IACR Transactions
ISSN XXXX-XXXX, Vol. 0, No. 0, pp. 1–23. DOI:XXXXXXXX

Grafted Trees Bear Better Fruit: An Improved Multiple-Valued Plaintext-Checking Side-Channel Attack against Kyber

Jinnuo Li[1], Chi Cheng[1,✉], Muyan Shen[2], Peng Chen[1], Qian Guo[3], Dongsheng Liu[4], Liji Wu[5] and Jian Weng[6]

[1] Hubei Key Laboratory of Intelligent Geo-Information Processing, School of Computer Science, China University of Geosciences, Wuhan, China
[2] School of Cryptology, University of Chinese Academy of Sciences, Beijing, China
[3] Lund University, Lund, Sweden
[4] Huazhong University of Science and Technology, Wuhan, China
[5] School of Integrated Circuits, Tsinghua University, Beijing, China
[6] College of Cyber Security, Jinan University, Guangzhou, China
chengchi@cug.edu.cn

Abstract. As a prominent category of side-channel attacks (SCAs), plaintext-checking (PC) oracle-based SCAs offer the advantages of generality and operational simplicity on a targeted device. At TCHES 2023, Rajendran et al. and Tanaka et al. independently proposed the multiple-valued (MV) PC oracle, significantly reducing the required number of queries (a.k.a., traces) in the PC oracle. However, in practice, when dealing with environmental noise or inaccuracies in the waveform classifier, they still rely on majority voting or the other technique that usually results in three times the number of queries compared to the ideal case.

In this paper, we propose an improved method to further reduce the number of queries of the MV-PC oracle, particularly in scenarios where the oracle is imperfect. Compared to the state-of-the-art at TCHES 2023, our proposed method reduces the number of queries for a full key recovery by more than 42.5%. The method involves three rounds. Our key observation is that coefficients recovered in the first round can be regarded as prior information to significantly aid in retrieving coefficients in the second round. This improvement is achieved through a newly designed grafted tree. Notably, the proposed method is generic and can be applied to both the NIST key encapsulation mechanism (KEM) standard Kyber and other significant candidates, such as Saber and Frodo. We have conducted extensive software simulations against Kyber-512, Kyber-768, Kyber-1024, FireSaber, and Frodo-1344 to validate the efficiency of the proposed method. An electromagnetic attack conducted on real-world implementations, using an STM32F407G board equipped with an ARM Cortex-M4 microcontroller and Kyber implementation from the public library pqm4, aligns well with our simulations.

Keywords: NIST post-quantum cryptography standardization · Lattice-based cryptography · Kyber · Side-channel attacks · multiple-valued plaintext-checking oracle.

1 Introduction

With the advent of quantum computing, mathematical problems integral to current public-key cryptographic systems, such as integer factorization or discrete logarithms, will be

efficiently solved by Shor's algorithm [Sho99]. Recognizing this threat, in 2016, the National Institute of Standards and Technology (NIST) initiated the standardization process for post-quantum cryptography (PQC), seeking candidates conjectured to resist both classical and quantum attacks [Moo16].

Lattice-based cryptographic algorithms constitute a significant portion of several rounds of NIST selection. Among them, CRYSTALS-Kyber [ABD+19], which is based on the Module Learning with Errors (M-LWE) problem, was ultimately selected for standardization of public-key encryption (PKE) or key encapsulation mechanism (KEM) in July 2022 [AAC+22]. As a consequence, in August 2023, NIST released FIPS 203 ML-KEM, a draft standard that adopts CRYSTALS-Kyber as the cornerstone [NIS23]. More recently, Kyber has been integrated into the transition to quantum-resistant cryptography at Google. This strategic move positions traffic on the Google Chrome browser to support a hybrid mechanism named X25519Kyber-768, combining the use of both conventional Elliptic Curve Cryptography X25519 and Kyber-768.

Given the anticipated widespread implementation of CRYSTALS-Kyber across diverse computational platforms and applications, it is imperative to assess its implementation security comprehensively. This evaluation includes a thorough examination of vulnerability to Side-Channel Attacks (SCAs). Initially introduced by Kocher in 1996 [Koc96], SCAs exploit various types of leaked information, such as timing, cache timing, power consumption or electromagnetic emanation, to glean information about the long-term secret key or message in cryptographic algorithms.

In the design of LWE-based NIST PQC PKEs/KEMs, a common approach is to utilize the Fujisaki-Okamoto (FO) transformation [FO99]. This involves starting with a construction secure against chosen plaintext attacks (CPA) and then transforming it to be secure against chosen ciphertext attacks (CCA). A prominent research focus has been to identify and mitigate side-channel leakages that could compromise the CCA protection offered by the FO transformation.

1.1 Related works

SCAs against CCA-secure KEMs. NIST explicitly encourages further research into the security analysis of post-quantum cryptographic schemes against SCAs, aligning with the need to fortify cryptographic systems against potential threats. In line with this concept, numerous SCAs against CCA-secure KEMs have been proposed, such as those in [DTVV19, GJN20, RRCB20, NDGJ21, HHP+21, GNNJ23]. These attacks vary in their models, making differences in the difficulties of mounting such attacks and the baseline of attacking sample complexities (i.e., the required number of queries or traces). The first type of adversary model, known as the plaintext-checking (PC) oracle, operates by comparing the decrypted message to a predetermined value [RRCB20, UXT+22]. In this attack model, the adversary gains 1 bit of information about the secret key with each query, gradually recovering the entire secret key. The second attack model is referred to as the decryption-failure (DF) oracle. It operates by exploiting a decryption failure correlated with the value of the secret key, which is particularly effective against the non-constant-time implementation of the algorithms [GJN20, BDH+21, DHP+21]. These two types of attacks gain limited information from each query, making them more generic. Yet another adversary model, the full-decryption (FD) oracle, exploits specific leaky operations of the decryption algorithm to further reduce the sample complexities [XPSR+21, NDGJ21]. Attacks based on the FD oracle require fewer traces and are thus more efficient. However, the operations vulnerable to FD oracle exploitation are relatively rapid compared to the whole re-encryption process targeted in other attack types. Consequently, implementing countermeasures against the PC oracle-based attacks could result in a substantially significant performance detriment.

Parallel variant of the PC oracle. The concept of the parallel PC oracle or Multiple-valued (MV) PC oracle is first introduced at TCHES 2023. These works were

motivated by the following observations: the binary PC oracle-based attack extracts only a single bit of information from an extensive leakage trace of the entire re-encryption process, which incorporates thousands of leakage points. These points in the power traces leak partial secret information. Thus, there is a potential to uncover multiple bits from a single trace, significantly enhancing attack efficiency. To strike a balance, the parallel PC oracle is introduced. It stands between the PC oracle-based and FD oracle-based attacks in terms of trace requirements: it needs fewer traces than the binary PC oracle-based attack (hundreds instead of thousands) but more than the FD oracle-based attack (which requires only a few). The MV-PC oracle-based attack deserves special attention for its generic feature and the significant challenges it presents in deploying countermeasures, a characteristic it shares with the binary PC oracle-based attack. In [RRD+23], Rajendran et al. use an improved method to construct ciphertexts, allowing the recovery of an arbitrary P number of bits of information in a single query. They applied this attack to the Kyber KEM, investigating various attack scenarios such as the existence of a clone device. The results showed a 2.89 to 7.65 times acceleration compared to binary PC oracle-based attacks. Another work proposed by Tanaka et al. uses a similar method but adopts a neural network for the multi-classification of side-channel leakage [TUX+23]. They reduce attack queries for dependable key recovery by as much as 87%, compared to existing binary PC oracle-based attacks against Kyber and other lattice-based KEMs.

Robust recovery in practical settings. A significant consideration in implementing these SCAs in real-world scenarios is the presence of environmental noise or inaccuracies in the waveform classifier. These factors potentially introduce errors into the oracle responses, compromising the integrity of the recovered secret key. To address this issue, a widely adopted method is majority voting. For example, in [RRCB20], Ravi et al. employ a strategy of three-vote majority voting for each oracle response, resulting in three times the total queries compared with its theoretically estimated sample complexity. A more advanced approach is to use techniques like negative-log likelihood (NLL) to improve the accuracy of the oracle, especially when the waveform classifier is constructed using deep learning [ISUH21, UXT+22, TUX+23].

In these works, a practical key recovery is initiated by first establishing a reliable or near-perfect oracle, operating under the assumption of an ideal environment. At TCHES 2023, Shen et al. introduce an alternative method for real-world binary PC attack [SCZ+23], conducting key recovery with a newly proposed error detection method. The fundamental idea involves generating and querying special ciphertexts to detect the positions of error coefficients. Notably, their method reduces 45.9% to 55.4% of total number of traces compared with the attack in [UXT+22] against Kyber512.

However, how to efficiently deal with errors in the responses of MV-PC oracle is still unsolved. The performance of using NLL, as described in [TUX+23], is superior to that of majority voting. But it typically leads to three times the number of queries compared to the ideal scenario in which no errors occur. A direct observation is that: Can we integrate the error detection method into the MV-PC oracle to further reduce the number of queries? We indicate that such a combination is non-trivial and challenging. This is because the ciphertext generation method for parallel key recovery is not compatible with Shen et al.'s approach. In this paper, we aim to propose a new approach to efficiently locating and correcting error coefficients in the MV-PC oracle, especially when the channel is not perfect. The primary objective of our approach is to facilitate parallel key recovery while substantially minimizing the number of queries needed.

1.2 Contributions

The principal contributions of this study are:

- We propose an efficient MV-PC oracle-based side-channel attack against Kyber,

designed for an imperfect oracle. Our innovative approach involves a novel method for efficient detection and correction of errors in the MV-PC oracle. There are 3 rounds in our improved method. In round 1, a parallel key recovery is executed. Our key observation is that the coefficients recovered in the first round can be regarded as prior information to facilitate key recovery in the second round. As a result, in the second round, we can considerably decrease the number of queries required through a newly created grafted tree. The grafted tree is designed to enable retrieval of each coefficient in nearly two queries. Compared to the latest attack [TUX+23][RRD+23] at TCHES 2023, our proposed attack reduces the number of queries for a full key recovery by more than 42.5%.

- We center our research on Kyber, recognized as the primary future NIST KEM standard. Notably, our method is generic: It is also applicable to similar schemes, such as Saber and Frodo, two LWE/LWR-based candidates from the second/third rounds of the NIST PQC competition. Our extensive experiments cover all security levels of Kyber (Kyber-512, Kyber-768, Kyber-1024), FireSaber, and Frodo-1344. We base the MV-PC oracle on a multiple-classification neural network model. Further, we validate our proposed method in real-world scenarios by conducting an electromagnetic attack on an STM32F407G board equipped with an ARM Cortex-M4 microcontroller and Kyber implementation from the publicly available testing and benchmarking library *pqm4*. The results from the real-world implementation align well with our prior simulations.

- In high-noise environments, we have enhanced our method to fit such scenarios. Further analysis reveals that compared to majority voting, our method can decrease the number of queries by 29.4% and 22.8% at an accuracy of 95% and 90%, respectively. This underscores the versatility and effectiveness of our approach.

1.3 Organizations

We present the background of Kyber and MV-PC oracle in Section 2. Our new method and the analysis of our attacks are presented in Section 3 and Section 4, respectively. Finally, we conclude our paper in Section 5.

2 Background

2.1 Kyber

Kyber operates over the polynomial ring $\mathcal{R}_q = \mathbb{Z}_q[x]/(x^n + 1)$, where $q = 3329$ is a prime modulo, and $n = 256$. Each coefficient a_i of the polynomial $f(x) = a_0 + a_1x + ... + a_{n-1}x^{n-1} \in \mathcal{R}_q$ belongs to $\mathbb{Z}_q$ for $0 \leq i \leq n-1$, with $\mathbb{Z}_q$ representing the ring of integers modulo q. All polynomial operations, including additions and multiplications, are performed modulo $x^n + 1$. We use a bold lower-case letter $\mathbf{a} \in \mathcal{R}_q$ and its vector form $(\mathbf{a}[0], \ldots, \mathbf{a}[n-1])$ interchangeably to represent a polynomial.

The security of Kyber is based on the M-LWE problem. In linear algebra, solving for $\mathbf{s}$ in the equation $\mathbf{b} = \mathbf{As}$ is a straightforward process. Here, $\mathbf{A}$ is a matrix, and both $\mathbf{b}$ and $\mathbf{s}$ are vectors. However, the LWE problem indicates that the addition of even small coefficients of noise can render the recovery of $\mathbf{b} = \mathbf{As} + \mathbf{e}$ challenging. As a variant of the LWE problem, the M-LWE problem is to distinguish $(\mathbf{A}, \mathbf{B} = \mathbf{As} + \mathbf{e}) \in \mathcal{R}_q^{l \times l} \times \mathcal{R}_q^l$ from uniformly selected $(\mathbf{A}, \mathbf{B}) \in \mathcal{R}_q^{l \times l} \times \mathcal{R}_q^l$. The parameter l is set to be 2, 3, or 4, which corresponds to three different security levels in Kyber: Kyber-512, Kyber-768, and Kyber-1024, respectively.

In Kyber, all the secret key coefficients and error vectors are sampled from a centered binomial distribution $\mathcal{B}_\eta$, where $\mathcal{B}_\eta$ represents the centered binomial distribution with parameter η. They can be generated by $\sum_{i=1}^{\eta}(a_i - b_i)$. Here a_i and b_i are uniformly random samples independently selected from $\{0, 1\}$. The value of η is different in three security levels, i.e., $\eta = 3$ in Kyber 512 and $\eta = 2$ in Kyber 768 and Kyber 1024.

Algorithm 1 KYBER.CCA KEM

KYBER.KEM.KeyGen

Output: sk', pk

Generate a pseudo-random coin z
$\circ$ **KYBER.CPA.KeyGen**
  $\mathbf{A} \xleftarrow{\$} \mathcal{R}_q^{l\times l}$, $\mathbf{s}, \mathbf{e} \xleftarrow{\$} \mathcal{B}_{\eta_1}^l$
  $\mathbf{b} := \mathbf{A}^T\mathbf{s} + \mathbf{e}$
  $sk = \mathbf{s}$, $pk := (\mathbf{A}, \mathbf{b})$
$sk' := (sk||pk||\mathcal{H}(pk)||z)$

KYBER.CCA.Encaps

Input: pk

Output: ct, K

$\mathbf{m} \xleftarrow{\$} \{0,1\}^{256}$
$(\bar{K}, r) = \mathcal{G}(\mathbf{m}, \mathcal{H}(pk))$
$\circ$ **KYBER.CPA.Enc**$(pk, \mathbf{m}, r)$
  $\mathbf{r}, \mathbf{e}_1, \mathbf{e}_2 \xleftarrow{\$} \mathcal{B}_{\eta_2}^l$
  $\mathbf{u} = \mathbf{A}^T\mathbf{r} + \mathbf{e}_1$
  $\mathbf{v} = \mathbf{b}^T\mathbf{r} + \mathbf{e}_2 + \mathrm{DeComp}_q(\mathbf{m}, 1)$
  $\mathbf{c}_1 := \mathrm{Comp}_q(\mathbf{u}, d_u)$
  $\mathbf{c}_2 := \mathrm{Comp}_q(\mathbf{v}, d_v)$
  $ct := (\mathbf{c}_1, \mathbf{c}_2)$
$K := \mathrm{KDF}(\bar{K}, \mathcal{H}(ct))$

KYBER.CCA.Decaps

Input: sk', ct

Output: K

$\circ$ **KYBER.CPA.Dec**(sk, ct)
  $\mathbf{u}' = \mathrm{DeComp}_q(\mathbf{c}_1, d_u)$
  $\mathbf{v}' = \mathrm{DeComp}_q(\mathbf{c}_2, d_v)$
  $\mathbf{m}' = \mathrm{Comp}_q(\mathbf{v}' - \mathbf{s}^T\mathbf{u}', 1)$
$(\bar{K}', r') = \mathcal{G}(\mathbf{m}', \mathcal{H}(pk))$
$ct' =$ **KYBER.CPA.Enc**$(pk, \mathbf{m}', r')$
if $ct = ct'$ **then**
  $K := \mathrm{KDF}(\bar{K}', \mathcal{H}(ct))$
else
  $K := \mathrm{KDF}(z, \mathcal{H}(ct))$
end if

Let$\lfloor x \rfloor$ denote the maximum integer not exceeding x, and $\lceil x \rfloor$ is the rounding function, i.e. $\lceil x \rfloor = \lfloor x + \frac{1}{2} \rfloor$. In the following, we first give the definitions of two functions: $\mathbf{Comp}_q(x, d)$ and $\mathbf{DeComp}_q(x, d)$.

Definition 1. The **Comp** function: $\mathbb{Z}_q \to \mathbb{Z}_{2^d}$:

$$\mathbf{Comp}_q(x, d) = \lceil (2^d/q) \cdot x \rfloor \pmod{2^d}. \tag{1}$$

Definition 2. The **DeComp** function: $\mathbb{Z}_{2^d} \to \mathbb{Z}_q$:

$$\mathbf{DeComp}_q(x, d) = \lceil (q/2^d) \cdot x \rfloor . \tag{2}$$

In $\mathbf{Comp}_q(x, d)$ and $\mathbf{DeComp}_q(x, d)$ the input x is selected from $\mathbb{Z}_q$. Likewise, when the input is a polynomial, the above operation will be performed separately for each coefficient.

Similarly, in $\mathbf{Comp}_q(x, d)$ and $\mathbf{DeComp}_q(x, d)$, the value of d is set as d_u or d_v for the different security levels of Kyber. In Table 1, we summarize the main parameters.

Typically, a KEM consists of three parts: key generation, encapsulation, and decapsulation. The CCA-secure Kyber KEM is constructed from a simple CPA-secure PKE, which can be divided into KYBER.CPA.KeyGen, KYBER.CPA.Enc and KYBER.CPA.Dec. The process of transforming a CPA-secured PKE into a CCA-secure KEM is based on Fujisaki-Okamoto transform and its variants. Two hash functions $\mathcal{G}(\cdot)$ and $\mathcal{H}(\cdot)$ are used in the encapsulation and decapsulation process, and $\mathrm{KDF}(\cdot)$ denotes a key-derivation

Table 1: Parameter sets of Kyber

Schemes	n	d	q	η	$d_{\mathbf{u}}$	d_v
Kyber-512	256	2	3329	3	10	4
Kyber-768	256	3	3329	2	10	4
Kyber-1024	256	4	3329	2	11	5

function. The main parts of Kyber encapsulation and decapsulation, ignoring some details such as number-theoretic transformations (NTTs), are depicted in Algorithm 1.

The following discusses the PC oracle and the corresponding MV-PC oracle.

2.2 From PC oracle to MV-PC oracle

2.2.1 PC oracle

Algorithm 2 PC oracle and the corresponding attack

PC oracle $\mathcal{O}$
Input: $ct, \mathbf{m}$
Output: 0 or 1
1: $\mathbf{m}' \leftarrow \mathbf{KYBER.CPA.Dec}(sk, ct))$
2: **if** $\mathbf{m} = \mathbf{m}'$ **then**
3: **Return** 1
4: **else**
5: **Return** 0
6: **end if**

PC oracle based attack
Input: PC oracle $\mathcal{O}$
Output: Secret $\mathbf{s}$
1: **for** $i = 0 \rightarrow l - 1$ **do**
2: **for** $j = 0 \rightarrow n - 1$ **do**
3: Set response sequence $seq \leftarrow$ “”
4: **while** $\mathbf{s}_i[j]$ cannot be recovered from seq **do**
5: Generate ct according to seq
6: Append $\mathcal{O}(ct, \mathbf{m})$ to seq
7: **end while**
8: Set $\mathbf{s}_i[j]$ according to seq
9: **end for**
10: **end for**
11: **Return s**

In this paper, we take into consideration a chosen-ciphertext attack against FO transformation with the help of side-channel leakages, which can be modeled as the well-known PC oracle-based attack [RRCB20]. In a PC oracle, there is an honest user Alice who implements Kyber on her devices for key establishment. An adversary Eve acts as a valid user Bob to negotiate shared keys with Alice. The adversary sends a series of well-chosen ciphertexts to Alice and exploits side-channel leakage to help recover Alice's secret key sk. To be specific, in a PC oracle, for chosen ciphertext ct and message $\mathbf{m}$, the oracle tells whether the decrypted $\mathbf{m}'$ (line 4 in **KYBER.CCA.Encaps**) equals $\mathbf{m}$ or not. This process is depicted in Algorithm 2. Since $\mathbf{m}$ and $\mathbf{m}'$ determine the final key K, we can also know whether the keys generated by the two parties match or not. Hence, a PC oracle-based attack is also called the key mismatch attack [QCZ+21].

In the following, we take Kyber-1024 as an example to introduce the PC oracle-based attack. Assume Alice's secret key is $\mathbf{s} = (\mathbf{s}_0, \mathbf{s}_1)$. At first Eve sets reference plaintext $\mathbf{m} = (1, 0, \ldots, 0)$, $\mathbf{u} = (\lceil q/32 \rfloor, 0, \cdots, 0)$ and $\mathbf{c}_2 = (k, 0, \ldots, 0)$, where $k \in \mathbb{Z}_q$ is a parameter used to help recover the secret key. Then, Eve computes $\mathbf{c}_1 = \mathbf{Comp}_q(\mathbf{u}, d_u)$ and packs $\mathbf{c}_1$ and $\mathbf{c}_2$ into ct and sends it to Alice. With $(\mathbf{c}_1, \mathbf{c}_2)$, Alice computes $\mathbf{u}' = \mathbf{DeComp}_q(\mathbf{c}_1, d_u)$ and $\mathbf{v}' = \mathbf{DeComp}_q(\mathbf{c}_2, d_v) = (\lceil (q/32)k \rfloor, 0, \cdots, 0)$. Alice goes on calculating $\mathbf{m}'$, and

the relationship between $\mathbf{m}'[0]$ and $\mathbf{s}_0[0]$ can be built as follows:

$$\begin{aligned}\mathbf{m}'[0] &= \mathbf{Comp}_q((\mathbf{v}-\mathbf{s}_0^T\mathbf{u})[0],1) && (3)\\ &= \lceil (2/q)\left(\mathbf{v}[0]-(\mathbf{s}_0^T\mathbf{u})[0]\right)\rfloor \bmod 2 && (4)\\ &= \lceil (2/q)(\mathbf{v}[0]-\mathbf{s}_0[0]\mathbf{u}[0])\rfloor \bmod 2 && (5)\\ &= \lceil (2/q)\left(\lceil (q/32)k\rfloor - \mathbf{s}_0[0]\lceil q/32\rfloor\right)\rfloor \bmod 2. && (6)\end{aligned}$$

We can see that the value of $\mathbf{m}'[0]$ relies only on k and $\mathbf{s}_0[0]$. Similarly, for $i \geq 1$, $\mathbf{m}'[i] = \lceil (2/q)\,(0-\mathbf{s}_0[i]\lceil q/32\rfloor)\rfloor \bmod 2$. Recall that in Kyber-1024, coefficients are chosen from interval [-2, 2], $\mathbf{s}_0[i]$ at most takes the value 2, we have

$$(2/q)\,(\mathbf{s}_0[i]\lceil q/32\rfloor) \approx 0.125 < 1/2. \tag{7}$$

That is, for $i \geq 1$, $\mathbf{m}'[i] = 0$. Hence, whether $\mathbf{m} = \mathbf{m}'$ is totally determined by the value of $\mathbf{m}'[0]$. To be specific, $\mathbf{m} = \mathbf{m}'$ (the oracle outputs 1) if and only if $\mathbf{m}'[0] = 1$. By choosing different k and inputting the corresponding ct into the oracle, we can get a sequence seq consisting of the corresponding outputs of the oracle. This sequence can directly correspond to a coefficient value.

Table 2: The value of $\mathbf{m}'[0]$ corresponding to different k and $\mathbf{s}_0[0]$

	$\mathbf{s}_0[0]=-2$	$\mathbf{s}_0[0]=-1$	$\mathbf{s}_0[0]=0$	$\mathbf{s}_0[0]=1$	$\mathbf{s}_0[0]=2$
$k=7$	1	0	0	0	0
$k=8$	1	1	0	0	0
$k=9$	1	1	1	0	0
$k=10$	1	1	1	1	0

As indicated in Table 2, for each coefficient $\mathbf{s}_0[0]$ within the range $[-2, 2]$, we can pre-calculate the relationship between k and $\mathbf{m}'[0]$. Subsequently, by constructing a specialized ciphertext ct and making multiple queries to the oracle, the adversary can progressively narrow down the possible values for $\mathbf{s}_0[0]$, eventually obtaining its precise value. Within an iteration, $\mathbf{u}$ remains fixed, and the value of k is adjusted based on the responses received from the PC oracle. Consequently, if k is appropriately chosen, as illustrated in Table 3, the attacker could efficiently recover $\mathbf{s}_0[0]$ with a minimal number of queries.

Table 3: Selection of parameters and the corresponding States

	State 1 $k=8$	State 2 $k=9$	State 3 $k=10$	State 4 $k=7$
$\mathcal{O}\to 0$	State 2	State 3	$\mathbf{s_0}[0]=2$	$\mathbf{s_0}[0]=-1$
$\mathcal{O}\to 1$	State 4	$\mathbf{s_0}[0]=0$	$\mathbf{s_0}[0]=1$	$\mathbf{s_0}[0]=-2$

2.2.2 BRT for PC oracle

In Table 3, we present an illustration of adaptive ciphertext selection, a process derived from [QCZ+21]. In their work, Qin et al. introduced the concept of the binary recovery tree (BRT), which serves as a common framework for explaining both binary PC oracle and MV-PC oracle concepts. The BRT serves as a visual representation of the adaptive selection of ciphertext based on oracle responses. It is structured as a rooted binary tree comprising a single root node and n-leaf nodes, where each leaf node corresponds to an element from the set of coefficients of Kyber. For Kyber-1024 and Kyber-768 the set is {-2, -1, 0, 1, 2} and for Kyber-512 the set is {-3, -2, -1, 0, 1, 2, 3}. For each node with children,

denote by 1 its left child and by 0 its right child, where 0 and 1 correspond to the response of the oracle. Here, in the context of a BRT, the depth represents the number of steps or hops from a leaf node to the root node. It signifies the cumulative number of queries made during the process. The BRT is constructed with the help of Huffman coding, i.e., the higher the probability of occurrence of a secret coefficient, the fewer number of queries (lower depth in the BRT) it will require to uniquely distinguish it.

For PC oracle-based attack against Kyber-1024, the BRT on the left side of Figure 1 aligns with the information presented in Table 3. Note that the BRT for Kyber-1024 is not uniquely determined, and we can find two such trees with the same number of queries.

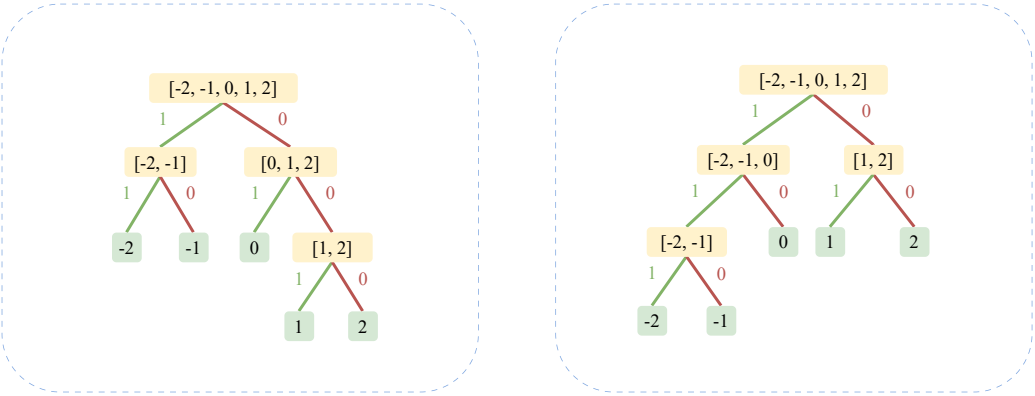


Figure 1: BRTs in PC oracle-based attack against Kyber-1024

2.2.3 MV-PC oracle

The PC oracle demands an extensive number of traces due to the limited information an attacker obtains with each query. Specifically, only one bit of information is garnered per query to the oracle. To further reduce the traces, Rajendran et al. [RRD+23] and Tanaka et al. [TUX+23] have proposed the concept of parallel PC oracle or MV-PC oracle, independently. In an MV-PC oracle, it is possible to recover a set of N bits of information about the secret key in a single query.

The core idea of MV-PC oracle is to construct special ciphertexts such that the first N bits of $\mathbf{m}'$ depend on the N corresponding coefficients of the secret key. Specifically, Eve selects $\mathbf{u} = (\lceil q/32 \rfloor, 0, \cdots, 0)$, and $\mathbf{c}_2 = (k_0, k_1, \cdots, k_{N-1}, 0, \cdots, 0)$, then Eve compresses $\mathbf{c}_1 = \mathbf{Comp}_q(\mathbf{u}, d_u)$ and sends $ct := (\mathbf{c}_1, \mathbf{c}_2)$ to Alice. With received ct, Alice decompresses $\mathbf{u} = \mathbf{DeComp}_q(\mathbf{c}_1, d_u)$ and $\mathbf{v} = \mathbf{DeComp}_q(\mathbf{c}_2, d_v)$. Similar to the analysis in (6) and (7), Alice decrypts and gets:

$$\mathbf{m}'[i] = \begin{cases} \lceil (2/q) \left(\lceil (q/32) k_i \rfloor - \mathbf{s}_0[i] \lceil q/32 \rfloor \right) \rfloor & \mod 2, \quad i = 0, \ldots, N-1, \\ \lceil (2/q) \left(0 - \mathbf{s}_0[i] \lceil q/32 \rfloor \right) \rfloor & \mod 2 = 0, \quad i \geq N. \end{cases} \tag{8}$$

With the parallelization factor N, the ciphertext ct can be decrypted to one of the 2^N

known plaintexts, which can be expressed as follows:

$$\mathbf{m}_0 = (\underbrace{0,0,0,\cdots,0}_{N\ bits}\ \underbrace{0,0,0,\cdots,0}_{256-N\ bits})$$
$$\mathbf{m}_1 = (\underbrace{1,0,0,\cdots,0}_{N\ bits}\underbrace{0,0,0,\cdots,0}_{256-N\ bits})$$
$$\mathbf{m}_2 = (\underbrace{0,1,0,\cdots,0}_{N\ bits}\ \underbrace{0,0,0,\cdots,0}_{256-N\ bits})$$
$$\cdots$$
$$\mathbf{m}_{2^N-1} = (\underbrace{1,1,1,\cdots,1}_{N\ bits}\ \underbrace{0,0,0,\cdots,0}_{256-N\ bits})$$

Similarly, we need to adjust our strategy to employ a parallel approach in an SCA-assisted chosen ciphertext attack. First, the waveforms generated during the decryption process are collected to train a multiple-valued classifier. We use $\mathcal{W}_0,\cdots,\mathcal{W}_{2^N-1}$ to denote different waveforms corresponding to $\mathbf{m}_0,\cdots,\mathbf{m}_{2^N-1}$, respectively.

Then, for each attack, the waveform $\mathcal{W}$ generated in the hash function $\mathcal{G}(\cdot)$ is collected and classified as one of $\mathcal{W}_0,\cdots,\mathcal{W}_{2^N-1}$. The formula $\mathcal{W}=\mathcal{W}_i$ means that $\mathcal{W}$ is classified to $\mathcal{W}_i$ and the attacker will be able to derive $\mathbf{m}'$ generated in **KYBER.CCA.Decaps**, which is equal to $\mathbf{m}_i$. The specific approach is described in Algorithm 3.

Algorithm 3 MV-PC oracle and the corresponding attack

MV-PC oracle $\mathcal{O}_{mv}$
Input: Ciphertext ct
Input: Reference plaintexts $\mathbf{m}$
Output: $\mathbf{m}'$
1: $\mathcal{W} \leftarrow$ SCA(KYBER.CCA.Decaps(sk', ct))
2: **for** i= 0 to 2^{N-1} **do**
3: **if** $\mathcal{W}=\mathcal{W}_i$ **then**
4: $\mathbf{m}'=\mathbf{m}_i$
5: **Return** $\mathbf{m}'$
6: **end if**
7: **end for**

MV-PCAttack
Input: MV-PC oracle $\mathcal{O}_{mv}$
Output: Secret key $\mathbf{s}$
1: **for** $i=0\rightarrow l-1$ **do**
2: **for** $j=0\rightarrow(256/N)-1$ **do**
3: Set response sequence $seq\leftarrow$ “”
4: **while** block $\mathbf{s}_i[j*N],\cdots,\mathbf{s}_i[j*N+N-1]$ cannot be fully recovered from seq **do**
5: Generate ct according to seq
6: Append $\mathcal{O}_{mv}(ct,\mathbf{m})$ to seq
7: **end while**
8: Set the values of the block according to seq
9: **end for**
10: **end for**
11: **Return** Recovered $\mathbf{s}$

2.2.4 BRT for MV-PC oracle

In an MV-PC oracle, the BRT used in the PC oracle may not be the optimal choice. This is because, in this scenario, we aim to recover N coefficients simultaneously, and the number of queries needed depends on the coefficient with the maximum depth. Here, depth refers to the position of the leaf node corresponding to the coefficient in the BRT.

For a set of N-tuple coefficients $\{s_0,s_1,s_2,\cdots,s_{N-1}\}$, if all the coefficients belong to $\{0,1,2\}$, the MV-PC oracle-based parallel recovery can be completed in two queries. However, if there are coefficients belonging to $\{-1,-2\}$, the parallel recovery for this N-tuple must be accomplished in 3 queries. For Kyber512, Rajendran et al. have proposed to adopt a more balanced BRT. As illustrated in Figure 2, the maximum depth on the left BRT is 4, while on the right, the maximum depth has decreased to 3. But this adjustment is only valid for Kyber-512. For Kyber-1024, and Kyber-768, the BRT in Figure. 1 cannot be more balanced.

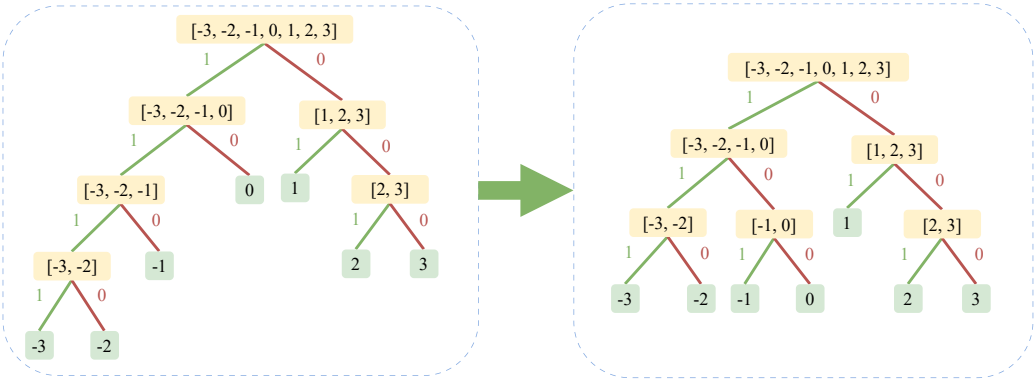


Figure 2: For Kyber-512, the optimal BRT in PC oracle (left) and a more balanced BRT for MV-PC oracle (right).

3 Challenges and our new method

3.1 Challenges from noises and our observations

Recall that both the PC oracle and MV-PC oracle are instantiated with side-channel attacks. In practice, the accuracy of the oracle may be influenced by environmental noises and the need to protect against side-channel attacks, such as shuffling and masking. Furthermore, the performance of the classifier itself can also impact accuracy, especially for the MV-PC oracle, since 2^N (where $N > 1$) classifications are more complex than binary classification, which can lead to lower accuracy. For example, in [TUX+23], they compare the results using NLL technique with a 9-round majority voting in their attack when $N > 6$. Even with NLL, the resulting number of queries usually reaches three times the number of that in the ideal case.

When the retrieved results contain errors, the traditional approach is to use majority voting, which leverages multiple inference results to improve the accuracy of the classifier further. For an oracle with accuracy α, after t rounds of majority voting, the accuracy becomes

$$\alpha' = 1 - \sum_{s=0}^{\lfloor t/2 \rfloor} \binom{t}{s} \alpha^s (1-\alpha)^{t-s}. \tag{9}$$

Instead of majority voting, Shen et al. treat the detection of errors as a coding problem and propose an efficient method called fast-checking to identify error locations [SCZ+23]. For the PC oracle, an attacker with two accesses to the oracle can employ fast-checking to verify the accuracy of secret key coefficients at four locations. To be specific, there are three steps in [SCZ+23]. In Step 1, a roughly correct key is retrieved. Then, in Step 2 the attacker selects some attack parameters to detect errors in the retrieved key coefficients. For example, to check whether the retrieved coefficient block $\mathbf{s_0}[0], \mathbf{s_0}[1], \mathbf{s_0}[2], \mathbf{s_0}[3]$ are correct or not, $\mathbf{u} = \mathbf{u_{atk}}[0] - \mathbf{u_{atk}}[1]x^{255} - \mathbf{u_{atk}}[2]x^{254} - \mathbf{u_{atk}}[3]x^{253}$ and $\mathbf{v} = \mathrm{v_{atk}}$ are generated from the values of $\mathbf{s_0}[0], \mathbf{s_0}[1], \mathbf{s_0}[2], \mathbf{s_0}[3]$. Here, the selection of attack parameters $\mathbf{u_{atk}}[0], \mathbf{u_{atk}}[1], \mathbf{u_{atk}}[2], \mathbf{u_{atk}}[3], \mathrm{v_{atk}}$ ensures that if the recovered coefficients $\mathbf{s_0}[0], \mathbf{s_0}[1], \mathbf{s_0}[2], \mathbf{s_0}[3]$ are correct, the resulted codeword is special. So an interesting question arises here: Can we adapt fast-checking to the MV-PC oracle? In fact, this is challenging. The challenge stems from conflicts in the selection of attack parameters. In MV-PC oracle, if we want to check two different coefficient blocks $\mathbf{s_0}[0], \mathbf{s_0}[1], \mathbf{s_0}[2], \mathbf{s_0}[3]$ and $\mathbf{s_0}[4], \mathbf{s_0}[5], \mathbf{s_0}[6], \mathbf{s_0}[7]$ at the same time, we need to set $\mathbf{u} = \mathbf{u_{atk}}[0] - \mathbf{u_{atk}}[1]x^{255} - \mathbf{u_{atk}}[2]x^{254} - \mathbf{u_{atk}}[3]x^{253}$ and $\mathbf{v} = \mathrm{v_{atk}}$ and gets 1 bit of

information from $\mathbf{m}'[0]$:

$$\mathbf{m}'[0] = \left\lceil \frac{2}{q}(\mathrm{v}_{\mathrm{atk}} - (\sum_{n=0}^{3} \mathbf{s_0}[n]\mathbf{u_{atk}}[n])) \right\rfloor \bmod 2. \tag{10}$$

Meanwhile, we need to set $\mathbf{u} = \mathbf{u'_{atk}}[0] - \mathbf{u'_{atk}}[1]x^{255} - \mathbf{u'_{atk}}[2]x^{254} - \mathbf{u'_{atk}}[3]x^{253}$ and $\mathbf{v} = \mathrm{v}'_{\mathrm{atk}}$ to get another 1 bit of information from $\mathbf{m}'[4]$:

$$\mathbf{m}'[4] = \left\lceil \frac{2}{q}(\mathrm{v}'_{\mathrm{atk}} - (\sum_{n=0}^{3} \mathbf{s_0}[n+4]\mathbf{u'_{atk}}[n])) \right\rfloor \bmod 2. \tag{11}$$

However, this incurs conflicts in the setting of $\mathbf{u}$ for two blocks with different coefficients.

An obvious question is whether, in the parallel case (where the attacker can get multiple bits of information), a more efficient method could be found – potentially by better leveraging prior knowledge – than the one proposed by [SCZ+23] to expedite the process.

3.2 Our basic idea

In this part, we describe the general full-key recovery framework of the new attack. We start by introducing the basic ideas.

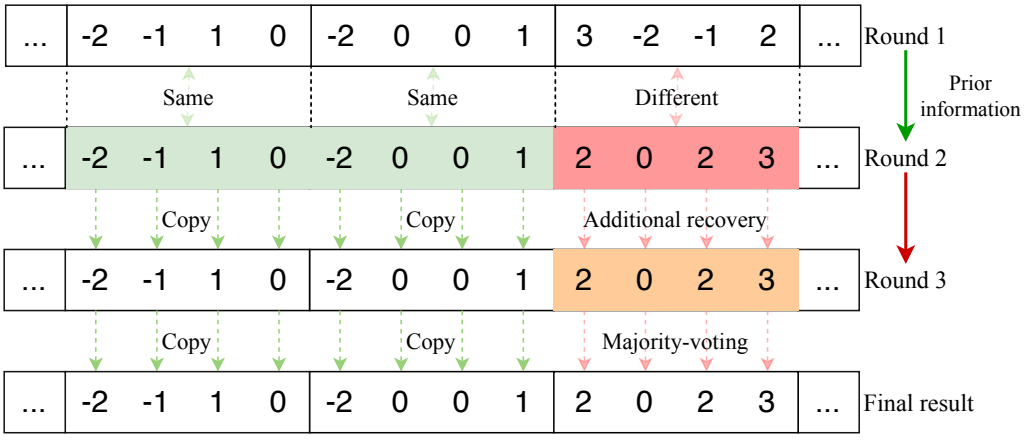


Figure 3: Our main idea

In Figure 3, we illustrate the main idea of our key recovery strategy. In majority voting, we need to repeat the same key recovery procedure several times. Our key observation is that the coefficients recovered in the previous round can be regarded as prior information to facilitate the recovery of coefficients in the subsequent round. Consequently, in our strategy, after employing an MV-PC oracle to recover the initial round coefficients, we establish a grafted tree to guide the recovery of the second round coefficients. This approach enables us to achieve significant reductions in the number of required queries.

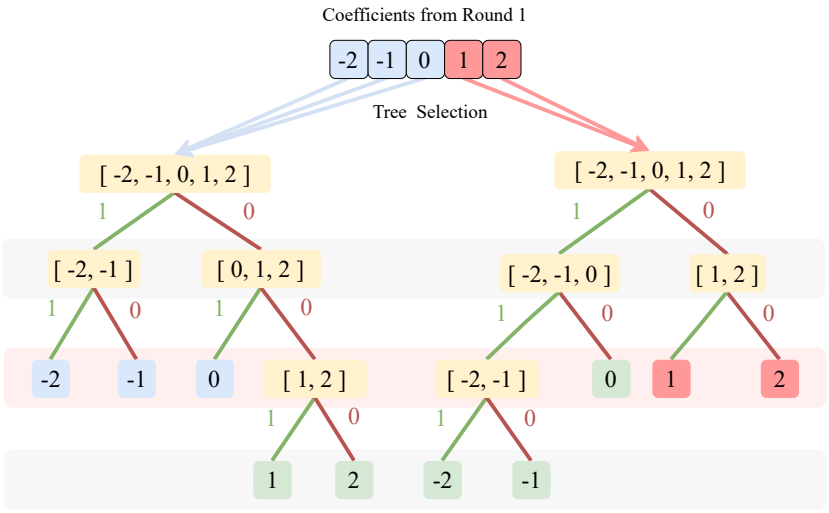


Figure 4: A grafted tree for Kyber-1024 and Kyber-768, including a BRT for coefficients -2, -1, 0 (left) and a BRT for coefficients 1, 2 (right)

We summarize the main process as follows:

- Round 1: Parallel key recovery is executed to obtain approximately correct secret key coefficients using MV-PC oracle.
- Round 2: After the parallel key recovery, with the prior information acquired in round 1, we construct the corresponding grafted tree and perform another parallel key recovery with the grafted tree. We compare each coefficient block recovered in round 2 with the corresponding coefficient block in round 2. If a recovered coefficient block exhibits a value different from that in the round 1 recovery, the corresponding coefficient block is labeled as suspicious.
- Round 3: If the retrieved coefficient blocks match in round 1 and round 2, we directly classify these blocks as correctly recovered. For the identified suspicious coefficient blocks, an additional parallel recovery is conducted on these blocks. Subsequently, the coefficient values obtained from the three rounds undergo a majority voting process. The result of the majority voting for coefficients is considered the correct outcome.

In the next subsection, we elaborate on how prior information is employed to construct a grafted binary recovery tree (grafted tree).

3.3 The grafted tree for key recovery

The grafted tree. Our key observation is that, with prior information about which coefficient has been retrieved in round 1, we can choose different branches in different BRTs to further reduce the needed number of queries. We call the new constructed the grafted tree. More specifically, taking Kyber-1024 as an example, for coefficients recovered as [-2, 0] in round 1, we employ the left and right branches of the left BRT in Figure 1; While for [1, 2], the right branch of the right BRT in Figure 1 is leveraged. In Figure 4, it can be seen that each coefficient retrieved in round 1 has depth of 2.

Here a question arises, for each coefficient, can we find a BRT in which the depth of the corresponding leaf node has the depth of 2? We summarize the results in the following.

Lemma 1. *In Kyber, for each coefficient, we can find a BRT in which the depth of the corresponding leaf node has depth* 2*.*

Proof. The problem is equivalent to proving that for each coefficient, we can distinguish it from other coefficients by accessing the oracle twice. We summarize all the results for Kyber in Table 4, including Kyber-1024, Kyber-768, and Kyber-512. In Table 4, $k1, k2$ denotes the value of k in the ciphertext constructed during the first and second access to the oracle, respectively. 2(11) means the retrieved coefficient is 2, and 11 is the output sequence of the oracle. From the result in Table 4, we conclude that for each coefficient, it is always possible to construct a BRT in which the leaf node corresponding to this coefficient has a depth of 2. □

Table 4: Distinguishing coefficients by ciphertext pairs

(a) In Kyber-1024

(k_1, k_2)	Distinguished coefficients	Others
(7,8)	-2(11), -1(01)	{0, 1, 2}(00)
(8,9)	0(01)	{-1, -2}(11), {1, 2}(00)
(9,10)	1(01), 2(00)	{0, 1, 2}(11)

(b) In Kyber-768

(k_1, k_2)	Distinguished coefficients	Others
(3,4)	-2(11), -1(01)	{0, 1, 2}(00)
(4,5)	0(01)	{-1, -2}(11), {1, 2}(00)
(5,6)	1(01), 2(00)	{-0, -1, -2}(11)

(c) In Kyber-512

(k_1, k_2)	Distinguished coefficients	Others
(2,3)	-3(11), -2(01)	{-1, 0, 1, 2, 3}(00)
(3,4)	-1(01)	{-3, -2}(11), {0, 1, 2, 3}(00)
(4,5)	0(01)	{-3, -2, -1}(11), {1, 2, 3}(00)
(5,6)	1(01)	{-3, -2, -1, 0}(11), {2, 3}(00)
(6,7)	2(01), 3(00)	{-3, -2, -1, 0, 1}(11)

In contrast to previous attack methods, in round 2, our approach enables the selection of distinct BRTs. This flexibility arises from our knowledge of the approximate correct coefficient values, which serve as valuable prior information when constructing the grafted tree. This enables us to utilize the optimal BRT for each coefficient, accelerating our attack and reducing overall attack overheads.

The remaining problem is how to construct a grafted tree. We treat the generation of a grafted tree as an expansion of the BRT generation process. Certain principles need to be followed: For each coefficient, position this coefficient as low as possible in the tree, without regard for the positions of the leaf nodes where the other coefficients are situated. There is no need to construct a large number of BRTs during preparation, as coefficients only need to be placed on the second level of a binary tree. In Kyber-1024 and Kyber-768, only 2 BRTs are needed to form a grafted tree, instead of 5, as shown in Figure 4. For Kyber-512 we need only 4 BRTs. Trees for Kyber-512 are included in Figure 5.

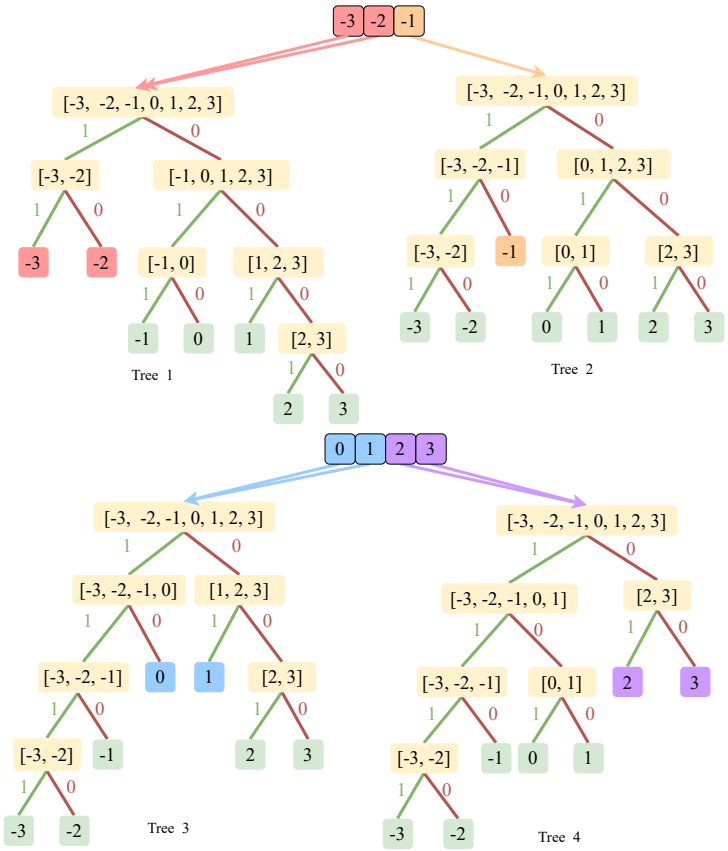


Figure 5: A grafted tree for Kyber-512, including a BRT for coefficients -3, -2 (left above) and -1 (right above), and a BRT for coefficients 0, 1 (below left) and 2, 3 (below right)

For the trees that have been selected, we store them in a collection of trees called **Trees**. When Eve selects $\mathbf{u} = (\lceil q/32 \rfloor, 0, \cdots, 0)$, and $\mathbf{c}_2 = (k_0, k_1, \cdots, k_{N-1}, 0, \cdots, 0)$. The value of k is determined by the coefficient value of the previous stage of recovery and the detailed process is shown in Algorithm 4.

Next, we briefly introduce how to recover a block with coefficients $(1, -2, 2, 0)$ retrieved from round 1 using a grafted tree in Kyber-1024.

1. In round 2, from Table 4, for coefficients 1 and 2, we need to set $k_1 = 9$; For -2 and -1, we set $k_1 = 7$; For -0, we let $k_1 = 8$. Hence, we first set $\mathbf{c}_2 = (9, 8, 9, 8, 0, \cdots, 0)$. Subsequently, we access the oracle, obtaining a response sequence $\mathbf{m}'_1$.

2. Then, from Table 4 again, we go on setting $\mathbf{c}_2 = (10, 7, 10, 9, 0, \cdots, 0)$ and access the oracle to get another response sequence $\mathbf{m}'_2$.

3. If $\mathbf{m}'_1 = 0100$ and $\mathbf{m}'_2 = 1101$, from Table 4, 01, 11, 00, 01 correspond to 1, -2, 2, 0, respectively. Thus, we know the retrieved block in round 2 is also $(1, -2, 2, 0)$, matching the result in round 1. If not, the results in the two rounds are different and we need a third round.

Algorithm 4 Constructing the grafted tree and recovery in rounds 1, 2, 3

Recovery in round 1
Input: An imperfect MV-PC oracle $\mathcal{O}_{mv}$
Output: Roughly correct secret key coefficients $\hat{\mathbf{s}}$
1: **Return MV-PCAttack**($\mathcal{O}_{mv}$)

ConsGraftedT
Input: Coefficients $\hat{\mathbf{s}}$ from round 1, **Trees**
Output: **tree_list**
1: Initialise **tree_list**
2: **for** $\hat{\mathbf{s}}[i]$ in $\hat{\mathbf{s}}$ **do**
3: Find the **tree** where $\hat{\mathbf{s}}[i]$ is at the second level of it from **Trees**
4: Append the corresponding **tree** to **tree_list**
5: **end for**
6: **Return tree_list**

Recovery in round 2
Input: An imperfect MV-PC oracle $\mathcal{O}_{mv}$ and coefficients $\hat{\mathbf{s}}$ form round 1
Output: Roughly secret key coefficients $\bar{\mathbf{s}}$
1: **tree_list** $\leftarrow$ **ConsGraftedT**($\hat{\mathbf{s}}$)
2: $\bar{\mathbf{s}} \leftarrow$ **MV-PCAttack**($\mathcal{O}_{mv}$) with **tree_list**
3: **Return** $\bar{\mathbf{s}}$

Recovery in round 3
Input: An imperfect MV-PC oracle $\mathcal{O}_{mv}$
Input: coefficients $\hat{\mathbf{s}}, \bar{\mathbf{s}}$ form round 1 and 2
Output: Secret key coefficients $\mathbf{s}$
1: **for** $i = 0 \rightarrow l - 1$ **do**
2: **for** $j = 0 \rightarrow (256/N) - 1$ **do**
3: **for** $k = 0 \rightarrow N - 1$ **do**
4: **if** $\bar{\mathbf{s}}_i[j * N + k] \neq \hat{\mathbf{s}}_i[j * N + k]$ **then**
5: Set this block suspicious
6: **end if**
7: **end for**
8: **end for**
9: **end for**
10: $\mathbf{s} \leftarrow$ **MV-PCAttack**($\mathcal{O}_{mv}$) for suspicious block
11: **for** $i = 0 \rightarrow l - 1$ **do**
12: **for** $j = 0 \rightarrow (256/N) - 1$ **do**
13: **for** $k = 0 \rightarrow N - 1$ **do**
14: **if** $\mathbf{s}_i[j * N + k] \neq \hat{\mathbf{s}}_i[j * N + k]$ **then**
15: $\mathbf{s}_i[j * N + k] = \bar{\mathbf{s}}_i[j * N + k]$
16: **end if**
17: **end for**
18: **end for**
19: **end for**
20: **Return s**

4 Experiments and analysis

This section presents the results and analysis of empirical studies designed to objectively assess the effectiveness of our improved method. We begin by demonstrating the efficiency of our improved method through software simulation. Next, we compare the experimental results with those of Tanaka et al. [TUX+23] and Rajendran et al. [RRD+23] for Kyber-1024, Kyber-768, Kyber-512, FireSaber, and Frodo-1344, respectively, showcasing how our method enhances existing work. Finally, we apply our method in a real-world scenario to illustrate that the simulation experiment results align with actual scenarios.

4.1 Software simulations

4.1.1 Simulation settings

All our simulations are conducted on a desktop featuring a 3 GHz Intel Core i5-7400 CPU and 16 GB RAM. Our code is derived from the C implementation of Kyber submitted to the third round of the NIST PQC project. Recall that an imperfect MV-PC oracle $\mathcal{O}mv$ succeeds with an accuracy α. That is, the MV-PC oracle returns the correct result with a probability α and, with a probability of $1 - \alpha$, provides incorrect outputs. When defining the MV-PC oracle to recover N coefficients simultaneously, we employ random selection from $m_0, m_1, \cdots, m_{2^N-1}$ to indicate instances when the MV-PC oracle returns an error.

4.1.2 Comparison with existing work

Table 5: The accuracy of the NN for 2^N-classification in [TUX+23]

	2^1	2^2	2^3	2^4	2^5	2^6	2^7	2^8
Accuracy	99.93%	99.98%	99.94%	99.93%	99.93%	99.93%	99.92%	99.92%

Table 6: Comparison of experimental results in Kyber

Schemes	N	**Method**	**#EvQuery**	**#EvError**
Kyber-512	4	Tanaka et al.	1152 (ref)	0.00/512 [1]
		Rajendran et al.	1150.8	0.0003/512
		Our Method	641.7 (−44.2%)	0.002/512
	8	Tanaka et al.	576 (ref)	0.00/512
		Rajendran et al.	576.0	0.001/512
		Our Method	321.2 (−44.2%)	0.003/512
Kyber-768	4	Tanaka et al.	1728 (ref)	0.00/768
		Rajendran et al.	1599.3	0.001/768
		Our Method	919.6 (−42.5%)	0.005/768
	8	Tanaka et al.	864 (ref)	0.00/768
		Rajendran et al.	849.7	0.0004/768
		Our Method	476.7 (−43.9%)	0.006/768
Kyber-1024	4	Tanaka et al.	2304 (ref)	0.00/1024
		Rajendran et al.	2132.4	0.001/1024
		Our Method	1226.0 (−42.5%)	0.002/1024
	8	Tanaka et al.	1152 (ref)	0.00/1024
		Rajendran et al.	1132.9	0.001/1024
		Our Method	635.5 (−43.9%)	0.003/1024

In our software simulations, we validate the improvement of our proposed method over the previous work of Tanaka et al [TUX+23] and Rajendran et al [RRD+23]. They instantiate the MV-PC oracle using a multiple-classification Neural Network (NN) model and a t-test-based classifier, respectively. Table 5 presents the accuracy of the trained NNs on the test sets for 2^N-classification.

In the work of Rajendran et al [RRD+23], they mention that it is possible to utilize majority voting or error correcting codes in [SCZ+23] to enhance the success rate. To give a fair comparison, in our experiment, we use majority voting ($t = 3$) to improve Rajendran et al.'s results in a noisy environment. It is worth noting that in our simulation, the oracle accuracy α can be set as 99.90%, slightly lower compared to Tanaka et al.'s 99.93% when $N = 4$ and 99.92% when $N = 8$.

We employ **#EvQuery** to signify the average number of queries needed for full key recovery and **#EvError** to denote the average number of error coefficients present in the final result. The latter serves as a crucial indicator of the effectiveness of the full key recovery. Both our approach and previous methods aim to minimize **#EvError** to below 1.0. As depicted in Table 6, we generate $10,000$ random secret keys for various security levels of Kyber to assess the performance of our method. The reported results represent the average number of queries required for the recovery of $10,000$ random secret keys.

[1] Tanaka et al. did not give the exact number of the **#EvError** in their experiments, but their method can achieve nearly 0 errors.

In comparison to existing work, our proposed method exhibits significant improvements. Specifically, for Kyber-1024, we achieve a reduction of 42.5% and 43.9% in total queries when $N = 4$ and 8, respectively. Similarly, for Kyber-512, our method leads to a reduction of 44.3% in total queries. Regardless of whether $N = 4$ or $N = 8$, our method consistently decreases the number of queries by more than 42.5% compared to existing work.

Even with $\alpha = 99.90\%$, **#EvError** remains significantly below 1.0 in our simulations. This suggests that achieving the desired outcomes is feasible even with a lower accuracy. In other words, our proposed method does not require an excessively high level of accuracy to achieve the expected results. In the next subsection, we will describe how we obtain a threshold for accuracy.

Table 7: Comparison of experimental results in FireSaber and Frodo-1344

Schemes	N	**Method**	**#EvQuery**	**#EvError**
FireSaber	4	Tanaka et al.	2304(ref)	0.00/1024
		Rajendran et al.	2301.6	0.005/1024
		Our Method	1283.6 (−44.2%)	0.005/1024
	8	Tanaka et al.	1152 (ref)	0.00/1024
		Rajendran et al.	1152.0	0.003/1024
		Our Method	642.5 (−44.2%)	0.004/1024
Frodo-1344	4	Tanaka et al.	21504(ref)	0.00/10752
		Our Method	16257.1 (−24.4%)	0.03/10752
	7	Tanaka et al.	18432 (ref)	0.00/10752
		Our Method	11433.2 (−38.0%)	0.229/10752

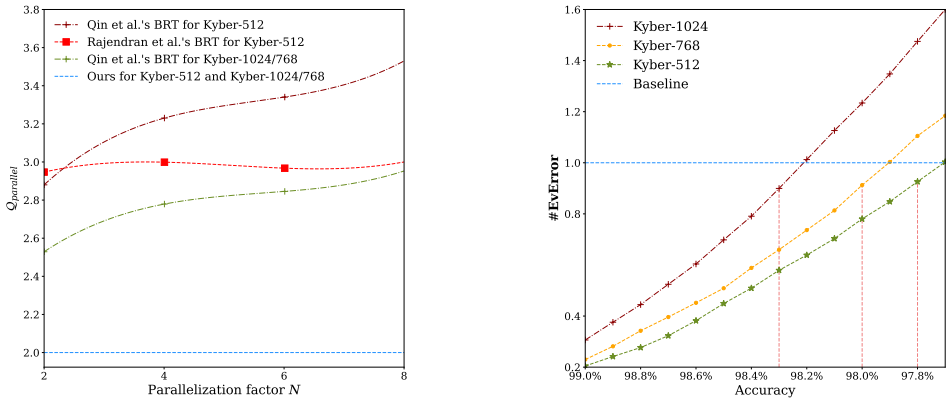
Moreover, our method is generic and can be applied to other LWE-based schemes such as Saber [DKRV19]. Saber is another third-round NIST candidate that provides three levels of security: LightSaber, Saber, and FireSaber. The security of Saber is based on the learning with rounding (LWR) problem.

The secret key coefficients of FireSaber are selected within the range of [-3,3], similar to Kyber-512. This means that we can use the same grafted tree approach that is used for Kyber-512 with FireSaber. We conduct an experimental validation of this method on FireSaber by testing it against 10,000 randomly generated secret keys. In Table 7, it can be observed that we achieve a reduction of 44.2% in the total number of queries, compared to previous work.

The experimental results demonstrate that our new method can be easily migrated to Saber and remains efficient. Furthermore, the results of the experiment for Frodo-1344 are also presented in Table 7, further demonstrating our approach can be applied to other KEM schemes that use variations of FO. It can be noted that the best experimental results for Frodo-1344 in [TUX+23] is given at $N = 7$, not $N = 8$. To give a fair comparison, in Table 7, we also set $N = 7$ in our experiment against Frodo-1344.

4.1.3 Comparison between the grafted tree and previous method in round 2

We compare our improved method with the approaches of Rajendran et al. [RRD+23] and Qin et al. [QCZ+21]. We let $\mathcal{Q}_{parallel}$ represent the average number of queries to recover a coefficient block in one round. Note that the $\mathcal{Q}_{parallel}$ provided is for key recovery in round 2, not for the first round, where the best $\mathcal{Q}_{parallel}$ is still the one proposed in [RRD+23] and [QCZ+21]. Thus, we focus on the performance of the graft tree method compared to the previous method in round 2. For all security levels of Kyber, we compare our results using the grafted tree with Qin et al.'s and Rajendran et al.'s BRTs. The values of $\mathcal{Q}_{parallel}$ for different parallelization factors N and our comparisons are presented in Figure 6a.



(a) $\mathcal{Q}_{parallel}$ with parallelization factor N in different methods against Kyber in round 2

(b) Changes in **#EvError** as the accuracy decreases

Figure 6: $\mathcal{Q}_{parallel}$ in round 2 and changes in **#EvError**.

We observe that for Kyber-512, $\mathcal{Q}_{parallel}$ increases with the parallelization factor N and converges to 4.0 with Qin et al.'s BRT, while it tends to be 3.0 with Rajendran et al.'s more balanced BRT. For Kyber-1024, Rajendran et al.'s BRT coincides with Qin et al.'s BRT, hence we only present the result with Qin et al.'s BRT. The value of $\mathcal{Q}_{parallel}$ also tends to 3.0.

The results also confirm our earlier statement in Section 2.2.4 that the number of queries required to recover N coefficients depends on the coefficient with the maximum depth. With our improved method, $\mathcal{Q}_{parallel}$ stabilizes around 2.0, representing a substantial reduction in attack overhead compared to other methods.

4.1.4 The threshold for accuracy

In [SCZ+23], a threshold is introduced to distinguish between high and low-accuracy cases. Similarly, we seek an accuracy threshold indicating the minimum acceptable accuracy for our proposed method, while ensuring **#EvError** remains below 1.0. This threshold is identified through extensive simulation experiments wherein we systematically reduce the oracle's accuracy rate and monitor the resulting **#EvError**. The accuracy rate is consistently decreased until **#EvError** surpasses 1.0.

Figure 6b illustrates how **#EvError** changes as the accuracy rate decreases. The blue line in the figure represents an **#EvError** of 1.0, serving as the baseline for comparison. Measuring the precise accuracy when **#EvError** reaches 1.0 experimentally poses challenges, but an approximate value can be determined to establish an accuracy threshold. Specifically, for Kyber-1024, our improved method ensures that **#EvError** is kept below 1.0 with an accuracy of 98.3%, while for Kyber-768 and Kyber-512 the respective accuracy thresholds are 98.0% and 97.8%.

4.1.5 Experiments for lower accuracy

Note that our method is generic and applicable to various classifiers. Moreover, it exhibits enhanced performance, especially with 2^N-classifiers, where $N > 8$, provided they maintain sufficient accuracy. Our adversary model (MV-PC oracle) is tailored for optimal performance in low-noise environments, where its effectiveness is most prominent. Our primary focus is to refine the MV-PC oracle to optimize its effectiveness in such

environments. Next, we will further evaluate the performance of our model under different noise levels (e.g., $\alpha = 95.00\%$ and $\alpha = 90.00\%$) to demonstrate the adaptive capability of the proposed method even in high noise environments, with some adjustments. Specifically, we first incorporate majority voting to improve the accuracy of the oracle, and then still use our methodology.

We conduct experiments for Kyber-1024 with $N = 8$ and compare our results with those in [RRD+23] (with majority voting). The results are displayed in Table 8, in which $\mathbf{t}$ represents the number of votes cast.

Table 8: Comparison between majority voting and proposed method in low accuracy

Accuracy	**Method**	**#EvQuery**	**#EvError**
95.00%	Majority Voting ($\mathbf{t} = 5$)	1888 (ref)	0.372/1024
	Our Method	1332 (-29.4%)	0.351/1024
90.00%	Majority Voting ($\mathbf{t} = 7$)	2643 (ref)	0.477/1024
	Our Method	2041 (-22.8%)	0.646/1024

Next, we provide a more detailed explanation of our method concerning low-accuracy scenarios. We still need three rounds. To compensate for the problems associated with accuracy degradation, we use majority voting ($\mathbf{t} = 3$) to help refine the output of the MV-PC oracle. This entails executing the same decryption process multiple times whenever the attacker accesses the oracle, resulting in multiple outcomes. A majority vote is then conducted for the result. For an accuracy of $\alpha = 95.00\%$, we employ a majority voting in rounds 2 and 3. When the accuracy decreases to $\alpha = 90.00\%$, majority voting is employed across all three rounds.

As depicted in Table 8, although the use of oracle combined with majority voting increases the overhead of the attack, our method applied to Kyber-1024 exhibits comparable **#EvError** to majority voting, and it reduces the **#EvQuery** by 29.4% and 22.8% when achieving accuracy of 95.00% and 90.00%, respectively. These findings highlight the efficiency of our method even in high-noise environments, where the performance of the MV-PC oracle is typically hindered.

4.2 Real-world experiments

Table 9: The hyperparameters of the NN for 2^8-class classifier

	Operator	Activation function	Batch normalization	Pooling	Stride
Conv1	conv1d (3)	SELU	Yes	Avg (2)	2
Conv2	conv1d (3)	SELU	Yes	Avg (2)	2
Conv3	conv1d (3)	SELU	Yes	Avg (2)	2
Conv4	conv1d (3)	SELU	Yes	Avg (2)	2
Conv5	conv1d (3)	SELU	Yes	Avg (2)	2
Conv6	conv1d (3)	SELU	Yes	Avg (2)	2
FLT	flatten	-	-	-	-
FC1	dense	SELU	No	No	-
FC2	dense	SELU	No	No	-
FC3	dense	Softmax	No	No	-

4.2.1 Experiment settings

In this subsection, we conduct experiments to validate the feasibility and efficiency of our proposed attack in real-world scenarios. The experiments are implemented on an

STM32F407G board, featuring an ARM Cortex-M4 microcontroller. We choose Cortex-M4 since NIST has recommended it for efficiency evaluation in their postquantum cryptography standardization. The ARM-optimized Kyber-512 implementation from the publicly available testing and benchmarking library *pqm4* [KRSS19] is executed on the board.

Then, we employ a PicoScope 3403D oscilloscope and a CYBERTEK EM5030-3 EM Probe to collect the waveforms and use the PicoSDK interface provided with the PicoScope oscilloscopes to drive the PicoScope series of digital oscilloscopes. The oscilloscope is connected to the probe via a CYBERTEK EM5020A signal amplifier. Whenever the trigger function is called, the oscilloscope collects waveforms generated from the $\mathcal{G}(\mathbf{m}', \mathcal{H}(pk))$. Each collected waveform contains 50,000 sample points, and the sample rate is set at 500 MHz.

The real-world attack is divided into two phases, the profiling phase and the attack phase. In the profiling phase, we first construct the special ciphertext and call the **KYBER.CCA.Decaps** function to collect the waveform. Specially, we set $\mathbf{u} = (0, \cdots, 0)$ and $\mathbf{c}_2 = (k_1, k_2, \ldots, k_{N-1}, 0, \ldots, 0)$. The value of k_i determines whether the value of $\mathbf{m}'[i]$ is 0 or 1. By constantly adjusting the value of $\mathbf{c}_2$, we can obtain waveforms corresponding to $\mathbf{m}_0, \cdots, \mathbf{m}_{2^N-1}$, respectively. For each plaintext, we collect 200 waveforms, of which 100 are used for training, 50 for validation, and 50 for testing. We aim to train a 2^8 class classifier with the collected waveforms. Specially, we employ CUDA 12.2, cuDNN 8.3, Keras 2.10.0, and Tensorflow-gpu 2.10.1 on a desktop equipped with Intel Core i9-12900K and NVIDIA GeForce RTX 3030 to train an NN model.

We adopt the NN framework proposed by Tanaka et al. [TUX+23] with some modifications according to the format of our collected waveforms. To ensure the robustness of our model, we get an average accuracy of 99.86% over 100 iterations. According to the threshold given above, this classifier is sufficient for our proposed attack.

In the attack phase, we first describe how to use the trained classifier to instantiate the MV-PC oracle. Whenever we construct the ciphertext ct according to the BRT or grafted tree and access the MV-PC oracle, the waveform generated from the $\mathcal{G}(\mathbf{m}', \mathcal{H}(pk))$ function is fed as an input to the 2^N-classifier, and then the classification result of the classifier is used as the output of the MV-PC oracle, and we thus obtain the value of $\mathbf{m}'$. Finally, we perform full key recovery as shown in Section 3 and record the number of times MV-PC oracle is queried.

4.2.2 Experiment results

Table 10: Comparison of results between real-world and simulated attacks on Kyber-512

$N = 8$	**#EvQuery**	**#EvError**
Real-world	321.9	0.20
Simulations	321.2	0.003

We conduct the full key recovery process 10 times using the trained classifier and finally get the values of **#EvQuery** and **#EvError**.

In Table 10, we present the results of both software simulations and real-world experiments. In the real-world experiments, we achieve a full key recovery with an average of only 321.9 queries using the proposed new method. The close correspondence between the results of real-world attacks and software simulations validates the usability and efficiency of our method in real-world scenarios.

During the experiment, we also record the number of error coefficients during round 1 and round 2, which are 1.1 and 0.3, respectively. The number of error coefficients in round 1 is equal to the number of errors in full key recovery if no measures (e.g. NLL, majority voting, and the proposed method) have been applied. It can be noticed that the use of the

grafted tree in round 2 not only reduces the number of queries in round 2 but also leads to a reduction in the number of errors, which in the end improves the reliability of the results.

5 Conclusions

In this paper, we have proposed an improved MV-PC oracle-based side-channel attack against LWE-based KEMs, leveraging grafted BRTs to efficiently utilize prior information. Simulations and real-world implementations have shown that our method reduces the number of queries for a full key recovery by more than 42.5%, compared to the state-of-the-art at TCHES 2023. We have also experimentally validated our attack on FireSaber and Frodo-1344 to show that our method is generic and can be applied to other important KEM candidates in the NIST PQC competition. To further demonstrate the generality of our proposed method, we add a supplementary experiment in environments characterized by relatively higher levels of noise and still reduce the number of queries by more than 22.8%.

In practice, the number of queries can be further reduced by combining our method with post-processing techniques such as lattice reduction. We can recover only a part of the coefficients and then recover the remaining ones via the lattice reduction framework in an offline manner. For example, following the work in [MJZ22], we can use the LWE estimator in [DSDGR20] to estimate the coefficients to be recovered and further reduce the number of queries for Kyber-512, Kyber-768, and Kyber-1024 by 34%, 29%, and 27%, respectively, with the ability to perform 2^{32} offline computations.

References

[AAC+22] Gorjan Alagic, Daniel Apon, David Cooper, Quynh Dang, Thinh Dang, John Kelsey, Jacob Lichtinger, Yi-Kai Liu, Carl Miller, Dustin Moody, et al. Status report on the third round of the nist post-quantum cryptography standardization process. *US Department of Commerce, NIST*, 2022.

[ABD+19] Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. Crystals-kyber: Algorithm specification and supporting documentation (version 2.0). In *Submission to the NIST post-quantum project (2019)*, 2019. https://pq-crystals.org/kyber.

[BDH+21] Shivam Bhasin, Jan-Pieter D'Anvers, Daniel Heinz, Thomas Pöppelmann, and Michiel Van Beirendonck. Attacking and defending masked polynomial comparison for lattice-based cryptography. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 334–359, 2021.

[DHP+21] Jan-Pieter D'Anvers, Daniel Heinz, Peter Pessl, Michiel Van Beirendonck, and Ingrid Verbauwhede. Higher-order masked ciphertext comparison for lattice-based cryptography. *Cryptology ePrint Archive*, 2021.

[DKRV19] Jan-Pieter D'Anvers, Angshuman Karmakar, Sujoy Sinha Roy, and Frederik Vercauteren. Saber: Mod-lwr based kem algorithm specification and supporting documentation. In *Submission to the NIST post-quantum project (2019)*, 2019. https://www.esat.kuleuven.be/cosic/publications/article-3055.pdf.

[DSDGR20] Dana Dachman-Soled, Léo Ducas, Huijing Gong, and Mélissa Rossi. Lwe with side information: attacks and concrete security estimation. In *Annual International Cryptology Conference*, pages 329–358. Springer, 2020.

[DTVV19] Jan-Pieter D'Anvers, Marcel Tiepelt, Frederik Vercauteren, and Ingrid Verbauwhede. Timing attacks on error correcting codes in post-quantum schemes. In *Proceedings of ACM Workshop on Theory of Implementation Security Workshop*, pages 2–9, 2019.

[FO99] Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In *Annual International Cryptology Conference*, pages 537–554. Springer, 1999.

[GJN20] Qian Guo, Thomas Johansson, and Alexander Nilsson. A key-recovery timing attack on post-quantum primitives using the fujisaki-okamoto transformation and its application on frodokem. In *Annual International Cryptology Conference*, pages 359–386. Springer, 2020.

[GNNJ23] Qian Guo, Denis Nabokov, Alexander Nilsson, and Thomas Johansson. SCA-LDPC: A code-based framework for key-recovery side-channel attacks on post-quantum encryption schemes. In Jian Guo and Ron Steinfeld, editors, *Advances in Cryptology - ASIACRYPT 2023 - 29th International Conference on the Theory and Application of Cryptology and Information Security, Guangzhou, China, December 4-8, 2023, Proceedings, Part IV*, volume 14441 of *Lecture Notes in Computer Science*, pages 203–236. Springer, 2023.

[HHP+21] Mike Hamburg, Julius Hermelink, Robert Primas, Simona Samardjiska, Thomas Schamberger, Silvan Streit, Emanuele Strieder, and Christine van Vredendaal. Chosen ciphertext k-trace attacks on masked cca2 secure kyber. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 88–113, 2021.

[ISUH21] Akira Ito, Kotaro Saito, Rei Ueno, and Naofumi Homma. Imbalanced data problems in deep learning-based side-channel attacks: analysis and solution. *IEEE Transactions on Information Forensics and Security*, 16:3790–3802, 2021.

[Koc96] Paul C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In Neal Koblitz, editor, *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer, 1996.

[KRSS19] Matthias J Kannwischer, Joost Rijneveld, Peter Schwabe, and Ko Stoffelen. pqm4: Testing and benchmarking nist pqc on arm cortex-m4. 2019. https://github.com/mupq/pqm4.

[MJZ22] Ruiqi Mi, Haodong Jiang, and Zhenfeng Zhang. Lattice reduction meets key-mismatch: New misuse attack on lattice-based nist candidate kems. *Cryptology ePrint Archive*, 2022.

[Moo16] Dustin Moody. *Post Quantum Cryptography Standardization: Announcement and outline of NIST's Call for Submissions.* PQCrypto 2016, Fukuoka, Japan, 2016. https://csrc.nist.gov/Presentations/2016/Announcement-and-outline-of-NIST-s-Call-for-Submis.

[NDGJ21] Kalle Ngo, Elena Dubrova, Qian Guo, and Thomas Johansson. A side-channel attack on a masked ind-cca secure saber kem. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 4:676âĂŞ707, 2021.

[NIS23] NIST. Module-lattice-based key-encapsulation mechanism standard, 2023. FIPS 203 (Draft), https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.203.ipd.pdf.

[QCZ+21] Yue Qin, Chi Cheng, Xiaohan Zhang, Yanbin Pan, Lei Hu, and Jintai Ding. A systematic approach and analysis of key mismatch attacks on lattice-based nist candidate kems. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 92–121. Springer, 2021.

[RRCB20] Prasanna Ravi, Sujoy Sinha Roy, Anupam Chattopadhyay, and Shivam Bhasin. Generic side-channel attacks on cca-secure lattice-based pke and kems. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(3):307–335, 2020.

[RRD+23] Gokulnath Rajendran, Ravi Ravi, Jan-Pieter D'Anvers, Shivam Bhasin, and Anupam Chattopadhyay. 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(2):418–446, 2023.

[SCZ+23] Muyan Shen, Chi Cheng, Xiaohan Zhang, Qian Guo, and Tao Jiang. Find the bad apples: An efficient method for perfect key recovery under imperfect sca oracles–a case study of kyber. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 89–112, 2023.

[Sho99] Peter W Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM review*, 41(2):303–332, 1999.

[TUX+23] Yutaro Tanaka, Rei Ueno, Keita Xagawa, Akira Ito, Junko Takahashi, and Naofumi Homma. Multiple-valued plaintext-checking side-channel attacks on post-quantum kems. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 473–503, 2023.

[UXT+22] Rei Ueno, Keita Xagawa, Yutaro Tanaka, Akira Ito, Junko Takahashi, and Naofumi Homma. Curse of re-encryption: A generic power/em analysis on post-quantum kems. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 296–322, 2022.

[XPSR+21] Zhuang Xu, Owen Michael Pemberton, Sujoy Sinha Roy, David Oswald, Wang Yao, and Zhiming Zheng. Magnifying side-channel leakage of lattice-based cryptosystems with chosen ciphertexts: The case study of kyber. *IEEE Transactions on Computers (Early Access)*, 2021.