# Full Secret Key Recovery of First-order Masked Crystals-Kyber implementation using multiple distinct chosen-ciphertexts

Souhayl Ben El Haj Soulami[1,3], Yann Connan[2], and Sylvain Duquesne[3]

[1] Valeo Brain Division, VMTC, 94000, Créteil, France,
souhayl.ben-el-haj-soulami@valeo.com
[2] Secure-IC S.A.S, Cesson-Sévigné, France, yann.connan@secure-ic.com
[3] Univ Rennes, CNRS, IRMAR - UMR 6625, F-35000 Rennes, France,
sylvain.duquesne@univ-rennes.fr

**Abstract.** We present a novel side-channel attack on first-order masked implementations of Crystals-Kyber. It deploys a new distinguisher in the context of post-quantum cryptography. It relies on combining the information from several instances of the same distinguisher via multiple ciphertexts decryption requests. The attack has been performed on simulation and illustrated on the masked implementation of *Bronchain et al.*. This attack is instantiated in a very noisy environment (Signal-to-Noise Ratio (SNR) of 0.67) and provides a success rate of 95% with 75 000 traces for full secret key recovery.

**Keywords:** Crystals-Kyber · masking · side-channel attack · distinguisher

## 1 Introduction

In 1 994, *Shor* introduced a quantum algorithm that recovers the order of an element inside a group within polynomial time, paving the way for polynomial-time solutions for prime factorisation and discrete logarithm problems [34]. As a consequence, classical algorithms for key exchange such as Rivest-Shamir-Adleman (RSA) and Elliptic-Curve-Diffie-Hellman (ECDH) whose security relies on the hardness of these problems are no longer secure from the perspective of a quantum attacker. With the recent advances in quantum computing, finding safe substitutes for those algorithms becomes paramount. To address this issue, the National Institute of Standards and Technology (NIST) launched a competition in 2 016 to standardise post-quantum Key Exchange Mechanism (KEM).

Candidates from various cryptographic families have been proposed: lattices, codes, isogenies among others. Within the panoply of choices, lattice-based alternatives offer the best compromise between performance and keys' sizes [18], making them suitable for most industrial applications. Among these candidates, the lattice-based Crystals-Kyber emerged as the only winner of the competition in 2 022 [40]. Crystals-Kyber was later standardised in FIPS 203 under the name

ML-KEM in 2 023 with minor modifications[4]. In this work, the attacks were conducted on Crystals-Kyber due to the open source masked implementations availability. However, the results can naturally be transposed to ML-KEM without any changes.

The security of the scheme is based on the Module-Learning-with-Error (MLWE) problem, a variant of the Learning-with-Error (LWE) problem proposed by *Regev* in [32] which is asymptotically as hard as GapSVP, a standard worst-case lattice problem [25,4]. Notwithstanding the computational hardness to solve this problem, implementations of cryptographic schemes can also be broken due to physical vulnerabilities. It is important to note that those physical vulnerabilities do not disprove the mathematical robustness above which the security of the cryptographic scheme is built. In fact, mathematical proofs are agnostic to the execution environment of the algorithms which may leak information about the sensitive data. Hence, physical attacks are in essence an alternative way to break a cryptographic scheme, and can be divided into two categories: fault-injection attacks [3] and side-channel attacks. Introduced by Paul Kocher in [20], side-channel attacks rely on physical measurements leaked by hardware circuits such as power consumption [23], electromagnetic emanation [22], execution time [7] or cache access time [8], to extract information about sensitive values. Classical countermeasures include masking [5], shuffling [37], dual-rail logic [9] or inserting dummy operations [21]. While masking remains the most popular countermeasure, it is still difficult to mask post-quantum asymmetric algorithms compared with symmetric cryptography where masking has proven to be effective [24,12]. One major obstacle is the necessity to switch between boolean and arithmetic masking [10,36].

Another major challenge is the small and non-uniform support of the coefficients of the secret vectors (only five possible values for the long-term key coefficients and two for the message in the case of Crystals-Kyber), which facilitate template attacks, due to the restricted number of distributions needed to be distinguished by an attacker [24]. Several masking implementations for Crystals-Kyber have been proposed such as the famous `mkm4` [16]. This implementation masks all the sensitive variables, such as the secret key, the seeds and the small error vectors and performs all operations such as compression functions, the NTT operation and the hashing on masked data. In the article that introduced the implementation, it is not explicitly stated that the mask is refreshed at each execution or not. However, in the source code, the mask of the secret variables is static, which means that it is generated only once during the key generation and is never refreshed. This implies that the boolean mask of the message is unchanged at each execution of the algorithm, opening the door to attacks targeting each share independently. By doing so, the message can be recovered, which in turn allows to retrieve the entire secret key using LDPC-codes [15]. Refreshing the mask at each execution of the algorithm seems like a logical and natural way of securing the implementation and avoiding this

[4] The standard is available in https://nvlpubs.nist.gov/nistpubs/fips/nist.fips.203.pdf

flaw as it is done in [6]. However, we will show that even in the case where the mask is refreshed at each execution, an attacker can exploit some leakages and mount an attack to recover the secret key. Indeed, splitting the secret into two or more pieces inevitably implies that the shares are mathematically correlated for a fixed secret $x$. By scrupulously analysing the energy consumption issued during the writing of each share in registers, it is possible to discriminate some parts of the masked secret value. Our work will show how it is possible, by exploiting specific chosen ciphertexts, to mount an attack based on their leakages that will recover the entire secret key.

**Contribution.** In this paper, our contributions are the following:

1. We introduce a new side-channel attack that recovers all secret key coefficients of masked implementations of Crystals-Kyber. The attack uses a series of decryption queries of several ciphertexts repeated a certain number of times. Moreover, the attack does not require having access to a clone device. Despite the attack requiring a substantial number of traces in comparison with the state-of-the art, it does not need a pre-processing phase.
2. The traces are analysed thanks to a distinguisher previously introduced to break the AES in [28]. It cannot be used directly to recover the entire secret key (that has 768 coefficients versus 8 for the AES). Then we introduce a variant of this distinguinsher that is specific to the context of lattice-based cryptography and we show how to combine different instances of this distinguisher to drastically improve its accuracy. We then get a new global distinguisher that can recover the whole secret key.
3. The attack is instantiated on the implementation of *Bronchain et al.*. For example, it is possible to recover the entire secret key with 75 000 traces under the usually considered low SNR of 0.67 with a 95% probability of success.

**Outline.** The remainder of this paper is organised as follows. Section 2 presents some previous attacks on Crystals-Kyber. Section 3 sets the notations. Section 4 briefly presents the Crystals-Kyber scheme. Section 5 describes the bias when splitting a secret value into two shares and introduces a distinguisher to exploit this bias. Section 6 shows the results of the attack on simulation. Section 7 concludes the article.

## 2 Related Work

Previous attacks on masked Crystals-Kyber implementations are numerous and can be classified according to a number of criteria such as targeted asset (secret key or shared key), targeted node (NTT, hash function, message encoding, etc...), technique used (machine learning, DPA/CPA, SASCA, ...), number of traces, noise tolerance (low-noise, high-noise) and targeted implementation (software or hardware implementation).

### 2.1 Message Recovery Attacks

Message recovery attacks on lattice-based schemes first appears in 2 020, targeting the message encoding operation of the Newhope key-exchange protocol and recovering the entire shared secret from a single trace [1]. The same year, a similar attack was conducted on several post-quantum key exchange mechanisms, namely an optimized implementation of Crystals-Kyber, Saber, FrodoKEM and NTRUPrime [35]. The use of the Fujisaki-Okamoto transform [13] in Crystals-Kyber induces a side-channel weakness exploited in a chosen-ciphertext attack using error-correcting codes and recovering the message's coefficients one at a time [31]. In 2 021, *Xu et al.* [39] mounted another chosen-ciphertext attack which consists to modulate the leakage of a target device, recovering the entire message thought simple power analysis with a single trace. The next year, *Mengyao et al.* [33] showed that the message could also be recovered by leveraging the information leakage during the decoding operation.

Following these series of attacks, since the message encoding/decoding proved to be vulnerable, various masked implementations of Crystals-Kyber, namely *Heinz et al.* [16] and *Bronchain et al.* [6] were proposed. In 2 022, the first practical attack on a masked message encoding was introduced by *Cao et al.* [38]. The following year, *Jendral et al.* presented a fault injection attack by-passing the shuffling and recovering the message using deep learning-assisted profiled power analysis [17].

### 2.2 Long-term Key Recovery

Attacks aiming at recovering directly the long-term secret key can be made by very different angles. The first attack by *Kannwischer et al.* targets the Crystals-Kyber scheme, and more specifically the KECCAK function during the key generation or the decapsulation [19]. The idea is to guess the input value of the hash function from the leakage, and more specifically the seeds used to generate the secret key and the small error vectors.

Another possibility is to target the NTT during the decryption, as shown by *Primas et al.* [27]. It is the first attack on masked lattice-based algorithms using Soft-Analytical-Side-Channel-Attack (SASCA) which fully recovers the secret key. This attack was further improved by *Pessl and Primas* [26].

Finally, the most efficient attacks are template attacks which exploit a combination of specific chosen-ciphertexts queries and target the message encoding step during the re-encryption procedure [30]. The idea is to handcraft a set of ciphertexts such that the corresponding messages, obtained after decryption, are dependent on a portion of the secret key. This could be, among others, the value of a specific bit $m'$ of the decrypted message. This information is obtained through side-channel leakage during the decapsulation procedure. Once the value of $m'$ is determined, we can infer that portion of the secret key. In practice, the attacks work on two phases. The first phase is the *pre-processing* where we build templates according to different values of $m'$. Then, there is a *key recovery phase* where the attacker records traces based on chosen-ciphertext

queries and matches them with the templates to guess the secret key. These attacks were initially used by *d'Anvers et al.* on LAC, a Ring-LWE-based KEM [11]. Using timing analysis, each query allowed the recovery of a message bit which in turn revealed a single bit of the secret key. Then, *Ravi et al.* designed oracle queries to attack Crystals-Kyber using power/EM leakage [31]. With this oracle, each query recovers a single-bit of information about the secret key.

This attack was later improved by *Rajendran et al.* [29] by proposing oracle queries that recover $P$ bits of information (with $P < 10$), thus reducing the number of queries/traces by a factor of $P$. However, this comes at an exponential cost $O(2^P)$ in the number of templates during the *pre-processing* phase.

*Xu et al.* pushed the number of long-term secret key bits recovered to 256 in a single-trace [39]. This allows the recovery of the full secret-key in 4 traces in the case of `Kyber1024`. On-going research aims at reaching optimality in terms of number of queries and reducing the data and computational complexity for the *profiling* phase [15].

To the best of our knowledge, our work takes a different approach from those of the state-of-the-art by targeting the behavior of the distribution of the shares when combined with certain inputs. While profiling attacks are unarguably very attractive in terms of number of traces, they need to have access to a clone device for the training phase. Our attack becomes more attractive in a context where this is not possible.

## 3 Preliminaries and Notations

### 3.1 Polynomial Rings and Vectors

- Sets will be denoted by calligraphic letter, e.g. $\mathcal{S}$.
- A bold upper-case letter $\mathbf{M}$ is a matrix of polynomials and bold lower-case letter $\mathbf{v}$ is a column vector of polynomials. For a vector $\mathbf{v}$ (resp. a matrix $\mathbf{A}$), $\mathbf{v}^\intercal$ (resp. $\mathbf{A}^\intercal$) denotes the transpose and $\omega(\mathbf{v})$ denotes the Hamming weight of $\mathbf{v}$.
- We denote by $\mathcal{R}_q$ the ring $\mathbb{Z}_q[X]/(X^N+1)$ whose elements are polynomials of degree $N-1$ with coefficients in $\mathbb{Z}_q$, where $\mathbb{Z}_q$ denotes the ring $\mathbb{Z}/q\mathbb{Z}$. $\mathcal{R}_q^{k\times l}$ denotes the set of matrices of size $k \times l$ of elements in $\mathcal{R}_q$. We denote by $[\![a; b]\!]$ the set of integers $e \in \mathbb{Z}$ such that $a \leqslant e \leqslant b$.
- For polynomials, we use lower case. If $\mathbf{s}$ is a vector of polynomials, $\mathbf{s}[i]$ denotes its $i$-th polynomial, and $s[i]_j$ denotes the $j$-th coefficient of the polynomial, *i.e.* $\mathbf{s}[i] = \sum_{j=0}^{N-1} s[i]_j X^j$.

### 3.2 Distance

Throughout this paper, distance will always refer to the Euclidean distance.

### 3.3 Centered Binomial Distribution

- For random variables $X$ and $Y$, $X \perp\!\!\!\perp Y$ denotes that $X$ and $Y$ are independent variables.
- For a random variable $X$, $\mathbb{E}[X]$ denotes the expectation of $X$ and $Var(X)$ denotes its variance.
- For a set $\mathcal{S}$, we write $x \leftarrow \mathcal{U}(\mathcal{S})$ to denote that $x$ is chosen uniformly at random from $\mathcal{S}$.
- For a polynomial $p = \sum_{i=0}^{N-1} p_i X^i$, we denote by $p \leftarrow \beta_\eta(\mathcal{R}_q; \sigma)$ that $p$ is sampled according to the centered binomial distribution of parameters $\eta, \mathcal{R}_q$ and $\sigma$ if for all $i \in [\![0, N-1]\!], p_i \in \mathcal{R}_q$ and $p_i \leftarrow \beta_\eta$ generated from a seed $\sigma$.

### 3.4 Compression/Decompression Functions

Crystals-Kyber is using compression and decompression functions. Their purpose is to reduce the size of the keys and the ciphertext. They allow to discard some bits of the ciphertext while introducing a cryptographically low decryption failure probability. They have consequences on our attack, more precisely on the choice of the ciphertext request, but it is not necessary to give their exact formula (see section A.1 for more details).

### 3.5 Number Theoretic Transform

Let $(\zeta_i)_{1 \leq i \leq n}$ the $n$ roots of unity modulo $q$ and $f \in \mathcal{R}_q$.

$$\mathsf{NTT}(f) = \sum_{i=0}^{n} \hat{f}_i X^i \quad \text{s.t.} \quad \hat{f}_i = \sum_{j=0}^{n} f_j \zeta_i^j.$$

The inverse of the NTT is called the Inverse Number Theoretic Transform and is noted as INTT. The pointwise polynomial multiplication of two polynomials is defined as: $\forall u, v \in \mathcal{R}_q, \forall i \in [\![0, n-1]\!], (u \circ v)[i] := u[i] \cdot v[i]$. All these operations performed on $\mathcal{R}_q$ are extended to $\mathcal{R}_q^k$ by applying them component-wise.

### 3.6 Encode/Decode Functions

In Crystals-Kyber, polynomials need to be serialised and deserialised at different moments of the algorithm. `Encode` (resp. `Decode`) allow the serialisation (resp. deserialisation) of the polynomials. The serialization is the process of converting the data into a format that is suitable for storage and transmission.

### 3.7 Traces

When acquiring a trace $t$ during execution, we measure for example the power consumption at different moments $i$ (sample) of the algorithm, we hence obtain a series of samples per trace $t$. We denote by $l_i^{(t)}$ the leakage of the trace $t$ for the $i^{th}$ sample. During the acquisition of traces, the imperfections in the measurements induce a noise $n_i^{(t)}$, which denotes the noise associated with the trace $t$ at sample $i$. This noise usually follows the Gaussian distribution with mean $\mu$ and standard deviation $\sigma$ denoted by $\mathcal{N}(\mu, \sigma)$ [24].

### 3.8 Masking Technique

Masking was introduced by Goubin and Patarin in [14] as a countermeasure against electromagnetic/power side-channel attacks. The goal of the countermeasure is to break the correlation between the observed traces and the sensitive value. Let $x$ be a sensitive value and $f$ a function. Computing $f(x)$ involves manipulating $x$. The idea of masking is to split the value $x$ into multiple variables, called shares, and to perform computations on the shares rather than using $x$ itself. For example, if $f$ is linear and $x = x_1 + x_2$, computing $f(x)$ boils down to compute $f(x_1) + f(x_2)$. Although the expressions are mathematically equivalent, $f(x_1)$ and $f(x_2)$ are not correlated to the sensitive value $x$ if $x_1$ is chosen uniformly at random. The sensible variable can be split into more than two shares. When divided into two shares, the mask is called of order 1, generally, a split into $n$ shares is called a masking of order $n-1$. The way the shares are generated depends on the computations to be executed. For logical operations such as `AND` and `OR`, the split will be done using the boolean `XOR` operation $x = x_1 \oplus x_2$.

## 4 Crystals-Kyber

### 4.1 Presentation

Crystals-Kyber is a lattice-based post-quantum key exchange scheme whose security is based on the hardness of the MLWE problem over lattices[5]. Crystals-Kyber involves computations over the polynomial ring $\mathcal{R}_q$. The secret key is comprised of polynomials $\mathbf{s} = (s[0], \ldots, s[k-1]) \in \mathcal{R}_q^k$. Each coefficient $s[i]_j$ of the polynomial $s[i] = \sum_{j=0}^{N-1} s[i]_j X^j$ is sampled according to the Centered Binomial Distribution (CBD) over $\mathcal{S} = [\![-\eta_1; \eta_1]\!]$ (cf. Tab 1). The size $k$ of the vector and the size $\eta_1$ of the interval depend on the security level (cf. Tab 4). In this article, we choose the `kyber768` version as a case study although the same phenomenons are observed for `kyber512` and `kyber1024`. A simplified description of the algorithm can be found in the appendix, section A.3.

Table 1: Distribution law for secret values when $\eta_1 = 2$

| $i$ | -2 | -1 | 0 | 1 | 2 |
|---|---|---|---|---|---|
| $P(s = i)$ | 0.0625 | 0.25 | 0.3725 | 0.25 | 0.0625 |

### 4.2 Crystals-Kyber Construction

During the key exchange mechanism protocol, the attacker is supposed to be active, which means that not only he can eavesdrop communications, but he can also encrypt, decrypt and send ciphertexts of his choice to both parties. To

[5] See https://pq-crystals.org/kyber/data/kyber-specification-round3-20210131.pdf for the full specification

prevent this type of attacks, the structure of Crystals-Kyber consists of an IND-CPA-secure encryption scheme (described in Appendix section A.3), based on the MLWE, nested within an IND-CCA2-secure Key Encapsulation Mechanism.

The public key encryption part ensures that the scheme is secure against chosen plaintext attacks, meaning an attacker cannot distinguish between two ciphertexts despite having access to a decryption oracle. To resist chosen ciphertext attacks (CCA2), the IND-CPA scheme is transformed using the Fujisaki-Okatomo Transform (FO-transform) [13], which adds checks to prevent attackers from tampering with ciphertexts and exploiting decryption oracles (see Appendix section A.4). Following figure 1, it begins with *Party 1* generating a fresh key pair using KYBER.CCA.KEM.Keygen() (cf. figure 1). *Party 1* sends the public key to *Party 2*. *Party 2* generates a message $m$ at random and encapsulates it using KYBER.CCA.KEM.Encaps() then, sends the ciphertext $c$ to *Party 1* and derives the shared key. *Party 1* decapsulates the ciphertext using KYBER.CCA.KEM.Decaps(). It decrypts it first with KYBER.CPA.PKE.Dec(), then re-encrypts the message using KYBER.CPA.PKE.Enc() and compares it with the original ciphertext $\mathbf{c}$ to ensure the correctness of the decryption. The shared key $K$ is derived if and only if the ciphertexts match each other.

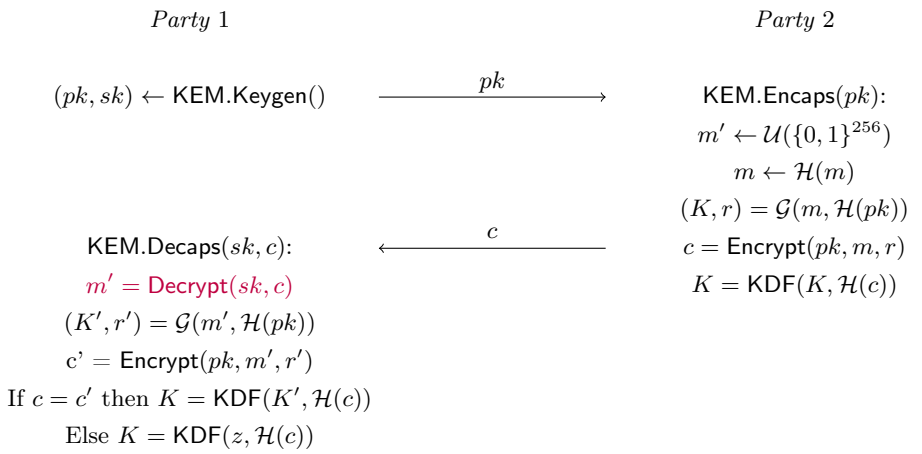



Fig. 1: Crystals-Kyber shared key establishment protocol

The attack targets the Decrypt operation. In the following sections, we exhibit the bias in the joint distribution of the shares, we explain how it can be exploited by a distinguisher to recover coefficients of the secret key and we explain how to leverage this observation to mount a full attack on a masked implementation of Crystals-Kyber.

# 5 Bias Exploitation

## 5.1 Adversary Model

The goal of the attacker is to retrieve the long-term key $sk$ through $\mathbf{s}$ ($sk$ before encoding, see section A.3 for more details). We assume that the attacker has access to the Device Under Analysis (DUA) and can run the KYBER.CCA-.KEM.Decaps() decapsulation algorithm of Crystals-Kyber. The attacker is endowed with necessary equipment and sufficient time to measure the leakage of the device. Moreover, we suppose that the attacker can run the decapsulation function with the ciphertext of their choice as many times as they wish. This is a valid hypothesis in the context of an IND-CCA2 protocol.

## 5.2 Leakages from the Hamming Weight of the Shares

Our attack is based on the exploitation of leakages occurring during the shares generation in a masking implementation. Let us consider a classical masking of a secret $s$ using two shares (that is a masking of order 1) $x_1$ and $x_2$ such that $s = x_1 + x_2 \mod q$. Consider an adversary that analyzes the Hamming weight of $x_1$ and $x_2$ by evaluating the energy consumption of those values during the splitting process. The distributions associated with those Hamming weight couples can be described as $\mathcal{D}_s = \{(\omega(x_1), \omega(x_2)) \in \mathbb{Z}_q^2, \forall x_1 \in \mathbb{Z}_q, x_2 \mid s = x_1 + x_2\}$ and are visually described in figure 2.

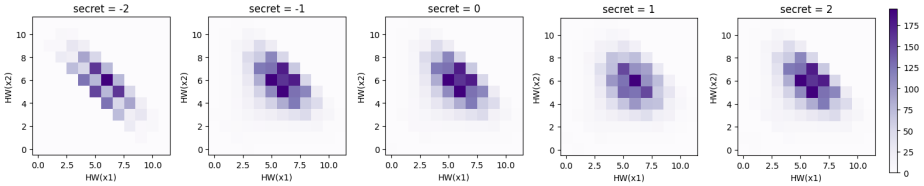



Fig. 2: Compound distribution $(\omega(x_1), \omega(x_2))_s$, $q = 3329$

As we can see, those distributions differ according to the secret value $s$. Notice that these discrepancies in the distributions are not specific to the Crystals-Kyber modulus $q = 3329$, they exist for other values of $q$, with different shapes and forms. This is not a new phenomenon, as it has already appeared in a previous work [24], in a less detailed form. As we can see, some distributions are distinguishable in naked eyes from the others, as is the case for $s = -2$ and $s = 1$, but the others are too close to distinguish them in an efficient way.

What becomes really interesting is when we analyze the distributions $\mathcal{D}_s^a = \{(\omega(a \cdot x_1), \omega(a \cdot x_2)) \in \mathbb{Z}_q^2, \forall x_1 \in \mathbb{Z}_q, x_2 \mid s = x_1 + x_2\}$ for some well-chosen integers $a$, as depicted in figure 3 with the example of $a = 13$.

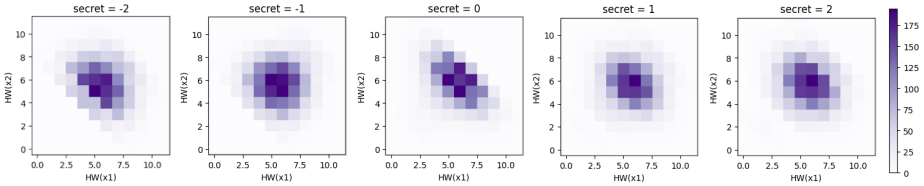



Fig. 3: Compound distributions $\mathcal{D}_s^{13}$

Not only are the distributions different, but more importantly we are able to distinguish different secrets compared to the previous distributions. This property is at the heart of our attack, which will combine the information leaked from different distributions to recover all the coefficients of the secret value $\mathbf{s}$.

We stress that in the case of Crystals-Kyber this approach is particularly suitable as the polynomial coefficients of the secret key $s$ belong to small sets, reducing the number of distribution templates required to mount the attack.

This preliminary analysis did not take into account the noise that existed during the acquisition of those leakages. Therefore, the following section introduces a distinguisher that allows us to exploit those leakages even in a very noisy environment.

### 5.3 Leakage Model

Let $x_i$ be an element of $\mathbb{Z}_q$, corresponding to a share of the secret key, for $i \in \{1, 2\}$ and $l(x_i)$ the leakage associated to it. We assume as usual that the leakage follows the classical Hamming weight model [23]:

$$l(x_i) = \omega(x_i) + n_i^{(t)}.$$

This means that the leakage corresponds to the power consumption/ElectroMagnetic (EM) emanation comprised of a deterministic part which is the Hamming weight of the input $x_i$ of the algorithm and a random noise. The random noise is generated due to many factors such as the imperfections in the circuit or the influence of external sources. We assume classically as well that the deterministic part and the noise are independent from one another, that the different noise sources are independent and that the noise follows a Gaussian distribution $n_t \leftarrow \mathcal{N}(0, \sigma)$ [23]. The noise level is crucial because it impacts the quality of the traces and hence the relevance of the attacks. One metric to measure the quality of the traces is through the SNR defined as :

$$SNR = \frac{\mathrm{Var}_{x_i}(\omega(x_i))}{\mathrm{Var}(n_t)} = \frac{\mathrm{Var}_{x_i}(\omega(x_i))}{\sigma^2}.$$

We can compute the exact distribution of $\omega(x_i)$ which hovers between 0 and 11, $[1, 12, 66, 219, 478, 728, 784, 596, 313, 108, 22, 2, 0]$. This allows us to compute the exact value of the variance and therefore, the SNR becomes:

$$SNR \approx \frac{2.67}{\sigma^2}.$$

### 5.4 Leakage Product Distinguisher

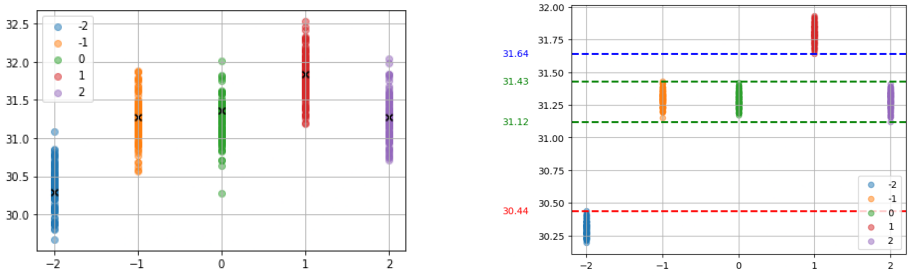
In the case of masked implementations (as considered in this paper), each share $x_i$ does not bring any information on the secret, so studying their leakage one by one is not useful. However the correlation between $x_1$ and $x_2$ is related to the secret key, so it is interesting to jointly observe the behaviour of $\omega(x_1)$ and $\omega(x_2)$. One natural way to do that is via considering the product $\omega(x_1) \times \omega(x_2)$ as it is done for example in [28] in the context of a masked AES implementation. We then first apply the Leakage Product Distinguisher to the Crystals-Kyber context. Let $X_1$ and $X_2$ the random variables associated to the first and second share. The expectation of $\omega(X_1) \times \omega(X_2)$ can be described as follows:

$$\mathbb{E}_s[\omega(X_1) \cdot \omega(X_2)] = \frac{\sum_{x_1 \in \mathbb{Z}_q} \omega(x_1) \cdot \omega(s - x_1)}{q}.$$

Let $l_1^{(t)}$ and $l_2^{(t)}$ be the leakages resulting from the manipulation of the first and the second shares at the execution number $t$ and $l^{(t)} = l_1^{(t)} \cdot l_2^{(t)}$. The total number of traces is denoted by $T$. To estimate $\mathbb{E}_s[\omega(X_1) \cdot \omega(X_2)]$, we use the empirical mean of the leakage

$$\bar{l} = \frac{\sum_{t=0}^{T-1} l^{(t)}}{T}.$$

Therefore, $\bar{l}$ converges towards $\mathbb{E}_s[\omega(X_1) \cdot \omega(X_2)]$ when we increase the number of traces. Indeed, as shown in Fig. 4a, when we add the noise, the distribution of $\bar{l}$ is centered around $\mathbb{E}_s[\omega(X_1).\omega(X_2)]$ (marked with $\times$).




(a) $T = 6,000$, $\sigma = 2$ (b) $T = 100,000$ and $\sigma = 2$

Fig. 4: Empirical leakage with different number of traces, $\sigma = 2$, $q = 3\,329$

Hence, we choose the number of traces $T$ such as the distributions of the empirical leakages of some secret values are disjoint with a very high probability

from the rest as it is the case in Fig. 4b for $-2$ and 1. When $s = -2$, $\bar{l} \leq 30.44$, when $s = 1$, $\bar{l} \geq 31.64$ and when $s \in \{-1, 0, 2\}$, $31.12 \leq \bar{l} \leq 31.43$.

This way, our guess knowing $\bar{l}$, is the $s$ that minimises the distance between $\mathbb{E}_s[\omega(X_1) \cdot \omega(X_2)]$ and the observed $\bar{l}$. Formally:

$$s_{guess} = \arg\min_{s \in \mathcal{S}}(|\bar{l} - \mathbb{E}_s[\omega(X_1) \cdot \omega(X_2)]|). \tag{1}$$

With this strategy directly drawn from the state of the art, we can recover all the coefficients of the secret that are equal to $-2$ or 1 with a reasonable number of traces. This corroborates the observations in figure 2. In practice, this allows on the average case to recover around 31% of the secret key according to Tab. 1. So, there are about 529 coefficients left with the three potential values $-1$, 0 and 2. This amounts to $3^{529}$ remaining possibilities for the secret key which far exceeds the considered security level and makes the classical strategy inoperative in the Kyber context. In the next section, we then propose a new variant of this distinguisher that improves the results.

### 5.5 New Variant of Leakage Product Distinguisher

In the previous section, we have seen that we are able to distinguish $-2$ and 1 from $\omega(X_1) \cdot \omega(X_2)$. Our first idea of improvement is to explore what information can be obtained from observing $\omega(a \cdot X_1) \cdot \omega(a \cdot X_2)$ for a given $a$. Indeed, we will show in section 5.7.3 that, in practice, an attacker can get his hand on this kind of distributions by manipulating the choice of $a$ with carefully fabricated ciphertext requests.
Let $a \in \mathbb{Z}_q$. If $a \cdot s = a \cdot X_1 + a \cdot X_2$, then:

$$\begin{aligned}
\mathbb{E}_a &:= \mathbb{E}[\omega(a \cdot X_1) \cdot \omega(a \cdot X_2)] \\
&= \sum_{\substack{(x_1,x_2) \in \mathbb{Z}_q^2 \\ x_1 + x_2 = s \mod q}} \omega(a \cdot x_1) \cdot \omega(a \cdot x_2) \cdot P[(X_1, X_2) = (x_1, x_2)] \\
&= \sum_{\substack{(x_1,x_2) \in \mathbb{Z}_q^2 \\ x_1 + x_2 = s \mod q}} \omega(a \cdot x_1) \cdot \omega(a \cdot (s - x_1)) \cdot P[(X_1, X_2) = (x_1, s - x_1)] \\
&= \sum_{x_1 \in \mathbb{Z}_q} \omega(a \cdot x_1) \cdot \omega(a \cdot (s - x_1)) \cdot P(X_1 = x_1) \\
&= \frac{\sum_{x_1 \in \mathbb{Z}_q} \omega(a \cdot x_1) \cdot \omega(a \cdot (s - x_1))}{q}.
\end{aligned}$$

While the expectation of $\omega(a \cdot X_1) \cdot \omega(a \cdot X_2)$ depends on $s$, it also depends on $a$ as well.

**Estimator of the Product.** Similarly, it is difficult to compute $\mathbb{E}_a$ because, the measurements are contaminated with the noise. Therefore, we use the same distinguisher to estimate it.

$$\bar{l} = \frac{\sum_{t=0}^{T-1} l^{(t)}}{T}.$$

Let $N_1$ and $N_2$ be random variables associated to the noise during the acquisition of the first and the second share. Let's break down this expression:

$$l^{(t)} = l_1^{(t)} \cdot l_2^{(t)} = (\omega(a \cdot X_1^{(t)}) + N_1^{(t)}) \cdot (\omega(a \cdot X_2^{(t)}) + N_2^{(t)}).$$

$$l^{(t)} = \omega(a \cdot X_1^{(t)}) \cdot \omega(a \cdot X_2^{(t)}) + \omega(a \cdot X_1^{(t)}) \cdot N_2^{(t)} + \omega(a \cdot X_2^{(t)}) \cdot N_1^{(t)} + N_1^{(t)} \cdot N_2^{(t)}.$$

Let $L_1, L_2, L_3$ and $L_4$ denote the following sums:

- $L_1 = \sum_{t=0}^{T-1} \omega(a \cdot X_1^{(t)}) \cdot \omega(a \cdot X_2^{(t)})$
- $L_2 = \sum_{t=0}^{T-1} \omega(a \cdot X_1^{(t)}) \cdot N_2^{(t)}$
- $L_3 = \sum_{t=0}^{T-1} \omega(a \cdot X_2^{(t)}) \cdot N_1^{(t)}$
- $L_4 = \sum_{t=0}^{T-1} N_1^{(t)} \cdot N_2^{(t)}$.

With those notations, we have: $\bar{l} = \frac{L_1}{T} + \frac{L_2}{T} + \frac{L_3}{T} + \frac{L_4}{T}$. Using the Law of Large Numbers, we are able to remove the non-deterministic terms that have been polluted by the noise. So, we are left only with the deterministic term that depends on the secret.

- $\lim_{t \to +\infty} \frac{L_1}{T} = \mathbb{E}_s[\omega(a \cdot X_1) \cdot \omega(a \cdot X_2)]$.
- $\lim_{t \to +\infty} \frac{L_2}{T} = \mathbb{E}[\omega(a \cdot X_1) \cdot N_2] = \mathbb{E}[\omega(a \cdot X_1)] \cdot \mathbb{E}[N_2] = 0$ (because $\omega(a \cdot X_1) \perp\!\!\!\perp N_2$).
- $\lim_{t \to +\infty} \frac{L_3}{T} = \mathbb{E}[\omega(a \cdot X_2) \cdot N_1] = \mathbb{E}[\omega(a \cdot X_2)] \cdot \mathbb{E}[N_1] = 0$ (because $\omega(a \cdot X_2) \perp\!\!\!\perp N_1$).
- $\lim_{t \to +\infty} \frac{L_4}{T} = \mathbb{E}[N_1 \cdot N_2] = \mathbb{E}[N_1] \cdot \mathbb{E}[N_2] = 0$ (because $N_1 \perp\!\!\!\perp N_2$).

Therefore, $\bar{l}$ converges towards $\mathbb{E}_s[\omega(a \cdot X_1) \cdot \omega(a \cdot X_2)]$ when we increase the number of traces in this case as well.
In practice, only 1 024 values of $a \in \mathbb{Z}_q$ can be exploited (see subsection 5.7.3 for the details). But we noticed that the multiplication by a scalar $a$ has an influence on the behaviour of the distributions we get. In other words, when we multiply each share with a scalar $a$, we obtain completely different distributions which open new opportunities.

To choose the best scalar $a$ to distinguish the secret values $s \in \{-2, -1, 0, 1, 2\}$, we consider, for each $a$, the perfect distribution associated to each $s$, *i.e.* evaluate the perfect theoretical mean value of $\mathbb{E}_a$. We have then five different values $\bar{s}_i = \mathbb{E}_{s_i}[\omega(a \cdot X_1) \cdot \omega(a \cdot X_2)]$, one for each $s_i \in \{-2, -1, 0, 1, 2\}$. We then evaluate the distances between each pair of $\bar{s_i}$ values, and evaluate the minimal distance among them. The best scalar value will be the one that maximizes the minimal distance between secrets. By searching on all the 1 024 possible values of $a$, we identified that the distinguisher that separates the most different values of the secret coefficients is $a = 3\,131$ with a minimal distance of 0.315 between secrets.

In fig. 5, we can distinguish 0, 1 and −2. However, there is a non negligible overlap between −1 and 2 that makes this distinguisher insufficient to mount an attack on the entire secret key in practice.

This significantly improves the quality of the distinguisher that can be directly derived from the literature and might allow to distinguish all the possible values of a coefficient of the secret in the case of low noise and high number of traces. Nevertheless, it is still not sufficient to mount an attack on the entire secret. For that, we need to go further and propose a new idea to enhance the distinction between the secret values.

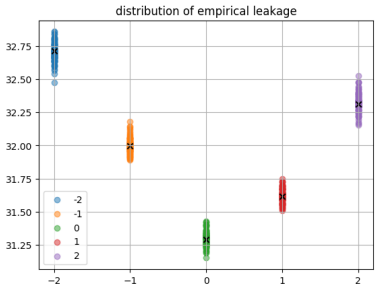



Fig. 5: $\mathbb{E}_s[\omega(a \cdot X_1) \cdot \omega(a \cdot X_2)]$ for $s \in \{-2, -1, 0, 1, 2\}$, $a = 3\,131$, $q = 3\,329$, $T = 100\,000$ and $\sigma = 2$

### 5.6 Combination of Different Scalar Values

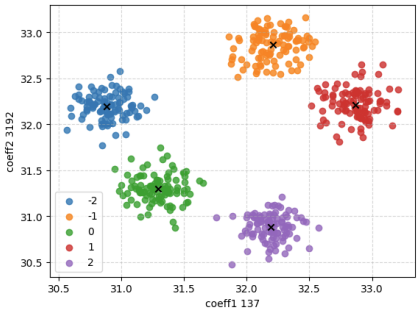
A major obstacle in recovering a secret value from a distribution $(\omega(a \cdot X_1), \omega(a \cdot X_2))$ is that, for all possible scalar value $a$, there are always several distributions that are so close from each other that they are undistinguishable with a reasonable number of traces.

The idea is then not to try to differentiate the secret value using a single scalar, but instead to combine the information gathered from several ones. So we need to find out how we could exploit the information extracted from different scalars, and how to choose those scalars.

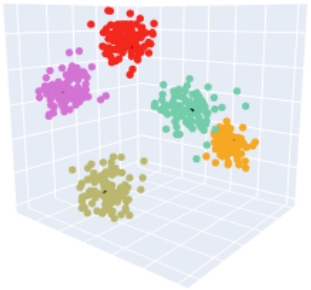
But how to proceed if we want to combine two different scalars together? Let's consider two scalars, $a_1$ and $a_2$. For each of them, we evaluate $\mathbb{E}_s[\omega(a \cdot X_1) \cdot \omega(a \cdot X_2)]$, *i.e.* we calculate $x = \mathbb{E}_s[\omega(a_1 \cdot X_1) \cdot \omega(a_1 \cdot X_2)]$ and $y = \mathbb{E}_s[\omega(a_2 \cdot X_1) \cdot \omega(a_2 \cdot X_2)]$, for each value of $s$. We evaluate, as we did for a single value $a$, the distances between those secrets. The difference here being that each secret value is represented with a couple $(x, y)$. In the same way as in subsection 5.4, we consider the couple of scalar values $(a_1, a_2)$ that maximises the minimal distance between those secrets. Such a couple of scalar values will be called a 2D-filter. The best 2D-filter that we found by exhaustive search is $(81, 234)$ with a minimal distance between secrets of approximately 0.9814.

To illustrate this idea of combination of scalar, we have represented, in figure 6a, the $\bar{s}_i$ values with a cross, and the mean values, obtained from random mask

generation and with a noise $\sigma = 2$ with a point. For each secret, the experiment has been repeated 100 times, as in section 5.4.




(a) Two dimensional representation with 2D-filter $(137, 3\,192)$



(b) Three dimensional representation with 3D-filter $(137, 1\,365, 3\,192)$

Fig. 6: Representation of distributions in 2 and 3 dimensions

We use the same methodology for three scalars called a 3D-filter. It is illustrated in figure 6b.

We have also pushed the idea for higher dimensions, namely 4, 5 and 6. As those dimensions were too high for an exhaustive search of each $n$D-filters, we have decided to reduce the space of coefficients to only those who seems to be more eligible candidates. Each time we add a new dimensional space, we observe a non negligible increase in the minimal distance between secrets. That increase in the minimal distance between secret suggests that adding new dimension can theoretically increase the efficiency of the search. For a total number of traces of 24 000 and assuming for simplicity that the number of traces is evenly distributed between the dimensions, the table 2 show how those dimensions behave.

As we can see on this table, a real gain in efficiency is made when jumping from the two dimension to the three dimensional space. For this number of traces, *i.e.* 24 000, the gain of adding more dimensions in more mitigated. Those higher dimensions would probably become more efficient than the lower ones with a higher number of traces and the noise level may also affect the result so deciding which dimension to use depends on the context of the attack. However, these are only general estimations, because in practice, it is possible to use a different number of traces per coefficient, making the efficiency comparison of the different dimensions more difficult.

**Remark:** One important note is that, when we perform the attack for a specific filter, let's say the 2D-filter (137, 3 192), we end up having five clusters forming the same constellation regardless of the device used: the cluster of the coefficients that are equal to $-1$ will always be on top , while the one of $-2$ will

Table 2: Wrong guess comparison between different $n$D-filters for a total number of 24 000 traces with a noise level $\sigma = 2$. Results are derived from a total of 10 000 experiments per secret (50 000 in total). The filters are respectively : 1D-filter*=(1) (current state of the art), 1D-filter=(3 131), 2D-filter=(81, 234), 3D-filter=(137, 322, 2 672), 4D-filter=(137, 306, 3 007, 3 192), 5D-filter=(137, 202, 234, 3 007, 3 192) and 6D-filter=(65, 137, 306, 3 007, 3 023, 3 192)

| Dimension | 1D* | 1D | 2D | 3D | 4D | 5D | 6D |
|---|---|---|---|---|---|---|---|
| $s = -2$ | 0 | 624 | 19 | 11 | 7 | 12 | 2 |
| $s = -1$ | 5 222 | 1 638 | 56 | 3 | 9 | 11 | 4 |
| $s = 0$ | 9 903 | 2 441 | 15 | 8 | 2 | 2 | 2 |
| $s = 1$ | 322 | 2 597 | 129 | 17 | 4 | 7 | 6 |
| $s = 2$ | 4 965 | 1 182 | 67 | 12 | 3 | 9 | 9 |
| **Total** | **20 412** | **8 482** | **286** | **51** | **25** | **41** | **23** |
| Traces per scalar | 24 k | 24 k | 12 k | 8 k | 6 k | 4,8 k | 4 k |
| **Min. dist.** | **0,002** | **0.315** | **0.981** | **1.497** | **1.874** | **2.059** | **2.281** |

always be at the bottom and so on (cf. figure 6a). This makes the attack feasible even without having access to a clone device.

### 5.7 Attack Scenario

After explaining the bias in the distribution, we explain how to exploit it to build the full attack. The attacker uses a divide-and-conquer approach. This means that they will retrieve each vector of the secret key independently. For each vector of the secret key, recovering each coefficient is performed in parallel by exploiting the same traces for each of them. Thus, without loss of generality, we can focus on retrieving a single coefficient $s$ of the secret key.

**5.7.1 Masked Implementation of *Bronchain et al.*** We illustrate our attack on the implementation of *Bronchain et al.* that was introduced in [6][6]. During the key generation, the secret key is transformed to the NTT form to perform polynomial multiplication. It is then stored in this form because it is more convenient since the decapsulation operation performs polynomial multiplications as well on the secret key. Therefore, during decapsulation, a mask of the secret key in the NTT form is generated and the computation is performed on each share of the secret. The pseudo-code of the masked implementation can be found in Alg. 1. The original unmasked version can be found in Alg. 4.

[6] The source code of the implementation can be found in https://github.com/uclcrypto/pqm4_masked

**Algorithm 1** KYBER.CPA.PKE.Dec$(s, c)$ (masked)

1: $\mathbf{u} = \text{Decompress}_q(\text{Decode}_{d_u}(c_1), d_u)$
2: $v = \text{Decompress}_q(\text{Decode}_{d_v}(c_2), d_v)$
3: $\mathsf{NTT}(\mathbf{s}) = \text{Decode}_{12}(s)$
4: $\mathbf{X}_1 \leftarrow \mathcal{U}(\mathcal{R}_q^k)$
5: $\mathbf{X}_2 = \mathsf{NTT}(\mathbf{s}) - \mathbf{X}_1$
6: $prod = \mathsf{INTT}\Big(\mathbf{X}_1 \circ \mathsf{NTT}(\mathbf{u})\Big) + \mathsf{INTT}\Big(\mathbf{X}_2 \circ \mathsf{NTT}(\mathbf{u})\Big)$ (*which is* $\mathbf{s}^\top\mathbf{u}$)
7: $m = \text{Encode}_1(\text{Compress}_q(v - prod, 1))$
8: **return** $m$

**5.7.2 Targeted Operation.** The attack takes place during the decapsulation operation. The decapsulation `crypto_kem_dec` calls the decryption function `masked_indcpa_dec`, which in turn calls the `masked_poly_invntt` which computes the inverse NTT on each share. Now, we explain how the attacker can get their hand on the distributions $(\omega(a \cdot x_1),\ \omega(a \cdot x_2))$ for a chosen $a$. The attack recovers each vector individually using a chosen-ciphertext request. To recover the coefficients of the vector $\mathbf{s}[i]$, the attacker sets the vector $\mathbf{u}$ as follows:

$$\forall l \in [\![1,\ k]\!], \mathbf{u}[l] = \begin{cases} a \in \mathcal{R}_q & \text{if } l = i \\ 0 & \text{if } l \neq i. \end{cases}$$

$\mathsf{NTT}(\mathbf{u}[i])$ is the evaluation of the constant polynomial $a$ on the roots of unity. Hence, $\mathsf{NTT}(\mathbf{u}[i]) = [a, \ldots, a]$.

$$\mathbf{X}_1 \circ \mathsf{NTT}(\mathbf{u}) = \begin{bmatrix} 0 \\ \ldots \\ \mathbf{X}_1[i] \circ \mathsf{NTT}(\mathbf{u}[i]) \\ \ldots \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ \ldots \\ a \cdot \mathbf{X}_1[i] \\ \ldots \\ 0 \end{bmatrix}.$$

$$\mathsf{INTT}\Big(\mathbf{X}_1 \circ \mathsf{NTT}(\mathbf{u})\Big) = \begin{bmatrix} 0 \\ \ldots \\ \mathsf{INTT}(\mathbf{X}_1[i] \circ \mathsf{NTT}(\mathbf{u}[i])) \\ \ldots \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ \ldots \\ \mathsf{INTT}(a \cdot \mathbf{X}_1[i]) \\ \ldots \\ 0 \end{bmatrix}.$$

The same is true for $\mathsf{INTT}\Big(\mathbf{X}_2 \circ \mathsf{NTT}(\mathbf{u})\Big)$. Furthermore, we know that:

$$(\mathbf{s}^\intercal\mathbf{u}) = \sum_{l=0}^{k-1} \mathsf{INTT}\Big(\mathsf{NTT}(\mathbf{s}[l]) \circ \mathsf{NTT}(\mathbf{u}[l])\Big) = \mathbf{s}[i] \circ \mathbf{u}[i] = a \cdot \mathbf{s}[i].$$

In the masked case, $\mathbf{s}[i] \circ \mathbf{u}[i]$ is computed as follows:

$$(\mathbf{s}^\intercal\mathbf{u}) = \mathbf{s}[i] \circ \mathbf{u}[i] = \mathsf{INTT}\Big(\mathbf{X}_1 \circ \mathsf{NTT}(\mathbf{u})\Big)[i] + \mathsf{INTT}\Big(\mathbf{X}_2 \circ \mathsf{NTT}(\mathbf{u})\Big)[i].$$

$$\begin{aligned}\mathbf{s}[i] \circ \mathbf{u}[i] &= \mathsf{INTT}\Big(\mathbf{X}_1[i] \circ \mathsf{NTT}(\mathbf{u}[i])\Big) + \mathsf{INTT}\Big(\mathbf{X}_2[i] \circ \mathsf{NTT}(\mathbf{u}[i])\Big) \\ &= \mathsf{INTT}\Big(a \cdot \mathbf{X}_1[i])\Big) + \mathsf{INTT}\Big(a \cdot \mathbf{X}_2[i]\Big) \\ &= a \cdot \mathsf{INTT}\Big(\mathbf{X}_1[i])\Big) + a \cdot \mathsf{INTT}\Big(\mathbf{X}_2[i]\Big).\end{aligned}$$

We set $\mathbf{x}_1 = \mathsf{INTT}\Big(\mathbf{X}_1\Big)$ and $\mathbf{x}_2 = \mathsf{INTT}\Big(\mathbf{X}_2\Big)$. Therefore,

$$\mathbf{s}[i] \circ \mathbf{u}[i] = a \cdot \mathbf{s}[i] = a \cdot \mathbf{x}_1[i] + a \cdot \mathbf{x}_2[i].$$

So, if we focus on a single coefficient $j \in [0, n-1]$,

$$a \cdot \mathbf{s}[i]_j = \underbrace{a \cdot \mathbf{x}_1[i]_j}_{\omega(a \cdot \mathbf{x}_1[i]_j)} + \underbrace{a \cdot \mathbf{x}_2[i]_j}_{\omega(a \cdot \mathbf{x}_2[i]_j)}.$$

Let's show that the distribution of $\mathbf{x}_1[i]_j$ is uniform. We recall that $\mathbf{X}_1$ is uniform over $\mathcal{R}_q^k$ which means that $\mathbf{X}_1[i]$ is uniform over $\mathcal{R}_q$.

Let $e \in \mathcal{R}_q$, $\mathsf{NTT}$ is bijective over $\mathcal{R}_q$ and $\mathcal{R}_q$ is a finite set, therefore:

$$P(\mathbf{x}_1[i] = e) = P[\mathsf{NTT}(\mathbf{x}_1[i]) = \mathsf{NTT}(e)] = P[\mathbf{X}_1[i] = \mathsf{NTT}(e)].$$

Hence, $\mathbf{x}_1[i]$ is uniform over $\mathcal{R}_q$. Thus, $\mathbf{x}_1[i]_j$ is uniform over $\mathbb{Z}_q$. The proof is analogous for $\mathbf{x}_2[i]_j$. Since operations on each coefficient occur sequentially, the attacker is able to recover the Hamming weight of each share individually.

**5.7.3 Ciphertext Choice.** In this section, we explain how the attacker sets the values of $\mathbf{u}[i] = a$ and what is the space of possible values. The objective is to control the values of $\mathbf{u}$ via the choice of the ciphertext $c$. The ciphertext is comprised of two parts: $c = c_1||c_2$. From $c_1$ and $c_2$, $\mathbf{u}$ and $v$ are extracted after *Decode* and *Decompress* operations. $v$ being irrelevant to the attack, $c_2$ can be set to a random value. On the other hand, the coefficients of the resulting polynomial $\mathbf{u}[i]_j$ take values in : $\mathcal{A} = \{\lfloor \frac{q}{2^{d_u}} x \rceil \mid x \in [\![0, 2^{d_u} - 1]\!]\}$. This leaves the attacker with 1024 choices of $a$ because in the considered case of `kyber768` version, we have $d_u = 10$.

If we want to use $a \in \mathcal{A}$, we choose $c_1$ such that $\mathbf{u} = [a, 0, 0]$. This is always possible because *decode* and *decompression* functions are surjective by design: we just have to encode and compress this $\mathbf{u} = [a, 0, 0]$ to get a valid $c_1 = \text{Encode}_{d_u}(\text{Compress}_q(\mathbf{u}, d_u))$. We set $c_2$ to a random value and build the ciphertext like that: $c = c_1||c_2$. We are able to recover the distributions $(\omega(a \cdot x_1), \omega(a \cdot x_2))$ for all the coefficients of $s[0]$. However, we need to repeat this operation with $\mathbf{u} = [0, a, 0]$ and $\mathbf{u} = [0, 0, a]$ to recover $\mathbf{s}[1]$ and $\mathbf{s}[2]$.

**5.7.4 Full Attack Scenario.** To perform the attack, if the attacker uses for example a 3D-filter $(a_1, a_2, a_3)$, he needs three ciphertexts:

$$c_{a_m} = \text{Encode}_{d_u}(\text{Compress}_q([a_m, 0, 0], d_u)).$$

For each ciphertext $c_{a_m}$, the attacker asks *Party 1* (cf. figure 1) to decapsulate the ciphertext $c_{a_m}$ $T$ times resulting in $T$ power traces. During the decryption step of the decapsulation algorithm, the attacker will measure the power after computing each share $(\mathbf{u} \cdot \mathbf{x}_1[0]_i, \mathbf{u} \cdot \mathbf{x}_2[0]_i)$ (cf. alg. 1) of the coefficient $s$ of the secret key. The attacker obtains leakages $(l_1^{(t)}, l_2^{(t)})_{0 \leq t \leq T-1}$ associated with the coefficient's shares, as defined in section 3. With this set of leakages $(l_1^{(t)}, l_2^{(t)})_{0 \leq t \leq T-1}$, the attacker computes the distinguisher $\bar{l}_{a_m}$ described in section 5.4 for each filter, then the secret key is chosen based on the rule introduced in 1. This process is repeated for each element of the secret vector.

## 6 Simulated Attack

### 6.1 Simulated Leakage Model

The attack was performed on simulation. The simulation of the power consumption or electromagnetic leakage is generated with Gaussian noise added to the $\omega$ obtained from the storage of the masking. To simplify the simulation, it is assumed that the noise level remains the same throughout the system. The simulations, which were performed using the Python programming language (v3.13), represent a typical unprotected implementation on a 16-bit processor.

### 6.2 Results of the Simulation

For the simulation, we used the 3D-filter $(137, 322, 2\,672)$. We target a success rate of 95% of recovering the entire secret key. Therefore, we need to achieve a success rate of 99.993% on a single coefficient because $0.99993^{768} \approx 0.95$. In order to estimate the success rate of the attack, we do the following experiments: we generate a secret key and test whether we are able to recover all its 768 coefficients with a determined number of traces. So, we capture $\omega(a \cdot x_1)$ and $\omega(a \cdot x_2)$ for a given number of traces, and check whether the guessed value from the 3D-filter is correct. The experiment is repeated 100 times, and the total number of traces $T$ is validated if the success rate is above 95%. The experiment is repeated for different noise level as well. The values considered for the noise level $\sigma$ are common in the literature [2] and still quite high. Tab. 3 summarises the results for those evaluations. The results obtained are experimentally determined approximations. As we can see, the attack is still feasible even in the case where the noise is superior to the signal which highlights the fact that our filter is resilient.

## 7 Conclusion and Perspectives

This article presented a novel second-order chosen ciphertext side-channel attack. We have shown that the distribution of the shares for a given secret is also affected by the coefficient $a$ handled, which can be set by the adversary via

Table 3: Number of traces required for different noise levels to recover the entire key with more than 95% probability

| Noise level sigma | 0.5 | 1 | 2 |
|---|---|---|---|
| SNR | 10.7 | 2.67 | 0.67 |
| Number of traces | 42 000 | 54 000 | 75 000 |

well chosen ciphertexts. This phenomenon is used to combine the information leaked from different distributions to recover the secret. This allows a significant improvement in the efficiency of a straightforward second-order attack, as shown in table 2. The attack has been instantiated in simulation against a masked implementation of the Crystals-Kyber key encapsulation mechanism, standardized afterwards as ML-KEM. Despite the fact that our attack requires a significant amount of traces, it does not need access to a clone device, in contrast with machine learning based or template attacks. Our first results show that a similar approach can be used against second-order masking using a third-order attack, with a significant increase in terms of the required number of traces. Therefore, the case of higher-masking order might be interesting to study as well. Most importantly, it should be investigated whether this attack extends to other cryptographic schemes. Finally, ciphertexts used in the attack have a specific shape, they are sparse and contain a constant value, they don't follow the uniform distribution as honest ciphertexts that are instances of LWE. Therefore, the use of a sanity check as introduced in [30] could be sufficient to discard this attack.

**Acknowledgement.** The authors would like to acknowledge the EU funding received under the ALLEGRO project (grant No. 101092766) and the financial support of the French National Investment Bank (BPI) for the X7PQC project (contract DOS0209793 – DOS0209794). This work was supported in part by French projects ANR-11-LABX-0020-01 "Centre Henri Lebesgue" and ANR-18-EURE-0004 "Cyberschool". We also would like to thank the reviewers for their relevant feedbacks that improved the quality of the paper.

# A Crystals-Kyber description

## A.1 Compression/Decompression Functions

The purpose of the compression and decompression functions is to reduce the size of the keys and the ciphertext. They allow to discard some bits of the ciphertext while introducing a ridiculously low decryption failure probability.

$$\begin{aligned} Compress_q\colon \mathbb{Z}_q \times \mathbb{N}^* &\to [\![0, 2^d - 1]\!] \\ (x, d) &\mapsto \lfloor \frac{2^d}{q} x \rceil \ mod^+ \ 2^d. \end{aligned}$$

$$\begin{aligned} Decompress_q\colon [\![0, 2^d - 1]\!] \times \mathbb{N}^* &\to \mathbb{Z}_q \\ (x, d) &\mapsto \lfloor \frac{q}{2^d} x \rceil. \end{aligned}$$

- For a polynomial $p \in \mathcal{R}_q$, $Compress_q$ corresponds to evaluating the function over each polynomial.
- The operation is extended to vectors of polynomials $\mathbf{u} \in \mathcal{R}_q^k$ by applying it component-wise.

## A.2 Presentation

Crystals-Kyber is a lattice-based post-quantum key exchange scheme whose security is based on the hardness of the MLWE problem over lattices[7]. Crystals-Kyber involves computations over the polynomial ring $\mathcal{R}_q$. The secret key is comprised of polynomials $\mathbf{s} = (s[0], \ldots, s[k-1]) \in \mathcal{R}_q^k$. Each coefficient $s[i]_j$ of the polynomial $s[i] = \sum_{j=0}^{N-1} s[i]_j X^j$ is sampled according to the CBD over $\mathcal{S} = [\![-\eta_1; \eta_1]\!]$. The size $k$ of the vector and the size $\eta_1$ of the interval depend on the security level (cf. table 4).

Table 4: Parameter sets for NIST security levels

| | $N$ | $q$ | $k$ | $\eta_1$ | $d_u$ | $d_v$ |
|---|---|---|---|---|---|---|
| Kyber512 | 256 | 3 329 | 2 | 3 | 10 | 4 |
| Kyber768 | 256 | 3 329 | 3 | 2 | 10 | 4 |
| Kyber1024 | 256 | 3 329 | 4 | 2 | 11 | 5 |

[7] See https://pq-crystals.org/kyber/data/kyber-specification-round3-20210131.pdf for the full specification

### A.3 Crystals-Kyber, CPA PKE Version

We first introduce the CPA public key encryption version. It is composed of three algorithms: A Keygen, an Encryption and a Decryption algorithm. The Keygen algorithm (see algorithm 2) generates a public key $pk$ and a private key $sk$.

**Algorithm 2** KYBER.CPA.PKE.Keygen() (simplified)

$(\rho, \theta) \leftarrow \mathcal{U}(\{0,1\}^{256})$
$\mathbf{A} \leftarrow \mathcal{U}(\mathcal{R}_q^{k \times k}; \rho)$
$\mathbf{s}, \mathbf{e} \leftarrow \beta_{\eta_1}(\mathcal{R}_q^{k \times 1}; \theta)$
$\mathbf{t} \leftarrow \text{Encode}_{12}(\mathbf{As} + \mathbf{e})$
$s \leftarrow \text{Encode}_{12}(\mathbf{s})$
**return** $(pk = (t, \rho), sk = s)$

The Encryption algorithm encrypts the message $m$ using the public key $pk$ and a seed $r$, and outputs a ciphertext $c = (c_1, c_2)$ (see algorithm 3).

**Algorithm 3** KYBER.CPA.PKE.Enc$(pk = (t, \rho), m, r)$ (simplified)

$\mathbf{t} \leftarrow \text{Decode}_{12}(t)$
$\mathbf{A} \leftarrow \mathcal{U}(\mathcal{R}_q^{k \times k}; \rho)$
$\mathbf{r} \leftarrow \beta_{\eta_1}(\mathcal{R}_q^{k \times 1}; r)$
$\mathbf{e}_1 \leftarrow \beta_{\eta_2}(\mathcal{R}_q^{k \times 1}; r);\ e_2 \leftarrow \beta_{\eta_2}(\mathcal{R}_q^{1 \times 1}; r)$
$\mathbf{u} = \mathbf{A}^\intercal \cdot \mathbf{r} + \mathbf{e}_1$
$v = \mathbf{t}^\intercal \cdot \mathbf{r} + e_2 + \text{Decompress}_q(m, 1)$
$c_1 = \text{Encode}_{d_u}(\text{Compress}_q(u, d_u))$
$c_2 = \text{Encode}_{d_v}(\text{Compress}_q(v, d_v))$
**return** $c = (c_1, c_2)$

Finally, the Decryption algorithm takes as input the ciphertext $\mathbf{c}$ and the secret key $s$, and recovers the message $m$ (see algorithm 4).

**Algorithm 4** KYBER.CPA.PKE.Dec$(s, c)$ (simplified)

$\mathbf{u} = \text{Decompress}_q(\text{Decode}_{d_u}(c_1), d_u)$
$v = \text{Decompress}_q(\text{Decode}_{d_v}(c_2), d_v)$
$\mathsf{NTT}(\mathbf{s}) = \text{Decode}_{12}(s)$
$m = \text{Encode}_1(\text{Compress}_q(v - \mathsf{INTT}(\mathsf{NTT}(\mathbf{s}) \circ \mathsf{NTT}(\mathbf{u}), 1))$
**return** $m$

### A.4 Crystals-Kyber, CCA KEM Version

A key exchange mechanism is a protocol that allows to securely exchange a secret key over an insecure communication channel. It is composed of three different parts : A Keygen, an Encapsulation and a Decapsulation algorithm. As for the CPA version, the Keygen generates a couple of keys $(pk, sk)$ (see algorithm 5).

**Algorithm 5** KYBER.CCA.KEM.Keygen() (simplified)

$z \leftarrow \mathcal{U}(\{0,1\}^{256})$
$(pk, \mathbf{s}) = \mathsf{KYBER.CPA.PKE.Keygen}()$
$sk = (\mathbf{s}, pk, \mathcal{H}(pk), z)$
**return** $(pk, sk)$

The Encapsulation algorithm generates a shared key $K$ along with a ciphertext $c$ which embeds the shared key information.

**Algorithm 6** KYBER.CCA.KEM.Encaps($pk$) (simplified)

$m \leftarrow \mathcal{U}(\{0,1\}^{256})$
$m = \mathcal{H}(m)$
$(\tilde{K}, r) = \mathcal{G}(m, \mathcal{H}(pk))$
$c = \mathsf{KYBER.CPAPKE.Enc}(pk, m, r)$
$K = \mathsf{KDF}(\tilde{K}, \mathcal{H}(c))$
**return** $(c, K)$

Finally, the Decapsulation algorithm recovers the corresponding shared key $K$ or outputs a random value, depending of the ciphertext and secret key correctness.

**Algorithm 7** KYBER.CCA.KEM.Decaps($sk = (s, pk, \mathcal{H}(pk), z), c$) (simplified)

$m' = \mathsf{KYBER.CPA.PKE.Dec}(s, c)$
$(\tilde{K}', r') = \mathcal{G}(m', \mathcal{H}(pk))$
$c' = \mathsf{KYBER.CPA.PKE.Enc}(pk, m', r')$
**if** $c = c'$ **then**
    **return** $K = \mathsf{KDF}(\tilde{K}', \mathcal{H}(c))$
**else**
    **return** $K = \mathsf{KDF}(z, \mathcal{H}(c))$
**end if**

**Disclosure of Interests.** The authors have no competing interests to declare that are relevant to the content of this article.

## References


1. Amiet, D., Curiger, A., Leuenberger, L., Zbinden, P.: Defeating newhope with a single trace. In: Post-Quantum Cryptography: 11th International Conference, PQCrypto 2020, Paris, France, April 15–17, 2020, Proceedings 11. pp. 189–205. Springer (2020)
2. Béguinot, J., Cheng, W., Guilley, S., Rioul, O.: Be my guess: Guessing entropy vs. success rate for evaluating side-channel attacks of secure chips. In: 2022 25th Euromicro Conference on Digital System Design (DSD). pp. 496–503. IEEE (2022)
3. Biham, E., Shamir, A.: Differential fault analysis of secret key cryptosystems. In: Advances in Cryptology—CRYPTO'97: 17th Annual International Cryptology Conference Santa Barbara, California, USA August 17–21, 1997 Proceedings 17. pp. 513–525. Springer (1997)
4. Bos, J., et al.: CRYSTALS-Kyber: a CCA-secure module-lattice-based KEM. In: 2018 IEEE European Symposium on Security and Privacy (EuroS&P). pp. 353–367. IEEE (2018)
5. Bos, J.W., Gourjon, M.O., Renes, J., Schneider, T., Vredendaal, C.v.: Masking kyber: First-and higher-order implementations. IACR Transactions on Cryptographic Hardware and Embedded Systems **2021**(4), 173–214 (2021)
6. Bronchain, O., Cassiers, G.: Bitslicing arithmetic/boolean masking conversions for fun and profit: with application to lattice-based KEMs. IACR Transactions on Cryptographic Hardware and Embedded Systems pp. 553–588 (2022)
7. Brumley, D., Boneh, D.: Remote timing attacks are practical. Computer Networks **48**(5), 701–716 (2005)
8. Canteaut, A., Lauradoux, C., Seznec, A.: Understanding cache attacks. Ph.D. thesis, INRIA (2006)
9. Coron, J.S.: Resistance against differential power analysis for elliptic curve cryptosystems. In: Cryptographic Hardware and Embedded Systems: First InternationalWorkshop, CHES'99 Worcester, MA, USA, August 12–13, 1999 Proceedings 1. pp. 292–302. Springer (1999)
10. Coron, J.S., Großschädl, J., Tibouchi, M., Vadnala, P.K.: Conversion from arithmetic to boolean masking with logarithmic complexity. In: Fast Software Encryption: 22nd International Workshop, FSE 2015, Istanbul, Turkey, March 8-11, 2015, Revised Selected Papers 22. pp. 130–149. Springer (2015)
11. D'Anvers, J.P., Tiepelt, M., Vercauteren, F., Verbauwhede, I.: Timing attacks on error correcting codes in post-quantum schemes. In: Proceedings of ACM Workshop on Theory of Implementation Security Workshop. pp. 2–9 (2019)
12. Duc, A., Dziembowski, S., Faust, S.: Unifying leakage models: from probing attacks to noisy leakage. In: Advances in Cryptology–EUROCRYPT 2014: 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings 33. pp. 423–440. Springer (2014)
13. Fujisaki, E., Okamoto, T.: Secure integration of asymmetric and symmetric encryption schemes. In: Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology. p. 537–554. CRYPTO '99, Springer-Verlag, Berlin, Heidelberg (1999)
14. Goubin, L., Patarin, J.: DES and differential power analysis the "duplication" method. In: Koç, Ç.K., Paar, C. (eds.) Cryptographic Hardware and Embedded Systems. pp. 158–172. Springer Berlin Heidelberg, Berlin, Heidelberg (1999)

15. Guo, Q., Nabokov, D., Nilsson, A., Johansson, T.: SCA-LDPC: A code-based framework for key-recovery side-channel attacks on post-quantum encryption schemes. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 203–236. Springer (2023)
16. Heinz, D., Kannwischer, M.J., Land, G., Pöppelmann, T., Schwabe, P., Sprenkels, A.: First-order masked kyber on ARM cortex-M4. Cryptology ePrint Archive (2022)
17. Jendral, S., Ngo, K., Wang, R., Dubrova, E.: A single-trace message recovery attack on a masked and shuffled implementation of CRYSTALS-kyber. Cryptology ePrint Archive (2023)
18. Kannwischer, M.J., Krausz, M., Petri, R., Yang, S.Y.: pqm4: Benchmarking NIST additional post-quantum signature schemes on microcontrollers. Cryptology ePrint Archive (2024)
19. Kannwischer, M.J., Pessl, P., Primas, R.: Single-trace attacks on keccak. Cryptology ePrint Archive (2020)
20. Kocher, P., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M. (ed.) Advances in Cryptology — CRYPTO' 99. pp. 388–397. Springer Berlin Heidelberg, Berlin, Heidelberg (1999)
21. Lee, J., Han, D.G.: Security analysis on dummy based side-channel countermeasures—case study: AES with dummy and shuffling. Applied Soft Computing **93**, 106352 (2020). https://doi.org/https://doi.org/10.1016/j.asoc.2020.106352, https://www.sciencedirect.com/science/article/pii/S1568494620302921
22. Longo, J., De Mulder, E., Page, D., Tunstall, M.: SoC it to EM: electromagnetic side-channel attacks on a complex system-on-chip. In: Cryptographic Hardware and Embedded Systems–CHES 2015: 17th International Workshop, Saint-Malo, France, September 13-16, 2015, Proceedings 17. pp. 620–640. Springer (2015)
23. Mangard, S., Oswald, E., Popp, T.: Power analysis attacks: Revealing the secrets of smart cards, vol. 31. Springer Science & Business Media (2008)
24. Pay, D., Standaert, F.X.: Side-channel analysis of arithmetic encodings for post-quantum cryptography: Cautionary notes with application to kyber. In: Vaudenay, S., Petit, C. (eds.) Progress in Cryptology - AFRICACRYPT 2024. pp. 260–281. Springer Nature Switzerland, Cham (2024)
25. Peikert, C.: Public-key cryptosystems from the worst-case shortest vector problem. In: Proceedings of the forty-first annual ACM symposium on Theory of computing. pp. 333–342 (2009)
26. Pessl, P., Primas, R.: More practical single-trace attacks on the number theoretic transform. In: Progress in Cryptology–LATINCRYPT 2019: 6th International Conference on Cryptology and Information Security in Latin America, Santiago de Chile, Chile, October 2–4, 2019, Proceedings 6. pp. 130–149. Springer (2019)
27. Primas, R., Pessl, P., Mangard, S.: Single-trace side-channel attacks on masked lattice-based encryption. In: Cryptographic Hardware and Embedded Systems–CHES 2017: 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings. pp. 513–533. Springer (2017)
28. Prouff, E., Rivain, M., Bevan, R.: Statistical analysis of second order differential power analysis. IEEE Transactions on computers **58**(6), 799–811 (2009)
29. 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 Transactions on Cryptographic Hardware and Embedded Systems (2023)

30. 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 Transactions on Embedded Computing Systems **23**(2), 1–54 (2024)
31. Ravi, P., Roy, S.S., Chattopadhyay, A., Bhasin, S.: Generic side-channel attacks on CCA-secure lattice-based PKE and KEMs. IACR transactions on cryptographic hardware and embedded systems pp. 307–335 (2020)
32. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. J. ACM **56**(6) (sep 2009). https://doi.org/10.1145/1568318.1568324, https://doi.org/10.1145/1568318.1568324
33. Shi, M., Wang, Z., Peng, T., Li, F.: Message recovery attack of kyber based on information leakage in decoding operation. In: International Conference on Security and Privacy in Communication Systems. pp. 630–647. Springer (2022)
34. Shor, P.W.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM Journal on Computing **26**(5), 1484–1509 (1997). https://doi.org/10.1137/S0097539795293172, https://doi.org/10.1137/S0097539795293172
35. Sim, B.Y., et al.: Single-trace attacks on message encoding in lattice-based KEMs. IEEE Access **8**, 183175–183191 (2020)
36. Van Beirendonck, M., D'Anvers, J.P., Verbauwhede, I.: Analysis and comparison of table-based arithmetic to boolean masking. IACR Transactions on Cryptographic Hardware and Embedded Systems pp. 275–297 (2021)
37. Veyrat-Charvillon, N., Medwed, M., Kerckhof, S., Standaert, F.X.: Shuffling against side-channel attacks: A comprehensive study with cautionary note. In: Advances in Cryptology–ASIACRYPT 2012: 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings 18. pp. 740–757. Springer (2012)
38. Wang, J., Cao, W., Chen, H., Li, H.: Practical side-channel attack on masked message encoding in latticed-based KEM. Cryptology ePrint Archive (2022)
39. Xu, Z., Pemberton, O., Roy, S.S., Oswald, D., Yao, W., Zheng, Z.: Magnifying side-channel leakage of lattice-based cryptosystems with chosen ciphertexts: The case study of kyber. IEEE Transactions on Computers **71**(9), 2163–2176 (2021)
40. Yesina, M., Ostrianska, Y., Gorbenko, I.: Status report on the third round of the NIST post-quantum cryptography standardization process. Radiotekhnika pp. 75–86 (09 2022). https://doi.org/10.30837/rt.2022.3.210.05