IACR Transactions on Cryptographic Hardware and Embedded Systems
ISSN 2569-2925, Vol. 2023, No. 1, pp. 89–112. DOI:10.46586/tches.v2023.i1.89-112

Find the Bad Apples: An efficient method for perfect key recovery under imperfect SCA oracles – A case study of Kyber

Muyan Shen[1,2], Chi Cheng[1,2,✉], Xiaohan Zhang[1,2], Qian Guo[3] and Tao Jiang[4]

[1] Hubei Key Laboratory of Intelligent Geo-Information Processing, School of Computer Science, China University of Geosciences, Wuhan, China
[2] State Key Laboratory of Integrated Services Networks, Xidian University, Xian, China
[3] Lund University, Lund, Sweden
[4] Research Center of 6G Mobile Communications, School of Cyber Science and Engineering, Huazhong University of Science and Technology, Wuhan, China
chengchi@cug.edu.cn,qian.guo@eit.lth.se

Abstract. Side-channel resilience is a crucial feature when assessing whether a post-quantum cryptographic proposal is sufficiently mature to be deployed. In this paper, we propose a generic and efficient adaptive approach to improve the sample complexity (i.e., the required number of traces) of plaintext-checking (PC) oracle-based side-channel attacks (SCAs), a major class of key recovery chosen-ciphertext SCAs on lattice-based key encapsulation mechanisms (KEMs). This new approach is preferable when the constructed PC oracle is imperfect, which is common in practice, and its basic idea is to design new detection codes that can determine erroneous positions in the initially recovered secret key. These secret entries are further corrected with a small number of additional traces. This work benefits from the generality of PC oracle and thus is applicable to various schemes and implementations.
Our main target is Kyber since it has been selected by NIST as the KEM algorithm for standardization. We instantiated the proposed generic attack on Kyber512 and then conducted extensive computer simulations against Kyber512 and FireSaber. We further mounted an electromagnetic (EM) attack against an optimized implementation of Kyber512 in the pqm4 library running on an STM32F407G board with an ARM Cortex-M4 microcontroller. These simulations and real-world experiments demonstrate that the newly proposed attack could greatly improve the state-of-the-art in terms of the required number of traces. For instance, the new attack requires only 41% of the EM traces needed in a majority-voting attack in our experiments, where the raw oracle accuracy is fixed.

Keywords: Lattice-based cryptography · Side-channel attacks · Plaintext-checking oracle · NIST Post-Quantum cryptography standardization · Kyber · Key mismatch attacks.

1 Introduction

To continue protecting our data once quantum computers become mature, we need to transit from current factoring or discrete-log-based public key cryptography to post-quantum cryptography (PQC). In October 2021, the US National Institute of Standards and Technology (NIST) and the Department of Homeland Security collaborated and released a roadmap to transition to the PQC standard [ND21], which is anticipated to be ready by 2024. Their goal is to complete the transition by 2030.

Received: 2022-07-15 Accepted: 2022-09-15 Published: 2022-11-29

Starting in 2016, NIST's PQC selection process [Moo16] has attracted attention from all over the world. On the third-round list [MAA+20], there are 4 finalists and 5 alternative candidates for Public Key Encryption (PKE) or Key Encapsulation Mechanism (KEM). Lattice-based KEMs occupy the majority, i.e., 3 out of 4 finalists, demonstrating their fundamental role in the PQC standard. Just recently in July 2022, Kyber [ABD+19], the KEM part of the Cryptographic Suite for Algebraic Cipher Suite (CRYSTALS), has been selected as the main KEM algorithm for final standardization [AAC+22]. Besides other desirable security properties, NIST has placed a priority on the resistance of these KEMs to side-channel attacks (SCAs) before deploying these PQC algorithms in the real world, especially in the scenario where an attacker can physically access an embedded device.

SCAs were firstly introduced by Kocher in 1996 [Koc96]. By focusing on side-channel measurements from timing, power consumption, or electromagnetic (EM) emanation, the implemented cryptographic algorithms leak information about the long-term secret key or message. In the light of this thought, several SCAs against lattice-based KEMs in the NIST PQC standardization process have been proposed (e.g., [DTVV19, GJN20, RRCB20, XPSR+21, NDGJ21, HHP+21, REB+22, UXT+22]). Most of them are chosen-ciphertext attacks (CCAs) since NIST PQC KEMs generally target provable CCA security, which can be achieved by using the Fujisaki-Okamoto (FO) transformation.

Key recovery SCAs against lattice-based KEMs recovering the long-term secret key is a central research topic in this field since key recovery is a much stronger attack model than only message recovery. We can classify these attacks into two main types. The first type of attack [GJN20, BDH+21] builds an oracle to check whether the decryption is successful, thus being closer to reaction attacks [Ble98, HGS99, MU10], an attack model weaker than the CCA. We call this class of attacks reaction-type SCAs. This type of attack is generic as it can be extended to attacking code-based KEMs [GJN20, UXT+22, GHJ+22], and there is no need to design message-recovery techniques.

The second type of attack, initially proposed by D'Anvers et al. [DTVV19], connects the entries in the long-term secret key to certain chosen messages and achieves key recovery through a message-recovery approach. We call this class of attacks message-recovery-type SCAs. The initial message-recovery-type attacks [DTVV19, RRCB20], also named plaintext-checking (PC) oracle-based SCA in [RR21], can only gain at most one bit of the secret information from one decryption function call. Later, more advanced attacks [XPSR+21, NDGJ21, REB+22] were discovered, which could recover a large chunk of the message/secret vectors simultaneously. However, it is worth noting that the latter is much more powerful but relies on strong leakages from specific implementations. For example, Xu et al. used only 4 traces to get full-key recovery for the reference C version of Kyber512, but the traces rise to hundreds when targeting the assembly-optimized version [XPSR+21]. The different compiler-optimization levels also make a huge impact on the number of traces, resulting in 8 traces compiled by GCC with the optimization level -O0 and 960 traces compiled with the maximal optimization level -O3, respectively. Just recently, Ueno et al. showed that the generality of PC oracle could help launch a generic SCA against most NIST PQC third-round KEMs [UXT+22]. What makes their work more interesting is the realization of a deep-learning-based distinguisher with high accuracy, which helps build the PC oracle to attack the lattice-based KEMs even when there are SCA counter-measurements like masking in the implementation.

Although there has been much work on PC oracle-based SCAs, the way these attacks deal with oracle inaccuracy needs more attention. The oracle inaccuracy may occur due to the environmental noises, or simply the measurement limitations in implementing the PC oracle, such as the inaccuracy in the deep-learning-based SCA distinguisher in [ISUH21, UXT+22]. Since the practical SCA oracle cannot be perfect, the recovered secret key may have slight or big corruption due to the inaccuracy rate. To successfully recover the full or almost full secret key, previous works aim to improve the oracle accuracy with

techniques like majority-voting. Compared to the recovery under a perfect oracle, we need more traces under an imperfect oracle. For example, in [RRCB20], Ravi et al. need 2560 traces to recover the full key of Kyber512 when the PC oracle is perfect. But to migrate the measurement errors of SCA, they repeatedly query the oracle three times and then use the majority-voting to recover the full key. This causes three times total traces, i.e. $2560 \cdot 3 = 7680$. Therefore, an interesting problem is, can we find a more efficient way to launch a perfect recovery under an imperfect oracle?

In this paper, we investigate the message-recovery-type SCA against Kyber, aiming to improve the sample complexity when imperfect measuring is assumed. Our focal point is the PC oracle-based SCAs, as these attacks have more generic applications (e.g., [RDB+21]) compared to its improved version that can in parallel recover a large number of bits of information. Also, PC oracle-based SCAs could exploit a long leakage trace (e.g., corresponding to the whole FO transform) to recover just one bit of the secret information, making such attacks difficult to be thwarted [ABH+22]. This research is of great practical significance since real platforms can be either too noisy, or incur measurement errors, to seriously hurt the accuracy of measurements.

1.1 Contributions

We present a new *checking* approach in the PC oracle-based SCAs to efficiently find the problematic entries in the recovered secret key. These entries are further corrected with a small number of additional traces. Compared to the most used method performing majority-voting with multiple traces to increase the attack success probability, the new adaptive method shows a substantial improvement in sample complexity.

The main contributions of this paper are summarized in the following.

- Firstly, we propose a general SCA framework with improved sample complexity that could be widely applied for attacking NIST lattice-based KEMs. In the high-oracle-accuracy region, we treat the detection of corruptions as a coding problem and propose an efficient method to find the erroneous locations. The novel idea is that if the targeted secret block is erroneously recovered, then the check procedure will return a codeword different from the designed ones. We then extend the attack to the low-oracle-accuracy region and propose a new approach called *mixed voting* to improve the decision accuracy using a confidence array.
- Furthermore, we instantiate the described attack framework on Kyber512 and show the details in each step of the new procedure.
- We perform extensive simulations on Kyber512 and FireSaber for various oracle-accuracy levels and mount a real-world EM attack against an optimized implementation of Kyber512 from *pqm4* [KRSS19], a famous testing and benchmarking library optimized for the ARM microcontrollers. We include the computer simulation results for FireSaber to demonstrate that the new attack framework is generic and has wider applications beyond Kyber. Our experimental results show that the new checking approach can improve the majority-voting method significantly, i.e., the required number of traces is approximately halved for the simulated instances. We make our code and data open-source. They are available at https://github.com/7a17/Find-the-Bad-Apples/.

1.2 Related Works

In [RR21], Ravi and Roy categorized the major SCAs on lattice-based KEMs into three classes, decryption-failure (DF) oracle-based, PC or key-mismatch oracle-based, and full-decryption (FD) oracle-based. DF oracle-based SCA is reaction-type, while PC and FD oracle-based SCAs are message-recovery-types.

DF-oracle-based SCAs were initially proposed in [GJN20], where a generic SCA model focusing on the leakage of the FO transformation is presented. This attack model was instantiated as a timing attack [GJN20] on FrodoKEM and also timing attacks [GHJ+22] on code-based NIST candidates HQC and BIKE with variable execution time of the rejection sampling procedure. Attacks and protections against power/EM adversaries were further investigated in [BDH+21, UXT+22].

A series of similar analyses in the presence of PC oracle has also been presented. For example, D'Anvers et al. [DTVV19] exploited the variable runtime information of its non-constant-time decapsulation implementation on the LAC and successfully recovered its long-term secret key. At CHES 2020, Ravi et al. proposed a generic EM chosen-ciphertext SCA by exploiting the leaked information about FO transformation or Error Correcting Codes (ECC) and applied it to six CCA-secure lattice-based KEMs [RRCB20]. Afterward, Qin et al. [QCZ+21] proposed a systematic approach to evaluating the key reuse resilience of CPA-secure lattice-based KEMs and further mounted it on side-channel attacks against the CCA-secure ones, which greatly reduced the needed side-channel traces/queries compared to Ravi et al. In [ZCD21], Zhang et al. investigated the key reuse resilience of the CPA-secure NTRU-HRSS KEM, whose method can also attack the CCA-secure NTRU-HRSS KEM with the help of side-channel leakages. Recently, Ravi et al. further successfully mounted the previous attacking method on two NTRU-based schemes, which are NTRU and NTRU Prime [REB+22].

The idea of FD oracle-based SCA is first proposed by Xu et al. [XPSR+21] and they demonstrated that an adversary only needs 8 traces in recovering the secret key of Kyber512 for ARM-specific implementation compiled at the -O0 optimization level. Compared to PC-oracle-based SCA, Xu et al. can gain the complete message information for the chosen ciphertext, which is equivalent to launching multiple chosen-ciphertext attacks simultaneously. Subsequently, Ravi et al. fully exploited the vulnerabilities of the message decoding function on the lattice-based KEMs and launched the corresponding side-channel attacks for different implementations of CCA-secure lattice-based KEMs [RBRC21]. Their attack can be extended to several implementations with side-channel countermeasures such as shuffling and masking but only perform message recovery. More recently, there is a side-channel attack successfully recovering the long-term secret key for the masked implementation of SABER [NDGJ21], which removes Ravi et al.'s restriction.

Organizations. The remaining of the paper is organized as follows. We present the necessary background in Section 2 and our main strategy in Section 3. This is followed by a concrete instantiation for attacking Kyber512 in Section 4 and the experimental results in Section 5. Finally, we conclude the work in Section 6.

2 Previous PC-based SCA against Kyber

2.1 Kyber and the PC oracle

The security of Kyber [ABD+19] is based on the hardness of solving the module learning with errors (M-LWE) problem. From linear algebra, we know that it is easy to retrieve $\mathbf{s}$ from linear equations $\mathbf{b} = \mathbf{As}$. Here $\mathbf{A}$ is a matrix and $\mathbf{b}$ and $\mathbf{s}$ are vectors. But if we add noises even with small coefficients, the resulted LWE problem [Reg09], i.e. recovering $\mathbf{b} = \mathbf{As} + \mathbf{e}$ can be hard. The Ring-LWE problem [LPR10] is to replace matrices and vectors with polynomials, thus significantly reducing the computation and communication costs. In an M-LWE problem [LS15, ADPS16], we select $\mathbf{s}$ and rows of $\mathbf{A}$ in a module rather than in a polynomial ring, and in this setting, Kyber enjoys the advantages of relatively easy scalability and high efficiency.

To be specific, Kyber is defined over a polynomial ring $\mathcal{R}_q = \mathbb{Z}_q[x]/(x^n + 1)$. Here $q = 3329$ is a modulo and $n = 256$. For every polynomial $f(x) = a_0 + a_1 x + \cdots + a_{n-1}x^{n-1} \in \mathbb{R}_q$,

each coefficient $a_i \in \mathbb{Z}_q$ $(0 \leq i \leq n-1)$, where $\mathbb{Z}_q$ represents a ring with all elements are integers modulo q. All the polynomial additions and multiplications are operated modulo $x^n + 1$. We interchangeably use $\mathbf{c} \in \mathcal{R}_q$ and its vector form $(\mathbf{c}[0], \cdots, \mathbf{c}[n-1])$ to represent a polynomial. For a matrix $\mathbf{A} \in \mathcal{R}_q^{k \times k}$, $\mathbf{s}, \mathbf{e} \in \mathcal{B}_\eta^k$, where $\mathcal{B}_\eta$ represents the centered binomial distribution with parameter η, and 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 $\{0, 1\}$. Then, the M-LWE problem is to distinguish $(\mathbf{A}, \mathbf{B} = \mathbf{As} + \mathbf{e}) \in \mathcal{R}_q^{k \times k} \times \mathcal{R}_q^k$ from uniformly selected $(\mathbf{A}, \mathbf{B}) \in \mathcal{R}_q^{k \times k} \times \mathcal{R}_q^k$.

Algorithm 1 The encapsulation and decapsulation in Kyber

◇ **CCAKEM.Encaps**
Input: Public Key $\mathbf{p}$
Output: Ciphertext $\mathbf{ct} = (\mathbf{c}_1, \mathbf{c}_2)$
Output: Shared Key K
$\mathbf{m} \xleftarrow{\$} \{0,1\}^{256}$
$(\bar{K}, r) = \mathrm{G}(\mathbf{m}, \mathrm{H}(\mathbf{p}))$
▷ $\mathrm{CPA.Enc}(\mathbf{p}, \mathbf{m}, r)$
  $\mathbf{A} \xleftarrow{\$} \mathcal{R}_q^{k \times k}$
  $\mathbf{r} \xleftarrow{\$} \mathcal{B}_{\eta_1}^k$, $\mathbf{e}_1, \mathbf{e}_2 \xleftarrow{\$} \mathcal{B}_{\eta_2}^k$
  $\mathbf{u} = \mathbf{A}^T\mathbf{r} + \mathbf{e}_1$
  $\mathbf{v} = \mathbf{p}^T\mathbf{r} + \mathbf{e}_2 + \mathbf{Decomp}_q(\mathbf{m}, 1)$
  $\mathbf{c}_1 = \mathbf{Comp}_q(\mathbf{u}, d_\mathbf{u})$
  $\mathbf{c}_2 = \mathbf{Comp}_q(\mathbf{v}, d_v)$
$K = \mathrm{KDF}(\bar{K}, \mathrm{H}(\mathbf{c}_1, \mathbf{c}_2))$

◇ **CCAKEM.Decaps**
Input: Ciphertext $\mathbf{ct} = (\mathbf{c}_1, \mathbf{c}_2)$
Input: Secret Key $\mathbf{s}$
Output: Shared Key K
▷ $\mathrm{CPA.Dec}(\mathbf{s}, \mathbf{ct})$
  $\mathbf{u}' = \mathbf{Decomp}_q(\mathbf{c}_1, d_u)$
  $\mathbf{v}' = \mathbf{Decomp}_q(\mathbf{c}_2, d_v)$
  $\mathbf{m}' = \mathbf{Comp}_q(\mathbf{v}' - \mathbf{s}^T\mathbf{u}', 1)$
$(\bar{K}', r') = \mathrm{G}(\mathbf{m}', \mathrm{H}(\mathbf{p}))$
$(\mathbf{c}_1', \mathbf{c}_2') = \mathrm{CPA.Enc}(\mathbf{p}, \mathbf{m}', r')$
if $(\mathbf{c}_1, \mathbf{c}_2) = (\mathbf{c}_1', \mathbf{c}_2')$ **then**
  $K = \mathrm{KDF}(\bar{K}', \mathrm{H}(\mathbf{c}_1', \mathbf{c}_2'))$
else
  $K = \mathrm{KDF}(z, \mathrm{H}(\mathbf{c}_1', \mathbf{c}_2'))$
end if

The security of Kyber can be shifted by simply modifying k. More specifically, there are three security levels in Kyber: Kyber512, Kyber768, and Kyber1024, corresponding to $k = 2$, $k = 3$, and $k = 4$, respectively. As we can see in Algorithm 1, η_1 determines the noise of $\mathbf{r}$, while η_2 determines the noise of $\mathbf{e}_1, \mathbf{e}_2$. In Kyber512, $(\eta_1, \eta_2) = (3, 2)$; in Kyber 768 and Kyber1024, $(\eta_1, \eta_2) = (2, 2)$. Generally, a KEM consists of key generation, encapsulation, and decapsulation. But a PC-based SCA is done against the decapsulation part. Thus, in Algorithm 1, we only depict the main parts of encapsulation and decapsulation of Kyber, ignoring details such as the Number Theoretic Transform (NTT).

Let $\lceil x \rfloor$ denote the common rounding function, i.e., the nearest integer to x. In the following, we first define two functions, $\mathbf{Comp}_q(x, d)$ and $\mathbf{Decomp}_q(x, d)$.

Definition 1. The Compression function is defined as: $\mathbb{Z}_q \to \mathbb{Z}_{2^d}$

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

Definition 2. The Decompression function is defined as: $\mathbb{Z}_{2^d} \to \mathbb{Z}_q$

$$\mathbf{Decomp}_q(x, d) = \left\lceil \frac{q}{2^d} \cdot x \right\rfloor. \tag{2}$$

Similarly, when the inputs of $\mathbf{Comp}_q(x, d)$ and $\mathbf{Decomp}_q(x, d)$ are polynomials, i.e., $x \in \mathcal{R}_q^k$, the above operation is separately done on each coefficient. Let $\xleftarrow{\$}$ represent random selection and T represent the transpose of a matrix. In both the encapsulation and decapsulation, two hash functions G and H, as well as a key derivation function KDF, are used. CCA-secure Kyber is achieved through the well-known Fujisaki-Okamoto (FO) transform based on a CPA-secure PKE. In the encapsulation, a CPA-secure encryption algorithm is used to output $\mathbf{c}_1$ and $\mathbf{c}_2$. Here d_u and d_v used in $\mathbf{Comp}_q(x, d)$ and $\mathbf{Decomp}_q(x, d)$ functions are also determined by different security levels. For example, in

Kyber-512, $d_u = 10$, $d_v = 4$. In the decapsulation, a CPA-secure decryption algorithm is firstly used to obtain $(\bar{K}', r')$, which is then re-encrypted to get $(\mathbf{c}_1', \mathbf{c}_2')$.

The CPA-secure KEMs are vulnerable to chosen-ciphertext attacks when the secret key is reused. These attacks are generally operated in a key-mismatch or PC Oracle. Algorithm 2 first depicts the PC oracle $\mathcal{O}$, in which the adversary sends ciphertext $\mathbf{ct}$ and a reference plaintext $\mathbf{m}$ to the oracle. The oracle tells whether $\mathbf{m}$ equals the CPA decryption result $\mathbf{m}'$ or not. On the right part of Algorithm 2, we introduce the process of recovering the secret key by employing the response sequence from PC oracle[1] $\mathcal{O}$. In the recovery process, special ciphertexts are crafted to combine every possible coefficient value (such as $[-3, 3]$ in Kyber512) with a certain oracle response sequence. For example, if $\mathcal{O}$ is always accurate, Ravi et al. needed 5 queries to recover one coefficient and $256 \cdot 2 \cdot 5 = 2560$ queries in total for the second round Kyber512. After that, Qin et al. reduced the queries to an average of 1312 for the third round Kyber512 by using the optimal binary recovery tree. We next briefly introduce the main process proposed by Qin et al., an improved method close to Huffman coding to achieve key recovery, and more details can be found in [QCZ+21].

Algorithm 2 PC oracle $\mathcal{O}$ and the key recovery process

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

◇ **KeyRecovery**
Input: PC oracle $\mathcal{O}$
Output: Secret Key $\mathbf{s}$
1: **for** All coefficients location *loc* in $\mathbf{s}$ **do**
2: ▷ **CoefficientRecovery**(*loc*)
3: *seq* = “”
4: **while** $\mathbf{s}$[*loc*] cannot be recovered from *seq* **do**
5: Generate $(\mathbf{ct}, \mathbf{m})$ from *loc* and *seq*
6: $res = \mathcal{O}(\mathbf{ct}, \mathbf{m})$
7: Append *res* to the response sequence *seq*
8: **end while**
9: Set $\mathbf{s}$[*loc*] the corresponding value of *seq*
10: **end for**

2.2 The attack on Kyber from [QCZ+21]

We now take Kyber512 as an example to explain the main attack procedure proposed in [QCZ+21], which will be employed as the initial processing steps in our new attack. We start with building a PC oracle $\mathcal{O}$ shown on the left of Algorithm 2 by instantiating the CPA.Dec() function with Kyber.CPA.Dec().

The victim Alice's secret key is $\mathbf{s}$ and $\mathbf{s} = (\mathbf{s_0}, \mathbf{s_1})$ in Kyber512. On the right of Algorithm 2, the **KeyRecovery** is used to recover $\mathbf{s}$. The aim of the **CoefficientRecovery** is to select proper $\mathbf{ct} = (\mathbf{c}_1, \mathbf{c}_2)$ and $\mathbf{m}$ as inputs to $\mathcal{O}$. Then, the attacker is able to recover $\mathbf{s}$ from the response sequence *seq* of $\mathcal{O}$. In the following, we show the approach to recover the first coefficient $\mathbf{s_0}[0]$ in $\mathbf{s_0}$, and the remaining coefficients can be recovered similarly.

To launch the attack, the attacker sets $\mathbf{m} = (1, 0, \cdots, 0)$ and $\mathbf{u} = (\mathbf{u_0}, \mathbf{u_1})$, where $\mathbf{u_0} = (\lceil \frac{q}{16} \rfloor, 0, \ldots, 0)$ and $\mathbf{u_1} = \mathbf{0}$. Then, the attacker is able to calculate $\mathbf{c}_1 = \mathbf{Comp}_q(\mathbf{u}, d_u)$, as well as setting $\mathbf{c}_2 = (g, 0, \ldots, 0)$. Here g is later set and used to recover $\mathbf{s_0}[0]$.

Next, $\mathbf{ct} = (\mathbf{c}_1, \mathbf{c}_2)$ is sent as input to the Oracle, which is employed to generate $\mathbf{u}' = \mathbf{Decomp}_q(\mathbf{c}_1, d_u)$ and $\mathbf{v}' = \mathbf{Decomp}_q(\mathbf{c}_2, d_v)$, where $\mathbf{u}' = (\mathbf{u_0'}, \mathbf{u_1'})$. Thus, a

[1] We present here a general description of the PC oracle. In PC oracle-based CCA SCAs, the message $\mathbf{m}$ is limited to be chosen from a set of $\{\mathbf{m}_0, \mathbf{m}_1\}$ and the oracle outputs the chosen message.

relationship between $\mathbf{m}'[0]$ and $\mathbf{s_0}[0]$ can be built as follows:

$$\begin{aligned}
\mathbf{m}'[0] =& \mathbf{Comp}_q((\mathbf{v}' - \mathbf{s}^T\mathbf{u}')[0], 1) && (3)\\
=& \left\lceil \frac{2}{q}\left(\mathbf{v}'[0] - (\mathbf{s}^T\mathbf{u}')[0]\right)\right\rfloor \bmod 2 && (4)\\
=& \left\lceil \frac{2}{q}(\mathbf{v}'[0] - (\mathbf{s_0}[0]\mathbf{u}'_\mathbf{0}[0] + \mathbf{s_1}[0]\mathbf{u}'_\mathbf{1}[0]))\right\rfloor \bmod 2 && (5)\\
=& \left\lceil \frac{2}{q}(\mathbf{v}'[0] - \mathbf{s_0}[0]\mathbf{u}'_\mathbf{0}[0])\right\rfloor \bmod 2 && (6)\\
=& \left\lceil \frac{2}{q}\left(\left\lceil \frac{q}{16}g\right\rfloor - \mathbf{s_0}[0]\left\lceil \frac{q}{16}\right\rfloor\right)\right\rfloor \bmod 2. && (7)
\end{aligned}$$

The last equation holds due to the fact that $\mathbf{v}'[0] = \left\lceil \frac{q}{16}g\right\rfloor$ and $\mathbf{u}'_\mathbf{0}[0] = \left\lceil \frac{q}{16}\right\rfloor$.

Further, the attacker could adaptively set different g to recover $\mathbf{s_0}[0]$ based on the sequence outputted from the oracle. Take $g = 4$ as an example: recall that each $\mathbf{s_0}[0]$ is selected from $[-3, 3]$. When $\mathbf{s_0}[0] \in [-3, -1]$, $\mathbf{m}' = (1, 0, \ldots, 0)$, which means $\mathbf{m} = \mathbf{m}'$. So, the output of the oracle is 1. Similarly, when $\mathbf{s_0}[0] \in [0, 3]$, $\mathbf{m}' = (0, 0, \ldots, 0)$, which results the output of the oracle to be 0. From the above analysis, the attacker succeeds in deciding which subinterval, $[-3, -1]$ or $[0, 3]$, $\mathbf{s_0}[0]$ is in using a query. If the attacker chooses proper g, then he could recover $\mathbf{s_0}[0]$ with as few queries as possible. This is achieved by using the optimal binary recovery tree in [QCZ+21].

In Table 1, the selections of g and the corresponding changes of States are depicted. For Kyber512, the States in Table 1 are shifted by the output of the oracle. To be specific, the attacker first lets $g = 4$ and observes the output of the oracle. If the oracle outputs 1, the attacker goes to State 5 and further sets $g = 3$. Then, if the oracle outputs 0, the attacker succeeds in determining $\mathbf{s_0}[0] = -1$. If the oracle outputs 1, the attacker goes to State 6 and sets $g = 2$. Finally, the attacker is able to decide $\mathbf{s_0}[0] = -2$ or $\mathbf{s_0}[0] = -3$ according to the output of the oracle. In fact, in this way, the attacker succeeds in building a relationship between each symbol in $[-3, -1]$ and the response sequence. For example, $\mathbf{s_0}[0] = -1$ corresponds to 10, $\mathbf{s_0}[0] = -2$ corresponds to 110, while 111 is the corresponding response sequence of $\mathbf{s_0}[0] = -3$.

Finally, all the coefficients can be recovered and the total number of queries needed for Kyber512 is 1312.

Table 1: Selections of g and the corresponding States

	State 1	State 2	State 3	State 4	State 5	State 6
g	4	5	6	7	3	2
$\mathcal{O} \to 0$	State 2	State 3	State 4	$\mathbf{s_0}[0] = 3$	$\mathbf{s_0}[0] = -1$	$\mathbf{s_0}[0] = -2$
$\mathcal{O} \to 1$	State 5	$\mathbf{s_0}[0] = 0$	$\mathbf{s_0}[0] = 1$	$\mathbf{s_0}[0] = 2$	State 6	$\mathbf{s_0}[0] = -3$

2.3 PC oracle-based SCA attacks

In the CCA-KEMs such as Kyber and Saber, by employing the FO transform, the above attacks do not work. However, with the help of side-channel information, such as analyzing the power consumption or electromagnetism waveforms during decapsulation, the adversary could directly spot the CPA-secure operations inside the CCA-secure function, and launch the same attacks.

At CHES 2020, Ravi et al. introduced a PC-based SCA attack on NIST KEMs by exploiting the information of the re-encryption procedure in the FO transform [RRCB20]. Again we take the attack against Kyber as an example, the adversary simply let $\mathbf{m}'$ be $\mathbf{m_0} = (0, 0, 0, 0, \cdots)$ or $\mathbf{m_1} = (1, 0, 0, 0, \cdots)$. Correspondingly, the waveform of the

Algorithm 3 PC oracle $\mathcal{O}_{SCA}$ and the Practical Key Recovery process under SCA

◇ **PC oracle $\mathcal{O}_{SCA}$ enacted by SCA**
Input: Ciphertext **ct**
Input: Profiled waveforms $\mathbf{W_{m_0}}$, $\mathbf{W_{m_1}}$
Output: 0 or 1
1: Query the device with **ct** and collect waveform $\mathbf{W_{m'}}$ from FO re-encryption
2: **if** $\mathbf{Dist}(\mathbf{W_{m'}}, \mathbf{W_{m_1}}) < \mathbf{Dist}(\mathbf{W_{m'}}, \mathbf{W_{m_0}})$
3: **Return** 1
4: **else**
5: **Return** 0

◇ **RoughKeyRecovery**
Input: An imperfect PC oracle $\mathcal{O}_{SCA}$
Output: Secret Key **s**
1: **Return KeyRecovery**($\mathcal{O}_{SCA}$)

◇ **SCA PC oracle $\mathcal{O}_V$ with majority-voting**
Input: An imperfect PC oracle $\mathcal{O}_{SCA}$
Output: 0 or 1
1: Query the $\mathcal{O}_{SCA}$ **t** times and add the response value to *sum*
2: **if** $sum > \mathbf{t}/2$ **then**
3: **Return** 1
4: **else**
5: **Return** 0
6: **end if**

◇ **PracticalKeyRecovery**
Input: oracle $\mathcal{O}_V$ with majority-voting
Output: Secret Key **s**
1: **Return KeyRecovery**($\mathcal{O}_V$)

re-encryption procedure (i.e lines 5-6 on the right of Algorithm 1) is also fixed to two types, allowing us to enact a PC oracle $\mathcal{O}_{SCA}$ with a side-channel waveform distinguisher. In [RRCB20], Ravi et al. designed the SCA distinguisher mainly by calculating the Euclidean distance to the profiled waveform templates. More specifically, they firstly repeatedly collect re-encryption waveforms of $\mathbf{m'} = \mathbf{m_0}$ and $\mathbf{m'} = \mathbf{m_1}$. Then they run a Test Vector Leakage Assessment (TVLA) between the two waveform sets to select Points of Interest (PoI). In the next attacking stage, they achieve binary classification by calculating the Euclidean distance between the collected waveforms of PoI and two waveform templates. We summarize the attacking procedure as $\mathcal{O}_{SCA}$ on the left of Algorithm 3. Note that since we restrict $\mathbf{m'}$ to be $\mathbf{m_0}$ or $\mathbf{m_1}$, the problem of distinguishing whether $\mathbf{m'} = \mathbf{m_0}$ can be simplified to deciding which template of $\mathbf{m_0}$ or $\mathbf{m_1}$ has more close Euclidean distance to the collected waveform.

Affected by the noises from the environment or the masking countermeasures, as well as accuracy problems in the side-channel distinguisher itself, $\mathcal{O}_{SCA}$ is imperfect and cannot always tell the truth. We set its accuracy as α_o. Even if α_o reaches 0.990, if we simply instantiate the **KeyRecovery** in Algorithm 2 with $\mathcal{O}_{SCA}$ to recover the secret key, on average we will meet 13.0 error coefficients among all the recovered 512 coefficients (with ciphertext construction method in [QCZ+21]). Since we cannot decide the positions of the errors, the brute force complexity is quite high, which can be estimated as: $\binom{512}{13} \cdot 7^{13} \approx 2^{120.7}$.

Therefore, additional techniques are needed to strengthen the recovery procedure. A commonly used technique, which is also applied in Ravi et al.'s attack, is majority-voting. With a majority-voting of **t** times, we can obtain a more accurate oracle $\mathcal{O}_V$ with accuracy α_{ov}. That is,

$$\alpha_{ov} = 1 - \sum_{s=0}^{\lfloor t/2 \rfloor} \left(\begin{array}{c} t \\ s \end{array} \right) \alpha_o^s (1-\alpha_o)^{t-s}.$$

In the right part of Algorithm 3, we show the oracle $\mathcal{O}_V$ with a majority-voting and the practical key recovery under SCA. With majority-voting of $\mathbf{t} = 3$, the $\mathcal{O}_{SCA}$ with $\alpha_o = 0.990$ mentioned above can be transformed to $\mathcal{O}_V$ with $\alpha_{ov} = 0.9997$. If we instantiate the **KeyRecovery** with $\mathcal{O}_V$ to recover the secret key, we will get less than 1.0 error coefficients among all the 512 coefficients on average. We set it a successful full-key recovery since the remaining complexity now becomes $\binom{512}{1} \cdot 7^1 \approx 2^{11.8}$.

At CHES 2022, Ueno et al. mainly use the deep-learning technique to design the side-channel distinguisher and achieve a similar binary-classification (line 2 on the left of Algorithm 3) [UXT+22]. They use Negative Log-Likelihood (NLL) with $\mathbf{t'}$ traces to gain an oracle with higher accuracy and thus perform key recovery under this oracle. Their

overall idea is highly similar to the procedures on the right of Algorithm 3.

Both of their methods need $\mathbf{t}$ or $\mathbf{t}'$ times the total traces required from a perfect SCA oracle. For example, Ravi et al. apply majority-voting with $\mathbf{t} = 3$ for practical key recovery and need $2560 \cdot 3 = 7680$ queries. Ueno et al. apply the NLL with $\mathbf{t}' = 2$ for key recovery for non-protected software and need $1536 \cdot 2 = 3072$ queries.

In the next section, we propose a more efficient full-key recovery strategy compared with the previously mentioned ones.

3 Our main strategy

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

3.1 Our basic idea

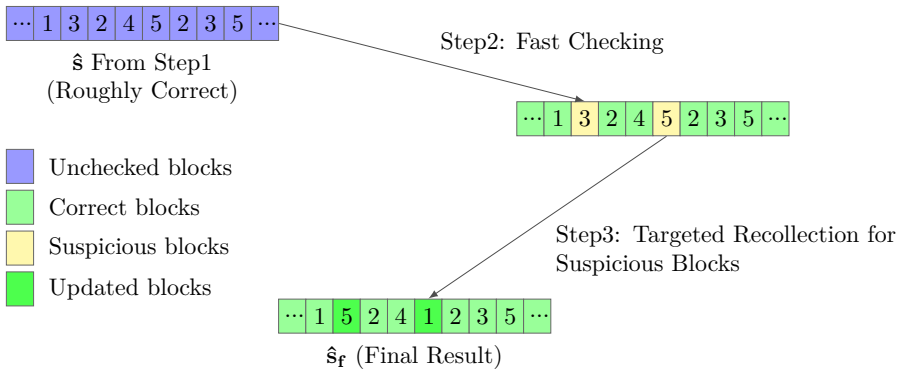

Figure 1: Our full-key recovery strategy

In Figure 1, we illustrate the basic idea of our full-key recovery strategy. Instead of strengthening the accuracy of the oracle, the basic idea of our full-key recovery strategy is to detect error positions in the roughly recovered secret key, and then use an improved method to correct the errors. Let the real secret key in the targeted KEM be $\mathbf{s}$, we summarize our main procedure as follows:

Step 1: Perform **RoughKeyRecovery** in Algorithm 3 with $\mathcal{O}_{SCA}$ to get a roughly correct $\hat{\mathbf{s}}$.

Step 2: Based on the knowledge of $\hat{\mathbf{s}}$, we apply a fast checking method to detect the errors.

Step 3: For all suspicious blocks we found in $\hat{\mathbf{s}}$, we perform targeted recollection on them. Finally, we update the coefficient blocks in $\hat{\mathbf{s}}$ and get the final $\hat{\mathbf{s}}_\mathbf{f}$.

The major advantage of our *full-key recovery strategy* is that we can make best use of current information, since a great number of coefficients is correct by performing *ideal key recovery method* once.

The major advantage of our full-key recovery strategy is that we make the best use of current information of the roughly correct $\hat{\mathbf{s}}$, and just perform recollection for the suspicious coefficients. During this procedure, we need a *checking* method to construct special ciphertexts to distinguish the locations of error coefficients. A *checking* method should meet the following requirements: Firstly, the *checking* method should be fast and

requires as few queries as possible. Secondly, different from the previous **KeyRecovery** methods, the *checking* method just needs to detect the positions of error coefficients rather than correcting these erroneous coefficients.

In the following, we propose our fast checking method in detail, which plays a central role in our full-key recovery.

3.2 Our fast checking method

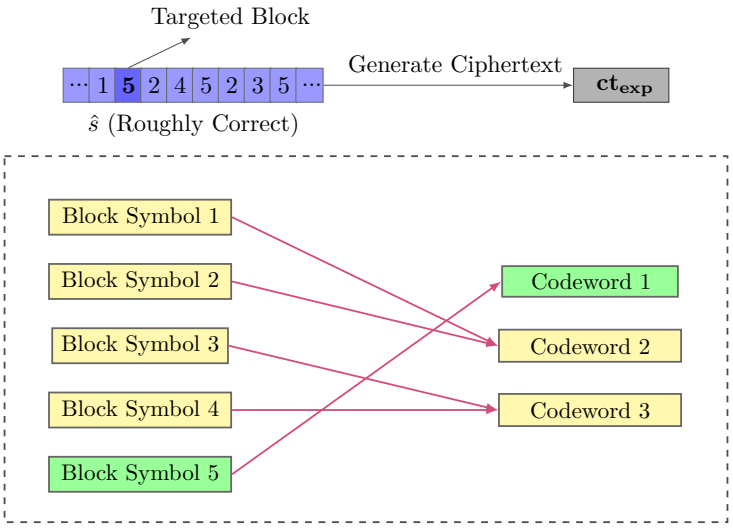

Figure 2: The expected coding for $\mathbf{ct_{exp}}$

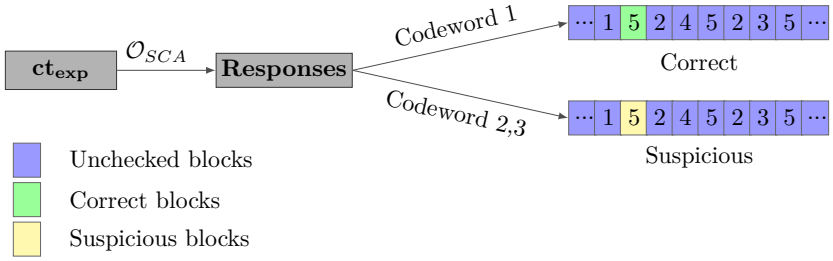

Figure 3: Procedure of checking the targeted block

The main idea of our fast checking method is to treat the detection of corruptions in $\hat{\mathbf{s}}$ as a coding problem. First, we divide coefficients of $\hat{\mathbf{s}}$ into blocks, while each block may contain one or several coefficients. Then, we treat the possible values of a coefficient block as symbols and the oracle responses (*the 0,1 sequence*) as codewords. To successfully check the targeted block, we generate proper ciphertexts to encode the targeted block symbol to a special codeword $\mathbf{c_{succ}}$. At the same time encoding the remaining blocks to other different codewords. As shown in Figure 2, the block symbols can be numbers between 1 and 5. To check block symbol 5, we expect a $\mathbf{ct_{exp}}$ to encode it into a special "Codeword 1", while other block symbols are encoded into "Codeword 2" or "Codeword 3".

By setting the oracle responses we collect in checking procedure as $\mathbf{c_{get}}$, we get:

$$\begin{cases} \text{there are } \textbf{no errors} \text{ in the block}, & \mathbf{c_{get}} = \mathbf{c_{succ}}, \\ \text{there are } \textbf{errors} \text{ in the block}, & \mathbf{c_{get}} \neq \mathbf{c_{succ}}. \end{cases}$$

Figure 3 depicts the procedure of checking the targeted block with symbol 5. First, we query the oracle with $\mathbf{ct_{exp}}$ and analyze the response sequence. When we receive "Codeword 1", we set the targeted block as correct. Otherwise, we mark the block as suspicious. By continuously querying the oracle, we can check the whole recovered $\hat{\mathbf{s}}$.

We have described the idea and general framework of the new checking method. The main difficulty is to design properly chosen ciphertexts as inputs to the PC oracle to produce the required responses (or codewords). We give examples of the challenges of generating properly chosen ciphertexts for Kyber512 in the following subsection.

3.3 Challenges on generating proper ciphertexts

In Table 2, we present two examples of how to design codewords for Kyber512, considering $\mathbf{BlockLen} = 1$. Here **BlockLen** represents the number of coefficients in a block. In Kyber512, the coefficients are generated from the interval $[-3, 3]$. We intend to distinguish a coefficient value **1** from other coefficient values using one query. As shown in **Example 1**, we require a coding scheme to uniquely map **1** to the codeword "0" while other values are mapped to the codeword "1". We next show that it is impossible to find qualified ciphertexts for such a coding scheme.

Table 2: Designed codewords to distinguish coefficient **1** from other coefficients

	-3	**-2**	**-1**	**0**	**1**	**2**	**3**
Example 1	"1"	"1"	"1"	"1"	"0"	"1"	"1"
Example 2	"11"	"11"	"11"	"11"	"01"	"00"	"00"

To be specific, **Example 1** requires us to find a ciphertext **ct** with which the resulted oracle response belongs to $\{1, 1, 1, 1, 0, 1, 1\}$ for coefficients in $[-3, 3]$. To give a more general result, we need the following Lemma.

Lemma 1. *In a PC oracle-based attack against Kyber, when* $\mathbf{BlockLen} = 1$*, it holds that* $|\mathbf{u}_\mathbf{0}'[0]| \leq \frac{q}{4\eta_1}$.

Proof. In a PC oracle-based attack against Kyber, the decrypted message $\mathbf{m}'$ is either $(1, 0, \cdots, 0)$ or $(0, 0, \cdots, 0)$. Thus, $\mathbf{m}'[1] = \left\lceil \frac{2}{q}(\mathbf{v}'[1] - t\mathbf{u}_\mathbf{0}'[0]) \right\rfloor \bmod 2 = 0, \forall t \in [-\eta_1, \eta_1]$.

Taking $t_1, t_2 \in [-\eta_1, \eta_1]$ where $t_1 < t_2$, we have

$$\frac{1}{2} + 2k_1 > \frac{2}{q}\left(\mathbf{v}'[1] - t_1\mathbf{u}_\mathbf{0}'[0]\right) \geq -\frac{1}{2} + 2k_1, \tag{8}$$

$$\frac{1}{2} + 2k_2 > \frac{2}{q}\left(\mathbf{v}'[1] - t_2\mathbf{u}_\mathbf{0}'[0]\right) \geq -\frac{1}{2} + 2k_2. \tag{9}$$

In Kyber, $\mathbf{u}_\mathbf{0}'[0]$ is taken from $\left\lceil \frac{q}{2^{d_u}} \cdot x \right\rfloor$ with $x \in [-\frac{2^{d_u}}{2}, \frac{2^{d_u}}{2})$. To prove our result, we firstly assume $\mathbf{u}_\mathbf{0}'[0] \geq 0$, and the remaining case can be proved similarly. From (8) and (9), we can infer that

$$\frac{1}{2} + 2k_1 > -\frac{1}{2} + 2k_2 \Rightarrow k_1 - k_2 > -\frac{1}{2} \Rightarrow k_1 - k_2 \geq 0 \tag{10}$$

We also know that

$$\frac{2}{q}(t_2 - t_1)\mathbf{u}_\mathbf{0}'[0] > 2(k_1 - k_2) - 1, \tag{11}$$

$$\frac{2}{q}(t_2 - t_1)\mathbf{u}_\mathbf{0}'[0] < 1 + 2(k_1 - k_2), \tag{12}$$

Since inequality (11) holds for all t_1, t_2, we can simply let $t_2 - t_1 = 1$. Recall that $\mathbf{u'_0}[0] \leq \left\lceil \frac{q}{2^{d_u}}(\frac{2^{d_u}}{2} - 1) \right\rfloor$, we have

$$1 > \frac{2}{q}\left\lceil \frac{q}{2} - \frac{q}{2^{d_u}} \right\rfloor > \frac{2}{q}(t_2 - t_1)\mathbf{u'_0}[0] > 2(k_1 - k_2) - 1. \tag{13}$$

That is, $1 > k_1 - k_2$. Together with (10), and the fact that k_1, k_2 are integers, we know that $k_1 = k_2$. Moreover, from (12), we conclude

$$\frac{2}{q}(t_2 - t_1)\mathbf{u'_0}[0] < 1 \tag{14}$$

To make sure that (14) is true for all t_1, t_2, since $t_2 - t_1 \leq 2\eta_1$, $\mathbf{u_0}'[0]$ should satisfy $\mathbf{u_0}'[0] \leq \frac{q}{4\eta_1}$. When $\mathbf{u'_0}[0] < 0$, we can similarly prove that $|\mathbf{u'_0}[0]| \leq \frac{q}{4\eta_1}$. □

Lemma 2. *In a PC oracle-based attack against Kyber, there is less than 2 flips in the designed codeword when* **BlockLen** $= 1$*, i.e., the designed codeword should be either of the form* $(1, \cdots, 1, 0, \cdots, 0)$ *or* $(0, \cdots, 0, 1, \cdots, 1)$*.*

Proof. Let $f(t) = \mathbf{m}'[0] = \left\lceil \frac{2}{q}(\mathbf{v}'[0] - t\mathbf{u'_0}[0]) \right\rfloor \bmod 2$, then $f(t) = 0$ or 1 corresponds to the case that the oracle outputs 0 or 1, respectively. Let t_1, t_2, t_3 denote three possible coefficient values, without loss of generality, let $-\eta_1 \leq t_1 < t_2 < t_3 \leq \eta_1$. We first prove that it is impossible to find ciphertexts which simultaneously satisfy $f(t_1) = 0, f(t_2) = 1$ and $f(t_3) = 0$. From the definition of $f(t)$, the following inequalities hold:

$$\frac{1}{2} + 2k_1 > \frac{2}{q}\left(\mathbf{v}'[0] - t_1\mathbf{u'_0}[0]\right) \geq -\frac{1}{2} + 2k_1, k_1 \in \mathbb{Z}, \tag{15}$$

$$\frac{3}{2} + 2k_2 > \frac{2}{q}\left(\mathbf{v}'[0] - t_2\mathbf{u'_0}[0]\right) \geq \frac{1}{2} + 2k_2, k_2 \in \mathbb{Z}, \tag{16}$$

$$\frac{1}{2} + 2k_3 > \frac{2}{q}\left(\mathbf{v}'[0] - t_3\mathbf{u'_0}[0]\right) \geq -\frac{1}{2} + 2k_3, k_3 \in \mathbb{Z}. \tag{17}$$

We prove the case when $\mathbf{u'_0}[0] \geq 0$, and the case $\mathbf{u'_0}[0] < 0$ can be proved similarly. Since $t_1 < t_2 < t_3$, from (15) and (16), we have $\frac{1}{2} + 2k_1 > \frac{1}{2} + 2k_2$, which leads to $k_1 - k_2 > 0$. Similarly we can prove that $k_2 - k_3 \geq 0$. Thus, $k_1 - k_3 > 0$, which further induces $k_1 - k_3 \geq 1$ since k_1 and k_3 are integers.

Next, from (15) and (17), we know that

$$\frac{2}{q}\left(\mathbf{v}'[0] - t_1\mathbf{u'_0}[0]\right) - \frac{2}{q}\left(\mathbf{v}'[0] - t_3\mathbf{u'_0}[0]\right) > -\frac{1}{2} + 2k_1 - (\frac{1}{2} + 2k_3). \tag{18}$$

That is, $\frac{2}{q}\left(t_3 - t_1\right)\mathbf{u'_0}[0] > 2(k_1 - k_3) - 1 \geq 1$. Or equivalently, $2\eta_1 \geq (t_3 - t_1) > \frac{q}{2\mathbf{u'_0}[0]}$. Thus, $\mathbf{u'_0}[0] > \frac{q}{4\eta_1}$, a contradiction to the result in Lemma 1.

Similarly, we can prove that it is impossible to find ciphertexts satisfying $f(t_1) = 1, f(t_2) = 0$ and $f(t_3) = 1$. This completes our proof. □

The coding scheme for **Example 2** is equivalent to finding two ciphertexts resulting in $\{1, 1, 1, 1, 0, 0, 0\}$ and $\{1, 1, 1, 1, 1, 0, 0\}$, respectively, and through similar analysis such ciphertexts are available. However, as the block length increases, it is quite hard to find qualified ciphertexts directly from the initially designed codewords. Hence, we need to find candidate ciphertexts and try to generate qualified codewords from them.

We could use a computer to search for chosen ciphertexts achieving required good encoding patterns. The main steps are summarized in the following.

Step 1: For a concrete scheme and **BlockLen**, find the PC oracle restrictions.

Step 2: Generate enough candidate ciphertexts and store them to **ct_list**. For all pairs of ciphertexts in **ct_list**, calculate the response sequences for all block symbols and store them in **c_list**. Try to find the special codeword $\mathbf{c_{succ}}$ from **c_list**.

4 Our adaptive full-key recovery for Kyber

In this section, we select the proper parameters to launch a practical PC oracle-based full-key recovery against Kyber512. We first discuss how to find restrictions on generating qualified ciphertexts. Then, we show how to check 4 coefficients by 2 queries, which is the most efficient instantiation of the new attack that we could achieve.

4.1 Restrictions on generating qualified ciphertexts

In Subsection 3.3, we have shown that it is impossible to find qualified ciphertexts for Kyber512, which could check a coefficient by using 1 query (i.e. $\mathbf{BlockLen} = 1$). Although we can achieve checking a coefficient by using 2 queries, the resulted number of queries for checking is $512 \cdot 2 = 1024$, which is unacceptable. Therefore, we turn to the cases $\mathbf{BlockLen} \geq 2$ trying to check more coefficients by 2 queries. But in practice, the possible **BlockLen** is very limited since searching for qualified ciphertexts with $\mathbf{BlockLen} >= 5$ is rather hard. Among these limited options (i.e. $\mathbf{BlockLen} = 2, 3, 4$), a larger **BlockLen** means fewer total queries. Therefore, in the following we turn to $\mathbf{BlockLen} = \mathbf{4}$, which is our best result and on the basis of the case $\mathbf{BlockLen} = \mathbf{2}$.

We also take the recovery of coefficients in $\mathbf{s_0}$ as an example. We need to determine attacking parameters such as $\mathbf{u_{atk}}$ and $\mathrm{v_{atk}}$, where $\mathbf{u_{atk}}$ is an array with **BlockLen** integers to craft $\mathbf{u}$ and $\mathrm{v_{atk}}$ is an integer to craft $\mathbf{v}$.

When $\mathbf{BlockLen} = 2$, the attacker also sets $\mathbf{m} = (1, 0, \cdots, 0)$, $\mathbf{u} = (\mathbf{u_0}, \mathbf{u_1})$, and $\mathbf{v} = (\mathrm{v_{atk}}, 0, \ldots, 0)$. Here $\mathbf{u_0} = \mathbf{u_{atk}}[0] - \mathbf{u_{atk}}[1] \cdot x^{255}$ and $\mathbf{u_1} = \mathbf{0}$. Again, similar to the analysis in subsection 2.2, in a PC oracle the decrypted message $\mathbf{m'}$ is either $(1, 0, \cdots, 0)$ or $(0, 0, \cdots, 0)$. That is, for $i \in [1, 255]$, $\mathbf{m'}[i] = 0$. Recall that

$$\mathbf{m'}[i] = \left\lceil \frac{2}{q}(-\mathbf{s_0}[i]\mathbf{u_{atk}}[0] - \mathbf{s_0}[i+1]\mathbf{u_{atk}}[1]) \right\rfloor \bmod 2, \ i \in [1, 255]. \tag{19}$$

Since $(\mathbf{s_0}[i], \mathbf{s_0}[i+1]) \in [-3, 3] \times [-3, 3]$, to achieve $\mathbf{m'}[i] = 0$, we choose to set

$$0 \leq \frac{2}{q}(3\mathbf{u_{atk}}[0] + 3\mathbf{u_{atk}}[1]) < \frac{1}{2} \quad \text{i.e. } 0 \leq \mathbf{u_{atk}}[0] + \mathbf{u_{atk}}[1] < \frac{q}{12}. \tag{20}$$

When $\mathbf{BlockLen} = 4$, similarly we can find the restrictions

$$0 \leq \mathbf{u_{atk}}[0] + \mathbf{u_{atk}}[1] + \mathbf{u_{atk}}[2] + \mathbf{u_{atk}}[3] < \frac{q}{12}. \tag{21}$$

Meanwhile, according to the decryption procedure of Kyber (line 2 in Algorithm 1), for $i \in [0, \mathbf{BlockLen} - 1]$, $\mathbf{u_{atk}}[i]$ also needs to satisfy

$$\mathbf{u_{atk}}[i] = \left\lceil \frac{q}{1024} \cdot y_i \right\rfloor, \ y_i \in [0, 1023]. \tag{22}$$

Algorithm 4 depicts the ciphertext generation procedure with attacking parameters satisfying equations (20), (21), and (22). We use **sp** and **bp** to locate the targeted block, where $\mathbf{sp} = 0$ or 1 to indicate the recovery of $\mathbf{s_0}$ or $\mathbf{s_1}$, and **bp** represents the index of the beginning of the block.

4.2 Generating proper ciphertexts when $\mathrm{BlockLen} = 2$ or 4

When $\mathbf{BlockLen} = 2$, we take how to check $(\mathbf{s_0}[\mathbf{bp}], \mathbf{s_0}[\mathbf{bp}+1])$ as an example. As shown in Algorithm 4, by setting $\mathbf{ct}$ as $\mathbf{ct} = \mathbf{GeneCiphertext_{2,0,bp}}(\mathbf{u_{atk}}, \mathrm{v_{atk}})$, the attacker knows the following relationship:

$$\mathcal{O}(\mathbf{ct}) = \mathbf{m'}[0] = \left\lceil \frac{2}{q}(\mathrm{v_{atk}} - (\mathbf{s_0}[\mathbf{bp}]\mathbf{u_{atk}}[0] + \mathbf{s_0}[\mathbf{bp}+1]\mathbf{u_{atk}}[1])) \right\rfloor \bmod 2.$$

Algorithm 4 Generating candidate ciphertexts from given parameters

Input: $\mathbf{sp} = 0$ or 1 for recovering $\mathbf{s_0}$ or $\mathbf{s_1}$ in $\mathbf{s}$, $\mathbf{bp}$ = index of the beginning of the block
Input: Attacking parameters $\mathbf{u_{atk}}, \mathrm{v_{atk}}$
Output: Ciphertext $\mathbf{ct}$
◇ $\mathbf{GeneCiphertext_{BlockLen,sp,bp}}(\mathbf{u_{atk}}, \mathrm{v_{atk}})$
$(\mathbf{u_0}, \mathbf{u_1}) = (\mathbf{0}, \mathbf{0})$; $\mathbf{v} = \mathbf{0}$
if $\mathbf{bp} = 0$ **then**
  $\mathbf{u_{sp}} = \mathbf{u_{atk}}[0] - \sum_{i=1}^{\mathbf{BlockLen}-1} \mathbf{u_{atk}}[i] \cdot x^{256-i}$
else
  $\mathbf{u_{sp}} = -\sum_{i=0}^{\mathbf{BlockLen}-1} \mathbf{u_{atk}}[i] \cdot x^{256-\mathbf{bp}-i}$
end if
$\mathbf{v}[0] = \mathrm{v_{atk}}$
return $((\mathbf{u_0}, \mathbf{u_1}), \mathbf{v})$

Recall that both $\mathbf{s_0}[\mathbf{bp}]$ and $\mathbf{s_0}[\mathbf{bp}+1]$ lie in $[-3, 3]$. To check all 7×7 value of $(\mathbf{s_0}[\mathbf{bp}], \mathbf{s_0}[\mathbf{bp}+1])$, we also need 7×7 $(\mathbf{ct_1}, \mathbf{ct_2})$. In Algorithm 5 we express our brute force strategy as pseudocode and successfully find all of them.

Algorithm 5 Generating proper ciphertexts for "Checking" when $\mathbf{BlockLen} = 2$

Output: Parameters to check all possible block symbols, with $\mathbf{BlockLen} = 2$
for All available $\mathrm{v_{atk}}$ and $(\mathbf{u_{atk}}[0], \mathbf{u_{atk}}[1])$ satisfying Equations (20) and (22) **do**
  $\mathbf{ct} = \mathbf{GeneCiphertext}_{2,0,0}((\mathbf{u_{atk}}[0], \mathbf{u_{atk}}[1]), \mathrm{v_{atk}})$
  Append $\mathbf{ct}$ to $\mathbf{ct\_list}$
end for
for All pairs $(\mathbf{ct_1}, \mathbf{ct_2})$ in $\mathbf{ct\_list}$ **do**
  Calculate the response sequence $\mathcal{O}(\mathbf{ct_1})||\mathcal{O}(\mathbf{ct_2})$ for all block symbols and store to $\mathbf{c\_list}$
  if Special codeword $\mathbf{c_{succ}}$ is found from $\mathbf{c\_list}$ **then**
    Append the corresponding block symbols, $(\mathbf{ct_1}, \mathbf{ct_2})$ and $\mathbf{c_{succ}}$ to **result**
  end if
end for
return result

Suppose we have known the roughly correct $\hat{\mathbf{s}} = (\hat{\mathbf{s}}_\mathbf{0}, \hat{\mathbf{s}}_\mathbf{1})$, and we want to check whether $(\hat{\mathbf{s}}_0[0], \hat{\mathbf{s}}_0[1])$ is equal to $(0, 2)$ or not. We execute the following steps:

1. Query the oracle with $\mathbf{ct_1} = \mathbf{GeneCiphertext}_{\mathbf{2,0,0}}((32, 48), 5)$ and store the response $\mathcal{O}(\mathbf{ct_1})$.

2. Query the oracle with $\mathbf{ct_2} = \mathbf{GeneCiphertext}_{\mathbf{2,0,0}}((22, 43), 5)$ and store the response $\mathcal{O}(\mathbf{ct_2})$. In Table 3, we list all the response sequences $\mathcal{O}(\mathbf{ct_1})||\mathcal{O}(\mathbf{ct_2})$ for all possible block symbols. Here we can see that $(0, 2)$ corresponds to a special codeword "01", while other block symbols are encoded to "11" or "00".

3. Since $\mathbf{c_{succ}}$ = "01", it holds that

$$(\mathbf{s_0}[0], \mathbf{s_0}[1]) = \begin{cases} (0, 2)\ , & \mathbf{c_{get}} = \text{"01"}\ , \\ others, & \mathbf{c_{get}} = others. \end{cases}$$

To accelerate the checking process, we increase the $\mathbf{BlockLen}$ to be 4. Then, we show how to check 4 coefficients with 2 queries, which can significantly improve the efficiency. Similarly, by setting $\mathbf{ct}$ as $\mathbf{ct} = \mathbf{GeneCiphertext}_{\mathbf{4,0,bp}}(\mathbf{u_{atk}}, \mathrm{v_{atk}})$, the attacker knows

Table 3: $\mathcal{O}(\mathbf{ct_1})||\mathcal{O}(\mathbf{ct_2})$ for $[-3,3]\times[-3,3]$

$s_0[0]\backslash s_0[1]$	*-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"

the following equation:

$$\mathcal{O}(\mathbf{ct}) = \left\lceil \frac{2}{q}(\mathrm{v_{atk}} - (\sum_{n=0}^{3} \mathbf{s_0}[\mathbf{bp}+n]\mathbf{u_{atk}}[n])) \right\rceil \bmod 2. \tag{23}$$

To check all 7^4 values of $(\mathbf{s_0}[\mathbf{bp}], \mathbf{s_0}[\mathbf{bp}+1], \mathbf{s_0}[\mathbf{bp}+2], \mathbf{s_0}[\mathbf{bp}+3])$, we also need to generate proper 7^4 ciphertext pairs $(\mathbf{ct_1}, \mathbf{ct_2})$. However, the computational complexity can be quite high if we apply a similar brute-force in Algorithm 5. After carefully analyzing the table of oracle responses and the corresponding ciphertext $\mathbf{ct}$, we observe that the problem can be solved by using the idea of divide-and-conquer. That is, we first get the needed attacking parameter $\mathbf{u_{atk}}[0], \mathbf{u_{atk}}[1], \mathrm{v_{atk}}$ using Algorithm 5. Then, with the found $\mathbf{u_{atk}}[0], \mathbf{u_{atk}}[1], \mathrm{v_{atk}}$, we try to find proper $\mathbf{u_{atk}}[2], \mathbf{u_{atk}}[3]$ satisfying Eq. (21). Next we select those $(\mathbf{u_{atk}}[0], \mathbf{u_{atk}}[1], \mathbf{u_{atk}}[2], \mathbf{u_{atk}}[3], \mathrm{v_{atk}})$ which could generate the special code we want. Our main process is given in Algorithm 6.

Algorithm 6 Generating proper ciphertexts for "Checking" when **BlockLen** = 4

Output: Parameters to check all possible block symbols, with **BlockLen** = 4

Run Algorithm 5 to get **result_bl2**
for Every tuple $(\mathbf{ct_1}, \mathbf{ct_2})$ in **result_bl2 do**
  for All available extra $(\mathbf{u_{atk}}[2], \mathbf{u_{atk}}[3])$ satisfying Equations (21) and (22) **do**
    $\mathbf{ct} = \mathbf{GeneCiphertext}_{4,0,0}((\mathbf{u_{atk}}[0], \mathbf{u_{atk}}[1], \mathbf{u_{atk}}[2], \mathbf{u_{atk}}[3]), \mathrm{v_{atk}})$
    Append **ct** to **ct_list**
  end for
  for All pairs $(\mathbf{ct_1}, \mathbf{ct_2})$ in **ct_list do**
    Calculate the response sequence $\mathcal{O}(\mathbf{ct_1})||\mathcal{O}(\mathbf{ct_2})$ for all symbols and store them in **c_list**
    if Special codeword $\mathbf{c_{succ}}$ is found from **c_list then**
      Append the corresponding block symbols, $(\mathbf{ct_1}, \mathbf{ct_2})$ and $\mathbf{c_{succ}}$ to **result**
    end if
  end for
end for
return result

For example, we want to check if $(\hat{\mathbf{s}}_\mathbf{0}[0], \hat{\mathbf{s}}_\mathbf{0}[1], \hat{\mathbf{s}}_\mathbf{0}[2], \hat{\mathbf{s}}_\mathbf{0}[3])$ equals to $(-2,-1,0,1)$. Then, we execute the following steps:

1. Query the oracle with $\mathbf{ct_1} = \mathbf{GeneCiphertext}_{4,0,0}((10,1,66,4),13)$ and store the response $\mathcal{O}(\mathbf{ct_1})$.
2. Query the oracle with $\mathbf{ct_2} = \mathbf{GeneCiphertext}_{4,0,0}((10,1,65,5),13)$ and store the response $\mathcal{O}(\mathbf{ct_2})$. In Table 4, we list codewords and their corresponding counts for all possible block symbols. We can see that $(-2,-1,0,1)$ corresponds to a special codeword "01", while other block symbols are encoded to "11", "00" or "01".

3. Since $\mathbf{c_{succ}}$ = "01", we have

$$(\mathbf{s_0}[0], \mathbf{s_0}[1], \mathbf{s_0}[2], \mathbf{s_0}[3]) = \begin{cases} (-2, -1, 0, 1)\ , & \mathbf{c_{get}} = \text{"01"}\ , \\ others, & \mathbf{c_{get}} = others. \end{cases}$$

Table 4: $\mathcal{O}(\mathbf{ct_1})||\mathcal{O}(\mathbf{ct_2})$ for $[-3, 3]^4$

Codewords	"00"	**"01"**	"10"	"11"
Counts	1532	**1**	9	859

Finally, we are able to find 2382 $(\mathbf{ct_1}, \mathbf{ct_2})$ sequences, and thus, we can check 2382 block symbols in all 7^4 (= 2401) possibilities. For the remaining blocks in which symbols cannot respond to available $(\mathbf{ct_1}, \mathbf{ct_2})$, we can simply redivide these blocks. In this way, we can check all the coefficients. A natural question here is, can we go on increasing the block length to further improve the efficiency? The increment in the block length also brings much burden in finding proper ciphertexts, and the searching complexity for checking more than 4 coefficients (for instance **BlockLen** = 5) is already prohibitively high. We need to balance efficiency and searching complexity for ciphertexts.

In practice, due to different noise levels or distinguisher accuracy, the resulted oracle accuracy can be different. For example, in [UXT+22], Ueno et al. realized a side-channel distinguisher with a trained Neural-Network (NN) model. The model accuracy (also the oracle accuracy) for non-protected implementation is 0.998, and for masked implementation is 0.960. In the following, we propose our adaptive key recovery strategy, which can deal with different accuracy levels adaptively.

4.3 Our adaptive key recovery strategy for different accuracy levels

When the oracle accuracy $\mathcal{O}_{SCA}$ is high (such as 0.998), our full-key recovery strategy in Subsection 3.1 works well, but it may fail to achieve full-key recovery for lower oracle accuracy (such as 0.960). This is because the checking and recollection procedures are also done under the same SCA oracle, which may also be corrupted. The corruptions on these two procedures are negligible under a high oracle accuracy but need to be considered when the oracle has lower accuracy.

Algorithm 7 depicts our adaptive full-key recovery strategy for high and lower oracle accuracy, respectively. First, we use **RoughKeyRecovery** (in Algorithm 3) to get $\hat{\mathbf{s}}_\mathbf{f}$, which is then checked using **CheckSkByBlock**. Here we instantiate **CheckSkByBlock** with the proposed "Checking 4 coefficients by 2 queries" method in Subsection 4.2. Then, we employ $\mathbf{E_b}$ to record the blocks in which errors are found, and the number of errors is represented by $len(\mathbf{E_b})$. By setting the bound $\alpha_b = 0.990$, we can estimate the oracle accuracy α_o by $\alpha_o \approx 1 - len(\mathbf{E_b})/Q_{RR}$, where Q_{RR} represents the number of queries needed in **RoughKeyRecovery**. For Kyber512, we set $Q_{RR} = 1312$.

With the estimated α_o, we compare it with an accuracy bound α_b to determine the accurate level and turn to **FullKeyRecHighAccu** or **FullKeyRecLowerAccu** for high or lower accuracy conditions, respectively. In **FullKeyRecHighAccu**, we simply employ **CoefficientRecovery** (in Algorithm 2) to recover all the coefficients in $\mathbf{E_b}$.

In **FullKeyRecLowerAccu**, we introduce a *mixed voting* technique to extend our full-key recovery strategy, making it suitable for lower oracle accuracy. Our main idea is to make full use of the information in the checking and recollection phases. More specifically, we define an array **Confidence** to store the votes of every possible value $\mathbf{s_f}[loc]$ for every coefficient location *loc*. For the coefficient locations not in $\mathbf{E_b}$ (i.e. these blocks have been checked and regarded as correct in this round), we add $\mathbf{cc_1}$ votes to the values in

Algorithm 7 Our Adaptive Full-Key Recovery Strategy for Different Accuracy Level

◇ **AdaptiveFullKeyRec**
Input: An imperfect PC oracle $\mathcal{O}_{SCA}$
Input: Accuracy bound α_b
Output: Recovered secret key $\mathbf{s_f}$
Run Algorithm 6 to prepare the ciphertexts
$\mathbf{s_f} = \mathbf{RoughKeyRecovery}(\mathcal{O}_{SCA})$
$\mathbf{E_b} = \mathbf{CheckSkByBlock}(\mathbf{s_f})$
Estimate α_o of $\mathcal{O}_{SCA}$ from $len(\mathbf{E_b})$
if $\alpha_o \geq \alpha_b$ **then**
  goto FullKeyRecHighAccu
else
  goto FullKeyRecLowerAccu
end if

FullKeyRecHighAccu:
for each block $B \in \mathbf{E_b}$ **do**
  for each coefficient location loc in B **do**
    $\mathbf{s_f}[loc] = \mathbf{CoefficientRecovery}(loc)$
  end for
end for
return $\mathbf{s_f}$

FullKeyRecLowerAccu:
Precompute $\mathbf{cc_1}, \mathbf{cc_2}, \mathbf{Threshold}[\,]$ from α_o
$round = 0$, $\mathbf{Finished} = \{false\}$
while Not all $\mathbf{Finished}[loc] = true$ **do**
  for each block $B \notin \mathbf{E_b}$ **do**
    for each coefficient location loc in B **do**
      $\mathbf{Confidence}[loc][\mathbf{s_f}[loc]]$ += $\mathbf{cc_1}$
    end for
  end for
  for each block $B \in \mathbf{E_b}$ **do**
    for each coefficient location loc in B **do**
      $nv = \mathbf{CoefficientRecovery}(loc)$
      $\mathbf{Confidence}[loc][nv]$ += $\mathbf{cc_2}$
    end for
  end for
  $cct = \mathbf{Threshold}[round]$
  for each coefficient location loc in $\mathbf{s_f}$ **do**
    Set $vl = j$ with the largest $\mathbf{Confidence}[loc][j]$
    Update $\mathbf{s_f}[loc]$ with vl
    if $\mathbf{Confidence}[loc][vl] > cct$ **then**
      $\mathbf{Finished}[loc] = true$
    end if
  end for
  $\mathbf{E_b} = \mathbf{CheckSkByBlock}(\mathbf{s_f})$
  $round$++
end while
return $\mathbf{s_f}$

targeted positions. Otherwise, for the coefficient locations in $\mathbf{E_b}$, we run recollection (i.e. **CoefficientRecovery**) and cast $\mathbf{cc_2}$ votes for the recovered value from recollection. Note that in each round, we recheck $\mathbf{s_f}$ and get different $\mathbf{E_b}$, which means errors may be found in different blocks. Thus, for each coefficient location loc, there may exist different $\mathbf{Confidence}[loc][j]$ corresponding to it. We set vl as the j when $\mathbf{Confidence}[loc][j]$ is the largest, and then update $\mathbf{s_f}[loc]$ with vl. We also use an array **Finished** to record the status of each loc. When $\mathbf{Confidence}[loc][vl]$ reaches a bound $\mathbf{Threshold}[round]$, we set $\mathbf{Finished}[loc]$ as true. We repeat this procedure round by round until all the $\mathbf{Finished}[loc]$ labels are true.

For **FullKeyRecLowerAccu**, $\mathbf{cc_1}, \mathbf{cc_2}$, and $\mathbf{Threshold}[\,]$ can be determined in advance through simulations. That is, we randomly generate a set of 20 secret keys and select the best $\mathbf{cc_1}, \mathbf{cc_2}$ and $\mathbf{Threshold}[\,]$, which result in the lowest total number of traces. Table 5 gives the precomputed $\mathbf{cc_1}, \mathbf{cc_2}$, and $\mathbf{Threshold}[\,]$ for $\alpha_o = 0.960$, 0.950, and 0.900, respectively.

Table 5: The precomputed parameters for $\alpha_o = 0.960$, 0.950, and 0.900

	$\mathbf{cc_1}$	$\mathbf{cc_2}$	$\mathbf{Threshold}[\,]$
$\alpha_o = 0.960$	4	3	$\{5, 9, 11, 14, 14, 18, 18, 18, 18, 22, 22, ...\}$
$\alpha_o = 0.950$	4	3	$\{5, 9, 12, 13, 13, 16, 17, 17, 21, 21, 21, ...\}$
$\alpha_o = 0.900$	3	4	$\{5, 9, 13, 16, 20, 20, 20, 22, 24, 25, 26, ...\}$

5 Experiments and Analysis

In this section, we present the experimental results and analysis, including results from software simulations and also real-world experiments on an ARM Cortex-M4 microcontroller. Our aim is to understand the performance of the new algorithm, so we first set

several oracle accuracy levels and then run computer simulations to simulate the sample complexity under different accuracy levels. We last launch an EM attack to demonstrate that the simulation results match the real-world scenarios.

5.1 Software Simulations

5.1.1 Simulation settings

We firstly introduce our software simulations to show the efficiency of our full-key recovery method under practical SCA. Recall that $\mathcal{O}$ is the ideal PC oracle, to simulate the practical $\mathcal{O}_{SCA}$, we let the outputs of $\mathcal{O}_{SCA}$ equal the outputs of $\mathcal{O}$ with a probability α_o. Then the two outputs are different with $1-\alpha_o$ probability. That is,

$$\mathcal{O}_{SCA}(\mathbf{ct}) = \begin{cases} \mathcal{O}(\mathbf{ct}) \ , & \text{with } \alpha_{\mathrm{o}} \text{ probability,} \\ !\mathcal{O}(\mathbf{ct}) \ , & \text{with } (1-\alpha_{\mathrm{o}}) \text{ probability.} \end{cases}$$

Now the probability α_o represents the accuracy of the practical $\mathcal{O}_{SCA}$ in each query. In our experiments, we let $\alpha_o = 0.995, 0.950, 0.900$ to simulate different accuracy levels. Our method is adaptive since we first evaluate the accuracy level and compare it with a bound $\alpha_b = 0.990$. In case $\alpha_o = 0.995 > \alpha_b$, we set it as high oracle accuracy, and then use the **FullKeyRecHighAccu** in Algorithm 7 to launch the attack. In other cases, $\alpha_o = 0.950$ and $\alpha_o = 0.900$ simulate the conditions with lower accuracy, and we use the **FullKeyRecLowerAccu** in Algorithm 7 to launch the attack.

5.1.2 Traces evaluation under different accuracy levels

To show the advantage of our proposed full-key recovery strategy against majority-voting, we run tests with 10,000 randomly generated secret keys for Kyber512 and FireSaber. Saber is another third-round NIST candidate, which offers three security levels: LightSaber, Saber, and FireSaber [DKRV19]. FireSaber shares similar algorithm structure in using Compress/Decompress functions while without Encode/Decode functions. In addition, the coefficients of FireSaber are also selected from [-3,3]. Thus, the adversary can choose proper attacking parameters in an approach similar to the attack against Kyber512.

The codes for both Kyber512 and FireSaber are available at our given GitHub link. Our main target is still Kyber since it is the selected KEM algorithm to be standardized, and we include simulation results of attacking FireSaber to demonstrate that the new attack is generic.

We let **#TotalTrace** represent the average number of total traces for the full-key recovery. We also use **#ErrCof** to represent the average number of error coefficients in the finally recovered secret key. Both majority-voting and our proposed method aim to reduce **#ErrCof** to an amount less than 1.0. All the reported numbers are calculated by taking an average. As shown in Table 6, compared to majority-voting, our method on Kyber512 achieves similar **#ErrCof** but reducing $58.2\%, 57.8\%, 46.1\%$ of the total traces when $\alpha_o = 0.995, 0.950, 0.900$, respectively. For FireSaber, we can achieve fewer **#ErrCof** and reduce the number of traces by $51.2\%, 54.1\%$, and 40.4% on average, correspondingly, compared to majority-voting. These results fully show the efficiency of our method in a wide range of oracle accuracy.

5.1.3 Improving Ueno et al.'s work

In [UXT+22], Ueno et al. employed a side-channel distinguisher with trained Neural Network models. Their distinguisher corresponds to our imperfect oracle, and their model accuracy (also the oracle accuracy) for different implementations is given in Table 7. We can see that for non-protected software implementation, their distinguisher lies in the

Table 6: Comparison between majority-voting and our proposed method for full-key recovery under $\mathcal{O}_{SCA}$ ($\mathbf{t}$ represents the number of votes cast)

Accuracy	Schemes	Method	#TotalTrace	#ErrCof
$\alpha_0 = 0.995$	Kyber512	Majority Voting ($\mathbf{t} = 3$)	3936.5 (ref)	0.10/512
		Our Method	1645.1 (-58.2%)	0.51/512
	FireSaber	Majority Voting ($\mathbf{t} = 3$)	7871.7 (ref)	0.19/1024
		Our Method	3841.6 (-51.2%)	0.08/1024
$\alpha_0 = 0.950$	Kyber512	Majority Voting ($\mathbf{t} = 7$)	9185.0 (ref)	0.25/512
		Our Method	3874.5 (-57.8%)	0.15/512
	FireSaber	Majority Voting ($\mathbf{t} = 7$)	18367.1 (ref)	0.50/1024
		Our Method	8428.3 (-54.1%)	0.20/1024
$\alpha_0 = 0.900$	Kyber512	Majority Voting ($\mathbf{t} = 11$)	14433.3 (ref)	0.39/512
		Our Method	7773.9 (-46.1%)	0.25/512
	FireSaber	Majority Voting ($\mathbf{t} = 11$)	28862.0 (ref)	0.78/1024
		Our Method	17196.1 (-40.4%)	0.36/1024

Table 7: Accuracy of the NN side-channel distinguisher in [UXT+22]

	Non-protected software	Masked software
Accuracy	0.998	0.960

high accuracy region, while for masked implementation, their distinguisher achieves lower accuracy. To achieve full-key recovery, they use negative log likelihood (NLL) to enact a more accurate oracle. More specifically, they use NLL to merge 2 and 5 traces for an oracle response for the non-protected and masked software implementation, and the total number of the required traces for Kyber512 are $2 \cdot 1536 = 3072$ and $5 \cdot 1536 = 7680$, respectively. Similarly, for FireSaber, they need $2 \cdot 3072 = 6144$ and $5 \cdot 3072 = 15360$ traces on average, correspondingly.

Table 8: Comparisons between Ueno et al.'s work and our proposed method for full-key recovery under $\mathcal{O}_{SCA}$

Accuracy	Schemes	Method	#TotalTrace	#ErrCof
$\alpha_0 = 0.998$	Kyber512	Ueno et al.'s	3072.0 (ref)	0.00/512[2]
		Our Method	1663.3 (-45.9%)	0.04/512
	FireSaber	Ueno et al.'s	6144.0 (ref)	0.00/1024
		Our Method	3724.1 (-39.4%)	0.02/1024
$\alpha_0 = 0.960$	Kyber512	Ueno et al.'s	7680.0 (ref)	0.00/512
		Our Method	3424.9 (-55.4%)	0.05/512
	FireSaber	Ueno et al.'s	15360.0 (ref)	0.00/1024
		Our Method	6745.6 (-56.1%)	0.27/1024

To show that our method could further optimize Ueno et al.'s work, we use the software simulation mentioned above with $\alpha_o = 0.998$ and 0.960, and run tests with 10,000 randomly generated secret keys for Kyber512 and FireSaber, respectively. The result is averaged and given in Table 8. Compared to Ueno et al.'s results, our method on Kyber512 reduces $45.9\%, 55.4\%$ of the total traces when $\alpha_o = 0.998$ and 0.960, respectively. At the same

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

time, similar to the case in [UXT+22], we could recover nearly all the coefficients, i.e. on average only 0.04 or 0.05 error coefficients occur among the 512 coefficients. For FireSaber, we have successfully reduced the number of traces by 39.4% and 56.1% with $\alpha_o = 0.998$ and 0.960, respectively, compared to Ueno et al.'s results.

5.2 Real-world experiments

5.2.1 Experiment Setup

For real-world validation, as shown in Figure 4, we conduct our experiments on an STM32F407G board, which is equipped with an ARM Cortex-M4 microcontroller. We compile the ARM-optimized Kyber512 implementation from the *pqm4* library and run it on the microcontroller. The clock frequency of the target board is set to be 24 MHz. A PicoScope 3403D oscilloscope and a CYBERTEK EM5030-3 EM Probe are used to collect the EM traces. We connect the probe and the oscilloscope with a CYBERTEK EM5020A signal amplifier. The traces are sampled at 1 GHz.

To instantiate the PC oracle-based SCA distinguisher, we use the same method as that in [RRCB20], which is introduced in the first part of Subsection 2.2. The right part of Figure 4 shows one of the TVLA results between the $\mathbf{m_0}$ and $\mathbf{m_1}$ templates.

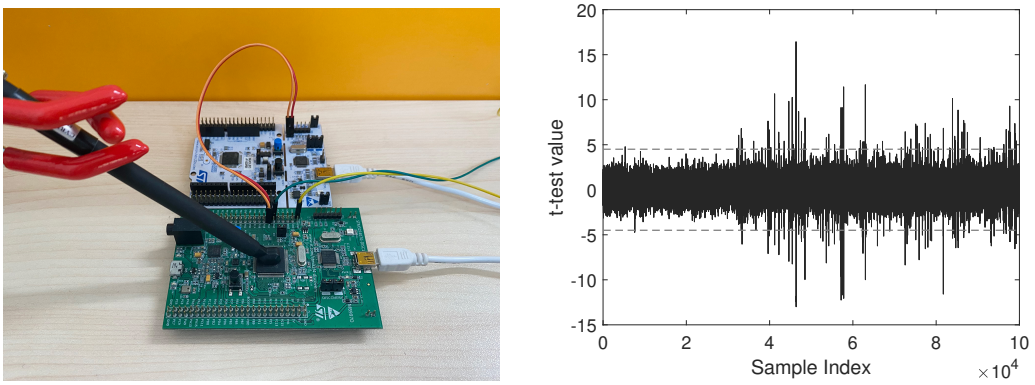

Figure 4: Our SCA equipment (left) and TVLA results between re-encryption of $\mathbf{m_0}$ and $\mathbf{m_1}$ (right)

5.2.2 Experimental Results

We run tests for both the majority-voting approach and our proposed full-key recovery strategy, with 10 randomly generated secret keys. We first implement the **RoughKeyRecovery** and **CheckSkByBlock** (i.e. the result of lines 1-2 in Algorithm 7), then we estimate the oracle accuracy from the erroneous block numbers. The estimated oracle accuracy in our SCA setup is 0.996, and thus we choose to use the **FullKeyRecHighAccu** version of our adaptive full-key recovery strategy.

The experimental results are averaged and shown in Table 9, where both methods lower the average error coefficients to an amount of less than 1.0. Note that under such an experimental setup, with neither the majority-voting nor the new attack strategy applied, we will expect around 4.1 error coefficients. We can see from the table that the real-world experimental results match the computer simulations very well (i.e., with a difference of less than 1%). Also, compared with the majority-voting approach, our new method reduces 58.9% of the required EM traces under our real-world setup.

Table 9: Comparison between majority-voting and our proposed method for full-key recovery in real-world experiments and simulations

	#TotalTrace (Real world)	**#ErrCof** (Real world)	**#TotalTrace** (Simulations)	**#ErrCof** (Simulations)
Majority Voting ($\mathbf{t} = 3$)	3943.5 (ref)	0.20/512	3936.5	0.06/512
Our Method	1618.9 (−58.9%)	0.40/512	1629.9	0.34/512

6 Conclusions

We have presented a novel adaptive approach for improving the sample complexity of the PC oracle-based CCA SCAs on lattice-based KEMs, when the constructed PC oracle is imperfect. The proposed framework consists of two variants targeting the low/high oracle-accuracy regions, respectively, and is instantiated on Kyber, the KEM algorithm selected to be standardized in the NIST PQC standardization project. Targeting Kyber512, we ran extensive computer simulations and also mounted an EM attack against the optimized implementation in the pqm4 library running on an STM32F407G board with an ARM Cortex-M4 microcontroller. The experimental results demonstrated that the new approach could provide a substantial gain in terms of the required number of traces.

Moreover, we ran computer simulations against Saber, another NIST third-round KEM finalist, to show that the new adaptive approach is generic and can be applied to other LWE/LWR-based KEMs. It is an interesting future direction to extend the new attack framework to NTRU-type KEMs.

We remark that, following previous research (e.g., [RRCB20, UXT+22]), our work determines the required number of traces to fully recover the secret entries. The sample complexity can be reduced if a certain amount of post-processing – such as key enumeration or lattice reduction – is allowed. The simplest strategy is to first recover a proportion of secret entries with side-channel leakages and then solve a new LWE problem with a smaller dimension using lattice reduction. Also, [DSDGR20] characterized side-channel leakages called "hints" that can be progressively integrated into lattice reduction to accelerate the secret-key recovery. It is interesting to combine the research advances in both the side-channel (such as our checking method) and the lattice-reduction steps and optimize attack strategies when the cost of post-processing is fixed.

Last, this work has made the known protections such as masking and shuffling more vulnerable since the adversary needs fewer attack traces. It is always appealing but challenging to design safer and more efficient countermeasures.

Acknowledgment

We would like to thank the reviewers for their valuable feedbacks. This work has been partly funded by the National Natural Science Foundation of China under Grant no. 62172374. Qian Guo was partly supported by the Swedish Research Council (Grants No. 2019-04166 and No. 2021-04602), by the Swedish Civil Contingencies Agency (Grants No. 2020-11632), and by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

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.

[ABH+22] Melissa Azouaoui, Olivier Bronchain, Clément Hoffmann, Yulia Kuzovkova, Tobias Schneider, and François-Xavier Standaert. Systematic study of decryption and re-encryption leakage: the case of kyber. Cryptology ePrint Archive, Report 2022/036, 2022. https://ia.cr/2022/036.

[ADPS16] Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. Post-quantum key exchange–a new hope. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 327–343, 2016.

[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.

[Ble98] Daniel Bleichenbacher. Chosen ciphertext attacks against protocols based on the rsa encryption standard pkcs# 1. In *Annual International Cryptology Conference*, pages 1–12. Springer, 1998.

[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.

[GHJ+22] Qian Guo, Clemens Hlauschek, Thomas Johansson, Norman Lahr, Alexander Nilsson, and Robin Leander Schröder. Don't reject this: Key-recovery timing attacks due to rejection-sampling in hqc and bike. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 223–263, 2022.

[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.

[HGS99] Chris Hall, Ian Goldberg, and Bruce Schneier. Reaction attacks against several public-key cryptosystems. In *ICICS*, 1999.

[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.

[LPR10] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 1–23. Springer, 2010.

[LS15] Adeline Langlois and Damien Stehlé. Worst-case to average-case reductions for module lattices. *Designs, Codes and Cryptography*, 75(3):565–599, 2015.

[MAA+20] Dustin Moody, Gorjan Alagic, Daniel C Apon, David A Cooper, Quynh H Dang, John M Kelsey, Yi-Kai Liu, Carl A Miller, Rene C Peralta, Ray A Perlner, et al. *Status Report on the Second Round of the NIST Post-Quantum Cryptography Standardization Process*. US Department of Commerce, National Institute of Standards and Technology, 2020. https://nvlpubs.nist.gov/nistpubs/ir/2020/NIST.IR.8309.pdf.

[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.

[MU10] Alfred Menezes and Berkant Ustaoglu. On reusing ephemeral keys in diffie-hellman key agreement protocols. *International Journal of Applied Cryptography*, 2(2):154–158, 2010.

[ND21] NIST and DHS. *Preparing for Post-Quantum Cryptography: Infographic*. 2021. https://www.dhs.gov/sites/default/files/publications/post-quantum_cryptography_infographic_october_2021_508.pdf.

[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*, 4:676–707, 2021.

[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.

[RBRC21] Prasanna Ravi, Shivam Bhasin, Sujoy Sinha Roy, and Anupam Chattopadhyay. On exploiting message leakage in (few) nist pqc candidates for practical message recovery attacks. *IEEE Transactions on Information Forensics and Security*, 2021.

[RDB+21] Prasanna Ravi, Suman Deb, Anubhab Baksi, Anupam Chattopadhyay, Shivam Bhasin, and Avi Mendelson. On threat of hardware trojan to post-quantum lattice-based schemes: A key recovery attack on SABER and beyond. In Lejla Batina, Stjepan Picek, and Mainack Mondal, editors, *Security, Privacy, and Applied Cryptography Engineering - 11th International Conference, SPACE 2021, Kolkata, India, December 10-13, 2021, Proceedings*, volume 13162 of *Lecture Notes in Computer Science*, pages 81–103. Springer, 2021.

[REB+22] Prasanna Ravi, Martianus Frederic Ezerman, Shivam Bhasin, Anupam Chattopadhyay, and Sujoy Sinha Roy. Will you cross the threshold for me? generic side-channel assisted chosen-ciphertext attacks on ntru-based kems. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 722–761, 2022.

[Reg09] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM (JACM)*, 56(6):1–40, 2009.

[RR21] Prasanna Ravi and Sujoy Sinha Roy. Side-channel analysis of lattice-based pqc candidates. NIST Round 3 Seminars - Post-Quantum Cryptography, 2021. https://csrc.nist.gov/CSRC/media/Projects/post-quantum-cryptography/documents/round-3/seminars/mar-2021-ravi-sujoy-presentation.pdf.

[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.

[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.

[ZCD21] Xiaohan Zhang, Chi Cheng, and Ruoyu Ding. Small leaks sink a great ship: an evaluation of key reuse resilience of pqc third round finalist ntru-hrss. In *International Conference on Information and Communications Security*, pages 283–300. Springer, 2021.