Single-Trace Side-Channel Attacks on CRYSTALS-Dilithium: Myth or Reality?

Ruize Wang, Kalle Ngo, Joel Gärtner, and Elena Dubrova

KTH Royal Institute of Technology, Stockholm, Sweden
`{ruize,kngo,jgartner,dubrova}@kth.se`

Abstract. We present a side-channel attack on CRYSTALS-Dilithium, a post-quantum secure digital signature scheme, with two variants of post-processing. The side-channel attack exploits information leakage in the secret key unpacking procedure of the signing algorithm to recover the coefficients of the polynomials in the secret key vectors $\mathbf{s}_1$ and $\mathbf{s}_2$ by profiled deep learning-assisted power analysis. In the first variant, one half of the coefficients of $\mathbf{s}_1$ and $\mathbf{s}_2$ is recovered by power analysis and the rest is derived by solving a system of linear equations based on $\mathbf{t} = \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2$, where $\mathbf{A}$ and $\mathbf{t}$ are parts of the public key. This case assumes knowledge of the least significant bits of the vector $\mathbf{t}$, $\mathbf{t}_0$. The second variant waives this requirement. However, to succeed, it needs a larger portion of $\mathbf{s}_1$ to be recovered by power analysis. The remainder of $\mathbf{s}_1$ is obtained by lattice reduction. Once the full $\mathbf{s}_1$ is recovered, all the other information necessary for generating valid signatures can be trivially derived from the public key. We evaluate both variants on an ARM Cortex-M4 implementation of Dilithium-2. The profiling stage (trace capture and neural network training) takes less than 10 hours. In the attack assuming that $\mathbf{t}_0$ is known, the probability of successfully recovering the full vector $\mathbf{s}_1$ from a single trace captured from a different from profiling device is non-negligible (9%). The success rate approaches 100% if multiple traces are available for the attack. Our results demonstrate the necessity of protecting the secret key of CRYSTALS-Dilithium from single-trace attacks and call for a reassessment of the role of compression of the public key vector $\mathbf{t}$ in the security of CRYSTALS-Dilithium implementations.

Keywords: Dilithium · post-quantum digital signature · key recovery attack · side-channel attack · lattice reduction

1 Introduction

CRYSTALS-Dilithium (referred to as Dilithium in the sequel) is a digital signature scheme which is strongly existential unforgeability under chosen message attack (EUF-CMA-secure) in the classical and quantum random oracle models [7]. This means that adversaries having access to a signing oracle cannot produce a signature of a message whose signature they have not yet seen, nor produce a different signature of a message that they already saw signed. The security of Dilithium relies on the hardness of finding short vectors in lattices.

In July 2022, the National Institute of Standards and Technology (NIST) selected Dilithium as a new, post-quantum secure digital signature scheme to be standardized under the name of ML-DSA [47]. The National Security Agency (NSA) has included Dilithium in the commercial national security algorithm (CNSA) suite 2.0 recommended for national security systems [1]. This makes it important to evaluate the resistance of Dilithium to side-channel attacks on its implementations executed on physical devices.

A particularly interesting implementation platform for Dilithium is ARM Cortex-M4, as evidenced by NIST's focus on ARM Cortex-M4 when assessing post-quantum cryptography (PQC) standardization process candidate performance [32]. The interest is motivated by an increased popularity of resource-constrained embedded systems such as internet of things devices. However, since many of these devices are physical accessible by an attacker, side-channel attacks become a concern. To address the threat, the NIST considers the aspect of resistance to side-channel attacks in the PQC standardization process.

Side-channel attacks exploit information leakage through measurable channels, such as power consumption, electromagnetic radiation, timing, heat, etc. Timing attacks, in which an adversary measures the execution time of an algorithm, can be mitigated by making implementations run in constant time, thus eliminating the relation between the time and secret information [37]. Power analysis attacks, in which an adversary measures the power consumption of a device, are more challenging to counter and usually require more expensive mitigation techniques than timing attacks [36]. Among the common countermeasures against power analysis is masking [17], which randomizes the secret data, and shuffling [60], which randomizes the execution order of the secret data.

It has been demonstrated in the past that the theoretical EUF-CMA-security of Dilithium can be bypassed by a side-channel attack on its implementation [9, 14,19,34,44,54], as some of these attacks enable digital signature forgery. Further work in this direction continues to be an important research topic considering that the adaptation of Dilithium is quickly moving from research to standards, implementation, and deployment. The 3GPP is planning to introduce quantum-resistant cryptographic algorithms in 5G as soon as the final NIST and IETF security protocol standards have been published [45]. The above-mentioned NSA CNSA suite 2.0 requires that all network equipment supports Dilithium by 2026 and all other equipment - by 2030 [1]. Therefore, it is important to continue searching for potential leakage points and vulnerabilities in Dilithium implementations as well as investigating new ways to exploit them. Each discovered implementation weakness gives the developers of Dilithium implementations an opportunity to strengthen the subsequently released versions, thereby contributing to the deployment of more secure products in the future.

Our contributions: We present a new side-channel attack on Dilithium which exploits information leakage in the secret key unpacking procedure of the signing algorithm. This particular leakage point has not been explored in previous side-channel attacks on Dilithium. In these attacks, information leakage of the secret key vector $\mathbf{s}_1$ is usually modeled via inner products of $\mathbf{s}_1$ with some known vector. The presented attack can directly recover a large part of polynomial coefficients of $\mathbf{s}_1$ and $\mathbf{s}_2$ in the range $[-\eta, \eta]$ using deep learning-assisted profiled power analysis. To recover the rest of the coefficients, we propose two variants of post-processing.

In the first one, at least one half of the coefficients of both $\mathbf{s}_1$ and $\mathbf{s}_2$ are recovered by power analysis and the rest are derived by solving a system of linear equations based on $\mathbf{t} = \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2$, where $\mathbf{A}$ is a public polynomial matrix and $\mathbf{t}$ is the second component of (an expanded form of) the public key. This case assumes knowledge of the low order bits of $\mathbf{t}$, $\mathbf{t}_0$. According to the FIPS 204 draft [27]: "The vector $\mathbf{t}$ is compressed in the actual public key by dropping the d least significant bits from each coefficient, ... This compression is an optimization for performance, not security. The low order bits of $\mathbf{t}$ can be reconstructed from a small number of signatures and, therefore, need not be regarded as secret."

In the second variant, the knowledge of $\mathbf{t}_0$ is not necessary. We recover a larger portion of $\mathbf{s}_1$ by power analysis and derive the remainder of $\mathbf{s}_1$ by lattice reduction. Once the complete $\mathbf{s}_1$ is recovered, all the other information necessary for generating valid signatures can be trivially derived from the public key [41]. We evaluate the side-channel attack with both post-processing variants on an implementation of Dilithium in ARM Cortex-M4 presented in [2] and suggest countermeasures.

In our opinion, the main contribution of this paper is highlighting the possibility of recovering the complete secret vector $\mathbf{s}_1$ from a *single* trace with a non-negligible probability (9% in our experiments) in the case when $\mathbf{t}_0$ is known. None of the previous attacks on Dilithium can recover the full $\mathbf{s}_1$ from fewer than 100 traces. Our results demonstrate the necessity of protecting the secret key of Dilithium from single-trace attacks. They also prompt a reassessment of the role of $\mathbf{t}_0$ in the security of Dilithium implementations.

Paper organization: The rest of this paper is organized as follows. Section 2 provides the necessary background for understanding the paper. Section 3 describes previous work on side-channel analysis of Dilithium implementations. Section 4 describes assumptions on the adversary model. Section 5 presents the two variants of post processing. Section 6 explains the power analysis strategy. Section 7 summarizes experimental results. Section 8 suggests potential countermeasures. Section 9 concludes the paper.

2 Background

This section describes notation used in the paper, the specification of Dilithium [7], profiled side-channel attacks, and the lattice basis reduction.

2.1 Notation

Let $\mathbb{Z}_q$ be the ring of integers modulo a prime q and R_q be the quotient ring $\mathbb{Z}_q[X]/(X^n + 1)$. We use regular font letters for elements in R_q, bold lower-

Table 1. Parameters of different versions of Dilithium.

Version	n	q	(k, ℓ)	η	d	γ_1	γ_2	β	ω
Dilithium-2	256	8380417	(4, 4)	2	13	2^{17}	$(q-1)/88$	78	80
Dilithium-3	256	8380417	(6, 5)	4	13	2^{19}	$(q-1)/32$	196	55
Dilithium-5	256	8380417	(8, 7)	2	13	2^{19}	$(q-1)/32$	120	75

case letters for vectors with coefficients in R_q, and bold upper-case letters for matrices. To simplify the notation, when $\mathbf{v} \in R_q^k$ we use $\mathbf{v}[i]$ to denote ith entry of a vector consisting of the coefficients of the k polynomials in $\mathbf{v}$. As such, it corresponds to coefficient $i \bmod n$ of entry $\lfloor i/n \rfloor$ in $\mathbf{v}$.

The term $x \leftarrow S$ stands for sampling x from a set S uniformly. The concatenation for two bit/byte strings a and b is denoted by $a \mid\mid b$. The blank symbol $\perp$ is used to indicate failure or lack of an output from the algorithm. The infinity norm is denoted by $\|\cdot\|_\infty$. The operation inside the double square brackets $[\![\cdot]\!]$ is evaluated as a Boolean. Norms for elements in R_q are given as if the coefficients were entries in $(-q/2, q/2]$ of an n dimensional vector, and this is naturally extended to vectors over R_q. As in the Dilithium specification, we let S_η be the set of elements $\mathbf{w} \in R_q$ with $\|\mathbf{w}\|_\infty \leq \eta$.

2.2 Dilithium specification

The security of Dilithium [7] is based on the assumed hardness of the module learning with error (M-LWE) problem over R_q and a version of the module short integer solution (M-SIS) problem. It adopts the Fiat-Shamir with aborts technique [40] to transform a lattice-based identification scheme to a signature scheme. The signature rejection is applied to avoid the leakage of any secret information from the signature.

There are three main algorithms in Dilithium: key pair generation, KeyGen, message signing, Sign, and signature verification, Verify, see Fig. 1. The inputs and outputs of all these algorithms are byte arrays. Special functions, Encode and Decode, perform the conversion between the byte arrays and the polynomial coefficients.

Dilithium utilizes four different sampling functions. $\mathsf{ExpandA}(\rho)$ is used for generating the public matrix $\mathbf{A}$ from a seed $\rho \in \{0,1\}^{256}$. The secret key vectors $\mathbf{s}_1$ and $\mathbf{s}_2$ are generated by $\mathsf{ExpandS}(\rho')$, where $\rho' \in \{0,1\}^{512}$. $\mathsf{ExpandMask}(\rho', \kappa)$, $\kappa \geq 0$, is employed for generating the randomness of the signature scheme. The sparse polynomial is obtained from $\mathsf{SampleInBall}(\tilde{c})$, where $\tilde{c} \in \{0,1\}^{256}$.

Dilithium applies compression algorithms to optimize the key size. The d low-order bits of each coefficient of the polynomials in the vector $\mathbf{t} = \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2$ are dropped from the public key using the $\mathsf{Power2Round}(\mathbf{t}, d)$ function. To reconstruct information necessary to verify the signature using the compressed public key, hints $\mathbf{h}$ are created by MakeHint during signing and then used by UseHint during verification. During signing, the coefficients of the polynomials

```
KeyGen()
 1: $\zeta \leftarrow \{0,1\}^{256}$
 2: $(\rho, \rho', K) \in \{0,1\}^{256} \times \{0,1\}^{512} \times \{0,1\}^{256} = \mathsf{H}(\zeta)$
 3: $\mathbf{A} \in R_q^{k \times \ell} = \mathsf{ExpandA}(\rho)$
 4: $(\mathbf{s}_1, \mathbf{s}_2) \in S_\eta^\ell \times S_\eta^k = \mathsf{ExpandS}(\rho')$
 5: $\mathbf{t} = \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2$
 6: $(\mathbf{t}_1, \mathbf{t}_0) = \mathsf{Power2Round}(\mathbf{t}, d)$
 7: $tr \in \{0,1\}^{256} = \mathsf{H}(\rho \mid\mid \mathbf{t}_1)$
 8: $pk = \mathsf{pkEncode}(\rho, \mathbf{t}_1)$
 9: $sk = \mathsf{skEncode}(\rho, K, tr, \mathbf{s}_1, \mathbf{s}_2, \mathbf{t}_0)$
10: return $(pk, sk)$

Sign$(sk, M)$
 1: $(\rho, K, tr, \mathbf{s}_1, \mathbf{s}_2, \mathbf{t}_0) = \mathsf{skDecode}(sk)$
 2: $\mathbf{A} \in R_q^{k \times \ell} = \mathsf{ExpandA}(\rho)$
 3: $\mu \in \{0,1\}^{512} = \mathsf{H}(tr \mid\mid M)$
 4: $\kappa = 0, (\mathbf{z}, \mathbf{h}) = \perp$
 5: $\rho' \leftarrow \{0,1\}^{512}$
 6: while $(\mathbf{z}, \mathbf{h}) = \perp$ do
 7:     $\mathbf{y} = \mathsf{ExpandMask}(\rho', \kappa)$
 8:     $\mathbf{w} = \mathbf{A}\mathbf{y}$
 9:     $\mathbf{w}_1 = \mathsf{HighBits}(\mathbf{w}, 2\gamma_2)$
10:     $\tilde{c} \in \{0,1\}^{256} = \mathsf{H}(\mu \mid\mid \mathbf{w}_1)$
11:     $c = \mathsf{SampleInBall}(\tilde{c})$
12:     $\mathbf{z} = \mathbf{y} + c\mathbf{s}_1$
13:     $\mathbf{r}_0 = \mathsf{LowBits}(\mathbf{w} - c\mathbf{s}_2, 2\gamma_2)$
14:     if $||\mathbf{z}||_\infty \geq \gamma_1 - \beta$ or $||\mathbf{r}_0||_\infty \geq \gamma_2 - \beta$ then $(\mathbf{z}, \mathbf{h}) = \perp$
15:     else
16:         $\mathbf{h} = \mathsf{MakeHint}(-c\mathbf{t}_0, \mathbf{w} - c\mathbf{s}_2 + c\mathbf{t}_0, 2\gamma_2)$
17:         if $||c\mathbf{t}_0||_\infty \geq \gamma_2$ or # of 1's in $\mathbf{h}$ is $> \omega$ then $(\mathbf{z}, \mathbf{h}) = \perp$
18:     $\kappa = \kappa + \ell$
19: return $\sigma = (\tilde{c}, \mathbf{z}, \mathbf{h})$

Verify$(pk, M, \sigma = (\tilde{c}, \mathbf{z}, \mathbf{h}))$
 1: $(\rho, \mathbf{t}_1) = \mathsf{pkDecode}(pk)$
 2: $\mathbf{A} \in R_q^{k \times \ell} = \mathsf{ExpandA}(\rho)$
 3: $\mu \in \{0,1\}^{512} = \mathsf{H}(\mathsf{H}(\rho \mid\mid \mathbf{t}_1) \mid\mid M)$
 4: $c = \mathsf{SampleInBall}(\tilde{c})$
 5: $\mathbf{w}_1' = \mathsf{UseHint}(\mathbf{h}, \mathbf{A}\mathbf{z} - c\mathbf{t}_1 \cdot 2^d, 2\gamma_2)$
 6: return $[\![ ||\mathbf{z}||_\infty < \gamma_1 - \beta ]\!]$ and $[\![ \tilde{c} = \mathsf{H}(\mu \mid\mid \mathbf{w}_1') ]\!]$ and $[\![$# of 1's in $\mathbf{h}$ is $\leq \omega ]\!]$
```

Fig. 1. The pseudocode of Dilithium main algorithms [7].

in the vector $\mathbf{w} = \mathbf{A}\mathbf{y}$, where $\mathbf{y} = \mathsf{ExpandMask}(\rho', \kappa)$, are decomposed as $r = r_1 \cdot 2\gamma_2 + r_0$, where r_1 is the output of HighBits function and r_0 is the output of LowBits function.

Dilithium uses the number-theoretic transform (NTT) to perform multiplications in R_q efficiently. The NTT details are omitted from Fig. 1 to simplify the pseudocode.

There are three versions of Dilithium representing different security levels: Dilithium-2, Dilithium-3 and Dilithium-5, Table 1 gives a summary of their parameters. We refer the reader to the Dilithium specification [7] for further details. In this paper, we focus on Dilithium-2.

2.3 Profiled side-channel attacks

Side-channel attacks can be classified into two types: profiled and non-profiled.

Profiled attacks first model information leakage of the target implementation using one or more devices similar to the device under attack, called *profiling devices*. This can be done either by creating templates [4,16,30], or by training a neural network model [13,15,35,42]. The resulting templates/model is used to recover the secret variable, e.g. the secret key, from the implementation of the cryptographic algorithm executed on the device under attack [42].

Non-profiled attacks are applied directly, without a beforehand modeling of the leakage of the target implementation [59]. Non-profiled attacks may involve statistical methods such as correlation power analysis [11], or unsupervised machine learning algorithms such as k-means [58].

2.4 Lattice basis reduction

Lattice-based cryptography relies on the assumed hardness of problems which can be interpreted as lattice problems. For instance, the security of Dilithium relies on the hardness of the module-LWE problem, which can be interpreted as version of the closest vector problem (CVP) in a structured q-ary lattice. Furthermore, by the Kannan embedding, this CVP instance can be solved by finding an unusually short vector in a related lattice [23], which is a version of the shortest vector problem (SVP).

A *lattice* in $\mathbb{Z}^n$ can be defined via the set of all linear combinations of m linearly independent basis vectors $\mathbf{B} = \{\mathbf{b}_1, \dots, \mathbf{b}_m\}$, $\mathbf{b}_i \in \mathbb{Z}^n$, for all $i \in \{1, \dots, n\}$ and $m \leq n$. The integers m and n are called the *rank* and the *dimension* of the lattice, respectively. Lattice basis reduction algorithms aim to find a new basis for the same lattice but with shorter, more orthogonal basis vectors.

One of the most well-known lattice basis reduction algorithms is the Lenstra–Lenstra–Lovász (LLL) algorithm [38]. It has a polynomial time complexity in the lattice dimension n but is only able to find lattice vectors that are exponentially longer than the shortest possible lattice vectors. In order to find shorter vectors in lattices, variants of the block Korkin-Zolotarev (BKZ) lattice reduction algorithm [56] are typically used. These algorithms are parameterized by a blocksize β, which impacts both the runtime and the length of vectors found by the algorithm. These algorithms work by finding very short vectors in projected sublattices of the original lattice, with β being the maximal dimension of these sublattices. Larger blocksizes allow for shorter vectors to be found, but this comes at the cost of time complexity that is exponential in β.

A rough estimate for the cost of solving a problem with BKZ is given by the Core-SVP hardness. For blocksize β, this is an estimate of the cost to find very short vectors in a single β-dimensional lattice. A common estimate, used for example in the Dilithium submission [7], is that this can be solved in time essentially $2^{0.292\beta}$, based on the asymptotic performance of the algorithm in [8].

To estimate the size of short vectors which BKZ with blocksize β can actually find, a good first estimate is given by the formula from [18], giving that, based on some heuristic assumptions, BKZ will asymptotically find vectors of length $\delta_\beta^n \det(\mathcal{L})^{1/n}$ in an n dimensional lattice $\mathcal{L}$ where

$$\delta_\beta = \left(\frac{\beta}{2\pi e} \cdot (\pi\beta)^{\frac{1}{\beta}} \right)^{\frac{1}{2(\beta-1)}} .$$

For solving concrete instances of variants of the LWE problem, a better estimate can be given by an LWE-estimator, such as the ones developed in the works [3, 24].

3 Previous Work

Several side-channel attacks on Dilithium implementations have been presented in the past. The attacks primarily focus on the recovery of the secret key vector $\mathbf{s}_1$, as all the other information required for generating valid signatures can be derived from the public key [41].

In the pioneering work of Ravi et al. [54], a simulated leakage of the polynomial multiplication operation $c \cdot \mathbf{s}_1$ in the signing procedure of Dilithium is used to recover $\mathbf{s}_1$ by power analysis. The attack considers an early Dilithium implementation using the schoolbook polynomial multiplication and optimised polynomial multiplication algorithms, rather than point-wise multiplication in the NTT domain used in the later implementations of Dilithium.

In the follow up work, Fournaris et al. [28] demonstrates a correlation power analysis (CPA)-based attack on an ARM Cortex-M4 implementation of Dilithium using the leakage of the polynomial multiplication operation in the signing procedure. The attack is applicable to various multiplication algorithms, including in the NTT domain.

Chen et al. [19] presents several improved CPA-based attacks on Dilithium which also exploit the leakage of the polynomial multiplication. For Dilithium-2, one of the attacks of [19] can recover a coefficient of $\mathbf{s}_1$ with a 99.99% probability using 157 power traces and a key coefficient guessing table of size 2^{22}. This attack requires 110 min to recover a single coefficient. Thus, it would take 2.5 months to recover all 1024 coefficients of $\mathbf{s}_1$ with a success probability of $0.9999^{1024} = 0.9027$. The most efficient attack presented in [19] can recover a coefficient of $\mathbf{s}_1$ with a 96.1% probability from 10,000 power traces in 846 min. Therefore, it would take 10 days to recover the full $\mathbf{s}_1$ with a nearly-zero success probability of $0.961^{1024} = 2.04 \times 10^{-18}$.

In the work of Miglore et al. [46], side-channel resistance of an unprotected ARM Cortex M3 implementation of Dilithium is evaluated with focus on three functions: LowBits, HighBits, and rejection operation. It is concluded that these functions are highly leaky. In addition, a masking countermeasure is proposed and verified. In the masked implementation, each of the secret key vectors $\mathbf{s}_1$ and $\mathbf{s}_2$ is split into two shares during the key generation. The three above-mentioned functions no longer leak in the masked implementation.

Kim et al. [34] presents a machine learning-assisted profiled attack exploiting a leakage of load, save and Montgomery reduction operations in the signing procedure to recover $\mathbf{s}_1$. The attack is demonstrated in a simulated setting on an unprotected and masked versions of Dilithium-2.

Karabulut et al. [33] proposes an attack on small polynomial sampling, with the idea being that, when the challenge c is generated, the valid values are in the small range of $\{-1, 0, 1\}$. The attack aims to determine the sign of the non-zero coefficients through side-channel analysis.

Marzougai et al. [44] demonstrates a machine learning-assisted profiled side-channel attack targeting the nonce $\mathbf{y}$. The attack exploits a leakage in the function ExpandMask which is used for sampling $\mathbf{y}$ pseudo-randomly from a seed and a counter. Given a power trace captured during an execution of the signing algorithm, a machine learning classifier is used to decide whether a given coefficient $\mathbf{y}[i]$ of $\mathbf{y}$ is zero or not. If $\mathbf{y}[i] = 0$, then $\mathbf{y}[i] = 0 + (c\mathbf{s}_1)[i]$. Since the response vector $\mathbf{z}$ and verifier's challenge c are known once the signature is computed, the full $\mathbf{s}_1$ can be recovered from multiple signatures (about 0.6M) by using linear algebra.

Berzati et al. [9] shows that, by exploiting a leakage in the `Decompose` function which partitions a given vector into high and low order bits, the low bits of the commitment vector $\mathbf{w}$, $\mathbf{w}_0$, can be recovered by a template attack. Templates are used to decide whether a given coefficient in one coordinate of $\mathbf{w}_0$ is zero or not. Since the commitment $\mathbf{w}$ is related to the nonce $\mathbf{y}$ through a public matrix, $\mathbf{w} = \mathbf{A}\mathbf{y}$, and the high order bits of $\mathbf{w}$, $\mathbf{w}_1$, can be derived from the signature $(\mathbf{z}, c)$, $\mathbf{s}_1$ can be recovered from multiple signatures (about 0.7M) in the same way as in the attack of Marzougai et al. [44]. Other attacks based on conceptually similar ideas are presented in [39, 52]

In the interesting recent work by Qiao et al. [53], a CPA-based attack on an unprotected and a masked implementations of Dilithium in ARM Cortex-M4 combined with a small integer solution (SIS)-based post-processing is presented. The attack targets leakage of point-wise multiplication in the NTT domain, more specifically the output from the Montgomery reduction operation. For unprotected Dilithium-2 implementation, the combined attack can recover a subset of 256 secret key coefficients from 2000 power traces in 0.5 min. For the first-order masked implementation, in which the secret key is divided into two shares during the key generation, the shares are first recovered independently. Then, the coefficients recovered at the same position are added, and, finally, a post-processing step based on SIS is applied to derive the remaining 256 coefficients.

While searching for other side-channel attacks on Dilithium that exploit information leakage in the secret key unpacking procedure of the signing algorithm, we came across an attack on NTRU key encapsulation mechanism (KEM) by Askeland et al. [5] that uses the secret key unpacking procedure of the decapsulation algorithm as an attack point. Their method extracts about 400 secret key coefficients (about 78% of the total 509) by side-channel analysis and then applies BKZ lattice reduction to derive the rest of the key. The attack presented in this paper significantly differs from [5] in side-channel analysis technique. Note that the Dilithium case in more difficult than NTRU because Dilithium's secret key is twice the size of NTRU's secret key.

In summary, none of the previous attacks on Dilithium exploit leakage in the secret key unpacking procedure of the signing algorithm to directly recover polynomials coefficients of $\mathbf{s}_1$ in the range $[-\eta, \eta]$. Instead, they recover inner products of $\mathbf{s}_1$ with some known vector. Furthermore, none of the previous attacks on Dilithium can recover the full secret key vector $\mathbf{s}_1$ from fewer than 100 traces. The attacks such as [9, 44] require over half a million traces.

4 Adversary model

This section defines the three main components of an adversary model [25]: adversary goals, assumptions and capabilities.
Assumptions: We assume that an adversary has a physical access to the device under attack which runs the Dilithium digital signature algorithm. We also assume that the adversary possesses fully controllable profiling devices that are similar to the device under attack.
Capabilities: The adversary is a clever outsider who has equipment and tools for power analysis, as well as expertise in side-channel attacks, Dilithium, and deep learning. The adversary is capable to capture power traces from the device under attack during the execution of the signing algorithm.
Goals: The goal of the adversary is to perform a digital signature forgery, i.e. to create a pair consisting of a message, M and a signature σ, that is valid for M, but has not been created in the past by the legitimate signer. To achieve this goal, the adversary first attempts to extract partial information about the secret key vector $\mathbf{s}_1$ from the implementation of Dilithium running on the device under attack through side-channels. Then, the attacker applies post-processing methods to deduce the rest of $\mathbf{s}_1$. Recovery of the full $\mathbf{s}_1$ is sufficient to generate valid signatures [41].

5 Two Variants of Post-Processing

As we mentioned in the introduction, the Dilithium public key does not contain the full public key vector $\mathbf{t}$. Instead, it only includes a compressed representation of $\mathbf{t}$, $\mathbf{t}_1$, that does not depend on d least significant bits of $\mathbf{t}$, $\mathbf{t}_0$. However, this compression is an optimization for performance, not security and, the FIPS

2024 draft states that the full $\mathbf{t}$ can be recovered from a small number of signatures [27]. Therefore, as the first post-processing variant, we assume knowledge of not only $\mathbf{t}_1$, but also $\mathbf{t}_0$, i.e. knowledge of the full $\mathbf{t}$ vector.

Note, however, that, in some applications of a signature scheme, gathering a sufficient number of signatures for the complete reconstruction of the $\mathbf{t}$ vector may not be possible. Thus, we also consider an alternative post-processing variant that does not require knowledge of the full $\mathbf{t}$ vector.

5.1 With knowledge of $\mathbf{t}_0$

If $\mathbf{t}_0$ is known, we can use knowledge of some of the coefficients of vectors $\mathbf{s}_1$ and $\mathbf{s}_2$ to recover the remaining coefficients by solving a system of linear equations based on $\mathbf{t} = \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2$.

This equation can be rewritten with the polynomials in $\mathbf{t}$, $\mathbf{s}_1$ and $\mathbf{s}_2$ represented by column vectors of coefficients $\overline{\mathbf{t}}$, $\overline{\mathbf{s}}_1$ and $\overline{\mathbf{s}}_2$ with each coefficient represented by an element in $\mathbb{Z}_q$. Furthermore, the matrix of polynomials $\mathbf{A}$ can be represented by a $nk \times n\ell$ dimensional matrix $\overline{\mathbf{A}}$ over $\mathbb{Z}_q$ such that $\overline{\mathbf{t}} = \overline{\mathbf{A}}\overline{\mathbf{s}}_1 + \overline{\mathbf{s}}_2$.

Each known coefficient of the polynomials in $\mathbf{s}_1$ allows for removing the dependence of this coefficient in the vector $\overline{\mathbf{s}}_1$ and constructing an equivalent equation of type

$$\overline{\mathbf{t}}^* = \overline{\mathbf{A}}^* \overline{\mathbf{s}}_1^* + \overline{\mathbf{s}}_2 \tag{1}$$

where all elements of $\overline{\mathbf{t}}^*$ and $\overline{\mathbf{A}}^*$ are known and the dimension of $\overline{\mathbf{s}}_1^*$ is $n\ell - x$, where x is the number of known coefficients of $\mathbf{s}_1$.

Meanwhile, each known coefficient of the polynomials in $\mathbf{s}_2$ gives us one known element of the vector $\overline{\mathbf{s}}_2$. With y known elements of $\overline{\mathbf{s}}_2$ in $\overline{\mathbf{s}}_2'$, a subset of the rows of (1) corresponds to the equation

$$\overline{\mathbf{b}} = \overline{\mathbf{t}}' - \overline{\mathbf{s}}_2' = \overline{\mathbf{A}}' \overline{\mathbf{s}}_1^* \tag{2}$$

with a known $\overline{\mathbf{b}}$ and a known matrix $\overline{\mathbf{A}}'$ of dimension $y \times (n\ell - x)$.

If $y \geq n\ell - x$, i.e. at least $n\ell$ coefficients in total are recovered by side-channel analysis, we can, with a high probability, recover the remaining coefficients of $\mathbf{s}_1$. One way to accomplish this is to find an invertible $(n\ell - x) \times (n\ell - x)$ dimensional submatrix of $\overline{\mathbf{A}}'$, from which we can easily recover the remaining coefficients of $\mathbf{s}_1$ by multiplying the relevant portion of $\overline{\mathbf{b}}$ by the inverse of this submatrix. For Dilithium-2, it is sufficient to recover a subset of 1024 coefficients as $n = 256$ and $\ell = 4$. As k also equals 4 for these parameters, there is a total of 2048 coefficients in $\mathbf{s}_1$ and $\mathbf{s}_2$ combined, and it is thus sufficient to recover half of these coefficients for this attack.

5.2 Without knowledge of $\mathbf{t}_0$

In the case where $\mathbf{t}_0$ is not known, but we are able to recover a large percentage of the coefficients of $\mathbf{s}_1$ by side-channel analysis, the remainder of $\mathbf{s}_1$ can be recovered by lattice reduction using the public information $\mathbf{A}$ and $\mathbf{t}_1$ and the fact that $\mathbf{t}_1 \cdot 2^d = \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2 - \mathbf{t_0}$.

The matrix of polynomials $\mathbf{A}$ can be represented by a $nk \times n\ell$ dimensional matrix $\overline{\mathbf{A}}$ over $\mathbb{Z}_q$, while $2^d \cdot \mathbf{t}_1$ and $\mathbf{s}_1$ can be represented as nk and $n\ell$ dimensional vectors $\overline{\mathbf{b}}$ and $\overline{\mathbf{s}}$ respectively. Furthermore, we let $\mathbf{s}_2 - \mathbf{t_0}$ be represented by another nk dimensional vector $\overline{\mathbf{e}}$. Then the equation $\overline{\mathbf{b}} = \overline{\mathbf{A}}\overline{\mathbf{s}}_1 + \overline{\mathbf{e}}$ can be solved by considering the lattice spanned by the columns of of the matrix

$$\begin{bmatrix} q\mathbf{I}_{nk} & \overline{\mathbf{A}} & \overline{\mathbf{b}} \\ \mathbf{0} & \mathbf{I}_{n\ell} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & 1 \end{bmatrix}$$

which contains the unusually short secret vector $(-\overline{\mathbf{e}}, \overline{\mathbf{s}}, 1)$.

In addition, each known element of $\overline{\mathbf{s}}$ allows for decreasing the rank of this lattice by one, making the problem of finding the secret vector easier. Finding the unusually short secret vector allows deriving the remaining elements of $\overline{\mathbf{s}}$ and the recovery of the full $\mathbf{s}_1$ vector. With sufficiently many coefficients of $\mathbf{s}_1$ recovered by side-channel analysis, obtaining the full secret vector $\mathbf{s}_1$ using variants of BKZ is practically feasible as our experimental results show.

For the attack, we actually consider a slight modification of this lattice where we make use of the fact that the elements in the $\overline{\mathbf{e}}$ vector, corresponding to $\mathbf{s}_2 - \mathbf{t}_0$, are significantly larger than the elements in the secret vector $\overline{\mathbf{s}}$. To this end, parts of the lattice are scaled to ensure that it contain an unusually short vector where all coordinates of this vector have essentially the same size, and which is directly related to the secret vector we want to find. This significantly decrease the cost of finding the unknown secret vector. The actual cost of finding the unknown secret vector is estimated through the use of the LWE-estimator developed as a part of [3].

6 Side-Channel Attack on Dilithium

This section describes the target implementation on Dilithium and outlines the process of selecting points of interest, collecting and pre-processing traces, training neural networks at the profiling stage, and using them for inference at the attack stage.

6.1 Target implementation

In the experiments, we use the ARM Cortex-M4 implementation of Dilithium by Abdulrahman et al. [2]. This implementation does not contain any countermeasures against side-channel attacks.

We use the highest optimization level `-O3` (recommended default) to compile the C code of Dilithium-2 implementation to a binary using `arm-none-eabi-gcc` command. Higher optimization levels are typically more difficult for side-channel analysis [57].

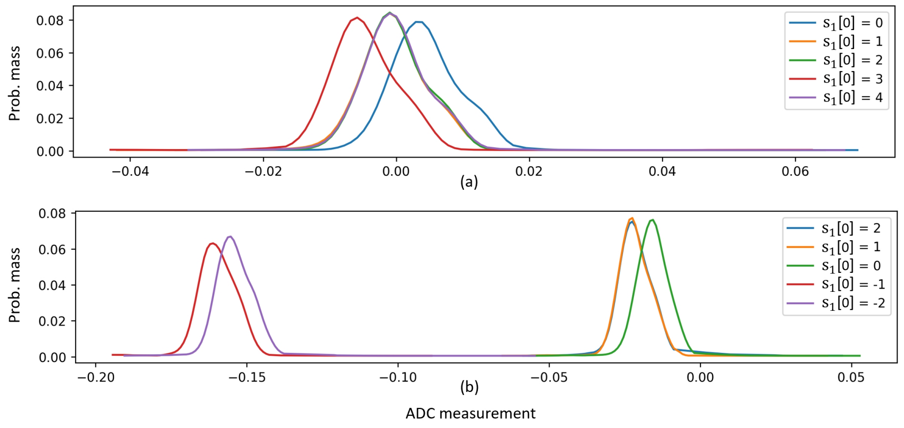

Fig. 2. Distributions of power consumption during the processing of the first coefficient of $\mathbf{s}_1$ by `small_polyeta_unpack()` procedure of Dilithium-2: (a) in the range $[0, 2\eta]$ (lines 2-9 in Fig. 3), (b) in the range $[-\eta, \eta]$ (lines 10-17 in Fig. 3).

6.2 Attack point

The presented side-channel attack on Dilithium targets the secret key unpacking function skDecode. It is carried out at the first step of the signing algorithm, see line 1 of Sign in Fig. 1. Fig. 3 shows the C code of the procedure `unpack_sk()` which realizes the function skDecode in the Dilithium implementation of [2]. In the first **for**-loop of `unpack_sk()` (lines 2-4), `small_polyeta_unpack()` is called ℓ times to unpack 96 bytes of each of the ℓ polynomials in $\mathbf{s}_1 \in S_\eta^\ell$ into $n = 256$ coefficients in the range $[-\eta, \eta]$. In the next **for**-loop (lines 6-8), the byte array representing the k polynomials in vector $\mathbf{s}_2 \in S_\eta^k$ is unpacked similarly.

Since power consumption in a software implementation is typically proportional to the Hamming weight of processed data [36], our ability to distinguish among numbers in a given range is related to their Hamming weight. In `small_polyeta_unpack()` procedure of the Dilithium implementation of [2], the polynomial coefficients of $\mathbf{s}_1$ and $\mathbf{s}_2$ are processed twice: first, in the range $[0, 2\eta]$ (lines 2-9 in Fig. 3) and second, in the range $[-\eta, \eta]$ (lines 10-17 in Fig. 3). The coefficients are defined as 16-bits integers and negative numbers are represented in two's complement, e.g. -1 and -2 are represented by `0xFFFF` and `0xFFFE`, respectively. Hence, for Dilithium-2, the Hamming weights of the five elements of $C_1 = (4, 3, 2, 1, 0)$ during the first processing are $HW(C_1) = (1, 2, 1, 1, 0)$. Likewise, the Hamming weights of the five elements of $C_2 = (-2, -1, 0, 1, 2)$ during the second processing are $HW(C_2) = (15, 16, 0, 1, 1)$. One can see that the pairs of Hamming weights corresponding to the same coefficient are unique: $(HW(C_1), HW(C_2)) = ((1, 15), (2, 16), (1, 0), (1, 1), (0, 1))$. Therefore, they are potentially distinguishable by power analysis.

```
void unpack_sk(uint8_t rho, uint8_t tr, uint8_t key, polyvec *t0,
smallpoly s1, smallpoly s2, uint8_t sk)
unsigned int i;
 1:  ... unpacking rho, tr and key ...
 2: for (i = 0; i < L; ++i)  do
 3:     small_polyeta_unpack(&s1[i], sk + i*POLYETA_PACKEDBYTES);
 4: end for
 5: sk += L*POLYETA_PACKEDBYTES;
 6: for (i = 0; i < K; ++i)  do
 7:     small_polyeta_unpack(&s2[i], sk + i*POLYETA_PACKEDBYTES);
 8: end for
 9: sk += K*POLYETA_PACKEDBYTES;
10:  ... unpacking t0 ...

void small_polyeta_unpack(smallpoly *r, uint8_t *a)
/* a is the input byte array of s1 or s2 in the secret key*/
/* r is the corresponding output polynomial coefficients of of s1 or s2*/
unsigned int i;
 1: for (i = 0; i < N/8; ++i)  do /* N = 256, ETA = 2 in Dilithium-2 */
 2:    r->coeffs[8*i+0] = (a[3*i+0] >> 0) & 7;
 3:    r->coeffs[8*i+1] = (a[3*i+0] >> 3) & 7;
 4:    r->coeffs[8*i+2] = ((a[3*i+0] >> 6) | (a[3*i+1] << 2)) & 7;
 5:    r->coeffs[8*i+3] = (a[3*i+1] >> 1) & 7;
 6:    r->coeffs[8*i+4] = (a[3*i+1] >> 4) & 7;
 7:    r->coeffs[8*i+5] = ((a[3*i+1] >> 7) | (a[3*i+2] << 1)) & 7;
 8:    r->coeffs[8*i+6] = (a[3*i+2] >> 2) & 7;
 9:    r->coeffs[8*i+7] = (a[3*i+3] >> 5) & 7;

10:    r->coeffs[8*i+0] = ETA - r->coeffs[8*i+0];
11:    r->coeffs[8*i+1] = ETA - r->coeffs[8*i+1];
12:    r->coeffs[8*i+2] = ETA - r->coeffs[8*i+2];
13:    r->coeffs[8*i+3] = ETA - r->coeffs[8*i+3];
14:    r->coeffs[8*i+4] = ETA - r->coeffs[8*i+4];
15:    r->coeffs[8*i+5] = ETA - r->coeffs[8*i+5];
16:    r->coeffs[8*i+6] = ETA - r->coeffs[8*i+6];
17:    r->coeffs[8*i+7] = ETA - r->coeffs[8*i+7];
18: end for
```

Fig. 3. The C code of secret key unpacking procedure `unpack_sk()` which realizes the function skDecode in the Dilithium implementation of [2].

Fig. 2 illustrates distributions of power consumption during the processing of the first polynomial coefficient of $\mathbf{s}_1$, $\mathbf{s}_1[0]$, by `small_polyeta_unpack()` procedure for the case of Dilithium-2. Plots in Fig. 2(a) and (b) are made based on 10K traces at the trace point with the maximum absolute Welch's t-test score within the intervals covering the execution of lines 2-9 and lines 10-17 in Fig. 3, respectively. The overlap between the plots representing different numbers de-

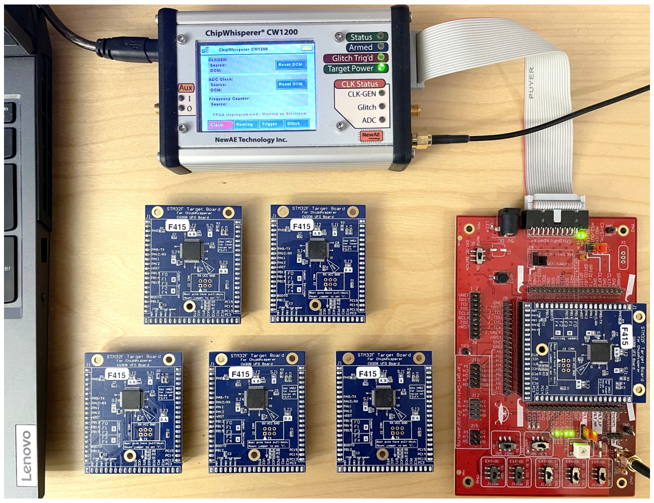

Fig. 4. The equipment used for trace acquisition.

termines the difficulty of distinguishing these numbers. We can see that numbers with different Hamming weights are clearly separable. Contrary, those with the same Hamming weight overlap almost completely.

6.3 Equipment

Our equipment for trace acquisition is shown in Fig. 4. It consists of a Chip-Whisperer-Pro, a CW308 UFO main board and six CW308T-STM32F4 target boards. Five of the CW308T-STM32F4 are used for profiling, and one for the attack. Multiple profiling devices are used for minimizing the negative effect of inter-device variability on the neural network's classification accuracy [62].

The ChipWhisperer-Pro is a hardware security evaluation toolkit based on an open hardware platform and open-source software [48]. It can be used to measure power consumption and control communication between the target device and the computer. ChipWhisperer-Pro measures the voltage over a shunt resistor which is placed in series with the target device. The voltage is proportional to the current draw of the device. Hence, for a constant supply voltage, the measured voltage is proportional to the total power consumption of the device.

The CW308 UFO board is a general-purpose platform for evaluating multiple targets [21]. The target board is plugged into a dedicated U connector.

Each CW308T-STM32F4 target board contains a STM32F415-RGT6 chip based on ARM Cortex-M4 32-bit RISC core operating at a frequency of 24Mhz [22]. The traces are sampled at 96MS/s, i.e. four data points per clock cycle. The choice of sampling rate is limited by the size of ChipWhisperer-Pro buffer which

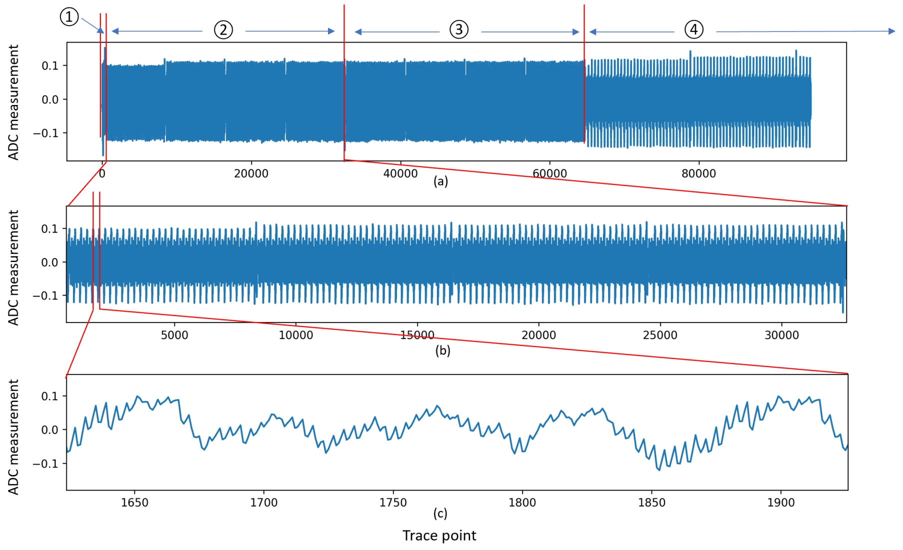

Fig. 5. (a) An average power trace representing `unpack_sk()` procedure of Dilithium-2 in which: ① is unpack ρ, tr and K, ② is unpack $\mathbf{s}_1$, ③ is unpack $\mathbf{s}_2$, and ④ is unpack $\mathbf{t_0}$. (b) A segment corresponding to the unpacking of $\mathbf{s}_1$ by `small_polyeta_unpack()`. (c) A 440-point segment representing eight coefficients of $\mathbf{s}_1$ (input to neural networks).

is 98K samples[1]. If a higher sampling rate is used, all points of interest do not fit into the buffer.

6.4 Trace acquisition

Using the equipment described above, we capture from each profiling device traces for training neural networks. We also capture traces for testing from the device under attack. Both profiling and attack traces are captured for different messages selected at random.

Fig. 5(a) shows a full trace recorded by the ChipWhisperer-Pro during the execution of the signing algorithm of Dilithium-2. It is computed as an average over 1000 measurements to reduce the noise. The trace covers a nearly complete execution of `unpack_sk()` except for the last part of t_0 unpacking. Fig. 5(b) gives a zoomed in view of the segment representing the unpacking of $\mathbf{s}_1$. The segment representing the unpacking of $\mathbf{s}_2$ looks similarly. In both cases, there are four blocks of 32 identical patterns corresponding to the four calls of `small_polyeta_unpack()` in the **for**-loop. Fig. 5(c) shows a further zoomed in

[1] ChipWhisperer-Pro has an option of streaming, however, streaming can be used only at a maximum of 10 MHz sampling frequency.

view of the segment representing the processing of eight consecutive coefficients of $\mathbf{s}_1$. Such segments are used as input for neural networks.

The traces which are captured using ChipWhisperer platform do not require an additional synchronisation because the ChipWhisperer-Pro provides the clock signal for the target board and hence the sampling frequency is perfectly synchronized with the clock frequency of the target board. Furthermore, the ChipWhisperer platform is essentially noise free. Hence, the conditions in our experiments represent a best case scenario for the attacker.

6.5 Profiling stage

Let $\mathbf{T} \in \mathbb{R}^{r \times u}$ be a set containing the union of traces captured from the profiling devices, where r is the total number of traces and $u = pn(\ell + k)/8$ is the total number of data points in each trace, with p being the number of data points covering the processing of eight consecutive polynomial coefficients of $\mathbf{s}_1$ or $\mathbf{s}_2$ in `unpack_sk()` (they are processed in the same way).

To create a training set, we first extract from $\mathbf{T}$ two subsets, $\mathbf{T}_1^* \in \mathbb{R}^{r \times pn\ell/8}$ and $\mathbf{T}_2^* \in \mathbb{R}^{r \times pnk/8}$, representing the unpacking of $\mathbf{s}_1$ and $\mathbf{s}_2$ by the procedure `unpack_sk()`, respectively (see Fig. 5(b)). We then partition $\mathbf{T}_1^*$ into $n\ell/8$ segments $\mathbf{T}_1^*[j] \in \mathbb{R}^{r \times p}$, $j \in \{0, 1, \dots, n\ell/8 - 1\}$, representing the processing of eight consecutive polynomial coefficients of $\mathbf{s}_1$ (see Fig. 5(c)). Similarly, we partition $\mathbf{T}_2^*$, into $nk/8$ segments $\mathbf{T}_2^*[j] \in \mathbb{R}^{r \times p}$, $j \in \{0, 1, \dots, nk/8 - 1\}$, representing the processing of eight consecutive polynomial coefficients of $\mathbf{s}_2$. The training set is composed as a union of all these segments:

$$\mathbf{T}_{tr} = (\bigcup_{j=0}^{n\ell/8-1} \mathbf{T}_1^*[j]) \bigcup (\bigcup_{j=0}^{nk/8-1} \mathbf{T}_2^*[j]) \in \mathbb{R}^{((\ell+k)nr/8) \times p}.$$

Finally, we standardize $\mathbf{T}_{tr}$ by transforming each trace $T = (t_1, \dots, t_p) \in \mathbf{T}_{tr}$, to $T' = (t'_1, \dots, t'_p)$ such that $t'_i = \frac{t_i - \mu_i}{\sigma_i}$, for all $i \in \{1, \dots, p\}$, where and μ_i and σ_i are the mean and the standard deviation of the traces of $\mathbf{T}_{tr}$ at the ith data point.

We use the same multilayer perceptron (MLP) architecture as in the side-channel attack on CRYSTALS-Kyber of [26] except for the input size and the number of output classes. In the attack of [26], neural networks are binary classifiers. In our case, the neural networks classify into five classes corresponding to the five values of polynomial coefficients of $\mathbf{s}_1$ and $\mathbf{s}_2$. We experimented with many variations of the architecture and types of neural networks, but none yielded better results.

From the pseudocode of the procedure `small_polyeta_unpack()` in Fig. 3, it is clear that each group of eight polynomial coefficients of the input byte array is unpacked differently. For this reason, we train eight neural network models $\mathcal{N}_b$, one per each $b \in \{0, 1, \dots, 7\}$. One can also see from the pseudocode of the procedure `unpack_sk()` in Fig. 3 that vectors $\mathbf{s}_1$ and $\mathbf{s}_2$ are unpacked in exactly the same way. Therefore, we train the same models for both $\mathbf{s}_1$ and $\mathbf{s}_2$. To train

$\mathcal{N}_b$, each trace $\mathbf{T}_i^*[j] \in \mathbf{T}_{tr}$ is labeled by the value of the polynomial coefficient $\mathbf{s}_i[8j+b]$, for $b \in \{0, 1, \ldots, 7\}$ and $j \in \{0, 1, \ldots, m/8-1\}$, where $m = n\ell$ for $i = 1$ and $m = nk$ for $i = 2$.

The neural networks are trained with a batch size of 1024 for a maximum of 100 epochs using early stopping with patience 10. We use Nadam optimizer with a learning rate of 0.001 and a numerical stability constant $\epsilon =$ 1e-08. Categorical cross-entropy is used as a loss function to evaluate the network classification error. 70% of the training set is used for training and 30% is left for validation. Only the model with the highest validation accuracy is saved.

6.6 Attack stage

At the attack stage, the neural networks $\mathcal{N}_{j \bmod 8}$ trained at the profiling stage are used to recover the coefficients $\mathbf{s}_i[j]$, for $j \in \{0, 1, \ldots, m-1\}$, where $m = n\ell$ for $i = 1$ and $m = nk$ for $i = 2$.

The neural network $\mathcal{N}_{j \bmod 8}$ inputs a trace segment $T \in \mathbb{R}^p$ related to $\mathbf{s}_i[j]$ and outputs a score vector $S_{i,j} = \mathcal{N}_{j \bmod 8}(T)$ whose elements quantify the likelihood that $\mathbf{s}_i[j] = a$ in T, for an integer $a \in [-\eta, \eta]$. For an attack using multiple traces, the most likely value of $\mathbf{s}_i[j]$ is determined by computing a cumulative probability of N score vectors, where N is the number of traces used in the attack.

At the final step, we decide which recovered polynomial coefficients to accept as correct and which ones to leave for the post-processing recovery. The decision depends on the post-processing variant used.

In the first variant of post-processing, which assumes the knowledge of $\mathbf{t}_0$, we sort the predicted polynomial coefficients of both $\mathbf{s}_1$ and $\mathbf{s}_2$ vectors (which are $n(\ell + k)$ in total) in the descending order according to the maximum probability of their score vectors. In other words, we determine which of the coefficients are predicted by neural networks which the highest confidence. Then, the top half of the elements of the ordered list are accepted as correct. The other half if left for the the post-processing to recover.

In the second variant of post-processing, which does not require the knowledge of $\mathbf{t}_0$, we sort the predicted polynomial coefficients of $\mathbf{s}_1$ (which are $n \cdot \ell$ in total) in a similar way as above. We accept as correct $\lfloor x \cdot n \cdot \ell \rfloor$ top elements of the ordered list, where x is a fraction in the range $[0, 1]$. The rest is left for the the post-processing to recover. We quantify the attack success probability as a function of the fraction x in the experimental results section.

7 Experimental Results

In this section, we evaluate the presented side-channel attack with both post-processing variants on an implementation of Dilithium-2 in ARM Cortex-M4 by Abdulrahman et al. [2].

Table 2. Time for capturing the training and test sets.

Test set	Time for capturing N traces		
	$N = 1$	$N = 100$	$N = 1000$
	1.2 sec	36.6 sec	358.2 sec

Training set	Time for capturing 5×2.5K traces
	4.8 hrs

Table 3. Empirical probability to recover a single coefficient of $\mathbf{s}_1$, $\mathbf{s}_1[j]$, by power analysis using N traces; each entry in the middle column is a mean probability over all $\mathbf{s}_1[j]$ with the same j mod 8, for $j \in \{0, 1, \cdots, 1023\}$.

N	j mod 8								Avg.
	0	1	2	3	4	5	6	7	
1	0.942	0.925	0.975	0.967	0.945	0.920	0.924	0.784	0.923
10	0.980	0.951	0.999	0.993	0.988	0.951	0.956	0.863	0.960
100	0.983	0.952	0.999	0.995	0.991	0.954	0.959	0.870	0.963
1000	0.983	0.954	0.999	0.995	0.991	0.956	0.960	0.871	0.964

7.1 Recovering the coefficients of $\mathbf{s_1}$ and $\mathbf{s_2}$ by power analysis

At the profiling stage, we capture from each of the five profiling devices 2.5K power traces for neural network training. Each trace is captured during the execution of the signing algorithm for a secret key and a message selected at random. After applying the trace expansion strategy described in Section 6.5, the total number of training traces becomes 2.5K $\times$ 5 $\times$ 128 $\times$ 2 = 3.2M, with $p = 440$ data points in each trace. Using the resulting training set, we train eight neural network models $\mathcal{N}_b$, $b \in \{0, 1, \ldots, 7\}$. The training time for a single model is less than 40 min on a PC with an Intel Core i7-10750H CPU with a 16GB RAM running at 2.6GHz. When combined with the 4.8 hours for trace capture listed in Table 2, the total profiling time is less than 10 hours.

At the attack stage, we select at random 100 different secret keys and, for each key, capture traces during the execution of the signing algorithm by the device under attack for 1000 messages selected at random[2]. Then, the model $\mathcal{N}_{j \bmod 8}$ trained at the profiling stage is used to recover the polynomial coefficient $\mathbf{s}_i[j]$, for all $i \in \{1, 2\}$ and $j \in \{0, 1, \ldots, 1023\}$. For an attack using N traces, the value of $\mathbf{s}_i[j]$ is determined by computing a cumulative probability of N score vectors $S_{i,j}$ inferred by the model $\mathcal{N}_{j \bmod 8}$.

Tables 3 and 4 summarise the attack results for $\mathbf{s}_1$ and $\mathbf{s}_2$, respectively. They list empirical probabilities to recover a single coefficient of $\mathbf{s}_i$ by power analysis

[2] It is does not matter whether the messages are selected random, or kept fixed, because skDecode takes only the secret key as input (see line 1 of Sign in Fig. 1).

Table 4. Empirical probability to recover a single coefficient of $\mathbf{s}_2$, $\mathbf{s}_2[j]$, by power analysis using N traces; each entry in the middle column is a mean probability over all $\mathbf{s}_2[j]$ with the same j mod 8, for $j \in \{0, 1, \cdots, 1023\}$.

N	j mod 8								Avg.
	0	1	2	3	4	5	6	7	
1	0.943	0.925	0.973	0.970	0.950	0.924	0.919	0.781	0.923
10	0.978	0.948	0.999	0.996	0.988	0.957	0.946	0.857	0.959
100	0.981	0.950	0.999	0.997	0.991	0.961	0.947	0.861	0.961
1000	0.981	0.950	0.999	0.997	0.992	0.962	0.948	0.865	0.962

using a different number of traces $N \in \{1, 10, 100, 1000\}$. Since a table showing all 1024 coefficients of $\mathbf{s}_i$ would be too large, we grouped the probabilities into eight groups according to the neural network $\mathcal{N}_{j \bmod 8}$ which is used the recover the coefficient. Each entry in the middle column of Tables 3 and 4 is a mean probability over all $\mathbf{s}_i[j]$ with the same index j mod 8, for $j \in \{0, 1, \cdots, 1023\}$ and $i \in \{0, 1\}$.

From the Tables 3 and 4, one can see that the coefficients $\mathbf{s}_i[j]$ with $j \bmod 8 = 7$ have a considerably lower recovery probability than the rest of the coefficients. To explain this, we examined the assembly of `small_polyeta_unpack()` procedure. We found that, for the coefficients $\mathbf{s}_i[j]$ with $j \bmod 8 = 7$, the compiler does not allocate `store` and `mask` instructions in the same way as for the other coefficients. For this reason, the values associated with $\mathbf{s}_i[j]$ with $j \bmod 8 = 7$ are manipulated less, hence a weaker leakage.

From the last columns of both tables, one can see that the probability to recover a single coefficient of $\mathbf{s}_1$ or $\mathbf{s}_2$ first increases with the growth of N and then flattens after $N = 100$. By increasing the number of traces further, we do not gain much. For $N = 1000$, the probability to recover a single coefficient of $\mathbf{s}_1$ is 0.964, thus the probability to recover the full $\mathbf{s}_1$ is $0.964^{1024} \approx 0$. Similarly, for $N = 1000$, the probability to recover the full $\mathbf{s}_2$ is $0.962^{1024} \approx 0$. This shows that pure power analysis cannot recover all polynomial coefficients of $\mathbf{s}_1$ or $\mathbf{s}_2$ reliably.

7.2 Recovering complete $\mathbf{s}_1$ and $\mathbf{s}_2$ using linear equations

Next, we evaluate the first post-processing variant, based on solving a system of the linear equations induced by $\mathbf{t} = \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2$.

We sort 2048 polynomial coefficients $\mathbf{s}_i[j]$, $i \in \{0, 1\}$, $j \in \{0, 1, \cdots, 1023\}$, recovered by power analysis in the descending order according to the maximum probability of the corresponding score vectors $S_{i,j}$ inferred by the models $\mathcal{N}_{j \bmod 8}$. We accept as correct the top half of the resulting sorted list. Since the elements of score vectors represent the likelihood that $\mathbf{s}_i[j] = a$, for $a \in \{-2, -1, 0, 1, 2\}$, the higher is the maximum probability in the score vector $S_{i,j}$,

Table 5. Empirical probability to recover 1024 coefficients of $\mathbf{s}_1$ and $\mathbf{s}_2$ by power analysis using N traces. The last column shows an average CPU time required to recover the rest of coefficients by using linear algebra (LA).

Probability to recover 1024 coefficients of $\mathbf{s}_1$ and $\mathbf{s}_2$				LA post-processing
$N = 1$	$N = 10$	$N = 100$	$N = 1000$	CPU time
0.09	0.83	0.99	1	2 sec

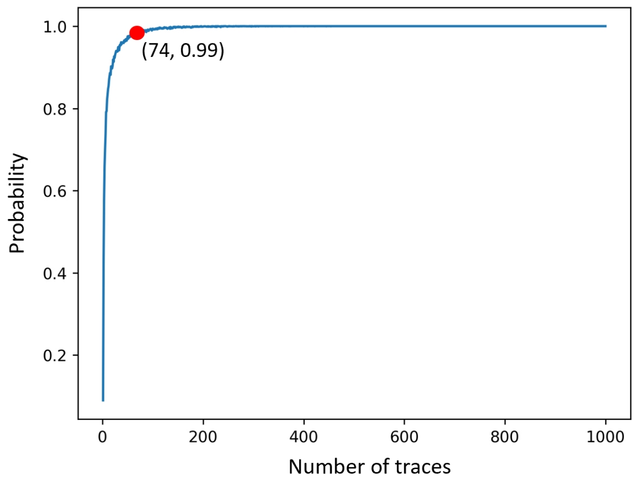

Fig. 6. Empirical probability to recover 1024 coefficients of $\mathbf{s}_1$ and $\mathbf{s}_2$ by power analysis as a function of the number of traces; the saturation point is marked in red.

the more confident[3] is the prediction $\mathbf{s}_i[j] = a$. The rest of the coefficients of $\mathbf{s}_1$ is derived by solving a system of linear equations as described in Section 5.1.

Table 5 shows empirical probabilities to recover 1024 coefficients of $\mathbf{s}_1$ and $\mathbf{s}_2$ by power analysis using a different number of traces N. The last column gives an average CPU time required to recover the rest of coefficients by solving linear equations. We can see that this variant of post-processing takes only a few seconds. We can also see that the likelihood to recover 1024 coefficients of $\mathbf{s}_1$ and $\mathbf{s}_2$ by power analysis increases sharply with the growth of N. The success rate of a single-trace attack is 9%. In the attacks using 1000 traces, we always recover all 1024 coefficients correctly. Clearly, this may be due to an insufficiently large test set.

Fig. 6 further illustrates the relation between of the attack success probability and the number of traces N. We can see that the curve grows sharply and then

[3] Clearly, even if a neural network predicts $\mathbf{s}_i[j] = k$ with 100% probability, it does not mean that this is correct, since the network is not a perfect classifier.

Table 6. Empirical probability to recover $\lfloor x \cdot 1024 \rfloor$ coefficients of $\mathbf{s}_1$ by power analysis using N traces; α is an estimate of the Core-SVP bit hardness required to recover the remaining coefficients of $\mathbf{s}_1$ by BKZ with blocksize β.

Fraction x	Probability to recover $\lfloor x \cdot 1024 \rfloor$ coeff. of $\mathbf{s}_1$				BKZ post-processing	
	$N = 1$	$N = 10$	$N = 100$	$N = 1000$	β	α
3/4	0	0.16	0.70	0.82	160	46.7
4/5	0	0.04	0.42	0.54	115	33.6
5/6	0	0.01	0.20	0.26	86	25.2

flattens. The saturation point is reached at $N = 74$. The corresponding success probability is 0.99.

7.3 Recovering complete $\mathbf{s}_1$ using lattice reduction

Finally, we evaluate the second post-processing variant using lattice reduction.

In the same way as in the first variant, we the sort 1024 polynomial coefficients of $\mathbf{s}_1$ recovered by power analysis in descending order according to the maximum probability of the corresponding score vectors $S_{i,j}$ inferred by the models $\mathcal{N}_{j \bmod 8}$. We accept as correct the top $\lfloor x \cdot 1024 \rfloor$ coefficients of the resulting sorted list, where x is a given fraction. The rest of the coefficients of $\mathbf{s}_1$ is derived by BKZ lattice reduction.

Table 6 shows empirical probabilities to recover $\lfloor x \cdot 1024 \rfloor$ coefficients of $\mathbf{s}_1$ by power analysis using N traces for different fractions x. The BKZ blocksize β required to find the remaining coefficients of $\mathbf{s}_1$ by lattice reduction is estimated by using the lattice estimator developed as a part of [3]. The last column gives an estimate of the corresponding Core-SVP bit hardness, $\alpha \approx 0.292\beta$. For the case of $x = 3/4$, recovering the remaining coefficients requires a similar amount of work as the records for the SVP challenge [55]. On the other hand, for the $x = 4/5$ case, the remaining coefficients should be recoverable with relatively powerful hardware and a bit of time. Finally, for the $x = 5/6$ case, one can run the attack on a consumer-grade laptop.

We can also see that, with this variant of post-processing, single-trace attacks are not successful. For the same number of traces N, the attack success probability is lower than then one in Table 5.

Fig. 7 plots attack success probabilities as functions of N for different fractions $x = 3/4, 4/5$ and $5/6$. We can see that, compared to the curve in Fig. 6, all three curves grow less sharply. The saturation points are (283, 0.79), (326, 0.51) and (221, 0.25) for $x = 3/4, 4/5$ and $5/6$, respectively. We can also see that the curves flatten, implying that the success probability cannot be increased considerably by using more traces for the attack.

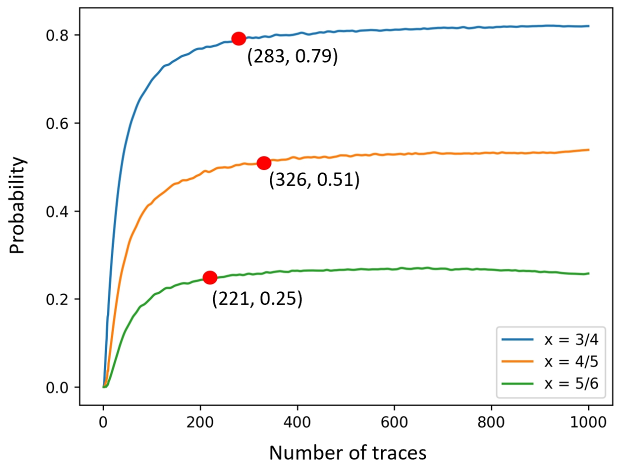

Fig. 7. Empirical probability to recover $\lfloor x \cdot 1024 \rfloor$ coefficients of $\mathbf{s}_1$ by power analysis for different fractions x; the saturation points are marked in red.

7.4 Extension to Dilithium-3 and 5

The presented attack is extendable to other versions of Dilithium, although it will be considerably more difficult. The first reason for this, relevant only for Dilithium-3, is a larger range $[-\eta, \eta] = [-4, 4]$ of polynomial coefficients of the secret key. This makes the classification problem more challenging as neural networks have to distinguish among nine classes instead of five. The fact that Hamming weights of numbers in the ranges $[0, 8]$ and $[-4, 4]$ do not form unique pairs further complicates the case.

The second reason for increased difficulty of the attack is a larger absolute number of coefficients of $\mathbf{s}_1 \in S_\eta^\ell$ and $\mathbf{s}_2 \in S_\eta^k$ to be recovered through side-channel analysis due to the larger parameters ℓ and k, see Table 1. This raises requirements on the probability to recover a single coefficient. In the presented attack, we achieve good results by recovering the jth coefficient using a single neural network model $\mathcal{N}_{j \bmod 8}$ trained once. More sophisticated methods such as ensemble learning [50, 61] and iterative re-training [49] may be needed to further increase the single coefficient recovery probability. For the attacks in noisier conditions, convolutional neural networks [42, 51], or transformers [12, 29] may be more suitable neural network architectures than MLP. Noise reduction, e.g. by using autoencoders [42, 63], may also be helpful.

8 Countermeasures

Our presented side-channel attack would be substantially more difficult if the signing algorithm would take the secret key as input in a masked form as, e.g., in the masked implementation of Dilithium by Coren et al. [20]. If the coefficients of random masks are in the range $(-q/2, q/2]$, then the coefficients of the masked shares representing $\mathbf{s}_1$ and $\mathbf{s}_2$ are also in the range $(-q/2, q/2]$. Thus, the classification problem would be considerably more difficult[4]. In addition, the attacker would need to recover the coefficients of all shares of $\mathbf{s}_1$ or $\mathbf{s}_2$ *in the same position* to recover a given coefficient of $\mathbf{s}_1$ or $\mathbf{s}_2$. It may be worth mentioning that the template attack on CRYSTALS-Kyber KEM presented in [10] can distinguish values in the range $(-q/2, q/2]$ with the help of q^2 templates. However, since CRYSTALS-Kyber uses $q = 3329$ while Dilithium uses $q = 8380417$, the Dilithium's case is considerably more difficult.

A k-order masking increases the size of the secret key representation by a factor of $k+1$ which may not be desirable. Another possible countermeasure against the presented attack is to shuffle the secret key at the end of the key generation algorithm. In this case, the signing algorithm would take as input a shuffled version of the key together with the shuffling permutation. To overcome the shuffling countermeasure, the attacker could try either to recover the shuffling permutation by a side-channel attack on the signing algorithm, or to stop shuffling from being executed by the key generation algorithm using a fault attack. The former is known to be possible for KEMs, see the attacks on the decapsulation algorithms of Saber and CRYSTALS-Kyber presented in [6]. However, these attacks require many traces captured for chosen ciphertexts. They can recover at most two shuffling indexes from a single trace. The latter is also known to be possible for KEMs, see the attack on the decapsulation algorithm of CRYSTALS-Kyber presented in [31]. A similar fault attack could in principle by mounted on the key generation algorithm of Dilithium; verifying its feasibility remains a future work.

Finally, the presented side-channel attack would be more difficult if the secret key would be encoded in a constant-weight code, e.g., using the method of Maghrebi et al. [43]. In theory, encoding the coefficients of $\mathbf{s}_1$ or $\mathbf{s}_2$ to the same weight should result in uniform power consumption during their unpacking, eliminating any potential leakage. However, since coding-based countermeasures may leave exploitable correlations, assessing their efficacy in protecting Dilithium from single-trace attacks is a topic for future research.

9 Conclusion

We have presented a practical side-channel attack on an implementation of Dilithium-2 that exploits leakage in the secret key unpacking procedure of the signing algorithm. The attack is not specific to Dilithium-2, it is extendable to other versions as well. It may also be applicable to other PQC algorithms which use a similar implementation of the secret key unpacking procedure.

[4] Note that some masked implementations of Dilithium, e.g. [46], do not protect the secret key at the input of the signing algorithm. Such implementations are potentially vulnerable to the presented attack.

Our experimental results show that, if the low order bits of the public key vector $\mathbf{t}$ are known, it is possible to recover the full secret key vector $\mathbf{s}_1$ key from a single trace with a non-negligible probability. No previous side-channel attack on Dilithium has successfully recovered the full $\mathbf{s}_1$ with fewer than 100 traces. Apart from highlighting the importance of protecting the secret key of Dilithium from single-trace attacks, our results call for a re-evaluation of the role of compression of the public key vector $\mathbf{t}$ in the security of Dilithium implementations.

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

10 Acknowledgments

This work was supported in part by the Swedish Civil Contingencies Agency (Grant No. 2020-11632) and the KTH Center for Cyber Defense and Information Security (CDIS).

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. Albrecht, M., Player, R., Scott, S.: On the concrete hardness of learning with errors. Journal of Mathematical Cryptology **9** (10 2015). `https://doi.org/10.1515/jmc-2015-0016`
4. Archambeau, C., Peeters, E., Standaert, F.X., Quisquater, J.J.: Template attacks in principal subspaces. In: Cryptographic Hardware and Embedded Systems. pp. 1–14 (2006)
5. Askeland, A., Rønjom, S.: A side-channel assisted attack on NTRU. Cryptology ePrint Archive, Paper 2021/790 (2021), `https://eprint.iacr.org/2021/790`
6. Backlund, L., Ngo, K., Gärtner, J., Dubrova, E.: Secret key recovery attack on masked and shuffled implementations of crystals-kyber and saber. In: International Conference on Applied Cryptography and Network Security. pp. 159–177. Springer (2023)
7. Bai, S., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schwabe, P., Seiler, G., Stehlé, D.: CRYSTALS-Dilithium algorithm specifications and supporting documentation (2021), `https://pq-crystals.org/dilithium/data/dilithium-specification-round3-20210208.pdf`
8. Becker, A., Ducas, L., Gama, N., Laarhoven, T.: New directions in nearest neighbor searching with applications to lattice sieving. In: Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms. p. 10–24. SODA '16, Society for Industrial and Applied Mathematics, USA (2016)
9. 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)

10. Bock, E.A., Banegas, G., Brzuska, C., Łukasz Chmielewski, Puniamurthy, K., Šorf, M.: Breaking DPA-protected Kyber via the pair-pointwise multiplication. Cryptology ePrint Archive, Paper 2023/551 (2023), `https://eprint.iacr.org/2023/551`
11. Brier, E., Clavier, C., Olivier, F.: Correlation power analysis with a leakage model. In: Joye, M., Quisquater, J.J. (eds.) Cryptographic Hardware and Embedded Systems. pp. 16–29. Springer (2004)
12. Brisfors, M.: Advanced Side-Channel Analysis of USIMs, Bluetooth SoCs and MCUs. Master's thesis, School of EECS, KTH (2021)
13. Brisfors, M., Forsmark, S., Dubrova, E.: How deep learning helps compromising USIM. In: Proc. of the 19th Smart Card Research and Advanced Application Conference (CARDIS'2020) (Nov 2020)
14. Bruinderink, L.G., Pessl, P.: Differential fault attacks on deterministic lattice signatures. IACR Transactions on Cryptographic Hardware and Embedded Systems pp. 21–43 (2018)
15. Cagli, E., Dumas, C., Prouff, E.: Convolutional neural networks with data augmentation against jitter-based countermeasures. In: Cryptographic Hardware and Embedded Systems – CHES 2017. pp. 45–68 (2017)
16. Camurati, G., Poeplau, S., Muench, M., Hayes, T., Francillon, A.: Screaming channels: When electromagnetic side channels meet radio transceivers. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. pp. 163–177 (2018)
17. Chari, S., Jutla, C.S., Rao, J.R., Rohatgi, P.: Towards sound approaches to counteract power-analysis attacks. In: Advances in Cryptology - CRYPTO '99. vol. 1666, pp. 398–412. Springer (1999)
18. Chen, Y.: Réduction de réseau et sécurité concrète du chiffrement complètement homomorphe (2013), `https://api.semanticscholar.org/CorpusID:170791320`
19. 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`
20. Coron, J.S., Gérard, F., Trannoy, M., Zeitoun, R.: Improved gadgets for the high-order masking of dilithium. Cryptology ePrint Archive, Paper 2023/896 (2023), `https://eprint.iacr.org/2023/896`
21. CW308 UFO Board: `https://rtfm.newae.com/Targets/CW308%20UFO/`
22. CW308T-STM32F4 target board: `https://rtfm.newae.com/Targets/UFO%20Targets/CW308T-STM32F/`
23. Dachman-Soled, D., Ducas, L., Gong, H., Rossi, M.: LWE with side information: Attacks and concrete security estimation. In: Advances in Cryptology – CRYPTO 2020. vol. 12171, pp. 329–358. Springer (2020)
24. Dachman-Soled, D., Ducas, L., Gong, H., Rossi, M.: Lwe with side information: Attacks and concrete security estimation. In: Micciancio, D., Ristenpart, T. (eds.) Advances in Cryptology – CRYPTO 2020. pp. 329–358. Springer International Publishing, Cham (2020)
25. Do, Q., Martini, B., Choo, K.K.R.: The role of the adversary model in applied security research. Computers & Security **81**, 156–181 (2019)
26. Dubrova, E., Ngo, K., Gärtner, J., Wang, R.: Breaking a fifth-order masked implementation of CRYSTALS-Kyber by copy-paste. In: Proceedings of the 10th ACM Asia Public-Key Cryptography Workshop. pp. 10–20 (2023)
27. FIPS, P.: Fips 204 (draft): Module-lattice-based digital signature standard. National Institute of Standards and Technology `https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.204.ipd.pdf`

28. Fournaris, A.P., Dimopoulos, C., Koufopavlou, O.: Profiling Dilithium digital signature traces for correlation differential side channel attacks. In: Orailoglu, A., Jung, M., Reichenbach, M. (eds.) Embedded Computer Systems: Architectures, Modeling, and Simulation. pp. 281–294. Springer International Publishing, Cham (2020)
29. Hajra, S., Chowdhury, S., Mukhopadhyay, D.: Estranet: An efficient shift-invariant transformer network for side-channel analysis. IACR Transactions on Cryptographic Hardware and Embedded Systems **2024**(1), 336–374 (2024)
30. Hoffman, C., Gebotys, C., Aranha, D.F., Cortes, M., Araújo, G.: Circumventing uniqueness of XOR arbiter PUFs. In: 2019 22nd Euromicro Conference on Digital System Design (DSD). pp. 222–229 (2019)
31. Jendral, S., Ngo, K., Wang, R., Dubrova, E.: A single-trace message recovery attack on a masked and shuffled implementation of CRYSTALS-Kyber. Cryptology ePrint Archive, Paper 2023/1587 (2023), `https://eprint.iacr.org/2023/1587`
32. Kannwischer, M.J., Rijneveld, J., Schwabe, P., Stoffelen, K.: pqm4: Testing and benchmarking NIST PQC on ARM Cortex-M4. Cryptology ePrint Archive, Paper 2019/844 (2019), `https://eprint.iacr.org/2019/844`
33. Karabulut, E., Alkim, E., Aysu, A.: Single-trace side-channel attacks on -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 Computer Society, Los Alamitos, CA, USA (dec 2021). `https://doi.org/10.1109/HOST49136.2021.9702284`
34. 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`
35. Kim, J., Picek, S., Heuser, A., Bhasin, S., Hanjalic, A.: Make some noise. Unleashing the power of convolutional neural networks for profiled side-channel analysis. IACR Transactions on Cryptographic Hardware and Embedded Systems **2019**(3), 148–179 (May 2019)
36. Kocher, P., Jaffe, J., Jun, B.: Differential power analysis. In: Advances in Cryptology — CRYPTO' 99. pp. 388–397. Springer (1999)
37. Kocher, P.C.: Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In: Proc. of the 16th Annual Int. Cryptology Conf. on Advances in Cryptology. pp. 104–113 (1996)
38. Lenstra, A.K., Hendrik W. Lenstra, J., Lovász, L.: Factoring polynomials with rational coefficients. Mathematische Annalen **261**, 515–534 (1982)
39. Liu, Y., Zhou, Y., Sun, S., Wang, T., Zhang, R., Jingdian, M.: On the security of lattice-based Fiat-Shamir signatures in the presence of randomness leakage. IEEE Transactions on Information Forensics and Security **PP**, 1–1 (12 2020). `https://doi.org/10.1109/TIFS.2020.3045904`
40. 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)
41. Lyubashevsky, V.: Round 1 official comment: CRYSTALS-Dilithium. NIST PQC Project document (19 September 2018), `https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/round-1/official-comments/CRYSTALS-DILITHIUM-official-comment.pdf`
42. Maghrebi, H., Portigliatti, T., Prouff, E.: Breaking cryptographic implementations using deep learning techniques. In: Carlet, C., Hasan, M.A., Saraswat, V. (eds.) Security, Privacy, and Applied Cryptography Engineering. pp. 3–26. Springer International Publishing, Cham (2016)

43. Maghrebi, H., Servant, V., Bringer, J.: There is wisdom in harnessing the strengths of your enemy: Customized encoding to thwart side-channel attacks. In: Fast Software Encryption. pp. 223–243 (2016)
44. 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`
45. Mattsson, J., Thormarker, E., Smeets, B.: Migration to quantum-resistant algorithms in mobile networks. Ericsson blog (8 February 2023), `https://www.ericsson.com/en/blog/2023/2/quantum-resistant-algorithms-mobile-networks`
46. Migliore, V., Gérard, B., Tibouchi, M., Fouque, P.A.: Masking Dilithium: Efficient implementation and side-channel evaluation. Cryptology ePrint Archive, Paper 2019/394 (2019), `https://eprint.iacr.org/2019/394`
47. Moody, D.: Status Report on the Third Round of the NIST Post-Quantum Cryptography Standardization Process. Nistir 8309 pp. 1–27 (2022), `https://nvlpubs.nist.gov/nistpubs/ir/2022/NIST.IR.8413.pdf`
48. NewAE Technology Inc.: Chipwhisperer, `https://rtfm.newae.com/Capture/ChipWhisperer-Pro/`
49. Ngo, K., Dubrova, E.: Side-channel analysis of the random number generator in STM32 MCUs. In: Proc. of the Great Lakes Symposium on VLSI (GLSVLSI '22) (2022)
50. Perin, G., Chmielewski, Ł., Picek, S.: Strength in numbers: Improving generalization with ensembles in machine learning-based profiled side-channel analysis. IACR Transactions on Cryptographic Hardware and Embedded Systems pp. 337–364 (2020)
51. Picek, S., Samiotis, I.P., Kim, J., Heuser, A., Bhasin, S., Legay, A.: On the performance of convolutional neural networks for side-channel analysis. In: Chattopadhyay, A., Rebeiro, C., Yarom, Y. (eds.) Security, Privacy, and Applied Cryptography Engineering. pp. 157–176. Springer International Publishing, Cham (2018)
52. Qiao, Z., Liu, Y., Zhou, Y., Ming, J., Jin, C., Li, H.: Practical public template attacks on CRYSTALS-Dilithium with randomness leakages. IEEE Transactions on Information Forensics and Security **18**, 1–14 (2023)
53. 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`
54. 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`
55. Schneider, M., Gama, N.: Darmstadt svp challenges (2010), `https://www.latticechallenge.org/svp-challenge/index.php`
56. Schnorr, C.P., Euchner, M.: Lattice basis reduction: Improved practical algorithms and solving subset sum problems. Mathematical programming **66**, 181–199 (1994)
57. Sim, B.Y., Kwon, J., Lee, J., Kim, I.J., Lee, T.H., Han, J., Yoon, H., Cho, J., Han, D.G.: Single-trace attacks on message encoding in lattice-based KEMs. IEEE Access **8**, 183175–183191 (2020)
58. Sim, B.Y., Park, A., Han, D.G.: Chosen-ciphertext clustering attack on CRYSTALS-KYBER using the side-channel leakage of Barrett reduction. IEEE Internet of Things Journal (2022)
59. Timon, B.: Non-profiled deep learning-based side-channel attacks. IACR Cryptology ePrint Archive, Report 2018/196 (2018)

60. Veyrat-Charvillon, et al.: Shuffling against side-channel attacks: A comprehensive study with cautionary note. In: Advances in Cryptology – ASIACRYPT 2012. pp. 740–757. Springer Berlin Heidelberg (2012)
61. Wang, H., Dubrova, E.: Tandem deep learning side-channel attack against FPGA implementation of AES. In: 2020 IEEE International Symposium on Smart Electronic Systems (iSES) (Formerly iNiS). pp. 147–150 (2020)
62. Wang, H., Forsmark, S., Brisfors, M., Dubrova, E.: Multi-source training deep learning side-channel attacks. In: IEEE 50th International Symposium on Multiple-Valued Logic (ISMVL'2020) (2020)
63. Wu, L., Picek, S.: Remove some noise: On pre-processing of side-channel measurements with autoencoders. IACR Transactions on Cryptographic Hardware and Embedded Systems pp. 389–415 (2020)