Handling Noisy Plaintext Checking Oracles with SPiRiT

Application to Kyber

Paco Poilbout*[1], Thomas Roche[2], and Laurent Imbert[3]

[1]Independent Researcher, Mons, Belgium , pacopoilbout@gmail.com
[2]NinjaLab, Montpellier, France , thomas.roche@ninjalab.io
[3]CNRS, Université de Montpellier, LIRMM, Montpellier, France ,
laurent.imbert@lirmm.fr

Abstract. Post-Quantum key encapsulation mechanisms based on the re-encryption framework of Fujisaki and Okamoto have proved very sensitive to Plaintext Checking Oracle (PCO) attacks. The first theoretic works on PCO attacks were rapidly followed by practical attacks on real implementations, notably on NIST standardized ML-KEM. The actual realization of a PCO relies on side-channel leakages that are inherently noisy ; even more so if the implementation embeds side-channel countermeasures.

In this paper we tackle the often overlooked complications caused by highly noisy PCOs. We demonstrate that the impact of wrong oracle answers can be very efficiently reduced with the use of the so-called *Sequential Probability Ratio Test* (SPRT). This test can be seen as an elegant and natural early abort strategy on top of the commonly used approaches based on majority-voting or the likelyhood ratio test. As far as we know, this is the first use of SPRT in the context of side-channel attacks. We show that it allows to divide by a factor up to 3 the attack complexity compared to the traditional approaches. By establishing new comparisons with recently published noisy PCO attacks we emphasize that SPRT should be considered as the novel baseline for all future works in this line of research.

Keywords: ML-KEM, Side-channel attacks, Plaintext checking oracle, Majority voting, SPRT

*This work was done during Paco Poilbout Master Research Internship at NinjaLab and LIRMM

Contents

1 Introduction

After a thorough evaluation process, NIST standardized five algorithms for digital signature and encryption that are believed to remain secure even in the presence of a quantum adversary. In particular, NIST endorsed ML-KEM [MLK24], a Module-Lattice-Based Key-Encapsulation Mechanism (KEM for short) based on CRYSTALS-Kyber [SAB$^+$22].

Building solid protection against side-channel attacks for these novel class of algorithms remains a challenge of utmost importance. In particular, numerous side-channel attacks against several implementations of Kyber have already been proposed. In this paper, we are interested in a family of attacks that take advantage of the re-encryption process present in most PQC KEMs through the FO-transform, by creating a so-called *Plaintext-Checking Oracle* (PCO for short) [QCD19, RRCB20, HV20]. The practical realization of such an oracle exploits side-channel leakages in a way that has proven to be extremely advantageous for the attacker compared to traditional side-channel attacks [ABH$^+$22].

Over the past years, several improvements have been suggested to drastically reduce the complexity of PCO attacks (in terms of number of oracle queries, i.e. number of side-channel traces) ; notably the formalization of efficient binary search trees due to Qin *et al.* [QCZ$^+$21], and the multi-plaintext (or multi-valued) oracle paradigm [XIU$^+$21, RRD$^+$23, TUX$^+$23].

In this paper, we study a largely overlooked issue of side-channel based PCO attacks: depending on the side-channel leakage strength, the practical realization of a PCO might provide a wrong answer with non-negligible probability. While most of the literature assumes (or practically builds) PCOs with negligible probability of error, it seems reasonable to ask what happens when the attacker is unable to do so. For instance, due to the use of dedicated side-channel countermeasures which reduce the amount of information that an attacker is able to capture through side-channel (see *e.g.* a formal study of masking in this context [ABH$^+$22]).

The ultimate goal of a PCO attack is to recover all the digits of the private key. For each digit, the attacker queries an oracle whose response allows to walk down one step in a decision tree until the corresponding digit is fully recovered. The same process is repeated for each digit. Since the side-channel realization of this oracle might provide a wrong answer, a natural and widely used solution is to rely on majority-voting (see, *e.g.* [RRCB20]), i.e. each step in descending the decision tree is taken based on the most frequent response obtained after repeated independent queries addressed to the oracle.

Over the past years, majority-voting (or its generalization to the likelihood ratio test) has thus become the method of choice for addressing the issue of noisy PCOs, and also the baseline for comparing advanced techniques for handling noisy PCOs [SCZ$^+$23, GNNJ23, LCS$^+$25]. Surprisingly, in all these works, majority voting (or likelihood ratio test) is always exploited in its most naive version. In particular, although natural, early abort strategies are never considered.

In this work, we propose to set a new baseline provided by the so-called *Sequential*

Probability Ratio Test (SPRT for short). In its simplest expression, SPRT can be seen as an elegant early-abort strategy on top of majority-voting, de facto reducing the number of oracle queries for free. To our knowledge, this is the first time that SPRT is proposed in the side channel context. This could be explained by the *online* nature of the early-abort strategy (the attacker must analyze a side-channel trace before the next side-channel acquisition), which could be an issue in some contexts. However, PCO attacks are inherently online side-channel attacks (at least for Kyber) and therefore the use of SPRT does not add new constraints for the attacker.

We thus reconsider state-of-the-art algorithms for dealing with noisy PCOs in Kyber and show that the usage of SPRT as the proper baseline reduces their complexity gains to almost nothing at best, and even incur significant negative returns in the worst cases.

As with the literature on this subject, while all our contributions are presented in the context of Kyber, they are completely generic and would also apply to any KEM based on the re-encryption framework.

Related Works:

In [UXT$^+$22], the authors tackled a Kyber implementation involving side-channel countermeasures with a deep-learning trained PCO which revealed a relatively high probability of error. In order to reduce this probability of error, their solution refined the majority-voting test into a more general likelihood ratio test by exploiting all the deep-learning network output information.

To our knowledge, there are only three recent papers that propose sophisticated solutions to the issue of noisy oracles, namely Shen *et al.* [SCZ$^+$23] and Guo *et al.* [GNNJ23] in 2023, Li *et al.* [LCS$^+$25] in 2025.

We first observe that the SCA-LDPC attack[GNNJ23] *on Kyber* exploits the Full Domain (FD) oracle model (this is also the case for the more recent [DG25]). However, such an oracle model represents a much more powerful attacker than the PCO model and therefore goes beyond the scope of this paper. Although the use of SPRT in this model may also have an impact, we will leave this investigation for future work. We shall present and analyze [LCS$^+$25] and [SCZ$^+$23] in details in Sections 6 and 7 respectively.

Contributions and Paper Organization:

In Section 2 we present the notations used throughout this paper, and all necessary information about our target key encapsulation scheme ML-KEM. We recall the principles of plaintext checking (and multi-plaintext checking) oracle attacks in Section 3. In Section 4, we then explain how these idealized oracles are implemented in real life using side-channel analysis and we introduce the concept of *noisy* plaintext checking oracles. Our contributions follow:

- In Section 4.4 we show that side-channel based noisy oracles should not be handled with the traditional likelihood ratio test but with the SPRT, an elegant early-abort strategy built on top of the likelihood ratio test. Our simulations demonstrate that the use of SPRT results in dividing the number of oracle queries by a factor ranging from 2 to 3, at no extra cost.

- In Section 5, we propose a simple, yet effective way to improve the multi-plaintext checking oracle attack from Tanaka *et al.* [TUX+23]. We show that their approach can be combined with SPRT when the oracle is noisy.

- In Section 6, we study the recent attack from [LCS+25]. This algorithm is well adapted to muti-plaintext checking oracles and has been specifically developed to handle noisy oracles. We demonstrate that the use of SPRT as the baseline instead of majority voting significantly reduces or voids the complexity gains claimed in [LCS+25]: from the claimed -43.9% gain we end up with a modest -12% gain in the best case (very low level of noise). In the worst case (*i.e.* higher level of noise) [LCS+25] solution incurs a drastic penalty of $+63.4\%$ compared to the new baseline (see Tables 4 and 5).

- Finally, in Section 7, we study another recent algorithm [SCZ+23] proposed to handle noisy oracles, in the specific case of a single plaintext checking oracle. We show that the use of SPRT as the baseline considerably reduces the complexity gain claimed in [SCZ+23]: from -63.7% down to -18.8% gain in the best case (very low level of noise) to a $+18.8\%$ penalty in the worst cases (higher level of noise) (see Tables 8 and 9). Yet, we show that combining SPRT together with the approach from [SCZ+23] is advantageous: the resulting algorithm is much simpler and results in the best attack complexity.

2 Notations And ML-KEM

In this section, we introduce the notations used in this paper. We recall the set of algorithms from the CRYSTAL-Kyber suite, in particular the CPA Public Key Encryption scheme and CCA Key Encapsulation Mechanism. We also give the NIST standardized (under ML-KEM) security parameters. For more details about these algorithms, we refer to their specifications [MLK24].

2.1 Notations

Let $R = \mathbb{Z}[X]/(X^n + 1), R_q = \mathbb{Z}_q[X]/(X^n + 1)$ with $q = 3329$ a prime number and $n = 2^8$. For a distribution ψ, $x \overset{\$}{\leftarrow} \psi$ denotes a random sample from ψ, $x \overset{\$}{\leftarrow} \psi^n$ denotes n random, independent samples from ψ and $x \overset{\sigma}{\leftarrow} \psi$ denotes the deterministic sample of the coin σ

from ψ. The same notations hold when is ψ a set, in this case the random samples are drawn uniformly from the set ψ.

Moreover, $U(q)$ designates the uniform distribution over the set $\mathbb{Z}_q$ and β_η the centered binomial distribution implemented as $\sum_{i=1}^{\eta}(a_i - b_i)$, with $a_i, b_i \overset{\$}{\leftarrow} \{0,1\}$ for all $i = 1, \ldots, \eta$.

A vector of n bits or a polynomial in R_q will be both denoted with small caps, *e.g.* x. And the i^{th} coefficient of x is $x[i-1]$ (*i.e.* the indexes start at 0). A vector or set of polynomials will be denoted with bold font (*e.g.* $\mathbf{x}$), whereas a matrix of polynomials with bold capital font (*e.g.* $\mathbf{A}$).

Finally, $\lceil s \rfloor$ denotes the closest integer from the value s (and when s is at equal distance between two integers, the greater of the two is chosen), $|s|$ the absolute value of s and $\Pr(A)$ the probability of event A.

2.2 CRYSTAL-Kyber PKE Scheme

With the above notations, we recall hereafter the CRYSTAL-Kyber PKE scheme, with its three algorithms: key generation, encryption and decryption. Security parameters are recalled in Table 1.

Algorithm 1 Kyber.CPAPKE.KeyGen():

```
Output: $(\mathbf{pk}, \mathbf{sk})$ with $\mathbf{pk} \in R_q^{k \times k} \times R_q^k$ and $\mathbf{sk} \in R_q^k$
$(\rho, \sigma) \overset{\$}{\leftarrow} \{0,1\}^{256} \times \{0,1\}^{256}$
$\mathbf{A} \overset{\rho}{\leftarrow} U(q)^{k \times k}$
$(\mathbf{s}, \mathbf{e}) \overset{\sigma}{\leftarrow} \beta_{\eta_1}^k \times \beta_{\eta_1}^k$
$\mathbf{t} \leftarrow \mathbf{A}\mathbf{s} + \mathbf{e}$
$\mathbf{pk} \leftarrow (\mathbf{A}, \mathbf{t})$
$\mathbf{sk} \leftarrow \mathbf{s}$
return $(\mathbf{pk}, \mathbf{sk})$
```

Algorithm 2 Kyber.CPAPKE.Dec($\mathbf{sk}, \mathbf{c}$):

```
Input: Secret key $\mathbf{sk} \in R_q^k$ and Ciphertext $\mathbf{c} := (\mathbf{c_1}, c_2)$
Output: Message $m$
$\mathbf{u}' := \mathsf{Decompress}_q(\mathbf{c_1}, d_u)$
$v' := \mathsf{Decompress}_q(c_2, d_v)$
$m' \leftarrow v' - \mathbf{sk}^T \mathbf{u}'$
$m := \mathsf{Compress}_q(m', 1)$
return $m$
```

Algorithm 3 Kyber.CPAPKE.Enc($\mathbf{pk}, m, r$):

```
Input: Public key $\mathbf{pk} := (\mathbf{A}, \mathbf{t})$ with $A \in R_q^{k \times k}$ and $t \in R_q^k$
Input: Message $m \in \{0,1\}^{256}$
Input: Random coin $r \in \{0,1\}^{256}$
Output: Ciphertext $\mathbf{c} := (\mathbf{u}, v)$
$(\mathbf{r}, \mathbf{e_1}, e_2) \xleftarrow{r} \beta_{\eta_1}^k \times \beta_{\eta_2}^k \times \beta_{\eta_2}$
$\mathbf{u'} \leftarrow \mathbf{A}^T\mathbf{r} + \mathbf{e_1}$
$v' \leftarrow \mathbf{t}^T\mathbf{r} + e_2 + \lceil \frac{q}{2} \rfloor m$
$\mathbf{u} := \mathsf{Compress}_q(\mathbf{u'}, d_u)$
$v := \mathsf{Compress}_q(v', d_v)$
return $(\mathbf{u}, v)$
```

The $\mathsf{Compress}_q$ and $\mathsf{Decompress}_q$ functions are defined as:

$$\mathsf{Compress}_q(x, d) = \left\lceil (2^d/q) \cdot x \right\rfloor \bmod 2^d$$

$$\mathsf{Decompress}_q(x, d) = \left\lceil (q/2^d) \cdot x \right\rfloor$$

2.3 ML-KEM

From the PKE scheme of Kyber, a key encapsulation mechanism is constructed (standardized under the name ML-KEM [MLK24]. It is based on a variant of the FO (Fujisaki-Okamoto) transform [FO99] to ensure IND-CCA2 security.

The Kyber.CCAKEM.Decaps($\mathbf{sk'}, \mathbf{c}$) algorithm is the target of the plaintext-checking oracle attacks (that can be seen as a subclass of the *key-mismatch* attacks) studied in this paper. In order to avoid chosen ciphertexts attacks, the FO-transform ensures that the Kyber.CCAKEM.Decaps input $\mathbf{c}$ is correct, otherwise the decapsulated session key K is not output by the algorithm. To this end, after the decryption of the ciphertext $\mathbf{c}$ into the message $\hat{m}$, the message is re-encrypted into $\hat{\mathbf{c}}$ and the equality of $\mathbf{c}$ and $\hat{\mathbf{c}}$ is verified.

Algorithm 4 Kyber.CCAKEM.KeyGen():

```
Output: Public key $\mathbf{pk}$ and Secret key $\mathbf{sk'}$
$z \xleftarrow{\$} \{0,1\}^{256}$
$(\mathbf{pk}, \mathbf{sk}) \leftarrow$Kyber.CPAKEM.Keygen()
$\mathbf{sk'} := (\mathbf{sk}||\mathbf{pk}||\mathbf{H}(\mathbf{pk})||z)$
return $(\mathbf{pk}, \mathbf{sk'})$
```

In the above algorithms, J, G and H are hash functions based on the Keccak permutation.

Algorithm 5 Kyber.CCAKEM.Encaps($\mathbf{pk}$):

```
Input: Public key $\mathbf{pk} := (\mathbf{A}, \mathbf{t})$
$m' \xleftarrow{\$} \{0,1\}^{256}$
$(K, r) \leftarrow \mathbf{G}(m||\mathbf{H}(\mathbf{pk}))$
$\mathbf{c} \leftarrow$ Kyber.CPAKEM.Enc($\mathbf{pk}, m, r$)
return $(\mathbf{c}, K)$
```

Algorithm 6 Kyber.CCAKEM.Decaps($\mathbf{sk}', \mathbf{c}$):

```
Input: Ciphertext $\mathbf{c} := (\mathbf{c_1}, c_2)$, secret key $\mathbf{sk}' := (\mathbf{sk}||\mathbf{pk}||\mathbf{H}(\mathbf{pk})||z)$
Output: Shared key $K$
$\hat{m} \leftarrow$ Kyber.CPAPKE.Dec($\mathbf{sk}, \mathbf{c}$)
$(K, r') \leftarrow \mathbf{G}(\hat{m}||\mathbf{H}(\mathbf{pk}))$
$\hat{\mathbf{c}} \leftarrow$ Kyber.CPAKEM.Enc($\mathbf{pk}, \hat{m}, r'$)
if $\mathbf{c} \neq \hat{\mathbf{c}}$ then
    $K := \mathbf{J}(z||\mathbf{c})$
return $K$
```

2.4 Parameters

Table 1: Parameter sets of ML-KEM

parameter sets	n	k	q	η_1	η_2	d_U	d_V
Kyber512	256	2	3329	3	2	10	4
Kyber768	256	3	3329	2	2	10	4
Kyber1024	256	4	3329	2	2	11	5

3 Attack on Kyber using Plaintext Checking oracles

3.1 Plaintext Checking Oracles

Following the seminal work of Huguenin-Dumittan and Vaudenay [HV20], one can define a *single* PCO as follows. Let $\bar{m}$ be an arbitrarily chosen reference plaintext, and Dec(sk, c) a generic decryption procedure. Then, a PCO can be defined as:

$$\mathcal{O}(c, \bar{m}) = \begin{cases} 1 & \text{if } \mathrm{Dec}(sk, c) = \bar{m} \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

In [TUX$^+$23], the authors extend the single PCO to multiple reference plaintexts

$\{\bar{m}_i\}_{0\leq i<n}$. The so-called *multi-plaintext checking oracle* (M-PCO) is defined as:

$$\mathcal{O}(c, \{\bar{m}_i\}_{0\leq i<n}) = i \text{ iff } \mathrm{Dec}(sk, c) = \bar{m}_i \tag{2}$$

It is worth noticing that the above oracle is well defined since the forged cyphertext c can only decrypt to one of the $\bar{m}_i$ by construction (see Section 3.3).

In all the previous works (starting with [HV20]), the authors assume that $\bar{m} = 0$ in the case of a single PCO and, that $\bar{m}_i$ is the null vector except for the first $\lceil \log_2(n) \rceil$ coefficients that encode the binary representation of i for $0 \leq i < n$ in the case of a M-PCO. Moreover n is chosen to be a power of 2. We shall use the same assumptions in the sequel.

In the rest of this section we will detail the key-recovery PCO and M-PCO attacks on Kyber. In Section 4, we shall define *noisy* versions of these oracles that can be constructed in practice through side-channel analysis.

3.2 Plaintext Checking Oracle Attack

The attack will recover, one by one, each of the 256 coefficients of the k polynomials in R_q that form Kyber secret key $\mathbf{sk}$. To this end, the attacker creates specific ciphertexts $\mathbf{c} = (\mathbf{c_1}, c_2)$ and send them through Kyber.CCAKEM.Decaps (Algorithm 6). From the description of Kyber.CPAPKE.Dec (Algorithm 2), we have the following computation of the decompressed message m':

$$m' = v' - \mathbf{sk}^T\mathbf{u}',$$

where $\mathbf{u'} = \mathsf{Decompress}_q(\mathbf{c_1}, d_u)$ and $v' = \mathsf{Decompress}_q(c_2, d_v)$. By choosing $\mathbf{c_1}$ such that all its polynomials are null except for a single one, it can be easily checked that $\mathbf{u'}$ has the same structure and the above equation rewrites:

$$m' = v' - sk \times u', \tag{3}$$

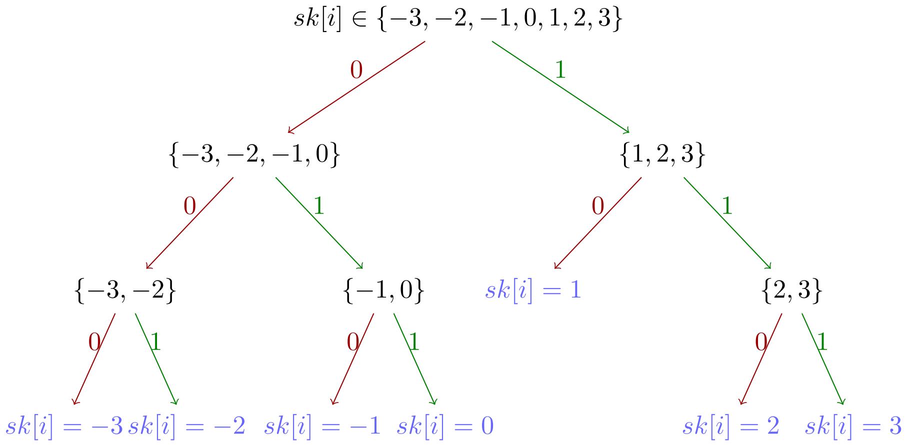
where u' (*resp.* c_1) denotes the non-null polynomial of $\mathbf{u'}$ (*resp.* $\mathbf{c_1}$) and sk the corresponding polynomial of $\mathbf{sk}$. In all the following we will then, *w.l.o.g.*, consider a unique polynomial.

Now recall that the decrypted message m is the 256 long binary vector $m = \mathsf{Compress}_q(m', 1)$. The attacker will construct ciphertexts (c_1, c_2) such that m is the null vector except for its first element and $m[0]$ depends on the value of a single coefficient of sk, say $sk[i]$ for some $0 \leq i < 256$,.

In order to recover $sk[i]$, one first needs to shift its position to the first one. This is possible thanks to u': by taking $u' = -\alpha X^{256-i}$, for some non-null value α, it is easy to see that the first coefficient of m' will be

$$m'[0] = v'[0] + \alpha sk[i]$$

Figure 1: A *balanced* binary research tree for Kyber512



If we take α such that, $\forall s \in \{-\eta_1, -\eta_1+1, \cdots, \eta_1\}$, $\mathsf{Compress}_q(\alpha s, 1) = 0$, then by setting v' a constant polynomial (*i.e.* $v'[i] = 0$ for $i = 1, \dots, 255$), the message $m = \mathsf{Compress}_q(v' - sk \times u', 1)$ has all its coefficients equel to zero except the first one, whose value is:

$$m[0] = \mathsf{Compress}_q(v'[0] + \alpha sk[i], 1) = \begin{cases} 0 & \text{if } |(v'[0] + \alpha sk[i]) \bmod q - \frac{q}{2}| > \lceil \frac{q}{4} \rfloor \\ 1 & \text{otherwise} \end{cases}$$

Since $sk[i] \in \{-\eta_1, \cdots, \eta_1\}$, for well chosen values of $v'[0]$ and α, one can divide the search space by 2. For instance:

$$m[0] = \mathsf{Compress}_q(v'[0] + \alpha sk[i], 1) = \begin{cases} 0 & \text{if } sk[i] \leq 0 \\ 1 & \text{otherwise} \end{cases}$$

The attack will then adaptively choose successive values of $v'[0]$ and follow a binary search (as illustrated in figure 1). Since the search space is of size $2\eta_1 + 1$, the attacker can expect to recover the value of a coefficient in about $\lceil \log_2(2\eta_1 + 1) \rceil$ oracle queries, assuming that the tree is balanced.

Taking into account the fact that the coefficients of sk are randomly drawn from the centered binomial distribution β_{η_1}, Qin *et al.* [QCZ+21] were able to exhibit, in the case of Kyber 512 (*resp.* 768, 1024), a lower bound on the average calls to the oracle: 2.375 (*resp.* 2.125, 2.125). Moreover, they propose the most efficient binary tree for the search: for Kyber 512 (*resp.* 768, 1024), this tree results in 2.56 (*resp.* 2.31, 2.31) calls on average to the oracle.

Notwithstanding the fact that our contributions are completely independent from the choice of the selected binary tree, our costs estimations for the PCO attacks will be conducted with [QCZ+21] so-called *optimal tree.*

Assuming that the binary search has an average number of calls to the oracle of o to retrieve a key coefficient, the overall attack will require $k \times 256 \times o$ calls in average. It can be observed that if the oracle answers incorrectly once during this whole attack process, it will affect a unique coefficient. It is usually observed in the literature (when considering oracles that can be sometimes wrong) that a single coefficient error in the final key (at an unknown position) can be brute-forced (this is true for all versions of Kyber). Therefore, an oracle that would answer incorrectly with probability $p_e = \frac{1}{k \times 256 \times o}$ would be *correct enough* to apply the above attack. For Kyber-1024, with the optimal tree, this corresponds to $p_e \approx 5 \cdot 10^{-4}$. Hence, we often find in the literature the threshold $p_e = 10^{-4}$, and equivalently the success rate's lower bound $\sigma = 1 - 10^{-4}$ for a *correct enough* oracle.

3.3 Multi-Plaintext Checking Oracle Attack

Tanaka *et al.* [TUX+23] show that the access to a M-PCO (as defined in Section 3.1) allows to reduce the number of oracle queries. Moreover, they demonstrate that such an oracle can be built in practice from side-channel leakages.

Following [TUX+23], the number of references messages will be set to 2^N. Moreover, $\forall i \in \{0, \cdots, 2^N - 1\}$ and $\forall j \in \{0, \cdots, 255\}, \bar{m}_i[j] = \lfloor \frac{i}{2^j} \rfloor \bmod 2$. In practice, N cannot be very large because the number of reference messages (and then the cost of the oracle's training) grows exponentially with N, *w.l.o.g.* we will take $N = 8$ in our cost estimations.

From the above description, the extension from single plaintext to multi-plaintext is quite straightforward: for some block index $a \in \{0, \cdots, 255 - N\}$, we now define

$$u' = -\alpha \cdot X^{256-a} \text{ and } v' = \sum_{i=0}^{N-1} \beta_i X^i$$

with α as before and for some chosen coefficient values β_i. We then have $m = \mathsf{Compress}_q(v' - sk \times u', 1)$ with all its coefficients null but the N firsts, whose values are, for $0 \leq i < N$,

$$\begin{aligned} m[i] &= \mathsf{Compress}_q(\beta_i + \alpha sk[a+i], 1) \\ &= \begin{cases} 0 & \text{if } |(\beta_i + \alpha sk[a+i]) \bmod q - \frac{q}{2}| > \lceil \frac{q}{4} \rfloor \\ 1 & \text{otherwise} \end{cases} \end{aligned}$$

Hence, m takes the value of one of the reference plaintexts, depending on the values $(\beta_i, \alpha, sk[a+i])$, and the oracle will tell which one. This allows to work simultaneously (and without dependence since the β_i can be chosen independently from each others) on

N consecutive key coefficients. When the binary search is finished for all of them, the next block is processed. This is extremely powerful and results in a gain up to 87% oracle calls (see [TUX$^+$23]) compared to the single plaintext oracle approach.

Let us remark that the authors of [TUX$^+$23] also show that, for Kyber 512 [1], the *optimal tree* [QCZ$^+$21] is too unbalanced (its longest path leads to 4 oracle calls) and should be replaced by a more balanced binary tree (that has a maximum of 3 oracle calls). This comes from the fact that in the M-PCO attack, all coefficient of a block must be retrieved before starting to work on the next block. In Section 5, we will show how to slightly relax this constraint and improve the overall cost of M-PCO attacks.

4 Noisy Plaintext Checking Oracles

We have seen that the (M-)PCOs defined in Section 3 lead to efficient key-recovery attacks on Kyber. In practice, such oracles are usually built from side-channel leakages, see *e.g.* [UXT$^+$22, TUX$^+$23, SCZ$^+$23, LCS$^+$25]. In this context, and very often in practice, the oracle may answer incorrectly with some non-null probability, we will refer to a *noisy oracle* (and denote it $\tilde{\mathcal{O}}$). In the literature, the considered noisy oracles have often a very small probability of error, and few papers actually tackle the issue of non-negligible error probabilities. This is however typically what one would expect from a Kyber implementation embedding side-channel countermeasures (*e.g.* masking). For instance, [UXT$^+$22] considers masked implementations and indeed, in this case, the oracle error probability is important.

A naive, yet effective, way to handle this noise is to make repeated calls to the oracle and choose the most frequent answer: this is the *majority-voting* approach. As we will see, side-channel based PCOs provide richer information than just a binary value. This extra information can be used to reduce the number of repeated calls thanks to a probability ratio test (as it is done in [UXT$^+$22] and traditionally in profiled side-channel analysis literature). In this section, we will first properly define a noisy oracle based on side-channel leakages, recall the classical approach of Probability Ratio Testing (PRT for short) and then introduce the Wald's test [Wal45] (or Sequential Probability Ratio Test, SPRT for short). We will show that using the latter allow to divide the number of repeated oracle calls by a factor between 2 and 3. We will see that SPRT is just PRT with early-arbort and has then a unique drawback: making the attack *online* (the attacker cannot harvest the side-channel traces in advance). This drawback does not apply for PCO attacks since they are already online attacks due to the decision tree procedure presented in the previous section.

[1] For Kyber 768 and 1024, there is no difference

To our knowledge – in the context of PCO attack on Kyber – there are only two papers that specifically focus on strongly noisy oracles, namely [SCZ$^+$23, LCS$^+$25], which propose high-level attack procedures to cope with the noise. These two approaches will be presented and compared to in the next sections 7 and 6 respectively.

4.1 Practical Considerations

Many recent publications (see, *e.g.*, [XPR$^+$21, NDGJ21, UXT$^+$22, TUX$^+$23, SCZ$^+$23, LCS$^+$25]) demonstrate that the message value can be reliably recovered using profiled side-channel analysis. These publications focus on different implementations of Kyber (and other FO-transform based KEM schemes), on different Hardware technology and using different side-channel acquisition chains. We identify two main reasons to explain the success of these attacks in practice:

- As recalled in [ABH$^+$22], the side-channel attacker is in a very advantageous position compared to classical (profiled) side-channel analysis contexts. The manipulation of $\hat{m}$ (see line 2 and 3 of Algorithm 6) results in *"[...] hundreds, thousands or even millions of intermediate bytes/words"* that are deterministically determined by the value $\hat{m}$. Moreover, in its simple form, the oracle only needs to distinguish two different messages (which, thanks to the hash function $\mathbf{G}$, have nothing in common very quickly during the computation).

- As recalled in many of the cited publications, the side-channel profiling step can be executed directly on the target device. Indeed, for any chosen message $\hat{m}$ it is easy to build a ciphertext $\mathbf{c} = (\mathbf{c_1}, c_2)$ which will be decrypted (for all secret key $\mathbf{sk}$) into $\hat{m}$.

To our knowledge, this situation is unique in the field of side-channel and gives to the attacker a huge advantage that results in very efficient attacks, even when the implementation involves side-channel countermeasures.

The purpose of this paper is not to provide another proof that this threat is real, but to show that, even in the case where the oracle is not perfect (and then answers with some relatively high probability of error), these kind of attacks are more efficient than expected from the state-of-the-art.

Concretely, there are two different approaches in the literature to construct an oracle in practice: using template profiling or deep-learning (DL for short) training. In both cases, the attacker must run the Kyber.CCAKEM.Decaps algorithm over chosen ciphertexts that decrypt into the reference message $\bar{m}$ (or, in case of multi-plaintext, one of the messages $\bar{m}_i$) and train the DL network (or equivalently infer the multivariate Gaussian distribution for the templates) with the captured side-channel execution traces. In the following section, we formally define the *noisy* PCOs that result from this training.

4.2 Noisy Plaintext Checking Oracles Definition

Let us consider a side-channel attacker able to capture a side-channel execution trace l during the execution of Kyber.Decaps(c, sk) for a chosen value of c and an unknown key sk. We define a generic noisy PCO as follows (given a reference message $\bar{m}$):

$$\tilde{\mathcal{O}}(c, \bar{m}, l) = \Pr(\mathrm{Dec}(sk, c) = \bar{m}|l)$$

and a noisy M-PCO (given multiple reference plaintexts $\{\bar{m}_i\}_{0\leq i<n}$):

$$\tilde{\mathcal{O}}(c, \{\bar{m}_i\}_{0\leq i<n}, l) = \{\Pr(\mathrm{Dec}(sk, c) = \bar{m}_i|l)\}_{0\leq i<n}$$

Note that these definitions are slightly more general than what is usually found in the literature, and provide more granularity. From a noisy oracle, we can define its *Bernoulli* version, denoted $\tilde{\mathcal{O}}_{\mathcal{B}}$, which is more common in the literature:

$$\tilde{\mathcal{O}}_{\mathcal{B}}(c, \bar{m}, l) = \begin{cases} 1 & \text{if } \tilde{\mathcal{O}}(c, \bar{m}, l) > 0.5 \\ 0 & \text{otherwise.} \end{cases}$$

Equivalently, for the noisy M-PCO, its (categorical) Bernoulli version is:

$$\tilde{\mathcal{O}}_{\mathcal{B}}(c, \{\bar{m}_i\}_{0\leq i<n}, l) = \mathrm{argmax}_i(\tilde{\mathcal{O}}(c, \{\bar{m}_i\}_{0\leq i<n}, l))$$

For the Bernoulli versions of the oracles, one can define a probability p_e of giving an erroneous answer (it can be computed analytically if the probability mass function of the corresponding noisy oracle is known, or estimated a posteriori otherwise). The popularity of the Bernoulli oracle in the literature certainly comes from their simplicity and notably the fact that one can compare attack algorithms costs with respect to the unique parameter p_e. Hence, when comparing our results with the literature, we will need to use these Bernoulli versions.

However, they are truncated version of the original noisy oracles, since part of the information is lost. And in fact, as observed in [UXT$^+$22], the side-channel based oracles (built from template profiling or DL training) coincide with our original definition.

From a Noisy M-PCO to N Noisy PCOs The M-PCO attack principle is recalled in Section 3.3, when the oracle is not noisy. In this case, with a single call to the oracle, the attacker retrieves information about n key coefficients and will treat these n information independently (n independent decision trees are followed) as if, these information were coming from n distinct calls to a PCO. To this end, the reference messages set cannot be arbitrarily selected, in fact they are chosen such that $\bar{m}_i$ (which is a vector of 256 binary coefficients) $N = \lceil \log_2(n) \rceil$ first coefficients coincide with the binary representation of i

and the rest is set to 0. In this specific context, the above definition of a noisy M-PCO is not adapted, and, *w.l.o.g.*, one can transform such an oracle as follows:

$$\widetilde{\mathcal{MO}}(c, \{\bar{m}_i\}_{0\leq i<N}, l) = \{\Pr(\mathrm{Dec}(sk, c)[i] = 0|l)\}_{0\leq i<N},$$

where $\mathrm{Dec}(sk, c)[i]$ is the i^{th} coefficient of $\mathrm{Dec}(sk, c)$.
And we have, for $0 \leq i < N$:

$$\Pr(\mathrm{Dec}(sk, c)[i] = 0|l) = \sum_{m_j \ s.t. \ m_j[i]=0} \Pr(\mathrm{Dec}(sk, c) = \bar{m}_j|l)$$

One can remark that, for $0 \leq j < n$:

$$\Pr(\mathrm{Dec}(sk, c) = \bar{m}_j|l) = \prod_{i=0}^{N-1} \Pr(\mathrm{Dec}(sk, c)[i] = \bar{m}_j[i]|l)$$

Therefore $\widetilde{\mathcal{MO}}(c, \{\bar{m}_i\}_{0\leq i<N}, l)$ and $\tilde{\mathcal{O}}(c, \{\bar{m}_i\}_{0\leq i<s}, l)$ are completely equivalent, since it is possible to construct one from the other and vice versa.

We then end up with N *virtual* (because they come from a single oracle call) noisy single PCO. Following the attack strategy, we choose to handle them independently. We hence only have to focus on noisy single PCOs in the rest of this section.

First let us present the natural *Probability Ratio Testing* (*PRT* for short) which is systematically used in the literature. Then we will introduce Wald's test [Wal45] (or *Sequential Probability Ratio Testing*, *SPRT* for short) in this context.

4.3 PRT Applied to a Noisy Plaintext-Checking Oracle

Given a side-channel sample l, the probability mass function of the associated message m being equal to the reference message $\hat{m}$ (or not) is directly given by the noisy PCO call:

$$\tilde{\mathcal{O}}(c, \bar{m}, l) = \Pr(\mathrm{Dec}(sk, c) = \bar{m}|l) = 1 - \Pr(\mathrm{Dec}(sk, c) \neq \bar{m}|l)$$

In a more general setting, this oracle can be seen as two probability mass functions:

$$\begin{aligned} f(l, \theta_0) &= \Pr(\mathrm{Dec}(sk, c) = \bar{m}|l) \\ f(l, \theta_1) &= \Pr(\mathrm{Dec}(sk, c) \neq \bar{m}|l) \end{aligned}$$

where θ_0 (*resp.* θ_1) is the distribution parameter of the side-channel data when Kyber.CCAKEM.Decaps manipulates a decrypted message $m = \bar{m}$ (*resp.* $m \neq \bar{m}$).

As previously stated, in a noisy oracle context (*i.e.* $f(l, \theta_i)$ takes other values than just 0 or 1), the attacker will repeatedly request the oracle with side-channel data $\{l_i\}_{0 \le i < T}$ generated from T executions of Kyber.CCAKEM.Decaps over a single crafted ciphertext c. From this data, the attacker wants to choose (with a very small error probability) between one of these two hypotheses:

$$\begin{aligned} \mathbf{H}_0 &: \quad m = \bar{m} \\ \mathbf{H}_1 &: \quad m \neq \bar{m} \end{aligned}$$

where m is the decrypted message from the ciphertext c. In the setting of a side-channel attack, the hypotheses are fully symmetric: accepting $\mathbf{H}_0$ is rejecting $\mathbf{H}_1$ and vice versa.

Let

$$\Lambda_T = \prod_{i=0}^{T-1} \frac{f(l_i, \theta_0)}{f(l_i, \theta_1)}$$

be the *likelihood ratio* between these two distributions. Assuming that the oracle is properly estimated (*i.e.* the profiling step has converged), the Neyman–Pearson lemma [NP33] tells us that Λ_T is the strongest metric to accept or reject hypothesis $\mathbf{H}_0$. In our simple symmetric case, this can be re-stated in:

$$\begin{aligned} \text{accept } \mathbf{H}_0 \quad & if \quad \Lambda_T \geq 1 \\ \text{accept } \mathbf{H}_1 \quad & if \quad \Lambda_T < 1 \end{aligned}$$

This probability ratio test of hypothesis (so-called *likelihood test*) is typically followed in all the literature of (profiled) side-channel analysis (since the seminal work of Chari *et al.* [CRR03]), often without mentioning it. For instance, the authors of [UXT$^+$22] strictly apply this test over the a trained DL network and compare the message complexity of their attack to the majority-voting approach. It can be observed that the majority-voting approach is equivalent to conducting the likelihood test over an oracle that involves Bernoulli distributions (*i.e.* the so-called Bernoulli oracle in the previous section) with the probability of error p_e as parameter.

In the classical Gaussian template setting, the probability mass functions follows a multivariate Gaussian distribution where the mean and covariance matrix have been estimated during the profiling phase.

It is hard to tell, a priori, what will be the best tool to build the oracle (DL or template), it will depend on the target processor and the means of capturing the sensitive leakages. We can remark however that, while it is always possible to turn a noisy oracle into a Bernoulli oracle (as seen in the previous section), this is never a good idea as it only means loosing information (as observed in [UXT$^+$22]). Nevertheless, most of the literature on PCO attacks illustrate their work on a Bernoulli oracle, for its simplicity but also maybe to avoid over-tuning a method for a specific profiling technique or leakage function.

4.4 SPRT Applied to a Noisy Plaintext-Checking Oracle

In this section, we present the *Sequential Probability Ratio Testing* (*SPRT* for short), also referred to as Wald's test [Wal45] in the literature. The seminal work of Wald in the early 1940's has had a profound impact in extremely diverse scientific domains, from medical studies to econometrics. Extensions and improvements of the Wald's test have been proposed in many different cases, depending on the a priori knowledge of the experimenter: the number of distributions, theirs forms, etc. Surprisingly, as far as we know, it was never considered in the field of side-channel analysis.

However, (supervised) side-channel analysis perfectly fits the basic setup of Wald's test, and we will see that it is particularly interesting in the case of PCO attacks.

Applying Wald's test in our context is straightforward as it is directly based on the likelihood test. It can be seen as an *early abort* strategy in the likelihood test (which is inherently a fixed sample-size test). In our context, this early abort strategy will have a huge impact on the message complexity of the attack. And it is pretty easy to see it: the value of T (the number of repeated calls to the noisy oracle) must be high enough such that wrongly accepting $\mathbf{H}_0$ happens with very low error probability (a typical chosen value for this error probability is 10^{-4}, see Section 3.2). It can be observed that the likelihood ratio value Λ_T actually provides a direct information on the probability of error of accepting or rejecting $\mathbf{H}_0$ (the closer Λ_T is to 1, the higher is the probability of error). Hence, in many cases, one can safely stop the oracle calls before reaching T.

In the SPRT process, the number of oracle requests is not a priori bounded and, at each new oracle output, the likelihood ratio is updated. At step t,

$$\Lambda_t = \prod_{i=0}^{t-1} \frac{f(l_i, \theta_0)}{f(l_i, \theta_1)},$$

and the test checks the value of Λ_t with respect to the thresholds A and B [2]:

$$\begin{aligned}
\text{If } A \geq \log \Lambda_t, &\quad \text{accept the hypothesis } \mathbf{H}_0 \text{ with error prob. } \alpha; \\
\text{If } B \leq \log \Lambda_t, &\quad \text{accept the hypothesis } \mathbf{H}_1 \text{ with error prob. } \beta; \\
\text{If } A < \log \Lambda_t < B, &\quad \qquad\qquad \text{got to next call to the oracle,}
\end{aligned}$$

[2] we use here the log likelihood notations as it is usually done in practice

where $A = \log\left(\frac{\alpha}{1-\beta}\right)$ and $B = \log\left(\frac{1-\alpha}{\beta}\right)$. Here, α and β are the probability of Type 1 and Type 2 errors. In our symmetric context, one usually wants $\alpha = \beta = 10^{-4}$). Moreover, we denote by σ the expected probability of success, *i.e.* $\sigma = 1 - \alpha$.

4.5 Oracle Cost Estimations and Comparisons

In order to estimate the impact of the use of SPRT in the PCO attacks, we conducted several simulations based on simple models: the Bernoulli oracle (as it is the most common in the literature) and the Gaussian template oracle.

For the former, we simply choose a probability of error p_e to define the Bernoulli distributions, p_e will take various values from (close to) 0 to 0.44. For the latter, we use a simple leakage model: pooled univariate Gaussian distribution. Hence, the message $m = 0$ (*resp.* $m = 1$) will produce a synthetic side-channel trace randomly drawn from a Gaussian distribution with mean μ_0 (*resp.* μ_1) and standard deviation s (common to 0 and 1). μ_0 (*resp.* μ_1) is arbitrarily set to 0 (*resp.* 1) while s will takes various values from 0 to 3.34. From $\mu_1 - \mu_0$ and s, we computed the error probability p_e of the Bernoulli version of this noisy oracle (s is then chosen such that the error probability spectrum is the same for both noisy oracles). This allows to draw the Bernoulli and Gaussian experiments on the same figure, for each error probability p_e tested.

Then, for each case, we systematically estimate, through simulations (over 10^6 tests), the average number of oracles calls in order to reach a decision ($m = 0$ or $m = 1$) with success probability $\sigma = 1 - 10^{-4}$. The experiments are conducted both for the likelihood test and Wald's test, the results are depicted in Figure 2. On the right sub-figure, the ratio gain of using SPRT over likelihood is shown. Interestingly, for both types of distributions, the gain is very similar, it reaches 2 as soon as the oracle is noisy (*i.e.* when repeated calls are actually needed). It even gets close to 3 for highly noisy oracles ($p_e \geq .25$).

5 Impact on Noisy Multi-plaintext Checking Oracle Attacks

As explained in Section 3.3, the attack using a multi-plaintext checking oracle (M-PCO) does not use the optimal tree from [QCZ$^+$21] but the more balanced one instead. Tanaka *et al.* [TUX$^+$23] explain why: for each block, the total number of oracle calls for the full recovery of the block will be the highest number of calls among all coefficients of the block.

Let us consider the following example on Kyber-512. For the block $(sk[0], sk[1], sk[2], sk[3]) = (0, 2, -1, -3)$, if we are using the optimal tree, -3 is recovered in 4 oracle queries whereas 0 requires only 2 queries. Since the coefficients of a block are all retrieved together, one must make 4 queries in order to retrieve -3 even if 0

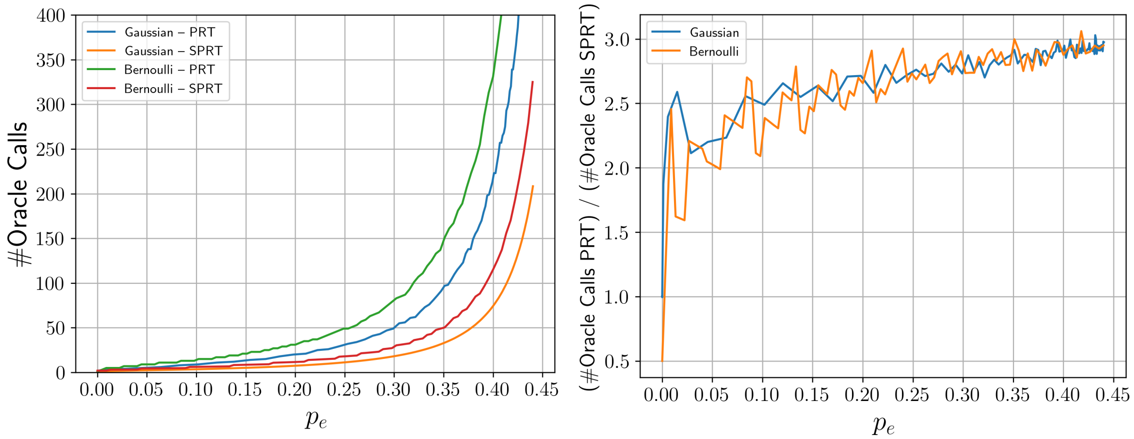


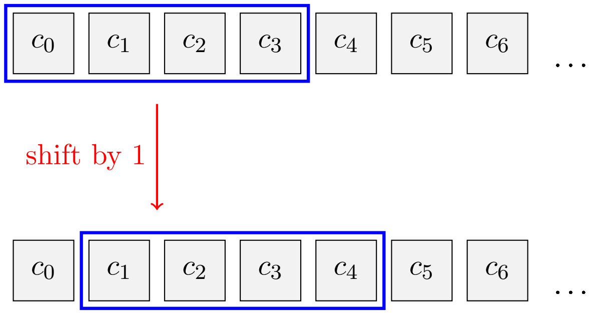
Figure 2: Number of Oracle Calls Comparison for PRT vs. SPRT

is already found.

Ideally, we would like to replace the finished coefficients by new ones as soon as possible, in order to remove the useless queries. However, if we want to treat blocks of coefficients that are not following each other, this would require the crafted Kyber ciphertext $c = (c_1, c_2)$ to be such that the polynomial c_1 has more than one non-null monomial. In turn, this would make a message coefficients dependent from two or more key coefficients (this comes from the polynomial multiplication in equation (3)). And, as far as we known, this situation does not allow to mount an attack.

All in all, the only degree of liberty the attacker has, is to choose a position a, a block length N and shift the block of N consecutive coefficients of sk into the N first positions. A simple, yet modest, improvement of the multi-plaintext checking oracle attack described in [TUX$^+$23] is then, whenever the leading coefficient of a block has been found, to shift the selected block by one[3] coefficient instead of waiting for all the N coefficients of sk to be found before shifting by N as it was originally proposed.

[3] In practice, if multiple leading coefficients are already known, the next block of computation will begin with the next unknown coefficient.



This transformation is at least as good as the classical method because we introduce flexibility in the positions of the blocks and then takes less oracle calls in average. We will still use the balanced tree as it gives better results, even with this shifting mechanism.

Table 2 gives the average number of calls to recover the full secrete key with a block size of $N = 8$ for a perfect (non-noisy) oracle (these averages have been estimated over 10^4 different secret keys, as it will be the case for all results shown in this section and the following). The discrepancy of results between Kyber 512 and the two others comes from the difference in the attack decision binary trees (see Section 3.2).

Table 2: Comparison of calls between estimation of classical multi-plaintext attack from [TUX+23] and simulation of shifting multi-plaintext attacks for different versions of Kyber with $N = 8$, under the assumption of a perfect oracle.

Accuracy	**Method**	**#TotalCalls**
Kyber 512	Classical	192 *(ref)*
	Shifting	189.81 (–1.1%)
Kyber 768	Classical	283.21 *(ref)*
	Shifting	253.82 (–10.4%)
Kyber 1024	Classical	377.61 *(ref)*
	Shifting	338.18 (–10.4%)

Table 3 shows the results for a noisy Bernoulli multi-plaintext oracle (for comparison purpose with the literature), depending on the use of majority-voting [4] (*i.e.* PRT) or SPRT and the use of shifting or not. As expected, the use of SPRT provides a huge benefit without any disadvantage. Moreover, it can be checked that the shifting method

[4] They are denoted "MV ($t = 11$)" in the table, as the number of repetition is 11

gives even better results when combined with SPRT. This comes from the fact that the standard deviation of the number of calls is quite high for SPRT (compared to PRT) and then benefits from the greater granularity of the new shifting method.

Table 3: Comparison of number of oracle calls and number of errors per key between majority voting and SPRT methods, with and without shifting, for all versions of Kyber. Block size $N = 8$ and noisy oracle with $p_e = 0.1$.

Accuracy	**Method**	**#TotalCalls**	**#Error/#Coeff**
Kyber 512	Classical MV (t=11)	2111.99 *(ref)*	0.007/512
	Shifting MV (t=11)	2087.95 (–1.1%)	0.008/512
	Classical SPRT	1158.65 (–45.1%)	0.001/512
	Shifting SPRT	1084.41 (–48.7%)	0.000/512
Kyber 768	Classical MV (t=11)	3115.53 *(ref)*	0.012/768
	Shifting MV (t=11)	2792.34 (–10.4%)	0.010/768
	Classical SPRT	1323.37 (–57.5%)	0.015/768
	Shifting SPRT	1159.88 (–62.8%)	0.013/768
Kyber 1024	Classical MV (t=11)	4153.44 *(ref)*	0.015/1024
	Shifting MV (t=11)	3721.96 (–10.4%)	0.013/1024
	Classical SPRT	1763.45 (–57.6%)	0.019/1024
	Shifting SPRT	1545.98 (–62.8%)	0.020/1024

6 Grafting Tree

In [LCS$^+$25] the authors propose an original method to correct key coefficients errors after a M-PCO attack with a noisy oracle. Their results (reproduced in Table 4) show siginficant gains (ranging from about -22% to -44% depending on the oracle's p_e). We will however advocate that the extent of these gains can be largely explained by their comparison to the naive majority voting approach, i.e. without using any early abort strategy. The core idea of [LCS$^+$25] correction follows three steps:

1. Recover a *rough* coefficient $\hat{sk}[i]$ by performing the M-PCO attack

2. Build an optimal decision tree assuming that the value $\hat{sk}[i]$ is correct and perform the coefficient recovery using this tree. Compare the new value $\hat{sk}'[i]$ with $\hat{sk}[i]$. If they are the same, stop at this step and return $\hat{sk}[i]$.

3. Perform another coefficient recovery with the original tree and compare the new value $\hat{sk}''[i]$ to the others. If $\hat{sk}''[i] = \hat{sk}'[i]$, then return $\hat{sk}'[i]$, otherwise return $\hat{sk}[i]$.

The algorithm is well-suited to oracles that are moderately noisy. Indeed, as the number of errors in the first or second stage are relatively low, stage 3 is rarely required. Also, since the second tree is built assuming that $\hat{sk}[i]$ is correct, it only needs two calls to the oracle if $\hat{sk}[i]$ is confirmed. For noisier oracles, the authors use majority-voting to artificially raise the quality of the oracle to a more practical threshold. It is thus very easy and natural to replace the majority-voting with SPRT in their method.

For Kyber-1024 [5] the authors of [LCS$^+$25] propose different strategies depending on the oracle's probability of error (p_e):

- If $p_e \in [0.001; 0.017]$, the Grafting Tree method does not produce more than one incorrect coefficient per key (on average), so it can be used without majority-voting.
- If $p_e \in]0.017; 0.050]$, the authors propose to improve the quality of the oracle with majority-voting for the second and third steps only.
- If $pe > 0.050$, the authors propose to improve the quality of the oracle using majority-voting for all steps.

As a result, their approach uses majority-voting to reach a probability of error within the *healthy zone* $[0; 0.017]$.

Table 4, which is directly extracted from [LCS$^+$25], shows the comparison between the classical majority voting method and the method from [LCS$^+$25]. The attack simulation is on Kyber-1024 using a noisy Bernoulli M-PCO of block size $N = 8$. In Table 4, MV stands for majority voting and the value t is the number of experiments.

Table 5 shows the attack's cost for Kyber-1024 with a noisy Bernoulli M-PCO of block size $N = 8$. For fair comparison, the [LCS$^+$25] results of Table 5 have been recomputed using the shifting mechanism presented in the previous section and the appropriate algorithm depending on the probability of error: for $p_e \in \{0.001, 0.017\}$ we did not use any correction on the oracle, for $p_e = 0.050$ we used majority voting (t=3) in the second and third steps and for $p_e = 0.100$ we used majority voting (t=3) for every steps.

The goal of Table 5 is to show that the gain evaluation of the attack from [LCS$^+$25] is overestimated as it was originally compared to under-performing statistical method.

The results indicate that for $p_e \leq 0.017$, the average gain of the method proposed in [LCS$^+$25] over the SPRT is relatively modest, approximately -12%. This contrasts with the original paper, where the reported improvement over previous methods was about -44%. For noisier oracles, the advantage even disappears. In fact, under such conditions, the method of [LCS$^+$25] requires more calls on average than the SPRT, with overheads of approximately 17.8% for $p_e = 0.050$ and 63.4% for $p_e = 0.1$.

The differences between the number of errors (see the **#Error/#Coeff** column) in Table 5 are explained by the discret aspect of the different approaches which makes some

[5] Kyber-1024 is the only context where the authors of [LCS$^+$25] consider really noisy oracles, *i.e.* for $p_e > 0.017$

accuracy values unreachable. While our goal is to have fewer than 1 erroneous coefficient per full key recovery (as explained in Section 3.2), reaching exactly this goal is often not possible. It is then important to take into consideration the combination of the average number of traces and the average number of errors to have a more accurate interpretation of the results.

Taking this into account, the advantage of using [LCS+25] compared to SPRT alone is close to nothing for $p_e \leq 0.017$ and even negative after that point.

Table 4: Comparison between majority-voting and [LCS+25] for Kyber 1024 with $N = 8$ under a noisy oracle with $p_e \in \{0.001, 0.050, 0.100\}$.

Accuracy	**Method**	**#TotalCalls**	**#Error/#Coeff**
$p_e = 0.001$	MV (t = 3)	1152 *(ref)*	0.000/1024
	[LCS+25]	635.5 (–43.9%)	0.003/1024
$p_e = 0.050$	MV (t = 5)	1888 *(ref)*	0.372/1024
	[LCS+25]	1332 (–29.4%)	0.351/1024
$p_e = 0.100$	MV (t=7)	2643 *(ref)*	0.477/1024
	[LCS+25]	2041 (–22.8%)	0.641/1024

Table 5: Comparison of calls and number of error per key between our SPRT method and the method from [LCS+25] with shifting for Kyber 1024 with $N = 8$ under a noisy oracle with $p_e \in \{0.001, 0.017, 0.050, 0.100\}$.

Accuracy	**Method**	**#TotalCalls**	**#Error/#Coeff**
$p_e = 0.001$	Shifting SPRT	677.66 *(ref)*	0.000/1024
	Shifting [LCS+25]	596.34 (-12.0%)	0.003/1024
$p_e = 0.017$	Shifting SPRT	692.85 *(ref)*	0.172/1024
	Shifting [LCS+25]	626.44 (-9.6%)	0.924/1024
$p_e = 0.050$	Shifting SPRT	1086.54 *(ref)*	0.041/1024
	Shifting [LCS+25]	1279.7 (+17.8%)	0.518/1024
$p_e = 0.100$	Shifting SPRT	1165.65 *(ref)*	0.341/1024
	Shifting [LCS+25]	1905.10 (+63.4%)	0.707/1024

7 Checking by block

In this section, we assume that the attacker does not have access to a M-PCO [6], and therefore consider a single PCO. In [SCZ$^+$23], Shen *et al.* propose a checking method to quickly (in terms of number of oracle calls) detect if a block of $N \in \{2, 4\}$ consecutive key coefficients were correctly guessed. In fact, the *Grafted Tree* method [LCS$^+$25] from the same group of authors, can be seen as an extension of the checking-by-block method to a M-PCO.

Similarly to [LCS$^+$25], the authors of [SCZ$^+$23] exhibit impressive gains (results are reproduced in Table 8 [7]), ranging from -45% to about -63% depending on the oracle's p_e. Here again, we will show that shifting from naive majority voting to SPRT as a baseline mecanism removes a large part of these gains. This is particularly interesting in the case of the checking-by-block method as it is actually built upon a complicated, ad-hoc early abort strategy (in the case of a *low accuracy oracle*, *i.e.* noisy oracle with $p_e \geq 0.010$, as per [SCZ$^+$23]). In fact, we will show next that using SPRT for the oracle calls allows to completely remove their ad-hoc early abort strategy and solely keep the core of checking-by-block strategy, greatly simplifying the approach and making it the exact equivalent of the *Grafted Tree* method for single PCOs.

7.1 Method Description

The main idea is to use well-constructed ciphertexts to check coefficient blocks of size N with only 2 queries to the oracle. The authors of [SCZ$^+$23] demonstrate that such ciphertexts exist and can be iteratively computed for each particular block (for $N \leq 4$).

The attack procedure starts by building a *rough* key $\hat{sk}$ using a noisy PCO. Since some coefficients of $\hat{sk}$ might be erroneous, the attacker can check the correctness of a block of N coefficients with only two calls to the oracle. To do so, the attacker uses pairs of expressly crafted ciphertexts (ct_1, ct_2) allowing them to decide whether ot not a block is correct.

Example: Let us consider the first block of size 2 from $\hat{sk}$, *i.e.* $(\hat{sk}[0], \hat{sk}[1])$. [SCZ$^+$23] shows that there exists a pair of ciphertexts (ct_1, ct_2) such that $(\mathcal{O}(ct_1, \hat{m}), \mathcal{O}(ct_2, \hat{m})) = (0, 1)$ if and only if $(sk[0], sk[1]) = (\hat{sk}[0], \hat{sk}[1])$, and $(\mathcal{O}(ct_1, \hat{m}), \mathcal{O}(ct_2, \hat{m})) \neq (0, 1)$ otherwise. In all the following we shall assume, *w.l.o.g.*, that $\hat{m}$ is the message with all 0 coefficients.

For instance (for Kyber-512), for $(\hat{sk}[0], \hat{sk}[1]) = (0, 2)$, the ciphertexts $ct_1 = (32 - 48 \cdot X^{255}, 5)$ and $ct_2 = (22 - 43 \cdot X^{255}, 5)$ produce the following oracle outputs:

[6] Several reasons could force an attacker to use a single PCO. For instance because the oracle training becomes too expensive for multi-plaintext, or leads to higher probabilities of error. Another practical reason could be that the attacker uses a pre-trained oracle in order to reduce the overall attack complexity.

[7] All results in this section are given for Kyber-512, to follow the choice of [SCZ$^+$23].

Table 6: $\mathcal{O}(\mathrm{ct}_1)||\mathcal{O}(\mathrm{ct}_2)$ for $[-3,3]\times[-3,3]$ from [SCZ+23]

$sk[0]\backslash sk[1]$	-3	-2	-1	0	1	2	3
-3	“11”	“11”	“11”	“11”	“11”	“11”	“11”
-2	“11”	“11”	“11”	“11”	“11”	“11”	“11”
-1	“11”	“11”	“11”	“11”	“11”	“11”	“11”
0	“11”	“11”	“11”	“11”	“11”	“01”	“00”
1	“11”	“11”	“11”	“11”	“00”	“00”	“00”
2	“11”	“11”	“00”	“00”	“00”	“00”	“00”
3	“11”	“00”	“00”	“00”	“00”	“00”	“00”

Because $\mathcal{O}(\mathrm{ct}) = \mathsf{Compress}_q(v' - sk[0]u'[0] - sk[1]u'[255], 1)$, such ciphertexts pair exist for each different blocks of size 2, with $sk[i] \in \{-3, \ldots, 3\}$, which represents 49 different pairs. But for larger blocks, the existence of these cipher pairs is not guaranteed, and when they do not exist, the blocks are divided in half to test smaller blocks. The ciphertexts pairs are constructed by exhaustive search through the whole space of coefficients and ciphertexts. To reduce computation time, the construction of these pairs is limited to blocks of size 4.

Correction and recovery are then carried out according to two different algorithms, one for high-accuracy oracles with reliable responses, and the other for low-accuracy oracles requiring additional corrections. In [SCZ+23], the accuracy threshold (between high and low) is around 0.99 (or equivalently $p_e = 0.01$). These two processes are described hereafter:

For high-accuracy:

1. Recover a rough key $\hat{sk}$ by executing the classical attack method with the optimal tree
2. Check all blocks using the method explained above and mark any suspicious blocks
3. For each suspect block, perform another retrieval for each coefficient in the blocks and replace the suspect coefficients with these coefficients. Finally return the key.

For low-accuracy:

1. Define constants cc_1, cc_2 and the list cc_t depending on the oracle's probability of error p_e (see Table 7). A round counter r is initialized to 1.

2. Recover a rough key $\hat{sk}$ by executing the classical attack method with the optimal tree.

3. Check all blocks using the method explained above and mark any suspicious blocks.

4. For each block, perform coefficient recovery. If the block has been marked as non-suspicious, add cc_1 to the *confidence level* of the value found. If the block was marked as suspicious, add cc_2.

5. For each coefficient, replace the value of $sk[i]$ by the value with the highest confidence level. If this confidence level is greater than the threshold $cc_t[r]$ for round r, mark the coefficient as "Finished".

6. If all coefficients are marked as "Finished", return the key, otherwise increment the round counter r by one and go back to step (2).

Table 7: The precomputed parameters from [SCZ+23]

p_e	cc_1	cc_2	$cc_t[r]$, for $r = 1, 2, \ldots$
0.04	4	3	{5, 9, 11, 14, 14, 18, 18, 18, 18, 22, 22, ...}
0.05	4	3	{5, 9, 12, 13, 13, 16, 17, 17, 21, 21, 21, ...}
0.10	3	4	{5, 9, 13, 16, 20, 20, 20, 22, 24, 25, 26, ...}

The figures from Table 7 have been found experimentally by testing and selecting the best couple (cc_1, cc_2) and the list cc_t for a given probability of error p_e.

For reference, the results of [SCZ$^+$23] approach are recalled in Table 8. As before, the gains are relative to the naive majority voting. As in the previous section, we will show that these impressive gains are, for a large part, artificial and are lost when using SPRT as the baseline.

7.2 Comparison with SPRT

Similarly to the previous section 5, we study here the addition of SPRT to the checking-by-block method. The intuition is that the ad-hoc early-abort strategy of [SCZ$^+$23] in the case of low accuracy oracles can be removed by the use of SPRT (which is, inherently, an early-abort strategy).

To compare with the algorithm cost (in terms of number of call to the noisy oracle) of [SCZ$^+$23]'s approach for the low-accuracy ([SCZ$^+$23, LA] for short) cases, we tested two approaches:

- SPRT alone;

Table 8: Comparison between majority-voting and [SCZ+23, LA].

Accuracy	Method	#TotalCalls	#Error/#Coeff
$p_e = 0.04$	MV (t = 7)	9185.0 *(ref)*	0.11/512
	[SCZ+23, LA]	3424.91 (–62.7%)	0.05/512
$p_e = 0.05$	MV (t = 7)	9185.0 *(ref)*	0.11/512
	[SCZ+23, LA]	3874.5 (–57.8%)	0.15/512
$p_e = 0.10$	MV (t = 11)	14433.3 *(ref)*	0.39/512
	[SCZ+23, LA]	7773.9 (–46.1%)	0.25/512

- SPRT to reach the oracle accuracy of $p_e = 0.01$ and then, with this less-noisy (but more costly) oracle, apply [SCZ+23]'s approach for the high-accuracy ([SCZ+23, HA] for short) cases.

To compare with [SCZ+23, LA]'s results, we focus on their considered cases: $p_e = 0.04, 0.05$ and 0.1, the results are given in Table 9.

Table 9: Comparison between SPRT alone, [SCZ+23, LA] and SPRT + [SCZ+23, HA]

Accuracy	Method	#TotalCalls	#Error/#Coeff
$p_e = 0.04$	SPRT	4217.71 *(ref)*	0.10/512
	[SCZ+23, LA]	3424.91 (–18.8%)	0.05/512
	SPRT + [SCZ+23, HA]	3457.1 (–18.0%)	0.08/512
$p_e = 0.05$	SPRT	4371.74 *(ref)*	0.20/512
	[SCZ+23, LA]	3874.5 (–11.4%)	0.15/512
	SPRT + [SCZ+23, HA]	3561.8 (–18.5%)	0.18/512
$p_e = 0.10$	SPRT	6557.8 *(ref)*	0.21/512
	[SCZ+23, LA]	7773.9 (+18.5%)	0.25/512
	SPRT + [SCZ+23, HA]	5946.3 (-9.3%)	0.06/512

Firstly, the results of the tables 8 and 9 show that the gain is considerably reduced if the SPRT method is used as the reference method instead of majority voting. The gain drops from −63% to −19% in the best case and, for the noisier oracles with $p_e = 0.10$, the

[SCZ$^+$23, LA] method is even less efficient than the reference SPRT method. These results highlights the importance of choosing the SPRT method as the reference.

The results also show that the SPRT + [SCZ$^+$23, HA] is always the best, demonstrating that the use of SPRT completely voids the [SCZ$^+$23, LA] ad-hoc early-abort strategy.

8 Conclusion

In this paper we consider *plaintext-checking oracle* attacks on Kyber when oracles are *noisy* (*i.e.* oracles that can be erroneous). We show that, in this context of online (*i.e.* adaptive) attacks, the use of Sequential Probability Ratio Test (SPRT) can efficiently replace the classical majority-voting or likelihood ratio test strategies.

To evaluate the proposed method, we conducted several simulations using oracles with varying noise levels and compared the outcomes with previous results obtained under similar models and attack strategies.

Our results demonstrate that using SPRT – in comparison to other classical baselines – achieves a significant reduction in the number of oracle calls while maintaining the same level of flexibility in online attack scenarios. Furthermore, we showed that recent improvement techniques proposed in [LCS$^+$25] and [SCZ$^+$23] are substantially affected by this change – sometimes performing worse, but also potentially serving as complementary methods. Overall, the proposed SPRT-based strategy establishes a new baseline in this particular case, combining strong efficiency with high adaptability compared to existing approaches.

Future work could extend this approach to other attack scenarios (*e.g.* the *Full Domain* oracle model) and other post-quantum KEMs. Evaluating the SPRT-based strategy across different schemes would help assess its generality and potential as a versatile tool for efficient online attacks.

References

[ABH$^+$22] M. Azouaoui, O. Bronchain, C. Hoffmann, Y. Kuzovkova, T. Schneider, and F.-X. Standaert. Systematic study of decryption and re-encryption leakage: The case of Kyber. In J. Balasch and C. O'Flynn, editors, *COSADE 2022*, volume 13211 of *LNCS*, pages 236–256. Springer, Cham, April 2022.

[CRR03] S. Chari, J. R. Rao, and P. Rohatgi. Template attacks. In B. S. Kaliski, Jr., Çetin Kaya. Koç, and C. Paar, editors, *CHES 2002*, volume 2523 of *LNCS*, pages 13–28. Springer, Berlin, Heidelberg, August 2003.

[DG25] H. Dong and Q. Guo. Multi-value plaintext-checking and full-decryption oracle-based attacks on HQC from offline templates. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2025(4):254—289, 2025.

[FO99] E. Fujisaki and T. Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In M. J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 537–554. Springer, Berlin, Heidelberg, August 1999.

[GNNJ23] Q. Guo, D. Nabokov, A. Nilsson, and T. Johansson. SCA-LDPC: A code-based framework for key-recovery side-channel attacks on post-quantum encryption schemes. In J. Guo and R. Steinfeld, editors, *ASIACRYPT 2023, Part IV*, volume 14441 of *LNCS*, pages 203–236. Springer, Singapore, December 2023.

[HV20] L. Huguenin-Dumittan and S. Vaudenay. Classical misuse attacks on NIST round 2 PQC - the power of rank-based schemes. In M. Conti, J. Zhou, E. Casalicchio, and A. Spognardi, editors, *ACNS 2020, Part I*, volume 12146 of *LNCS*, pages 208–227. Springer, Cham, October 2020.

[LCS$^+$25] J. Li, C. Cheng, M. Shen, P. Chen, Q. Guo, D. Liu, L. Wu, and J. Weng. Grafted trees bear better fruit: An improved multiple-valued plaintext-checking side-channel attack against kyber. In *2025 Design, Automation & Test in Europe Conference (DATE)*, pages 1–7, 2025.

[MLK24] Module-Lattice-Based Key-Encapsulation Mechanism Standard. National Institute of Standards and Technology, NIST FIPS PUB 203, U.S. Department of Commerce, August 2024.

[NDGJ21] K. Ngo, E. Dubrova, Q. Guo, and T. Johansson. A side-channel attack on a masked IND-CCA secure Saber KEM implementation. *IACR TCHES*, 2021(4):676–707, 2021.

[NP33] J. Neyman and E. S. Pearson. On the problem of the most efficient tests of statistical hypotheses. *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character*, 231:289–337, 1933.

[QCD19] Y. Qin, C. Cheng, and J. Ding. An efficient key mismatch attack on the NIST second round candidate Kyber. Cryptology ePrint Archive, Report 2019/1343, 2019.

[QCZ$^+$21] Y. Qin, C. Cheng, X. Zhang, Y. Pan, L. Hu, and J. Ding. A systematic approach and analysis of key mismatch attacks on lattice-based NIST candidate KEMs. In M. Tibouchi and H. Wang, editors, *ASIACRYPT 2021, Part IV*, volume 13093 of *LNCS*, pages 92–121. Springer, Cham, December 2021.

[RRCB20] P. Ravi, S. S. Roy, A. Chattopadhyay, and S. Bhasin. Generic side-channel attacks on CCA-secure lattice-based PKE and KEMs. *IACR TCHES*, 2020(3):307–335, 2020.

[RRD$^+$23] G. Rajendran, P. Ravi, J.-P. D'Anvers, S. Bhasin, and A. Chattopadhyay. 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.

[SAB$^+$22] P. Schwabe, R. Avanzi, J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, G. Seiler, D. Stehlé, and J. Ding. CRYSTALS-KYBER. Technical report, National Institute of Standards and Technology, 2022. available at `https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022`.

[SCZ$^+$23] M. Shen, C. Cheng, X. Zhang, Q. Guo, and T. Jiang. 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.

[TUX$^+$23] Y. Tanaka, R. Ueno, K. Xagawa, A. Ito, J. Takahashi, and N. Homma. Multiple-valued plaintext-checking side-channel attacks on post-quantum KEMs. *IACR TCHES*, 2023(3):473–503, 2023.

[UXT$^+$22] R. Ueno, K. Xagawa, Y. Tanaka, A. Ito, J. Takahashi, and N. Homma. Curse of re-encryption: A generic power/EM analysis on post-quantum KEMs. *IACR TCHES*, 2022(1):296–322, 2022.

[Wal45] A. Wald. Sequential tests of statistical hypotheses. *Ann. Math. Statist.*, 16(4):117–186, 1945.

[XIU$^+$21] K. Xagawa, A. Ito, R. Ueno, J. Takahashi, and N. Homma. Fault-injection attacks against NIST's post-quantum cryptography round 3 KEM candidates. In M. Tibouchi and H. Wang, editors, *ASIACRYPT 2021, Part II*, volume 13091 of *LNCS*, pages 33–61. Springer, Cham, December 2021.

[XPR$^+$21] Z. Xu, O. Pemberton, S. Roy, D. Oswald, W. Yao, and Z. Zheng. Magnifying side-channel leakage of lattice-based cryptosystems with chosen ciphertexts: The case study of kyber. *IEEE Transactions on Computers*, 71:1–1, 01 2021.