Noise-Tolerant Plaintext-Checking Oracle Attacks

A Soft-Analytic Approach Applied to ML-KEM

Julius Hermelink[1], Erik Mårtensson[2,3,4], Richard Petri[1], and Maggie Tran[2]

[1] Max Planck Institute for Security and Privacy, Bochum, Germany, {julius.hermelink,richard.petri}@mpi-sp.org
[2] Lund University, Lund, Sweden, maggie.tran@eit.lth.se
[3] Advenica AB, Malmö, Sweden
[4] Linköping University, Linköping, Sweden, erik.martensson@liu.se

Abstract. Plaintext-checking oracle (PCO) attacks are among the most prominent types of attacks against NIST's recently standardized ML-KEM. Previous works have drastically reduced the number of queries needed to recover the secret key. Although this number is now close to the information-theoretic bound, current attacks have not yet been adapted to settings with increased noise and highly imperfect oracles. In attacks targeting real-world protected implementations, noisy leakage may lead to oracles that only provide a small advantage over guessing.
This work shows how to efficiently exploit imperfect oracles arising from highly noisy side channels. We present several soft-analytic techniques that enable highly noise-tolerant parallel PCO attacks. These techniques allow for successful attacks in relatively few traces from information that gives just a slight advantage over guessing. Additionally, we extend the generic framework for side-channel information from Eurocrypt 2025 and thereby relate our techniques to previous work.
We then discuss several oracle instantiations that are based on the noisy Hamming weight model. These oracles rely on widely accepted assumptions, are easy to simulate, and allow for fair comparisons between different attacks. Furthermore, they allow taking countermeasures into account, and we evaluate the impact of masking. Our evaluations in these and previous models show that PCO attacks are noise-tolerant on an entirely different scale compared to previous work. These improvements are of an algorithmic nature and orthogonal to the fact that the Fujisaki-Okamoto transform in ML-KEM offers a large attack surface. Finally, we discuss the implications of our findings for protected ML-KEM implementations.

1 Introduction

The National Institute of Standards and Technology (NIST) standardization process for post-quantum cryptography is still ongoing, and the first schemes have been standardized [42]. This includes the lattice-based schemes Module-Lattice-based Key-Encapsulation Mechanism (ML-KEM) [41] and Module-Lattice-based Digital Signature Algorithm (ML-DSA) [40], both relying on a variant of the

Learning with Errors (LWE) problem [56] called Module Learning with Errors (MLWE) [34]. Real-world messaging applications such as Signal and iMessage already employ ML-KEM [33,12] and Cloudflare reported that 52% of the connections relied on ML-KEM in December 2025 [3]. In the area of side-channel attacks on LWE-based schemes, one of the more prominent and investigated types of attacks is the so-called Plaintext-Checking Oracle (PCO) attacks (see, e.g., [54]). These attacks exploit that ML-KEM relies on a Fujisaki-Okamoto (FO) transform [14,15] to derive a Key Encapsulation Mechanism (KEM) from a Public-Key Encryption (PKE), thereby achieving IND-CCA (Indistinguishability under Chosen-Ciphertext Attack). In practice, this means that the KEM version not only decrypts a ciphertext, but also re-encrypts it. The re-encrypted ciphertext is compared with the submitted one, and the ciphertext is rejected if they do not match, preventing Chosen-Ciphertext Attacks (CCAs). However, side channels might leak information during the re-encryption or the comparison, thereby allowing for CCAs. PCO attacks exploit leakage on the decrypted message, which enables key mismatch attacks [13,55] on the underlying PKE from which the KEM is derived using the FO-transform.

While generic in theory, PCOs have mainly been realized through power or electromagnetic radiation side channels. Most works have focused on reducing the number of queries in key mismatch attacks, corresponding to the number of traces required in a Side-Channel Attack (SCA). Fluhrer introduced key mismatch attacks against lattice-based schemes in [13]. They were applied to ML-KEM[5] in [51], requiring 1855 queries for Kyber768. Subsequent work in [52] reduced this to 1776 queries (as clarified in [19]), and [19] lowered the requirement to 1588 queries. The authors of [20] improved the query complexity, explained how the balance between the cost of queries and post-processing in a parallel mismatch attack can be improved, and thereby showed that giving the adversary as few as two (perfect) mismatch queries is enough to break the security level of ML-KEM.

Several works have also investigated practical attacks: Binary PCO attacks, i.e., instantiations of key mismatch attacks using a side channel on ML-KEM and other lattice-based schemes were studied in [55]. More recent work [53] showed that multiple coefficients can be targeted in parallel if the adversary can distinguish more than two messages (a concept also proposed in [57]), effectively reducing the number of required queries by a factor of N, compared to [55], when targeting N positions in parallel.

The number of traces is a commonly used metric in SCAs: For masking countermeasures [6], we can expect the leaked information to decrease exponentially in the number of shares *if sufficient noise is present* [10]. Therefore, the number of traces an attack requires is a relevant metric, as it is expected to grow exponentially with the number of shares in many attacks. However, noise conditions are another important factor to consider. Without noise, it is unsurprising that higher-order attacks succeed, as (Boolean) masking may provide little or no reduction of leaked information (see, e.g., [37]). Boolean masking (in contrast to arithmetic masking) offers particularly little benefit in low noise

[5] We do not differentiate between Kyber and ML-KEM.

conditions [37,25]. Moreover, not taking noise conditions into consideration when designing countermeasures can lead to devastating attacks [26,29].

In PCO attacks, adversaries may target a comparably long-running computation. They are only required to distinguish 2^N messages, where N is the number of targeted positions in parallel and can be set to 1. Therefore, attacks under low-noise conditions or targeting unprotected implementations achieve high accuracy, especially when evaluations are performed on platforms with relatively low noise rather than real-world devices. As a result, handling higher noise levels or imperfect oracles has not been necessary for successful attacks, even on physical devices and masked implementations (see, e.g., [62,60,53]). However, real-world protected implementations are expected to exhibit higher noise levels and employ various countermeasures, particularly given that prior work has highlighted the threat posed by PCO attacks.

While some works have considered imperfect oracles, the area is relatively underdeveloped: Dong and Guo [9] recently showed how to deal with soft information in PCO attacks against HQC [1] using an enhanced information set decoding algorithm. However, these techniques do not translate to PCO attacks on ML-KEM, and the oracle itself is assumed to give hard information. For ML-KEM512, the approach of [58] has an error rate of 0.25 coefficients per key for an oracle accuracy of $\alpha = 0.9$. At the same accuracy level, $\alpha = 0.9$ (considered to be a high-noise setting), an error rate of 0.646 coefficients per ML-KEM1024 key for targeting $N = 8$ positions is reported by [35] while already relying on majority voting and requiring 2041.6 queries. For comparison, our binary PCOs with $\alpha = 0.51$ can be targeted with an error rate of less than 0.001 per subkey; for $N \in \{4, 8\}$ and $\alpha \in \{0.9, 0.95\}$, we observe no failure in 10000 experiments. As a result, we require only approximately 700 queries at $N = 8$ and $\alpha = 0.9$, and roughly 1000 traces at $\alpha = 0.7$ compared to the around 7000 queries reported in [35]. Further, attacks in [35] require significantly more queries to achieve relatively low error rates, particularly as α decreases (see Table 2). Moreover, these works assume a "tournament-based" oracle. Such an oracle is instantiated by comparing how well, e.g., a template fits two messages and choosing the more likely one until no more potential messages are left – just as in a tournament; the remaining message is assumed to be the correct one. This approach may result in a considerable information loss (see Figure 6): All information apart from which message is the most likely is being discarded. Information often obtained in common attacks cannot be modeled in this abstraction; for example, in a standard template attack [6]—which does not require a clone device in the PCO attack setting—the adversary obtains information on all messages.

In summary, targeting the FO-transform allows for distinguishing messages with high accuracy—even in masked implementations (see, e.g., [62]). Previous attacks assume similar accuracies and focus on low-noise settings, whereas real-world implementations aim to reduce the attack surface using masking and hiding countermeasures under sufficient noise. Designing effective countermeasures requires understanding how much of an advantage in distinguishing messages translates into the number of required traces for a successful attack.

Moreover, higher-order masking is costly to implement, and the assumption that these attacks quickly deteriorate for imperfect oracles may lead to insufficient countermeasures. Decryption Failure (DF) attacks [18] have been shown to be highly noise-tolerant [27,24][6], yet handling highly imperfect and noisy oracles in PCO attacks remains an open question. This question is orthogonal to the fact that the FO-transform offers a large attack surface.

Our contribution. We present an information-theoretic approach to PCO attacks. Our new abstraction for oracles allows for improved modeling. Furthermore, we employ soft-analytic [63] techniques to deal with noisy and imperfect PCOs. To our knowledge, this is the first soft-analytic approach to PCO attacks[7].

First, we present several new attacks that range from being computationally expensive but optimally exploiting all information to being very efficient but losing some information. The optimal attack keeps track of the joint probability distribution of all targeted key coefficients conditioned on the information given by the oracle. The most efficient version only keeps track of an approximation of the marginals of the joint probability distribution. The former is computationally expensive – but easily carried out on a laptop for ≤ 5 positions in parallel in seconds – while the latter uses the information less efficiently. As a trade-off, we provide a Belief Propagation (BP) instantiation that (better) approximates the marginals of the joint probability distribution. For the optimal attack, we discuss different ways to choose oracle queries – again providing a trade-off between loss in computational and information-theoretic efficacy. For imperfect oracles, our attacks require fewer queries compared to previous works.

Secondly, we discuss several oracle instantiations and define two new models for evaluating PCO attacks. These oracles allow for a unified evaluation of attacks while being rooted in a realistic, long-standing assumption, namely the noisy Hamming weight (HW) model. Our oracles assume that the adversary is able to obtain the noisy HW of k uniformly random intermediate values during the re-encryption; this can be done by, e.g., targeting the last layer of the Number Theoretic Transform (NTT). Each value in turn corresponds to $t + 1$ shares that mask a secret, or the value itself in an unmasked implementation. We provide an information-theoretic analysis of the previously used tournament-based oracle and discuss such evaluations for the noisy HW-based models.

Thirdly, we evaluate our attacks in the presented oracle models. Overall, the optimal and the BP-based attacks have (almost) no failures when using a sufficiently strict abort criterion. Our criterion, the entropy of the subkey being less than 0.001, resulted in a single failure in all settings in the preliminary evaluations and none in the final evaluations. Stricter abort criteria allow for a further reduction in the failure rate. Moreover, we show that even naïve soft-analytic approaches achieve considerable improvements for imperfect oracles. We target an abstract tournament-based oracle and simulate leakage for first-, second-, and third-order leakage in masked implementations. In all settings,

[6] The fault attack obtains the same information as decryption-failure attacks.
[7] Apart from instantiating an oracle using BP.

our attacks achieve failure rates close to zero under drastically increased noise compared to previous attacks. For evaluation on a physical device, we target the assembly-optimized NTT in the masked mkm4 implementation [23] on an STM32F415 chip using a LeCroy Waverunner 9254m oscilloscope. We show that a standard template attack [6] on unsigned values of the last NTT layer is sufficient to fully recover the secret key using our techniques—using no heavy machinery like neural networks.

In addition, in Appendix A, we analyze the shortcomings of the generic framework for side information presented in [28]. We note that the definition of distribution hints always assumes independence and does not allow for hints that are correlated. To address this limitation, we introduce *joint distribution hints*, extending the solver of [28] to our definition with an optimized implementation[8].

Overview of results. For binary PCO attacks, we show that an adversary may successfully use a tournament oracle with $\alpha = 0.51$ at less than one error per 10000 samples[9] without repeating queries. To fully recover an ML-KEM1024 key when targeting $N = 8$ positions, we require 1529.6 queries at $\alpha = 0.5$, 706.56 at $\alpha = 0.9$, and 944.64 at $\alpha = 0.7$ ([35] requires over 7000 queries at $\alpha = 0.7$). The noisy HW-based oracles depend on the number of values we target in the NTT during the re-encryption and the standard deviation σ of the noise. These values are targeted individually, and we exploit no relations between them. For 100 targeted values, we require 238 traces at $\sigma = 3$ and 1835 at $\sigma = 10$. Targeting the last layer of the NTT in a second-order attack requires just 742 traces for full key-recovery at $\sigma = 1.0$. Furthermore, we show that even with just a *single intermediate* for which both shares leak, we may on average recover the secret key with less than 270000 traces at $\sigma = 1.0$ in a first-order masked implementation.

Practical implications. These attacks are known to not require a clone device (see, e.g., [53]). Moreover, the attack surface is large and allows for targeting many values that hold algebraic relations, e.g., during the NTT or Keccak. These have previously been exploited in different contexts [50,47,31,22]. Furthermore, the work of [62] achieves very high success rates for a masked implementation. Our improvements in this work are orthogonal to the large attack surface. We demonstrate that even with a tiny attack surface, these attacks are highly noise tolerant and potentially devastating. Thereby, we show that these attacks present an even greater threat than previously believed.

Concurrent work. In a concurrent work [49], the authors propose using the Sequential Probability Ratio Test to improve upon majority voting in PCO attacks. The authors combine their approach with improvements that are orthogonal to our work, which can likely be adapted to our work. They compare against two methods presented in this work[10]: against the optimal method[11] in the binary

[8] For our evaluations, we provide a more intuitive Python implementation.
[9] This means very little information; $\alpha = 0.5$ corresponds to no information for $N = 1$.
[10] Note that a preprint of our work had been made available.
[11] Note the conditions for optimality outlined in Section 3.2.

case and against our BP-based method for $N = 8$. In the binary case, their combined improvements have a slight advantage over our method; when targeting 8 key coefficients at once, our method substantially outperforms their approach in terms of numbers of queries.

In [2], the authors evaluate and compare several concrete instantiations for PCO attacks on ML-KEM including the one presented in this work. They provide concrete measurements and argue that soft-analytic approaches—such as the one presented in this paper – have substantial advantages.

In [21], it is shown that DF attacks may require fewer queries than PCO attacks. They compare with our work in the binary case, where they require fewer queries. Note that our improvements mainly come into play when targeting multiple key coefficients at once. Moreover, our approach allows for extracting far more information than available in their model, and we allow fewer errors in key recovery; for a similar error rate, we achieve very comparable results. In addition, it is unclear if the underlying abstractions allow for a fair comparison: while both rely on a tournament-based approach, the actual leakage obtained differs, and this may result in incomparable accuracies. In summary, the PCO attacks still require far fewer queries than DF attacks, and how to compare the binary case is an open question. In this work, we also argue that a (slightly corrected) definition from an earlier unpublished version of [21] is the correct definition to distinguish between these types of attacks.

Open source. Our simulations are written in Python and rely on PQClean [32]. The generic BP-based solver described in Appendix A.1 is written in Rust and extends the implementation of [28]. We do not provide an implementation for the greedy solver described in Appendix A.2, nor do we provide evaluations in more general settings for the BP-based solver; these are left for future work. Our implementations are open-source and can be found under `https://github.com/juliusjh/sasca_imperfect_pc_oracles`.

2 Preliminaries

ML-KEM has been standardized by NIST as the first post-quantum KEM, and PCO attacks are among the most notable side-channel attacks against ML-KEM. We show that soft-analytic techniques enable far more noise-resistant PCO attacks and require far fewer traces, compared to previous works. We first introduce notation, then give an overview of ML-KEM [41], PCO attacks, and a Soft Analytical Side-Channel Attack (SASCA) [63].

2.1 Notation

Vectors and matrices are denoted in bold, e.g., $\mathbf{a}$ and matrices are capitalized. For values, e.g., $m \in \mathbb{Z}$, the notation m_i denotes the i-th bit. The secret, usually an N-dimensional subkey of the ML-KEM key, is denoted by $\mathbf{s}$. It is assumed to be drawn from a centered binomial distribution $\mathbf{B}_\eta$ with parameter η, i.e., the

distribution arising from sampling 2η bits and subtracting η from the number of sampled ones. We write PCOs as $\mathcal{O}$, which can be queried for a vector $\mathbf{a}$, representing smaller-equal inequalities in the secret. The oracle responds with a message m that can be interpreted as $\mathbf{m} = (m_0, \ldots, m_{N-1})$, where 1 corresponds to "true". By $[K]$ we mean $\{0, \ldots, K-1\}$. We often interpret polynomials in $R_q = \mathbb{F}_q[x]/(x^n - 1)$ as vectors when the multiplicative structure is irrelevant.

2.2 ML-KEM

ML-KEM is the first Post-Quantum (PQ) KEM standardized by NIST. Its security is based on the MLWE problem, a variant of the LWE problem. LWE asks to recover $\mathbf{s} \in \mathbb{F}_q^k$ from samples $(\mathbf{a}, \mathbf{as} + e) \in \mathbb{F}_q^k \times \mathbb{F}_q$ where $\mathbf{a}$ is uniformly random, and e follows a "small" error distribution. In practice, both $\mathbf{s}$ and e are often sampled from the same distribution and the number of samples usually equals the dimension of $\mathbf{s}$. The problem can be written in matrix form as recovering $\mathbf{s}$ given $\mathbf{As} + \mathbf{e}$ and $\mathbf{A}$. In the MLWE problem, the coefficients of $\mathbf{s}$, $\mathbf{a}$, and $\mathbf{e}$ are chosen from R_q^k where $R_q = \mathbb{F}_q[x]/(x^n - 1)$, enabling fast multiplication using an NTT. For $k = 1$, this gives the Ring Learning with Errors (RLWE) problem; for $n = 1$, we recover the LWE problem. Solving MLWE – for parameter sets where it is feasible – is usually done by seeing it as an LWE problem.

In ML-KEM, $\mathbf{s}, \mathbf{e}$ are sampled from $\mathbf{B}_\eta$, with η set to 3, 2 and 2 for security levels 512, 768 and 1024, respectively. The value of n is set to 256 and $q = 3329$. The value of k is 2, 3, or 4, resulting in $\mathbf{e}, \mathbf{s}$ being of dimension 512, 768, 1024, respectively, corresponding to the security levels.

ML-KEM first defines an IND-CPA-secure PKE, then uses an FO transform to obtain an IND-CCA2-secure KEM. Intuitively, the KEM deterministically encrypts a randomly sampled message $\mathbf{m}$, then, instead of just decrypting the ciphertext, the other party decrypts and then re-encrypts the message. This yields a re-computed ciphertext which is then compared against the submitted one. A shared secret is returned only if both ciphertexts match, thereby preventing CCAs. However, if a side channel allows observing the re-encryption or the comparison operation, the scheme becomes vulnerable to CCAs again.

Figure 2 shows a simplified version of ML-KEM that relies on the internally defined PKE shown in Figure 1. In Figure 2, re-encryption and comparison operations are highlighted. In Figure 1, we highlight the generation of $\mathbf{r}$; this value is transformed using an NTT – an operation that will become relevant in Section 4.2; for simplicity, we do not include the NTT or the Inverse Number Theoretic Transform (INTT).

The PKE. The key generation samples a public matrix $\mathbf{A}$ uniformly random but deterministically from a seed ρ; secrets $\mathbf{e}, \mathbf{s}$ are sampled from a binomial distribution with parameter $\eta = \eta_1$. The public key, given by $\mathbf{t} = \mathbf{As} + \mathbf{e}$ and the seed ρ, and the secret key, given by $\mathbf{s}$, are returned as a key pair. The encryption retrieves the public key from ρ, deterministically samples vectors $\mathbf{r}_1, \mathbf{r}, \mathbf{e}_1 \in R_q^k$, and a value $e_2 \in R_q$ from a binomial distribution with parameter η_1 (for $\mathbf{r}_1$)

Algorithm 1 PKE.KeyGen

Input: Seeds ρ, σ
Output: Secret Key $\mathtt{sk}$, Public Key $\mathtt{pk}$

$\mathbf{A} \in R_q^{k \times k} \xleftarrow{\$} \mathcal{U}(\rho)$
$\mathbf{e}, \mathbf{s} \in R^k \xleftarrow{\$} \mathcal{B}_{\eta_1}(\sigma)$
$\mathbf{t} \leftarrow \mathbf{A}\mathbf{s} + \mathbf{e}$
return $\mathtt{pk} = (\mathbf{t}, \rho), \mathtt{sk} = \mathbf{s}$

Algorithm 2 PKE.Dec

Input: $\mathtt{sk} = \mathbf{s}$, $\mathtt{ct} = (c_1, c_2)$
Output: Decrypted message m'

$\mathbf{u}', v' \leftarrow \text{Decompress}(\mathtt{ct})$
$m_{\text{pol}} \leftarrow v' - \langle \mathbf{s}, \mathbf{u}' \rangle$
$m' \leftarrow \text{Compress}(m_{\text{pol}}, 1)$
return m'

Algorithm 3 PKE.Enc

Input: $\mathtt{pk} = (\mathbf{t}, \rho)$, m, coins $\mathtt{coins}$
Output: Ciphertext $\mathtt{ct} = (c_1, c_2)$

$\mathbf{A} \in R^{k \times k} \xleftarrow{\$} \mathcal{U}(\rho)$
$\mathbf{r}_1 \in R^k \xleftarrow{\$} \mathcal{B}_{\eta_1}(\mathtt{coins})$
$\mathbf{r} \in R^k \xleftarrow{\$} \mathcal{B}_{\eta_2}(\mathtt{coins})$
$\mathbf{e}_1 \in R^k \xleftarrow{\$} \mathcal{B}_{\eta_2}(\mathtt{coins})$
$e_2 \in R \xleftarrow{\$} \mathcal{B}_{\eta_2}(\mathtt{coins})$
$m_{\text{poly}} \leftarrow \text{Decompress}(m, 1)$
$\mathbf{u} \leftarrow \mathbf{A}^\top \mathbf{r} + \mathbf{e}_1$
$v \leftarrow \langle \mathbf{t}, \mathbf{r} \rangle + e_2 + m_{\text{pol}}$
$c_1 \leftarrow \text{Compress}(\mathbf{u}, d_u)$
$c_2 \leftarrow \text{Compress}(v, d_v)$
return $\mathtt{ct} = (c_1, c_2)$

Fig. 1: Simplified PKE defined by ML-KEM. The recovery of m' and the sampling of $\mathbf{r}$ are highlighted.

and η_2 (for the remaining vectors and e_2). The bit message m is encoded to a polynomial m_{pol} using the decompression function $\text{Decompress}(x, d) = \lceil xq/2^d \rfloor$ with $d = 1$. This means that 0 bits are mapped to $0 \in \mathbb{F}_q$ and 1 bits are mapped to $\lceil q/2 \rfloor$. To encrypt the message, $\mathbf{u} = \mathbf{A}^\top \mathbf{r} + \mathbf{e}_1 \in R_q^k$ and $v = \mathbf{t}^\top \mathbf{r} + e_2 + m_{\text{pol}} \in R$ are computed and compressed to c_1, c_2, respectively, giving the ciphertext.

To decrypt a ciphertext back to a message m', $\mathbf{u}$ and v are recovered from the compressed version given by the ciphertext. Then, m'_{pol} is computed as $\text{Compress}(v - \mathbf{s}^\top \mathbf{u}, 1)$. The compression function is defined as

$$\text{Compress}(x, d) = \left\lceil x \frac{2^d}{q} \right\rfloor \bmod 2^d, \tag{1}$$

i.e., it maps coefficients in $\{0, \ldots, q-1\}$ to 0 if closer to 0 than to $q/2$. With high probability, this results in $m = m'$ as $v - \mathbf{s}^\top \mathbf{u} = m_{\text{pol}} + \mathbf{e}^\top \mathbf{r} - \mathbf{s}^\top \mathbf{e}_1 + e_2$, and the noise term polynomial $\mathbf{e}^\top \mathbf{r} - \mathbf{s}^\top \mathbf{e}_1 + e_2$ is small.

The KEM. The KEM relies on the PKE: the secret key is amended by the public key, its hash, and a rejection value z. The encapsulation randomly samples a message encrypting it with coins derived from the message and the public key's hash. The future shared secret is derived from the ciphertext, the message's hash and the public key. During decapsulation, the ciphertext is decrypted to recover the message m, allowing the other party to derive the same randomness from m and use the public key to re-encrypt the message into an honestly generated ciphertext $\mathtt{ct}'$. Only if $\mathtt{ct}$ equals $\mathtt{ct}'$ is the shared secret returned.

In the following, we explain and evaluate our and previous attacks on ML-KEM768, i.e., $k = 3$, ignoring the algebraic structure, we see polynomials as

Algorithm 4 KEM.KeyGen

Input: Randomness seeds ρ, σ
Output: Public key $\mathtt{pk}$, secret key $\mathtt{sk}$

1: $z \xleftarrow{\$} \mathcal{U}()$
2: $\mathtt{pk}, \mathtt{sk}_{\text{pke}} \leftarrow \text{PKE.KeyGen}()$
3: $h = \text{H}(\mathtt{pk})$
4: $\mathtt{sk} \leftarrow (\mathtt{sk}_{\text{pke}}, \mathtt{pk}, h, z)$
5: **return** $\mathtt{pk}, \mathtt{sk}$

Algorithm 5 KEM.Encaps

Input: $\mathtt{pk}$
Output: Ciphertext $\mathtt{ct}$, shared secret $\mathtt{K}$

1: $m \xleftarrow{\$} \mathcal{U}()$
2: $\bar{\mathtt{K}}, \mathtt{r} \leftarrow \text{G}((m, \text{H}(\mathtt{pk}))$
3: $\mathtt{ct} \leftarrow \text{PKE.Enc}(\mathtt{pk}, \text{m}, \mathtt{r})$
4: $\mathtt{K} \leftarrow \text{KDF}((\bar{\mathtt{K}}, \text{H}(\mathtt{ct})))$
5: **return** $\mathtt{ct}, \mathtt{K}$

Algorithm 6 KEM.Decaps

Input: $\mathtt{sk} = (\mathtt{sk}_{\text{pke}}, \mathtt{pk}, h, z)$, $\mathtt{ct}$
Output: Shared secret $\mathtt{K}$

1: $m' \leftarrow \text{PKE.Dec}(\mathtt{sk}_{\text{pke}}, \mathtt{ct}_{\text{pke}})$
2: $\bar{\mathtt{K}}', \mathtt{r}' \leftarrow \text{G}((m', h))$
3: $\mathtt{ct}' \leftarrow \text{PKE.Enc}(\mathtt{pk}, m', \mathtt{r}')$
4: $b \leftarrow \text{Compare}(\mathtt{ct}, \mathtt{ct}')$
5: **if** b **then**
6: **return** $\mathtt{K} = \text{KDF}(\mathtt{K}', \text{H}(\mathtt{ct}))$
7: **else**
8: **return** $\mathtt{K} = \text{KDF}(z, \text{H}(\mathtt{ct}))$
9: **end if**

Fig. 2: Simplified version of ML-KEM. G and H are hash functions. The re-encryption and comparison operations are highlighted.

vectors of dimension 768 over $\mathbb{F}_q$, and the secret sampled from $\mathbf{B}_\eta$, $\eta = 2$ with the support $\{-2, -1, \ldots, 2\}$.

2.3 PCO Attacks

The terms key mismatch attacks and PCO attacks are not always used consistently. In this work, we call an attack that exploits a side channel to carry out a key mismatch attack a PCO attack. Key mismatch attacks were first introduced by Fluhrer [13] in 2016 and later studied in the context of ML-KEM in [51] and [52], and more recently in [19] and [20]. In [55], Ravi et al. introduced PCO attacks against several lattice-based KEMs. These attacks have since seen many improvements, e.g., in [58,53,57]. The authors of [62] showed that even masked implementations may be highly vulnerable. They rely on the observation that one can select a subkey and choose ciphertexts that decrypt to messages depending solely on the subkey. Concretely, if we set $\mathbf{u} = (208x^0, 0, 0) \in R_q^3$ and $v = t \cdot 208 = t \cdot 208x^0 + 0x^1 + \cdots + 0x^{n-1}$ for some t, then

$$v - \mathbf{u}^\top \mathbf{s} = (t \cdot 208 - 208s_0, -208s_1, \ldots, -208s_{n-1}) \in R_q \tag{2}$$

is compressed, i.e., "recovered" in line 3 of Algorithm 2 (see highlight), to

$$m' = \left(\left\lceil \frac{2}{q} \left(t \cdot 208 - 208 s_0 \right) \right\rfloor, 0, \ldots, 0 \right) = \left(\left\lceil \frac{416}{q} \left(t - s_0 \right) \right\rfloor, 0, \ldots, 0 \right). \tag{3}$$

Let us fix $t = 3$. Then $m'_0 = 0$ implies that $s_0 > -2$, while $m'_0 = 1$ implies that $s_0 \leq -2$. If we let $t = 4$, then $m'_0 = 0$ implies that $s_0 > -1$, while $m'_0 = 1$ implies

that $s_0 \leq -1$; and so on. If a side channel allows us to observe the message, then we may learn the value of s_0 and then vary the position of the value in v, and then the positions of the 1 polynomial in $\mathbf{u}$, to recover more coefficients of the components of $\mathbf{s}$. This is called a *binary PCO attack*.

Parallel PCOs and Tournament oracles. In a *parallel PCO attack*, proposed against ML-KEM in [60], several coefficients of v are set to $t_i \cdot \lceil q/16 \rfloor$. The adversary then learns information about all these secret key coefficients in parallel. However, setting N values results in 2^N messages that need to be distinguished using the side channel. In PCO attacks, the side channel is usually abstracted as an oracle that returns N positions of the message, i.e., N inequalities, given a query represented by a vector $\mathbf{a}$ representing the right side of the inequalities.

The work of [52] proposes using a binary decision tree and the authors of [60] and [53] both propose using binary decision trees for parallel PCO attacks, where [53] proposes an optimized tree. In [53], the authors call this a tournament approach, as the side-channel information for messages is compared pairwise, always keeping the current best-matching candidate – essentially a 2^N participant tournament leading to a winning message. This leads to the abstraction used, e.g., in [53]: the oracle returns the correct message with probability α and a random different message with probability $1 - \alpha$.

Query performance and noise resistance. In [19], 1588 queries are required to fully recover the secret key with a binary oracle. Their result is close to the Shannon entropy bound of 1560 queries. However, assuming a key mismatch attack, they do not consider an oracle that can give incorrect answers.

The PCO attack of [53] achieves an accuracy of 1 when recovering the message from side-channel information in a tournament-based setting using traces from a physical attack. In case of higher noise, they refer to the methods of [44,58]. These works propose recovering a partially correct key that is then checked and corrected using additional oracle queries. In [58], recovering an ML-KEM512 key with a few errors using a binary oracle (i.e., $N = 1$) requires over 3000 queries at $\alpha = 0.96$, which is considered low accuracy. For $\alpha = 0.90$, more than 7700 queries are needed with an average of 0.25 incorrect coefficients per key.

The work of [35] handles noisier oracles while requiring fewer queries. They propose using so-called "grafted trees" – a multiphase approach improving upon and sharing similarities with majority voting. Their proposal first attempts to recover a subkey from an imperfect oracle. Then, they construct an improved tree based on the information in the first round. Finally, the subkey is recovered if the results from the first two rounds match; otherwise, they perform an additional round and use majority voting. Targeting $N = 8$ coefficients of ML-KEM1024, they achieve an error rate of 0.351 coefficients per recovered key with 1332 queries at $\alpha = 0.95$; and an error rate of 0.646 coefficients with 2021 queries at $\alpha = 0.90$.

In this work, we show that our soft-analytic approach drastically reduces error rates while dealing with much higher noise levels. For example, for $N = 1$, we can handle $\alpha = 0.51$ with an error rate of 0; for $\alpha = 0.9$, we require merely

10368.0 queries for ML-KEM768 with 0 incorrect coefficients. For $N = 8$ setting and $\alpha = 0.5$, an average of 1113.6 queries at an error rate of 0; the error rate is also 0 for 10000 experiments at $\alpha = 0.1$. Moreover, the number of queries can be further reduced by relaxing the abort criteria and allowing a few errors.

2.4 Soft-Analytic Side-Channel Attacks

Side-channel attacks often recover noisy information on several intermediates. The seminal work of [63] proposes using "soft-analytic" methods to recover the secret key from such information – in contrast to "hard" algebraic methods. Recovery from noisy intermediates under algorithmic knowledge can be viewed as a decoding problem, leading the authors to propose techniques such as BP. These methods can also be combined with algebraic techniques as shown, e.g., in [24] for targeting ML-KEM.

BP was first described in the context of Low-Density Parity Check (LDPC) codes by Gallager [16] and in a more generic way in [46]. For a thorough introduction, we refer to [36]. Given a factorization of the joint density function as a *factor graph*, BP computes or approximates the marginals. A factor graph consists of two types of nodes: *variable nodes* and *factor nodes*. The former represent unknown variables, the latter represent the relations between variable nodes.

For a vector of Random Variable (RV)s, $\mathbf{X} = (X_0, \ldots, X_{n-1})$, with joint density $p(\mathbf{X})$ and $f_i : I_i \to [0, 1] \subseteq \mathbb{R}$ such that $p(\mathbf{X}) = \prod_{i=0}^{n'} f_i(I_i)$ for $n' < n$ and $I_i \subseteq \{X_0, \ldots, X_{n-1}\}$, BP approximates the marginals for p, i.e., approximates $Z_i(x) = \sum_{\mathbf{x}, x_i = x} p(\mathbf{x})$ The components of $\mathbf{X}$ are represented by the variable nodes, the functions f_i by factor nodes. We denote the message sent from variable node i to factor node j by $p_{i,j}$, and the message from factor node j to variable node i by $q_{j,i}$. The variable nodes compute $p_{i,j}(x) \leftarrow \prod_{j', j' \neq j} q_{j',i}(x)$. The update from the factor nodes is computed as $q_{j,i}(x) \leftarrow \sum_{\mathbf{x}, x_i = x} f_j(\mathbf{x}) \prod_{i', i' \neq i} q_{j,i'}(x)$. In simpler terms: each node computes the update for a node based on the message from all *other* connected nodes and the relation it represents.

If the graph is acyclic, BP is a reordering of the chain rule and thus is exact. For cyclic graphs, BP is called *loopy belief propagation* and offers only an approximation. In SCA, loopy BP has proven to be a valuable tool, often providing good approximations (see, e.g., [39] for an overview).

In this work, we provide three different approaches to handle side-channel information from an oracle: a computationally intensive but optimal updating of the joint density, an efficient but rough approximation, and a trade-off between the two, offered by our BP. Further, we propose an extension to [28] that includes a BP instantiation. However, our main algorithm does not rely on BP; it instead tracks the full joint probability distribution. While not using BP, we argue that this should clearly still be seen as a soft-analytic technique.

3 Exploiting Imperfect PCOs

In this work, we propose a soft-analytic [63] approach to PCO- attacks. Instead of processing hard information and considering that it could be incorrect, we

consider the key coefficients to be random variables and keep track of their probability distributions. Thereby, noise-tolerance and information-theoretic considerations are intrinsic to these attacks. In Section 5, we show that even the naïve updating performs better than non-SASCA approaches. We believe that this is clear evidence for the advantage of soft-analytic techniques.

Oracle abstraction. We adopt the ciphertext construction from previous works, as described in Section 2.3, where *querying the oracle for* $\mathbf{a}$ means constructing a chosen ciphertext with N positions being non-zero leading to one of 2^N possible messages, depending on $\mathbf{a}$, and submitting it to the device under attack. The *actual message* m_{actual} is the unique value in $\left[2^N\right]$ that fulfills $s_i \leq a_i \Leftrightarrow m_i = 1 \ \forall i \in [N]$, where s_i denotes secret coefficients and m_i are the corresponding message bits. An oracle response is a probability distribution on all possible messages $m \in \left[2^N\right]$.

Definition 1. *We call a random variable* $\mathcal{O}(\mathbf{a}) = (p_0, \ldots, p_{2^N-1})$*, where* $\sum_{j=0}^{2^N-1} p_i = 1$*, a* distribution oracle *and associate with it the probability distribution* $\left\{m : p_m \mid m \in \left[2^N\right]\right\}$.

In the following, by oracle, we mean distribution oracle, and we implicitly assume that the Mutual Information (MI) between oracles and the respective secret key is non-zero. Whenever an oracle returns an actual message and not a probability distribution, we interpret the message as the probability distribution with probability one for that message.

For example, the initial query for $N = 3$ would typically be $\mathbf{a} = (0, 0, 0)$, that is, asking for all three coefficients whether they are smaller than or equal to 0. Then, for a subkey with the first coefficient smaller than or equal to 0 and the next two coefficients positive, $m_{\text{actual}} = 4$, which we, by abuse of notation, also denote as $m_{\text{actual}} = (0, 0, 1)$. A response could be the distribution $\mathcal{D}$ given by

$$P(m = m') = \begin{cases} \frac{1}{20} & m' \in \{0, 3, 5, 6, 7\} \\ \frac{1}{4} & m' \in \{1, 2, 4\} \end{cases}. \tag{4}$$

We denote querying and responding by $\mathcal{D} = \mathcal{O}(\mathbf{a})$.

Terminology. By the *joint probability distribution*, we mean the joint probability distribution of all key coefficients given all (previous) queries. By the *marginals* we mean the marginals of the joint probability distribution, i.e., the distribution for individual key coefficients given all previous queries. We often only work with an approximation of the marginals. *Targeted key coefficients*, or sometimes just key coefficients, refer to the coefficients of the subkey currently being targeted – recall that we iteratively recover subkeys of dimension N.

High-level overview. In all our attacks, we target N positions of the secret key in parallel. We query the oracle about the current N positions until we recover them and then proceed to the next block of N positions. We describe how to recover one such block – the remaining positions are recovered the same way.

We keep track of (approximations of) the marginal distributions of the joint probability of the key coefficients conditioned on all previous oracle responses. These distributions serve as a basis for selecting new queries to the oracle. After each update, we check whether the sum of entropies of all margins is smaller than the *entropy bound* h_{abort}; in that case we set the key coefficients of the currently targeted block to the most likely values according to our distributions.

In the following, we present three types of update and two types of query selection. The methods range from a naïve approximation of the marginals to their exact computation, of which the latter allows for far better query selection.

3.1 Updating Distributions given Oracle Responses

We first describe how we update our approximation of the marginals given a response to a query; later, we will describe how to select the queries. In all of our methods, we have a current state that is updated with each query. Note that the updating process is entirely independent of how the queries are chosen, while the query selection depends on the available information; thus, we begin with explaining the update processes. All methods follow the same reasoning:

Lemma 1. *For a subkey* $\mathbf{s}$ *and queries* $(\mathcal{D}_j = \mathcal{O}(\mathbf{a}_j))_{j\in[K]}$ *for that subkey, we have*

$$P\left(s_i = v \mid (\mathcal{D}_j = \mathcal{O}(\mathbf{a}_j))_{j\in[K]}\right) \propto P\left(s_i = v\right) \sum_{(m_j)_j \in [2^N]^K} \prod_{j=0}^{K-1} P_{\mathcal{D}_j}\left(m_j\right) \delta_{v\leq a_{j,i}}(m_{j,i}), \tag{5}$$

where the bit value $\delta_A(b)$*, for a statement* A *and a bit* b*, is* 1 *if and only if* A *is true and* $b = 1$ *or* A *is false and* $b = 0$*.*

The proof can be found in Appendix C.

Naïve Updating of Marginals. In each query, we get information about each of the N targeted key coefficients. A simple and inexpensive approach is to store N probability distributions – one for each targeted key coefficient – and update them independently. We compute

$$P\left(s_i = v \mid (\mathcal{D}_j = \mathcal{O}(\mathbf{a}_j))_{j\in[K]}\right) \approx P\left(s_i = v\right) \prod_{j=0}^{K-1} \sum_{m=0}^{2^N-1} \delta_{v\leq a_{j,i}}(m_i) P_{\mathcal{D}_j}\left(m\right). \tag{6}$$

Clearly, this disregards the fact that the inequalities given by the oracle response are dependent, and, thus, loses a substantial amount of information. However, this approach can be realized very efficiently: Denoting the current probabilities for coefficient i by $p_i(v)$, we initialize all $p_i(v)$ to the prior, i.e., $\mathcal{B}_\eta(v)$ for ML-KEM. For each query response $\mathcal{D}$, we update $p_i(v)$ by computing

$$p_i(v) \leftarrow p_i(v) \sum_{m=0}^{2^N-1} \delta_{v\leq a_i}(m_i) P_{\mathcal{D}}\left(m\right). \tag{7}$$

We continue querying the oracle and updating the approximations to the marginals until the sum of their entropies is smaller than h_{abort}.

For each of the $2\eta + 1$ possible values that a secret entry can take, updating each of the N positions according to (7) computes a sum with 2^N terms. Assuming that we need K queries to recover the block of N positions, then the total number of terms adds up to $(2\eta + 1)KN2^N$. While exponential in N, as N is generally a small value in practice, the attack is cheap.

Using Belief Propagation. BP, see Section 2.4, allows computing approximations to the marginals of a joint probability distribution given a factorization of the joint mass function. In our case, we may see the message of a query response as a variable node, where the actual message is the true value. The query gives N factor nodes that are "incomplete" inequalities as they depend on the input of the message nodes and are connected to the variable nodes representing the secret key. An example of such a graph is given in Figure 3 for a three-dimensional subkey and two queries; note that we represent inequalities as smaller-equal and message nodes represent a potential multiplication by -1 and adding 1 to the right side, essentially turning them into the respective greater-than inequality.

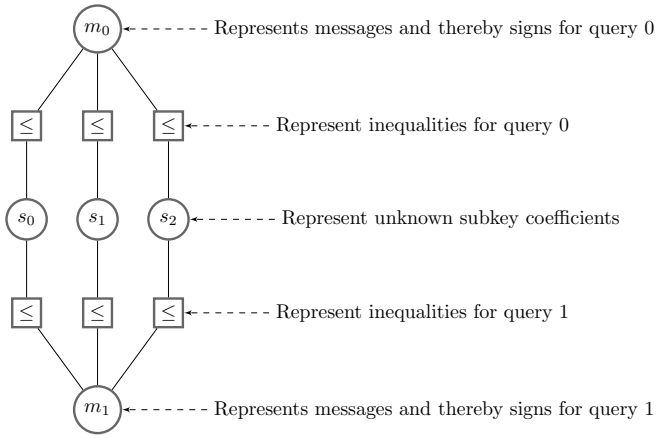


Fig. 3: Graph for three coefficients and two queries: The inequality nodes represent smaller-equal inequalities for some value $a_{j,i}$ for query j and coefficient i on the right side, and the secret key on the left side. The message nodes represent the query response, potentially changing inequalities to greater-than.

Initialization and update computation. Our graph features one type of non-trivial factor nodes: "incomplete" inequalities. These hold the values of the coefficients' query, a_i, and are connected to the message nodes, from which the inequalities receive a distribution on the messages. The i-th bit represents the probability of

the i-th inequality being a smaller-equal inequality. The variable nodes for the secret subkey coefficients are implicitly connected to factor nodes representing the prior[12]. The prior for the message nodes is the query response $\mathcal{D}_j$.

We denote the probabilities coming from an inequality with index i at query j in a step as follows: the probabilities for subkey coefficients are denoted p_i, the probabilities for messages as q_j. The probabilities are updated as

$$p_i(v) \propto q_j(1)\delta_{v\leq a_i}(1) + q_j(0)\delta_{v\leq a_i}(0) = \begin{cases} q_j(1) & \text{if } v \leq a_i \\ q_j(0) & \text{if } v > a_i \end{cases}. \tag{8}$$

The update for the message nodes, coming from inequality i, is computed as

$$q_j(1) \leftarrow \sum_{v=0}^{a_i} p_i(v) \text{ and then clearly } q_j(0) = \sum_{v=a_i+1}^{\eta} p_i(v). \tag{9}$$

Abort criteria and integrating new queries. From the BP, we obtain approximation for marginals after every single step. Again, we abort once the sum of entropies is below h_{abort}. Note that we have to restart the BP with every newly obtained query; this leads to a linear overhead compared to the other two methods that can update their probabilities without having to recompute the complete state. One viable strategy for an attacker would be to obtain as many queries as possible and then run BP once to recover the secret key.

Intuitive description and iterative approach. Without knowing about BP, an intuitive approach could be to just use Bayes' formula on the current state. The state is initialized with the prior – just as in BP. After each query, we compute the probabilities for each given message based on the current state, i.e., the current prior. We may then update the distributions of the key coefficients using Bayes' formula as normalized product of the current prior and the messages conditioned on the key. Then, after the update, a new query is submitted.

The approach is similar to the BP instantiation with a maximum number of iterations of 1 and restarting after every query. Using the BP with a larger number of iterations but without restarting offers an option to reduce the computational cost by a linear factor; however, it may increase the number of required queries.

Complexity. Variable nodes representing secret key coefficients compute K products of K message; however, this can be done in $2K(2\eta+1)$ multiplications (see, e.g., approaches from [48,27,24]) leading to a cost of $2NK(2\eta+1)$. The factor nodes have only two inputs and have a total cost of $3NK(2\eta+1)$ multiplications and additions and $2^N NK(2\eta+1)$ inequality checks. Finally, the query nodes, representing the messages, compute $K(2N(2\eta+1)+N)$ multiplications.

[12] We choose to make priors implicit to variable nodes in an effort to reduce notations.

Optimal Attacks. Both using only the marginals and using BP clearly lose information. While BP can provide a reasonably good approximation of the marginals, it is not exact as the factor graph is not acyclic. Instead of approximating the marginals, we can compute the joint probability distribution and obtain the exact marginals after every update. Unfortunately, this requires storing a table of size $(2\eta+1)^N$. Nevertheless, as seen in Section 5, this is a reasonably fast (in runtime) attack that may outperform BP for up to 4-dimensional subkeys[13].

To keep track of the joint probability distribution, we store the probability of every N-tuple of subkey values, i.e., of every possible subkey. These are initialized using the prior, i.e., $\mathcal{B}_\eta$ in ML-KEM, denoted by p_{prior}. Denoting the probability for a subkey $\mathbf{v} = (v_0, \ldots, v_{N-1})$ by $p(\mathbf{v})$, given a query response $\mathcal{D}$ for $\mathbf{a}$, we compute the updated probabilities as

$$p(\mathbf{v}) \leftarrow p(\mathbf{v}) \sum_{m=0}^{2^N-1} P_{\mathcal{D}}(m) \prod_{i=0}^{N-1} \delta_{v_i \le a_i}(m_i). \tag{10}$$

Lemma 2. *For K, $(\mathcal{D}_j = \mathcal{O}(\mathbf{a}_j))_{j\in[K]}$, the marginals of p, computed according to* (10)*, equal the conditional probabilities $P\left(s_i = v \mid (\mathcal{D}_j = \mathcal{O}(\mathbf{a}_j))_{j\in[K]}\right)$.*

The proof can be found in Appendix C.

Complexity. Naïvely Updating $p(\mathbf{v})$ according to (10) costs $N2^N$ inequality checks. For a block of N positions, we must perform this update for $(2\eta+1)^N$ different values of $\mathbf{v}$. Assuming K queries are needed to recover the entire block, the total number of operations to recover N positions is $K(2\eta+1)^N N2^N = KN(4\eta+2)^N$. While this scales badly with N, the attack works quickly up to 5 dimensions. Moreover, it allows us to compare the more efficient naïve and BP-based methods with using the query output in an information-theoretically optimal way.

Intuition on the sub-optimality of the marginal approach. While the marginals clearly hold less information than the full joint distribution, we would like to provide some intuition: Figure 4 shows an example where the entropy of the full joint distribution is already greatly reduced, yet the marginals remain unchanged. Knowing that $s_0 = s_1$, the entropy is cut in half compared to initial distribution. However, marginal distributions cannot capture this information.

Combining with lattice reduction. Both the BP and the optimal approach can be combined with lattice reduction as proposed in [24]. In BP we have only the marginals while in the optimal approach we know the full joint probability distribution, which may enable improved hint integration following [7,8,38].

[13] Note that our implementations are not highly optimized.

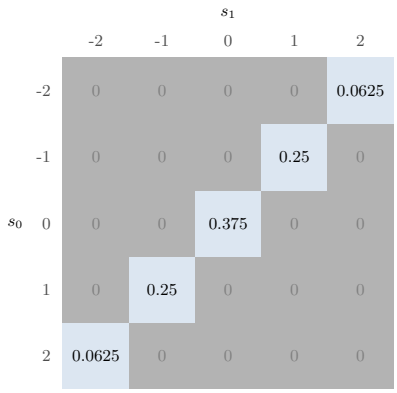


Fig. 4: Example of a joint distribution for two targeted key coefficients in which the marginals have not changed, but the entropy is greatly reduced – $s_0 = s_1$.

3.2 Choosing Oracle Queries

Apart from updating (approximations of) probability distributions, the adversary must select queries. A query is a chosen ciphertext, modeled as a vector $\mathbf{a}$ representing $s_i \leq a_i \Leftrightarrow m_i = 1$. These chosen ciphertexts lead to one of 2^N messages $\mathbf{m}$ during decapsulation – depending on inequalities in the secret key. These messages, which the adversary attempts to distinguish using a side channel, determine the randomness used for re-encryption (see Algorithm 6). The oracle response is modeled as a probability distribution over $\mathbf{m}$. All previous queries are again denoted as $(\mathcal{D}_j = \mathcal{O}(\mathbf{a}_j))_{j\in[K]}$.

In general, the goal is to maximize the MI between the oracle given our query and the secret key, both conditioned on all previous queries. For a perfect oracle[14], this is exactly the entropy of the oracle query conditioned on previous queries, as

$$I\left(\mathbf{s}; \mathcal{O}(\mathbf{a}) \mid (\mathcal{D}_j = \mathcal{O}(\mathbf{a}_j))_{j\in[K]}\right) = I\left(\mathcal{O}(\mathbf{a}); \mathbf{s} \mid (\mathcal{D}_j = \mathcal{O}(\mathbf{a}_j))_{j\in[K]}\right) \tag{11}$$

$$= H\left(\mathcal{O}(\mathbf{a}) \mid (\mathcal{D}_j = \mathcal{O}(\mathbf{a}_j))_{j\in[K]}\right) - H\left(\mathcal{O}(\mathbf{a}) \mid (\mathcal{D}_j = \mathcal{O}(\mathbf{a}_j))_{j\in[K]}, \mathbf{s}\right), \tag{12}$$

which is $H(\mathcal{O}(\mathbf{a}) \mid (\mathcal{D}_j = \mathcal{O}(\mathbf{a}_j))_{j\in[K]})$ as $\mathcal{O}(\mathbf{a})$ is known, given $\mathbf{s}$. For a query $\mathbf{a}$, define $A_m(\mathbf{a})$ as the region cut by the inequalities that $\mathbf{a}$ and m represent, i.e.,

$$A_m(\mathbf{a}) = \left\{\mathbf{v} \in \{-\eta, \ldots, \eta\}^N \mid (-1)^{m_i} v_i > (-1)^{m_i} a_i - m_i \ \forall i \in [N]\right\}, \tag{13}$$

for the bits m_i of m. The entropy $H(\mathcal{O}(\mathbf{a}) \mid (\mathcal{D}_j = \mathcal{O}(\mathbf{a}_j))_{j\in[K]})$ then becomes

$$H\left(\mathcal{O}(\mathbf{a}) \mid (\mathcal{D}_j = \mathcal{O}(\mathbf{a}_j))_{j\in[K]}\right) = \sum_{m=0}^{2^N-1} \sum_{\mathbf{v}\in A_m} -p(\mathbf{v}) \log_2 \left(\sum_{\mathbf{v}\in A_m} p(\mathbf{v})\right), \tag{14}$$

[14] Here we write the perfect oracle as a distribution oracle that assigns m_{actual} the probability 1.

where $p(\mathbf{v})$ denotes the current joint probability distribution, i.e., the current "state", $p(\mathbf{v}) = P(\mathbf{s} = \mathbf{v} \mid (\mathcal{D}_j = \mathcal{O}(\mathbf{a}_j))_{j\in[K]})$.

For an imperfect (distribution) oracle, the same reasoning applies: The entropy conditioned on the secret key is for an imperfect oracle – by definition – not zero. However, in our abstraction, we assume no further information on the nature of the imperfectness of the oracle, i.e., on $H(\mathcal{O}(\mathbf{a}) \mid (\mathcal{D}_j = \mathcal{O}(\mathbf{a}_j))_{j\in[K]}, \mathbf{s})$. Thus, by (11)-(12) we have to maximize $H(\mathcal{O}(\mathbf{a}) \mid (\mathcal{D}_j = \mathcal{O}(\mathbf{a}_j))_{j\in[K]})$, which means maximizing (14). In the following, we discuss this for all three attack types. In the naïve and BP-based attacks, only the marginals of the joint distribution are available. In the optimal attack, however, the adversary can choose much better queries. Thus, we get the following statement:

Lemma 3. *Given queries $(\mathcal{D}_j = \mathcal{O}(\mathbf{a}_j))_{j\in[K]}$, the MI between the secret and the oracle is maximized by maximizing* (14).

In the following, we will refer to queries/splits according to Lemma 3 as optimal. By this, we mean optimal for a strategy that

- only targets N positions at a time instead of the blocks with maximal entropy,
- only considers the maximal entropy reduction for the next query,
- and only queries the oracle for inequalities along one of the secret key axes; see [57] for a parallelized mismatch attack that performs other splits.

Query selection approaches. Depending on the attack, our "state" consists of either an approximation of the marginals or a full joint probability distribution. In the former case, we may rely solely on the marginals while in the latter we can achieve better results.

Using the marginals. For a perfect binary oracle, prior work [52] (clarified in [19]) selected each a_i such that $\sum_{v=-\eta}^{l} p_v - \sum_{v=l+1}^{\eta} p_v$ is minimized. Our reasoning above formalizes and generalizes this way of selecting queries. Given an imperfect oracle, we may approximate the joint probability distribution using the marginal distributions – essentially assuming independence – and apply the heuristic that follows Lemma 3. Thus, we choose the query for each position independently so that the entropy of the approximated split distribution is maximized.

Using the joint probability distribution. For optimal updating, we iterate over all queries $\mathbf{a} \in \{-\eta, \dots, \eta - 1\}^N$ and select the one that maximizes the entropy of the split distributions. The drawback is the computational cost: $(2\eta)^N$ possible splits must each iterate over $(2\eta + 1)^N$ values in the joint distribution, resulting in $(2\eta)^N(2\eta + 1)^N = (4\eta^2 + 2\eta)^N$ additions. Each split also requires an entropy computation involving 2^N logarithms. While asymptotically insignificant, this step can take up a large part of the runtime for practical values of N. Improved methods for finding optimal splits and approximations are left for future work.

Intuition on the sub-optimality of using the marginals. It is not obvious that choosing queries based on the marginals gives suboptimal results compared to the method described in this section. Consider the example given in Figure 5: the left side shows an optimal split, while the right shows the marginal-based split.

With marginal splits, two messages provide no information, whereas in the optimal case, all but one message holds information. Marginal splits reveal at most 1 bit of information: a perfect oracle gives 1 bit of information – two messages cannot occur and the other two exclude half the possible values. While positive probabilities may be assigned to the impossible messages, as they are known to be zero, they have no effect in the updating. For an optimal query, the split distribution has the probabilities 0.5, 0.25 and 0.25. Thus, for this query, an perfect oracle reveals 1.5 bits of information.

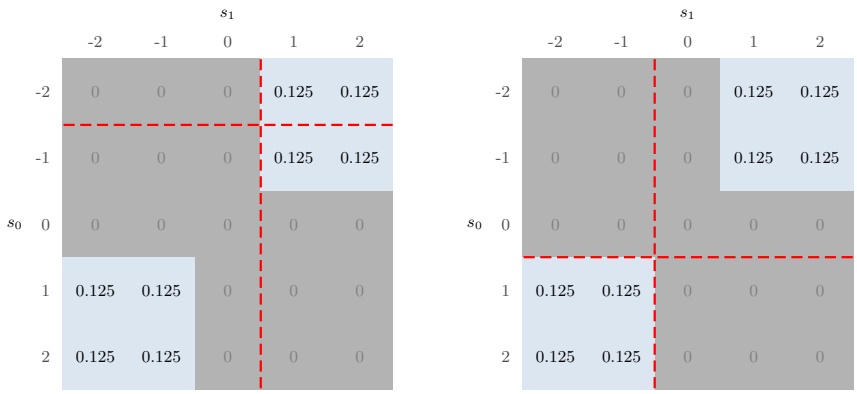


Fig. 5: Joint distribution given previous queries, with an optimal split according to the joint distribution (left) and a split obtained from the marginals (right).

Choosing targeted positions adaptively. Our methods target a block of N positions until all corresponding secret values are determined with sufficient certainty. Alternatively, the adversary could adaptively choose which positions to target, with – again – MI as the rationale behind: the less information they have about a position, the more they may gain from querying the oracle about it.

4 Oracle Instantiations

Many different oracle instantiations can be derived from concrete side channels. Prior work focused mostly on perfect or tournament-based oracles – the latter abstracting real-world side channels well but often losing information when instantiating. We introduce two oracles – based on the noisy HW model – that are closer to real attacks, providing less abstraction but exploiting more information and better suited for high-noise scenarios. We also discuss the information-theoretic aspects of tournament-based oracles, deriving a lower bound on the

required number of queries. In Section 5, we compare our results against previous attacks and this lower bound.

4.1 Tournament-Based Oracle

Recall from Section 3 that a tournament-based oracle assumes that the adversary assigns scores to messages – e.g., via a template attack – and compares them pairwise, discarding lower-scored ones, i.e., not all messages are being compared. Therefore, a message is selected as the assumed actual message after $N-1$ comparisons, where N is the number of targeted coefficients, resulting in efficiency even for many positions. This approach has been used for measurements of a physical device in [53]. A tournament-based oracle, $\mathcal{O}_{\mathrm{tou},N,\alpha}$, is instantiated by its correctness probability α, the probability that the returned message is correct. In simulations, the adversary receives the correct message, m_{actual}, with probability α, and a random incorrect message from $\left[2^N\right]$ with probability $1-\alpha$.

Information-theoretic lower limit. Lemma 3 shows how to optimally choose queries. For a tournament-based oracle, the mutual information $I(\mathcal{O}_{\mathrm{tou},N,\alpha};\mathbf{s})$ can be computed in closed form. We first derive an upper limit on how many bits of information an oracle can maximally output, which in turn gives a lower limit on the expected number of queries required to recover the key with certainty. Targeting N coefficients, we obtain an inequality per coefficient, and at most N bits. The entropy of the oracle response is

$$H(\mathcal{O}_{\mathrm{tou},N,\alpha}) = -\alpha\cdot\log_2(\alpha) - \left(2^N-1\right)\frac{1-\alpha}{2^N-1}\log_2\left(\frac{1-\alpha}{2^N-1}\right) \tag{15}$$

$$= h(\alpha) + (1-\alpha)\cdot\log_2(2^N-1) \approx h(\alpha) + (1-\alpha)\cdot N, \tag{16}$$

where $h(\alpha) = \alpha\log_2(\alpha) + (1-\alpha)\log_2(1-\alpha)$, and the approximation holds well for large N. For the case $N=1$, the oracle is simply a binary symmetric channel. The minimum number of queries to recover the whole secret then becomes

$$\left\lceil \frac{H(\mathbf{s})}{N - (h(\alpha) + (1-\alpha)\cdot\log_2(2^N-1))} \right\rceil \approx \left\lceil \frac{H(\mathbf{s})}{\alpha\cdot N - h(\alpha)} \right\rceil, \tag{17}$$

where $H(\mathbf{s})$ is the entropy of the whole secret key. Table 1 shows the minimum expected number of queries needed for certain key recovery per targeted number of positions and oracle accuracy, for Kyber768.

Information loss. The tournament-based oracle ignores that a side channel usually results in a probability distribution for the targeted values, in turn giving a probability distribution for the messages. Returning only the most likely value and discarding the rest of the distribution is clearly suboptimal. For instance, measurements that show a uniform distribution with a tiny bias towards one value are treated similarly as measurements that single out one message with probability one. Our definition of distribution oracles avoids this loss of information.

Table 1: Minimum number of queries per targeted positions N and α.

N/α	0.9	0.8	0.7	0.6	0.5	0.4	0.3	0.2	0.1
1	2936.95	5608.37	13137.42	53685.46	∞	53685.46	13137.42	5608.37	2936.95
2	1136.26	1622.69	2424.57	3947.54	7515.13	19975.57	168865.37	154379.88	14918.29
4	496.62	624.64	801.14	1063.59	1490.16	2276.97	4062.49	10222.45	105353.27
8	231.67	274.6	330.38	407.05	519.35	698.58	1024.21	1767.0	4640.27
10	182.8	214.27	254.86	310.07	389.81	514.71	735.73	1219.15	2929.95

4.2 Noisy Hamming Weight Oracle

PCO attacks allow for on-device profiling and template attacks [6] without requiring a clone device. The adversary learns a probability distribution from each measurement. This type of soft information is the motivation behind distribution oracles and our oracle instantiation allows for more efficient attacks.

Online template attacks. In (online) template attacks, the adversary first chooses a leakage model, usually including Gaussian noise. This means that, for some α, σ, and $\mathcal{L}$, a processed intermediate value u leaks $\alpha\mathcal{L}(u) + \mathcal{N}(0, \sigma)$. The leakage function $\mathcal{L}$ is often assumed to be the HW – the model is referred to as the noisy HW model. In the following, we assume $\alpha = 1$ for simplicity. The adversary then sends in chosen ciphertexts for which they know the resulting message and records the distribution at a Point of Interest (PoI) for intermediate values u, learning the expected value and standard deviation σ for given intermediates. Given a measurement x at the PoI, they may now match against the templates and recover probabilities for each intermediate as

$$P(u = u' \mid x) = \frac{p(x \mid u = u')P(u = u')}{\sum_{u''} P(x \mid u = u'')P(u = u'')} = \frac{p_{\mathcal{N}(\alpha\mathcal{L}(u),\sigma)}(m)P(u = u')}{\sum_{u''} P(x \mid u = u'')P(u = u'')}, \tag{18}$$

by Bayes' rule. In turn, as the relation between messages and intermediates is public, the probabilities on intermediate values occurring during the re-encryption map to probability distributions on messages.

Noise HW oracle instantiation. To instantiate our model of an oracle, $\mathcal{O}_{\mathrm{nhw},N,r,\sigma}$, we assume a template attack on the last layer of the NTT in Algorithm 3 (see highlighted line). These $k \cdot n$ values, i.e., 512, 768, or 1024 are distributed uniformly random over $\mathbb{F}_q$. Concretely, the adversary is assumed to carry out the following attacks:

1. Choose r values to be targeted.
2. Submit honestly-generated ciphertexts to record a template for each r value.
3. For a query $\mathbf{a}$, submit the corresponding ciphertext and perform measurements of all r values.
4. Compute a probability distribution for each of the values of r based on the templates and measurements.

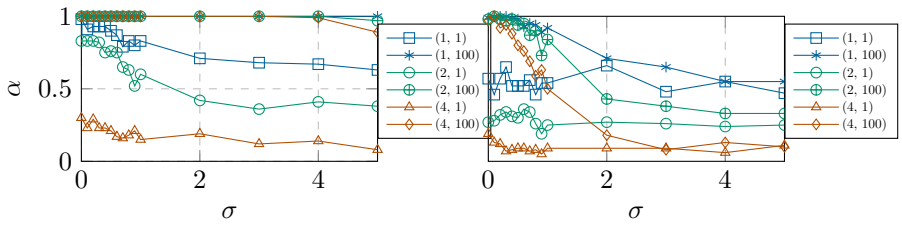
5. Assign each possible message the normalized product of the probability of the intermediates corresponding to the message.

Computing the leaked information involves Gaussian mixture models over HW distributions, which have no closed form expression [30]. Our implementation offers a routine to evaluate the leaked information per oracle call on a fresh secret key or during complete attacks. However, for the optimal attack, this information is entirely contained in the number of oracle calls.

Masked Noisy HW oracle instantiation. Masking [5] is one of the most notable countermeasures against side-channel analysis, and proposals for masked ML-KEM have been published [4,23]. Arithmetic masking shares a value $x \in \mathbb{F}_q$ as $x = \sum_{i=0}^{t} x_i$, where x_i are uniformly random over $\mathbb{F}_q$ and t is the masking order. The NTT is linear and can thus be computed on each share individually. Consequently, we assume that the adversary targets the $t+1$ shares of each of the targeted intermediate values. They then retrieve the distribution of the intermediate as the convolution (modulo q) of the distributions of the shares. From the distribution on the intermediates, they then derive the distribution on the messages as before, giving an oracle $\mathcal{O}_{\mathrm{shw},N,r,\sigma}$.

Relevance of exploiting all information. In unmasked implementations or under insufficient noise, abstracting the noisy HW model as a tournament-based oracle may often yield high accuracies. With masking, the information given on the secret intermediate value by measuring the shares decreases exponentially in the masking order (see, e.g., [11] for a thorough analysis with practical results). This may be less noticeable for low-noise few-shares settings, e.g., on evaluation devices. However, in real-world settings under sufficient noise, attacks are unlikely to succeed if much of the given information is ignored – even for distinguishing attacks, which are particularly hard to defend against. This is especially true when targeting arithmetic masking, which often behaves more favorably [37,25].

We may instantiate a tournament oracle based on the noisy HW model. This allows us to assess information loss caused by fitting leakage into an abstract tournament-based oracle. To evaluate the cost of abstraction, we 1) sample a fresh key, 2) issue the initial query to the oracle, 3) perform a tournament for the queries, and 4) record a success if it matches m_{actual}, a failure otherwise. We set α to the accuracy over 100 samples, as shown in Figure 6, yielding an optimal instantiation of a tournament-based oracle from this type of leakage; here, we target the output of the NTT of $\mathbf{r}_1$ during the re-encryption with coefficients in $\{0,\ldots,q-1\}$ and in arithmetic masking over $\{0,\ldots,q-1\}$. For low noise or without countermeasures, α can be very high, close to or even equal to 1.0. However, with masking and moderate noise, or when only a few positions can be targeted (e.g., due to hiding countermeasures), α quickly drops. Together, Figures 6, 7 and 10 show that instantiating a tournament-based oracle from noisy HW leakage requires far more traces.



(a) Unmasked. (b) Two shares.

Fig. 6: Comparing tournament and noisy HW-based models: for a given σ, we achieve α if we instantiate a tournament-based oracle using a noisy HW oracle for an unmasked setting and two shares; graphs for $(N, \text{targeted NTT values})$.

5 Evaluation

We evaluate the proposed attacks against the three proposed oracle instantiations. Our attacks are denoted "optimal" for the full joint distribution, "bp" for the BP approximation, and "naïve" for the naïve approximation. Under the assumption of Lemma 3, the optimal attack exploits the given information optimally. Thus, we do not provide a separate information-theoretic analysis – this is implicitly given by the optimal attack. Our evaluations show the number of traces for recovering the full key in ML-KEM768 (1024 in Table 2), i.e., $T \cdot \frac{768}{N}$ where T is the number of traces to recover an N-dimensional subkey.

5.1 Simulation

To assess varying noise levels and provide reproducible results, we evaluate our method using simulations. We provide results for abstract tournament-based oracles as well as with our new oracle definition instantiated as described in the previous section (i.e., relying on the noisy HW model). In the next section, we additionally targeted a first-order masked and highly optimized implementation.

Methodology and implementation. Our attacks are implemented in Python and rely on PQClean [32] for an ML-KEM implementation. We always first sample a random secret key using the de-randomized key generation routine with a random seed. Each oracle call first obtains the actual message m_{actual} for the submitted chosen-ciphertext. Then, in the tournament-based oracle, we obtain randomness in $[0, 1]$ and return m_{actual} if it is smaller than α; otherwise, we rejection-sample a different uniform message. The returned message is assigned probability α, all other messages are assigned probability $\frac{1-\alpha}{2^N-1}$.

The noisy HW oracles sample leakage on the targeted intermediate NTT value(s) or their shares, perform a template attack, then return the resulting distribution. We target the last layer of the NTT of $\mathbf{r}_1$ during the re-encryption (see Algorithm 3); in the unmasked case, the coefficients are assumed to be stored in $\{0, \ldots, q-1\}$, and we simulate unsigned arithmetic masking.

All attacks can be combined with any oracle instantiation and the optimal attack also allows for different types of split selection. However, we only evaluate the optimal, w.r.t. the assumptions in Section 3.2, split selection. Our unoptimized attacks can still run on a laptop for a reasonable number of targeted positions N.

For our evaluations, we target the first N coefficients 10 times per setting. Exceptions are the failure evaluations, Table 2, and Figure 6 from the previous section, for which we used 10000, 100, and 100 samples[15] per setting, respectively.

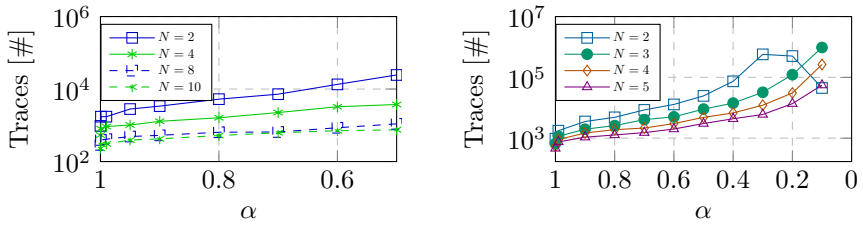
Note that both targeting the NTT and the number of targeted values are chosen as an example. In principle any variable during the re-encryption that depends on the message[16] may be targeted in combination with all other such variables. Targeting more values increases the effort for profiling and template matching for the adversary. Furthermore, an adversary might be able to obtain clear leakage on some values while having almost no information on others (see Section 5.2). Therefore, we evaluate for several numbers of values instead of just targeting all coefficients.

Abort criteria and failure rate. We abort both the BP and the optimal attacks if the marginals have entropy below 0.001, or after a maximum of 10000 oracle calls for all settings except the binary tournament-based oracle at $\alpha = 0.51$, where we allow up to 100000 calls. This results in very low failure rates, typically zero; for instance, the BP had one to four failures in the tournament-based setting for $\alpha \in \{0.9, 0.95\}$, $N \in \{4, 8\}$ across 10000 samples, while the optimal attack had no failures for $\alpha \in \{0.9, 0.95\}$, $N = 4$. Failing examples are excluded from the computation of the average number of oracle calls, as an adversary would likely detect when an excessive number of calls yield little information while we continue iterating up to 10000 calls.

The same entropy bound gives different guarantees depending on the attack. Therefore, in some settings, the BP yields very comparable results. However, it should be noted that the BP failure rate, although low, is still higher than that of the optimal attack. The optimal attack can be improved in terms of the number of oracle calls by allowing a higher error rate, i.e., aborting earlier. In addition, in the optimal attack, we may also use the entropy of the joint distribution instead of the sum of the entropies of the marginals; the latter is usually higher (and always greater or equal). A subsequent lattice reduction step may fully recover the secret key if the distance between guess and secret is sufficiently small using, e.g., [24]. We use the sum of the marginals for all three attacks in this setting.

Tournament-based oracle. We evaluate the tournament-based oracle for $\alpha \in \{0.1, \ldots, 0.99\}$ using both BP and the optimal attack, with results shown in Figure 7. The BP becomes excessively slow when many queries are needed, so we evaluate it only for settings with fewer queries and leave further optimization to future work. For a binary oracle and the optimal attack, a mean of 19319142.912

[15] "Samples" refers to simulated attacks that we average over to obtain our results.
[16] That includes random values as the randomness is derived from the message.



(a) BP. (b) Optimal.

Fig. 7: Results for BP and the optimal attack targeting a tournament-based oracle in terms of number of calls per oracle accuracy α. Note that the least information occurs at $\alpha = \frac{1}{2^N}$, which explains the plateau for $N = 2$.

Table 2: Comparison against [35] for $N = 8$ targeting ML-KEM1024.

α	0.95	0.9	0.8	0.7	0.6	0.5
[35]	1336.8	2041.6	4625.8	7295.8	n.a.	n.a.
This work	623.36	706.56	817.92	944.64	1166.08	1529.6

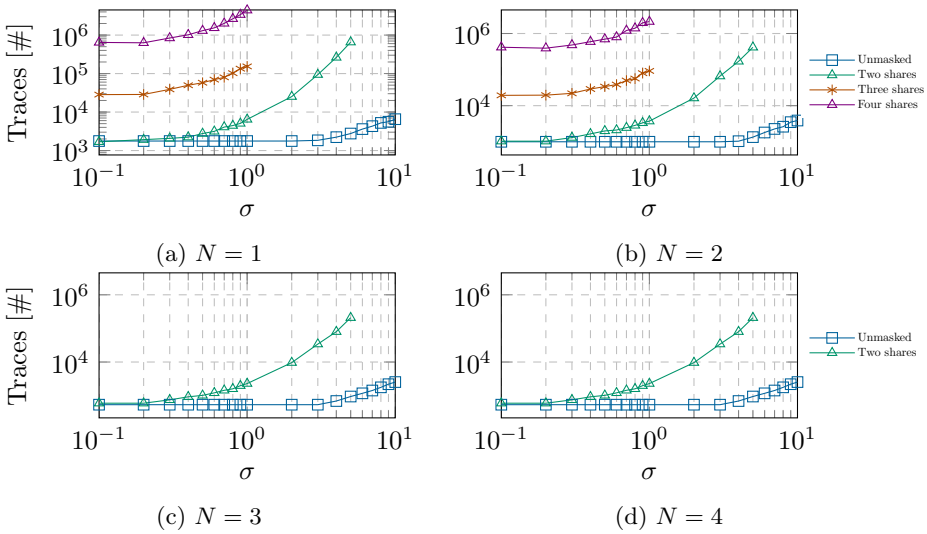
queries is required for a success rate of 1 at $\alpha = 0.51$ (10,000 samples), while at $\alpha = 0.55$, 722380.8 queries are needed (10 samples).

The most recent proposal [35] targets imperfect PCOs with α as low as 0.7 for N up to 8, outperforming previous works. Thus, we compare our attacks to theirs, focusing on the BP due to the high computational cost of the optimal attack for $N = 8$. Table 2 provides results for $N = 8$ for different values of α, showing that our method requires fewer traces while achieving zero error. The results in Table 2 provide a comparison *in the previous model*; note that our new model allows for further improvements.

Noisy HW Oracle. The noisy HW oracle depends on the dimension N of the targeted subkey, the number of targeted NTT values, and σ. The masked noisy HW oracle also depends on the number of shares $t + 1$. We target the output, i.e., the last layer of the NTT of $\mathbf{r}_1$ (see Algorithm 3); in the masked setting, we assume arithmetic masking over $\{0, \ldots, q - 1\}$.

For $N \in \{1, 2\}$ (Figures 8a and 8b), we evaluate the optimal attack in the unmasked setting for σ up to 10. For the masked settings, we consider two to four shares: σ ranges up to 5 for two shares, and up to 1 for three and four shares. For $N \in \{3, 4\}$ (Figures 8c and 8d), we evaluate σ up to 10 in the unmasked setting and up to 5 for first-order masking. For the masked setting, we observed a single case in third-order masking with $\sigma = 1$ where the subkey was not recovered after 10000 oracle calls using an entropy bound of 0.01.

Targeting more than 5 positions quickly becomes infeasible with the full updating. However, both the BP and the naïve attack can easily handle this as shown in Figure 9 for $\sigma \in \{0.1, 0.2, \ldots, 1, 2, \ldots, 5\}$ and $N \in \{1, \ldots, 10\}$ for 100



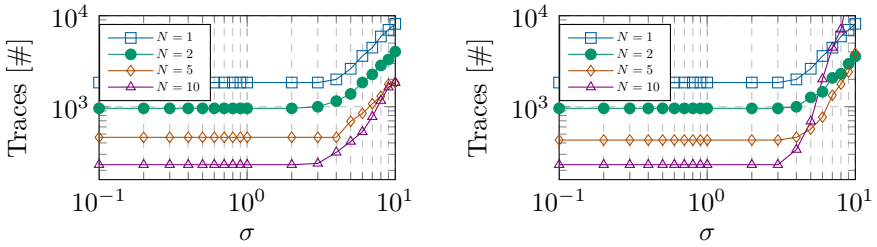
(a) $N = 1$ (b) $N = 2$

(c) $N = 3$ (d) $N = 4$

Fig. 8: Results in terms of oracle queries per noise level for different number of targeted positions in an unmasked noisy HW oracle and different order masked noisy HW oracles targeting 100 NTT values.

targeted values. The success rate for the BP approach is always 1. For the naïve approach, it is 0.8 at $N = 10$ and $\sigma \geq 4$, and 1 otherwise. The mean computation time for different settings lies between 22 and 1492 seconds. It should be noted that the time for key recovery is unlikely to be the limiting factor in practice: BP can be parallelized and Figure 9 shows that the advantage of larger N quickly diminishes for increased noise levels. Appendix D contains figures for each number of shares with graphs for each N.

Concurrent improvements on DF attacks. DF attacks are similar to PCO attacks, but the adversary is not given the information which message resulted from the



(a) BP (b) Naïve.

Fig. 9: Targeting up to 10 positions and 100 NTT values.

decryption, but only whether it was *one specific* message[17]. The concurrent work of Guo, Nabokov, and Johansson [21] shows that DF attacks may require fewer queries than *binary* PCO attacks. The authors of [21] also already provide a comparison against a previous version of our work comparing their attack against our binary case. It should be noted that our main improvements are in the multi-value case, and we did not optimize queries to the same extent as [21] .

More importantly, the comparison of [21] relies on a tournament oracle whereas our main improvements come from leveraging all available information by relying on our distribution oracles. In addition, [21] allows for some errors while we aim at direct full key recovery; these errors are of little importance in practice as an inexpensive lattice reduction step may still recover the full secret [24]. The authors report that they require 9 690 509 queries for $\alpha = 0.51$, and we require 19 319 143. Adding to their comparison, we compare their approach to using our new model definition: we simulate an oracle based on the noisy *value* model at $\sigma = 20$ leading to $\alpha = 0.51$; we carry out 1000 recoveries of the first two coefficients in this model. Using an entropy bound of 0.001, we require a mean of 12 357 628 queries at an error rate of 0 in our 1000 experiments. For an entropy bound of 0.01, we encounter one error and 9 572 626 traces; at 0.05, we observe 6 errors and 7 401 622 queries. In all of these cases, full key recovery would be feasible without much effort. Note that we have considerably more information available compared to [21]; however, in the settings we consider, this information is commonly available, and our improvements are comparably small in the binary case compared to the multi-value settings. In general, these comparisons are difficult to make based on abstract models and depend on the concrete attack, i.e., the information available, and error rates need to be taken into consideration.

5.2 Physical Evaluation

For physical verification, we target the arithmetically masked NTT of mkm4 [23] on an `STM32F415`. The targeted device is placed on a ChipWhisperer `CW308` [45,43] base-board and its power consumption measured using a LeCroy WaveRunner 9254m oscilloscope with a series of two 50 MHz and two 25 MHz analog filters and a low-noise amplifier to condition the power signal. We target the NTT using a standard template attack [6], in which profiling is carried out using random inputs and selecting the five best PoIs per coefficient. Note that this type of profiling can be carried out in practice *on the targeted device*. To construct the templates, we rely on linear discriminant analysis. Our profiling leads to an unequal number of samples for different HWs, which potentially decreases the retrieved information. In addition, the combination of arithmetic masking and unsigned coefficients substantially reduces the leaked information compared to Boolean masking or signed arithmetic masking (see, e.g., [37,61,25]). Our analysis shows that 32 of 256 coefficients have an average Signal-to-Noise ratio (SNR) (of targeting one share) of 0.971 with an SNR of 0.113 at the remaining coefficients.

[17] A variant of this definition was first provided in an earlier, unpublished version of [21], but focused on the oracle instead of the attack.

We ran 50 experiments for the optimal attack targeting an $N \in \{2, 3, 4\}$ subkey aborting at 200 calls or an entropy of less than 0.001; in addition, we targeted 10 full keys for $N = 3$ with an entropy abort of 0.01 and a maximum of 200 calls; in the latter case, we use the joint entropy as abort criteria. Targeting the subkeys with $N = 2$, we require an average of 95.53 queries with 3 aborts and one subkey not being recovered but being off by one in one coefficient; at $N = 3$, we are at 93.18 queries, 5 aborts, and no incorrect subkeys; at $N = 4$, we require 103.28 queries with one abort (but correct convergence) and one incorrect convergence. In targeting full keys with a reduced entropy abort (allowing more errors but speeding up the attack), we achieve an average Euclidean distance of 10 to the true key, which results in an easily solvable lattice problem; using information on the key distribution, the key can most likely also be found by enumeration. We describe our results, additional evaluations, and our setup in more detail in Appendix B.

6 Conclusion

In this work, we present several PCO attacks based on soft-analytic techniques, approximating or computing the marginals of the joint distribution of the targeted subkey. The attacks exhibit significantly higher noise tolerance. Furthermore, we introduce a new oracle abstraction that allows for further exploiting available information. Using our oracles, we discuss instantiations that more closely reflect protected real-world implementations.

We draw several conclusions. First, PCO attacks can be highly noise tolerant. In combination with the FO transform offering a large attack surface, this makes them even more threatening. Secondly, even rather naïve soft-analytic techniques may outperform different, more sophisticated approaches. This observation matches previous works [17] despite limitations [59]. Our work further motivates exploring SASCAs, improving techniques, and analyzing existing algorithms (see, e.g., [39]). It exposes weaknesses in previous abstractions: tournament-based oracles can severely underestimate the information an adversary obtains. Moreover, we show how information from these and other attacks can be formalized and exploited more exhaustively. Finally, our models provide a basis for developing evaluation methods. However, the models are still abstractions. Adversaries can learn more information from actual traces/better models.

Our work also poses several new questions: We have not yet explored strategies that look multiple steps ahead to find the optimal queries. Currently, we target one block at a time – potentially, the number of traces can be reduced by maximizing entropy reduction and using lattice reduction (e.g., [24,7]). In addition, regarding query selection, an adversary could use queries that do not only split along the subkey's axes. Furthermore, performance improvements through better implementations and algorithmic improvements could increase the number of positions targeted before the recovery becomes computationally infeasible. Moreover, the BP "with restart" is needlessly inefficient for high-noise cases.

While these are interesting open questions, we believe that the most pressing matter is finding evaluation models for countermeasures against PCO attacks.

Acknowledgments

We would like to thank Qian Guo and Rei Ueno for the helpful discussions. J.H. was funded by the German Federal Ministry of Research, Technology and Space (BMFTR) through the projects 6GEM (16KISK038). E.M. and M.T. were funded by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation. E.M. was also funded by the Excellence Center at Linköping-Lund in Information Technology (ELLIIT). We used Overleaf's AI assistant[18] as well as the free version of LanguageTool[19] in combination with ltex[20] for spell and grammar checks, but did not use the generative features. For Figures 4 and 5, the free version of ChatGPT[21] was used to find syntax errors and select better colors; conception and initial figures were done manually. The color scheme for all other figures was also suggested by ChatGPT, but no further capabilities of the tool were used. A bug in mkm4 was located and corrected with the help of Claude Opus 4.6[22].

References

1. Aguilar-Melchor, C., Aragon, N., Bettaieb, S., Bidoux, L., Blazy, O., Deneuville, J.C., Gaborit, P., Persichetti, E., Zémor, G., Bos, J., Dion, A., Lacan, J., Robert, J.M., Veron, P.: HQC. Tech. rep., National Institute of Standards and Technology (2022), available at https://csrc.nist.gov/Projects/post-quantum-cryptography/round-4-submissions
2. Balon, B., Cassiers, G., Schoenauen, T., Standaert, F.X.: How strong is the FOcalypse, really? instantiating plaintext-checking oracles against masked software implementations of ML-KEM. Cryptology ePrint Archive, Paper 2026/777 (2026), https://eprint.iacr.org/2026/777
3. Belson, D.: Cloudflare 2025 year in review (2025), https://radar.cloudflare.com/year-in-review/2025#post-quantum-encryption
4. Bos, J.W., Gourjon, M., Renes, J., Schneider, T., van Vredendaal, C.: Masking Kyber: First- and higher-order implementations. IACR TCHES **2021**(4), 173–214 (2021). https://doi.org/10.46586/tches.v2021.i4.173-214, https://tches.iacr.org/index.php/TCHES/article/view/9064
5. 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

[18] https://www.overleaf.com/learn/how-to/AI_features
[19] https://languagetool.org
[20] https://valentjn.github.io/ltex/
[21] https://openai.com/index/chatgpt/
[22] https://www.anthropic.com/news/claude-opus-4-6

6. 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
7. 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
8. 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
9. Dong, H., Guo, Q.: OT-PCA: New key-recovery plaintext-checking oracle based side-channel attacks on HQC with offline templates. Cryptology ePrint Archive, Report 2024/1715 (2024), https://eprint.iacr.org/2024/1715
10. Duc, A., Dziembowski, S., Faust, S.: Unifying leakage models: From probing attacks to noisy leakage. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 423–440. Springer, Berlin, Heidelberg (May 2014). https://doi.org/10.1007/978-3-642-55220-5_24
11. Duc, A., Faust, S., Standaert, F.X.: Making masking security proofs concrete - or how to evaluate the security of any leaking device. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015, Part I. LNCS, vol. 9056, pp. 401–429. Springer, Berlin, Heidelberg (Apr 2015). https://doi.org/10.1007/978-3-662-46800-5_16
12. Engineering, A.S., Architecture: imessage with pq3: The new state of the art in quantum-secure messaging at scale (2024), https://security.apple.com/blog/imessage-pq3/
13. Fluhrer, S.: Cryptanalysis of ring-LWE based key exchange with key share reuse. Cryptology ePrint Archive, Report 2016/085 (2016), https://eprint.iacr.org/2016/085
14. Fujisaki, E., Okamoto, T.: Secure integration of asymmetric and symmetric encryption schemes. In: Wiener, M.J. (ed.) CRYPTO'99. LNCS, vol. 1666, pp. 537–554. Springer, Berlin, Heidelberg (Aug 1999). https://doi.org/10.1007/3-540-48405-1_34
15. Fujisaki, E., Okamoto, T.: Secure integration of asymmetric and symmetric encryption schemes. Journal of Cryptology **26**(1), 80–101 (Jan 2013). https://doi.org/10.1007/s00145-011-9114-1
16. Gallager, R.G.: Low-density parity-check codes. IRE Trans. Inf. Theory **8**(1), 21–28 (1962). https://doi.org/10.1109/TIT.1962.1057683, https://doi.org/10.1109/TIT.1962.1057683
17. Grosso, V., Standaert, F.X.: ASCA, SASCA and DPA with enumeration: Which one beats the other and when? In: Iwata, T., Cheon, J.H. (eds.) ASIACRYPT 2015, Part II. LNCS, vol. 9453, pp. 291–312. Springer, Berlin, Heidelberg (Nov / Dec 2015). https://doi.org/10.1007/978-3-662-48800-3_12
18. Guo, Q., Johansson, T., Nilsson, A.: A key-recovery timing attack on post-quantum primitives using the Fujisaki-Okamoto transformation and its application on FrodoKEM. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020, Part II. LNCS, vol. 12171, pp. 359–386. Springer, Cham (Aug 2020). https://doi.org/10.1007/978-3-030-56880-1_13
19. Guo, Q., Mårtensson, E.: Do not bound to a single position: Near-optimal multi-positional mismatch attacks against Kyber and Saber. In: Johansson, T., Smith-Tone, D. (eds.) Post-Quantum Cryptography - 14th International Workshop,

PQCrypto 2023. pp. 291–320. Springer, Cham (Aug 2023). https://doi.org/10.1007/978-3-031-40003-2_11
20. Guo, Q., Mårtensson, E., Åström, A.: The perils of limited key reuse: Adaptive and parallel mismatch attacks with post-processing against Kyber. CiC **1**(3), 21 (2024). https://doi.org/10.62056/a3n5qj888
21. Guo, Q., Nabokov, D., Johansson, T.: Unlocking the true potential of decryption failure oracles: A hybrid adaptive-ldpc attack on ml-kem using imperfect oracles. In: 35th USENIX Security Symposium (USENIX Security 26). USENIX Association, Baltimore, MD (Aug 2026)
22. 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
23. Heinz, D., Kannwischer, M.J., Land, G., Pöppelmann, T., Schwabe, P., Sprenkels, A.: First-order masked Kyber on ARM Cortex-M4. Cryptology ePrint Archive, Report 2022/058 (2022), https://eprint.iacr.org/2022/058
24. 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
25. Hermelink, J., Ning, K.C., Petri, R.: Finding and protecting the weakest link - on side-channel attacks on y in masked ML-DSA. In: Kalai, Y.T., Kamara, S.F. (eds.) CRYPTO 2025, Part V. LNCS, vol. 16004, pp. 3–37. Springer, Cham (Aug 2025). https://doi.org/10.1007/978-3-032-01901-1_1
26. 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
27. 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
28. Hermelink, J., Streit, S., Mårtensson, E., Petri, R.: A generic framework for side-channel attacks against LWE-based cryptosystems. In: Fehr, S., Fouque, P.A. (eds.) EUROCRYPT 2025, Part VIII. LNCS, vol. 15608, pp. 3–32. Springer, Cham (May 2025). https://doi.org/10.1007/978-3-031-91101-9_1
29. Hesse, D., Feldtkeller, J., Güneysu, T., Hermelink, J., Krausz, M., Land, G.: t-probing (in-)security: Pitfalls on noise assumptions. Cryptology ePrint Archive, Report 2025/1202 (2025), https://eprint.iacr.org/2025/1202
30. Huber, M.F., Bailey, T., Durrant-Whyte, H.F., Hanebeck, U.D.: On entropy approximation for gaussian mixture random vectors. In: IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems, MFI 2008, Seoul, South Korea, August 20-22, 2008. pp. 181–188. IEEE (2008). https://doi.org/10.1109/MFI.2008.4648062, https://doi.org/10.1109/MFI.2008.4648062
31. Kannwischer, M.J., Pessl, P., Primas, R.: Single-trace attacks on Keccak. IACR TCHES **2020**(3), 243–268 (2020). https://doi.org/10.13154/tches.v2020.i3.243-268, https://tches.iacr.org/index.php/TCHES/article/view/8590
32. 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
33. Kret, E.: Quantum resistance and the signal protocol (2023), https://signal.org/blog/pqxdh/
34. Langlois, A., Stehlé, D.: Worst-case to average-case reductions for module lattices. DCC **75**(3), 565–599 (2015). https://doi.org/10.1007/s10623-014-9938-4
35. Li, J., Cheng, C., Shen, M., Chen, P., Guo, Q., Liu, D., Wu, L., Weng, J.: Grafted trees bear better fruit: An improved multiple-valued plaintext-checking side-channel attack against kyber. In: Design, Automation & Test in Europe Conference, DATE 2025, Lyon, France, March 31 - April 2, 2025. pp. 1–7. IEEE (2025). https://doi.org/10.23919/DATE64628.2025.10992764, https://doi.org/10.23919/DATE64628.2025.10992764
36. MacKay, D.J.C.: Information Theory, Inference & Learning Algorithms. Cambridge University Press, USA (2002)
37. 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
38. 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
39. Nagpal, R., Cassiers, G., Primas, R., Knoll, C., Pernkopf, F., Mangard, S.: On loopy belief propagation for SASCAs: An analysis and empirical study of the inference problem. CiC **1**(4), 15 (2024). https://doi.org/10.62056/ayl8ksdja
40. 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
41. National Institute of Standards and Technology: Module-lattice-based key-encapsulation mechanism standard. Tech. rep., Department of Commerce, Washington, D.C. (2023), federal Information Processing Standards Publication (FIPS) NIST FIPS 203 ipd. https://doi.org/10.6028/NIST.FIPS.203.ipd
42. National Institute of Standards and Technology: Status report on the fourth round of the nist post-quantum cryptography standardization process. Tech. rep., Department of Commerce, Washington, D.C. (2025), nIST Internal Report NIST IR 8545 https://doi.org/10.6028/NIST.IR.8545
43. NewAE Technology Inc.: CW308 UFO Board Documentation (2025), https://chipwhisperer.readthedocs.io/en/latest/Targets/CW308%20UFO.html
44. Ngo, K., Dubrova, E., Guo, Q., Johansson, T.: A side-channel attack on a masked IND-CCA secure Saber KEM implementation. IACR TCHES **2021**(4), 676–707 (2021). https://doi.org/10.46586/tches.v2021.i4.676-707, https://tches.iacr.org/index.php/TCHES/article/view/9079
45. O'Flynn, C., Chen, Z.D.: ChipWhisperer: An open-source platform for hardware embedded security research. In: Prouff, E. (ed.) COSADE 2014. LNCS, vol. 8622, pp. 243–260. Springer, Cham (Apr 2014). https://doi.org/10.1007/978-3-319-10175-0_17
46. Pearl, J.: Reverend bayes on inference engines: A distributed hierarchical approach. In: Waltz, D.L. (ed.) Proceedings of the National Conference on Artificial Intelli-

gence, Pittsburgh, PA, USA, August 18-20, 1982. pp. 133–136. AAAI Press (1982), http://www.aaai.org/Library/AAAI/1982/aaai82-032.php
47. 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
48. 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
49. Poilbout, P., Roche, T., Imbert, L.: Handling noisy plaintext checking oracles with SPiRiT. In: Bardet, M., Niederhagen, R. (eds.) Post-Quantum Cryptography. pp. 311–341. Springer Nature Switzerland, Cham (2026)
50. 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
51. Qin, Y., Cheng, C., Ding, J.: An efficient key mismatch attack on the NIST second round candidate Kyber. Cryptology ePrint Archive, Report 2019/1343 (2019), https://eprint.iacr.org/2019/1343
52. Qin, Y., Cheng, C., Zhang, X., Pan, Y., Hu, L., Ding, J.: A systematic approach and analysis of key mismatch attacks on lattice-based NIST candidate KEMs. In: Tibouchi, M., Wang, H. (eds.) ASIACRYPT 2021, Part IV. LNCS, vol. 13093, pp. 92–121. Springer, Cham (Dec 2021). https://doi.org/10.1007/978-3-030-92068-5_4
53. Rajendran, G., Ravi, P., D'Anvers, J.P., Bhasin, S., Chattopadhyay, A.: Pushing the limits of generic side-channel attacks on LWE-based KEMs - parallel PC oracle attacks on Kyber KEM and beyond. IACR TCHES **2023**(2), 418–446 (2023). https://doi.org/10.46586/tches.v2023.i2.418-446
54. Ravi, P., Chattopadhyay, A., D'Anvers, J.P., 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) (Mar 2024). https://doi.org/10.1145/3603170, https://doi.org/10.1145/3603170
55. Ravi, P., Roy, S.S., Chattopadhyay, A., Bhasin, S.: Generic side-channel attacks on CCA-secure lattice-based PKE and KEMs. IACR TCHES **2020**(3), 307–335 (2020). https://doi.org/10.13154/tches.v2020.i3.307-335, https://tches.iacr.org/index.php/TCHES/article/view/8592
56. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. In: Gabow, H.N., Fagin, R. (eds.) 37th ACM STOC. pp. 84–93. ACM Press (May 2005). https://doi.org/10.1145/1060590.1060603
57. Shao, M., Liu, Y., Zhou, Y.: Pairwise and parallel: Enhancing the key mismatch attacks on Kyber and beyond. In: Zhou, J., Quek, T.Q.S., Gao, D., Cárdenas, A.A. (eds.) ASIACCS 24. ACM Press (Jul 2024). https://doi.org/10.1145/3634737.3637661
58. Shen, M., Cheng, C., Zhang, X., Guo, Q., Jiang, T.: Find the bad apples: An efficient method for perfect key recovery under imperfect SCA oracles - A case study of Kyber. IACR TCHES **2023**(1), 89–112 (2023). https://doi.org/10.46586/tches.v2023.i1.89-112
59. Strieder, E., Ilg, M., Heyszl, J., Unterstein, F., Streit, S.: ASCA vs. SASCA - A closer look at the AES key schedule. In: Kavun, E.B., Pehl, M. (eds.) COSADE 2023. LNCS, vol. 13979, pp. 65–85. Springer, Cham (Apr 2023). https://doi.org/10.1007/978-3-031-29497-6_4

60. Tanaka, Y., Ueno, R., Xagawa, K., Ito, A., Takahashi, J., Homma, N.: Multiple-valued plaintext-checking side-channel attacks on post-quantum KEMs. IACR TCHES **2023**(3), 473–503 (2023). https://doi.org/10.46586/tches.v2023.i3.473-503
61. 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
62. Ueno, R., Xagawa, K., Tanaka, Y., Ito, A., Takahashi, J., Homma, N.: Curse of re-encryption: A generic power/EM analysis on post-quantum KEMs. IACR TCHES **2022**(1), 296–322 (2022). https://doi.org/10.46586/tches.v2022.i1.296-322
63. Veyrat-Charvillon, N., Gérard, B., Standaert, F.X.: Soft analytical side-channel attacks. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014, Part I. LNCS, vol. 8873, pp. 282–296. Springer, Berlin, Heidelberg (Dec 2014). https://doi.org/10.1007/978-3-662-45611-8_15

Appendix

A Joint Distribution Hints

Distribution hints [28] offer a rather generic way to exploit side-channel information linear in the secret key. In our setting, however, they cannot be used, since both distribution hints and previous hint definitions (e.g., [7]) assume *independent* linear information, whereas we obtain *dependent* linear hints in the secret key. To overcome this limitation, we propose *joint distribution hints*:

Definition 2. *Let* $(\mathbf{w}_j)_{j\in[L]} \in (\mathbb{R}^N)^L$*, where* N *is the dimension of the secret key* $\mathbf{x} = (\mathbf{e}, \mathbf{s})$*,* $\mathcal{D}$ *be a probability distribution on* $\mathbb{R}^L$*, such that*

$$(\mathbf{w}_j^\top \mathbf{x})_{j\in[L]} \sim \mathcal{D}. \tag{19}$$

We then call the tuple $((\mathbf{w}_j)_{j\in[L]}, \mathcal{D})$ *a* joint distribution hint*.*

This definition is a strict generalization of distribution hints: distribution hints correspond to $L = 1$, while $L \geq 2$ with a distribution $\mathcal{D}$ that does not factor into independent distributions on each coefficient captures information that distribution hints do not cover. Note that when obtaining multiple joint distribution hints, L need not be the same for each hint.

A.1 BP-based solver.

Joint distribution hints may be solved using BP on a similar factor graph as in [28]: Each secret key coefficient is represented by a variable node. A single joint distribution hint is represented by a factor node that is connected to all variable nodes for which any $w_{i,j}$ is non-zero. The factor nodes compute the distributions of $\sum_{i,i\neq k}^{N-1} w_{j,i}s_i$ for all $k \in [L]$ and $j \in [L]$. Denote the probabilities of these distributions by $q_{j,k}(v)$. To update the belief for key coefficient i, denoted by $p_i(v)$ for $s_i = v$, we compute

$$p_i(v) \leftarrow \sum_{\mathbf{a}\in \operatorname{supp}\mathcal{D}} P_{\mathcal{D}}(\mathbf{a}) \prod_{j=0}^{N-1} q_{j,i}(a_j - v), \tag{20}$$

thereby updating the probability for $s_k = v$ using

$$\left(\sum_{i=0}^{N} w_{j,i}s_i\right)_j = \left((w_{j,k}s_k) + \left(\sum_{i=0,i\neq k}^{N} w_{j,i}s_i\right)\right)_j \sim \mathcal{D} \tag{21}$$

by computing the distribution of $(\sum_{i=0,i\neq k}^{N} w_{j,i}s_i)_j$ from the incoming messages – the current belief in the (other) variables. The latter computation can be implemented in $O(N\log(N))$ using a Fast Fourier Transform (FFT) and a forward-backward iteration as described in [28].

For joint distribution hints, the solver described here gives a BP instantiation equivalent to, though less straightforward than, the one in Section 3.1. For evaluations, we use an easily accessible Python implementation, and additionally provide a highly optimized, generic Rust version based on the open-source implementation of [28]. For hints with low information content, a filtering technique as in [25] may be necessary. Further, the BP, as well as the greedy solver below, may be improved using subsequent lattice reduction as described in [24,28].

A.2 Greedy solver.

Derived from the BP instantiation similarly to [28], a greedy solver stores a guess, scores the potential changes of the guess, then performs the highest-scored changes. For joint distribution hints, the score for changing coefficient i by $+c$ can be computed as $s_i(c) = \sum_{a \in \operatorname{supp} \mathcal{D}} P_{\mathcal{D}}(a) \prod_{j=0}^{N} |\mathbf{v}_{j,i}^{\top} \mathbf{x}' + v_{j,i} c - a|$. Thus, the computationally intensive calculation of N partial sums of $(N-1)$ random variables per hint is avoided and replaced by computing a distance. Implementation and evaluation of a greedy solver is left for future work.

B Physical Evaluation on a Masked Implementation

To verify that our attack works in practice, we implemented an experimental setup targeting an `STM32F415`. The targeted device is placed on a ChipWhisperer `CW308` [45,43] base-board and its power consumption measured using a LeCroy WaveRunner 9254m oscilloscope. We used a series of two 50 MHz and two 25 MHz analog filters and a low-noise amplifier to condition the power signal, as the integrated switched-mode power supply of the target produces a considerable amount of high-frequency noise. We target the first-order arithmetically masked mkm4 [23] NTT. We modified mkm4 to emit a trigger signal for the oscilloscope during NTT operations to aid trace alignment; furthermore, we fixed an off-by-one error in the message recovery of mkm4. Furthermore, we implemented a bug-fix for the polynomial to binary message conversion, which exhibited an off-by-one error for some messages. We rely on a simple template-attack [6], where profiling is carried out on the NTT function with random inputs and selecting the five best PoIs per coefficient; this type of profiling can be carried out in practice *on the targeted device*. To identify PoIs within the profiling traces we correlated the samples against the Hamming weight of the known NTT output coefficients. We use linear discriminant analysis to train a template to predict the probability of each HW during the attack phase. This type of profiling leads to an unequal number of samples for different HWs, which potentially decreases the retrieved information. In addition, the coefficients in mkm4 are reduced to $\{0, \ldots, 3329\}$, and we treat 0 as equal to 3329, i.e., work on $\{0, \ldots, 3328\}$. The combination of arithmetic masking and unsigned coefficients substantially reduces the leaked information compared to Boolean masking or signed arithmetic masking (see, e.g., [37,61,25]). Our analysis shows that 32 of 256 coefficients have an average SNR (of targeting one share) of 0.971 with an SNR of 0.113 at the remaining coefficients.

Results for recovering a subkey. We ran 50 experiments with each targeting an $N \in \{2, 3, 4\}$ subkey using the optimal attack; we abort after a maximum of 200 calls and an entropy bound of 0.001; here, we use the entropy of the joint distribution as an abort criteria. Interestingly, there is only a small difference in the average required number of calls to recover a subkey: for $N = 2$, it is 95.58 with 3 runs being aborted at 200 queries of which one had not fully recovered the subkey but is off by one in one coefficient; for $N = 3$, we have 93.18 queries on average with 5 aborts at 200 and no incorrect keys; and for $N = 4$, we have 103.28 queries with one aborted case that converged against the correct key and one true error.

$N = 8$ *in the BP-based attack.* We evaluate the BP-based attack at $N = 8$ using 20 experiments, a maximum of 200 calls, and an abort entropy of 0.01. After an average of 158.35 calls, we achieve an average distance to the true subkey of 1.8 and 11 correct subkeys. Note that an entropy abort of 0.01 is less strict for the BP than for the optimal attack.

Comparing to tournament-based oracles. To assess the difference between using a tournament-based and a distribution oracle, we emulate a tournament-based oracle: We first run 1000 measurements and compute α as the ratio of "the most likely message is the correct message" to the total number of trials. During the attack, we use the distribution oracle to assign the most likely message the probability α. In the case of $N = 2$ (50 experiments), we get $\alpha = 0.477$, an average of 115.68 calls, and 5 aborts of which one is incorrect. However, for $N = 3$, at $\alpha = 0.264$, we already see 30 aborts and 6 incorrect subkeys (and 179.46 average calls, but the abort criteria biases this downwards). At $N = 4$, α is 0.248, and we observe one incorrect subkey before abort at 200, 32 aborts, and a total of 19 incorrect subkeys. At $N = 8$ (BP), using a tournament-based approach results in $\alpha = 0.025$, and average distance of 2.8, and no correct subkeys (in comparison to 11 correct subkeys and an average distance of 1.7). The results clearly show the advantage of using a distribution oracle over a tournament oracle.

Recovery of a full key. In addition, we ran our optimal attack against a full key, i.e., all subkeys of a fixed key, using $N = 3$. To reduce the overall runtime, we decreased the entropy abort criteria to 0.01 (instead of 0.001) and a maximum numbers of call of 200 resulting in slightly more incorrect subkey. We ran 10 experiments and obtained an average success rate of 91.69% recovery rate *per subkey*. One key was fully recovered immediately; on average, the Euclidean distance to the true key was 10.55 with an average of 28028.6 calls, which results in an almost trivially solvable lattice instance; notably, many non-recovered subkeys can be identified as incorrect as they ran into the maximum number of oracle calls or oscillated more frequently between states (on average 42.8 oscillations for correct subkeys compared to 100.4 for incorrect subkeys).

Further improvements. In practice, the adversary can drastically reduce the number of traces by using a higher entropy bound and correcting errors by

subsequent lattice reduction using [24]. We leave choosing these parameters, comparing against different target locations (e.g., arithmetic versus Boolean masking), advanced templates attacks, using neural networks, and extensive evaluations for future work.

C Proofs of Lemmas 1 and 2

Lemma 1. *For a sub-key* $\mathbf{s}$ *and queries* $(\mathcal{D}_j = \mathcal{O}(\mathbf{a}_j))_{j\in[K]}$ *for that sub-key, we have*

$$P\left(s_i = v \mid (\mathcal{D}_j = \mathcal{O}(\mathbf{a}_j))_{j\in[K]}\right) \propto P\left(s_i = v\right) \sum_{\mathbf{m}\in[2^N]^K} \prod_{j=0}^{K-1} P_{\mathcal{D}_j}\left(m_j\right) \delta_{v\leq a_{j,i}}(m_{j,i}), \tag{22}$$

where $\delta_A(b)$*, for a statement* A *and a bit* b*, is* 1 *if and only if* A *is true and* $b = 1$ *or* A *is false and* $b = 0$*.*

Proof. Given the actual message, the queries give no further information. Therefore, we have

$$\begin{aligned} &P\left(s_i = v \mid (\mathcal{D}_j = \mathcal{O}(\mathbf{a}_j))_{j\in[K]},\ m_{\text{actual},j} = m_j\ \forall j \in [K]\right) && (23)\\ =&P\left(s_i = v \mid m_{\text{actual},j} = m_j\ \forall j \in [K]\right), && (24)\end{aligned}$$

and as the messages correspond to queries that represent inequalities, this is, by Bayes' rule, proportional to

$$\begin{aligned} &P\left(s_i = v \mid m_{\text{actual},j} = m_j\ \forall j \in [K]\right) && (25)\\ \propto&P\left(s_i = v\right) P\left(m_{\text{actual},j} = m_j\ \forall j \in [K]\ \mid s_i = v\right) && (26)\\ \propto&P\left(s_i = v\right) \prod_{j=0}^{K-1} \delta_{v\leq a_{j,i}}(m_{j,i}). && (27)\end{aligned}$$

Note that the prior, a binomial distribution in ML-KEM, appears here as not all values fulfilling the inequality are equally likely; instead, they are proportional to the prior and the normalization factor is the sum of the prior probabilities of all values fulfilling the inequality. Now, putting it all together and using the law of total probability, we get

$$\begin{aligned} &P\left(s_i = v \mid (\mathcal{D}_j = \mathcal{O}(\mathbf{a}_j))_{j\in[K]}\right) && (28)\\ =&\sum_{\mathbf{m}\in[2^N]^K} P\left(s_i = v \mid\ m_{\text{actual},j} = m_j\ \forall j \in [K]\right) P_{\mathcal{D}}\left(\mathbf{m}\right) && (29)\\ =&\sum_{\mathbf{m}\in[2^N]^K} P\left(s_i = v \mid\ m_{\text{actual},j} = m_j\ \forall j \in [K]\right) \prod_{j=0}^{K-1} P_{\mathcal{D}_j}\left(m_j\right) && (30)\\ \propto&\sum_{\mathbf{m}\in[2^N]^K} P\left(s_i = v\right) \prod_{j=0}^{K-1} \delta_{v\leq a_{j,i}}(m_{j,i}) \prod_{j=0}^{K-1} P_{\mathcal{D}_j}\left(m_j\right). && (31)\end{aligned}$$

□

Lemma 2. *For K, $(\mathcal{D}_j = \mathcal{O}(\mathbf{a}_j))_{j\in[K]}$, the marginals of p, computed according to* (10)*, equal the conditional probabilities $P\left(s_i = v \mid (\mathcal{D}_j = \mathcal{O}(\mathbf{a}_j))_{j\in[K]}\right)$.*

Proof. We can write the probabilities computed according to (10) as

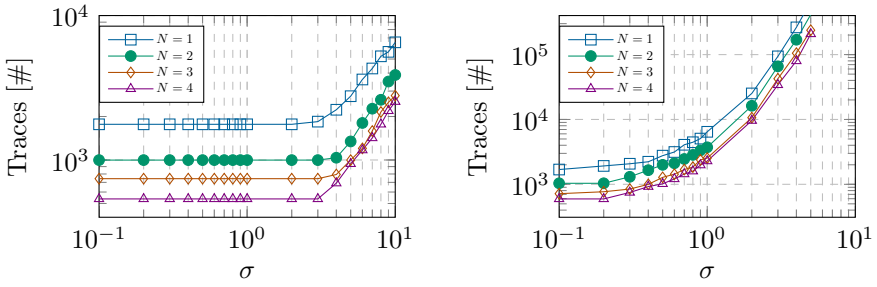
$$p(\mathbf{v}) = p_{\text{prior}}(\mathbf{v}) \sum_{(m_j)_j \in [2^N]^K} \prod_{j=0}^{K-1} P_{\mathcal{D}_j}(m_j) \prod_{i=0}^{N-1} \delta_{v_i \leq a_{j,i}}(m_{j,i}). \tag{32}$$

Computing the marginals for position i' results in

$$\sum_{\mathbf{v}, v_{i'}=v} p_{\text{prior}}(\mathbf{v}) \sum_{(m_j)_j \in [2^N]^K} \prod_{j=0}^{K-1} P_{\mathcal{D}_j}(m_j) \prod_{i=0}^{N-1} \delta_{v_i \leq a_{j,i}}(m_{j,i}) \tag{33}$$

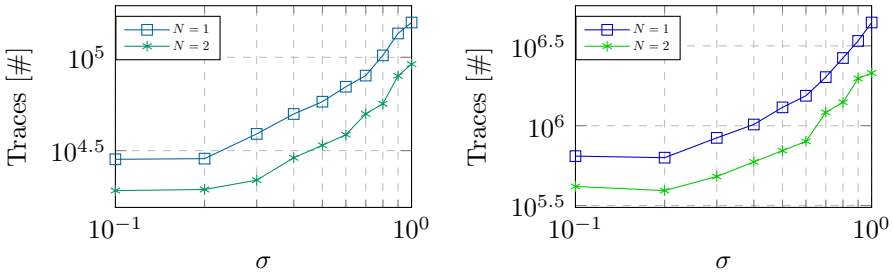
$$= \sum_{(m_j)_j \in [2^N]^K} \prod_{j=0}^{K-1} P_{\mathcal{D}_j}(m_j) \sum_{\mathbf{v}, v_{i'}=v} p_{\text{prior}}(\mathbf{v}) \prod_{i=0}^{N-1} \delta_{v_i \leq a_{j,i}}(m_{j,i}). \tag{34}$$

Now the last term, $\sum_{\mathbf{v}, v_{i'}=v} p_{\text{prior}}(\mathbf{v}) \prod_{i=0}^{N-1} \delta_{v_i \leq a_{j,i}}(m_{j,i})$, is the sum over all probabilities according to the prior for values that fulfill the inequalities given by the query responses and have $v_{i'} = v$. Given that the prior of $\mathbf{v}$ is just the product over the prior probabilities of the coefficients of $\mathbf{v}$, varying v changes the term by a factor given by the prior probability for the respective v. For the set of $v_{i'} = v$ that fulfills the inequality for i', this is proportional (in the choice for $v_{i'}$) to the prior; otherwise, it is zero. Therefore, (33) is proportional to the term (5) from Lemma 1. □



(a) Unmasked. (b) Two shares.

Fig. 10: Results in terms of oracle queries per noise level for an unmasked noisy HW oracle and a first-order masked noisy HW oracle targeting 100 NTT values.



(a) Three shares. (b) Four shares.

Fig. 11: Results in terms of oracle queries per noise level for second and third-order masked noisy HW oracle targeting 100 NTT values.

D Additional Figures

This section provides our evaluation results in terms of traces per noise level with graphs for each N and a figure for each masking order; in contrast to Section 5.1, where the same information is depicted with a graph for each masking order and a figure for each N. In Figure 10, we evaluate our attacks on an unmasked noisy HW oracle as well as on a first-order masked noisy HW oracle. In both cases, we target 100 values of the last layer of the NTT. In Figure 11, we evaluate a second and a third order masked noisy HW oracle.