Exploiting Intermediate Value Leakage in Dilithium: A Template-Based Approach

Alexandre Berzati[1], Andersson Calle Viera[1,2], Maya Chartouny[1,3], Steven Madec[1], Damien Vergnaud[2] and David Vigilant[1]

[1] Thales DIS, France
{alexandre.berzati,andersson.calle-viera,maya.saab-chartouni,steven.madec,david.vigilant}@thalesgroup.com
[2] Sorbonne Université, CNRS, Inria, LIP6, F-75005 Paris, France
[3] Université Paris-Saclay, UVSQ, CNRS, Laboratoire de mathématiques de Versailles, 78000, Versailles, France

Abstract. This paper presents a new profiling side-channel attack on CRYSTALS-Dilithium, the new NIST primary standard for quantum-safe digital signatures. An open source implementation of CRYSTALS-Dilithium is already available, with constant-time property as a consideration for side-channel resilience. However, this implementation does not protect against attacks that exploit intermediate data leakage. We show how to exploit a new leakage on a vector generated during the signing process, for which the costly protection by masking is still a matter of debate. With a corpus of 700 000 messages, we design a template attack that enables us to efficiently predict whether a given coefficient in one coordinate of this vector is zero or not. By gathering signatures and being able to make the correct predictions for each index, and then using linear algebra methods, this paper demonstrates that one can recover part of the secret key that is sufficient to produce universal forgeries. While our paper deeply discusses the theoretical attack path, it also demonstrates the validity of the assumption regarding the required leakage model from practical experiments with the reference implementation on an ARM Cortex-M4. We need approximately a day to collect enough representatives and one more day to perform the traces acquisition on our target.

Keywords: Dilithium · Digital signature · Lattice-based cryptography · Post-quantum cryptography · Side-channel attacks · Template Attacks · Learning with Errors

1 Introduction

Public-key cryptography relies on well-defined mathematical assumptions to enable secure communication over unsecured channels. For instance, RSA [RSA78] security relies upon the difficulty of factoring large integers, and the Diffie-Hellman key exchange [DH76] is based on the hardness of the discrete logarithm. However, assuming a sufficiently powerful quantum computer would be available, Shor has proposed a quantum algorithm [Sho94] that can break those cryptosystems in polynomial time.

Shor's algorithm highlights the need for new cryptographic schemes, and *post-quantum cryptography* (PQC) is an emerging field of cryptography focused on developing cryptographic algorithms that are secure against quantum (and classical) computers. The National Institute of Standards and Technology (NIST) has initiated a process to standardize new quantum-resistant public key cryptographic algorithms. Three PQC signature algorithms were selected in 2022: CRYSTALS-Dilithium [LDK+22], Falcon [SBN+21], and

SPHINCS+ [BHK+19]. This paper focuses on CRYSTALS-Dilithium (also referred as Dilithium in the sequel) signature algorithm, that NIST recommends for most of the use cases [AASA+22]. Dilithium is a signature scheme based on the Fiat-Shamir heuristic and its security relies on the hardness of the Module Learning with Errors and Module Short Integer Solutions problems. Given a public matrix $\mathbf{A}$ of elements in the ring $R_q = \mathbb{Z}_q[X]/(X^n + 1)$ for integers n and q, the signing algorithm samples a vector $\mathbf{y}$ and computes $\mathbf{w} = \mathbf{A}\,\mathbf{y}$. The algorithm extracts high-order and low-order bits out of each coefficient of $\mathbf{w}$ using the `Decompose` function that outputs the `HighBits` and `LowBits` vectors denoted respectively $\mathbf{w}_1$ and $\mathbf{w}_0$. Similarly to Schnorr signature [Sch91], a challenge c is derived from the message being signed and $\mathbf{w}_1$ by using some hash function (modelized as a random oracle in the security analysis) and the signing algorithm computes $\mathbf{z}$ as $\mathbf{y} + c\,s_1$ (where s_1 is part of the signer's secret key) and returns the pair $(\mathbf{z}, c)$ as the signature (if it satisfies some security properties). The vector $\mathbf{w}_1$ can be recovered publicly from the signature $(\mathbf{z}, c)$ but the vector $\mathbf{w}_0$ cannot. We show here that a side-channel attack on $\mathbf{w}_0$ may actually reveal the signer's secret key.

Side-channel attacks exploit the physical implementation of cryptographic systems, such as the power consumption, electromagnetic emanation or timing information to infer secrets. Dilithium reference implementation [DKL+22] already proposes a very compact constant-time open source code. Still, this implementation does not claim protection against more powerful attacks exploiting intermediate data leakage, such as differential power analysis (DPA) [KJJ99]. It was shown in [MGTF19] that some data leakages exist, and some may be exploitable (see also [MGTF19, RJH+18, MUTS22, ABC+22, LZS+21, QLZ+23] and references therein). Many works were devoted to present secure implementation of Dilithium signatures against side-channel attacks [MGTF19, RJH+18, MUTS22, ABC+22]. Similarly to the group-based setting or Schnorr/ECDSA signatures, it was advocated that the nonce $\mathbf{y}$ should be protected in order not to leak information on the signing key.

Our Contributions. In this paper, we present a new attack path on both randomized and deterministic versions of Dilithium. We show that we are able to exploit a leakage on the `Decompose` function in practice on a publicly available implementation through a template approach.

The vulnerability of the vector $\mathbf{w}_0$ was until more recently still unclear. In [MGTF19], Migliore, Gérard, Tibouchi, and Fouque presented a masked implementation of Dilithium where $\mathbf{w}_0$ was not protected due to the important overhead of doing so and mentioned that the `Decompose` function is "*by far the most complex operations regarding masking*" in Dilithium. *A contrario*, Azouaoui, Bronchain, Cassiers, Hoffmann, Kuzovkova, Renes, Schneider, Schönauer, Standaert and van Vredendaal have recommended to protect $\mathbf{w}_0$ in [ABC+22] but without giving an attack path. For the first time we manage to exploit the leakage on $\mathbf{w}_0$ in a practical attack setting showing that it should be indeed protected.

Our theoretical attack relies on the simple observation that from a signature where one coordinate of $\mathbf{w}_0$ is null, one can obtain the value of the corresponding coordinate of another secret information used in the signing algorithm. Once this secret information has been completely reconstructed, it is possible to retrieve s_1 using linear algebra. The knowledge of s_1 allows to forge a valid signature on any arbitrary message [RJH+18] [MGTF19]. Note that if there is a loss of precision due to a signal acquisition or micro-architectural effect, error can be bounded using error management techniques such as described in Section 3.4.

For the *proof-of-concept* we consider the C cryptographic library PQClean [KSSW] and use the ChipWhisperer-Lite 32-bit with an STM32f3 micro-controller (powered by an ARM Cortex M4). We chose this setup for various reasons, mainly because the CW is used in the literature for its convenience and repeatability, and because the ARM Cortex M4 is widely used for PQC implementations and benchmarks. We design a template attack that enables us to predict whether a given coefficient of $\mathbf{w}_0$ is zero or not. For

the sake of simplicity, we intend to build the template on the first coefficient and then apply the same one to the remaining coefficients in order to distinguish if we have 0 or not. Our goal was to demonstrate that a template built on one `Decompose` could be applied to any other coefficient. It is reasonable to assume that the `Decompose` of any rejection round behaves similarly. We implement the template such as described in [CK13]. By doing the profiling and the matching phase on the above setup, we prove the feasibility of exploiting such leakage of $\mathbf{w}_0$ in practice. For the profiling phase, we compute incomplete signatures. Indeed, in order to speed up the process and to limit the storage, we stop after the execution of the `Decompose` step. We obtain side-channel leakages together with $\mathbf{w}_0$ without retrieving $\mathbf{y}$. In the matching phase, we only work with computation made on the first round of rejection of Dilithium. With the same template, we are able to match correctly the first 100 coefficients treated in the `Decompose` before any rejection.

With a corpus of 700 000 messages to build our template, we provide a comprehensive analysis of a relevant leakage model on $\mathbf{w}_0$ in Section 4. Without setup optimizations, it takes less than a day to collect enough representatives, and one more day to perform the traces acquisition on our target. In a realistic attack setting, detecting whether some coordinate of $\mathbf{w}_0$ is null using noisy power traces might not be error-free. Thus we provide a technique to filter potential values from signatures. We achieve manageable false positive (i.e. when we wrongly predict a coefficient $\mathbf{w}_0 = 0$) rate of 0.067%, and false-negative (i.e. when we predict a coefficient $\mathbf{w}_0 \neq 0$ but it actually is 0) rate of 0.1174%. Then supposing that we are able to retrieve at least a set of $k \times n$ signatures with the corresponding indices of $\mathbf{w}_0$, we show how to build and solve the system, while managing the errors.

2 Background

This section provides essential background information for a better understanding of the attack.

2.1 Notations

Let us note by $\mathbb{Z}_q$ the ring of integers modulo q and by $\mathbb{Z}_q[X] = (\mathbb{Z}/q\mathbb{Z})[X]$ the set of polynomials with integer coefficients modulo q. We define by $R = \mathbb{Z}[X]\Big/(X^n + 1)$ the ring of polynomials with integer coefficients, reduced by the cyclotomic polynomial $X^n + 1$ and $R_q = \mathbb{Z}_q[X]\Big/(X^n + 1)$ the ring of polynomials with integer coefficients modulo q, reduced by the cyclotomic polynomial $X^n + 1$.

In the following, all polynomial operations are performed in R_q except if specified otherwise.

For an even (resp. odd) positive integer α, we define $r_0 = r \mod^{\pm} \alpha$ to be the unique element r' in the range $-\frac{\alpha}{2} < r' \leq \frac{\alpha}{2}$ (resp. $-\frac{\alpha-1}{2} < r' \leq \frac{\alpha-1}{2}$) such that $r \equiv r \mod \alpha$. For $\alpha \in \mathbb{N}$, we define $r' = r \mod^{+} \alpha$ to be the unique element r' in the range $0 \leq r' < \alpha$.

For an element $w \in \mathbb{Z}_q$, we define $\|w\|_\infty$ as $|w \mod^{\pm} q|$. For an element $\mathbf{w} \in R$, i.e., $\mathbf{w} = w_0 + w_1 X + \ldots + w_{n-1} X^{n-1}$, we define $\|\mathbf{w}\|_\infty$ as $\max_i \|w\|_\infty$ and we define $\|\mathbf{w}\| = \sqrt{\|w_0\|_\infty^2 + \|w_1\|_\infty^2 + \ldots + \|w_{n-1}\|_\infty^2}$.

Let S_η be all the elements $\mathbf{w} \in R$ such that $\|\mathbf{w}\|_\infty \leq \eta$ and $\tilde{S}_\eta$ the set $\{\mathbf{w} \mod^{\pm} 2\eta : \mathbf{w} \in R\}$.

We will denote by $\mathtt{Rot}_j(c)$, the rotated coefficient vector of a polynomial c, rotated by j times in an anti-cyclic fashion. $[\![\mathtt{op}]\!]$ will be the boolean evaluation of the operation `op`.

2.2 Dilithium

Dilithium is a post-quantum digital signature proposed by the "Cryptographic Suite for Algebraic Lattices" (CRYSTALS) team. In 2022, NIST selected it as the new primary standard for quantum-safe digital signatures. It is based on the Fiat-Shamir with aborts principle [Lyu09], and its security can be reduced to the hardness of solving two lattices problems over module lattices, module learning with errors (M-LWE) [LDK+22], and SelfTargetMSIS [KLS18]. When writing this paper, the latest specification version is 3.1. It is designed with four main ideas in mind: simple to implement securely, conservative with parameters, minimizing the size of the public key and the size of the signature, and finally, easy to vary security. The essential functions of Dilithium scheme consist of `KeyGen` to generate the key, `Sign` to produce the signature of a message, and `Verify` to verify the signature.

Key Generation. The key generation algorithm is described in Algorithm 1. It generates a matrix $\mathbf{A}$ from a seed ρ via some function `ExpandA` (see [LDK+22]). Afterwards, the algorithm samples random secret key vectors s_1 and s_2. Each coefficient of these vectors is an element of R_q with small coefficients of size at most η. Finally, the second part of the public key is computed as $\mathbf{t} = \mathbf{A}\, s_1 + s_2$.

Algorithm 1 KeyGen

1 $\zeta \leftarrow \{0,1\}^{256}$
2 $(\rho, \rho', K) \in \{0,1\}^{256} \times \{0,1\}^{512} \times \{0,1\}^{256} := \mathrm{H}(\zeta)$ ▷ H instantiated as SHAKE-256
3 $\mathbf{A} \in R_q^{k \times l} := \texttt{ExpandA}(\rho)$ ▷ $\mathbf{A}$ is generated and stored in NTT Representation as $\hat{\mathbf{A}}$
4 $(s_1,\, s_2) \in S_\eta^l \times S_\eta^k := \texttt{ExpandS}(\rho')$
5 $\mathbf{t} := \mathbf{A}\, s_1 + s_2$ ▷ Compute $\mathbf{A}s_1$ as $\texttt{NTT}^{-1}(\hat{\mathbf{A}} \cdot \texttt{NTT}(s_1))$
6 $(\mathbf{t}_1,\, \mathbf{t}_0) := \texttt{Power2Round}_q(t,\, d)$
7 $tr \in \{0,1\}^{256} := \mathrm{H}(\rho \,||\, \mathbf{t}_1)$
8 **return** $pk = (\rho,\, \mathbf{t}_1)$, $sk = (\rho, K, tr, s_1, s_2, \mathbf{t}_0)$

Signature. The signing algorithm is described in Algorithm 2. It is constructed by a rejection sampling loop, generating a new signature until it satisfies some security properties. First, the algorithm generates a masking vector of polynomials $\mathbf{y}$ with coefficients less than γ_1. The signer then computes $\mathbf{w} = \mathbf{A}\mathbf{y}$, sets $\mathbf{w}_1$ to be the high-order bits of the coefficients in this vector, and $\mathbf{w}_0$ to be the low-order bits of $\mathbf{w}$. The challenge c is then created as the hash of the message and $\mathbf{w}_1$. The output c is a polynomial in R_q with exactly τ ± 1's and the rest 0's. The potential signature is computed, using s_1, as $\mathbf{z} = \mathbf{y} + c\, s_1$. To ensure that the signature does not leak information about the key and for correctness purposes, lines 13 and 17 execute some bound checks. If one of these verification fails, a new signature candidate is generated.

Signature verification. The verification algorithm is described in Algorithm 3. The verifier computes the high-order bits of $\mathbf{A}\,\mathbf{z} - c\,\mathbf{t}_1 \cdot 2^d$, and accepts if all the coefficients of $\mathbf{z}$ are less than $\gamma_1 - \beta$ provided that c is the hash of the message and $\mathbf{w}_1'$.

Decompose. To break up an element in $\mathbb{Z}_q$ into their "high-bits" and "low-bits", we can use the function `Decompose`. In fact, we divide w by α to get $w = w_1\,\alpha + w_0$ where $w_0 = r \bmod^{\pm} \alpha$ and $w_1 = (w - w_0)/\alpha$. Note that $\texttt{Decompose}(w) = (\texttt{HighBits}(w), \texttt{LowBits}(w)) = (w_1, w_0)$.

Algorithm 2 Sign

$\mathbf{A} \in R_q^{k \times l} := \texttt{ExpandA}(\rho)$ ▷ $\mathbf{A}$ is generated and stored in NTT Representation as $\hat{\mathbf{A}}$
$\mu \in \{0,1\}^{512} := \mathrm{H}(tr \,||\, M)$
$\kappa := 0$, $(\mathbf{z}, \mathbf{h}) := \perp$
$\rho' \in \{0,1\}^{512} := \mathrm{H}(K \,||\, \mu)$ (or $\rho' \leftarrow \{0,1\}^{512}$ for randomized signing)
while $(\mathbf{z}, \mathbf{h}) = \perp$ **do** ▷ Pre-compute $\hat{s}_1 := \texttt{NTT}(s_1)$, $\hat{s}_2 := \texttt{NTT}(s_2)$ and $\hat{\mathbf{t}}_0 := \texttt{NTT}(\mathbf{t}_0)$
  $\mathbf{y} \in \tilde{S}^l_{\gamma_1} := \texttt{ExpandMask}(\rho', \kappa)$
  $\mathbf{w} := \mathbf{A}\,\mathbf{y}$ ▷ $\mathbf{w} := \texttt{NTT}^{-1}(\hat{\mathbf{A}} \cdot \texttt{NTT}(\mathbf{y}))$
  $\mathbf{w}_1, \mathbf{w}_0 = \texttt{Decompose}_q(\mathbf{w}, 2\,\gamma_2)$
  $\tilde{c} \in \{0,1\}^{256} := \mathrm{H}(\mu \,||\, \mathbf{w}_1)$
  $c \in B_\tau := \texttt{SampleInBall}(\tilde{c})$ ▷ Store c in NTT representation as $\hat{c} = \texttt{NTT}(c)$
  $\mathbf{z} := \mathbf{y} + c\,s_1$ ▷ Compute cs_1 as $\texttt{NTT}^{-1}(\hat{c} \cdot \hat{s}_1)$
  $\mathbf{r}_0 := \mathbf{w}_0 - cs_2$ ▷ Compute cs_2 as $\texttt{NTT}^{-1}(\hat{c} \cdot \hat{s}_2)$
  if $\|\mathbf{z}\|_\infty \geq \gamma_1 - \beta$ or $\|\mathbf{r}_0\|_\infty \geq \gamma_2 - \beta$ **then**
    $(\mathbf{z}, \mathbf{h}) := \perp$
  else
    $\mathbf{h} := \texttt{MakeHint}_q(\mathbf{w}_1, \mathbf{w}_0 - cs_2 + c\,\mathbf{t}_0, 2\,\gamma_2)$ ▷ Compute $c\,\mathbf{t}_0$ as $\texttt{NTT}^{-1}(\hat{c} \cdot \hat{\mathbf{t}}_0)$
    if $||c\,\mathbf{t}_0||_\infty \geq \gamma_2$ or $|\mathbf{h}|_{\mathbf{h}_j=1} > \omega$ **then**
      $(\mathbf{z}, \mathbf{h}) := \perp$
  $\kappa := \kappa + l$
return $\sigma = (\tilde{c}, \mathbf{z}, \mathbf{h})$

Algorithm 3 Verify

$\mathbf{A} \in R_q^{k \times l} := \texttt{ExpandA}(\rho)$
$\mu \in \{0,1\}^{512} := \mathrm{H}(\mathrm{H}(\rho \,||\, \mathbf{t}_1) \,||\, M)$
$c := \texttt{SampleInBall}(\tilde{c})$
$\mathbf{w}'_1 := \texttt{UseHint}_q(\mathbf{h}, \mathbf{A}\mathbf{z} - c\mathbf{t}_1 \cdot 2^d, 2\gamma_2)$
return $[\![||\mathbf{z}||_\infty < \gamma_1 - \beta]\!]$ and $[\![\tilde{c} = \mathrm{H}(\mu \,||\, \mathbf{w}'_1)]\!]$ and $[\![|\mathbf{h}|_{\mathbf{h}_j=1} \leq \omega]\!]$

We give in Figure 1 the C code of the PQCLean `Decompose` function that is implemented in the same way in most of the available libraries. In line 3, the value of w is first loaded without prior knowledge of w_1 and w_0. However, since w_0 is mainly contained in the lower bytes of w, there will likely be some leakage of the two least significant bytes at the beginning of the `Decompose` function. Furthermore, in the same line, by adding 127 to w, the least significant byte of w - which primarily contains information about w_0 - is altered, leading to early leakage of this byte in the traces. While lines 4 and 5 don't offer much information about the least significant bytes, line 7 involves subtracting $w_1 \times 2\,\gamma_2$ from w, which approximates w based on the most significant bytes, thereby directly leaking information on w_0. Additionally, if $w_1 \times 2\,\gamma_2 = w$, a significant visible difference between w and w_0 will be observed thus magnifying the leakage of this operation for the 0 value.

Details on Reference Implementation. We can find several differences between the specification of Dilithium [LDK+22] and the various implementations, such as the reference implementation [DKL+22] or the one from the PQClean Library [KSSW].

```
int32_t PQCLEAN_DILITHIUM2_CLEAN_Decompose(int32_t *w0, int32_t w) {
  int32_t w1;
  w1  = (w + 127) >> 7;
  w1  = (w1 * 11275 + (1 << 23)) >> 24;
  w1 ^= ((43 - w1) >> 31) & w1;

  *w0  = w - w1 * 2 * GAMMA2;
  *w0 -= (((Q - 1) / 2 - *w0) >> 31) & Q;
  return w1;
}
```

Figure 1: C Code of the `Decompose` function.

In this paragraph, we point out specific details that will be of interest for the attack.

In Algorithm 2 line 12, in order to optimize the complexity, we compute $\mathbf{r}_0$ as $\mathbf{w}_0 - c\,s_2$ and then we check whether $\|\mathbf{w}_0 - c\,s_2\| < \gamma_2 - \beta$. If this inequality holds then $\mathbf{w}_0 - c\,s_2$ is the low part of $\mathbf{w} - c\,s_2$ according to Lemma 3 in [LDK+22].
Note that $\mathbf{w}_0$ is already computed in the Algorithm 2 line 8, no matter if we use the function `Decompose` or `HighBits`. Lastly, another optimization is done line 16 during the computation of the vector $\mathbf{h}$, since we already have the value $\mathbf{w}_0 - c\,s_2$ we add $c\,\mathbf{t}_0$ to it and compare the result with $\mathbf{w}_1$ to produce the hint $\mathbf{h}$.

In the following, and if not stated otherwise, we will refer to the PQClean implementation of Dilithium which is slightly equivalent to the reference implementation.

2.3 Side-Channel Attacks on Dilithium

Concrete security was not the only priority for the NIST PQC competition as for the first time, an emphasis is also put on security against side-channel attacks.

One of the paper's objectives is to state if some parts of Dilithium implementation require specific treatment concerning side-channel attacks (SCA). In fact, this section first recalls template attack principle, which we will use to demonstrate the practicability of our new attack. The remaining of this section will give an overview of the physical attacks already published.

Template Attacks. Side-Channel Attacks have proven to be extremely effective as a practical way to attack cryptographic algorithm implementations. In particular, when open devices are available, with the possibility to set and modify the key value, profiling attacks such as templates can be very powerful. First introduced by Chari, Rao and Rohatgi [CRR03], this attack is divided into four steps. The first step, using a copy of the target device, records a large number of power traces using many different keys and inputs. The second step is creating a template of the device operation by selecting points of interest that are supposed to contain a considerable proportion of the leakage information. Third, the attacker records a small number of power traces using multiple plaintexts on the victim's device. Finally, the template is applied to the attack traces. For each possible subkey, the attacker determines which value is most likely to be the correct one and repeats the same step until the entire key is recovered.

This paper describes an application of a template attack as in [CK13]. We may also apply other types of templates and, more generally, other profiling SCA methods. The efficiency of a profiling attack is better when the clone (hardware and software) is similar to the target. Since we are working on a public implementation and a public evaluation platform, this also shows that it is the best method to use here.

State of the art. Several practical Fault Injection Attacks leading to the key recovery have already been published. Indeed Bruinderink and Pessl [BP18] showed the applicability of differential fault attacks on Dilithium using multiple paths. The attacker can produce a correct signature and a faulty signature. By faulting somewhere in the generation of the challenge c, without changing the value of y, the attacker can produce a different variable z under the same number of rejection rounds. From the correct signature, we have $z = y + c\,s_1$, and from the faulty signature, we have $z' = y + c'\,s_1$. Hence, $z - z' = (c - c')\,s_1$ and therefore we can compute $s_1 = (c - c')^{-1}(z - z')$.

Furthermore, Bruinderink and Pessl also showed, as in other papers [RJH+18] [MGTF19], that it is possible to forge signatures with only the extracted portion of the secret key s_1.

Regarding SCA, previous work also has shown that some leakages during the signature could be exploited to infer the whole Dilithium key. For example, Fournaris, Dimopoulos and Koufopavlou [FDK20] demonstrated a correlation power analysis of the polynomial multiplication $c\,s_1$. The analysis focused on the polynomial multiplication operation during the sample rejection loop, regardless of the technique used (schoolbook polynomial multiplication, sparse matrix polynomial multiplication, or Number Theoretic Transform (NTT) representation).

Also, Kim, Lee, Han, Sim and Han [KLH+20] proposed a machine learning-based profiling attack on unprotected and masked versions of Dilithium-2. They targeted the leakage during load save and reduction operation. They showed that to obtain all the secret key parts in the algorithm, only two of s_1, s_2, t_0 are needed. Then by solving a simple equation, the attacker can find the third one. To validate their attack, the authors targeted the s_1 value and made experiments in a simulated setting. A uniform noise is added to the trace supposing 8-bit Hamming weight leakage and linear regression model.

More recently, Mazougui *et al.* [MUTS22] showed an end-to-end profiled side-channel attack on Dilithium targeting the nonce y. They identified a leak in the generation of one coefficient of y by using machine learning to distinguish whether the coefficient is zero or not. Their attack extracts information about the vector y from a single trace. Given coefficient $y_i = 0$ then $z_i = 0 + (c\,s_1)_i$. Since z and c are known at the end of the signature, collecting many of these equations allows the attacker to recover all s_1 coefficients using linear algebra.

3 A new theoretical attack on Dilithium

This section presents a new theoretical path of profiling attacks on Dilithium. The prerequisite is a sufficiently strong leakage of the value $\mathbf{w}_0$. To our knowledge, this is the first complete attack exploiting a leakage on $\mathbf{w}_0$, even if Azouaoui *et al.* already pointed out that protecting $\mathbf{w}_0$ would be a good recommendation [ABC+22]. We propose some methods to minimize the error during the matching phase so that we introduce as few errors as possible in our equations. We then show how the system of equations can be solved using the Least Squares Method (LSM). We also propose an error management technique, which computes the correct secret key, although we have introduced few errors. Finally, we highlight the number of signatures required to perform this attack and multiple vulnerable operations in the reference implementation that may render the attack possible.

3.1 Main Idea

Our goal is to recover the secret key s_1 because knowing s_1 is sufficient to sign arbitrary messages [RJH+18], [MGTF19] and [BP18]. The attack consists in collecting signatures for which one coefficient of $\mathbf{w}_0$ is known (from profiling attacks) to be equal to 0. As discussed at the end of Section 2.2, $\mathbf{w}_0$ value is computed in the reference implementation without any particular protection.

The signature is composed of $(c, \mathbf{z}, \mathbf{h})$. Since $\mathbf{z} = \mathbf{y} + c\,s_1$ and $\mathbf{t} = \mathbf{A}\,s_1 + s_2$ we have

$$\mathbf{A}\,\mathbf{z} - c\,\mathbf{t} = \mathbf{A}\,\mathbf{y} - c\,s_2. \tag{1}$$

By replacing $\mathbf{w} = \mathbf{A}\,\mathbf{y}$ and $\mathbf{t} = t_1\,2^d + \mathbf{t}_0$ in Equation (1), we get

$$\mathbf{A}\,\mathbf{z} - c\,\mathbf{t}_1\,2^d - c\,\mathbf{t}_0 = \mathbf{w} - c\,s_2,$$

and by rewriting this equation, we obtain

$$\mathbf{A}\,\mathbf{z} - c\,\mathbf{t}_1\,2^d = \mathbf{w} + c\,(\mathbf{t}_0 - s_2).$$

From the `Decompose` function we get that $\mathbf{w} = \mathbf{w}_1\,2\,\gamma_2 + \mathbf{w}_0$, the above equation thus becomes

$$\mathbf{A}\,\mathbf{z} - c\,\mathbf{t}_1\,2^d = \mathbf{w}_1\,2\,\gamma_2 + \mathbf{w}_0 + c\,(\mathbf{t}_0 - s_2),$$

and if we suppose that for some (i, j), $(\mathbf{w}_0)_{i,j} = 0$, then we have

$$\left(\mathbf{A}\,\mathbf{z} - c\,\mathbf{t}_1\,2^d\right)_{i,j} = \left(\mathbf{w}_1\,2\,\gamma_2 + c\,(\mathbf{t}_0 - s_2)\right)_{i,j}. \tag{2}$$

Given that the elements $\mathbf{A}$, $\mathbf{z}$, c, $\mathbf{t}_1$, d, γ_2 are public and that $\mathbf{w}_1$ can be computed from the verification part, we can recover $(\mathbf{t}_0 - s_2)_{i,j}$ from Equation (2). We repeat the same step for all i and j such that $0 \leq i < k$ and $0 \leq j < n$ in order to fully recover $\mathbf{t}_0 - s_2$. Finally, the knowledge of $\mathbf{t}_0 - s_2$ allows to find s_1:

$$\mathbf{t} = \mathbf{A}\,s_1 + s_2 = \mathbf{t}_1\,2^d + \mathbf{t}_0,$$

hence,

$$\mathbf{A}\,s_1 = \mathbf{t}_1\,2^d + (\mathbf{t}_0 - s_2). \tag{3}$$

However, $\mathbf{A}$ is not square in all the Dilithium versions, but if we multiply the Equation (3) by $\mathbf{A}^t$, we get that $(\mathbf{A}^t\,\mathbf{A})$ is square and is invertible with very high probability [ABC+22]. Thus, we get

$$s_1 = (\mathbf{A}^t\,\mathbf{A})^{-1}\,\mathbf{A}^t\,\left(\mathbf{t}_1\,2^d - (s_2 - \mathbf{t}_0)\right).$$

Knowing s_1 suffices to sign arbitrary messages [RJH+18].

Remark. In this case we focus on distinguishing signatures for which $\mathbf{w}_0 = 0$ from those where $\mathbf{w}_0 \neq 0$. We will denote by `Classifier` this procedure. In theory, any constant could have been used. Factoring in other known constants in Equation (2) does not affect the resolution. However 0 may be more practical for the attack. Indeed, assuming the different Hamming Weights (HW) of $\mathbf{w}_0$ can be distinguished with a template, then the value 0 would be uniquely discriminated by its HW.

3.2 Candidates Filtering

In practice, there might be some noise when measuring a signal. This may lead to a wrong prediction that $(\mathbf{w}_0)_{i,j}$ is equal to 0, with introductions of errors in our system of equations. In order to overcome this, we need to remove the values where $(\mathbf{w}_0)_{i,j} \neq 0$ as early as possible to reduce the number of wrong equations. Hence, we put a filter on the value of $\mathbf{A}\,\mathbf{z} - c\,\mathbf{t}_1\,2^d - \mathbf{w}_1\,2\,\gamma_2$.

Under the assumption that $(\mathbf{w}_0)_{i,j} = 0$ for some (i, j) and from Equation (2) we have

$$\left(\mathbf{A}\,\mathbf{z} - c\,\mathbf{t}_1\,2^d - \mathbf{w}_1\,2\,\gamma_2\right)_{i,j} = \left(c\,(\mathbf{t}_0 - s_2)\right)_{i,j}.$$

Since $(s_2)_{i,j} << (\mathbf{t}_0)_{i,j}$,

$$\left(\mathbf{A}\,\mathbf{z} - c\,\mathbf{t}_1\,2^d - \mathbf{w}_1\,2\,\gamma_2\right)_{i,j} \approx (c\,\mathbf{t}_0)_{i,j}.$$

According to the specification of Dilithium [LDK+22], we can assume that the low order bits are uniformly distributed, i.e., $\mathbf{t}_0 \sim \mathcal{U}([-2^{d-1}+1,\ 2^{d-1}])$. Therefore, since c is a vector with exactly τ coefficients in $\{-1,\ 1\}$ and the rest equal to 0, by the central limit theorem, $c\,\mathbf{t}_0$ is normally distributed with mean 0 and standard deviation σ, i.e.,

$$\begin{cases} c\,\mathbf{t}_0 & \sim \mathcal{N}(0,\sigma^2) \\ \sigma^2 & = \frac{(2^{d-1}+}{(}2^{d-1}+1)+1)^2 - 112\,\tau = \frac{2^{2d}-1}{12}\,\tau \end{cases}$$

Hence, the probability that $|(\mathbf{A}\,\mathbf{z} - c\,\mathbf{t}_1\,2^d - \mathbf{w}_1\,2\,\gamma_2)_{i,j}|$ is lower or equal to $2\,\sigma$ is equal to 0.954.
Therefore, if we assume that $(\mathbf{w}_0)_{i,j} = 0$ then,

$$|(\mathbf{A}\,\mathbf{z} - c\,\mathbf{t}_1\,2^d - \mathbf{w}_1\,2\,\gamma_2)_{i,j}| \leq 2\,\sigma. \tag{4}$$

For instance, in Dilithium-2, we have $2\,\sigma = 2\sqrt{\frac{2^{26}-1}{12}\,39} \approx 29\,537$. Experimentally, we can discard approximately 70% of the $N := n \times k = 1024$ values where we might not have zero.

Using this filter, we may remove around 5% of the equations where actually $(\mathbf{w}_0)_{i,j} = 0$. However, in case of the presence of noise, it might be preferable to perform more signatures rather than introducing errors in the system of equations.

For example, let us consider a side-channel analysis that exploits a HW leakage. In the presence of noise, this filter will be be especially efficient to not inject equations for which $\mathbf{w}_0$ has a large value but a low HW.

3.3 Resolution - Least Square Method

Let us write $e = (\mathbf{w}_0)_{i,j}$. If our classifying process manages to detect zero values with enough precision, then this coefficient vanishes, and we get Equation (2). On the other hand, if our classifier is wrong and tells us that the value of $(\mathbf{w}_0)_{i,j} = 0$, but in fact, we do not have a zero, then we will get samples of the form

$$\left(\mathbf{A}\,\mathbf{z} - c\,\mathbf{t}_1\,2^d\right)_{i,j} = \left(\mathbf{w}_1\,2\,\gamma_2 + c\,(\mathbf{t}_0 - s_2)\right)_{i,j} + e,$$

where e is the error. Note that for most of our equation $e = 0$. This equation can be rewritten as

$$\left(\mathbf{A}\,\mathbf{z} - c\,\mathbf{t}_1\,2^d - \mathbf{w}_1\,2\,\gamma_2\right)_{i,j} = \left(c\,(\mathbf{t}_0 - s_2)\right)_{i,j} + e. \tag{5}$$

Let L be the left-hand side of Equation (5) for all $(i,\,j)$, and let $\mathbf{C}$ be the negacyclic matrix representation of the corresponding $c_{i,j}$ for all $(i,\,j)$. Since $\|\mathbf{C}\,\mathbf{t}_0\| \leq \tau 2^{d-1}$, $\|\mathbf{C}\,s_2\| \leq \beta$ and $\|e\| \leq 2\,\gamma_2$, then $\|\mathbf{C}\,(\mathbf{t}_0 - s_2) + e\| < q$, and thus we do not have a modular reduction. Therefore, this problem can be viewed as an LWE without modular reduction [BDE+18]. We use some statistical techniques to find an approximate solution of $(\mathbf{t}_0 - s_2)$ from the noisy linear system formed by equation of the form (5). If the matrix is not invertible we can use the Least Squares Method described by Bootle *et al.* in [BDE+18], we get a candidate solution

$$(\mathbf{t}_0 \tilde{-}\, s_2) = (\tilde{C}^T\,\tilde{C})^{-1}\tilde{C}^T\,L,$$

where $\tilde{C}^T$ is the matrix where for each vector i, each line j is comprised of the rotated corresponding challenge $c_{i,\,j}$.

If $\|(\mathbf{t}_0 - s_2) - (\mathbf{t}_0 \tilde{-}\, s_2)\|_\infty < \frac{1}{2}$, then $\lceil(\mathbf{t}_0 \tilde{-}\, s_2)\rfloor = (\mathbf{t}_0 - s_2)$. We get that most of the coefficients of $(\mathbf{t}_0 \tilde{-}\, s_2)$ are correct and the rest are wrong by ± 1, so either rounding up or down should yield the correct solution. Let us denote by `SOLVE` this procedure.

3.4 Error Management

It is important to notice that in most cases, the error term in our equations is equal to zero under the assumption that our classifier is correct in most instances. This means the solution may result from a "noisy" set of equations, i.e., with a small number of incorrect equations. Several techniques that can recover the correct value from a disturbed set of equations are described hereafter.

LWE with side information. With the "noisy" equations, our system looks like the LWE problem. Dachman-Soled, Ducas, Gong and Rossi [DDGR20] proposed a framework to integrate SCA "hints" to solve a LWE instance. In order to use hints from side-channel information, the learning with errors with side information instance is first transformed into a distorted bounded distance decoding (DBDD) instance. This transformation allows us to track the distribution of the secret vector. By providing hints, we can potentially modify this distribution to make it easier to find the solution vector. If we give enough hints, the secret vector recovery becomes feasible using lattice reduction attacks.

ILP. One can also see the problem of finding a given polynomial from this set of noisy equations as identifying the polynomial that maximizes the number of equations. In other words, we want to find the polynomial that fits most equations in our system. To this end, Marzougui *et al.* [MUTS22] formulated an Integer Linear Program (ILP) to solve this problem. Using the solution candidate obtained from the least-squares method, the authors use the big M [BJS11] method to factor in the noisy equations. Ultimately, the solution obtained from the ILP must correspond to the correct value.

Majority Vote. If we want the probability, of finding all $N = n \times k$ coefficients, to be equal to 1 then we will necessarily have some coordinates (i, j) where $(\mathbf{w}_0)_{i,j}$ will be equal to 0 several times. Moreover, if our distinguisher is incorrect on one coefficient at most, by taking two different equations, there is a small chance that it is wrong on the same coefficient. Since we are doing the resolution just before this step, either by inverting the matrix or by the LSM, we will only have some rounding problems, i.e., we will be very close to the exact value. By solving several systems with different equations, we will have several $\mathbf{t}_0 - s_2$ candidates. Then, we can perform a majority vote on each coefficient (i, j) to find the correct intermediate value and at the end retrieve the correct $\mathbf{t}_0 - s_2$.

3.5 Attack Summary

Pseudo-Algorithm 4 describes all steps to find a solution candidate $(\mathbf{t}_0 - s_2)$.

Number of Needed Signatures. The number of needed signatures is an important performance indicator of an SCA. The purpose of this subsection is to find out how many signatures are required for a complete attack. To recover all the $N = n \times k$ coordinates, where $(\mathbf{w}_0)_{i,j} = 0$ for i and j such that $0 \leq i < k$ and $0 \leq j < n$. We know that for a fixed (i, j), $(\mathbf{w}_0)_{i,j} = 0$ with probability $\frac{1}{2\gamma_2}$. Hence, one coordinate is not equal to zero with probability $1 - \frac{1}{2\gamma_2}$. Over T experiences, a coordinate is not equal to zero with probability $(1 - \frac{1}{2\gamma_2})^T$. Hence, a coordinate is equal to zero at least once with probability $1 - (1 - \frac{1}{2\gamma_2})^T$. Therefore, over T experiences the n coordinates are equal to zero at least once with probability $\left(1 - \left(1 - \frac{1}{2\gamma_2}\right)^T\right)^n$.

Algorithm 4 Equivalent Secret Key Recovery

1 $L = [\,[0] \times n] \times k,\ \tilde{C} = [\,[0] \times n] \times k$
2 **for** sig in S **do**
3 **for** $i = 0$ to $k - 1$ **do**
4 **for** $j = 0$ to $n - 1$ **do**
5 $z, c, h \leftarrow sig$
6 $val = (\mathbf{A}\,\mathbf{z} - c\,\mathbf{t}_1 2^d - \mathbf{w}_1 2\gamma_2)_{i,j}$
7 **if** $|val| \leq 2\sqrt{\frac{2^{2d}-1}{12}\tau}$ **then**
8 $(\tilde{\mathbf{w}}_0)^{sig}_{i,j} := \texttt{Classifier}(T_{(\tilde{\mathbf{w}}_0)^{sig}_{i,j}})$
9 **if** $(\tilde{\mathbf{w}}_0)^{sig}_{i,j} = 0$ **then**
10 $L[i][j] = val$
11 $\tilde{C}[i][j] = \texttt{Rot}_{255-j}(c)$
12 **for** $i = 0$ to $k - 1$ **do**
13 $\mathbf{t}_0\_minus\_s_2 = \texttt{SOLVE}(L[i], \tilde{C}[i])$

For instance, for Dilithium-2, $\gamma_2 = 95\,232$. In order to recover the $N = 256 \times 4 = 1024$ coordinates where $(\mathbf{w}_0)_{i,j} = 0$, we need 2.43 millions signatures. In this case, we get a high probability of 0.997 of recovering all the coefficients.
Note that in practice, for Dilithium-2, 2.5 millions signatures were needed to recover the $N = 256 \times 4 = 1024$ coordinates where $(\mathbf{w}_0)_{i,j} = 0$. This is consistent with our analysis as seen from the graph in Figure 2.

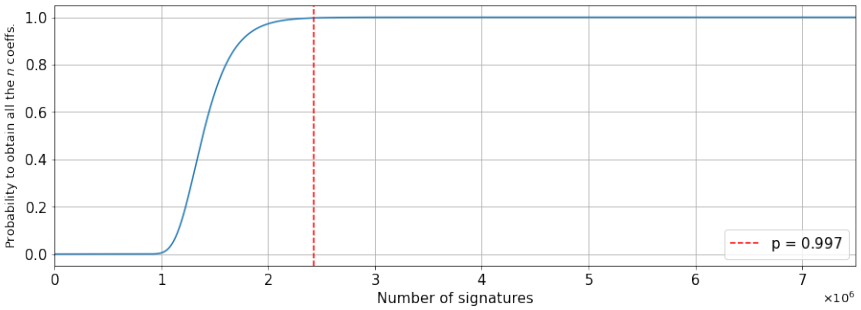

Figure 2: Probability of having recovered all the $N = n \times k$ coordinates for Dilithium-2.

3.6 Potential Leakage Spots Involving $\mathbf{w}_0$

This section discusses potential leaking operations in Dilithium.

Probably the most intuitive location is the `Decompose` function line 8, where the $\mathbf{w}$ value is split into its upper $\mathbf{w}_1$ and lower $\mathbf{w}_0$ parts. Especially in the reference implementation there are many accesses to $\mathbf{w}_0$. Previous work [MGTF19] has already shown that this sub-routine leaks information, but no attack on $\mathbf{w}_0$ was given, moreover it was shown to be hard to protect efficiently.

Furthermore, the computation of the $\mathbf{r}_0$ line 12 Algorithm 2 value performed in the normal domain by subtracting cs_2 to $\mathbf{w}_0$ is also a good candidate for an attack. If the targeted value is zero, we should have an apparent leakage in Hamming weight (HW). Even though it is a promising operation, this is let for future practical work.

Finally, the $\mathbf{r}_0$ value is also used in the `MakeHint` function on line 16 of Algorithm 2. It can also be considered as a leakage spot.

In this work, we focus on the leakage of the `Decompose` function.

4 Experimental Results

This section demonstrates the practical feasibility of the theoretical attack when no specific countermeasure is implemented. We aim to demonstrate the vulnerability of the $\mathbf{w}_0$ value in Dilithium. We simplify the attack as much as possible while still covering a wide range of study cases. Although we present extensively the practical results only for the first coefficient of the first coordinate of $\mathbf{w}_0$, it is important to note that the methodology and results can be extended to all the other indices of $\mathbf{w}_0$ with minimal changes, as the implementation manipulates each coefficient independently in the same way and performs the same constant-time operation. We provide at the end of this section arguments in favor of this hypothesis. All side-channel analyses were coded in Python and will soon be available as notebooks.

4.1 Setup

For our attack, we use the Post-quantum cryptographic library PQClean [KSSW], which offers "clean" standalone C implementations [KSSW] of most post-quantum schemes included in the NIST round 3 and 4 submissions [nis]. Focusing mainly on the embedded system domain, the ChipWhisperer-Lite 32-bit, with an STM32f3 micro-controller, is often used in related academic research. We chose it as a test bench to deploy the Dilithium-2 signature scheme. We compiled Dilithium-2 using the gcc-arm cross-compiler arm-none-eabi-gcc 10.2.1 (with the options -O3).

To go to the nitty-gritty, we use the ChipWhisperer (CW) "cw" library [OC14] for trace acquisition, and we use an open-source side-channel python library to process the side-channel information and perform the desired analysis. We also use the LSM and the rounding subroutines implemented in the "numpy" library [HMvdW+20]. Moreover, we use an optimized C reference implementation of Dilithium [DKL+22] running on an Intel Core i7 to speed-up the computation of intermediate values. We also use this code to determine the number of coefficients we discarded with our filter and the number of coefficients $\mathbf{w}_0 = 0$ not identified.

4.2 Learning Phase

The goal is to identify the points of interest (POI) where we have a leakage of the `Decompose` function and to build templates on these points. According to the source code and the design, we expect that we can distinguish $\mathbf{w}_0$ equal to the value 0 more easily than other values, as shown in Figure 1.

4.2.1 Reverse engineering

The ChipWhisperer Lite has limited space capacity and can only handle a maximum of 24 400 samples, making it challenging to identify essential features in traces or capture longer operation sequences such as an entire Dilithium signature.

To overcome this, we compute incomplete signatures by stopping when we obtain the value of the first $\mathbf{w}_0$. Hence, using the PQClean library default implementation, an extract of the executed code can be viewed in Figure 3, where the trigger signal is only active throughout the whole four `poly_Decompose` calls.

```
for (i = 0; i < K; ++i) {
  trigger_high();
  PQCLEAN_DILITHIUM2_CLEAN_poly_Decompose(&v1→vec[i],&v0→vec[i],&v→vec[i]);
  trigger_low();
}
```

Figure 3: PQClean Dilithium-2 signature generation code snippet.

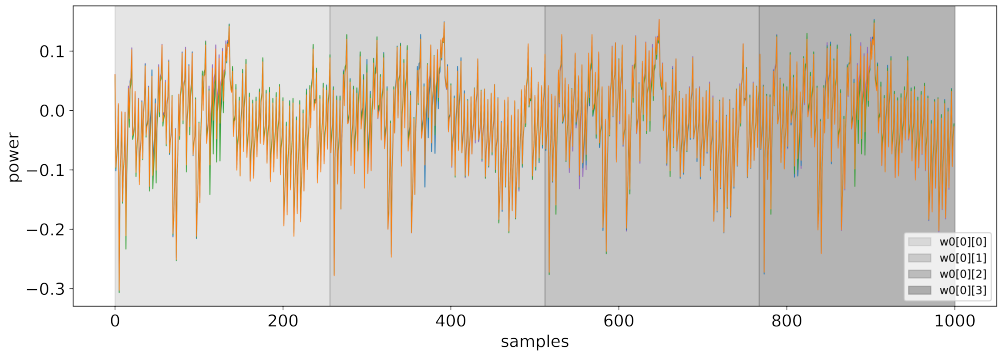

Figure 4: ChipWhisperer `Decompose`($\mathbf{w}$[0][j]), for j $\in \{0, 1, 2, 3\}$.

In Figure 4, we zoomed on a single `Decompose` for an acquisition performed directly on the ChipWhisperer. We highlight the four decompositions through the four shades of gray, where each coefficient of the polynomial $(\mathbf{w})_i$ is decomposed. We can see a repeating pattern that spans for about 260 samples, corresponding to the calls to the `Decompose` function.

Each polynomial of the vector $\mathbf{w}$ is independent as well as each value. Hence, $\mathbf{w}_0$ is also independent. Furthermore, the same operation `Decompose` is applied to each coefficient inside a loop. Without loss of generality, we can suppose that by creating a template on the first coefficient of $\mathbf{w}_0$, we can apply it to the other coefficients. One important particularity of the Dilithium signature scheme worth noting is the use of a rejection sampling loop. This can increase the complexity of the profiling step of the template. To overcome this difficulty, we decided only to consider the very first `Decompose` of the first round to build our template, even if more rejection rounds were computed.

4.2.2 Leakage model and $\mathbf{w}_0$ leakage detection

Since we are now able to identify the `Decompose` consumption profile, we will now focus on the leakage inferred by $\mathbf{w}_0$ and particularly when its value is equal to 0. To achieve this, we perform a reverse analysis of the intermediate value $\mathbf{w}_0$ for a relevant leakage model according to our setup and explain in detail how we constructed the dataset used to perform the corresponding analyses. For this purpose, we built a profiling data set with 700 000 traces collected from the ChipWhisperer.

Selection function. By default, the traces collected from the ChipWhisperer are easy to process as they do not require any synchronization. We know how the `Decompose` function works, and we can identify the inputs corresponding to the fixed outputs, so we choose the identity of $(\mathbf{w}_0)_{i,j}$ as a selection function to determine the intermediate values which will be targeted by our attack.

Distinguisher. As a second step of our analysis, we apply one of the many statistical tools widely used when building profiled attacks. In an unprotected setting, all these tests are expected to give the same results [MOS09]. Here, we choose to test the ANOVA distinguisher because it works with data partitioning, which is very similar to the templates partitioning mechanism[YJ21]. Based on the previous reverse analysis and the code review of the `poly_Decompose` function implementation presented in Figure 1, we identified a relevant leakage model.

Leakage model. We know that for all $0 \leq i < k$ and $0 \leq j < n$, $-\gamma_2 < (\mathbf{w}_0)_{i,j} \leq \gamma_2$. In our study case, for Dilithium-2, $\gamma_2 = (q-1)/88 = 95\,232$, which is `00 01 74 00` in hexadecimal, so if $(\mathbf{w}_0)_{i,j} \geq 0$ it results that $(\mathbf{w}_0)_{i,j}$ is bounded by this value. On the other hand if $(\mathbf{w}_0)_{i,j} < 0$ we have that $(\mathbf{w}_0)_{i,j}$ is between `FF FF FF FF` and `FF FE 8C 02`.

While only two classes are possible for the most significant byte (MSB), which is easily distinguishable, there are four possible groups for the second MSB. It can be trickier to distinguish between the cases `00` and `01` or the cases `FF` and `FE`.

As we will be performing the template only on the filtered values, we know that $|(\mathbf{w}_0 - c\,s_2 + c\,\mathbf{t}_0)_{i,j}| < 2\,\sigma$, but because $|c\,s_2| \leq \beta$ and $|c\,\mathbf{t}_0| < 2\,\sigma$ then it must hold that $|(\mathbf{w}_0)_{i,j}| < 2\,\sigma + \beta + 2\,\sigma$. Now, if $(\mathbf{w}_0)_{i,j} \geq 0$ the value is between `00 00 00 00` and `00 00 E7 0E` and if $(\mathbf{w}_0)_{i,j} < 0$ it is between `FF FF FF FF` and `FF FF 18 F2`. There are only two classes possible now for the two MSBs. Therefore, for Dilithium-2, the template attack can be simplified on the last two bytes to determine if we have a zero value which can potentially speed up the attack.

Since we are working on ChipWhisperer STM32F3 with a Cortex M4 32 bits processor, and since templates will be limited to the 2 LSBs for Dilithium-2, we considered the $\mathbf{w}_0$ value as four independent bytes. We then perform a reverse analysis on the value of each byte value separately. It leads to a 256 partitions analysis of each byte value.

However, this simplification does not hold for both Dilithium-3 and Dilithium-5 because of the different values of the bounds. Even though the MSB is still filtered, the bounds for the second MSB don't hold anymore. For these versions, we would have to analyze the four bytes or we may have to choose a leakage model more appropriate.

Moreover, we selected the HW leakage model for attack generality, making 0 a good choice due to its sole representation in this HW class (also true for HW = 32).

Building the profiling data set. We must consider having as many occurrences as possible for each class of the leakage models. For our leakage model, each byte value has a significant representation when building the data set with random messages. To do so, we used the C reference implementation. We coded a script that performs Dilithium signatures for a given security parameter until there is a significant number of representatives for each class. In order to run the attack, we arbitrarily decided to collect 2 700 messages per class, to guarantee a minimal number of representative in each class. We then complete with random messages to have a data set of 700 000 messages. In the end, this number of traces might be oversized regarding the targeted implementation, but our objective is only to demonstrate the feasibility of the end-to-end attack. Collecting the necessary messages corresponding to the 256 partitions amounts to a total of 24 hours approximately. We also need 24 more hours to perform the corresponding acquisitions with the ChipWhisperer.

4.2.3 POIs selection for templates

The templates must be created using a reduced number of samples in each trace to lower the computation cost. To do so, we keep only a few samples empirically from the ANOVA distinguisher shows highest amplitude.

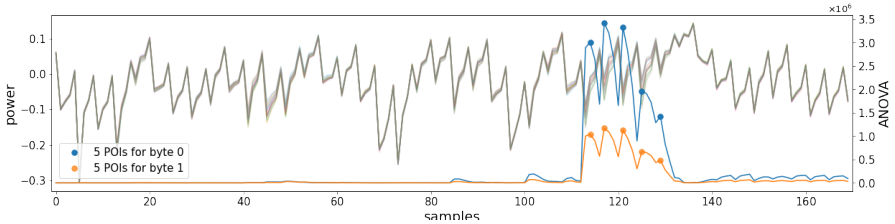

Figure 5: POIs selection for the Leakage model for the two MSBs.

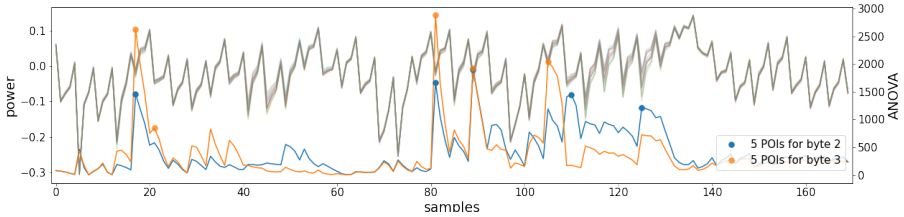

Figure 6: POIs selection for the Leakage model for the two LSBs.

Figure 5 and 6 illustrate our selection of the leakage peaks as Points of Interest to keep the best indices in our power traces for our leakage model. We have also plotted some power traces of the `Decompose` in the background, where we can see significant variability in multiple locations compared to the rest of the samples between different signature executions. In the end, we have kept 5 POIs for each byte. In Figure 6, we have plotted the results for the ANOVA statistical indicator, cited previously, for the four bytes. We can observe that they give essentially the same information. Clear leakage peaks can be seen from samples 0 to 250 at roughly the same position. Furthermore, we can see that they can be split in two categories, for the two MSBs and the two LSBs. This results will be useful later on for selecting the Points of Interest.

4.3 Matching Phase

We now validate our templates created using the POIs previously selected. For this purpose, we create a matching data set of 10 traces collected from the ChipWhisperer, which took about 1.4 seconds since we didn't try to optimize it. The data set is built with a targeted intermediate value fixed to $(\mathbf{w}_0)_{0,0} = 0$ for all traces. For our experiments, we do not consider cross-device template matching as we use the same target to build the profiling data set used for matching.

The results of our experiments are shown in Figures 7 and 8. For our leakage model, the best template score corresponds to our targeted value $(\mathbf{w}_0)_{0,0} = 0$. We also observe the evolution of the scores showing that our value is clearly distinguishable from the others right from the first trace prediction.

Divide and conquer $(\mathbf{w}_0)_{0,0}$ LSBs. The Figures 7 and 8 show that the template matching guesses correctly the first and the second LSB of $(\mathbf{w}_0)_{0,0} = 0$, and the results are shown for 10 traces. Furthermore, we can see that the corresponding score for the correct guess is clearly distinguishable from the rest. As stated previously, the score for the correct guess is already detectable with one trace. Still, surprisingly enough, the score does not seem to improve dramatically with the number of traces added. Maybe by adding artificial noise to the trace one could observe clear improvement of the score for the correct guess.

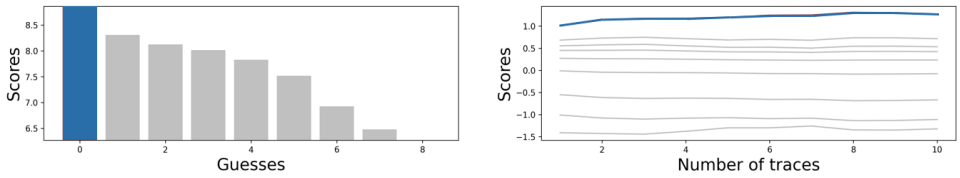

Figure 7: Matching value LSB 0.

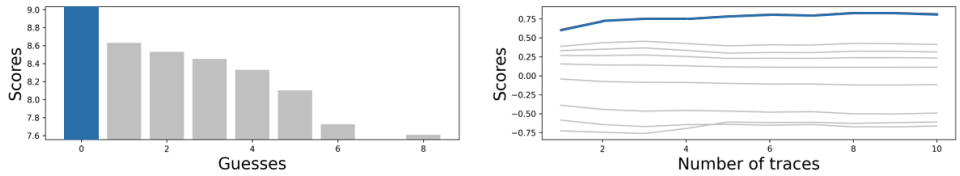

Figure 8: Matching value LSB 1.

Divide and conquer $(\mathbf{w_0})_{0,0}$ MSBs. For completeness and to show what a potential matching could be for the versions 3 and 5 of Dilithium, we decided to study also the MSB of the targeted value. From our leakage model, we know that without the filtering step only two partitions are required for the first MSB of $(\mathbf{w}_0)_{0,0}$ and four partitions for the second MSB in the case of Dilithium-2. Figure 9 show that the template matching guesses correctly the first and the second MSB of $(\mathbf{w}_0)_{0,0} = 0$.

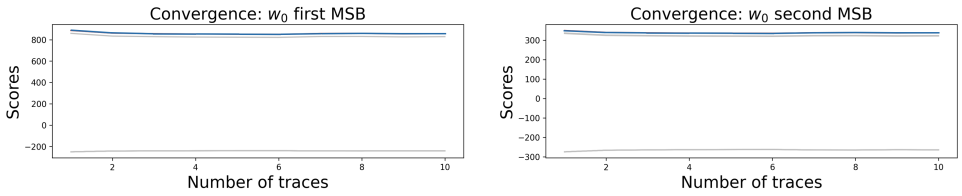

Figure 9: Matching HW MSB 0 and 1.

Even though the two MSBs are correctly matched, we can see that the scores of the second MSB for values 0 and 1 are really close to each other. It could be challenging to distinguish them from a weaker leakage or with more noise. This raises the importance of the filter step previously to the leakage assessment.

4.4 Results Interpretation on $(\mathbf{w_0})_{0,0}$

In our analysis of the $(\mathbf{w}_0)_{0,0} = 0$ coefficients, we found that it could be challenging to differentiate close values such as `00` and `01` or `FF` and `FE`. To speed up the attack, the implemented filter on the values increases the speed of the attack up to two times when focusing on the last two bytes, but this only applies to Dilithium-2. In our experiments, with over 10 000 000 signatures, the filter discarded approximately 69.3% of the $N = 1024$ coefficients, leaving on average only 314 coefficients that could potentially be equal to 0. Additionally, we evaluated the number of true $(\mathbf{w}_0)_{i,j} = 0$ detected by the filter and found that it was approximately 94.66%, which is close to the 95.4% theoretically expected from Subsection 3.2.

To test the false positives of our template on the first $\mathbf{w}_0$, i.e., when we wrongly predict a coefficient $(\mathbf{w}_0)_{0,0} = 0$, we decided to perform 100 000 signatures with 10 traces each time to complete the matching. In the end, we found that we have 67 false positives for this first coefficient. We also tested for false negatives on $(\mathbf{w}_0)_{0,0} = 0$ with our template, i.e when we wrongly predict a coefficient $(\mathbf{w}_0)_{0,0} \neq 0$. With the same number of traces as for the matching, we evaluated 50 000 messages with fixed intermediate value $(\mathbf{w}_0)_{0,0} = 0$.

We observe that we have 87 coefficients not detected. Combined with the values rejected by the filter, this amounts to acquiring more signatures before the attack.

Note that when we are in a deterministic setting, i.e., where all the parameters are fixed, we can obtain several traces for the same message. In this case, we can converge toward a more reliable trace and possibly reduce the number of false positives.

4.5 Resolution and Majority Vote

First, we give a method to get a candidate for $\mathbf{t}_0 - s_2$, and then we give our results on the majority vote to get the correct solution.

Having an implementation of the Dilithium-2 signing algorithm, we decided to proceed as described. For a given secret key, for a signature of a message, we can determine if there was a set of indices for which $(\mathbf{w}_0)_{i,j} = 0$ and if so, retrieve this (i, j). After 2.5 M signatures, we had all the $N = 256 \times 4 = 1024$ indices multiple times. With this set of coefficients, we randomly choose a set of N values, then with our false positive rate, we add a small error of e (in practice we used $e \in \{-1, 1\}$) to a coefficient with this probability. Finally, we construct the vector L and the matrix $\tilde{C}$ before testing if it is invertible and applying the corresponding resolution method. This resolution step takes under a second. Finally, we compare to the correct $\mathbf{t}_0 - s_2$.

We also tested the effect of this error on a given coefficient by solving the system with a single error on every coefficient between 0 and 255 for the 4 vectors. We observed that depending on the coefficient, the approximated $\mathbf{t}_0 \tilde{-} s_2$ can have between 0 and 256 values different from the correct $\mathbf{t}_0 - s_2$. We show in Figure 10 the number of indices that produces the according number of incorrect rounded coefficients. This number seems to be

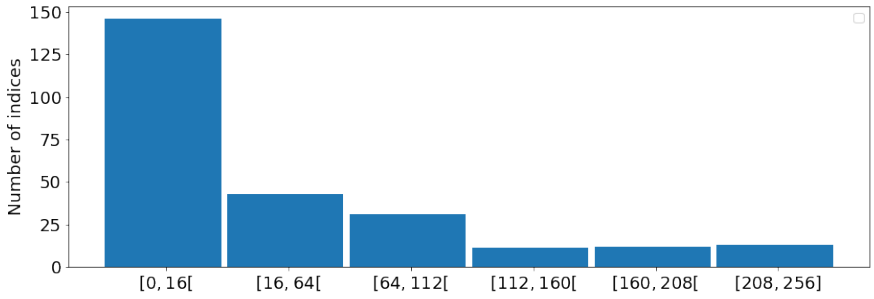

Figure 10: Number of wrongly rounded coefficients

related to the norm of the corresponding column in the matrix $\tilde{C}$, which could gives us beforehand a rough indication about the quality of our candidate.

On the other hand, we have tested the majority vote for when we have at most 12 wrongly rounded coefficients of the $\mathbf{t}_0 \tilde{-} s_2$ value, which happens for half of the positions in our tests. This method aims to make several calls to the `SOLVE` function with different equations to obtain multiple candidates and then apply the majority vote on these candidates. We tested the probability of success in this setting. For instance, we tested the probability of success for all the 5 possible combinations of candidates where $\mathbf{t}_0 \tilde{-} s_2$ has at most 12 wrong coefficients and found out that this probability is approximately 75%. This took around 6 days. Hence, testing all the possible combinations of candidates takes a lot of time and scaling this analysis with more candidates was not possible. To overcome this, we chose several combinations of random candidates instead of all the possible ones. In our example, we took 100 000 random combinations of 5 candidates where we have at most 12 wrong coefficients and got a success rate of approximately 76.12% which gives us a good estimate. This took around 15 seconds. For completeness, we decided to test

100 000 random combinations for a variable number of candidates to see the evolution of the success rate. We also decided to do the same analysis without bounding the number of errors of the candidates. The results are shown in Figure 11.

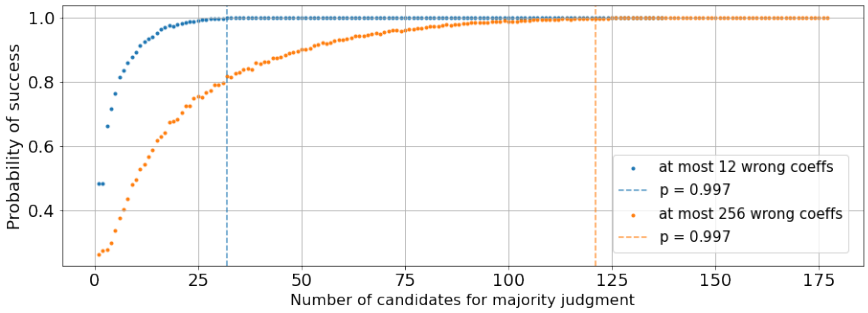

Figure 11: Evolution of the success probability with respect to the number of candidates

We can see that by bounding the number of errors to 12, which happens half of the time in our case we are able to converge sooner to the correct $\mathbf{t}_0 - s_2$, so by choosing carefully what indices are not certain before solving the system we are able to reduce the propagation of this error to a bounded number of coefficients. Then, by selecting those specific candidates we can limit the total number of candidates to 30 in our majority vote with still finding the correct $\mathbf{t}_0 - s_2$. However if one does not want or cannot do this selection beforehand, selecting 125 candidates for the majority vote has a very high probability of finding the correct $\mathbf{t}_0 - s_2$.

4.6 End-to-end attack and Modularity

So far, we confirmed that we could exploit the leakage to recover $(\mathbf{w}_0)_{0,0} = 0$. For the sake of simplicity, we intend to apply the same template to the rest of the coefficients to distinguish if we have 0 or not. With few samples at our disposition, we do not capture the totality of the decompositions, so we will need to make several acquisitions. For instance, as in our case with the ChipWhisperer, we can at most capture 95 coefficients on the $N = 256 \times 4 = 1024$ needed. Therefore, we must do at least 11 set of acquisitions per message to capture the entire coefficients. Since we have a good superposition of the first 95 coefficients, as shown in Figure 12, we can assume that we can apply the same template for the rest of the coefficients. We validated our attack on random coefficients of the first

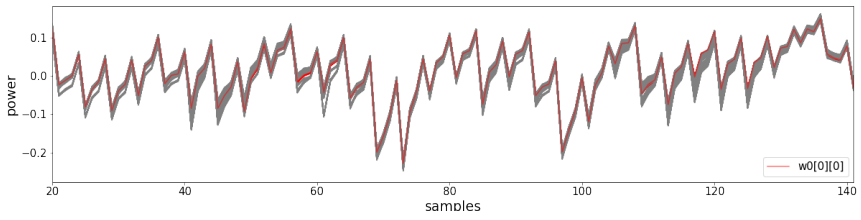

Figure 12: Superposition of the first 95 `Decompose`.

acquisition without any adjustment except shifting the POIs. Moreover, in practice, by testing on all the 95 coefficients, we got the same false positive and false negative rate, therefore confirming that targeting one `Decompose` among the $N = n \times k$ is sufficient to attack all of them. For some coefficients, we may have a loss of precision due to a signal acquisition or micro-architectural effect, which could increase the number of false positives,

i.e., when we wrongly predict a coefficient $\mathbf{w}_0 = 0$. In this case, if we don't get the correct value, we can mount a template attack per coefficient.

We now describe a scenario of an end-to-end attack. Here are the detailed steps.

- Collect suitable messages. The required time for this step can be optimized using PC implementation.
- Make the acquisitions on the profiling device.
- Select the POIs and build a template on the first coefficient of the first vector of $\mathbf{w}_0$.
- Make the acquisitions on the targeted device.
- Apply the filtering procedure, in order to exclude signatures for which $\mathbf{w}_0$ is different from 0 with high probability.
- Apply the same template on all $\mathbf{w}_0$ coefficients without any adjustment except shifting the POIs.
- After collecting the corresponding signatures for which $(\mathbf{w}_0)_{i,j}$ equal to 0 with high probability for each (i, j), build a system of equations corresponding to the data collected.
- Use the least square method to get a valid solution candidate.
- Apply the majority vote to get the correct solution.

Depending on the setup used, one can focus on a different model according to the corresponding leakage assessment or the version of Dilithium. Also, depending on the equipment at disposition, we can choose to use the filter or not. One can limit the number of acquisitions to perform by selecting which coefficients to focus on beforehand. The next step in the process, given a set of positions where $\mathbf{w}_0 = 0$ and corresponding signatures, is to use the Algorithm 4 to perform the Least Squares Method (LSM) resolution to find a solution candidate of $\mathbf{t}_0 - s_2$. We must also ensure that the values are signed for the LSM to solve the system correctly. With our implementation in Sage, the whole Algorithm 4 takes a little more than 2 minutes. We use the majority vote technique to find simply and efficiently the accurate $\mathbf{t}_0 - s_2$, as shown previously in 3.4. This process is repeated several times with different equations. For example, during our experiments, we repeated it 3 times, for a duration of 78 seconds each, and a majority vote was taken on each position (i, j) to find the correct $\mathbf{t}_0 - s_2$. We did not use the framework presented in [DDGR20] given that no implementation of a solver using the LWE with side information was available. Still, it could be interesting to study the performances of this solution compared to ours. Finally, we can recover the secret key s_1. To do so, we first convert the matrix A into the normal domain using the predefined functions of Dilithium. Then we solve Equation (3.1). We can verify if we have the correct s_1 by checking if the values are in $[-\eta, \eta]$.

5 Discussions and Countermeasures

In this paper, we have practically shown that exploiting the leakage of $\mathbf{w}$ during the decomposition can lead to the full recovery of the secret key s_1, which is sufficient to generate valid signatures. In their previous work, Migliore *et al.* [MGTF19] showed a masked version of Dilithium that does not mask the `Decompose` in line 8 of Algorithm 2. However, Azouaoui *et al.* recommended that it is an important operation to mask.

In the absence of any countermeasure, implementations using signed values may magnify some exploitable leakage, especially since the signed representation is the most efficient to implement on an ARM Cortex-M4 [GKS20]. For instance, the subtraction $\mathbf{w}_0 - c\,s_2$ with positive coefficients of $\mathbf{w}_0$ and $c\,s_2$ would suddenly generate a negative number when, for these specific coefficients, $\mathbf{w}_0 < c\,s_2$. This strong Hamming distance may give some hints to the attacker to classify very small values of $\mathbf{w}_0$. One way to prevent attacks is to

minimize potentially vulnerable operations in the algorithm. The computation $\mathbf{w}_0 - c\,s_2$ is a possible source of leakage, but it can be eliminated using the standard method to compute the $\mathbf{r}_0$ value, with some overhead due to the `Decompose` operation.

There exist simple known countermeasures. For example, shuffling and masking can easily make this side-channel attack impossible in practice when correctly implemented.

Shuffling. Shuffling the manipulation of coefficients for all identified sensitive values $\mathbf{w}_0$ during `Decompose` and during the calculation of $\mathbf{r}_0$ may render the attack very complex. Indeed during the matching phase, the attacker needs to link the observation that one coefficient of $\mathbf{w}_0$ equals zero, with the related indices i and j for this zero coefficient. This operation is needed to mount the correct system of equations and thus for the attack's success. The calculation of $\mathbf{r}_0$ and `Decompose` do not need to process individual coefficients in any specific order.

Secret sharing. Sharing sensitive variable $\mathbf{w}_0$ in d additive shares, for `Decompose` and for the subtraction $\mathbf{w}_0 - c\,s_2$ would avoid any $d-1$-th order attack. Detecting when a coefficient $(\mathbf{w}_0)_{i,j} = 0$ using leakage from d multiple shares exponentially increases the number of traces needed with increasing masking order. For instance, an implementation of `SecDecompose`, which is an efficient secure implementation of `Decompose`, is already discussed by Azouaoui *et al.* [ABC+22]. The arithmetic sharing must be kept until the complete computation of $\mathbf{r}_0$.

For an alternative exploitation of the leakage, one can notice that with the same setup and under the same hypothesis, the attack path described here could also be exploited through Machine Learning side-channel attack [MPP16] instead of a template attack.

For future work, we want to investigate if there are other sensible operations that we did not identify yet as well as evaluate the leakage of the $\mathbf{w}_0 - c\,s_2$ operation.

References

[AASA+22] Gorjan Alagic, Jacob Alperin-Sheriff, Daniel Apon, David Cooper, Quynh Dang, Carl Miller, Dustin Moody, Rene Peralta, Ray Perlner, Angela Robinson, Daniel Smith-Tone, and Yi-Kai Liu. Status report on the third round of the nist post-quantum cryptography standardization process, 2022-07-05 2022.

[ABC+22] Melissa Azouaoui, Olivier Bronchain, Gaëtan Cassiers, Clément Hoffmann, Yulia Kuzovkova, Joost Renes, Markus Schönauer, Tobias Schneider, François-Xavier Standaert, and Christine van Vredendaal. Leveling Dilithium against leakage: Revisited sensitivity analysis and improved implementations. *IACR Cryptol. ePrint Arch.*, page 1406, 2022.

[BDE+18] Jonathan Bootle, Claire Delaplace, Thomas Espitau, Pierre-Alain Fouque, and Mehdi Tibouchi. LWE without modular reduction and improved side-channel attacks against BLISS. In Thomas Peyrin and Steven Galbraith, editors, *Advances in Cryptology – ASIACRYPT 2018, Part I*, volume 11272 of *Lecture Notes in Computer Science*, pages 494–524, Brisbane, Queensland, Australia, December 2–6, 2018. Springer, Heidelberg, Germany.

[BHK+19] Daniel J. Bernstein, Andreas Hülsing, Stefan Kölbl, Ruben Niederhagen, Joost Rijneveld, and Peter Schwabe. The SPHINCS+ signature framework. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019: 26th Conference on Computer and Communications Security*, pages 2129–2146. ACM Press, November 11–15, 2019.

[BJS11] Mokhtar S Bazaraa, John J Jarvis, and Hanif D Sherali. *Linear programming and network flows*. John Wiley & Sons, 2011.

[BP18] Leon Groot Bruinderink and Peter Pessl. Differential fault attacks on deterministic lattice signatures. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2018(3):21–43, 2018. https://tches.iacr.org/index.php/TCHES/article/view/7267.

[CK13] Omar Choudary and Markus G. Kuhn. Efficient template attacks. Cryptology ePrint Archive, Paper 2013/770, 2013. https://eprint.iacr.org/2013/770.

[CRR03] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template attacks. In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2002*, volume 2523 of *Lecture Notes in Computer Science*, pages 13–28, Redwood Shores, CA, USA, August 13–15, 2003. Springer, Heidelberg, Germany.

[DDGR20] Dana Dachman-Soled, Léo Ducas, Huijing Gong, and Mélissa Rossi. LWE with side information: Attacks and concrete security estimation. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020, Part II*, volume 12171 of *Lecture Notes in Computer Science*, pages 329–358, Santa Barbara, CA, USA, August 17–21, 2020. Springer, Heidelberg, Germany.

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Trans. Inf. Theory*, 22(6):644–654, 1976.

[DKL+22] Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, Gregor Seiler, Peter Schwabe, and Damien Stehlé. PQ-CRYSTALS, Dilithium. https://github.com/pq-crystals/dilithium, 2022. GitHub repository. Accessed: 2022-12-15.

[FDK20] Apostolos P. Fournaris, Charis Dimopoulos, and Odysseas G. Koufopavlou. Profiling dilithium digital signature traces for correlation differential side channel attacks. In Alex Orailoglu, Matthias Jung, and Marc Reichenbach, editors, *Embedded Computer Systems: Architectures, Modeling, and Simulation - 20th International Conference, SAMOS 2020, Samos, Greece, July 5-9, 2020, Proceedings*, volume 12471 of *Lecture Notes in Computer Science*, pages 281–294. Springer, 2020.

[GKS20] Denisa O. C. Greconici, Matthias J. Kannwischer, and Daan Sprenkels. Compact Dilithium implementations on cortex-M3 and cortex-M4. Cryptology ePrint Archive, Report 2020/1278, 2020. https://eprint.iacr.org/2020/1278.

[HMvdW+20] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[KJJ99] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael Wiener, editor, *Advances in Cryptology — CRYPTO' 99*, pages 388–397, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[KLH+20] Il-Ju Kim, Tae-Ho Lee, Jaeseung Han, Bo-Yeon Sim, and Dong-Guk Han. Novel single-trace ML profiling attacks on NIST 3 round candidate Dilithium. Cryptology ePrint Archive, Report 2020/1383, 2020. https://eprint.iacr.org/2020/1383.

[KLS18] Eike Kiltz, Vadim Lyubashevsky, and Christian Schaffner. A concrete treatment of Fiat-Shamir signatures in the quantum random-oracle model. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018, Part III*, volume 10822 of *Lecture Notes in Computer Science*, pages 552–586, Tel Aviv, Israel, April 29 – May 3, 2018. Springer, Heidelberg, Germany.

[KSSW] Matthias J. Kannwischer, Peter Schwabe, Douglas Stebila, and Thom Wiggers. Pqclean. https://github.com/PQClean/PQClean. Accessed: 2022-12-15.

[LDK+22] Vadim Lyubashevsky, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Peter Schwabe, Gregor Seiler, Damien Stehlé, and Shi Bai. CRYSTALS-DILITHIUM. Technical report, National Institute of Standards and Technology, 2022. available at https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022.

[Lyu09] Vadim Lyubashevsky. Fiat-Shamir with aborts: Applications to lattice and factoring-based signatures. In Mitsuru Matsui, editor, *Advances in Cryptology – ASIACRYPT 2009*, volume 5912 of *Lecture Notes in Computer Science*, pages 598–616, Tokyo, Japan, December 6–10, 2009. Springer, Heidelberg, Germany.

[LZS+21] Yuejun Liu, Yongbin Zhou, Shuo Sun, Tianyu Wang, Rui Zhang, and Jingdian Ming. On the security of lattice-based Fiat-Shamir signatures in the presence of randomness leakage. *IEEE Trans. Inf. Forensics Secur.*, 16:1868–1879, 2021.

[MGTF19] Vincent Migliore, Benoît Gérard, Mehdi Tibouchi, and Pierre-Alain Fouque. Masking Dilithium - efficient implementation and side-channel evaluation. In Robert H. Deng, Valérie Gauthier-Umaña, Martín Ochoa, and Moti Yung, editors, *ACNS 19: 17th International Conference on Applied Cryptography and Network Security*, volume 11464 of *Lecture Notes in Computer Science*, pages 344–362, Bogota, Colombia, June 5–7, 2019. Springer, Heidelberg, Germany.

[MOS09] Stefan Mangard, Elisabeth Oswald, and Francois-Xavier Standaert. One for all - all for one: Unifying standard dpa attacks. Cryptology ePrint Archive, Paper 2009/449, 2009. https://eprint.iacr.org/2009/449.

[MPP16] Houssem Maghrebi, Thibault Portigliatti, and Emmanuel Prouff. Breaking cryptographic implementations using deep learning techniques. Cryptology ePrint Archive, Report 2016/921, 2016. https://eprint.iacr.org/2016/921.

[MUTS22] Soundes Marzougui, Vincent Ulitzsch, Mehdi Tibouchi, and Jean-Pierre Seifert. Profiling side-channel attacks on Dilithium: A small bit-fiddling

leak breaks it all. Cryptology ePrint Archive, Report 2022/106, 2022. https://eprint.iacr.org/2022/106.

[nis] National institutes of standards and technology, post-quantum cryptography. https://csrc.nist.gov/Projects/post-quantum-cryptography. Accessed: 2022-12-15.

[OC14] Colin O'Flynn and Zhizhang David Chen. Chipwhisperer: An open-source platform for hardware embedded security research. In *International Workshop on Constructive Side-Channel Analysis and Secure Design*, 2014.

[QLZ+23] Zehua Qiao, Yuejun Liu, Yongbin Zhou, Jingdian Ming, Chengbin Jin, and Huizhong Li. Practical public template attack attacks on CRYSTALS-Dilithium with randomness leakages. *IEEE Trans. Inf. Forensics Secur.*, 18:1–14, 2023.

[RJH+18] Prasanna Ravi, Mahabir Prasad Jhanwar, James Howe, Anupam Chattopadhyay, and Shivam Bhasin. Side-channel assisted existential forgery attack on Dilithium - A NIST PQC candidate. Cryptology ePrint Archive, Report 2018/821, 2018. https://eprint.iacr.org/2018/821.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, 1978.

[SBN+21] Deepraj Soni, Kanad Basu, Mohammed Nabeel, Najwa Aaraj, Marcos Manzano, and Ramesh Karri. *FALCON*, pages 31–41. Springer International Publishing, Cham, 2021.

[Sch91] Claus-Peter Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174, January 1991.

[Sho94] Peter W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *35th Annual Symposium on Foundations of Computer Science*, pages 124–134, Santa Fe, NM, USA, November 20–22, 1994. IEEE Computer Society Press.

[YJ21] Wei Yang and Anni Jia. Side-Channel Leakage Detection with One-Way Analysis of Variance. *Security and Communication Networks*, 2021:6614702, March 2021. Publisher: Hindawi.