# A Single Trace Fault Injection Attack on Hedged CRYSTALS-Dilithium

Sönke Jendral[1,2]

[1] KTH Royal Institute of Technology, Stockholm, Sweden
jendral@kth.se
[2] Ericsson Research, Stockholm, Sweden

**Abstract.** CRYSTALS-Dilithium is a post-quantum secure digital signature algorithm currently being standardised by NIST. As a result, devices making use of CRYSTALS-Dilithium will soon become generally available and be deployed in various environments. It is thus important to assess the resistance of CRYSTALS-Dilithum implementations to physical attacks. In this paper, we present an attack on a CRYSTALS-Dilithium implementation in hedged mode in ARM Cortex-M4 using fault injection. Voltage glitching is performed to skip computation of a seed during the generation of the signature. We identified settings that consistently skip the desired function without crashing the device. After the successful fault injection, the resulting signature allows for the extraction of the secret key vector. Our attack succeeds with probability 0.582 in a single trace. We also propose countermeasures against the presented attack.

**Keywords:** Fault injection · CRYSTALS-Dilithium · ML-DSA · Post-quantum digital signature · Key recovery attack

## 1 Introduction

CRYSTALS-Dilithium is a lattice-based digital signature scheme that is strongly unforgeable under chosen message attack (SUF-CMA)-secure in the classical and random oracle models [3]. SUF-CMA security means that an adversary with the public key and access to a signing oracle cannot produce a signature for a new message nor produce a different signature for a message that they have already seen. The security of Dilithium is based on the assumed hardness of the Module Learning-with-Errors (MLWE) and Module Shortest-Integer-Solution (MSIS) problems.

In 2022, Dilithium was selected by the National Institute of Standards and Technology (NIST) to be standardised as a new digital signature scheme under the name ML-DSA [29]. Dilithium was also added to the suite of cryptographic algorithms recommended for national security systems by the National Security Agency (NSA) [1]. With the standardisation, NIST introduced a "hedged" pseudorandom sampling procedure for the private seed $\rho'$ that replaces the deterministic sampling procedure as the default. A deterministic variant of the sampling

procedure is retained for ML-DSA, but the fully-randomised sampling method has been removed entirely. A number of fault injection attacks on Dilithium which allow for the extraction of secret key vectors have been demonstrated in the past [33,22,35,38,11]. However, to our knowledge, the differences between CRYSTALS-Dilithium and ML-DSA have not been assessed in this context.

In our digitalised world, devices running cryptographic algorithms are increasingly physically accessible to attackers. Often such devices operate in resource-constrained scenarios, limiting the available security features. Simultaneously, adoption of quantum resistant cryptographic algorithms is expected to proceed rapidly. The 3GPP intends to introduce quantum resistant algorithms to 5G as soon as final standards are published, while the NSA requires network equipment operating as part of US national security systems to support quantum resistant algorithms by 2026, and all other equipment by 2030 [26].

It is therefore important to assess the vulnerability of ML-DSA implementations to physical attacks, such as fault injection, to give implementers the opportunity to address security issues.

**Contributions:** In this paper, we present a fault injection attack on an implementation of Dilithium in hedged mode. The presented attack requires only a single faulty signature to recover the secret key vector. Previous approaches making use of fault injection have not been applicable to non-deterministic versions of Dilithium, or required a higher number of signatures and faults, or generated signatures that do not pass the verification and can thus be detected.

We demonstrate practical key recovery on a modified version of the software implementation of CRYSTALS-Dilithium by Abdulrahman et al. [2] to which the hedged mode has been added. A voltage fault injection using the crowbar technique of [30] is performed to skip absorption of data during the computation of the hash $\rho'$. We identified settings that consistently skip the desired function without crashing the device or disrupting other steps of the signature generation. From there, the secret key vector can be extracted from the generated signature directly. Our attack succeeds with recovery of the secret key vector from a single attempt with a probability of 52.8%.

In Dilithium, recovery of $\mathbf{s}_1$ is sufficient to achieve existential forgery [33], as an adversary may recompute the hint $\mathbf{h}$ to create a signature that passes verification.

**Organisation of the paper:** The rest of this paper is organised as follows. Section 2 describes previous work. Section 3 provides background information on the ML-DSA algorithm and voltage fault injection. Section 4 describes the adversary model and attack scenario. Section 5 describes the experimental setup. Section 6 describes the fault attack. Section 7 describes the secret key recovery method. Section 8 summarises the experimental results. Section 9 discusses possible countermeasures against the attack. Section 10 concludes the paper.

## 2 Previous work

In this section we provide an overview over previous attacks on Dilithium making use of fault injection or side-channel analysis.

Bindel et al. [8] presented a number of fault injection attacks against lattice-based signature schemes. One of their approaches is a randomisation attack changing individual coefficients of $\mathbf{s}_1$, allowing for the recovery from multiple signatures. They also propose skipping the addition during the computation of $\mathbf{z} = \mathbf{y} + c\mathbf{s}_1$, thus allowing for the recovery of $\mathbf{s}_1$ from the same signature. However, their attacks do not target Dilithium and have not been experimentally verified against it. It therefore remains unclear how applicable their approaches are in practice.

In a subsequent work, Ravi et al. [33] demonstrated that skipping the entire addition of $\mathbf{y}$ is not necessary. Instead, they present a fault injection attack targeting the addition of single coefficients. This allows for recovery of the full vector $\mathbf{s}_1$ in around 1-2k faulty signatures (corresponding to the same number of traces, as they report a fault probability of 100%).

Most recently, Krahmer et al. [22] extended this addition-skipping attack to randomised Dilithium and additionally presented an attack targeting the matrix $\hat{\mathbf{A}}$. As both approaches perturb single coefficients in the computation of $\mathbf{z} = \mathbf{y} + c\mathbf{s}_1$ in the signing procedure, they are able to use the verification to exhaustively search and correct the perturbation, thus allowing for recovery of a single coefficient. They practically demonstrate recovery of the full vector $\mathbf{s}_1$ using clock glitching in 22952 traces.

Separately, Ravi et al. [35] proposed a fault attack exploiting zeroisation of twiddle constants in the Number Theoretic Transform (NTT). They presented two different attacks targeting deterministic and randomised Dilithium. The first zeroises the NTT of $c$, thus allowing for recovery of $\mathbf{y}$, which can be used to extract $\mathbf{s}_1$ from a different signature. The second attack requires computation of the signature in the NTT domain and zeroises most of the coefficients of $\mathbf{y}$, directly allowing recovery from the signature. They report recovery of the full secret key vector $\mathbf{s}_1$ from 13 respective 3 faulted signatures. While they do not report the number of traces required to gather the necessary signatures, they report a fault probability of 26% respective 51% for the attacks.

Espitau et al. [14] proposed a fault injection attack targeting the generation of $\mathbf{y}$ in a number of Fiat-Shamir with Aborts-based signature schemes. Their approach works by aborting the loop that performs the sampling of $\mathbf{y}$, thus leaving a number of coefficients uninitialised. A signature generated from such a faulty vector will likely directly contain several of the coefficients of the product $c\mathbf{s}_1$, allowing for the recovery of the secret key vector. This approach was extended by Ulitzsch et al. [38] to an implementation of Dilithium with fault countermeasures. They show that using an Integer Linear Program, they are able to recover the full vector $\mathbf{s}_1$ using 5 faulted signatures gathered from 53 traces using clock glitching.

Bruinderink et al. [11] demonstrated a differential fault attack against the deterministic version of Dilithium. After generating a signature $\mathbf{z}$ for a message,

they use fault injection to induce nonce-reuse to obtain a second signature $\mathbf{z}'$ computed for the same $\mathbf{y}$, but a different $c$. The secret key vector can then be recovered from the difference of the signatures $\mathbf{z} - \mathbf{z}'$. While they do not report the number of traces necessary for exploitation, their attack works with faults anywhere in a large range of the execution, thus making it very likely the faulty signature could be obtained from a single attempt, thus allowing for recovery of the full key $\mathbf{s}_1$ from only two signatures.

There have also been several side-channel attacks on Dilithium that target recovery of the secret key [32,15,12,27,20,18,25,7,31]. These attacks are generally characterised by a larger number of traces required for successful key recovery. One noteworthy exception is the recent work by Wang et al. [39] who demonstrated recovery of the secret key vector $\mathbf{s}_1$ in a single trace through side-channel power analysis. They exploit leakage of the coefficients of secret key vectors $\mathbf{s}_1$ and $\mathbf{s}_2$ during the unpacking of the secret key at the beginning of the signing procedure, thus their attack is also applicable to implementations using hedged mode. By recovering half of these coefficients, they are able to solve a system of linear equations that enables recovery of the full vector $\mathbf{s}_1$ with probability of 9% from a single trace.

## 3 Background

This section describes the notation used in the remainder of this work, the ML-DSA algorithm specification, and the voltage fault injection method.

### 3.1 Notation

We will denote the ring of integers modulo $q$ as $\mathbb{Z}_q$, the ring of polynomials $\mathbb{Z}_q[X]/(X^n+1)$ as $R_q$ and the ring $\mathbb{Z}_q^n$ as $T_q$. Regular font letters denote elements in $\mathbb{Z}_q$ or $R_q$, bold font letters denote vectors with coefficients in $R_q$, the hat symbol denotes elements in $T_q$ and upper-case letters are used for matrices. We use $w_i$ to denote the $i$th coefficient of the polynomial $w = w_0 + w_1X + \cdots + w_{255}X^{255}$ and $v[i]$ to denote the $i$th entry of a vector $\mathbf{v}$. The infinity-norm is given by $||\cdot||_\infty$, the concatenation of bit/byte strings $a$ and $b$ is given by $a||b$. To denote boolean evaluation of an expression, we use $[\![\cdot]\!]$. The multiplication in $T_m$ is denoted by $\circ$ and the multiplication in $\mathbb{Z}_q$ or $R_q$ is denoted by $\cdot$. Assignment from the result of a function or sampling from a set are denoted by $\leftarrow$. The blank symbol $\perp$ is used to indicate lack of an output.

### 3.2 ML-DSA algorithm

ML-DSA is derived from the latest version of CRYSTALS-Dilithium [3], and differs in an increased length for parameters $tr$ and $\hat{c}$ in parameter sets ML-DSA-65 and ML-DSA-87, as well as the introduction of a "hedged" pseudorandom sampling procedure for the private seed $\rho'$ that replaces the deterministic sampling procedure as the default. A modified variant of the deterministic sampling

**Table 1.** ML-DSA parameter sets from [29].

| Parameter set | $n$ | $q$ | $d$ | $\tau$ | $\gamma_1$ | $\gamma_2$ | $(k,l)$ | $\eta$ | $\beta$ | $\omega$ |
|---|---|---|---|---|---|---|---|---|---|---|
| ML-DSA-44 | 256 | 8380417 | 13 | 39 | $2^{17}$ | $(q-1)/88$ | $(4,4)$ | 2 | 78 | 80 |
| ML-DSA-65 | 256 | 8380417 | 13 | 49 | $2^{19}$ | $(q-1)/32$ | $(6,5)$ | 4 | 196 | 55 |
| ML-DSA-87 | 256 | 8380417 | 13 | 60 | $2^{19}$ | $(q-1)/32$ | $(8,7)$ | 2 | 120 | 75 |

procedure is, however, retained for ML-DSA, while the previously present fully-randomised sampling method has been removed entirely. This paper focuses on the specifics of ML-DSA. Going forward, we will use the term Dilithium to refer to both CRYSTALS-Dilithium and ML-DSA, the term CRYSTALS-Dilithium to refer to the pre-standardisation submission to the NIST PQC project, and the term ML-DSA to refer to the variant currently being standardised.

An overview over the possible sets of parameters is given in Tab. 1. For further details we refer to the specification [29]. We will be focusing on ML-DSA-44 (respective Dilithium-2) in this paper, though other variations can be approached similarly.

Dilithium is considered secure in the (Quantum) Random Oracle model based on the assumed hardness of the Module Learning-with-Errors (MLWE) and Module Shortest-Integer-Solution problems [23,19]. The scheme uses the Fiat-Shamir with Aborts approach [24] in which an identification scheme is transformed into a signature scheme and rejection sampling is applied to sample a mask that prevents the secret key from being revealed through the signature.

The main components of the Dilithium scheme are the key generation procedure, the signing procedure and the verification procedure.

**Algorithm 1** ML-DSA.KeyGen() [3,29]

**Output:** Public key $pk$, private key $sk$

$\xi \leftarrow \{0,1\}^{256}$
$(\rho, \rho', K) \in \{0,1\}^{256} \times \{0,1\}^{512} \times \{0,1\}^{256} \leftarrow \mathrm{H}(\xi, 1024)$
$\hat{\mathbf{A}} \leftarrow \mathsf{ExpandA}(\rho)$
$(\mathbf{s}_1, \mathbf{s}_2) \leftarrow \mathsf{ExpandS}(\rho')$
$\mathbf{t} \leftarrow \mathsf{NTT}^{-1}(\hat{\mathbf{A}} \circ \mathsf{NTT}(\mathbf{s}_1)) + \mathbf{s}_2$
$(\mathbf{t}_0, \mathbf{t}_1) \leftarrow \mathsf{Power2Round}(\mathbf{t}, d)$
$pk \leftarrow \mathsf{pkEncode}(\rho, \mathbf{t}_1)$
$tr \leftarrow \mathrm{H}(\mathsf{BytesToBits}(pk), 512)$
$sk \leftarrow \mathsf{skEncode}(\rho, K, tr, \mathbf{s}_1, \mathbf{s}_2, \mathbf{t}_0)$
**return** $(pk, sk)$

**Key generation (Alg. 1)** The key generation samples the matrix $\mathbf{A}$ and private key polynomial vectors $\mathbf{s}_1$ and $\mathbf{s}_2$ by generating and expanding a random

seed using SHAKE256. The coefficients of $\mathbf{s}_1$ and $\mathbf{s}_2$ are short, i.e. are in the range $[-\eta, \eta]$. It then computes $\mathbf{t} = \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2$, which forms part of the public key. Dilithium applies compression to $\mathbf{t}$ by dropping the $d$ least significant bits to reduce its size, but an attacker is assumed to be able to recover the full value of $\mathbf{t}$. The most significant bits $\mathbf{t}_1$ and the seed $\rho$ used for expanding the matrix $\mathbf{A}$ form the public key. The private key includes the seed $\rho$, in addition to a private random seed $K$ and the hash $tr$ of the public key for use during signing, as well as vectors $\mathbf{s}_1$ and $\mathbf{s}_2$ and the $d$ least significant bits $\mathbf{t}_0$.

**Algorithm 2** ML-DSA.Sign$(sk, M)$ [29]

**Input:** Private key $sk$, message $M$
**Output:** Signature $\sigma$

$(\rho, K, tr, \mathbf{s}_1, \mathbf{s}_2, \mathbf{t}_0) \leftarrow \mathsf{skDecode}(sk)$
$\hat{\mathbf{s}}_1 \leftarrow \mathsf{NTT}(\mathbf{s}_1)$
$\hat{\mathbf{s}}_2 \leftarrow \mathsf{NTT}(\mathbf{s}_2)$
$\hat{\mathbf{t}}_0 \leftarrow \mathbf{NTT}(\mathbf{t}_0)$
$\hat{\mathbf{A}} \leftarrow \mathsf{ExpandA}(\rho)$
$\mu \leftarrow \mathrm{H}(tr||M, 512)$
$rnd \leftarrow \{0,1\}^{256}$ ▷ *In deterministic mode:* $rnd \leftarrow \{0\}^{256}$
$\rho' \leftarrow \mathrm{H}(K||rnd||\mu, 512)$
$\kappa \leftarrow 0$
$(\mathbf{z}, \mathbf{h}) \leftarrow \perp$
**while** $(\mathbf{z}, \mathbf{h}) = \perp$ **do**
  $\mathbf{y} \leftarrow \mathsf{ExpandMask}(\rho', \kappa)$
  $\mathbf{w} \leftarrow \mathsf{NTT}^{-1}(\hat{\mathbf{A}} \circ \mathsf{NTT}(\mathbf{y}))$
  $\mathbf{w}_1 \leftarrow \mathsf{HighBits}(\mathbf{w})$
  $\tilde{c} \in \{0,1\}^{2\lambda} \leftarrow \mathrm{H}(\mu||\mathsf{w1Encode}(\mathbf{w}_1), 2\lambda)$
  $(\tilde{c}_1, \tilde{c}_2) \in \{0,1\}^{256} \times \{0,1\}^{2\lambda-256} \leftarrow \tilde{c}$
  $c \leftarrow \mathsf{SampleInBall}(\tilde{c}_1)$
  $\hat{c} \leftarrow \mathsf{NTT}(c)$
  $cs_1 \leftarrow \mathsf{NTT}^{-1}(\hat{c} \circ \hat{\mathbf{s}}_1)$
  $cs_2 \leftarrow \mathsf{NTT}^{-1}(\hat{c} \circ \hat{\mathbf{s}}_2)$
  $\mathbf{z} \leftarrow \mathbf{y} + cs_1$
  $\mathbf{r}_0 \leftarrow \mathsf{LowBits}(\mathbf{w} - cs_2)$
  **if** $||\mathbf{z}||_\infty \geq \gamma_1 - \beta$ **or** $||\mathbf{r}_0||_\infty \geq \gamma_2 - \beta$ **then** $(\mathbf{z}, \mathbf{h}) \leftarrow \perp$
  **else**
    $c\mathbf{t}_0 \leftarrow \mathsf{NTT}^{-1}(\hat{c} \circ \hat{\mathbf{t}}_0)$
    $\mathbf{h} \leftarrow \mathsf{MakeHint}(-c\mathbf{t}_0, \mathbf{w} - c\mathbf{s}_2 + c\mathbf{t}_0)$
    **if** $||c\mathbf{t}_0||_\infty \geq \gamma_2$ **or** # of 1's in $\mathbf{h} > \omega$ **then** $(\mathbf{z}, \mathbf{h}) \leftarrow \perp$
  $\kappa \leftarrow \kappa + l$
$\sigma \leftarrow \mathsf{sigEncode}(\tilde{c}, \mathbf{z} \bmod^{\pm} q, \mathbf{h})$
**return** $\sigma$

**Signing (Alg. 2)** The signing procedure computes the message representative $\mu$ by hashing the hash of the public key and the message using SHAKE256. It then

computes an additional private random seed $\rho'$ by hashing the private random seed $K$, a 256-bit random value $rnd$ (or in deterministic mode, the value $\{0\}^{256}$) and the message representative $\mu$. The seed $\rho'$ is used to sample the nonce $\mathbf{y}$ from which the signer commitment $\mathbf{w}_1$ is computed as the high bits of $\mathbf{w} = \mathbf{A}\mathbf{y}$. The commitment hash $\tilde{c}$ is derived from $\mathbf{w}_1$ and $\mu$ and used to sample the challenge $c$. The signer's response is calculated as $\mathbf{z} = \mathbf{y} + c\mathbf{s}_1$ and its validity is checked to restart if necessary (i.e. the signing is aborted as per the Fiat-Shamir with Aborts approach [24]). Finally, the hint $\mathbf{h}$ is computed, which allows a verifier to reconstruct the entire value of $\mathbf{w}_1$. The values $\tilde{c}, \mathbf{z}$ and $\mathbf{h}$ form the signature $\sigma$.

**Algorithm 3** ML-DSA.Verify$(pk, M, \sigma)$ [3,29]

**Input:** Public key $pk$, message $M$, signature $\sigma$
**Output:** Boolean

1: $(\rho, \mathbf{t}_1) \leftarrow \mathsf{pkDecode}(pk)$
2: $(\tilde{c}, \mathbf{z}, \mathbf{h}) \leftarrow \mathsf{sigDecode}(\sigma)$
3: **if** $\mathbf{h} = \perp$ **then return** false
4: $\hat{\mathbf{A}} \leftarrow \mathsf{ExpandA}(\rho)$
5: $tr \leftarrow \mathrm{H}(\mathsf{BytesToBits}(pk), 512)$
6: $\mu \leftarrow \mathrm{H}(tr||M, 512)$
7: $(\tilde{c}_1, \tilde{c}_2 \in \{0,1\}^{256} \times \{0,1\}^{2\lambda-256} \leftarrow \tilde{c}$
8: $c \leftarrow \mathsf{SampleInBall}(\tilde{c}_1)$
9: $\mathbf{w}'_{\mathrm{Approx}} \leftarrow \mathsf{NTT}(\hat{\mathbf{A}} \circ \mathsf{NTT}(\mathbf{z}) - \mathsf{NTT}(c) \circ \mathsf{NTT}(\mathbf{t}_1 \cdot 2^d))$
10: $\mathbf{w}'_1 \leftarrow \mathsf{UseHint}(\mathbf{h}, \mathbf{w}'_{\mathrm{Approx}})$
11: $\tilde{c}' \leftarrow \mathrm{H}(\mu||\mathsf{w1Encode}(\mathbf{w}'_1), 2\lambda)$
12: **return** $[\![||\mathbf{z}||_\infty < \gamma_1 - \beta]\!]$ **and** $[\![\tilde{c} = \tilde{c}']\!]$ **and** $[\![$ # of 1's in $\mathbf{h} \leq \omega]\!]$

**Verification (Alg. 3)** The verification procedure derives the challenge $c$ from the signature and computes $\mathbf{w}'_{\mathrm{Approx}} = \mathbf{A}\mathbf{y} - c\mathbf{t}_1 \cdot 2^d$. Using the hint $\mathbf{h}$, the value $\mathbf{w}'_1$ is reconstructed from $\mathbf{w}'_{\mathrm{Approx}}$ and the commitment hash $\tilde{c}'$ is derived. The verification passes if the commitment hashes match and the additional validity criteria are fulfilled.

### 3.3 Fault injection techniques

Among the different techniques for fault injection, four main techniques for performing low-cost, minimally invasive fault injection can be identified in the form of software-based glitching, Electromagnetic (EM) glitching, clock glitching and voltage glitching [6,30].

**Software-based glitching** exploits properties of the executing hardware available through software. The various approaches differ from each other substantially, but include, for example, use of the Dynamic Voltage and Frequency

Scaling feature to induce byte-corruption faults through overclocking, as in the CLKscrew attack [37], or bit flips caused by repeated row access in a DRAM chip, as in the RowHammer attacks [21].

**EM glitching** involves precise placement of a probe from which an electromagnetic pulse is emitted. This pulse can affect clock signals or inject additional power into the chip [10]. Several fault injection attacks on Dilithium [35,34,36] employ EM glitching, making use of both its ability to skip instructions by affecting the clock and its ability to flip/zero bits in memory.

**Clock glitching** exploits manipulation of the clock signal by injecting or withholding rising edges, thus altering the execution of instructions [10,30]. Clock glitching has been employed in several attacks on Dilithium [11,16,38,22], which make use of its ability to skip instructions.

**Voltage fault injection** uses manipulation of the power to a processor to cause faults. Some approaches [5,4] use uniform underpowering to slow down logic gates to cause faults. These approaches do not require precise timing control, but also offer limited control over affected instructions and the resulting fault. Other approaches [30,9] instead use a precisely timed spike in the voltage to cause faults. While these approaches thus require greater timing control, they also allow specific instructions to be affected.

In this paper, we use the voltage fault injection technique of O'Flynn [30]. This technique uses a crowbar circuit to short the power rails of the processor, which induces oscillations in the target circuit, thus causing faults.

**Fault model** Using the previously described voltage fault injection technique, the resulting faults will include both (single/multiple) instruction skipping and instruction corruption. While our attack does not make use of instruction corruption, we empirically observed instances of corrupted instructions mentioned here for completeness.

## 4 Adversary model and attack scenario

In this section we define assumptions about the adversary, their capabilities and goals in accordance with the adversary model in [13].

**Assumptions:** We assume that the adversary has physical access to the device under attack and access to equipment that allows for fault injection to be performed during the execution of the signing procedure. We further assume the adversary to be an outsider without access to privileged information that has a high-level understanding of the implementation of Dilithium used on the device under attack (e.g. from reverse-engineering).

**Capabilities:** The adversary is capable of triggering execution of the signing procedure on the device under attack and inject a precise voltage fault during the execution of the procedure, as well as observe its output. Note that the adversary does not have to be able to control the message being signed, only triggering the signing and observing its output are necessary.

**Goals:** The goal of the adversary is to perform existential forgery, i.e. to generate a signature $\sigma$ for an adversary-chosen message $M$ that passes verification using the public key of the device under attack. Note that it has been demonstrated that knowledge of secret key vector $\mathbf{s}_1$ is sufficient to achieve existential forgery [33].

### 4.1 Attack scenario

The attacker triggers execution of the ML-DSA.Sign procedure on the device under attack. During the computation of the private random seed $\rho'$, a fault is injected to fix its value to a known constant. The attacker then observes the generated signature $\sigma$ and extracts from it the secret key vector $\mathbf{s}_1$. Note that fixing the value of $\rho'$ does not cause the generated signature to be invalid, thus circumventing any countermeasure that checks the validity of the signature after signing, as proposed by [8,16].

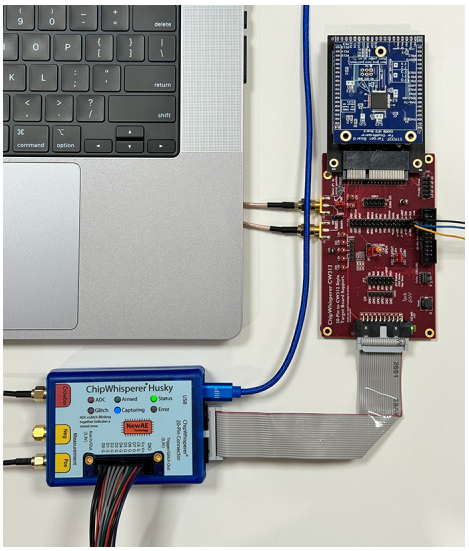


**Fig. 1.** ChipWhisperer-Husky, CW313 adapter board and CW308T-STM32F4 board used in the experiments.

```
/* Compute message representative μ ... */

#ifdef DILITHIUM_USE_HEDGED_MODE
  randombytes(rnd, SEEDBYTES);
#else
  memset(rnd, 0, SEEDBYTES);
#endif
  /* Compute ρ' */
  shake256(rhoprime, CRHBYTES, key, 2*SEEDBYTES + CRHBYTES);

/* Expand matrix A ... */
```

**Listing 1.1.** The modified C code of the signing procedure of the CRYSTALS-Dilithium implementation of [2] with hedged mode added.

## 5 Experimental Setup

This section describes the equipment used for the fault injection, as well as the target software implementation of ML-DSA.

### 5.1 Equipment

For our experiments, we use a ChipWhisperer-Husky, a CW313 adapter board and a CW308T-STM32F4 target device (see Fig. 1).

The target device contains an ARM Cortex-M4-based STM32F415RGT6, which we run at a frequency of 16 MHz.

To avoid having to make any changes to the source code to trigger the fault injection, we use ARM CoreSight ETM/DWT watchpoints. In a real attack scenario, alternative trigger sources such as reference waveforms of the power consumption or communication of the processor with peripheral devices can be used.

### 5.2 Target implementation

In our experiments, we use a modified version of the CRYSTALS-Dilithium implementation by Abdulrahman et al. [2], in which the hedged mode for sampling $\rho'$ is added to the `crypto_sign_signature` procedure, as shown in Listing 1.1. We believe this implementation to be representative for other implementations of the hedged mode, as it follows directly from the specification. Note that our attack targets the implementation of the `shake256` procedure, which we did not modify.

The implementation is compiled using `arm-none-eabi-gcc` with the highest optimization level `-O3` (recommended default).

```
1  size_t keccak_inc_absorb(uint64_t *state, size_t bytes_not_permuted,
2                           uint8_t *m, size_t mlen) {
3    while (mlen + bytes_not_permuted >= 136) {
4      KeccakF1600_StateXORBytes(state, m, bytes_not_permuted);
5      mlen -= 136 - bytes_not_permuted;
6      m += 136 - bytes_not_permuted;
7      bytes_not_permuted = 0;
8      KeccakF1600_StatePermute(state);
9    }
10
11   KeccakF1600_StateXORBytes(state, m, bytes_not_permuted, mlen);
12   return bytes_not_permuted + mlen;
13 }
```

**Listing 1.2.** The C code of the `keccak_inc_absorb` procedure. The function targeted by the fault injection is highlighted in green.

## 6 Fault Injection Attack

This section describes the fault injection attack method and the implementation of the SHAKE256 algorithm.

### 6.1 SHAKE256 algorithm

SHAKE256 is an extendable output function based on the Keccak family of permutations [28]. Keccak employs a so-called sponge construction [17] to realise a function with arbitrary output length. In a sponge construction, input data is first absorbed into a state, after which the state can be squeezed to generate the output of the function.

In the implementation of [2], the SHAKE256 algorithm is implemented using four high-level functions. The `keccak_inc_init` function zero-initialises the Keccak state (an array of 200 bytes). The `keccak_inc_absorb` function (see Listing 1.2) absorbs an arbitrary number of input bytes into the sponge. The `keccak_inc_finalize` function finalises the absorption of data and prepares for the extraction of output by applying a padding. Finally, the `keccak_inc_squeeze` function extracts an arbitrary number of output bytes from the state. A typical use case involves calling all of these functions in the order listed here.

### 6.2 Main idea

The attack targets the absorption of data into the sponge during the hash calculation of $\rho'$. Specifically, a single voltage fault is used to skip the branching to the `KeccakF1600_StateXORBytes` function (see line 11 of Listing 1.2). Note that the loop in lines 3 to 9 is never executed in our case, because the message length of 64 bytes for the message $K||rnd||\mu$ is less than the 136 bytes required

to trigger a permutation. As such, skipping the `KeccakF1600_StateXORBytes` function is sufficient for the sponge to be left empty. This allows an attacker to predict the output $\rho'$ of the hashing procedure.

## 7 Secret key recovery

This section describes the method used to recover the secret key vector $\mathbf{s}_1$ from a successfully faulted signature.

Recovering the secret key vector $\mathbf{s}_1$ from a signature for a known value of $\rho'$ is straightforward. A potential approach is shown in Alg. 4 and works by reconstructing the commitment hash $\tilde{c}'$ using $\rho'$ and a guess for the value $\kappa$. If the commitment hashes match, the challenge $c$ is reconstructed and the secret key vector $\mathbf{s}_1$ can be computed as $\mathbf{s}_1 = (\mathbf{z} - \mathbf{y}) \cdot c^{-1}$, where $c^{-1}$ is the inverse of $c$ in $T_q$.

Note that it is possible for $c$ to have entries with value 0 in the NTT domain. Those entries are not invertible in $T_q$ and the corresponding entries of $\mathbf{s}_1$ in NTT domain thus cannot be determined. Empirically, we found this to rarely be the case (sampling 1M random challenges $c$, we found 21 tuples, all of which contained exactly one entry with value 0). As such, we propose to simply enumerate the possible values of $\mathbf{s}_1$ in NTT domain (i.e. enumerate all possible values of the entry in $\mathbb{Z}_q$) when encountering this case. For the sake of simplicity, this enumeration procedure is omitted from Alg. 4. Note further that the choice of parameters for Dilithium is such that the expected number of iterations in the enumeration of $\kappa$ is low (around 4) [3], thus this approach is generally efficient.

**Algorithm 4** RecoverSecretKey$(pk, \sigma, \rho', \kappa_{\max})$

$(\rho, \mathbf{t}_1) \leftarrow \mathsf{pkDecode}(pk)$
$\hat{\mathbf{A}} \leftarrow \mathsf{ExpandA}(\rho)$
$(\tilde{c}, \mathbf{z}, \mathbf{h}) \leftarrow \mathsf{sigDecode}(\sigma)$
$\kappa \leftarrow 0$
**while** $\kappa < \kappa_{\max}$ **do**
  $\mathbf{y} \leftarrow \mathsf{ExpandMask}(\rho', \kappa)$
  $\mathbf{w} \leftarrow \mathsf{NTT}^{-1}(\hat{\mathbf{A}} \circ \mathsf{NTT}(\mathbf{y}))$
  $\mathbf{w}_1 \leftarrow \mathsf{HighBits}(\mathbf{w})$
  $\tilde{c}' \in \{0,1\}^{2\lambda} \leftarrow \mathrm{H}(\mu || \mathsf{w1Encode}(\mathbf{w}_1), 2\lambda)$
  **if** $[\![\tilde{c} = \tilde{c}']\!]$ **then**
    $(\tilde{c}_1, \tilde{c}_2) \in \{0,1\}^{256} \times \{0,1\}^{2\lambda - 256} \leftarrow \tilde{c}$
    $c \leftarrow \mathsf{SampleInBall}(\tilde{c}_1)$
    $\hat{c} \leftarrow \mathsf{NTT}(c)$
    $\mathbf{s}_1 \leftarrow \mathsf{NTT}^{-1}((\mathsf{NTT}(\mathbf{z}) - \mathsf{NTT}(\mathbf{y})) \circ \hat{c}^{-1})$
    **return** $\mathbf{s}_1$
  $\kappa \leftarrow \kappa + l$
**return** $\perp$

# 8 Experimental Results

This section describes the results of our fault injection attack and subsequent secret key recovery.

## 8.1 Glitch settings

We identified settings that consistently skip the desired function without crashing the device or disrupting other steps of the signature generation by conducting a grid search over the set of parameters offered by the ChipWhisperer-Husky. The results here were achieved by using the `'enable_only'` mode to insert a glitch lasting five clock cycles using both the high-power and low-power crowbar MOSFETs at an offset of 700 units[3] Using these settings, we managed to successfully skip execution of the `KeccakF1600_StateXORBytes` function in 52.8% of 1000 attempts. We believe that it is possible to further increase the success rate of the fault injection through additional optimisation of parameters.

## 8.2 Secret key recovery

We applied Alg. 4 on the signatures generated during the first phase of the attack. As a guess for $\rho'$, we use the output of SHAKE256 generated by applying the finalisation step on an empty state, which is a constant value easily derived by an attacker. Note that Alg. 4 handles cases in which the fault injection was unsuccessful by limiting the number of iterations using the bound $\kappa_{\max}$, thus no additional processing of the signatures is required. We managed to successfully recover the secret key vector $\mathbf{s}_1$ for all 52.8% of cases where the fault injection was successful. We did not encounter the scenario in which $c$ is not invertible for any of these signatures.

# 9 Countermeasures

The presented attack would be infeasible if the individual steps of the Keccak algorithm were inlined into the SHAKE256 routine instead of being separated into multiple subroutines. In fact, because the length of the parameters used in the calculation of $\rho'$ is fixed, it would even be possible to eliminate control flow operations entirely, though such an implementation may not be practical.

Implementations of the SHAKE256 routine should also verify that after absorbing data into the sponge and before squeezing output from the sponge, the state is not empty. If this is not the case, the signing procedure should be aborted, thus offering protection against this particular fault attack. Care should then be taken to ensure that the verification is not itself vulnerable to fault injection

[3] These units are dimensionless and depend on the internal frequency of the ChipWhisperer-Husky, but the offset corresponds to the distance between the rising edge of the clock cycle and the beginning of the glitch.

attacks. For the hedged mode itself, another alternative would be to randomly initialise the state. The presented fault attack would then be insufficient to recover the value of $\rho'$. This countermeasure is not applicable to the deterministic mode, as it introduces non-determinism. Additionally it should be noted that the specification of SHAKE256 makes no claims about the properties of the construction with a randomly initialised state.

More robust countermeasures would require changes to the signing procedure. One such approach may be to move the computation of $\rho'$ into the rejection sampling loop. In that case, an attacker would either be required to predict or affect (through e.g. additional fault injection) the number of times that the rejection sampling is run to inject a single fault during the computation of a signature that is not rejected, or have to inject faults into multiple iterations. This would increase the complexity of an attack. Given the generally high probability of success of the fault injection in this attack and the choice of parameters in Dilithium that inherently keep the number of iterations in the rejection sampling low (see [3]), it is unclear if this approach would be sufficient to prevent an attack.

A different approach is to increase the complexity in recovering the private key after a successful fault injection during the computation of $\rho'$. Here it may be possible to make the value of $\kappa$ used during the sampling of $\mathbf{y}$ unpredictable (e.g. by increasing its size and initialising it randomly), which would require additional randomness or extension of existing random values. Alternatively, it may be possible to include the attacker-unknown value $K$ (or in hedged mode, $rnd$ or a combination of both) in the sampling of $\mathbf{y}$, as proposed by [11]. This eliminates the single point-of-failure around the computation of $\rho'$ at the cost of increasing the input size to the hash function during the sampling of $\mathbf{y}$.

## 10 Conclusion

We presented a practical fault injection attack on a hedged implementation of Dilithium. We identified settings that consistently skip the desired function without crashing the devices or disrupting other steps of the signature generation. The attack can be applied to other parameter sets of ML-DSA (i.e. other variants of Dilithium).

Our work demonstrates that it is possible to recover the secret key vector in a single attempt with high probability, with the generated signature passing verification. This highlights the importance of protecting the calculations of the private random seed $\rho'$, especially when using the hedged mode. Previous work on fault attacks against Dilithium has focused exclusively on the pre-standardisation variant CRYSTALS-Dilithium, while the changes introduced by the ML-DSA variant currently being standardised have not been adequately assessed.

Future work includes developing stronger countermeasures against fault attacks on implementations of PQC algorithms.

## 11 Acknowledgements

This work is part of a master thesis project collaboration between Ericsson Research and KTH.

We would like to thank John Mattsson, Erik Thormarker, Håkan Englund, Jakob Sternby and Niklas Lindskog for taking the time to review and provide comments on earlier versions of this paper, as well as for providing support during the research. We would also like to thank Elena Dubrova, Kalle Ngo and Ruize Wang for their willingness to share their insights and expertise in our discussions, which helped shape our approach and inform our findings.

This work was supported in part by the Swedish Civil Contingencies Agency (Grant No. 2020-11632).

## References

1. Announcing the commercial national security algorithm suite 2.0. National Security Agency, U.S Department of Defense (Sep 2022), https://media.defense.gov/2022/Sep/07/2003071834/-1/-1/0/CSA_CNSA_2.0_ALGORITHMS_.PDF
2. Abdulrahman, A., Hwang, V., Kannwischer, M.J., Sprenkels, A.: Faster Kyber and Dilithium on the Cortex-M4. In: International Conference on Applied Cryptography and Network Security. pp. 853–871. Springer (2022)
3. Bai, S., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schwabe, P., Seiler, G., Stehlé, D.: CRYSTALS-Dilithium. NIST Post-Quantum Cryptography Standardization Round 3 (2021)
4. Barenghi, A., Bertoni, G., Breveglieri, L., Pellicioli, M., Pelosi, G.: Low voltage fault attacks to AES and RSA on general purpose processors. IACR Cryptol. ePrint Arch. p. 130 (2010), http://eprint.iacr.org/2010/130
5. Barenghi, A., Bertoni, G., Parrinello, E., Pelosi, G.: Low voltage fault attacks on the RSA cryptosystem. In: Breveglieri, L., Koren, I., Naccache, D., Oswald, E., Seifert, J. (eds.) Sixth International Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2009, Lausanne, Switzerland, 6 September 2009. pp. 23–31. IEEE Computer Society (2009). https://doi.org/10.1109/FDTC.2009.30, https://doi.org/10.1109/FDTC.2009.30
6. Barenghi, A., Breveglieri, L., Koren, I., Naccache, D.: Fault injection attacks on cryptographic devices: Theory, practice, and countermeasures. Proceedings of the IEEE **100**(11), 3056–3076 (2012)
7. Berzati, A., Viera, A.C., Chartouny, M., Madec, S., Vergnaud, D., Vigilant, D.: Exploiting intermediate value leakage in dilithium: a template-based approach. IACR Transactions on Cryptographic Hardware and Embedded Systems **2023**(4), 188–210 (2023)
8. Bindel, N., Buchmann, J., Krämer, J.: Lattice-based signature schemes and their sensitivity to fault attacks. In: 2016 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC). pp. 63–77. IEEE (2016)
9. Bozzato, C., Focardi, R., Palmarini, F.: Shaping the glitch: Optimizing voltage fault injection attacks. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2019**(2), 199–224 (2019). https://doi.org/10.13154/tches.v2019.i2.199-224, https://doi.org/10.13154/tches.v2019.i2.199-224

10. Breier, J., Hou, X.: How practical are fault injection attacks, really? IEEE Access **10**, 113122–113130 (2022)
11. Bruinderink, L.G., Pessl, P.: Differential fault attacks on deterministic lattice signatures. IACR Transactions on Cryptographic Hardware and Embedded Systems pp. 21–43 (2018)
12. Chen, Z., Karabulut, E., Aysu, A., Ma, Y., Jing, J.: An efficient non-profiled side-channel attack on the crystals-dilithium post-quantum signature. In: 2021 IEEE 39th International Conference on Computer Design (ICCD). pp. 583–590 (2021). https://doi.org/10.1109/ICCD53106.2021.00094
13. Do, Q., Martini, B., Choo, K.R.: The role of the adversary model in applied security research. Comput. Secur. **81**, 156–181 (2019). https://doi.org/10.1016/j.cose.2018.12.002, https://doi.org/10.1016/j.cose.2018.12.002
14. Espitau, T., Fouque, P.A., Gérard, B., Tibouchi, M.: Loop-abort faults on lattice-based fiat-shamir and hash-and-sign signatures. In: Selected Areas in Cryptography–SAC 2016: 23rd International Conference, St. John's, NL, Canada, August 10-12, 2016, Revised Selected Papers 23. pp. 140–158. Springer (2017)
15. Fournaris, A.P., Dimopoulos, C., Koufopavlou, O.: Profiling dilithium digital signature traces for correlation differential side channel attacks. In: International Conference on Embedded Computer Systems. pp. 281–294. Springer (2020)
16. Groot Bruinderink, L., Pessl, P.: Differential fault attacks on deterministic lattice signatures. IACR Transactions on Cryptographic Hardware and Embedded Systems **2018**(3), 21–43 (Aug 2018). https://doi.org/10.13154/tches.v2018.i3.21-43, https://tches.iacr.org/index.php/TCHES/article/view/7267
17. Guido, B., Joan, D., Michaël, P., Gilles, V.: Cryptographic sponge functions (2011)
18. Karabulut, E., Alkim, E., Aysu, A.: Single-trace side-channel attacks on $\omega$-small polynomial sampling: With applications to ntru, ntru prime, and crystals-dilithium. In: 2021 IEEE International Symposium on Hardware Oriented Security and Trust (HOST). pp. 35–45. IEEE (2021)
19. Kiltz, E., Lyubashevsky, V., Schaffner, C.: A concrete treatment of fiat-shamir signatures in the quantum random-oracle model. In: Advances in Cryptology–EUROCRYPT 2018: 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29-May 3, 2018 Proceedings, Part III 37. pp. 552–586. Springer (2018)
20. Kim, I.J., Lee, T.H., Han, J., Sim, B.Y., Han, D.G.: Novel single-trace ml profiling attacks on nist 3 round candidate dilithium. Cryptology ePrint Archive, Paper 2020/1383 (2020), https://eprint.iacr.org/2020/1383, https://eprint.iacr.org/2020/1383
21. Kim, Y., Daly, R., Kim, J., Fallin, C., Lee, J.H., Lee, D., Wilkerson, C., Lai, K., Mutlu, O.: Flipping bits in memory without accessing them: An experimental study of dram disturbance errors. ACM SIGARCH Computer Architecture News **42**(3), 361–372 (2014)
22. Krahmer, E., Pessl, P., Land, G., Güneysu, T.: Correction fault attacks on randomized crystals-dilithium. Cryptology ePrint Archive, Paper 2024/138 (2024), https://eprint.iacr.org/2024/138
23. Langlois, A., Stehlé, D.: Worst-case to average-case reductions for module lattices. Designs, Codes and Cryptography **75**(3), 565–599 (2015)
24. Lyubashevsky, V.: Fiat-Shamir with aborts: Applications to lattice and factoring-based signatures. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 598–616. Springer (2009)

25. Marzougui, S., Ulitzsch, V., Tibouchi, M., Seifert, J.P.: Profiling side-channel attacks on dilithium: A small bit-fiddling leak breaks it all. Cryptology ePrint Archive, Paper 2022/106 (2022), https://eprint.iacr.org/2022/106, https://eprint.iacr.org/2022/106
26. Mattsson, J.P., Thormarker, E., Smeets, B.: Migration to quantum-resistant algorithms in mobile networks, https://www.ericsson.com/en/blog/2023/2/quantum-resistant-algorithms-mobile-networks
27. Migliore, V., Gérard, B., Tibouchi, M., Fouque, P.A.: Masking dilithium: Efficient implementation and side-channel evaluation. In: Applied Cryptography and Network Security: 17th International Conference, ACNS 2019, Bogota, Colombia, June 5–7, 2019, Proceedings 17. pp. 344–362. Springer (2019)
28. National Institute of Standards and Technology: SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions. Tech. Rep. NIST FIPS 202, National Institute of Standards and Technology, Gaithersburg, MD (Aug 2015). https://doi.org/10.6028/NIST.FIPS.202
29. National Institute of Standards and Technology: Module-Lattice-Based Digital Signature Standard. Tech. Rep. NIST FIPS 204 ipd, National Institute of Standards and Technology, Gaithersburg, MD (Aug 2023). https://doi.org/10.6028/NIST.FIPS.204.ipd
30. O'Flynn, C.: Fault injection using crowbars on embedded systems. IACR Cryptol. ePrint Arch. p. 810 (2016), http://eprint.iacr.org/2016/810
31. Qiao, Z., Liu, Y., Zhou, Y., Shao, M., Sun, S.: When ntt meets sis: Efficient side-channel attacks on dilithium and kyber. Cryptology ePrint Archive, Paper 2023/1866 (2023), https://eprint.iacr.org/2023/1866, https://eprint.iacr.org/2023/1866
32. Ravi, P., Jhanwar, M.P., Howe, J., Chattopadhyay, A., Bhasin, S.: Side-channel assisted existential forgery attack on dilithium - a nist pqc candidate. Cryptology ePrint Archive, Paper 2018/821 (2018), https://eprint.iacr.org/2018/821, https://eprint.iacr.org/2018/821
33. Ravi, P., Jhanwar, M.P., Howe, J., Chattopadhyay, A., Bhasin, S.: Exploiting determinism in lattice-based signatures: practical fault attacks on pqm4 implementations of nist candidates. In: Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security. pp. 427–440 (2019)
34. Ravi, P., Roy, D.B., Bhasin, S., Chattopadhyay, A., Mukhopadhyay, D.: Number "Not Used" Once - Practical Fault Attack on pqm4 Implementations of NIST Candidates. In: Polian, I., Stöttinger, M. (eds.) Constructive Side-Channel Analysis and Secure Design, vol. 11421, pp. 232–250. Springer International Publishing, Cham (2019). https://doi.org/10.1007/978-3-030-16350-1_13
35. Ravi, P., Yang, B., Bhasin, S., Zhang, F., Chattopadhyay, A.: Fiddling the twiddle constants - fault injection analysis of the number theoretic transform. IACR Transactions on Cryptographic Hardware and Embedded Systems **2023**(2), 447–481 (Mar 2023). https://doi.org/10.46586/tches.v2023.i2.447-481
36. Singh, R., Islam, S., Sunar, B., Schaumont, P.: Analysis of EM Fault Injection on Bit-sliced Number Theoretic Transform Software in Dilithium. ACM Transactions on Embedded Computing Systems p. 3583757 (Mar 2023). https://doi.org/10.1145/3583757
37. Tang, A., Sethumadhavan, S., Stolfo, S.: {CLKSCREW}: Exposing the perils of {Security-Oblivious} energy management. In: 26th USENIX Security Symposium (USENIX Security 17). pp. 1057–1074 (2017)

38. Ulitzsch, V.Q., Marzougui, S., Bagia, A., Tibouchi, M., Seifert, J.P.: Loop aborts strike back: Defeating fault countermeasures in lattice signatures with ilp. IACR Transactions on Cryptographic Hardware and Embedded Systems **2023**(4), 367–392 (Aug 2023). https://doi.org/10.46586/tches.v2023.i4.367-392, https://tches.iacr.org/index.php/TCHES/article/view/11170
39. Wang, R., Ngo, K., Gärtner, J., Dubrova, E.: Single-trace side-channel attacks on crystals-dilithium: Myth or reality? Cryptology ePrint Archive, Paper 2023/1931 (2023), https://eprint.iacr.org/2023/1931