and $\mathbf{w}_0$ such that $\mathbf{w} = \mathbf{w}_1 \cdot 2\gamma_2 + \mathbf{w}_0$. The signing procedure requires the verifier to recover the value of $\mathbf{w}_1$ for successful signature verification. To facilitate the same, a hint vector $\mathbf{h} \in R_q^k$ with coefficients in $\{0, 1\}$ is also generated and output as part of the signature. Furthermore, a challenge polynomial $\mathbf{c}$ (sparse polynomial with only 60 non-zero coefficients in either ± 1) is also generated by hashing the ephemeral nonce along with the public key information and the message. The product $\mathbf{cs}_1$ is computed which is subsequently masked with the ephemeral nonce $\mathbf{y}$ through addition and the result is output as the primary signature component $\mathbf{z} \in R_q^\ell$ (Line 10 of Sign in Figure 1). The details of the verification procedure of Dilithium are provided for completeness in Algorithm 5. It is important to note that all attacks presented on the deterministic variant of Dilithium can also be easily adopted to the deterministic variant of qTESLA[1]. A detailed description of the qTESLA signature scheme along with the note on applicability of our attacks are deferred to the Appendix B.

3 MOTIVATION

In this section, we motivate our work by reviewing existing fault attacks on lattice-based signature schemes based on the "Fiat-Shamir with Aborts" framework. We observe that the generation of the *primary* signature component $\mathbf{z}$ has been the target of most of the reported attacks. Generation of $\mathbf{z}$ is done as follows:

$$\mathbf{z} = \mathbf{s}_1 \cdot \mathbf{c} + \mathbf{y} \quad (2)$$

We will henceforth refer to this step as $\mathrm{z_{gen}}$. We also refer to $\mathbf{s}_1$ alternatively as the *primary* secret of Dilithium as we will show later in this work that knowledge of $\mathbf{s}_1$ is enough to perform an existential forgery attack on Dilithium. While both $\mathbf{z}$ and $\mathbf{c}$ are revealed as part of the signature, $\mathbf{y}$ is the ephemeral masking polynomial used to mask the product $\mathbf{s}_1 \cdot \mathbf{c}$. Injection of faults in any of the operations within $\mathrm{z_{gen}}$ helps the attacker derive a direct relation of the faulted signature $\hat{\mathbf{z}}$ with the primary secret $\mathbf{s}_1$, thus naturally becoming a target of most of the previously reported attacks [8, 11, 13].

Algorithm 1: Dilithium Signature scheme

```
Procedure Sign($sk, M$)
    $\mathbf{A} \in R_q^{k\times\ell} := \mathsf{ExpandA}(\rho)$
    $\mu = \mathsf{CRH}(\mathrm{tr}\|M)$
    $\kappa = 0, (\mathbf{z}, \mathbf{h}) = \perp$
    while $(\mathbf{z}, \mathbf{h}) = \perp$ do
        $\mathbf{y} \in S_{\gamma_1-1}^{\ell} := \mathsf{ExpandMask}(K\|\mu\|\kappa)$
        $\mathbf{w} = \mathbf{A} \cdot \mathbf{y}$
        $\mathbf{w}_1 = \mathsf{HB}_q(\mathbf{w}, 2\gamma_2)$
        $\mathbf{c} \in B_{60} = H(\mu\|\mathbf{w}_1)$
        $\mathbf{z} = \mathbf{y} + \mathbf{c} \cdot \mathbf{s}_1$
        $(\mathbf{r}_1, \mathbf{r}_0) := \mathsf{D}_q(\mathbf{w} - \mathbf{c} \cdot \mathbf{s}_2, 2\gamma_2)$
        if $\|\mathbf{z}\|_\infty \geq \gamma_1 - \beta$ or $\|\mathbf{r}_0\|_\infty \geq \gamma_2 - \beta$ or $\mathbf{r}_1 \neq \mathbf{w}_1$ then
            $(\mathbf{z}, \mathbf{h}) = \perp$
        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 $\mathsf{wt}(\mathbf{h}) > \omega$ then
                $(\mathbf{z}, \mathbf{h}) = \perp$
        end
        $\kappa = \kappa + 1$
    end
    return $\sigma = (\mathbf{z}, \mathbf{h}, \mathbf{c})$
```

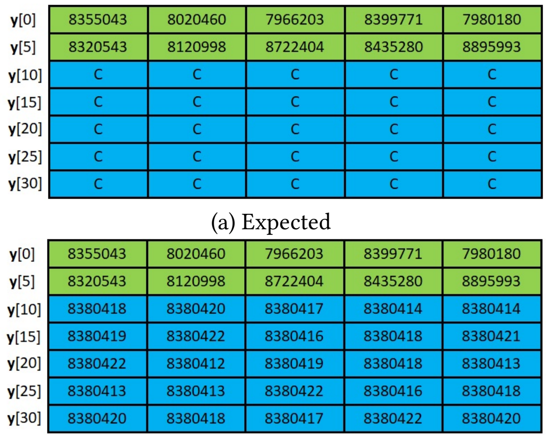
3.1 Note on Skipping fault attacks

Bindel *et al.* [8] proposed skipping fault attacks targeting the addition operation within $\mathrm{z_{gen}}$, over a number of non-deterministic Fiat-Shamir abort based signature schemes such as GLP [16], BLISS [12], and Ring-TESLA [1]. They proposed to completely skip the addition operation for all the n coefficients thereby yielding either $\mathbf{z} = \mathbf{s}_1 \cdot \mathbf{c}$ or $\mathbf{z} = \mathbf{y}$, depending on the order of operands for addition. While successful faults in the first case ($\mathbf{z} = \mathbf{s}_1\mathbf{c}$) directly yields the primary secret $\mathbf{s}_1$, the attack does not work in the latter case ($\mathbf{z} = \mathbf{y}$) since no information on the primary secret is revealed, also because the attacker cannot use the same $\mathbf{y}$ to generate another valid signature due to their non-deterministic nature. Thus, switching the order of operands was proposed as a potential countermeasure against skip-addition faults. The authors also proposed to store the result of addition in a variable different from the operands as another potential countermeasure against such *skip-addition* attacks. But, we later show that our attack is able to defeat both the aforementioned countermeasures.

Moreover, there are multiple questionable aspects with respect to the practicality of Bindel *et al.*'s [8] skipping fault attack. It requires to inject several hundreds of precisely targeted faults (1024 for recommended parameter sets of Dilithium) within a single run of the signing procedure. Such a scenario is highly unrealistic in a practical setting as it requires the attacker to achieve impeccable precision with respect to all the injected faults, especially when faults are not realizable with 100% repeatability. Achieving such faults would require very precise equipments along with high attacker's expertise. Secondly, the attacker has no apriori knowledge about the number of iterations of the signing procedure for a given message m. This makes it impossible if not very difficult to achieve precise synchronization with the target operation to be faulted. Thus, these practical aspects which were overlooked make the skipping attack of Bindel *et al.* very difficult, if not impossible to implement in practice.

3.2 Note on Loop Abort fault attacks

Espitau *et al.* [13] proposed a generic fault attack based on loop abort faults targeting both Fiat-Shamir Abort and Hash-and-Sign based signature schemes and later practically validated their attack in an updated work on the 8-bit Atmel XMEGA128 microcontroller [15]. It worked by converting the signature component $\mathbf{z}$ (generated using the $\mathrm{z_{gen}}$ step) into a solvable closest vector problem instance when the masking polynomial, $\mathbf{y}$, is limited to low degrees using loop-abort faults. The basic assumption was that all the non-sampled coefficients of the masking polynomial (due to the premature loop abort), would assume *zero* or a *constant* value C. We tried to validate this assumption for our target device (ARM Cortex-M4F microcontroller) but observed that the non-sampled coefficients retained random values on the contrary, instead of zero or a constant value, violating a critical pre-requisite of the attack. Figure 1 shows the comparison between the observed and expected values of coefficients of $\mathbf{y}$ after sampling 10 coefficients of $\mathbf{y}$. Thus, the attack proposed by Espitau *et al.* [15] was not directly applicable and motivated us to explore other attack settings.



y[0]	8355043	8020460	7966203	8399771	7980180
y[5]	8320543	8120998	8722404	8435280	8895993
y[10]	C	C	C	C	C
y[15]	C	C	C	C	C
y[20]	C	C	C	C	C
y[25]	C	C	C	C	C
y[30]	C	C	C	C	C

(a) Expected

y[0]	8355043	8020460	7966203	8399771	7980180
y[5]	8320543	8120998	8722404	8435280	8895993
y[10]	8380418	8380420	8380417	8380414	8380414
y[15]	8380419	8380422	8380416	8380418	8380421
y[20]	8380422	8380412	8380419	8380418	8380413
y[25]	8380413	8380413	8380422	8380416	8380418
y[30]	8380420	8380418	8380417	8380422	8380420

(b) Observed

Figure 1: Comparison between expected and observed values of the coefficients of y on our DUT after sampling 10 coefficients. The blocks colored in green denote the sampled coefficients while those colored in blue denote the non-sampled coefficients. (a) Expected values for coefficients of y to facilitate the loop-abort attack (C denotes a constant value) (b) Actual values of coefficients of y with random initial values.

3.3 Implementation Attacks on Lattice-based signatures

As stated earlier, the deterministic nature of ECC signature schemes, like ECDSA and EdDSA, were heavily exploited to demonstrate a number of fault attacks [2, 5]. Thus, one obvious question is whether such determinism can also be exploited in case of the similarly structured lattice-based signature schemes. The first attack on deterministic lattice-based signatures was proposed by Bruinderink and Pessl [11] who developed a differential style fault attack mainly exploiting the deterministic nature of the Dilithium and qTESLA signature schemes. The attack only required to inject a single random fault in the signing procedure and also showed that injection of such faults over a large section of the execution time of the signing procedure (~68%) could result in successful recovery of the primary secret. They also further propose a forgery algorithm to forge signatures for any message with only the primary secret[3].

The implementation security of lattice-based signature schemes have also been heavily scrutinized through a number of other side-channel attack vectors such as power/EM analysis [25], branch-tracing [14] and cache-timing [10, 26]. Most of these attacks demonstrated over the predecessor lattice-based schemes such as GLP [16], BLISS [12], and Ring-TESLA [1] were possible due to vulnerabilities in efficient implementations of the Gaussian samplers and rejection sampling procedures. The attacks on the Gaussian samplers and rejection sampling techniques later prompted the designers of the newer signature schemes like Dilithium and qTESLA, to instead sample from uniform distributions during signing. This both simplifies the rejection step in addition to simplifying the sampling procedure, albeit with the cost of increased size of the signatures.

4 FAULT ATTACK TO RETRIEVE THE PRIMARY SECRET

Our existential forgery attack on Dilithium consists of two parts. The first part involves recovery of the primary component of the secret key $\mathbf{s}_1$ using our *skip-addition* fault attack. Though the secret key sk of Dilithium consists of multiple other components apart from the primary secret $\mathbf{s}_1$, we propose a novel forgery signing procedure that can create valid signatures with only the knowledge of the primary secret thus completing our attack. This section explains recovery of the primary secret $\mathbf{s}_1$ through our *skip-addition* fault attack.

4.1 Adversary Model

We assume that the attacker has complete physical access to the victim device during computation of the signing procedure. The attacker should also be able to trigger the device arbitrarily many number of times into generating valid signatures for any message of his/her choice. The attacker should also be able to access the generated signatures. He/She should be able to actively interfere with the operation of the signing procedure through fault injection and passively observe through side-channels such as power/EM.

The underlying fault model we use for our attack is the instruction skip fault model. It has been widely studied and practically demonstrated on a range of devices (AVR and ARM microcontrollers) with high repeatability to satisfy our attack requirement [28–30]. Instructions skips over load, store, arithmetic and logical instructions been realized through multiple fault injection methodologies like laser shots [30] and electromagnetic injection [28, 29] thus serving as a basis for multiple cryptanalytic efforts.

[3]We note that the forgery signing algorithm proposed in this work operates in a different manner by exploiting the properties of the rounding algorithms of Dilithium.

4.2 Attack Methodology

Our attack mainly works by targeting the $\mathrm{z_{gen}}$ operation, specifically by performing skipping faults over the addition operation within $\mathrm{z_{gen}}$. But, unlike the attack from Bindel *et al.* [8], which proposes to fault multiple coefficients in a single run of the signing procedure, our attack leverages upon the deterministic property of Dilithium and works by injecting single targeted faults. For the attack, we will assume that the target is creating P signatures of the form $(\mathbf{z}[i][j], \mathbf{c}[i])$ with $i \in \{1, \ldots, P\}$, while $j \in \{0, \ldots, \ell - 1\}$ denotes the individual polynomials within the module $\mathbf{z}$ (we ignore the hint component $\mathbf{h}$ of the signature σ since it is not involved in our target operation, $\mathrm{z_{gen}}$). The signature component $\mathbf{z}$ is generated as follows:

$$\mathbf{z}[i][j] = \mathbf{s}_1[j] \times \mathbf{c}[i] + \mathbf{y}[i][j]$$

To simplify notations, we perform our analysis over Eqn.2 (i.e) $\mathbf{z} = \mathbf{s}_1\mathbf{c} + \mathbf{y}$, considering its individual components as polynomials in R_q, since the polynomials are handled independently of each other. The attacker faults the addition operation corresponding to a single coefficient $(\mathbf{z})_t$ of $\mathbf{z}$ with $t \in \{0, \ldots, n-1\}$, using single faults and aggregates information from multiple faulty signatures obtained by faulting different coefficients to recover the primary secret $\mathbf{s}_1$. We hereby propose two fault attacks based on the order of the operands in the addition operation.

4.2.1 Case 1. We first consider the $\mathrm{z_{gen}}$ step as is present in the reference implementation of the Dilithium signature scheme submitted to NIST. The signature component $\mathbf{z}$ is generated as follows:

$$\begin{aligned} \mathbf{z} &= \mathbf{s}_1 \cdot \mathbf{c} \\ \mathbf{z} &= \mathbf{z} + \mathbf{y} \end{aligned} \tag{3}$$

If the addition operation is performed according to Equation 3, then skipping the addition corresponding to a single coefficient, t, will result in a scenario where $(\hat{\mathbf{z}})_t = (\mathbf{s}_1\mathbf{c})_t$. The attacker with the knowledge of a single coefficient $(\mathbf{s}_1\mathbf{c})_t$ can construct the following equation:

$$(\hat{\mathbf{z}})_t = \langle \mathbf{s}_1, \mathrm{Rot}(\mathbf{c}, t) \rangle \tag{4}$$

Here, the attacker knows coefficient $(\hat{\mathbf{z}})_t$, of $\mathbf{z}$, and $\epsilon_t = \mathrm{Rot}(\mathbf{c}, t)$, the rotated coefficient vector of challenge $\mathbf{c}$ rotated by t times in an anti-cyclic fashion. Thus, Equation 4 is nothing but a modular linear equation with n unknowns, with the unknowns being the coefficients $(\mathbf{s})_t$ of the primary secret for $t \in \{0, \ldots, n-1\}$.

For a given message m, let $I(m)$ be the number of iterations of the non-faulted signing procedure. For our attack, we only consider single faults injected over addition in the last iteration $I(m)$ of the signing procedure. Let $I_f(m, t)$ denote the number of iterations when faulting the addition operation for the t^{th} coefficient and let $\Delta(I, I_f)_{m,t} = I(m) - I_f(m, t)$ be the difference between the number of iterations of the non-faulted and faulted signing procedures. Since we consider only single faults, the attacker uses a faulty pair $\mathcal{P}_i = (\epsilon_t, (\hat{\mathbf{z}})_t)$ for the attack if and only if $\Delta(I, I_f)_{m,t} = 0$. If $\Delta(I, I_f)_{m,t} \neq 0$, it means that the faulty signature $\hat{\mathbf{z}}$ corresponding to the last iteration has been rejected by the check on $\|\mathbf{z}\|_\infty$ and subsequently the signing procedure outputs another valid signature corresponding to another iteration $N > I(m)$, which cannot be used for our attack.

We address the issue of precisely targeting the addition operation of the last iteration for fault injection in Section 6.3, but for now we assume that the attacker is capable in doing so. Upon faulting, the attacker can easily estimate whether $\Delta(I, I_f)_{m,t} = 0$ by simply checking the equality of challenge polynomials $\mathbf{c}$ output as part of the faulty and non-faulty signatures. When $\Delta(I, I_f)_{m,t} = 0$ the challenge polynomials are the same, but are otherwise different. The goal of the attacker is to finally collect n pairs $\mathcal{P}_i$ for $i \in \{0, \ldots, n-1\}$ by faulting across different coefficients and possibly different message inputs. There is no restriction on both the aforementioned variables. Once collected, they can build a well defined system of linear equations as follows:

$$\mathbf{s}_1\mathbf{C} = \mathbf{L} \tag{5}$$

where the columns of matrix $\mathbf{C}$ (size $n \times n$) correspond to the vectors ϵ_t and the vector $\mathbf{L}$ (length n) is formed using the correspondingly faulty signature coefficients $(\hat{\mathbf{z}})_t$. The above system can be trivially solved using Gaussian elimination to recover the primary secret $\mathbf{s}_1$.

Considering the possibility of rejecting the faulty signature, the faulty coefficient $(\hat{\mathbf{z}})_t$ which is equal to the corresponding coefficient $(\mathbf{s}_1\mathbf{c})_t$ always lies in the allowable range for coefficient of $\mathbf{z}$, that is, $||\mathbf{z}||_\infty \le \gamma_1 - \beta$ while $||\mathbf{s}_1\mathbf{c}||_\infty < \beta$ with high probability and $\beta << \gamma_1 - \beta$. Thus, the probability of rejecting a faulted coefficient of $\hat{\mathbf{z}}$ is zero (i.e) ($p_{coeff\_rej} = 0$) and hence no faulty signature will be rejected by the signing procedure. Thus, the number of required signatures to be faulted for recovery of the primary secret is equal to its number of coefficients, n.

4.2.2 Case 2. We consider the alternate case wherein the $\mathsf{z_{gen}}$ operation is performed as follows:

$$\begin{aligned} \mathbf{z} &= \mathbf{y} \\ \mathbf{z} &= \mathbf{z} + \mathbf{s}_1 \cdot \mathbf{c} \end{aligned} \tag{6}$$

From Equation 6, we can infer that skipping the addition operation for coefficient t will result in a scenario where $(\hat{\mathbf{z}})_t = (\mathbf{y})_t$. This only provides the attacker with information about $(\mathbf{y})_t$ of the masking polynomial $\mathbf{y}$. Bindel *et al.* [8] proposed this case as a countermeasure for non-deterministic signatures since another signature using the same $\mathbf{y}$ could not be generated and hence no information about the primary secret can be extracted from the faulty coefficient. But, the deterministic nature of Dilithium ensures that the attacker can generate signatures for the same $\mathbf{y}$ multiple times for a given message m. Assuming the attacker faults the addition operation of the last iteration corresponding to the t^{th} coefficient and $\Delta(I, I_f)_{m,t} = 0$, the attacker computes the corresponding t^{th} coefficient of the product $\mathbf{s}_1\mathbf{c}$ as follows:

$$(\mathbf{s}_1\mathbf{c})_t = \langle \mathbf{s}_1, \mathrm{Rot}(\mathbf{c}, t)\rangle = (\mathbf{z})_t - (\hat{\mathbf{z}})_t \tag{7}$$

The attacker simply has to repeat the process across different coefficients corresponding to one or more message inputs until he can extract n such equations to be used for the attack. He can then construct Equation 5 similar to that of Case-1, which can be subsequently solved using Gaussian elimination for the primary secret $\mathbf{s}_1$. The faulted coefficient $(\hat{\mathbf{z}})_t$ in this case retains the value of the corresponding coefficient $(\mathbf{y})_t$ of $\mathbf{y}$ wherein $||\mathbf{y}||_\infty \le \gamma_1 - 1$. Since it is required that $||\mathbf{z}_\infty|| \le \gamma_1 - \beta$, the probability that the faulted coefficient lies in the bad range is $p_{coeff\_rej} = \dfrac{2(\beta-1)}{(2\gamma_1 - 1)}$ ($p_{coeff\_rej} \approx 5 \times 10^{-4}$ for recommended parameter sets). Since the number of iterations of the signing procedure varies with the input message m and is not known apriori, we do not derive a closed form expression for the rejection probability of faulty signatures, but empirically estimate the number of required faulted signatures for key recovery using fault simulations.

We perform fault simulations for both the cases (Case-1 and Case-2) of the $\mathsf{z_{gen}}$ operation, each over 10^5 runs of the signing procedure. Refer to Table 1 for the number of signatures to be faulted for recovery of the complete primary secret $\mathbf{s}_1$ (with ℓ polynomials) across the multiple parameter sets of Dilithium. We also provide corresponding numbers for an attacker with a brute-force capability of 2^{64}. For Case-1, the faulty signatures are never rejected and hence the number of signatures to be faulted is equal to the number of coefficients to be recovered ($\ell \times n$). For Case-2, since p_{coeff_rej} is very small ($\approx 5 \times 10^4$), we observe that the number of signatures to be faulted for successful recovery of the primary secret is still very close to that of Case-1 for all the parameters (that is, $\ell \times n$). We again emphasize the fact that our attack requires to only inject single faults and the attacker has to simply repeat the faults over multiple runs of the signing procedure to extract enough information to recover the primary secret of Dilithium.

5 EXISTENTIAL FORGERY WITH KNOWLEDGE OF PRIMARY SECRET

The attacker can successfully retrieve the primary secret $\mathbf{s}_1$ using our *skip-addition* fault attack. But the secret key of Dilithium secret key sk is composed of multiple components $(\rho, K, tr, \mathbf{s}_1, \mathbf{s}_2, \mathbf{t}_0)$. In this section, we show that the knowledge of the primary secret $\mathbf{s}_1$ is sufficient to perform an existential forgery attack on the Dilithium signature scheme.

For the following attack, refer to the function Sign in Algorithm 1. We consider an adversary $\mathcal{A}$, who has knowledge of $\mathbf{s}_1$, whose goal is to produce a valid signature of a new message. Execution of Lines 10 through 18 require knowledge of the primary secret $\mathbf{s}_1$ (Line 18), which is already recovered and the secret $K \in \{0,1\}^{256}$ (Line 14). K is simply used for generation of the ephemeral $\mathbf{y}$ in the signing procedure. Since the verifier does not check for the validity of $\mathbf{y}$, that is whether or not is $\mathbf{y}$ generated using the same valid K. Thus, $\mathcal{A}$ proceeds through Lines 10 to 18, choosing $\mathbf{y}$ uniformly at random from $S^{\ell}_{\gamma_1 - 1}$ in Line 14 and computing $\mathbf{z}$ in Line 18 using the recovered $\mathbf{s}_1$.

Signature verification requires knowledge of $\mathbf{w}_1$. Now $\mathbf{w} - \mathbf{c}\mathbf{s}_2 = (-\mathbf{c}\mathbf{t}_0) + (\mathbf{w} - \mathbf{c}\mathbf{s}_2 + \mathbf{c}\mathbf{t}_0)$. Writing $\mathbf{u} = -\mathbf{c}\mathbf{t}_0$, $\mathbf{r} = \mathbf{w} - \mathbf{c}\mathbf{s}_2 + \mathbf{c}\mathbf{t}_0$, and the fact that $\mathbb{P}[||\mathbf{u}||_\infty \le \gamma_2] \approx 1$ (indeed, $|| - \mathbf{c}\mathbf{t}_0||_\infty \le ||\mathbf{t}_0||_\infty < 2^d < \gamma_2$) together ensure that $\mathbf{w}_1$ can be computed as follows (Lemma 2.1):

$$\begin{aligned} \mathsf{UH}_q\left(\mathsf{MH}_q(\mathbf{u}, \mathbf{r}, 2\gamma_2), \mathbf{r}, 2\gamma_2\right) &= \mathsf{HB}_q(\mathbf{u} + \mathbf{r}, 2\gamma_2) \\ &= \mathsf{HB}_q(\mathbf{w} - \mathbf{c}\mathbf{s}_2, 2\gamma_2) \\ &= \mathbf{w}_1. \end{aligned}$$

But computation of $\mathsf{MH}_q(\mathbf{u}, \mathbf{r}, 2\gamma_2)$ requires knowledge of $\mathbf{u} = -\mathbf{c}\mathbf{t}_0$, and therefore subsequently the secret-key component $\mathbf{t}_0$. *The problem for $\mathcal{A}$ is now to compute hint matrix $\mathbf{h} = \mathsf{MH}_q(\mathbf{u}, \mathbf{r}, 2\gamma_2)$ without*

Table 1: Fault simulation results for our *skip-addition* fault attack on the various recommended parameter sets of Dilithium. Results were obtained over 10^5 runs of the signing procedure.

Attacker's brute-force capability	No. of Faulted Signatures							
	Weak		Medium		Recommended		High	
	Case-1	Case-2	Case-1	Case-2	Case-1	Case-2	Case-1	Case-2
None	512	512.065	768	768.092	1024	1024.017	1280	1280.034
2^{64}	496	496.062	751	751.089	1006	1006.016	1258	1258.033

the knowledge of $\mathbf{t}_0$. In the following we first show that UH_q function could be inverted to produce the correct hint. In particular $\mathsf{UH}_q(h, r, \alpha)$, given q, α, $r \in \mathbb{Z}_q$ and $\mathsf{HB}_q(u + r, \alpha)$, is invertible on h if $\|u\|_\infty \leq \alpha/2$. Lemma 5.1 summarizes this fact.

Lemma 5.1. *Let $u \in \mathbb{Z}_q$ with $\|u\|_\infty \leq \alpha/2$, where $\alpha > 0$. Then, given any $r \in \mathbb{Z}_q$ and $\mathsf{HB}_q(u+r, \alpha)$, it is easy to compute $\mathsf{MH}_q(u, r, \alpha)$.*

Proof: The algorithm in Figure 2, given below, illustrates this claim. The correctness of the algorithm follows immediately from Lemma 2.1 and Lemma 5.2.

Algorithm 2: Inverting UH_q for the hint bit h

```
Input  : $q, r, \alpha, \theta = \mathsf{HB}_q(u + r, \alpha)$
Output: $\mathsf{MH}_q(u, r, \alpha)$
Set $h = 0$
Compute $\phi = \mathsf{UH}_q(h, r, \alpha)$
if $\phi = \theta$ then
    return $h$
else
    return 1
end
```

Lemma 5.2 ([19]). *Let $r \in \mathbb{Z}_q$ and $h, h' \in \{0, 1\}$. If $\mathsf{UH}_q(h, r, \alpha) = \mathsf{UH}_q(h', r, \alpha)$, then $h = h'$.*

Thus, in order to compute $\mathbf{h}$, $\mathcal{A}$ must ensure that it has access to $\mathsf{HB}_q(\mathbf{u} + \mathbf{r}, 2\gamma_2) = \mathsf{HB}_q(\mathbf{w} - \mathbf{cs}_2)$, and $\mathbf{r} = \mathbf{w} - \mathbf{cs}_2 + \mathbf{ct}_0$. Computing $\mathbf{r}$ is easy as $\mathbf{w} - \mathbf{cs}_2 + \mathbf{ct}_0 = \mathbf{Az} - \mathbf{ct}_1 2^d$. To compute $\mathsf{HB}_q(\mathbf{w} - \mathbf{cs}_2)$ all $\mathcal{A}$ has access to is $\mathbf{w}_1$. But it is known that $\mathsf{HB}_q(\mathbf{w} - \mathbf{cs}_2, 2\gamma_2) = \mathbf{w}_1$ provided $\|\mathsf{LB}_q(\mathbf{w} - \mathbf{cs}_2, 2\gamma_2)\|_\infty \leq \gamma_2 - \beta$. A valid signer could ensure this check, but for $\mathcal{A}$ it is not possible as it requires knowledge of $\mathbf{s}_2$. Thus, the signer has to ignore the conditional checks on $\|\mathbf{r}_0\|$ and $\|\mathbf{c} \cdot \mathbf{t}_0\|$. Thus, it is natural for our forgery scheme to produce some invalid signatures with certain non-negligible probability. We now finally present complete details of our existential forgery attack in Alg.3.

We present the results of our forgery signing algorithm when implemented on an Intel Core-i5 (Haswell) processor running at 2.6 GHz with Turbo Boost and hyper-threading disabled and compiled with gcc-4.2.1 and compilation flags `-march=native -mtune=native -O0 -g`[4]. We obtained an average signing time of about 0.3253 msecs for our forgery signing procedure which is about 2.67 times faster than the original signing procedure which runs at 0.8689 msec. The improved speed can be attributed to the reduced number of operations, but also to the removal of conditional checks over $\|\mathbf{r}_0\|$ and $\|\mathbf{c} \cdot \mathbf{t}_0\|$. We attempted to empirically compute the failure probability of our forgery signing procedure. We ran our forgery algorithm for a total of 2^{28} times while not obtaining a single invalid signature. This along with its increased signing rate leads us to hypothesize if our forgery algorithm can be used as an alternative signing procedure for Dilithium. Concrete estimation of its error-probability and security of the generated signatures is left for future work.

Algorithm 3: Forgery(pk, $\mathbf{s}_1$, M)

```
input  : public-key $pk = (q, \rho, \mathbf{t}_1)$, Primary secret = $\mathbf{s}_1$, A message $M$
output: A forged Dilithium signature
$\mathbf{A} \sim R_q^{k \times \ell} := \mathsf{Sam}(\rho)$
$\mu = H(H(\rho \| \mathbf{t}_1) \| M)$
$\mathbf{y} \xleftarrow{\$} S_{\gamma_1 - 1}^{\ell}$
$\mathbf{w} = \mathbf{Ay}$
$\mathbf{w}_1 = \mathsf{HB}_q(\mathbf{w}, 2\gamma_2)$
$\mathbf{c} = H(\mu, \mathbf{w}_1)$
$\mathbf{z} = \mathbf{y} + \mathbf{cs}_1$
for $i = 1$ to $k$ do
    for $j = 0$ to $n - 1$ do
        $\mathbf{h}_{i,j} = 0$
    end
end
for $i = 1$ to $k$ do
    for $j = 0$ to $n - 1$ do
        $\theta_{i,j} = \mathsf{UH}_q(h_{i,j}, [\mathbf{Az} - \mathbf{ct}_1 2^d]_{i,j}$
    end
end
for $i = 1$ to $k$ do
    for $j = 0$ to $n - 1$ do
        if $\theta_{ij} \neq [\mathbf{w}_1]_{ij}$ then
            Set $h_{ij} = 1$
        end
    end
end
if $\mathsf{UH}_q(\mathbf{h}, \mathbf{Az} - \mathbf{ct}_1 2^d, 2\gamma_2) \neq \mathbf{w}_1$ or $\|\mathbf{z}\|_\infty \geq \gamma_1 - \beta$ or $\mathrm{wt}(\mathbf{h}) > \omega$ then
    Go to 3
else
    return $(\sigma = (\mathbf{z}, \mathbf{h}, \mathbf{c}))$
end
```

[4]Code is available online on https://github.com/jameshoweee/dilithium_forgery

6 EXPERIMENTAL RESULTS

In this section, we perform an experimental validation of our proposed attacks on a real device. We start by introducing our experimental setup, providing details of our device under target, implementation details and our attack setup. We analyze several implementation variants of the $\mathbf{z}_{gen}$ operation and its susceptibility to our proposed skipping fault attacks. We demonstrate practical faults over the different implementation variants with very high repeatability.

6.1 Experimental Setup

For our experiments, we target the reference implementation of Dilithium taken from the *pqm4* library, a benchmarking and testing framework for PQC schemes on the ARM Cortex-M4 family of microcontrollers [17]. We ported its reference implementation to the STM32F4DISCOVERY board (DUT) housing the STM32F407, ARM Cortex-M4 microcontroller. Our implementation (compiled with `-O3 -mthumb -mcpu=cortex-m4 -mfloat-abi=hard -mfpu=fpv4-sp-d16`) runs at a clock frequency of 24 MHz. We use the ST-LINK/v2.1 add-on board for USART communication with our DUT. We used the OpenOCD framework for flash configuration and on-chip hardware debugging with the aid of the GNU debugger for ARM (arm-none-eabi-gdb). We use Electromagnetic Fault injection (EMFI) to inject faults into our device.

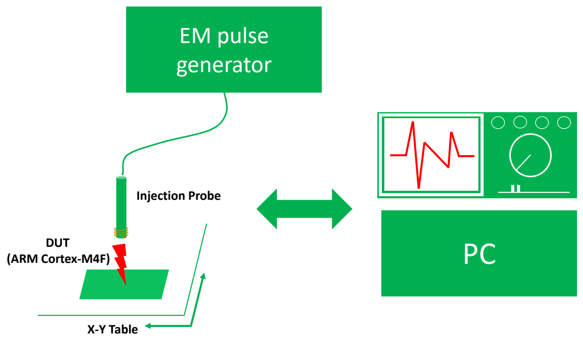


Figure 2: Experimental setup for the fault injection

Refer Figure 2 for our EMFI setup. The EMFI setup injects electromagnetic pulses with high voltage (upto 200 V) with low rise time (<4ns) in order to disturb the target operation. A controller software running on the laptop controls both the EM pulse generator and the DUT and synchronizes their operation through serial communication. The EM pulse generator is directly triggered by an external trigger signal from the DUT. The EM pulse injector is a customized hand-made EM probe designed as a simple loop antenna. Refer Figure 3 for the EM probe used for our experiments.

6.2 Implementation of EMFI attack

Given that our fault attack requires to inject targeted faults, it becomes necessary to precisely identify the exact instructions to be faulted within the implementation. We attempt to analyze the fault vulnerabilities of three different variants of the $\mathbf{z}_{gen}$ operation.

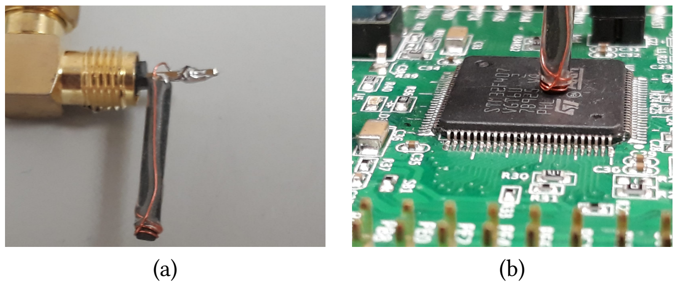


Figure 3: (a) Hand-made probe used for our EMFI setup (b) Probe placed over the DUT

We will give the following terminology to the three considered variants:

- Variant-1: Adding $\mathbf{y}$ to $\mathbf{s}_1\mathbf{c}$
- Variant-2: Adding $\mathbf{s}_1\mathbf{c}$ to $\mathbf{y}$
- Variant-3: Prevent Overwriting the result onto the operands

The first two variants are based on the order of the operands within the addition operation while the result of addition in both the variants is overwritten into the same variable as one of the operands. But, the third variant is based on storing the result into a new variable different from the operands. While we demonstrate our attack against all the three variants, it is important to note that Variant-2 and 3 were proposed as concrete countermeasures against the *skip-addition* attack in a number of works [8, 9].

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

Figure 4: Assembly code snippet from implementation of Dilithium containing the target store operation

6.2.1 Variant-1: Adding $\mathbf{y}$ *to* $\mathbf{s}_1\mathbf{c}$. This variant corresponds to Case-1 of $\mathbf{z}_{gen}$ where random information ($\mathbf{y}$) is added to secret information ($\mathbf{s}_1\mathbf{c}$) stored in $\mathbf{z}$. Refer Figure 5 for the C code snippet of the operations corresponding to $\mathbf{z}_{gen}$. It is important to note that all C-code snippets in this paper only include those operations (lines of code) relevant to our attack. Referring to the assembly code snippet in Figure 4 corresponding to the target addition operation, we can see that the result (Line 4 of Figure 4) in register $r3$ is stored back to the memory location pointed to by register $r0$ offset by 4 (Line 6 of Figure 4).

Utilizing the on-chip hardware debugging feature, we found that the pointers to both the source and destination memory locations of the addition operation contain the same value (i.e) $r4 = r0$ in line 1 and 6 of Figure 4 respectively. This confirms our claims that the result of addition is stored back into the memory location of $\mathbf{s}_1\mathbf{c}$ through the STR instruction (Line 6 in Figure 4). Skipping this STR (store) instruction once effectively has the same effect as skipping the addition operation for that coefficient. The addition operation

```
1   /* Sampling y */
2   for(i = 0; i < L; ++i)
3     poly_unif_gamma1m1(y.vec+i,key,nonce++);
4   /* Computing NTT(y) */
5   yhat = y;
6   polyvecl_ntt(&yhat);
7   /* Computing NTT(c) */
8   chat = c;
9   poly_ntt(&chat);
10  /* Computing product sc */
11  for(i = 0; i < L; ++i)
12  {
13    poly_ptwise_imont(z.vec+i,&chat,s1.vec+i);
14    poly_intt_mont(z.vec+i);
15  }
16  /* Last addition to generate z */
17  /* (y added to sc) */
18  polyvecl_add(&z,&y,&z);
```

Figure 5: Code snippet: C-Code of Variant-1, Adding $\mathbf{y}$ to $\mathbf{s}_1\mathbf{c}$ with the result of addition stored in the same variable that contains $\mathbf{s}_1 c$

```
1   /* Sampling y in z */
2   for(i = 0; i < L; ++i)
3     poly_unif_gamma1m1(z.vec+i,key,nonce++);
4   /* Computing NTT(y) */
5   yhat = z;
6   polyvecl_ntt(&yhat);
7   /* Computing NTT(c) */
8   chat = c;
9   poly_ntt(&chat);
10  /* Computing product sc */
11  for(i = 0; i < L; ++i)
12  {
13    poly_ptwise_imont(sc.vec+i,&chat,s1.vec+i);
14    poly_intt_mont(sc.vec+i);
15  }
16  /* Last addition to generate z */
17  /* (sc added to y) */
18  polyvecl_add(&z,&sc,&z);
```

Figure 6: Code snippet: C-Code of Variant-2, Adding $\mathbf{s}_1\mathbf{c}$ to $\mathbf{y}$ with the result for addition stored in the same variable that contains $\mathbf{y}$

corresponding to other coefficients could also be similarly faulted to yield multiple faulted signatures using which the attacker can recover the primary secret $\mathbf{s}_1$ using our analysis presented for Case-1.

6.2.2 Variant-2: Adding $\mathbf{s}_1\mathbf{c}$ *to* $\mathbf{y}$. Here, we consider the case when secret information ($\mathbf{s}_1\mathbf{c}$) is added to random information ($\mathbf{y}$) stored in $\mathbf{z}$ illustrating Case-2 of $\mathsf{z_{gen}}$. The C code snippet of the correspondingly modified implementation can be seen in Figure 6. The compiled assembly code generated for this case was similar to that of Variant-1 (barring changes in the register values and register locations). Thus, we use the same assembly code in Figure 4 for our analysis on Variant-2. Similar to our attack on Variant-1, the same STR (store) instruction (Line 6 of Figure 4) can be skipped to effectively skip the addition operation for the given coefficient, thus ensuring that the faulted coefficient retains the value of corresponding coefficient of $\mathbf{y}$. The attacker simply repeats the faults across different coefficients until he generates enough faulted signatures using which our analysis presented for Case-2 can be used to recover the primary secret $\mathbf{s}_1$ of Dilithium.

```
1   /* Sampling y */
2   for(i = 0; i < L; ++i)
3     poly_unif_gamma1m1(y.vec+i,key,nonce++);
4   /* Computing NTT(y) */
5   yhat = y;
6   polyvecl_ntt(&yhat);
7   /* Computing NTT(c) */
8   chat = c;
9   poly_ntt(&chat);
10  /* Computing product sc */
11  /* Result stored in ztemp */
12  for(i = 0; i < L; ++i)
13  {
14    poly_ptwise_imont(ztemp.vec+i,&chat,s1.vec+i);
15    poly_intt_mont(ztemp.vec+i);
16  }
17      /* Last addition to generate z */
18      /* Result stored in new variable z */
19      polyvecl_add(&z,&y,&ztemp);
```

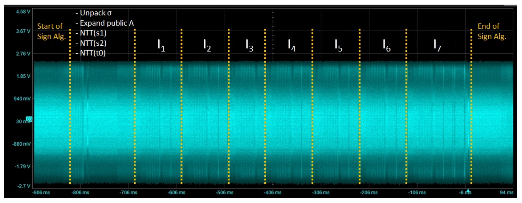
Figure 7: Code snippet: C-Code of Variant-3, Result is stored in a new variable compared to that of the operands

6.2.3 Variant-3: Prevent Overwriting the result onto the operands. We consider a third variant wherein the result of the addition operation is stored in a new variable (new memory location), which was the other technique that was proposed as a possible countermeasure against *skip-addition* fault attacks [8, 9]. Refer Figure 7 for the C Code of the correspondingly modified implementation of the $\mathsf{z_{gen}}$ step, wherein the result of the addition operation is stored in a new variable (Line 18) compared to that of the operands. We again observe a very similar compiled assembly implementation (barring changes in the register values and register location) and hence use the same assembly code snippet in Figure 4 for analysis.

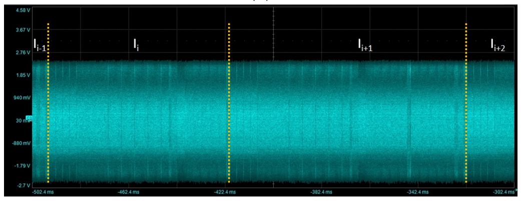
But for this case, we found that the pointers to the destination and source operands now point to different memory locations (i.e) $r0 \neq r4$. Thus, skipping the STR (store) instruction will not result in a meaningful fault since the faulted coefficient will retain a random value. But, we see that the ADD instruction (Line 4 in Figure 4) is implemented as `ADD r3,r1` wherein the source and destination registers are the same ($r3$). Thus, skipping this ADD instruction will ensure that register $r3$ which contains the first operand (coefficient of $\mathbf{s}_1\mathbf{c}$ of $\mathbf{y}$ depending on the order of operands), will be stored as the result of the addition operation. We attribute this nature of the ADD instruction (source register and destination register are the same) to the choice of maximum level of compiler optimization (O3). Upon successful fault injection, the attacker can use either of our attacks proposed for Case-1 or Case-2, depending on the order of operands used in the addition operation, to recover the primary secret $\mathbf{s}_1$.

6.3 Systematic Approach towards targeted fault injection in Dilithium

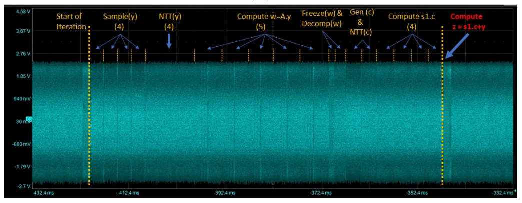
The success of our attack depends on the attacker's ability to precisely target the addition operation of $\mathsf{z_{gen}}$ in the last iteration of the signing procedure. For a given message, the attacker has to



(a)



(b)



(c)

Figure 8: First order Pattern Recognition (a) Identification of multiple iterations of signing procedure on EM trace (b) Zoomed in view of multiple iterations (c) Identifying repeating patterns in a single iteration and mapping them to the corresponding operations in the reference implementation

identify the time instance corresponding to the last iteration and subsequently, the target addition operation within the last iteration. While none of the previous works requiring to inject precisely targeted faults [8, 13] address this point, we leverage on the deterministic nature of the Dilithium scheme and use the information from the EM/power side channel for precise identification of our target operation.

EM Measurements are observed from the same DUT using a near-field probe and are processed using a Lecroy 610Zi oscilloscope at a sampling rate of 50MSam/sec. Refer Figure 8(a) for the EM trace of a single signing procedure wherein the repeating patterns corresponding to different iterations can be clearly distinguished through simple visual inspection[5]. Refer Figure 8(b) for a zoomed-in-view of the same trace. Thus, mere visual inspection of the trace allows us to approximately if not accurately locate the different iterations and also identify the last iteration which is of most interest to us for our attack. Given the deterministic nature of the signature scheme, the position of the last iteration for a given message is the same (i.e) $\text{loc}_{\text{last\_iter}}(m)$, unless targeted faults lead to rejection of the faulted signature.

The second and trickier task is to locate the position of addition operation within the last iteration. The offset of the addition operation (denoted as $\text{loc}_{\text{add}(m)}$) varies with respect to different iterations within the signing procedure owing to the rejection sampling approach used to sample for $\mathbf{y} \in S^{\ell}_{\gamma_1 - 1}$. We make two crucial observations over the reference implementation of the Dilithium scheme which aids us in locating the addition operation.

- Observation-1: The earliest rejection of the signature occurs during the infinity norm check of $\mathbf{z}$. This operation is performed **just after** our target addition operation.
- Observation-2: Almost all operations preceding our target addition operation are **repeating** for a certain number of times depending on the number of polynomials (k, ℓ) in the module.

We collect multiple fault-free EM traces t_i for $i = 1, \ldots, N$ corresponding to the signing procedures for corresponding message inputs m_i and further partition the traces into *segments* corresponding to the individual iterations as stated above. Based on Observation 1, we know that the offset of addition operation in $\mathbf{z}_{\text{gen}}$ will be lesser than the length of the *shortest segment* ($\text{len}_{\text{short\_seg}}$) and the addition operation will be approximately located near the end of the smallest segment, that is ($\text{loc}_{\text{add}}(m) \lesssim \text{len}_{\text{short\_seg}}$). Based on Observation 2 and knowledge of the implementation, we further perform a second order pattern recognition to further look for repeating patterns within a given segment. With the apriori knowledge of the order of execution of operations and their corresponding repetition counts, the attacker can look for patterns in the same order and same repetition counts. Refer to Figure 8(c) for the EM trace corresponding to a single iteration / segment. We can see that we are able to distinguish the repeating patterns within the segment and identify our target addition operation within a certain approximation. We earmark the repeating patterns on the trace in Figure 8(c) with the corresponding functions (with the corresponding repetition count in brackets) used in the reference implementation [19].

For a given message m to be signed, the identified $\text{loc}_{\text{last\_iter}}(m)$ is added to the estimated offset of the addition operation $\text{loc}_{\text{add}}(m)$ (i.e) $\text{t}_{\text{add}}(m) = \text{loc}_{\text{last\_iter}}(m) + \text{loc}_{\text{add}}(m)$ to estimate the location of the target addition operation from the start of the signing procedure. Considering the trigger-delay of about 130 nsec of our EM pulse generator, we identify a suitable time-window of sufficient length around $\text{t}_{\text{add}}(m)$ and simply sweep the fault injection process over the window until we are able to fault the individual coefficients of the signature component $\mathbf{z}$.

6.4 Fault Injection results

Considering all the three implementation variants, our attack requires to realize two different faults - *skip-STR* and *skip-ADD* faults. We scanned the entire top layer of the chip and could identify a precise location (close to the center of the chip near the ARM logo), where we could achieve *a 100% repeatability in skipping the store instruction.* With this achieved fault model, we were able to attack both the Variant-1 and Variant-2 implementations of Dilithium. But, we were not able to achieve practical faults to skip the ADD instruction required to attack Variant-3. Though we were not able to demonstrate a practical fault attack over Variant-3, we do not state this as a countermeasure against our attack since a more powerful fault attacker with enhanced capabilities like enhanced precision

[5]We advice the reader to zoom into Figure 8 to identify the repeating patterns. It is possible that the picture might pixelate upon printing.

and multiple laser injection might still be able to mount a successful fault attack on the Variant-3 implementation [29].

7 ZERO COST MITIGATION

There are certain generic countermeasures which provide protection against our attack. For example, double computation and verification-after-sign countermeasures provide preliminary protection against our attack but at a considerable increase in computational cost [9]. Use of additional randomness for sampling $\mathbf{y}$ can easily thwart our attack since it removes the deterministic assumption. But, the subsequent non-deterministic version of Dilithium is no longer secure in the quantum random oracle model [19]. Thus, in this section, we present a *zero-cost* mitigation strategy against our *skip-addition* fault attacks that only requires simple re-ordering of operations within the signing procedure.

We observe that the vulnerable *addition* operation is the last operation performed to generate the signature component $\mathbf{z}$. Targeting the addition operation thus ensures that the attacker can directly observe the effect of the injected fault from the faulty $\mathbf{z}$. Moreover, given that the addition is a point-wise operation in R_q, a single fault does not cause enough perturbation to be detected. These aspects enable the attacker to directly derive a relation between the faulty signatures and the primary secret $\mathbf{s}_1$, thus in-turn enabling the attack. We investigate the possibility of embedding the vulnerable addition operation deep enough inside the signing procedure to ensure that succeeding operations propagate the injected fault strong enough so that the faulty signatures never pass the rejection checks in the signing procedure.

7.1 Utilizing the Number Theoretic Transform to improve Fault Propagation

We observe that multiplication of polynomials in R_q in Dilithium is computed efficiently using the Number Theoretic Transform (NTT). The NTT operation is a deterministic linear transformation of a given polynomial from a normal domain to NTT domain in the same ring, $\text{NTT} : R_q \to R_q$. There is also an associated inverse NTT transformation (INTT) mapping polynomials back from NTT domain to normal domain. In the reference implementation of Dilithium, we observe that the addition operation within z_{gen} occurs in the normal domain. We know that ring R_q exhibits an isomorphism with itself under the NTT transform, that is $\text{NTT}(\mathbf{a}+\mathbf{b}) = \text{NTT}(\mathbf{a}) + \text{NTT}(\mathbf{b})\, \forall \mathbf{a}, \mathbf{b} \in R_q$. Moreover, we also take note of an interesting property of the NTT (resp. INTT) transform that every coefficient of the output is a unique linear combination of **all** the input coefficients, which we term as the *diffusion* property of the NTT transform. We thus examine the prospect of performing the addition in the NTT domain and further invert the result into the normal domain using the INTT transform.

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

We henceforth denote the NTT representation of a polynomial $\mathbf{x}$ as $\overline{\mathbf{x}}$, that is $\overline{\mathbf{x}} = \text{NTT}(\mathbf{x})$. In the above case, the INTT operation is performed after the addition of polynomials in the NTT domain. A detailed description of the NTT operation is provided in Appendix C. Lets say an attacker successfully faulted the addition operation in the NTT domain introducing a perturbation of $\delta_t > 0$ at the t^{th} coefficient. On applying the INTT transformation, its *diffusion* property ensures that the fault δ_t is effectively propagated to all the coefficients of the output polynomial, which in our case is $\hat{\mathbf{z}}$. This ensures that the coefficients of the faulty $\hat{\mathbf{z}}$ in the normal domain are all uniformly distributed in the range $[0, q-1]$. Since valid signatures are expected to satisfy the condition of $\|\mathbf{z}\|_\infty < \gamma_1 - \beta$, the faulted signatures are rejected by the signing procedure with a very high probability. The probability of acceptance of a faulty signature is estimated to be $\approx 2^{-4320}$ for recommended parameters of Dilithium.

But, we observe that there is one case when the addition operation is not faulted even on injection of a successful skipping fault, that is addition with a zero coefficient which results in $\delta_t = 0$. Here, we observe that our NTT-protected implementation is still attackable through a slight modification in the attack approach. Faulty signatures are never output due to the fault propagation by the INTT transform, hence the attacker should look for safe errors (valid-signatures) upon successful fault injection. On identification of a safe error in coefficient t, the attacker can construct a corresponding equation of the form $(\overline{\mathbf{s}_1\mathbf{c}})_t = 0$ (Case-1) or $(\overline{\mathbf{s}_1\mathbf{c}})_t = (\overline{\mathbf{z}})_t$ for Case-2 ($\because (\overline{\mathbf{s}_1\mathbf{c}})_t = 0$). Both equations are again linear equations in N variables with the unknowns being the coefficients of the primary secret $\mathbf{s}_1$. Thus, collecting N such equations will lead to successful recovery of the primary secret. Given that the coefficients of $\overline{\mathbf{y}}$ and $\overline{\mathbf{s}_1\mathbf{c}}$ are uniformly distributed in $[0, q-1]$, the probability to observe a zero coefficient in either $\overline{\mathbf{y}}$ or $\overline{\mathbf{s}_1\mathbf{c}}$ is about $1/q$, which equals about 2^{-23} for Dilithium and thus observation of n safe errors for key recovery will require to run the signing procedure $q \times \ell \times n$ times which amounts to about 2^{33} signing procedures for the recommended parameter set of Dilithium, while our proposed attack on the previous cases only required about $\ell \times n$ signatures, that is, $\approx 2^{10}$ signing procedures for recommended parameters. Though the theoretical increase is only about q, the large size of $q \approx 2^{23}$ exponentially increases the attacker's complexity.

7.2 Evaluation of our Mitigation Approach

We performed fault simulations of our NTT-protected implementation for about 2^{25} runs of the signing procedure and estimated that about 2^{30} signatures are required to collect enough information to construct the $\ell \times n$ equations for recovery of the primary secret of Dilithium. Referring to the results from Table 2, we can infer that attacking our NTT-protected implementation requires around **20 years** of signing time, under the same attack model, just to observe enough safe-errors to recover the primary secret $\mathbf{s}_1$ of Dilithium. This underlines an increase in attacker's complexity of 2^{20} in terms of both computational time and effort.

Implementing the mitigation comes at *zero-cost* since it involves only simple re-ordering of instructions within the z_{gen} step. To the best of our knowledge, we present the first use case of the NTT transform being used as a countermeasure against fault attacks. We thus propose our algorithm-level mitigation technique as a concrete countermeasure against attacks that possibly target the vulnerable secret-dependent addition operation in lattice-based signature schemes.

[6]The average signing time of the reference implementation is observed to be about 0.6067 secs on ARM Cortex-M4 running at 24 MHz.

Table 2: Efficacy of our Zero-cost mitigation technique against *skip-addition* fault attack. We provide the the numbers for recommended parameters of Dilithium assuming faults injected with 100% repeatability.

Implementation Type	No. of Faulted Signatures	Estimated Total Signing Time[6]
Reference	1024	621.26 secs
NTT-Protected	$\approx \mathbf{2^{30}}$	6.44×10^8 secs (≈**20 years**)

8 CONCLUSION

In this work, we have extended the practicality and applicability of *skip-addition* fault attacks to deterministic lattice-based signature schemes. We demonstrate practical fault attacks against deterministic variants of Dilithium and qTESLA signature schemes, which require only single-targeted faults in the signing procedure that allows key recovery and subsequently an existential forgery attack. We further demonstrate the efficacy of our attack against two well-known countermeasures used to protect against the *skip-addition* fault attack. We perform experimental validation of our attack using Electromagnetic fault injection over implementations from the pqm4 library on the ARM Cortex-M4 microcontroller. Finally, we also propose a zero-cost mitigation strategy using the NTT operation that exponentially increases the attacker's complexity to protect against *skip-addition* fault attacks and possibly against attacks that target the vulnerable addition operation in lattice-based signature schemes.

REFERENCES

[1] Sedat Akleylek, Nina Bindel, Johannes Buchmann, Juliane Krämer, and Giorgia Azzurra Marson. 2016. An efficient lattice-based signature scheme with provably secure instantiation. In *International Conference on Cryptology in Africa.* Springer, 44–60.

[2] Christopher Ambrose, Joppe W Bos, Björn Fay, Marc Joye, Manfred Lochter, and Bruce Murray. 2018. Differential attacks on deterministic signatures. In *Cryptographers' Track at the RSA Conference.* Springer, 339–353.

[3] Shi Bai and Steven D Galbraith. 2014. An Improved Compression Technique for Signatures Based on Learning with Errors.. In *CT-RSA*, Vol. 8366. 28–47.

[4] Rami Barends, Julian Kelly, Anthony Megrant, Andrzej Veitia, Daniel Sank, Evan Jeffrey, Ted C White, Josh Mutus, Austin G Fowler, Brooks Campbell, et al. 2014. Superconducting quantum circuits at the surface code threshold for fault tolerance. *Nature* 508, 7497 (2014), 500–503.

[5] Alessandro Barenghi and Gerardo Pelosi. 2016. A note on fault attacks against deterministic signature schemes (short paper). In *International Workshop on Security.* Springer, 182–192.

[6] Daniel J Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. 2012. High-speed high-security signatures. *Journal of Cryptographic Engineering* 2, 2 (2012), 77–89.

[7] Nina Bindel, Sedat Akleylek, Erdem Alkim, Paulo S. L. M. Barreto, Johannes Buchmann, Edward Eaton, Gus Gutoski, Juliane Kramer, Patrick Longa, Harun Polat, Jefferson E. Ricardini, and Gustavo Zanon. 2017. *qTESLA.* Technical Report. National Institute of Standards and Technology. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-1-submissions.

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

[9] Nina Bindel, Juliane Kramer, and Johannes Schreiber. 2017. Special session: hampering fault attacks against lattice-based signature schemes-countermeasures and their efficiency. In *Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2017 International Conference on.* IEEE, 1–3.

[10] Leon Groot Bruinderink, Andreas Hülsing, Tanja Lange, and Yuval Yarom. 2016. Flush, Gauss, and Reload–a cache attack on the BLISS lattice-based signature scheme. In *International Conference on Cryptographic Hardware and Embedded Systems.* Springer, 323–345.

[11] Leon Groot Bruinderink and Peter Pessl. 2018. Differential Fault Attacks on Deterministic Lattice Signatures. *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2018, 3 (2018). https://eprint.iacr.org/2018/355.pdf.

[12] Léo Ducas, Alain Durmus, Tancrède Lepoint, and Vadim Lyubashevsky. 2013. Lattice signatures and bimodal Gaussians. In *Advances in Cryptology–CRYPTO 2013.* Springer, 40–56.

[13] Thomas Espitau, Pierre-Alain Fouque, Benoît Gérard, and Mehdi Tibouchi. 2016. Loop-abort faults on lattice-based Fiat-Shamir and hash-and-sign signatures. In *International Conference on Selected Areas in Cryptography.* Springer, 140–158.

[14] Thomas Espitau, Pierre-Alain Fouque, Benoît Gérard, and Mehdi Tibouchi. 2017. Side-channel attacks on BLISS lattice-based signatures: Exploiting branch tracing against strongswan and electromagnetic emanations in microcontrollers. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security.* ACM, 1857–1874.

[15] Thomas Espitau, Pierre-Alain Fouque, Benoit Gerard, and Mehdi Tibouchi. 2018. Loop-Abort Faults on Lattice-Based Signatures and Key Exchange Protocols. *IEEE Trans. Comput.* (2018).

[16] Tim Güneysu, Vadim Lyubashevsky, and Thomas Pöppelmann. 2012. Practical lattice-based cryptography: A signature scheme for embedded systems. In *International Conference on Cryptographic Hardware and Embedded Systems.* Springer, 530–547.

[17] Matthias J. Kannwischer, Joost Rijneveld, Peter Schwabe, and Ko Stoffelen. [n. d.]. PQM4: Post-quantum crypto library for the ARM Cortex-M4. https://github.com/mupq/pqm4.

[18] Vadim Lyubashevsky. 2009. Fiat-Shamir with aborts: Applications to lattice and factoring-based signatures. In *International Conference on the Theory and Application of Cryptology and Information Security.* Springer, 598–616.

[19] Vadim Lyubashevsky, Leo Ducas, Eike Kiltz, Tancrede Lepoint, Peter Schwabe, Gregor Seiler, and Damien Stehle. 2017. *CRYSTALS-Dilithium.* Technical Report. National Institute of Standards and Technology. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-1-submissions.

[20] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. 2013. On Ideal Lattices and Learning with Errors over Rings. *J. ACM* 60, 6 (2013), 43.

[21] Daniele Micciancio. 2007. Generalized compact knapsacks, cyclic lattices, and efficient one-way functions. *computational complexity* 16, 4 (2007), 365–411.

[22] National Institute of Standards and Technology. 2016. Post-Quantum Crypto Project. http://csrc.nist.gov/groups/ST/post-quantum-crypto/.

[23] National Institute of Standards and Technology. 2019. *Round 2 Submissions, Post-Quantum Cryptography.* Technical Report. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions.

[24] NIST. 2016. Submission Requirements and Evaluation Criteria for the Post-Quantum Cryptography Standardization Process. https://csrc.nist.gov/csrc/media/projects/post-quantum-cryptography/documents/call-for-proposals-final-dec-2016.pdf.

[25] Peter Pessl. 2016. Analyzing the shuffling side-channel countermeasure for lattice-based signatures. In *INDOCRYPT 2016.* Springer, 153–170.

[26] Peter Pessl, Leon Groot Bruinderink, and Yuval Yarom. 2017. To BLISS-B or not to be: Attacking strongSwan's Implementation of Post-Quantum Signatures. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security.* ACM, 1843–1855.

[27] Thomas Pornin. 2013. *Deterministic usage of the digital signature algorithm (DSA) and elliptic curve digital signature algorithm (ECDSA).* Technical Report.

[28] Prasanna Ravi, Debapriya Basu Roy, Shivam Bhasin, Anupam Chattopadhyay, and Debdeep Mukhopadhyay. 2019. Number "Not Used" Once-Practical Fault Attack on pqm4 Implementations of NIST Candidates. In *International Workshop on Constructive Side-Channel Analysis and Secure Design.* Springer, 232–250.

[29] Lionel Riviere, Zakaria Najm, Pablo Rauzy, Jean-Luc Danger, Julien Bringer, and Laurent Sauvage. 2015. High precision fault injections on the instruction cache of ARMv7-M architectures. *arXiv preprint arXiv:1510.01537* (2015).

[30] Elena Trichina and Roman Korkikyan. 2010. Multi fault laser attacks on protected CRT-RSA. In *Fault Diagnosis and Tolerance in Cryptography (FDTC), 2010 Workshop on.* IEEE, 75–86.

A DESCRIPTION OF DILITHIUM SIGNATURE SCHEME

In this section, we present the rounding algorithms, key generation and verification procedures of the Dilithium signature scheme.

B qTESLA SIGNATURE SCHEME

The qTESLA signature scheme is an improved practical variant of the Bai-Galbraith signature scheme [3]. It is also based on the "Fiat-Shamir with Aborts" framework and hence structurally very similar to Dilithium. The hardness guarantees of qTESLA are derived from the Ring-LWE and Ring-SIS problem based in the ring

Algorithm 4: Rounding Algorithms

```
Procedure Decompose $D_q(r, \alpha)$
    Compute $r = r \pmod q$
    Compute $r_0 = r \pmod{^{\pm} \alpha}$
    if $r - r_0 = q - 1$ then
        $r_1 = 0$
        $r_0 = r_0 - 1$
    else
        $r_1 = (r - r_0)/\alpha$
    end
    return $D_q(r, \alpha) = (r_1, r_0)$
```

```
Procedure HighBits $HB_q((r, \alpha))$
    Compute $(r_1, r_0) = D_q(r, \alpha)$
    return $HB_q(r, \alpha) = r_1$
```

```
Procedure MakeHint $MH_q(u, r, \alpha)$
    Compute $r_1 = HB_q(r, \alpha)$
    Compute $v_1 = HB_q(u + r, \alpha)$
    if $r_1 = v_1$ then
        $h = 0$
    else
        $h = 1$
    end
    return $MH_q(u, r, \alpha) = h$
```

```
Procedure UseHint $UH_q((h, r, \alpha))$
    Compute $m = (q - 1)/\alpha$
    Compute $(r_1, r_0) = D_q(r, \alpha)$
    if $h = 1$ and $r_0 > 0$ then
        $r_1 = (r_1 + 1) \pmod m$
    end
    else if $h = 1$ and $r_0 \leq 0$ then
        $r_1 = (r_1 - 1) \pmod{^{+} m}$
    else
        $r_1 = r_1$
    end
    return $UH_q(h, r, \alpha) = r_1$
```

$R_q = \mathbb{Z}_q[X]/[X^n + 1]$ with $n \geq 1024$. Refer Alg.6 for a brief algorithmic level description of the qTESLA signature scheme. As stated earlier[1], our attacks only apply the deterministic variant of qTESLA and since its updated specification is a probabilistic one and hence is inherently protected against our attacks.

B.1 Adapting our attacks to qTESLA

The z_{gen} step (i.e) $\mathbf{z} = \mathbf{s}_1 \cdot \mathbf{c} + \mathbf{y}$ in Line 10 of Sign in Alg 6, which is of most interest to our attack computes almost identical operations as that of Dilithium due to use of the same operating base ring R_q. We analyzed the reference implementation of qTESLA signature scheme taken from the *pqm4* library and found that the z_{gen} operation is implemented similar to Variant-3 wherein the result ($\mathbf{z}$) of the target addition operation is stored in a new variable with the order of operands similar to Case-2 ($\mathbf{z} = \mathbf{y}$ upon faulting) of the addition operation. Thus, our fault attacks demonstrated for Dilithium directly apply to qTESLA, albeit with different fault complexities due to differing conditional checks and parameter sets.

Algorithm 5: Dilithium Signature scheme

```
Procedure KeyGen()
    $\rho, \rho' \leftarrow \{0,1\}^{256}$
    $K \leftarrow \{0,1\}^{256}$
    $N = 0$
    for $i$ from $0$ to $\ell - 1$ do
        $\mathbf{s}_1[i] = \mathsf{Sample}(\mathsf{PRF}(\rho', N))$
        $N := N + 1$
    end
    for $i$ from $0$ to $k - 1$ do
        $\mathbf{s}_2[i] = \mathsf{Sample}(\mathsf{PRF}(\rho', N))$
        $N := N + 1$
    end
    $\mathbf{a} \sim R_q^{k \times \ell} = \mathsf{ExpandA}(\rho)$
    $\mathbf{t} = \mathbf{a} \cdot \mathbf{s}_1 + \mathbf{s}_2$
    $\mathbf{t}_1 = \mathsf{Power2Round}_q(\mathbf{t}, d)$
    $tr \in \{0,1\}^{384} = \mathsf{CRH}(\rho || \mathbf{t}_1)$
    return $pk = (\rho, \mathbf{t}_1), sk = (\rho, K, tr, \mathbf{s}_1, \mathbf{s}_2, \mathbf{t}_0)$
```

```
Procedure Verify(pk, $M$, $\sigma = (\mathbf{z}, \mathbf{h}, \mathbf{c})$)
    $\mathbf{a} \in R_q^{k \times \ell} := \mathsf{ExpandA}(\rho)$
    $\mu = \mathsf{CRH}\,(\mathsf{CRH}(\rho || \mathbf{t}_1) || M)$
    $\mathbf{w}_1 := \mathsf{UH}_q(\mathbf{h}, \mathbf{a} \cdot \mathbf{z} - \mathbf{c} \cdot \mathbf{t}_1 \cdot 2^d, 2\gamma_2)$
    if $\mathbf{c} = H(\mu, \mathbf{w}_1)$ and $||\mathbf{z}||_\infty < \gamma_1 - \beta$ and $\mathrm{wt}(\mathbf{h}) \leq \omega$ then
        return 1
    else
        return 0
    end
```

Unlike Dilithium, the signatures of qTESLA only contain two components ($\mathbf{z}, \mathbf{c}$). Moreover, the LWE instance $\mathbf{t} \in R_q$ computed in the key-generation procedure is directly output as the public key *pk* and hence retrieval of $\mathbf{s}$ through the *skip-addition* fault attack results in a direct break of the signature scheme. Since the NTT operation is used for polynomial multiplication, our *zero-cost* mitigation technique exploiting the fault propagation characteristics of the NTT operation is also applicable for concrete protection of qTESLA against *skip-addition* fault attacks.

C NUMBER THEORETIC TRANSFORM

The Number Theoretic Transform (abbreviated as NTT) is widely used in increasing the efficiency of multiple lattice-based schemes, including the Dilithium and qTESLA signature schemes. The NTT operation enables the polynomial multiplication in the ring to be done in quasilinear time ($O(n\log(n))$) compared to the quadratic time ($O(n^2)$) time for the schoolbook polynomial multiplier. The operand polynomials are first converted into their respective representations in the NTT domain. They are then further multiplied point-wise in the NTT domain after which the final product is

Algorithm 6: qTESLA Signature scheme

```
1  Procedure KeyGen()
2      $seed_a, seed_y \xleftarrow{\$} \{0,1\}^k$
3      $\mathbf{a} \in R_q \leftarrow \mathsf{GenA}(seed_a)$
4      Sample :
5      $\mathbf{s}, \mathbf{e} \in R_q \leftarrow \mathbf{D}_\sigma$
6      $\mathbf{t} \in R_q = \mathbf{a} \cdot \mathbf{s} + \mathbf{e}$
7      while $(\mathsf{checkS}(\mathbf{s}) || \mathsf{checkE}(\mathbf{e}) = 0)$ do
8          Goto Sample
9      end
10     return $pk = (seed_a, \mathbf{t}), sk = (\mathbf{s}, \mathbf{e}, seed_y, seed_a)$
11 ____________________________________________
1  Procedure Sign(sk, M)
2      $\mathbf{a} \leftarrow \mathsf{GenA}(seed_a)$
3      $counter = 0$
4      $rand = \mathsf{PRF}_1(seed_y, m)$
5      Rej :
6      $\mathbf{y} \in R_q \leftarrow \mathsf{PRF}_2(rand, counter)$
7      $\mathbf{v} \in R_q = \mathbf{a} \cdot \mathbf{y} \pmod q$
8      $\mathbf{c} \in R_q = \mathsf{Enc}(H(\mathsf{Round}(\mathbf{v}), M))$
9      $\mathbf{z} \in R_q = \mathbf{s} \cdot \mathbf{c} + \mathbf{y}$
10     if $(\mathsf{Reject}(\mathbf{z}) = 0)$ then
11         $counter = counter + 1$
12         Goto Rej
13     end
14     $\mathbf{w} \in R_q = \mathbf{v} - \mathbf{e} \cdot \mathbf{c}$
15     if $(\mathsf{Reject}(\mathbf{w}) = 0)$ then
16         $counter = counter + 1$
17         Goto Rej
18     end
19     return $\sigma = (\mathbf{z}, \mathbf{c})$
20 ____________________________________________
1  Procedure Verify(pk, M, $\sigma = (\mathbf{z}, \mathbf{c})$)
2      $\mathbf{a} \leftarrow \mathsf{GenA}(seed_a)$
3      $\mathbf{w} = \mathbf{a} \cdot \mathbf{z} - \mathbf{t} \cdot \mathbf{c} \pmod q$
4      return $\mathbf{c} = H(\mathsf{Round}(\mathbf{w}), M)$
```

obtained through an inverse NTT operation. The NTT operation in itself is a bijective mapping from one polynomial to another in the same operating ring. An input sequence $\mathbf{p}$ with n elements $(\mathbf{p}_0, \ldots, \mathbf{p}_{n-1})$ is mapped to its representation $\bar{\mathbf{p}}$ in the NTT domain as

$$\bar{\mathbf{p}}_j = \sum_{i=0}^{n-1} \mathbf{p}_i \cdot \omega^{i \cdot j} \tag{8}$$

where $j \in [0, n-1]$ and ω being the n^{th} root of unity in the operating ring $\mathbb{Z}_q$.

Referring to Eqn.8 which represents the NTT operation, we can see that every element of the output polynomial in NTT domain is a unique function of all the elements of the input polynomial. The same also applies for the inverse NTT operation. Refer Figure 9 for the data-flow graph of the NTT, where we can clearly visualize this

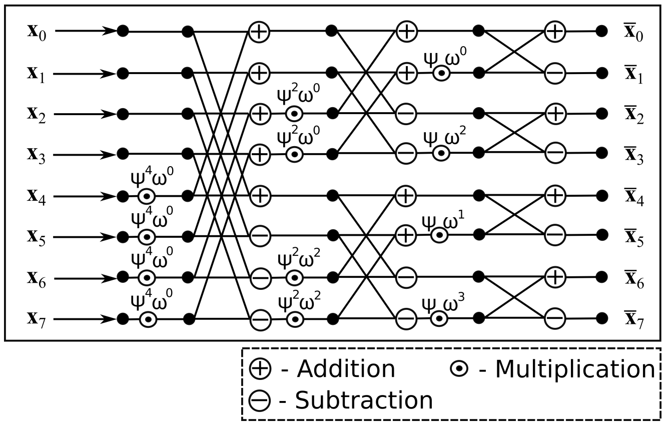


Figure 9: NTT operation on a polynomial x with degree 8

dependency between elements of the input and the output of the NTT transform. We leverage over this property, which we refer to as the *diffusion* property of the NTT operation to ensure that the fault injected in the addition operation for one element is propagated uniformly to all the elements in the signature component $\mathbf{z}$. This further ensures that the resulting faulty signatures are rejected by the signing procedure with very high probability.