Finding and Protecting the Weakest Link
On Side-Channel Attacks on Masked ML-DSA

Julius Hermelink[1], Kai-Chun Ning[1], and Richard Petri[1]

Max Planck Institute for Security and Privacy, Bochum, Germany,
firstname.lastname@mpi-sp.org

Abstract. NIST has standardized ML-KEM and ML-DSA as replacements for pre-quantum key exchanges and digital signatures. Both schemes have already seen analysis with respect to side-channels, and first fully masked implementations of ML-DSA have been published. Previous attacks have focused on unprotected implementations or assumed only hiding countermeasures to be in-place. Thus, in contrast to ML-KEM, the threat of side-channel attacks for protected implementations of ML-DSA is mostly unclear.

In this work, we analyze the side-channel vulnerability of masked ML-DSA implementations. We first systematically assess the vulnerability of several potential points of attacks in different leakage models using information theory. Then, we explain how an adversary could launch first, second, and higher-order attacks using a recently presented framework for side-channel information in lattice-based schemes. In this context, we propose a filtering technique that allows the framework to solve for the secret key from a large number of hints; this had previously been prevented by numerical instabilities. We simulate the presented attacks and discuss the relation to the information-theoretic analysis.

Finally, we carry out relevant attacks on physical devices, discuss recent masked implementations, and instantiate a countermeasure against the most threatening attacks. The countermeasure mitigates the attacks with the highest noise-tolerance while having very little overhead. The results on the physical devices validate our simulations.

Keywords: Dilithium · ML-DSA · SCA · Countermeasures · Belief Propagation

1 Introduction

In anticipation of the advent of sufficiently powerful quantum computers which threaten existing asymmetric cryptography, the National Institute of Standards and Technology (NIST) launched a standardization process of Post-Quantum Cryptography in 2016, with the goal of soliciting, evaluating, and eventually standardizing cryptographic schemes resistant to both classic and quantum attacks [38]. As this standardization process advanced to the next stage in 2022, NIST selected 2 lattice-based schemes Kyber [5] and Dilithium [17] as the first

quantum-resistant standards of key encapsulation mechanism (KEM) and digital signatures, and renamed them to Module-Lattice-based Key-Encapsulation Mechanism (ML-KEM) [41] and Module-Lattice-based Digital Signature Algorithm (ML-DSA) [40], respectively.

Because of their computational efficiency, ML-KEM and ML-DSA are particularly suitable to be deployed onto embedded systems, where computing power and resources are often restricted. Since embedded systems are more likely to be used in environments where an attacker has physical access to the device, NIST specifically listed resistance to side channel attacks as one of the evaluation criteria of the standardization process [38]. In light of this, side channel analysis of their implementation has become an even more prominent area of research in recent years [48].

Several previous attacks on ML-DSA and ML-KEM targeted the secret key in Number Theoretic Transform (NTT) domain [45,43,10,21]. In general, targeting the NTT in lattice-based schemes has two main advantages: the structure of the NTT may be exploited using Belief Propagation (BP) [45,43,21], and it allows recovering the full key from very few recovered coefficients using lattice reduction (see, e.g., [2,21,46]). The coefficients of $\mathbf{y}$ have been another valuable target as shown in, e.g., [51,7,47], which gives information on $\mathbf{s}_1$ as the equation

$$\mathbf{z} = \mathbf{y} + c\mathbf{s}_1 \tag{1}$$

for public $\mathbf{z}, c$ holds.

Bronchain et al. [7] give a BP instantiation that allows recovering the secret key from various types of leakages or fault attacks on $\mathbf{y}$ and $\mathbf{x} = c\mathbf{s}_1$. In contrast to previous works, their approach is generic and aims at exploiting all available information under high noise levels. The authors also show how hiding countermeasures can be circumvented. Another recent approach that seems to be relatively noise tolerant has been presented in [47]. Instead of using a soft-analytic technique, the authors show that the small range of $\mathbf{y}$ under the knowledge of $\mathbf{z}$ makes identifying noise-free equations from side-channel information practical. Thereby, they overcome some challenges posed by using BP. However, it is unclear whether these approaches apply to masked implementations.

Previous attacks on $\mathbf{y}$ do not yet target masked implementations and the relation of previous solvers that could apply is not yet clear. Also, the main study in this area [7] has not been validated on a physical device. Further, previous solvers have either been observed to either fail to solve from a large number of equations [47,26], which is required to target a masked implementation under realistic noise conditions, or it is unclear whether they apply to masked implementations in the first place. In addition, $\mathbf{y}$ is different for every signature; an adversary can therefore not target $\mathbf{y}$ over multiple traces to, e.g., fully recover its coefficients. Moreover, in current masked implementations, $\mathbf{y}$ is sampled in Boolean masking, converted to arithmetic masking, and then added to $c\mathbf{s}_1$. Therefore, $\mathbf{y}$ may be targeted at several locations in different representations, and it is unclear which is favorable for an adversary. In short: The question of how to target a masked ML-DSA implementation and whether attacks on $\mathbf{y}$ can even be carried out in such a setting is still open.

Our contribution. In this work, we aim at answering the question of how vulnerable ML-DSA is to leakage that arises in masked implementations when targeting $\mathbf{y}$. First, we investigate previous attempts to dealing with leakage in ML-DSA and provide a conceptual comparison. This includes reproducing the closed-source work of [7][1] using the solver of [26]. We also assess whether previous approaches could also apply to masked implementations.

Secondly, we systematically study different types of leakages of $\mathbf{y}$. We discuss possible types of leakage on $\mathbf{y}$ that can occur in a masked implementation of ML-DSA based on the example of the proposals of [13,11]. Then, we provide an information theoretic analysis that examines several leakage models in the worst case. For low to moderate noise levels, our analysis shows that $\mathbf{y}$ is particularly vulnerable when processed as Boolean sharing; this is in-line with the information-theoretic analysis of [35] for masking in symmetric schemes. In this setting, strong first-order leakage is more likely, and second-order leakage provides far more information. In contrast, targeting an arithmetically shared $\mathbf{y}$ allows for recovering much less information and has lower chances of accidental leakages that can be exploited efficiently. For higher noise levels, the leaked amount of information does not differ *as much* between masking types (c.f., [35]), but arithmetic representation still leak less information, and our attacks require far more computational resources to exploit leakage from arithmetic masking. Further, we discuss the effects of canonical and central reductions; while our results do not contradict [50], we show that using a central reduction for $\mathbf{y}$ leaks only very little extra information.

Thirdly, we show how an adversary may deal with leakage in a higher-order attack on several shares of $\mathbf{y}$. We show how the solver of [26] may be instantiated to target a masked implementation. Further, as already noted in [47] and [26], BP quickly becomes numerically instable if it has to handle too many equations. We present a filtering technique that may mitigate these problems and allows obtaining and processing useful information for very high noise levels. This technique applies more generally to the solver of [26]. Further, we discuss several other techniques applicable to BP that did not yield improvements in our specific case but could improve different attacks.

We evaluate our attacks against a masked implementation by simulation in various leakage models, and discuss the relation to our information-theoretic analysis; our results show that for a noise standard deviation of $\sigma = 1.0$ (SNR 4.5) in the noisy Hamming weight (HW) model, roughly 500 traces allow for full recovery of $\mathbf{s}_1$ in a second order attack against Boolean masking; for $\sigma = 2.0$ and $\sigma = 3.0$ (SNR 1.11 and SNR 0.50) we recovered the secret key with 4000[2] and 5000 traces, respectively. If individual bits of $\mathbf{y}$ leak with a success rate of just 2 percentage points better than guessing, an adversary can fully recover the secret from about 400 traces.

Finally, we carry out our attack against physical devices and instantiate a countermeasure that mitigates the most devastating attacks on $\mathbf{y}$. We target the

[1] To the best of our knowledge their implementation is not publicly available.
[2] Fewer traces might already allow for recovery.

masked implementation of [13,11] on two different microcontrollers to validate our results on physical devices; we recover the secret key from first-order HW leakage, first-order bit leakage, and second-order HW leakage. The first-order attack requires about 300 to 400 traces to recover the secret; the second-order attack requires about 700 traces. We note that in the case of Boolean masking, targeting the 8 Least Significant Bits (LSBs) is sufficient to fully recover the secret key, greatly reducing the required computational resources.

A higher Boolean masking order (compared to the arithmetic one) has already been briefly mentioned in [3] in the context that [35] shows that arithmetic masking to be more resilient to low-noise leakage. In the case of masked ML-DSA, our work suggests that such a countermeasure is effective for higher noise levels. Our evaluations show that doubling the Boolean masking order has an overhead of only 33% in the implementation of [11]. We also provide the first performance evaluations of the implementation of [13,11] on a microcontroller.

In summary: Our work provides a systematic analysis of the vulnerability of masked ML-DSA implementations with a focus on leakage on $\mathbf{y}$. We explain how and where an adversary may target ML-DSA, and evaluate how well such leakage may be exploited based on information theory, simulations, and attacks on physical devices. All our resources are open-source[3]. To the best of our knowledge, we provide the first attacks against masked ML-DSA implementations on a physical device.

Implications and practical recommendations. We draw several conclusions from our results: First, we find further evidence that targeting $\mathbf{y}$ allows for highly noise-tolerant attacks. Targeting only the HW of the LSBs with our filtering technique, this statement extends to Boolean masked $\mathbf{y}$: a second-order attack targeting the HW of both shares is still comparably noise-tolerant and small biases on single bits can in many cases be exploited efficiently by an adversary.

Secondly, the arithmetically shared $\mathbf{y}$ leaks far less information, especially for low-to-moderate noise levels (c.f., [35]). For higher noise levels, exploiting the information requires much more computational resources in our attacks. When using arithmetic masking, the difference between centrally and canonically reduced sharings of $\mathbf{y}$ is very small. Our results do not contradict [50] but show a more complex scenario. Central reductions, i.e., using signed integers, are more favorable with respect to performance, and our results show that side-channel attacks are most likely not a factor of high importance in this specific case.

Thirdly, our evaluations also show the need for optimized protected implementations of ML-DSA for embedded devices. The works of [3,13,11] propose the necessary gadgets and prove their security. However, our results suggest that the latest implementations of [13,11] do not achieve performances on a microcontroller that would be necessary for most real-world applications. Furthermore, a thorough comparison of masked implementations of ML-DSA that also considers, e.g., the amount of required randomness is still missing.

[3] Available under `https://github.com/juliusjh/mmldsa_attack`.

2 Preliminaries

We first give an overview over the used notation, Dilithium/ML-DSA, and previous attacks on them. Then, we reiterate the solver of [26], which we employ to recover the secret key from side-channel information. Finally, we give some basic information-theoretic definitions, which are relevant for our analysis in Section 3.

Notation. Throughout the paper, $\mathbf{y}, \mathbf{z}, \mathbf{s}_1$, and c denote the respective intermediate Dilithium/ML-DSA variables as defined in Section 2.1. By $\mathbf{x}$ we denote $c\mathbf{s}_1$ and y, z, x denote coefficients of components of the respective vector of polynomials. In general, vectors are denoted by bold letters. To reduce the number of notations we introduce, we do not use any notation to distinguish between an (unknown) value and the associated random variable whenever the meaning can be clearly inferred from context; otherwise, random variables are denoted by uppercase letters.

For $q, r \in \mathbb{Z}$, we denote the central reduction of r modulo q as $r \bmod^{\pm} q$, which is the unique integer r' in the range $-\frac{q}{2} < r' \leq \frac{q}{2}$ for an even q (and $-\frac{q-1}{2} \leq r' \leq \frac{q-1}{2}$ for an odd q) such that $r' \equiv r \mod q$. Similarly, we denote the canonical reduction of r modulo q as $r \bmod^{+} q$, which is the unique integer r' in the range $0 \leq r' < q$ such that $r' \equiv r \bmod q$. Let R_q be the polynomial ring $\mathbb{Z}_q[X]/(X^n + 1)$; we denote as S_η the set of small polynomials in R_q which only have coefficients whose absolute value is smaller than or equal to η after central reduction. In other words: $S_\eta := \{w = c_0 + c_1 X + \cdots + c_{n-1} X^{n-1} \mid 0 \leq i < n, |c_i \bmod^{\pm} q| \leq \eta\}$.

By L, we denote noise-free leakage in a certain representation. For example, $L_{\text{can.}}$ denotes the HW leakage on a canonically reduced value, while $L_{\text{cen.}}$ denotes the HW leakage on a centrally reduced value. If L is applied to a vector, i.e., to multiple shares, we mean pointwise application. We call an attack t-order if its feasibility depends on targeting t intermediates, even if we evaluate under the assumption that the adversary targets multiple intermediates/coefficients in a single trace (but only t shares/values for each unshared coefficient).

2.1 ML-DSA

ML-DSA is a signature scheme based on two hard problems: Module Learning with Errors (MLWE) and Module Short Integer Solution (MSIS) [18,30]. The scheme operates with matrices and vectors with polynomial entries in $R_q = \mathbb{Z}_q[X]/(X^n + 1)$, where $n = 256$ and $q = 2^{23} - 2^{13} + 1$. With security parameters $(d, \tau, \gamma_1, k, l, \eta)$ set, the signer generates a key pair by first randomly sampling a $k \times l$ matrix $\mathbf{A}$ and 2 vectors $\mathbf{s}_1 \in S_\eta^l$ and $\mathbf{s}_2 \in S_\eta^k$. $\mathbf{s}_1$ and $\mathbf{s}_2$ are short vectors whose entries are polynomials with coefficients uniformly random in $[-\eta, \eta]$. Subsequently, $t = \mathbf{A} \cdot \mathbf{s}_1 + \mathbf{s}_2 \in R_q^k$ is computed and split into 2 polynomials $\mathbf{t}_0$ and $\mathbf{t}_1$ by depositing the lower and higher bits of each coefficient of the polynomials in $\mathbf{t}$ into $\mathbf{t}_0$ and $\mathbf{t}_1$, respectively (see function Power2Round in [18]). The public key is then $(\mathbf{A}, \mathbf{t}_1)$ and the secret key $(\mathbf{s}_1, \mathbf{s}_2, \mathbf{t}_0)$. Note that while $\mathbf{t}_0$ is part of

the secret key and is not published, the security reduction of ML-DSA assumes that it is public [18], and a recent work demonstrates that an attacker can in fact recover $\mathbf{t}_0$ from less than 500 000 signatures with a desktop computer [42].

The signing algorithm of ML-DSA, see Algorithm 1, is based on the "Fiat-Shamir with Aborts" paradigm [34], where signatures are repeatedly generated and rejected until one signature satisfies a set of conditions. To sign a message, a masking[4] vector $\mathbf{y} \in R_q^l$ whose entries are polynomials with coefficients uniformly random in $(-\gamma_1, \gamma_1]$ is sampled. The signer then computes $\mathbf{w} = \mathbf{A} \cdot \mathbf{y}$ and split it into $\mathbf{w}_1$ and $\mathbf{w}_0$ (see functions Decompose, HighBits and LowBits in [18]). The message and $\mathbf{w}_1$ are then passed to a hash function H to produce the challenge c, which is interpreted as a polynomial in R_q. By design, c has exactly τ coefficients that are ± 1, and all of its remaining coefficients are zero. The signature candidate $\mathbf{z}$ is then computed as $\mathbf{z} = \mathbf{y} + c\mathbf{s}_1$. To avoid a dependency of $\mathbf{z}$ on the secret key, $\mathbf{z}$ is checked against 2 conditions to ensure it does not leak any information about $\mathbf{s}_1$ and $\mathbf{s}_2$. If any condition fails, the signature candidate is discarded and a new signing attempt starts. The parameters of ML-DSA are chosen such that the expected number of signing iterations is between 3.85 and 5.1 [40].

The masking vector $\mathbf{y}$ must not be used to sign more than one message, and must have sufficient entropy. To avoid flaws in the RNG, the "hedged" sampling of $\mathbf{y}$, which uses both the fresh randomness from RNG, and randomness derived pseudo-randomly from the message being signed, is the default. The fresh randomness can help mitigate SCA while the randomness derived from the message protects against scenarios where the RNG is compromised. The ML-DSA standard also allows an optional deterministic sampling, which only utilizes the randomness derived from the message being signed, for platforms where SCA is not a concern. Nevertheless, as we will see in the physical experiments described in Section 5, even with the hedged sampling, care must be taken in order to prevent leakage on $\mathbf{y}$. In this work, we follow the ML-DSA standard and focus on the hedged approach. The hedged approach is also the main difference between ML-DSA and Dilithium [18], which ML-DSA is based upon; Dilithium offers only a fully randomized sampling of $\mathbf{y}$, or deterministically generation of $\mathbf{y}$ based on the randomness derived from the message. Note that Algorithm 1 does not show key generation and verification; for a full description, we refer to [18,19,39].

2.2 Previous relevant attacks on ML-DSA

The attack surface of ML-DSA consists of the key generation and signing as verification only makes use of public information. In this work, we focus on attacks targeting $\mathbf{y}$ in masked implementations. We first provide an overview over previous attacks against unmasked implementations and previous solutions for recovering $\mathbf{s}_1$ from information on $\mathbf{y}$ that could apply in a masked setting as well.

[4] In the sense that it masks the sensitive $c\mathbf{s}_1$, and has nothing to do with masking the implementation.

Algorithm 1 Simplified ML-DSA Signing Algorithm

procedure SIGN($\mathbf{s}_1 \in S_\eta^l, \mathbf{s}_2 \in S_\eta^k, \mathtt{m}$)
  $\mathbf{z} \leftarrow \perp$
  while $\mathbf{z} = \perp$ **do**
    $\mathbf{y} \xleftarrow{\text{R}} S_{\gamma_1 - 1}^l$
    $\mathbf{w} \leftarrow \mathbf{A}\mathbf{y}$
    $\mathbf{w}_1, \mathbf{w}_0 \leftarrow \text{Decompose}(\mathbf{w}, 2\gamma_2)$
    $c \leftarrow \text{H}(\mathtt{m} \parallel \mathbf{w}_1)$
    $\mathbf{z} \leftarrow \mathbf{y} + c\mathbf{s}_1$
    $\mathbf{r} \leftarrow \mathbf{w}_0 - c\mathbf{s}_2$
    if RejectionSamplingCheck($\mathbf{z}, \mathbf{r}, \gamma_1, \gamma_2, \beta$) **then**
      $\mathbf{z} \leftarrow \perp$
    end if
  end while
  return signature $\leftarrow (\mathbf{z}, c)$
end procedure

Targeting the NTT. The NTT has been targeted in several previous attacks. In attacks against ML-KEM, BP has proven to be valuable tool [45,43,21]; these attacks can be expected to apply in a similar fashion but with reduced noise tolerance due to the larger modulus. The authors of [22] use a neural network to target the NTT in Diltihium and recover $\mathbf{s}_1$. In contrast to previous attacks, they only target the input, i.e., $\mathbf{s}_1$ in normal domain.

Targeting the NTT secret has several advantages: The computation of an NTT offers many intermediate values that an adversary can target. These intermediates are all arithmetically related, which can be exploited using, e.g., BP. Moreover, the secret has small coefficients and the NTT is linear; this makes recovering the full secret from partial information a lattice problem. This was first noted in [2], and the authors of [21] recover the full key from a partially recovered key in NTT domain using lattice reduction. The work of [33] provides further insights, and [50] proposes to exploit this technique in combination with their Correlation Power Analysis (CPA).

Targeting (zero) coefficients of $\mathbf{y}$. Another valuable target is $\mathbf{y}$; it is directly connected to $\mathbf{s}_1$ as

$$\mathbf{z} = \mathbf{y} + c\mathbf{s}_1 \tag{2}$$

holds and $c, \mathbf{z}$ are known for released signatures.

The work of [51] shows that an adversary may identify zero coefficients of $\mathbf{y}$ using a Side Channel Attack (SCA). Thereby, the adversary obtains a (potentially noisy) linear equation on $\mathbf{s}_1$. They propose a template attack [9] using a neural network that targets the unpacking of $\mathbf{y}$ and uses Integer Linear Programming (ILP) to recover $\mathbf{s}_1$; to recover the secret key they require more than 700k signatures. It should be noted that BP has empirically given better results than ILP in similar problems in ML-KEM [44,25].

An advanced approach based on a similar idea has been proposed in [47]: The authors propose to identify noise-free equations in the set of all equations derived from physical traces. Each equation is in the form of $c_i\mathbf{s}_1 = \mathbf{x} + \delta$, where c_i is the i-th row of the rotation matrix derived from the challenge c, $\mathbf{x}$ is the noise-free value, and δ the noise term. Their proposal, called COBRA, aims at minimizing the squared residual; they assume that the equations with the highest residuals are noisy and discard them, eventually arriving at a noise-free system of equations. Their method can identify the noise-free equations, even when the ratio of the noise-free equations is as low as 5%, and only roughly 200 noise-free equations are required to recover a polynomial of 256 coefficients in $\mathbf{s}_1$. Note that however that their method will not work when high noise levels lead to noise-free equations occurring only extremely rarely.

The framework of [7]. The authors of [7] propose using BP to deal with any kind of leakage that relates to (2). This approach can be seen as related to the BP instantiations used to decode Low-Density Parity Check (LDPC) codes and the graph used in [25] and the later work of [26]. Every secret key coefficient is modeled by a variable node; factor nodes model a noisy equation – i.e., one of the equations of the coefficients of (2). A factor node is connected to the τ key coefficients with non-zero indices in c, i.e., to all coefficients that appear in the equation. Thereby, the factor graph models the relations given by (2) as relations between random variables, and BP allows approximating the marginals of the joint distribution. This graph may be used very generically for both side-channel and fault attacks on various variables both with and without the knowledge of $\mathbf{z}$ and c. The authors provide simulations for various attacks and consider hiding (but no masking) countermeasures.

2.3 Masked Dilithium/ML-DSA.

Masking is a class of countermeasures, introduced in [8], that splits secrets into $t + 1$ shares; thereby, each share is independent of the secret and can be processed separately. Several masked ML-DSA proposals exist [37,3,13,11]. The latest works in this direction are [13] and [11]. We focus on these latest proposals and note that other implementations follow very similar principles for sampling $\mathbf{y}$.

The work of [13] presents gadgets for a fully masked Dilithium implementation. With their proposed gadgets, they protect the previously identified valuable targets by masking the sampling of $\mathbf{y}$ (Line 4 of Algorithm Algorithm 1), decomposition of $\mathbf{w}$ (Line 6 of Algorithm Algorithm 1), computation of $\mathbf{z}$ and $\mathbf{r}$ (Line 8 and 9 of Algorithm Algorithm 1, resp.), and the rejection sampling in its entirety. A crucial gadget for securing the sampling of $\mathbf{y}$ is their new Boolean to arithmetic masking conversion gadget, which uses the multi-bit Boolean to arithmetic masking conversion gadget proposed in [4]. The masked generation of $\mathbf{y}$ is performed by sampling in Boolean masking: the authors propose to sample $t + 1$ shares, with the result of XOR'ing them as a coefficient of a polynomial

in $\mathbf{y}$. This is possible as the coefficients of $\mathbf{y}$ are sampled uniformly random with support $2\gamma_1$. Subsequently, $\mathbf{y}$ is transformed to arithmetic masking with their conversion gadget. Arithmetic masking allows carrying out the remaining arithmetic computation of $\mathbf{z} = \mathbf{y} + c\mathbf{s}_1$.

In their subsequent work [11], they further improved the efficiency of the masked sampling of $\mathbf{y}$ and the rejection sampling by replacing their internal gadgets with more efficient ones. They also extended their proposal to cover ML-DSA, by protecting the hedged sampling of $\mathbf{y}$ which combines randomness derived from the message being signed using a PRNG and from the RNG. To protect the hedged sampling, they proposed to utilize masked Keccak [12]as PRNG to extract randomness from the message. Besides this additional source of randomness in the case of ML-DSA, the masked signing procedure remains essentially the same, and still depends on the same masking conversion gadget proposed in [4]. For both of their proposals, shown in Algorithm 2, the authors prove its security in the t-probing model [29]; Algorithm 2 shows Boolean masking of $\mathbf{y}$ in yellow and arithmetic masking of $\mathbf{y}$ in red.

While their gadgets and the associated implementation supports higher masking orders, the implementation is not yet highly optimized. The large memory overhead makes higher masking orders problematic for embedded systems. Ignoring the storage requirement for the public and masked secret key, the masked signing procedure of ML-DSA with parameters $k = 4, l = 4$ (ML-DSA-44) alone takes 73.5 KiB for masking order 1, 93.7 KiB for order 2, and 113.8 KiB for order 3 already. Consequently, in this work we focus on masking order 1.

2.4 The Framework of [14]

The authors of [14] show how to generically deal with side information, e.g., from a side-channel, in lattice-based schemes. They define several types of hints that can be integrated into the Distorted Bounded Distance Decoding Problem (DBDD) instance posed by the public key equation of the scheme; this results in a computationally easier problem. For a vector $\mathbf{v}$, secret $\mathbf{s}$, and integers l, k, and lattice Λ, the authors propose the following hints:

- **Perfect hint**: $\langle \mathbf{v}, \mathbf{s} \rangle = l$
- **Modular hint**: $\langle \mathbf{v}, \mathbf{s} \rangle = l \mod k$
- **Approximate hint**: $\langle \mathbf{v}, \mathbf{s} \rangle = l + \mathcal{N}$
- **Short vector hint**: $\mathbf{v} \in \Lambda$

These hints are defined on values and are often not well suited to directly deal with HW leakage. An approach that is similar to the integration of approximate hints may be used. However, this requires computing the distribution of the secret conditioned on the hints; i.e., computing (an approximating to) the marginals of the joint distribution given the hints. This can be achieved using the framework of [26], which we reiterate in the next section.

Several extensions and more efficient solutions for specific situations exist: The framework has been extended to decryption failure inequalities in [15], May

Algorithm 2 Simplified Masked ML-DSA Signing Algorithm [13]

procedure MaskedSign($\mathbf{ms}_1 = (\mathbf{s}_{1,i})_{i\in\{0,\cdots,t\}} \in (R_q^l)^t, \mathbf{ms}2 = (\mathbf{s}_{2,i})_{i\in\{0,\cdots,t\}} \in (R_q^k)^t, \mathtt{m}$)
  $\mathbf{z} \leftarrow \perp$
  while $\mathbf{z} = \perp$ **do**
    for $i = 0$ to t **do** $\mathbf{y}_i \xleftarrow{R} R_{2\gamma_1}^l$ **end for** ▷ $\mathbf{y}$ in Boolean masking
    $(\mathbf{y}_i)_{i\in\{0,\cdots,t\}} \leftarrow \mathrm{BtoA}_q\mathrm{Exact}((\mathbf{y}_i)_{i\in\{0,\cdots,t\}})$ ▷ $\mathbf{y}$ in arithmetic masking
    $\mathbf{y}_0 \leftarrow \mathbf{y}_0 - (\gamma_1 - 1)$ ▷ subtract from each coefficient of $\mathbf{y}_0$
    for $i = 0$ to t **do** $\mathbf{w}_i \leftarrow \mathbf{A}\mathbf{y}_i$ **end for**
    $\mathbf{w}_1, (\mathbf{w}_{0,i})_{i\in\{0,\cdots,t\}} \leftarrow \mathrm{SecDecompose}((\mathbf{w}_i)_{i\in\{0,\cdots,t\}}, 2\gamma_2)$
    $c \leftarrow \mathrm{H}(\mathtt{m} \,\|\, \mathbf{w}_1)$
    for $i = 0$ to t **do** $\mathbf{z}_i \leftarrow \mathbf{y}_i + c\mathbf{s}_{1,i}$ **end for**
    for $i = 0$ to t **do** $\mathbf{r}_i \leftarrow \mathbf{w}_{0,i} - c\mathbf{s}_{2,i}$ **end for**
    if $\mathrm{SecRejectionSamplingCheck}((\mathbf{z}_i)_{i\in\{0,\cdots,t\}}, (\mathbf{r}_i)_{i\in\{0,\cdots,t\}}, \gamma_1, \gamma_2, \beta)$ **then**
      $\mathbf{z} \leftarrow \perp$
    else
      $(\mathbf{z}_i)_{i\in\{0,\cdots,t\}} \leftarrow \mathrm{RefreshMasks}((\mathbf{z}_i)_{i\in\{0,\cdots,t\}})$
      $\mathbf{z} \leftarrow \mathrm{Unmask}((\mathbf{z}_i)_{i\in\{0,\cdots,t\}})$
    end if
  end while
  return signature $\leftarrow (\mathbf{z}, c)$
end procedure

and Nowakowski [36] show that in some cases computations can be done more efficiently, and [23] can be seen as a special case when combining with BP.

Note that the public key in ML-DSA does not contain the full $\mathbf{t} = \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2$ but only the higher bits. This essentially results in a much larger noise term.

2.5 The Framework of [26]

Recently, a generic framework for side-channel information has been presented in [26]. The authors provide a BP instantiation that allows recovering the secret key from generic side-channel hints in several types of schemes with a focus on lattice-based schemes. They define *distribution hints* as a tuple $\mathcal{H} = (\mathbf{v}, \mathcal{D})$ where $\mathbf{v}$ is a real vector of the same dimension as the secret and $\mathcal{D}$ is a (discrete) probability distribution on $\mathbb{R}$. The hint $\mathcal{H}$ represents the information that

$$\langle \mathbf{v}, \mathbf{s} \rangle \sim \mathcal{D}, \tag{3}$$

that is, that the inner product of $\mathbf{v}$ and the secret $\mathbf{s}$ follows the distribution $\mathcal{D}$. Distribution hints generalize the information obtained in several previous attacks as well as the definition of hints given in [14], and may be solved by either a BP-based solver or a solver that makes use of a greedy strategy. While [26] focuses its evaluations on lattice-based schemes in which the public key equation is available, their work applies more generally: no knowledge of the public key equation is required, and the scheme does not need to be lattice-based; the former allows us to apply the solver to Dilithium/ML-DSA as shown in Section 4.2.

2.6 Information Theory

Information theory quantifies the uncertainty of random variables. Central definitions are the entropy of a discrete random variable and the Mutual Information (MI) between two discrete random variables, both first introduced in [49][5]. Let X, Y be random variables with density functions (with respect to the counting measure) p_X, p_Y. The *entropy* $H(X)$ is defined as expected *information content*, i.e., as

$$\mathbb{E}[-\log_2(p_X(x))] = -\sum_x p(x)\log(p_X(x)). \tag{4}$$

The *mutual information* $I(X;Y)$ between X and Y measures how much information observing X or Y gives on Y or X, respectively. It is defined as

$$I(X;Y) = H(X) - H(X|Y) = H(Y) - H(Y|X) = H(X) + H(Y) - H(X,Y) \tag{5}$$

and fulfills $0 \leq I(X;Y) \leq \min(H(X), H(Y))$. MI has been used in side-channel analysis before, in particular, as a distinguisher, e.g, in Differential Power Analysiss (DPAs) [20], but has also been proposed to assess the security of masking schemes [16]. In [35], the authors use MI on simulated data to assess the impact of prime field versus Boolean masking in symmetric ciphers for varying noise levels. In this work, we use MI (non-simulated) to evaluate the maximum information that an adversary may obtain from different intermediate variables in varying leakage models, and simulate attacks to obtain the *exploitable* information for varying noise levels.

The Kullback-Leibler (KL) divergence [32] is a statistical distance between two distributions, which is closely related to MI. It is defined as

$$D_{\mathrm{KL}}(X||Y) = \sum_{x, p_X(x)>0} p_X(x)\log_2(p_X(x)/p_y(x)), \tag{6}$$

which is infinite if there exists x such that $p_X(x) > 0$ and $p_Y(x) = 0$. MI can be defined using KL: MI is the KL divergence of the joint distribution (X, Y) to the random variable that corresponds to the pointwise product of $p_X \cdot p_Y$.

A commonly used metric in SCAs is the signal-to-noise ratio (SNR). Defined as variance of the signal, i.e., the variance of the means for different groups of values or HWs, divided by the noise, i.e., the variance within a group, it relates the targeted information to the noise level. Note that it does not capture the amount of information that a variable holds and is, e.g., not invariant under scaling a random variable. If we assume the noisy HW model (c.f., [9]) for a uniformly random variable, i.e., that an implementation leaks the HW with an additive Gaussian error with standard deviation σ, the SNR is given by $\frac{b}{4\sigma}$ where b is the bitsize of the variable.

[5] Shannon defines the channel capacity as supremum of MI.

3 Information-Theoretic Analysis

To assess potential targets for side-channel attacks on masked ML-DSA implementations, we first analyze how much information an adversary may obtain on the secret in different leakage models and locations. All our leakage models assume the (noisy) HW model, i.e., an attacker obtains a (noisy) version of the HW of an intermediate, which has been shown to model the physical reality quite well. In this section, we assume noise-free HW leakage to assess the maximum amount of information an adversary may obtain in a certain representation and masking type; in Section 4 we simulate attacks under the assumption of a Gaussian error. We do not assume a specific implementation, but we assume a similar structure to the state-of-the-art masked ML-DSA implementation of [13,11].

3.1 Leakage Models and Locations

The proposals for masked implementations of [13,11] sample a coefficient y of $\mathbf{y}$ by uniformly randomly sampling its Boolean shares over $\{0, \dots, 2\gamma_1 - 1\}$; i.e., by sampling m_i, $i \in \{0, \dots, t\}$ and (implicitly) setting $y = \bigoplus_{i=0}^{t} m_i$. Subsequently, the shares are converted to arithmetic masking using the B2A conversion of [4], and then shifted by $-\gamma_1 + 1$[6].

To the best of our knowledge, there is no known securely masked method to sample $\mathbf{y}$ in arithmetic masking (both [37,3] use this approach as well). Therefore, we assume $\mathbf{y}$ to be processed as both arithmetic and Boolean shares. An adversary may target Boolean shares of $\mathbf{y}$ during sampling and B2A conversion or arithmetic shares in the addition of $\mathbf{y}$ with $c\mathbf{s}_1$; these locations are highlighted in Algorithm 2.

HW leakage. A common assumption in SCA is that the adversary may obtain a noisy version of the HW of a processed intermediate value instead of the value itself (see, e.g., [1]). Even though more fine-grained leakage models exist (see, e.g., [6]), this assumption has been proven to model real-world attacks reasonably well. If an implementation extracts single bits of an intermediate, the adversary may obtain a noisy version of those. If a targeted value has a very small range, an adversary may obtain its value directly from the HW; however, this is prevented by masking. Thus, in this section, we assume noise-free HW leakage to assess the maximum amount of information an adversary may obtain in different locations, representations, and masking types.

Masking ranges. Boolean masking as proposed in [13,11] uses $\log_2(2\gamma_1) = 18$ bit shares (20 for higher security levels). On the other hand, arithmetic masking is carried out over $\mathbb{F}_q$, where $\log_2(q)$ is close to 23. Under the knowledge of $\mathbf{z}$, the coefficients of $\mathbf{y}$ (with positive probability) have only about 8 bit entropy. However, as explained in the next paragraph, obtaining the HW of $\mathbf{y}$ directly is

[6] We however discovered a bug in the masked implementations [13,11]; the shares are shifted by $-\gamma_1$ instead.

less likely in arithmetic masking. Thus, the small range of $\mathbf{y}$ under the knowledge of $\mathbf{z}$ only helps an adversary in the case of Boolean masking. It is therefore reasonable to assume that the larger range in the case of arithmetic masking does cause less information to leak – this is confirmed by the results in Table 1.

Share recombining. Implementation mistakes, changes introduced by the compiler, and unexpected micro-architectural effects may cause first-order leakage. In the case of Boolean masking, there is no conceptual difference to targeting the HW of $\mathbf{y}$ in an unmasked implementation; however, we can expect the noise to be much larger. The case of arithmetic masking is slightly more complicated: Assume y to be arithmetically shared; then, for $m_i \in \mathbb{F}_q$, share recombination leaks

$$\mathrm{HW}(\bigoplus_{i=0}^{t} m_i) \text{ where } y \equiv \sum_{i=0}^{t} m_i \pmod{q} \tag{7}$$

instead of the HW of y. Clearly, this does give valuable information as well; however, much less than directly obtaining the HW of y.

Bit leakage. In some cases, bits of Boolean shares may leak individually. In particular, if some signal amplification is caused by, e.g., extracting or multiplying single bits by a larger integer (see, e.g., [24]). This is much less likely in arithmetic masking: while some bits of the unshared value can be computed from the bits of individual shares (e.g., the LSB), carries amplify noise.

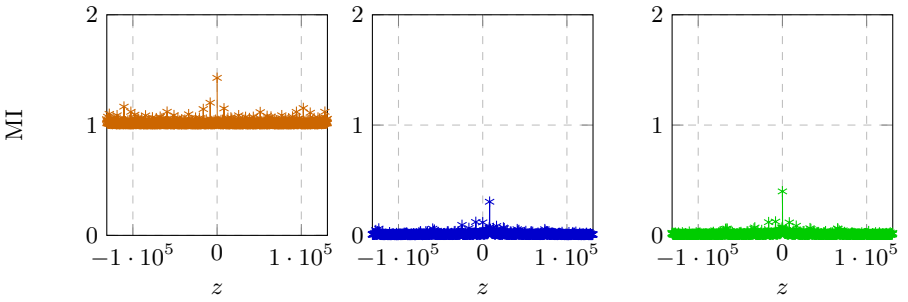
Leakage on the HW of both shares. If no first-order leakage or bit-leakage is present, the adversary has to resort to targeting the HW of both shares individually. The larger range of the arithmetic shares should make a second-order attack much more expensive and less noise-tolerant than against Boolean masking – again, Table 1 does confirm this.

3.2 Leaked Information

In ML-DSA, we obtain leakage on $\mathbf{y}$, but we are interested in information on $\mathbf{x} = c\mathbf{s}_1$, where the relation $\mathbf{x} = \mathbf{z} - \mathbf{y}$ holds. We compute the amount of information that may maximally be leaked in the noisy HW model for Boolean and arithmetic masking on the unmasked and first-order masked y, i.e., in the noise-free case. This allows us to compare different leakage models in the worst case. For noisy HW leakage, we evaluate the MI in Figure 2 and simulate the attacks in Section 4.3.

We may compute $I(x; L(\mathbf{m}) \mid z)$, where $\mathbf{m}$ is a sharing of y and L denotes HW leakage on the respective representation of the shares, by using that z is uniform over a support of $\mu = 2(\gamma_1 - \tau\eta)$; thus

$$I(x; L(\mathbf{m}) \mid z) = \frac{1}{\mu} \sum_{z=z'} I(x; L(\mathbf{m}) \mid z = z'). \tag{8}$$



(a) Shifted Boolean (b) Canonical Arithmetic (c) Centered Arithmetic

Fig. 1: $I(x; L(\mathbf{m}) \mid z = z')$ for every 100-th point.

Table 1: Mutual Information $I(x; L(\mathbf{m}) \mid z)$.

Variant	MI	Norm. MI	$H(x; L(\mathbf{m}))$	$H(L(\mathbf{m}))$	$H(x)$
Arithmetic cen. ($t = 2$)	0.01363	0.00263	12.72599	7.54986	5.1908
Arithmetic can. ($t = 2$)	0.01317	0.00254	11.74504	6.56844	
Boolean shifted ($t = 2$)	1.01949	0.19644	10.40409	6.23381	
Boolean shifted ($t = 1$)	2.16538	0.41740	5.189762	2.16538	
Arithmetic cen. ($t = 1$)	0.17219	0.03317	8.214697	3.19713	
Arithmetic can. ($t = 1$)	0.16708	0.03219	8.155458	3.13278	

Figure 1 shows MI $I(x; L(\mathbf{m}) \mid z = z')$ plotting every 100-th point. Table 1 shows the results for $I(x; L(\mathbf{m}) \mid z)$ for several leakage types represented by L; here, we computed $I(x; L(\mathbf{m}) \mid z = z')$ for all possible z' (i.e., for μ values). The parameter t denotes the attack order; $t = 2$ means that we target both shares, $t = 1$ means that we consider first-order leakage that arises from accidental share recombining. In both Table 1 and Figure 1, normalized MI is normalized with respect to $H(x)$.

Central versus canonical reduction. It might seem surprising and contradictory to [50] that using the central representation does not lead to a great increase in leaked information. This is because we are interested in information on $\mathbf{x} = c\mathbf{s}_1 = \mathbf{z} - \mathbf{y}$ instead of the HW of $\mathbf{y}$: Recall that

$$I(x; L(\mathbf{m}) \mid z) = H(L(\mathbf{m}) \mid z) - H(L(\mathbf{m}) \mid x, z) \tag{9}$$
$$= H(x \mid z) + H(L(\mathbf{m}) \mid z) - H(x, L(\mathbf{m}) \mid z). \tag{10}$$

As $L_{\text{cen.}}(y)$ often immediately leaks the sign of y, one could expect that

$$I(x; L_{\text{cen.}}(\mathbf{m}) \mid z) > I(x; L_{\text{can.}}(\mathbf{m}) \mid z), \tag{11}$$

by a much larger amount than shown in Table 1 where the normalized MI is almost the same. In fact, for $z = 0$, $I(x, L(\mathbf{m}))$ is much smaller for $L = L_{\text{can.}}$ than for $L = L_{\text{cen.}}$ (0.12 and 0.40). However, if we (negate and) shift x by z,

y is a variable with relatively small range centered around $z \neq 0$. Additionally, the entropy of $H(L(\mathbf{m}) \mid z, x)$ is dominated by the values close to z as these are the ones with the largest probability due to the distribution of x (i.e., a sum of uniform distributions). Depending on z, the reduced entropy of the distribution of HWs for $L = L_{\text{can.}}$ may actually lead to a reduced MI because of the reduced $H(x, L_{\text{can.}}(\mathbf{m}) \mid z)$. Thus, our evaluations are actually in-line with [50] but show a more complex scenario. We may interpret MI in the case of noise-free leakage on a shared variable as follows:

Proposition 1. *Let $L : M \to \mathcal{L}$ be a function modeling leakage on a variable from share domain to some leakage domain, $\mathbf{m} = (m_i)_{i \in \{0,\dots,t\}} \in M^{t+1}$ be a sharing of order t of a secret value x. Assume that m_0 is uniquely determined by y and $(m_i)_{i \geq 1}$, and that $y, (m_i)_{i \geq 1}$ are independent. Then, it holds that*

$$I(y; L(\mathbf{m})) = H(L(m_0) \mid (L(m_i))_{i \geq 1}). \tag{12}$$

Proof. As the secret y determines one share, we get that

$$P(\mathbf{m} \mid y) = \prod_{i=1}^{n} P(m_i) \text{ and thus } H(\mathbf{m} \mid y) = \sum_{i=1}^{n} H(m_i), \tag{13}$$

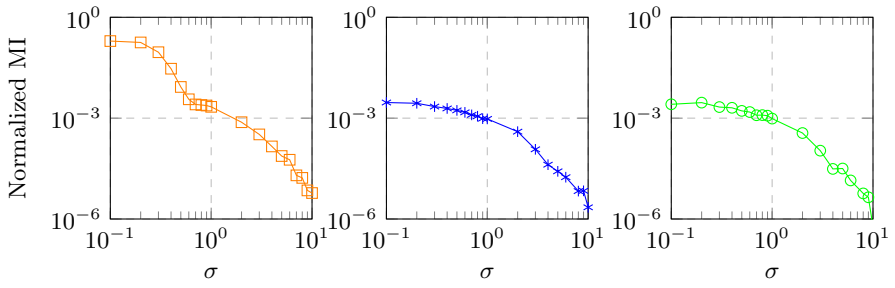
and we may compute $I(y, L(\mathbf{m}))$ as

$$\begin{aligned} I(y; L(\mathbf{m})) &= H(L(\mathbf{m})) - H(L(\mathbf{m}) \mid y) && (14)\\ &= H(L(m_0) \mid (L(m_i))_{\{i \geq 1\}}) + \sum_{i=1}^{n} H(L(m_i)) - \sum_{i=1}^{n} H(L(m_i)) && (15)\\ &= H(L(m_0) \mid (L(m_i))_{\{i \geq 1\}}). && (16) \end{aligned}$$

For two shares, this gives us the following intuition: The distribution of m_0 conditioned on m_1 is a shifted distribution of the secret's distribution (as the secret and m_1 uniquely determine m_0). Therefore, m_0 conditioned on $L(m_1) = l$ is the mixture distribution of the secret's distribution around all m_1' with $L(m_1') = l$. Now to get $L(m_0)$ conditioned on $L(m_1)$, we may apply L to the support of the previous distribution, summing up the probabilities for values with the same image under L. Thus, the MI depends on L evaluated on the range that m_0 still admits after knowing the value of L at m_1.

By using a central reduction, the number of possible HWs for share 0 given a HW for share 1 is greatly reduced if both shares have to sum up to a small range around 0: As already noted in [50], to, e.g. arrive at 0, it is much more likely to observe one large and one small HW, i.e., one negative and one positive value, than two HWs corresponding to two positive or two negative values. However, if we shift the distribution of the value that m_0, m_1 are a sharing of by some random value z, this relation is mostly lost.

Simulated mutual information. We may approximate the MI using the hint distributions in the simulation (see Section 4). Figure 2 shows the MI per noise level



(a) Boolean shifted. (b) Canonical arithmetic. (c) Centered arithmetic.

Fig. 2: MI for **noisy** HW leakage averaged over 5 simulated signatures.

σ averaged over 5 signatures. We observe similar results as [35]: for moderate and higher noise levels, the difference between Boolean and arithmetic masking is much smaller than for low noise levels. However, even for moderate to high noise levels, Boolean masking still leaks much more information.

Relation to $I(\mathbf{s}_1, L(\mathbf{m}))$ and successful attacks. The MI of $\mathbf{x}$ and $L(\mathbf{m})$, as computed above, is the same as $I(\mathbf{s}_1, L(\mathbf{m}))$. This follows simply from $\mathbf{x}$ being fully determined by $\mathbf{s}_1$ and the leakage being dependent only on $\mathbf{x}$; we have $H(L(\mathbf{m}) \mid \mathbf{s}_1) = H(L(\mathbf{m}) \mid \mathbf{x})$ and, therefore,

$$I(\mathbf{s}_1, L(\mathbf{m})) = H(L(\mathbf{m})) - H(L(\mathbf{m}) \mid \mathbf{s}_1) \tag{17}$$
$$= H(L(\mathbf{m})) - H(L(\mathbf{m}) \mid \mathbf{x}) = I(\mathbf{x}, L(\mathbf{m})). \tag{18}$$

In most cases, unless the adversary can fix the randomness, they will obtain "incomplete" information on several $c\mathbf{s}_1$ for varying c. They may target 256 coefficients of $\mathbf{y}$ per component of $\mathbf{s}_1$, but the information obtained for the same c is not independent. Thus, comparing the first column of Table 1 normalized by the entropy of $\mathbf{s}_1$ only gives a rough estimate on the information we have on $\mathbf{s}_1$. In addition, the amount of information on $\mathbf{s}_1$ that a solver can extract from incomplete information on $\mathbf{x}$ varies greatly depending on the leakage model. In the case of arithmetic masking and leakage on coefficients of $\mathbf{y}$, we can recover $\mathbf{s}_1$ with approximately three times the information indicated in Table 1. However, for Boolean masking, while still requiring much less information than for arithmetic masking, the overhead is much larger at a factor of about 150. In case of bit leakage, learning the LSB gives more information (1 bit) than the Most Significant Bit (MSB), which gives less than 1 bit because in some cases it is biased under the knowledge of z (without side information). Nevertheless, solving from the LSB[7] is fruitless even for very high numbers of traces/signatures, while solving from the MSB requires just about 5 signatures.

[7] Note the similarity to LPN if noise is present.

4 Targeting Masked Implementations

We may now explain how to target $\mathbf{y}$ in a protected implementation. Several solvers have previously been published [26,7]. We first compare the different approaches and then explain how to adapt the solver of [26] to a setting where each hint holds a very small amount of information. Finally, we apply the solver to several types of leakage discussed and analyzed in the previous section.

Notation and terminology. We make use of the notation of [26]: a hint is defined as tuple $(\mathbf{v}, \mathcal{D})$ and represents the information that $\langle \mathbf{v}, \mathbf{s}_1 \rangle$ follows the distribution $\mathcal{D}$. When applying the solvers of [26] (but not the subsequent lattice reduction), a coefficient is *correct* if it is equal to the respective key coefficient. We assume that the adversary orders the coefficients by the entropy of the respective variable node or the score of the actions involving that coefficients for the BP and greedy solver, respectively. We call coefficients *recovered* if it is correct and all smaller – in that ordering – coefficients are correct. Now, $\tilde{L}$ is a random variable modeling noisy leakage on intermediate variables. For example, $\tilde{L}$ could model HW leakage with Gaussian noise on an element of $\mathbb{F}_q$ represented canonically, i.e., in $\{0, \dots, q-1\}$ and $\tilde{L} = L_{can.} + \mathcal{N}(0, \sigma)$.

Instantiating the solver of [26]. When targeting $\mathbf{y}$ with published $\mathbf{z}$ and c, we may make use of

$$\mathbf{x} = c\mathbf{s}_1 = \mathbf{z} - \mathbf{y}. \tag{19}$$

This gives l separate equations for each component of $\mathbf{s}_1$. Information on coefficients of a component of $\mathbf{y}$ translates to a hint on $\mathbf{s}_1$ with the coefficients being the rows of the matrix representing multiplication with c, i.e., the rotation matrix of c, denoted by $\mathrm{rot}(c)$. Thus, if we obtain information on a coefficient of a component of $\mathbf{y}$, we first negate and then shift the distribution by the respective coefficients of the component of $\mathbf{z}$; then we instantiate a distribution hint for the respective component of $\mathbf{s}_1$ with that distribution and the row of $\mathrm{rot}(c)$ that corresponds to the coefficient of $\mathbf{y}$ and $\mathbf{z}$. We discuss how to compute the distribution in Section 4.2.

Knowledge of $\mathbf{t}_0$. In [26], the authors assume that the adversary has access to the LWE problem posed by the public key equation in ML-KEM, i.e., $\mathbf{t} = \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2$. This allows the authors to apply the techniques of [23] to obtain a lattice problem to solve for the secret key from partially recovered information. In ML-DSA, however, the public key contains only the higher bits of $\mathbf{t}$. While $\mathbf{t}_0$ may be recovered [42], we do not assume that this is feasible for the adversary in the targeted use case of ML-DSA. Further, as $\mathbf{s}_2$ is not assumed to be targeted or known, the adversary has less information than in the evaluations of [26]. Thus, in this section, we assume that the adversary only applies the solvers of [26] but not the lattice reduction of [23]. Our results are stated in terms of recovered coefficients and Euclidean distance to the secret key; those two variables directly influence the hardness of the resulting lattice problem if $\mathbf{t}_0$ is available and the remaining brute-force effort otherwise.

4.1 Comparison of Previous Solvers

Several solvers have previously been proposed: The solvers of [26] which deals with generic hints on lattice-based schemes, the method of [7], and the approach of [47]. In this work, we rely on [26], which can be instantiated to be equivalent (or at least very similar) to [7].

The method of [47]. The COBRA approach presented in [47] (see Section 2.2) relies on solving a noisy system of equations on $\mathbf{x}$ or $\mathbf{y}$ by identifying noise-free equations (i.e., with a zero noise term), which means this approach will not work when the number of noise-free equations is below a threshold. The authors of [47] observed that roughly 200 noise-free equations are required to recover a polynomial of 256 coefficients in $\mathbf{s}_1$.

As $\mathbf{y}$ is not under the adversary's control, unless the adversary is able to fix randomness by e.g. fault injection, the adversary can only extract information about coefficients of $\mathbf{y}$ from a single trace. When targeting unprotected implementations in a low-noise setting, the small range of coefficients of $\mathbf{y}$ (given the knowledge of $\mathbf{z}$) allows an adversary to recover relatively many correct coefficients of $\mathbf{y}$ from just the HW already, hence the applicability of their approach.

For masked implementations however, there is only a negligible chance of recovering noise-free coefficients of $\mathbf{y}$ from a single trace. Even when the adversary obtains leakages on the HW of two shares of an arithmetic masked $\mathbf{y}$, the adversary is extremely unlikely to recover the unshared value with just the HWs since both shares have a range of $\log_2(8380417) = 23.00$ bits.

In addition, the results in [47] indicate that more equations are needed for the COBRA approach in comparison to BP-based solvers. For example, when we simulate leakages on $\mathbf{y}$ and $\mathbf{x}$ at a single Point of Interest (PoI) with a $\sigma = 20$ in the noisy HW model (correspond to an SNR of 0.33 and 0.12 for centered and canonical reduction respectively), our BP solver can recover $\mathbf{s}_1$ with 20 and 30 traces, respectively. In contrast, [47] reports that 36 traces are required with an SNR of 1 using 20 PoIs.

The authors [47] noted that their attempts of using BP "temporarily failed to complete the solution when the number of equations is excessive". We confirm their observation and provide a method to work around this issue, thereby allowing belief-propagation based solvers to accept more hints in Section 4.2.

Belief propagation instantiations. The solvers presented in [26] and [7] rely on BP[8], and their graphs look very similar. In fact, both approaches model exactly the same computation if we instantiate the BP solver of [26] as discussed before: Given a distribution hint with coefficients $(\tilde{c}_i)_i$ and distribution $\mathcal{D}$, the respective factor node of [26] computes the distribution of $s_j = \sum_{i \neq j} \tilde{c}_i x_i$ for all $j \in \{0, \ldots, 255\}$. The x_i are assumed to be distributed according to the messages coming from the variable nodes – initialized as uniform. The hint distribution $\mathcal{D}$

[8] But [26] also presents a greedy solver.

is a distribution on

$$\tilde{\mathbf{c}}\mathbf{s}_1 = \sum_{i=0}^{255} \tilde{c}_i s_{1,i} \tag{20}$$

and the updated distribution for x_j can be computed from

$$\tilde{c}_j x_j = \sum_{i=0}^{255} \tilde{c}_i s_{1_i} - \sum_{i \neq j} \tilde{c}_i x_i. \tag{21}$$

The solver of [26] computes all s_j in the Fourier domain using a forward-backward pass approach. The BP instantiation of [7] represents the computation of s_j as a BP subgraph, and propagates it separately. As the subgraph is acyclic and scheduled independently, this part of the BP computation is exact, giving the same results as [26].

A factor node of [26] has complexity $O(n \log n)$ while the subgraph of [7] has complexity $O(n^2)$. In the setting of targeting Dilithium, the graph is relatively sparse, which greatly reduces the runtime: running both solvers of [26] for 50 iterations rarely takes more than a few seconds, and [7] reports similar runtimes.

Given the similarity of these solvers, it is not surprising that we find very comparable performance with respect to the number of required signatures to fully recover the secret key. We compared our instantiation of [26] to [7], and it achieves very similar results in the case of leakage on $\mathbf{y}$ and $c\mathbf{s}_1$. In fact, we get slightly better results, which may be for various reasons[9] and is not substantial.

4.2 Applying [26] to a Masked Implementation

We rely on the solvers of [26] as these are open source and seem to offer slightly improved performance over [7], and [47] does not fit our leakage model. First we explain how we compute distribution hints from leakage on several shares. Then, we show how to deal with decreasing information per hint; this is relevant when targeting shared values as hints in this case hold much less information.

Computing distribution hints. A distribution hint consists of a vector – the hint coefficients – and a distribution. As previously discussed, given c, z, and leakage on y, the hint coefficients are simply the rows of the rotation matrix of c as $\mathbf{x} = c\mathbf{s}_1 = z - y$ holds. The distribution models the leakage that is obtained on an intermediate value. Note that each y gives a hint on one of the l components of $\mathbf{s}_1$, depending on which component of $\mathbf{y}$ the coefficient belongs to.

For unshared values, the computation of the distributions on $\mathbf{x}$, i.e., of $\mathcal{D}$, is very similar to [7]. However, [7] evaluates their solver for leakage on $\mathbf{x}$ **and** $\mathbf{y}$, where the largest part of information comes from leakage on $\mathbf{x}$, which has a much smaller entropy when targeting the HW as it is centered around 0.

[9] For example, more BP iterations.

For a masked implementation, we cannot assume leakage on an unmasked $\mathbf{x}$, and in Section 4.3, we evaluate leakage on an unshared $\mathbf{y}$ as it could occur, e.g., due to accidental share recombining. In this case, $\mathcal{D}$ is computed as

$$P(x = x' \mid z, \tilde{L}(y)) \propto P(\tilde{L}(y) \mid x = x', z)P(x = x') \tag{22}$$

$$= P(\tilde{L}(z - x'))P(x = x'), \tag{23}$$

using $\mathbf{y} = \mathbf{z} - \mathbf{x}$, which is similar to the computations of [7].

In case of leakage on shares $\mathbf{m} = (m_i)_{i\in\{0,\dots,n\}}$ of y, we may compute $\mathcal{D}$ by using

$$P(x = x' \mid z, \tilde{L}(\mathbf{m})) \propto P(x = x')P(\tilde{L}(\mathbf{m}) \mid z, x = x') \tag{24}$$

where the last factor, $P(\tilde{L}(\mathbf{m}) \mid z, x = x')$, is

$$\sum_{(m_j)_{j\geq 1}=(m'_j)_{j\geq 1}} P((m_j)_{j\geq 1} = (m'_j)_{j\geq 1})P(\tilde{L}(\mathbf{m}) \mid z, x = x', (m_j)_{j\geq 1} = (m'_j)_{j\geq 1}) \tag{25}$$

$$= \sum_{(m_j)_{j\geq 1}=(m'_j)_{j\geq 1}} P(\tilde{L}(m'_0)) \prod_{i=1}^{n} P((m_j)_{j\geq 1} = (m'_j)_{j\geq 1}) \tag{26}$$

where m'_0 is the unique value such that $(m'_i)_{j\in\{0,\dots,n\}}$ is a sharing of $z - x'$. Note that the term $P(x = x')$ is not factored into $\mathcal{D}$ as it is implicitly considered due to the structure of the factor nodes and the messages sent by the variable nodes.

For $n = 1$ and HW leakage, we may efficiently compute this by looping over all values m'_1 with HW h_1 with positive probability p_1 and computing probability of x as

$$P(\mathrm{HW}(m_0))p_1, \tag{27}$$

where m_0 is the unique value such that m_0, m_1 are a sharing of $z - x$ in the respective representation. In our implementation, we instead compute the leakage on y and then negate and shift the distribution for easier debugging.

Filtering equations. With increased noise, many hints we obtain from a side-channel hold little to no information about the secret. The convergence of a solver that handles arbitrary precision would not be impacted by such hints – uniform messages simply do not update variables –, but the performance would decrease. But, as already mentioned in [26], in practice, these do affect both performance and convergence. In fact, using the solver of [26] with a large number of distribution hints with distribution that is close to uniform leads to worsened convergence and oscillations between a state of almost no information gain and some level of information gain. The multiplication of many messages that represent a distribution that is close to uniform causes numerical instabilities that lead to impossible states. These lead to messages that are all zero, which represents that there is no possible solution to an equation. Instead of aborting, the implementation of [26] chooses to reset such messages to a uniform distribution,

which leads to oscillation between a state where some information about the key coefficients is recovered and a state where many coefficients have been (re-)set to uniform because of conflicting information. Thereby, the noise level that [26] can handle is limited in practice.

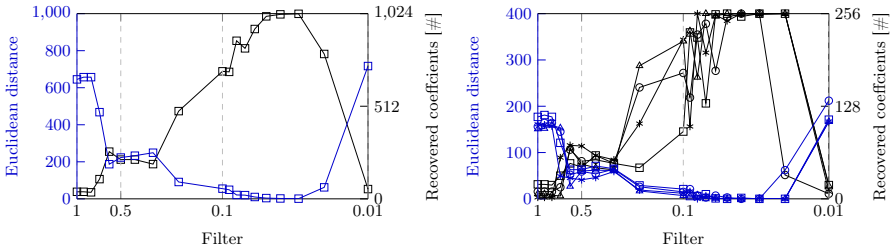
A naïve approach to circumvent this limitation would be to only use measurements that give a certain amount of information, i.e., where we are in edge cases. For example, if we assume HW leakage, we may only consider those HWs which are right at the mean of one group. For bit leakage, one could only consider those bits where a bit is recovered with high probability as, e.g., done in [24]. However, this ad-hoc approach ignores, e.g., the knowledge on the range of y given z.

Filtering equations by entropy of the hint distribution. We propose filtering distribution hints by the entropy of $\mathcal{D}$. Given hints $\mathcal{H}_i = \langle \mathbf{x}, \mathbf{v_i} \rangle \sim \mathcal{D}_i$ computed from leakage on $\mathbf{y}$, we compute the entropies h_i of $\mathcal{D}_i$, i.e., $h_i = H(\mathcal{D}_i)$. We then discard the f hints with the highest entropy for some filter setting $f \in [0, 1]$; i.e., if we obtained m signatures, we obtain $m \cdot n = m \cdot 256$ hints for each of the l polynomials of $\mathbf{s}_1$, we instantiate the BP with the $f \cdot n \cdot m$ distribution hints with the lowest entropy.

When applying the solver of [26] to masked implementations of Dilithium/ML-DSA, the majority of the computation time is spent during the computation of the hint distributions. This cannot be avoided by filtering as the distributions are required for filtering in the first place. An adversary could potentially combine filtering on distribution level with the approach described above, but as these computations can be parallelized with little effort, we did not explore such strategies. In addition, once a certain number of hints has been computed, several filter levels can be applied until an f that allows for recovering a subkey has been found. An adversary may also choose to filter with respect to the entropy of a uniform distribution (which maximizes the entropy).

The optimal filter level is difficult to predict and varies for different sets of hints. A strict filter may result in too little information to recover the key, a loose filter may cause numerical instabilities. In practice, we found filter levels that retain hints corresponding to between $50 - 200$ signatures to be a reasonable setting, but the exact level varies; our simulations and physical attacks show that it depends on the type of leakage, noise level, and number of hints.

As the BP is computationally inexpensive compared to the computation of the hint distributions, we run the solvers for several filter levels. Figure 3 shows the (maximum) number of correct and recovered coefficients for several filter levels for noisy HW leakage on first-order Boolean shared $\mathbf{y}$ with a standard deviation of $\sigma = 2.0$ (SNR 1.125) and 4000 equations. Figure 3a shows the results as sum over all $l = 4$ subkeys, i.e., the components of $\mathbf{s}_1$; Figure 3b shows the results for individual subkeys. It can be seen that the number of recovered coefficients varies between subkeys for a given filter level, but the distance is relatively consistent. We only used a single set of signatures to save computational resources; Section 5.1 includes evaluations for filtering for the attack on physical devices.



(a) Sum of L subkeys. (b) A single subkey.

Fig. 3: Maximum number of recovered coefficients and distance to the key for several filter levels for HW leakage on 2 Boolean shares with $\sigma = 2.0$ with 4000 signatures. In Figure 3a, the results are shown as sum over the results of the subkeys; Figure 3b shows the individual results for each of the $l = 4$ subkeys.

In the following, we will express filter levels as the number of signatures that they correspond to; i.e., for 2 signatures and $f = 0.5$ it is 1 even though we filter on hint level. This allows for an easier interpretation in relation to the number of signatures.

Filtering based on KL divergence. As shown in (21), the solver of [26] updates key coefficients based the current distribution modeled by the state of the variable nodes and the hint distributions of the factor nodes. In the first step, the variable nodes model the prior of $\mathbf{x} = c\mathbf{s}_1$. Even though the prior at each variable node, modeling a coefficient of $\mathbf{s}_1$, is uniform, the implicit prior on $\mathbf{x}$ is not uniform but the sum of τ uniform random variables.

The KL divergence expresses difference in information between two distributions, is closely related to MI, and it has previously been used in BP to adapt to countermeasures [27]. Thus, it might seem reasonable to filter hints by their "update potential", e.g., the KL divergence or the statistical distance to the prior of x. However, this takes only the initial state into account, and we did not find a way to make the updating step dependent on the KL divergence from hints to the distribution of the current state of variable nodes.

In our experiments, we observed filtering by KL divergence with respect to the prior to perform worse than filtering by entropy of the $\mathcal{D}$. Intuitively, by filtering by entropy of $\mathcal{D}$, we keep those hints which may – irrespective of the current state – reduce the entropy of the variable nodes the most. On the other hand, the KL filtering has a dependency on the prior distribution of $\mathbf{x}$. We suspect that this dependency is less relevant in later iterations and therefore leads to this type of filter being less effective.

Note that our entropy filter also increases the KL divergence. We believe that similar techniques could improve the solver [26]; however, as our technique proves to be highly efficient already, this is out of scope for our work, and we leave further improvements as future work.

4.3 Simulation

We implemented the methods described in the previous section, and simulate leakage in various models targeting ML-DSA-44. Here, we assume that the adversary launches a template attack [9], i.e., learns the leakage distribution in a training phase and subsequently matches the measurements against the templates during the attack phase.

To simulate leakage, we rely on a modified PQClean [31] implementation together with a Python implementation that makes heavy use of a Rust module for all tasks that are computationally expensive. The latter makes use of multithreading wherever applicable.

Almost all results are obtained using 50 iterations and filter levels f, such that $f \cdot m \in \{50, 100, \ldots, 300\}$ where m is the number of signatures. The exception is leakage on two Boolean shares of $\mathbf{y}$, where we use many more filters – $f \cdot m \in \{100, 110, \ldots, 400\}$. For first-order leakage, we ran 5 experiments, for second-order leakage we only ran a single one; in both cases, we use fresh randomness (i.e., independent sets of traces, signatures, and leakage) for every number of signatures. The figures in this section then show the average (over all experiments) of the maximum (over all iterations) number of recovered coefficients (as sum of $l = 4$ subkeys, each with 256 coefficients) and the minimum distance to the secret key. While the chosen filter levels are not necessarily optimal, we expect no drastic improvements for other settings, and the required computational power and time for our evaluations was already rather high.

Noisy HW model. In Section 3, we analyzed the maximum information that an adversary can obtain, i.e., we assumed noise-free leakage. For our simulations, we assume the noisy HW model (on different representations) and noisy bit leakage. This means that we assume the implementation to leak $h + \mathcal{N}(0, \sigma)$ or $b_i + \mathcal{N}(0, \sigma)$, where h is the HW of the intermediates, $\mathcal{N}(0, \sigma)$ denotes a normal distribution around 0 with variance σ^2, and b_i are the bits of the intermediate. We assume a template attack, i.e., the adversary knows the density functions p_h for each HW or bit h and assigns a HW or bit h' the probability $\frac{p_h(h')}{\sum_{h''} p_{h''}(h')}$.

Performance. While the BP itself usually finishes within minutes for all l targeted components of $\mathbf{s}_1$. The computational resources we required to compute the hint distributions are comparably high: for example, computing the distributions for Boolean masking for $\sigma = 5.0$ with 2000 signatures required over 200 GB of memory[10] and took over 2 hours with 500 threads. For arithmetic masking, the runtime is greatly increased, often taking several days with 500 threads even for noise levels below 1.0.

The performance depends greatly on the magnitude of the noise: for higher noise levels, we have to consider many more HWs. However, even for $\sigma = 0.5$, our implementation takes about 3 minutes per signature with 128 cores when targeting second-order arithmetic masking (around 800 signatures are required

[10] However, further memory optimizations may be possible.

for key recovery), i.e., for $4 \cdot 256$ hints. This can be tackled using multithreading, and it only affects the offline phase; thus, it is unlikely to prevent a determined attacker from carrying out an attack. Further, when we target Boolean shares, we may only consider the HW of the 8 LSBs of the coefficients of $\mathbf{y}$. This drastically reduces the runtime; in this section, we provide simulations targeting the full HW as otherwise the results are hardly comparable due to the switching noise caused by the remaining bits. In Section 5, we target the lower 8 bits.

Evaluation metrics. The work of [7] uses SNR to quantify their measurement noise. We express the noise in standard deviation σ of the normally distributed noise. The SNR considers both signal and noise, where the former is greatly dependent on the targeted location while the latter is also dependent on the (simulated) measurement setup. Thus, to give a clear picture of which location is more valuable for an adversary, we compare similar noise levels for values with different signal qualities. In a sense, the SNR hides information that we would like to express, while σ only quantifies the noise.

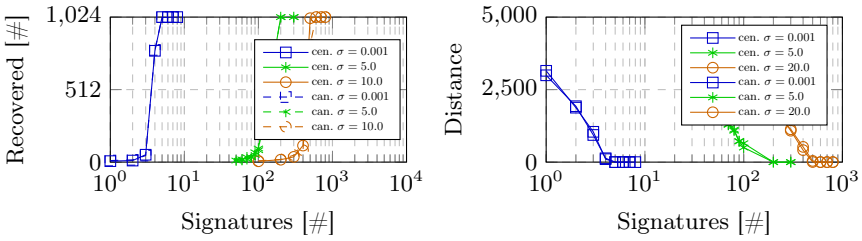
Further, we do not rely on the median recovered coefficients. Instead, we use the definition of recovered coefficients as given in [26], as well as the Euclidean distance of the best key guess to the true key. Together, recovered coefficients and Euclidean distance fully capture the hardness of the CVP from which the full key can be recovered when applying [23]. However, as the adversary might not have access to $\mathbf{t}_0$, we do not assess the hardness itself. If $\mathbf{t}_0$ is not available, those two variables influence the brute-force complexity.

Leakage on unshared variables. Implementation mistakes or unexpected microarchitectural effects might cause the unshared value of a secret variable to leak. Some scenarios that are more likely in the case of masked implementations have previously not been evaluated. In this section, we fill this gap and simulate several types of leakage.

Targeting $\mathbf{y}$ *or both* $\mathbf{x}$ *and* $\mathbf{y}$*.* The evaluations of [7] focus on targeting $\mathbf{y}$ and $\mathbf{x} = c\mathbf{s}_1$ simultaneously. This is a realistic assumption for an unmasked implementation, but not for a masked implementation, from which an adversary is less to be able to obtain leakage on unshared $\mathbf{x}$ and $\mathbf{y}$. In addition, targeting $c\mathbf{s}_1$ in an unmasked implementation gives far more information than $\mathbf{y}$ due to its reduced range.

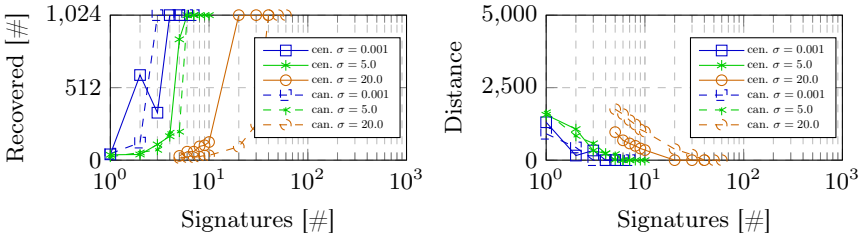
Leakage on unshared coefficients of $\mathbf{y}$, might happen due, e.g., accidental share recombining. To give a comparison, we evaluate HW leakage on $\mathbf{y}$ and on both $\mathbf{x}$ and $\mathbf{y}$. The former is shown in Figure 4 and the latter in Figure 5. It can be clearly seen that targeting $\mathbf{x}$ provides a great amount of additional information. For example, for $\sigma = 5.0$ and central reductions, targeting $\mathbf{y}$ requires about 200 signatures, and if we additionally have leakage on $\mathbf{x}$, it is less than 10.

The improved results when targeting $\mathbf{x}$ are not surprising. After all, $\mathbf{x}$ is the variable we want to recover in the first place **and** its HW has a much smaller entropy than the one of $\mathbf{x}$ (even under the knowledge of $\mathbf{z}$). However, in a masked



(a) Rec. $\mathbf{y}$. (b) Dist. $\mathbf{y}$.

Fig. 4: Targeting the HW of $\mathbf{y}$ in the HW model with standard deviation σ for the Gaussian noise.



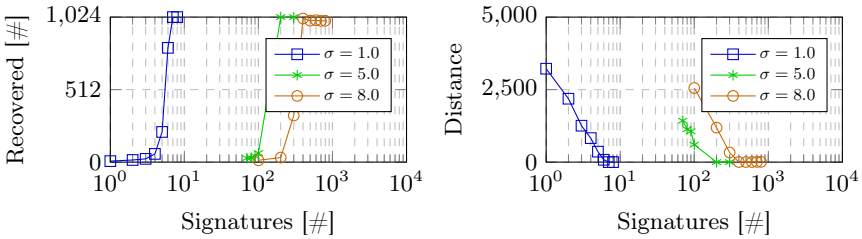
(a) Rec. $\mathbf{y}$ and $\mathbf{x}$. (b) Dist. $\mathbf{y}$ and $\mathbf{x}$.

Fig. 5: Targeting the HW of and both $\mathbf{y}$ and $\mathbf{x}$ in the noisy HW model with standard deviation σ for the Gaussian noise.

implementation, we are less likely to encounter first-order leakage on the coefficients of both $\mathbf{y}$ and $\mathbf{x}$.

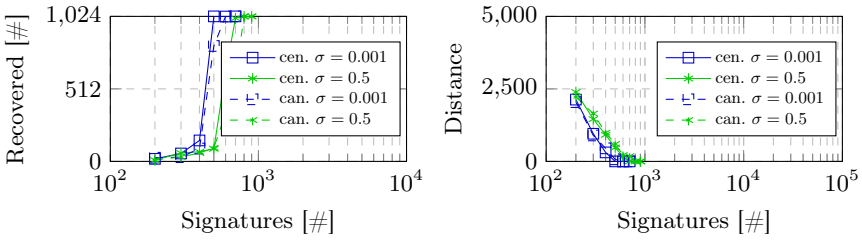
Targeting single bits of $\mathbf{y}$. An adversary may also obtain leakage on bits of coefficients of $\mathbf{y}$. This may lead to reasonably strong leakage if bits are processed individually, in particular, when they are multiplied by larger values (see e.g., [24]). Figure 6 show our evaluations for leakage on individual bits as it may occur during the B2A conversion. Note how a very small bias learned from a SCA already allows recovering $\mathbf{s}_1$ from a few hundred traces.

Second-order attacks. If no first-order leakage is present, the adversary may instead target the individual shares and compute the hints as described in Section 4.2. As shown in Figure 8, second-order attacks are still far from infeasible even for moderately high noise levels when targeting the first-order Boolean masked $\mathbf{y}$. For arithmetic masking, see Figure 7, more signatures are required and the necessary computational resources to process one signature are greatly increased. Note that both Figure 7 and Figure 8 show evaluations for a single experiment, i.e., targeting $l = 4$ separate polynomials using new randomness for each number of signatures.



(a) Rec. bits (b) Dist. bits

Fig. 6: Targeting the 8 least significant bits of $\mathbf{y}$ with a normally distributed error; $\sigma = 1.0, 5.0, 8.0$ correspond to success rates of 0.69, 0.54, and 0.52, respectively.



(a) Rec. arithmetic. (b) Dist. arithmetic.

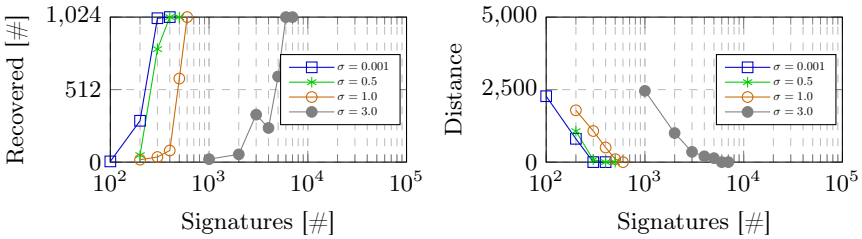
Fig. 7: Targeting the HW of two shares of $\mathbf{y}$ in arithmetic masking.

In Section 5, we only target the 8 LSBs, which drastically increases the performance, and an evaluation using 5 different sets of traces for each noise level is shown in Figure 11. This approach does not apply in the case of masking.

5 Physical Attack and Countermeasure

We verify our results by targeting physical devices, and then present a simple – yet effective – countermeasure. Our simulations suggest that an adversary can obtain the most information about the secret key from bit leakage of higher bits. Thus, we first assess whether we can target single bits during the B2A conversion used in the implementation of [11]. We consider two scenarios: on a ChipWhisperer Lite [28], which is known to exhibit very little noise, and using a Teledyne Lecroy WaveRunner 9254M oscilloscope – which shows a scenario that is closer to a real-world adversary. We then carry out several of the attacks simulated in the previous section in the latter measurement setup. In the course of our attack, we also provide some further evaluation on our filtering technique.

Both the information-theoretic analysis and the simulations suggest that the Boolean masked $\mathbf{y}$ is more vulnerable. Therefore, we suggest using a higher masking order during sampling and the B2A conversion of $\mathbf{y}$. The relative computational cost of sampling and conversion are low, thus, this countermeasure



(a) Rec. Boolean. (b) Dist. Boolean.

Fig. 8: Targeting the HW of two shares of $\mathbf{y}$ in Boolean masking.

has very little overhead. Further, the formal proofs of [11] still hold, and only very minor adaptations to the code base are necessary to implement it in practice. We evaluate the countermeasure in Section 5.2.

5.1 Attacking a State-of-the-Art Masked Implementation

In order to verify our results from simulation in the real world, we utilized our observation from Section 3.2 and attacked the latest masked ML-DSA-44 implementation [11]. In particular, since Boolean masking is expected to leak more information on the unshared secret, we target coefficients of $\mathbf{y}$ when they are still in the Boolean masking format.

Targeted implementation. We target the implementation of [11] in first-order masking; due to the relatively limited performance on a microcontroller, we did not target higher-order masking. While different proposals exist, [11] is the latest; it is publicly available. First-order leakage serves as a validation of our simulations and an example of how potential leakage could arise in a real-world implementation as well. Second-order leakage on the other hand will almost certainly be presented in a first-order masked implementation, and this scenario is therefore realistic.

Measurement setups. We ported the order-1 masked ML-DSA-44 implementation to `STM32F303R8T6` (Cortex-M3) and `STM32F415RGT6` (Cortex-M4) Microcontroller Unit (MCU) boards and compiled it with `-O3` and `-Os` flag respectively. For the `STM32F415RGT6` MCU, we use the ChipWhisperer UFO Board as the target device and a 10 MHz crystal oscillator as the clock source. To measure the power consumption we use a Teledyne Lecroy WaveRunner 9254M oscilloscope, connected to the SMA connector of the integrated shunt resistor of the target board. As the `STM32F415RGT6` target board exhibits a significant amount of high-frequency noise, we employ a cascade of one 50 MHz and two 25 MHz low-pass filters, and a low-noise amplifier. To ease measurements in our laboratory setting, we made small modifications to the implementation. To synchronize measurements, we placed trigger outputs for the oscilloscope around

the B2A conversion within the functions *gen_ y* and its fast variant *gen_ y_fast* that sample coefficients of $\mathbf{y}$.

Furthermore, we added control logic to the RNG used by both functions to ensure that signature generation only requires a single iteration. To be specific, we first run an identical implementation on a faster machine with a random RNG seed, retain the state of the RNG before the successful signing iteration, and use that state to generate a signature on the target device.

We employ template attacks since an attacker must extract information on $\mathbf{y}$ in a single trace. In the profiling phase, we measure N_L executions of *gen_ y* (resp. *gen_ y_fast*) on its own. During the attack phase, we generate N_S signatures for random 32 B messages, and use the segmented measurement functionality of the oscilloscope to capture all $4 \cdot 256$ calls to *gen_ y* (resp. *gen_ y_fast*). In the analysis phase, we use the profiling traces to create templates for various intermediate values, using Test Vector Leakage Assessment (TVLA) or Normalized Interclass Variance (NICV) to identify suitable PoI within a trace, and, during the subsequent attack phase, we derive probability distributions for each coefficient of each component of $\mathbf{y}$ for each signature.

For the `STM32F303R8T6` MCU, we use a ChipWhisperer Lite as the target device. The setup is similar to the `STM32F415RGT6`, except that we use the on-board clock source, and we use no filters. Due to the limited SRAM on this chip, it is not possible to perform a full signing operation with the masked ML-DSA-44 implementation [11]. Thus, we offload computations that we do not target to an `x64` laptop.

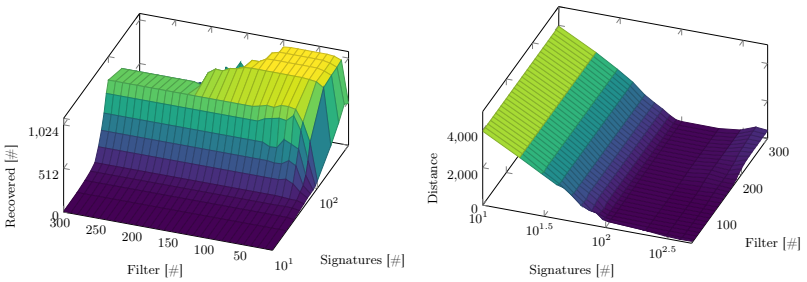
Targeting individual bits. As the attacks on both MCUs show, the Boolean to arithmetic conversion gadget [4] which the authors rely on for their various proposals [13,11], may leak individual bits of the unshared secret if not implemented carefully.

The accuracy of recovering individual bits in a coefficient of $\mathbf{y}$ is listed in Table 2: To obtain these results, we used 1 million traces to build the templates from 20 PoIs for both `STM32F303R8T6` and `STM32F415RGT6`; still, the success probability for all bits stays below 59.10% for `STM32F303R8T6` and 54.97% for `STM32F415RGT6`. The noise levels and SNR at the best PoI for each bit are also listed, which explain the marginal success probabilities. Nevertheless, this is sufficient to fully recover $\mathbf{s}_1$ from on average 27.6 and 247.0 signatures for the `STM32F303R8T6` and `STM32F415RGT6` setups, respectively. In Figure 9 we show the distance and number of recovered coefficients per number of signatures and filtering level averaged over 5 randomly selected sets of traces.

We can clearly see that distance begins to increase only very slowly with too-high filter levels, and is much less sensitive to suboptimal filter levels. This suggests that a potential subsequent lattice reduction may help to reduce the number of filter levels that an adversary has to try out. However, $\mathbf{t}_0$ will often not be available to an adversary and running with various filter levels is in fact computationally very inexpensive.

Table 2: Accuracy targeting individual bits of $\mathbf{y}$.

Bit Index	0	1	2	3	4	5	6	7
STM32F415RGT6, $N_L = 1M$								
σ	38.79	11.72	11.89	11.65	12.84	13.40	12.78	13.06
SNR_{max}	0.000166	0.001818	0.001767	0.001839	0.001514	0.001390	0.001528	0.001465
Accuracy (%)	49.98	50.64	53.80	54.73	54.21	54.39	54.97	54.19
STM32F303R8T6, $N_L = 1M$								
σ	2.37	2.84	2.66	2.46	2.51	2.47	2.52	2.40
SNR_{max}	0.044281	0.030956	0.035102	0.041014	0.039497	0.040864	0.039298	0.043089
Accuracy (%)	59.10	57.62	58.05	58.84	58.34	58.21	58.12	58.61

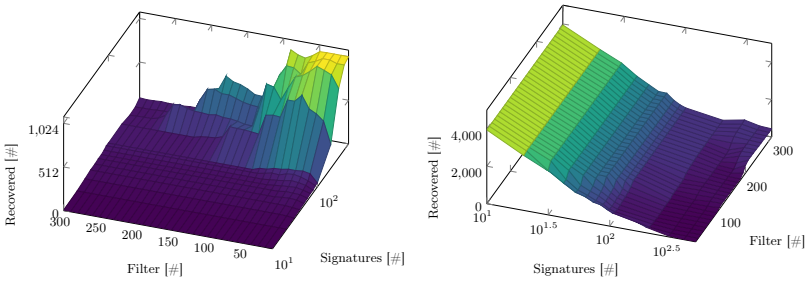


(a) Recovered. (b) Distance.

Fig. 9: Recovered coefficients and distance when targeting individual bits of $\mathbf{y}$ per number of signatures and filtering levels.

A note on applying templates correctly: A naïve approach to bit leakage would be to first compute a success probability for each bit, then selecting the more likely one in each equation, and then building hints by using the *overall* success probability. This works very well for low noise levels, but loses information in the process. The correct approach is to compute the probability to be 0 or 1 for each *individual* bit from the given measurement and the template.

Targeting the HW. We target the HW of the 8 LSB of coefficients of $\mathbf{y}$ (first-order leakage) and the (lower 8-bit) HW of both shares of coefficients of $\mathbf{y}$ (second-order attack). The former corresponds to our simulated results in Figure 4, while the latter corresponds to the Boolean masking in Figure 8. Figure 10 and Figure 11 depicts the results for first-order and second-order leakage per number of equations and filter levels as average over 5 runs. For first-order leakage we observe a mean SNR of 0.20 and require about 300 signatures for full recovery of $\mathbf{s}_1$; for second-order leakage it is 2.07 for the first share and 1.64 for the second share, and we require about 700 signatures to fully recovery $\mathbf{s}_1$. Note that the lower noise level for the second-order attack leads to a more stable BP.



(a) Recovered. (b) Distance.

Fig. 10: Recovered coefficients and distance when targeting the HW of $\mathbf{y}$ per number of signatures and filtering levels.

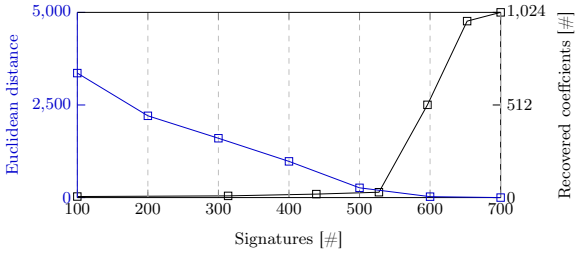


Fig. 11: Recovered coefficients and distance when targeting the HW of both Boolean shares of $\mathbf{y}$ per number of signatures using no filter and one that filters to 300 signatures. Note that this figure does **not** use logarithmically-scaled axes.

5.2 Countermeasures

Because boolean masking leaks more information on the unshared secret, one approach to protect it is to use different masking order for Boolean and arithmetic masking. This countermeasure has already been mentioned in [3] as prime-field masking has been shown to be more resistant to attacks that exploit low-noise leakage [35]. Our work suggests that this is relevant even for increased noise levels. In fact, to be effective, a moderate noise-level is required in the first place (see results of [35]).

We evaluate raising the Boolean masking order by a factor of 2. To understand the overhead of the countermeasure, we implemented it on top of the masked ML-DSA-44 implementation of [11]. We measured the cycle counts of signing a 32 B message on an `STM32F405RGT6` board, which has an ARM Cortex-M4 processor running at 168 MHz and 192 KiB SRAM, and repeat the experiments 15000 times.

Since the implementation [11] is not constant-time and its cycle count fluctuates drastically, we could not measure the cycles directly. To benchmark the overhead as precisely as possible, we fixed the PRNG of this implementation,

Table 3: Overhead of the countermeasure in cycles.

	Total signing	masked_sample_y	seed_y
Original	39118662.27	5488094 (14.02%)	175083 (0.44%)
With Countermeasure	52031577.69	17779727 (34.17%)	701690 (1.35%)

and we modified the rejection loop of the signing procedure such that a signature can be generated by sampling exactly one vector $\mathbf{y}$, i.e., we do not reject signatures. While signatures generated in this manner may or may not be valid, it is sufficient for benchmarking since all the necessary computations for generating one signature were performed. We can thus compute the percentage of the computation affected by our countermeasure and its overhead. The results can be found in Table 3.

As we can see, the cycle count of *masked_sample_y* increases by a factor of 3.23, and accounts for 34% of the total cycle count. In the hedged version of ML-DSA, which is the default per ML-DSA standard, both fresh randomness and randomness derived from the message being signed are used. The proposed masked implementation of [11] uses masked Keccak [12] to derive a random seed from the message, which is subsequently fed into a PRNG. While increasing by a factor of 4, this part of the computation only makes up 1.35% (see *seed_y* in Table 3) of the total cycle counts. In total, our countermeasure increases the cycle count by only a factor of 1.33.

References

1. Akkar, M.L., Bevan, R., Dischamp, P., Moyart, D.: Power analysis, what is now possible... In: Okamoto, T. (ed.) ASIACRYPT 2000. LNCS, vol. 1976, pp. 489–502. Springer, Berlin, Heidelberg (Dec 2000). https://doi.org/10.1007/3-540-44448-3_38
2. Albrecht, M.R., Deo, A., Paterson, K.G.: Cold boot attacks on ring and module LWE keys under the NTT. IACR TCHES **2018**(3), 173–213 (2018). https://doi.org/10.13154/tches.v2018.i3.173-213, https://tches.iacr.org/index.php/TCHES/article/view/7273
3. Azouaoui, M., Bronchain, O., Cassiers, G., Hoffmann, C., Kuzovkova, Y., Renes, J., Schneider, T., Schönauer, M., Standaert, F.X., van Vredendaal, C.: Protecting Dilithium against leakage revisited sensitivity analysis and improved implementations. IACR TCHES **2023**(4), 58–79 (2023). https://doi.org/10.46586/tches.v2023.i4.58-79
4. Bettale, L., Coron, J.S., Zeitoun, R.: Improved high-order conversion from Boolean to arithmetic masking. IACR TCHES **2018**(2), 22–45 (2018). https://doi.org/10.13154/tches.v2018.i2.22-45, https://tches.iacr.org/index.php/TCHES/article/view/873
5. Bos, J.W., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J.M., Schwabe, P., Seiler, G., Stehlé, D.: CRYSTALS - kyber: A cca-secure module-lattice-based KEM. In: 2018 IEEE European Symposium on Security and Privacy, EuroS&P 2018, London, United Kingdom, April 24-26, 2018. pp. 353–367. IEEE (2018). https://doi.org/10.1109/EUROSP.2018.00032, https://doi.org/10.1109/EuroSP.2018.00032
6. Brier, E., Clavier, C., Olivier, F.: Correlation power analysis with a leakage model. In: Joye, M., Quisquater, J.J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 16–29. Springer, Berlin, Heidelberg (Aug 2004). https://doi.org/10.1007/978-3-540-28632-5_2
7. Bronchain, O., Azouaoui, M., ElGhamrawy, M., Renes, J., Schneider, T.: Exploiting small-norm polynomial multiplication with physical attacks application to CRYSTALS-Dilithium. IACR TCHES **2024**(2), 359–383 (2024). https://doi.org/10.46586/tches.v2024.i2.359-383
8. Chari, S., Jutla, C.S., Rao, J.R., Rohatgi, P.: Towards sound approaches to counteract power-analysis attacks. In: Wiener, M.J. (ed.) CRYPTO'99. LNCS, vol. 1666, pp. 398–412. Springer, Berlin, Heidelberg (Aug 1999). https://doi.org/10.1007/3-540-48405-1_26
9. Chari, S., Rao, J.R., Rohatgi, P.: Template attacks. In: Kaliski, Jr., B.S., Koç, Çetin Kaya., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 13–28. Springer, Berlin, Heidelberg (Aug 2003). https://doi.org/10.1007/3-540-36400-5_3
10. 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: 39th IEEE International Conference on Computer Design, ICCD 2021, Storrs, CT, USA, October 24-27, 2021. pp. 583–590. IEEE (2021). https://doi.org/10.1109/ICCD53106.2021.00094, https://doi.org/10.1109/ICCD53106.2021.00094
11. Coron, J.S., Gérard, F., Lepoint, T., Trannoy, M., Zeitoun, R.: Improved high-order masked generation of masking vector and rejection sampling in dilithium. Cryptology ePrint Archive, Report 2024/1149 (2024), https://eprint.iacr.org/2024/1149

12. Coron, J.S., Gérard, F., Montoya, S., Zeitoun, R.: High-order table-based conversion algorithms and masking lattice-based encryption. IACR TCHES **2022**(2), 1–40 (2022). https://doi.org/10.46586/tches.v2022.i2.1-40
13. Coron, J.S., Gérard, F., Trannoy, M., Zeitoun, R.: Improved gadgets for the high-order masking of Dilithium. IACR TCHES **2023**(4), 110–145 (2023). https://doi.org/10.46586/tches.v2023.i4.110-145
14. Dachman-Soled, D., Ducas, L., Gong, H., Rossi, M.: LWE with side information: Attacks and concrete security estimation. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020, Part II. LNCS, vol. 12171, pp. 329–358. Springer, Cham (Aug 2020). https://doi.org/10.1007/978-3-030-56880-1_12
15. Dachman-Soled, D., Gong, H., Hanson, T., Kippen, H.: Revisiting security estimation for LWE with hints from a geometric perspective. In: Handschuh, H., Lysyanskaya, A. (eds.) CRYPTO 2023, Part V. LNCS, vol. 14085, pp. 748–781. Springer, Cham (Aug 2023). https://doi.org/10.1007/978-3-031-38554-4_24
16. Duc, A., Faust, S., Standaert, F.X.: Making masking security proofs concrete (or how to evaluate the security of any leaking device), extended version. Journal of Cryptology **32**(4), 1263–1297 (Oct 2019). https://doi.org/10.1007/s00145-018-9277-0
17. Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schwabe, P., Seiler, G., Stehlé, D.: Crystals-dilithium: A lattice-based digital signature scheme. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2018**(1), 238–268 (2018). https://doi.org/10.13154/TCHES.V2018.I1.238-268, https://doi.org/10.13154/tches.v2018.i1.238-268
18. Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schwabe, P., Seiler, G., Stehlé, D.: CRYSTALS-Dilithium: A lattice-based digital signature scheme. IACR TCHES **2018**(1), 238–268 (2018). https://doi.org/10.13154/tches.v2018.i1.238-268, https://tches.iacr.org/index.php/TCHES/article/view/839
19. Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schwabe, P., Seiler, G., Stehlé, D.: CRYSTALS-DILITHIUM. Tech. rep., National Institute of Standards and Technology (2020), available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions
20. Gierlichs, B., Batina, L., Tuyls, P., Preneel, B.: Mutual information analysis. In: Oswald, E., Rohatgi, P. (eds.) CHES 2008. LNCS, vol. 5154, pp. 426–442. Springer, Berlin, Heidelberg (Aug 2008). https://doi.org/10.1007/978-3-540-85053-3_27
21. Hamburg, M., Hermelink, J., Primas, R., Samardjiska, S., Schamberger, T., Streit, S., Strieder, E., van Vredendaal, C.: Chosen ciphertext k-trace attacks on masked CCA2 secure Kyber. IACR TCHES **2021**(4), 88–113 (2021). https://doi.org/10.46586/tches.v2021.i4.88-113, https://tches.iacr.org/index.php/TCHES/article/view/9061
22. Han, J., Lee, T., Kwon, J., Lee, J., Kim, I., Cho, J., Han, D., Sim, B.: Single-trace attack on NIST round 3 candidate dilithium using machine learning-based profiling. IEEE Access **9**, 166283–166292 (2021). https://doi.org/10.1109/ACCESS.2021.3135600, https://doi.org/10.1109/ACCESS.2021.3135600
23. Hermelink, J., Mårtensson, E., Samardjiska, S., Pessl, P., Rodosek, G.D.: Belief propagation meets lattice reduction: Security estimates for error-tolerant key recovery from decryption errors. IACR TCHES **2023**(4), 287–317 (2023). https://doi.org/10.46586/tches.v2023.i4.287-317
24. Hermelink, J., Ning, K.C., Petri, R., Strieder, E.: The insecurity of masked comparisons: SCAs on ML-KEM's FO-transform. In: Luo, B., Liao, X., Xu, J.,

Kirda, E., Lie, D. (eds.) ACM CCS 2024. pp. 2430–2444. ACM Press (Oct 2024). https://doi.org/10.1145/3658644.3690339

25. Hermelink, J., Pessl, P., Pöppelmann, T.: Fault-enabled chosen-ciphertext attacks on kyber. In: Adhikari, A., Küsters, R., Preneel, B. (eds.) INDOCRYPT 2021. LNCS, vol. 13143, pp. 311–334. Springer, Cham (Dec 2021). https://doi.org/10.1007/978-3-030-92518-5_15
26. Hermelink, J., Streit, S., Mårtensson, E., Petri, R.: A generic framework for side-channel attacks against LWE-based cryptosystems. Cryptology ePrint Archive, Report 2024/1211 (2024), https://eprint.iacr.org/2024/1211
27. Hermelink, J., Streit, S., Strieder, E., Thieme, K.: Adapting belief propagation to counter shuffling of NTTs. IACR TCHES **2023**(1), 60–88 (2023). https://doi.org/10.46586/tches.v2023.i1.60-88
28. Inc., N.T.: Cw1173: Chipwhisperer-lite (2018), https://media.newae.com/datasheets/NAE-CW1173_datasheet.pdf
29. Ishai, Y., Sahai, A., Wagner, D.: Private circuits: Securing hardware against probing attacks. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 463–481. Springer, Berlin, Heidelberg (Aug 2003). https://doi.org/10.1007/978-3-540-45146-4_27
30. Jackson, K.A., Miller, C.A., Wang, D.: Evaluating the security of CRYSTALS-dilithium in the quantum random oracle model. In: Joye, M., Leander, G. (eds.) EUROCRYPT 2024, Part VI. LNCS, vol. 14656, pp. 418–446. Springer, Cham (May 2024). https://doi.org/10.1007/978-3-031-58751-1_15
31. Kannwischer, M.J., Schwabe, P., Stebila, D., Wiggers, T.: Improving software quality in cryptography standardization projects. In: IEEE European Symposium on Security and Privacy, EuroS&P 2022 - Workshops, Genoa, Italy, June 6-10, 2022. pp. 19–30. IEEE Computer Society, Los Alamitos, CA, USA (2022). https://doi.org/10.1109/EuroSPW55150.2022.00010, https://eprint.iacr.org/2022/337
32. Kullback, S., Leibler, R.A.: On Information and Sufficiency. The Annals of Mathematical Statistics **22**(1), 79–86 (1951). https://doi.org/10.1214/aoms/1177729694, https://doi.org/10.1214/aoms/1177729694
33. Kuo, Y., Takayasu, A.: A lattice attack on crystals-kyber with correlation power analysis. In: Seo, H., Kim, S. (eds.) Information Security and Cryptology - ICISC 2023 - 26th International Conference on Information Security and Cryptology, ICISC 2023, Seoul, South Korea, November 29 - December 1, 2023, Revised Selected Papers, Part I. Lecture Notes in Computer Science, vol. 14561, pp. 202–220. Springer (2023). https://doi.org/10.1007/978-981-97-1235-9_11, https://doi.org/10.1007/978-981-97-1235-9_11
34. Lyubashevsky, V.: Fiat-Shamir with aborts: Applications to lattice and factoring-based signatures. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 598–616. Springer, Berlin, Heidelberg (Dec 2009). https://doi.org/10.1007/978-3-642-10366-7_35
35. Masure, L., Méaux, P., Moos, T., Standaert, F.X.: Effective and efficient masking with low noise using small-mersenne-prime ciphers. In: Hazay, C., Stam, M. (eds.) EUROCRYPT 2023, Part IV. LNCS, vol. 14007, pp. 596–627. Springer, Cham (Apr 2023). https://doi.org/10.1007/978-3-031-30634-1_20
36. May, A., Nowakowski, J.: Too many hints - when LLL breaks LWE. In: Guo, J., Steinfeld, R. (eds.) ASIACRYPT 2023, Part IV. LNCS, vol. 14441, pp. 106–137. Springer, Singapore (Dec 2023). https://doi.org/10.1007/978-981-99-8730-6_4

37. Migliore, V., Gérard, B., Tibouchi, M., Fouque, P.A.: Masking Dilithium - efficient implementation and side-channel evaluation. In: Deng, R.H., Gauthier-Umaña, V., Ochoa, M., Yung, M. (eds.) ACNS 19International Conference on Applied Cryptography and Network Security. LNCS, vol. 11464, pp. 344–362. Springer, Cham (Jun 2019). https://doi.org/10.1007/978-3-030-21568-2_17
38. National Institute of Standards and Technology: Submission requirements and evaluation criteria for the post-quantum cryptography standardization process (2016), https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/call-for-proposals-final-dec-2016.pdf
39. National Institute of Standards and Technology: Module-lattice-based digital signature standard. Tech. rep., Department of Commerce, Washington, D.C. (2023), federal Information Processing Standards Publication (FIPS) NIST FIPS 204 ipd. https://doi.org/10.6028/NIST.FIPS.204.ipd
40. National Institute of Standards and Technology: Module-lattice-based digital signature standard (2024), https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.204.pdf
41. National Institute of Standards and Technology: Module-lattice-based key-encapsulation mechanism standard (2024), https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.203.pdf
42. Oliveira, P.A., Viera, A.C., Cogliati, B., Goubin, L.: Uncompressing dilithium's public key. IACR Cryptol. ePrint Arch. p. 1373 (2024), https://eprint.iacr.org/2024/1373
43. Pessl, P., Primas, R.: More practical single-trace attacks on the number theoretic transform. In: Schwabe, P., Thériault, N. (eds.) LATINCRYPT 2019. LNCS, vol. 11774, pp. 130–149. Springer, Cham (Oct 2019). https://doi.org/10.1007/978-3-030-30530-7_7
44. Pessl, P., Prokop, L.: Fault attacks on CCA-secure lattice KEMs. IACR TCHES **2021**(2), 37–60 (2021). https://doi.org/10.46586/tches.v2021.i2.37-60, https://tches.iacr.org/index.php/TCHES/article/view/8787
45. Primas, R., Pessl, P., Mangard, S.: Single-trace side-channel attacks on masked lattice-based encryption. In: Fischer, W., Homma, N. (eds.) CHES 2017. LNCS, vol. 10529, pp. 513–533. Springer, Cham (Sep 2017). https://doi.org/10.1007/978-3-319-66787-4_25
46. 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, Report 2023/1866 (2023), https://eprint.iacr.org/2023/1866
47. Qiao, Z., Liu, Y., Zhou, Y., Zhao, Y., Chen, S.: Single trace is all it takes: Efficient side-channel attack on dilithium. Cryptology ePrint Archive, Report 2024/512 (2024), https://eprint.iacr.org/2024/512
48. Ravi, P., Chattopadhyay, A., D'Anvers, J., Baksi, A.: Side-channel and fault-injection attacks over lattice-based post-quantum schemes (kyber, dilithium): Survey and new results. ACM Trans. Embed. Comput. Syst. **23**(2), 35:1–35:54 (2024). https://doi.org/10.1145/3603170, https://doi.org/10.1145/3603170
49. Shannon, C.E.: A mathematical theory of communication. Bell Syst. Tech. J. **27**(3), 379–423 (1948). https://doi.org/10.1002/J.1538-7305.1948.TB01338.X, https://doi.org/10.1002/j.1538-7305.1948.tb01338.x
50. Tosun, T., Moradi, A., Savas, E.: Exploiting the central reduction in lattice-based cryptography. IEEE Access **12**, 166814–166833 (2024). https://doi.org/10.1109/ACCESS.2024.3494593, https://doi.org/10.1109/ACCESS.2024.3494593

51. Ulitzsch, V.Q., Marzougui, S., Tibouchi, M., Seifert, J.P.: Profiling side-channel attacks on dilithium - A small bit-fiddling leak breaks it all. In: Smith, B., Wu, H. (eds.) SAC 2022. LNCS, vol. 13742, pp. 3–32. Springer, Cham (Aug 2024). https://doi.org/10.1007/978-3-031-58411-4_1