1

A Systematic Approach and Analysis of Key Mismatch Attacks on Lattice-Based NIST Candidate KEMs

Yue Qin[1,2,6], Chi Cheng[1,2,3,✉], Xiaohan Zhang[1], Yanbin Pan[4], Lei Hu[5], and Jintai Ding[6,7]

[1] China University of Geosciences, Wuhan, 430074, China
{chengchi}@cug.edu.cn
[2] State Key Laboratory of Cryptology, P.O. Box 5159, Beijing 100878, China,
[3] Guangxi Key Laboratory of Trusted Software, Guilin University of Electronic Technology, Guilin 541004, China.
[4] Key Laboratory of Mathematics Mechanization, Academy of Mathematics and Systems Science, Chinese Academy of Sciences
[5] State Key Lab of Information Security, Institute of Information Engineering, Chinese Academy of Sciences
[6] Ding Lab, Yanqi Lake Beijing Institute of Mathematical Sciences and Applications, Beijing, China
[7] Yau Mathematical Sciences Center, Tsinghua University

Abstract. Research on key mismatch attacks against lattice-based KEMs is an important part of the cryptographic assessment of the ongoing NIST standardization of post-quantum cryptography. There have been a number of these attacks to date. However, a unified method to evaluate these KEMs' resilience under key mismatch attacks is still missing. Since the key index of efficiency is the number of queries needed to successfully mount such an attack, in this paper, we propose and develop a systematic approach to find lower bounds on the minimum average number of queries needed for such attacks. Our basic idea is to transform the problem of finding the lower bound of queries into finding an optimal binary recovery tree (BRT), where the computations of the lower bounds become essentially the computations of a certain Shannon entropy. The optimal BRT approach also enables us to understand why, for some lattice-based NIST candidate KEMs, there is a big gap between the theoretical bounds and bounds observed in practical attacks, in terms of the number of queries needed. This further leads us to propose a generic improvement method for these existing attacks, which are confirmed by our experiments. Moreover, our proposed method could be directly used to improve the side-channel attacks against CCA-secure NIST candidate KEMs.

⋆ Chi Cheng, Xiaohan Zhang, Yanbin Pan, Lei Hu, Jintai Ding, A Systematic Approach and Analysis of Key Mismatch Attacks on Lattice-Based NIST Candidate KEMs, in proceedings of ASIACRYPT 2021, Tibouchi and H. Wang (Eds.): LNCS 13093, pp. 92121, 2021. https://doi.org/10.1007/978-3-030-92068-5_4

1 Introduction

The Diffie-Hellman (DH) key exchange [24] and its Elliptic Curve counterpart have played a fundamental role in many standards, such as Transport Layer Security (TLS) and IP security (IPSec), securing communications over the Internet. However, these public key primitives based on number theoretic problems would be broken if quantum computers become practical. Due to the rapid progresses in quantum technology [32], the transition from the currently used public key cryptographic blocks to their post-quantum counterparts has become urgent.

Since February 2016, NIST has begun the call for post-quantum cryptographic algorithms from all over the world [44]. The goal of post-quantum cryptography standardization is to establish cryptographic systems that are secure against both quantum and classical computers, integrating with existing communication protocols and networks [19]. There are 17 public key encryption (PKE) or key encapsulation mechanism (KEM) candidates in the second round [2], among which 9 are based on lattices [1]. On the third-round list, there are still 3 lattice-based KEMs out of the 4 finalists [45].

Most of these candidates follow a similar structure: First a chosen-plaintext attack (CPA) secure construction is proposed, and then it is converted into a chosen-ciphertext attack (CCA) secure one using some transformation such as the Fujisaki-Okamoto (FO) transformation [29]. We have to point out that there is no security guarantee on the CPA secure ones when the public key is reused. However, first, it is an important part of the cryptographic assessment of these candidates to understand their key-reuse resilience in even misuse situations. Secondly, all LWE-based KEMs in Rounds 2 and 3 of the NIST standardization use an FO transform to achieve IND-CCA security. By doing so, the private key security is provided for only one party, while the other party is required that his secret key should be fully disclosed. What's more, the full re-encryption in the FO transform is typically the main cost during decapsulation, which makes it less efficient than the IND-CPA version. To improve the efficiency, there have been many efforts in designing various authenticated key exchanges using the CPA version without FO transform. In these cases, key reuse is no doubt essential. Therefore, analysis of the key reuse resilience of these CPA-secure schemes makes sense. Finally, as shown in [23,50], side-channel information can be employed to successfully mount similar chosen-ciphertext attacks against the CCA-secure ones in an efficient way. Therefore, the line of research focusing on the key reuse attacks against the CPA secure ones is important and has been actively studied.

Research on the security of IND-CPA secure public-key cryptosystem in the case of key reuse can be dated back to 1998, when Bleichenbacher considered the security of the RSA PKCS#1 [15]. After that, similar attacks have been proposed against several public key cryptosystems including the Diffie-Hellman key exchange [33,43]. There are two kinds of key reuse attacks against lattice-based key exchange. One is the signal leakage attack, which employs the additional signal information in the shared key reconciliation between two parties. The other key reuse attack is called key mismatch attack, which launches the attack by simply knowing whether the shared two keys match or not. In

[25], Ding, Alsayigh and Saraswathy first launched signal leakage attacks to the key exchange protocol in [28] by using the leaked information about the secret key from the signal messages. Then, a signal leakage attack is proposed in [39] against the reconciliation-based NewHope-Usenix protocol [6]. Just recently in [14], Bindel, Stebila and Veitch proposed an improved signal leakage attack and further showed how to apply their method to an authenticated scheme in [26].

The idea of key mismatch attack on lattice-based key exchange is first proposed by Ding, Fluhrer and Saraswathy [27] against the one-pass case of the protocol in [28]. In a key mismatch attack, a participant's public key is reused and its private key is recovered by comparing whether the shared keys between two participants match or not. In [10], Bauer et al. proposed a key mismatch attack against NewHope KEM [3], which is further analyzed and improved by Qin, Cheng, and Ding [48]. In [46], Okada, Wang, and Takagi improved the method in [48] to further reduce the number of queries. The work of [49] gave a similar key mismatch attack on Kyber. In [31] a key mismatch attack was proposed against LAC, requiring up to 8 queries for each coefficient. Recently, Zhang et al. proposed an efficient method to launch key mismatch attacks on NTRU-HRSS [55], which can recover the complete secret key with a probability of 93.6%.

Although there have been a number of key reuse attacks on the lattice-based key exchange schemes, a fundamental problem is still open: Can we find a unified method to evaluate the key reuse resilience of NIST candidates against key mismatch attacks? Since the key index of the efficiency of these attacks is the number of queries (matches and mismatches) needed to successfully mount such attacks, a unified method to find bounds with fewest queries for all the candidates is appealing. In Eurocrypt 2019, Bǎetu et al. tried to answer this problem, but most of their result is related to a limited number of the first-round candidates which did not enter into the second round [9]. In a recent work of Huguenin-Dumittan and Vaudenay [37], they proposed similar key mismatch attacks on only some of the lattice-based second-round candidates, Kyber-512, LAC-128, LightSaber, Round5 (HILA5 [11]) and Frodo640. But no unified theoretical bound is given in their work. Therefore, a big picture about the evaluation of key reuse resilience of these candidates is still missing.

Contributions. In this paper, we propose and develop a systematic approach to find the lower bounds on the minimum average number of queries needed for mounting key mismatch attacks, which further motivates us to propose a generic improvement method that is not only suitable for CPA-secure KEMs, but also for side-channel attacks against CCA-secure KEMs. The main contributions of this paper include:

- We propose a unified method to find lower bounds for all the lattice-based NIST candidate KEMs. Our basic idea is to convert the problem into finding an optimal binary recovery tree (BRT). By using the technique of Huffman coding, we successfully build the optimal BRT and get the bounds. Further analysis shows that the calculation of these bounds becomes essentially the computation of a certain Shannon entropy, which means that on average one

cannot find a better attack with fewer queries than our bound in the full key recovery.
- According to our proposed bound, in terms of number of needed queries, there is still a huge gap between the bound and practical attacks against some candidates such as NewHope, FrodoKEM, and Saber [46,37,48]. The introduction of the optimal BRT approach enables us to understand causes of these gaps, guiding us to select proper parameters to improve the practical attacks. Compared to the existing results in [37] and [46], we have improved attacks against Frodo640 and LightSaber with 71.99% and 27.93% reduced number of queries respectively, which is also confirmed by our experiments.
- Our improved method could be directly used to further optimize the efficiency of side-channel attacks against CCA-secure NIST candidate KEMs. For example, we can reduce the needed number of queries (or traces) from 2560 to 1183 for Kyber512.
- From the analysis of our proposed attacks, we find that the ranges of the coefficients in the secret key and the corresponding occurrence probabilities, as well as the employment of Encode/Decode functions are the three most important factors in evaluating their key reuse resilience. More specifically, larger ranges of the coefficients increase the needed number of queries. On the other side, encoding/decoding several coefficients at one time reduces the number of queries needed.

2 Preliminaries

2.1 Lattice-based key encapsulation mechanisms

In [21], Cramer and Shoup introduced the notion of KEM. Generally, a KEM consists of three algorithms: a probabilistic polynomial-time (PPT) key generation algorithm KEM.Gen, a PPT encryption algorithm KEM.Enc, and a deterministic polynomial-time decryption algorithm KEM.Dec.

The main difficulty in constructing a lattice-based DH-like key exchange protocol is how to effectively reconcile errors to negotiate a consistent shared key. In [28], Ding, Xie, and Lin first proposed a "robust extractor" to reconcile the errors, in which one of the participants needs to send an additional signal message to the other party, so that the two participants can agree on a shared key. Ding, Xie, and Lin's schemes base their security on the Learning with Errors (LWE) problem and Ring LWE problem. The latter can be seen as the polynomial version of the former. In [47] Peikert proposed a KEM using a similar error correction mechanism, and then in [17] the reformulated key exchange proposed by Bos et al. has been integrated into TLS. More and more lattice-based KEMs have been proposed since then. For example, in NIST's second-round list, there are FrodoKEM [4], NewHope [5,3], LAC [40], Kyber [16,7], Threebears [34], Round5 [8], Saber [22], NTRU [18] and NTRU Prime [13]. Recently, NIST [45] has announced the third-round finalists, among which the lattice-based KEMs include Kyber, NTRU and Saber. NIST also announced two alternate lattice-based candidates: FrodoKEM and NTRU Prime.

We can roughly divide the existing lattice-based KEMs into two categories. The first category is in line with the work of Regev [51], Lyubashevsky-Peikert-Regev [41], and lattice-based key exchange scheme proposed by Ding, Xie and Lin [28]. The other is NTRU [35] and NTRU Prime [12].

In Fig. 1 we present the meta structure of the CPA-secure KEMs in the first category of the NIST second-round candidates, in which

Fig. 1: The structure of CPA-secure LWE-based KEM

Alice		**Bob**
1. $\triangleright$KEM.Gen()		
1.1 Gen $\mathbf{a} \xleftarrow{\$} \mathcal{R}$		
1.2 $\mathbf{s}_A, \mathbf{e}_A \xleftarrow{\$} \chi$		2. $\mathbf{m} \xleftarrow{\$} \{0,1\}^\lambda$
1.3 $P_A \leftarrow \mathbf{a} \circ \mathbf{s}_A + \mathbf{e}_A$		3. KEM. Enc$(P_A, \mathbf{m})$
1.4 Output: $(P_A, \mathbf{s}_A)$	$\xrightarrow{P_A}$	3.1 Gen $\mathbf{a} \xleftarrow{\$} \mathcal{R}$
		3.2 $\mathbf{s}_B, \mathbf{e}_B, \mathbf{e}'_B \xleftarrow{\$} \chi$
		3.3 $P_B \leftarrow \mathbf{s}_B \circ \mathbf{a} + \mathbf{e}_B$
5. KEM.Dec$(P_B, \bar{\mathbf{c}}, \mathbf{s}_A)$		3.4 $\mathbf{k} \leftarrow$Encode$(\mathbf{m})$
5.1 $\mathbf{c}' \leftarrow$ Decompress$(\bar{\mathbf{c}})$		3.5 $\mathbf{c} \leftarrow \mathbf{s}_B \circ P_A + \mathbf{e}'_B + \mathbf{k}$
5.2 $\mathbf{k}' = \mathbf{c}' - P_B \circ \mathbf{s}_A$	$\xleftarrow{(P_B, \bar{\mathbf{c}})}$	3.6 $\bar{\mathbf{c}} \leftarrow$ Compress$(\mathbf{c})$
5.3 $\mathbf{m} \leftarrow$ Decode$(\mathbf{k}')$		3.7 Output: $(P_B, \bar{\mathbf{c}})$
5.4 Output: $\mathbf{m}'$		4. $K_B \leftarrow \mathbf{H}(\mathbf{m} \| (P_B, \bar{\mathbf{c}}))$
6.$K_A \leftarrow \mathbf{H}(\mathbf{m}' \| (P_B, \bar{\mathbf{c}}))$		

- $\mathcal{R}$ be some ring equipped with the multiplication $\circ$.
- $\mathbf{a}$ is generated by a public seed and pseudorandom function.
- The distribution χ is chosen to be the discrete Gaussian distribution or the central binomial distribution $\mathbf{B}_\eta$ whose sample is generated by $\sum_{i=1}^{\eta}(a_i - b_i)$, where a_i and b_i are independently uniformly randomly sampled from $\{0,1\}$. When we say a sample is chosen according to χ, we mean every component is chosen randomly according to χ.
- The Encode and Decode process is not necessary but usually employed. A typical code is $D4$-v lattice code with $v = 2$ or 4 that encodes every coefficient into v coefficients. We list the Encode and Decode functions in Algorithm 1.
- The Compress/Decompress function is usually used to decrease the communication cost. A typical compress function transforms a coefficient from module q to module p by

$$\text{Compress}_{\text{q}}(\mathbf{c}[\text{i}], \text{p}) = \lceil \mathbf{c}[\text{i}] \cdot \text{p}/\text{q} \rfloor \pmod{\text{p}},$$

Algorithm 1 The Encode and Decode functions for the D-v lattice code

$\diamond$ **Encode**$(\mathbf{m}, v)$
Input: $\mathbf{m} \leftarrow \{0,1\}^{\lambda}, v$
Output: $\mathbf{k}$
1: **for** $i = 0$ *to* $\lambda - 1$ **do**
2: **for** $j = 0$ *to* $v - 1$ **do**
3: $\mathbf{k}[\mathrm{i} \cdot v + \mathrm{j}] = \mathbf{m}[\mathrm{i}] \cdot \frac{q-1}{2}$
4: **end for**
5: **end for**
6: **Return k**

$\diamond$ **Decode**$(\mathbf{k}, v)$
Input: $\mathbf{k} \leftarrow \{0, \frac{q-1}{2}\}^{v\lambda}, v$
Output: $\mathbf{m}'$
7: **for** $i = 0$ *to* $\lambda - 1$ **do**
8: **if** $\sum_{j=0}^{v-1} |\mathbf{k}[i \cdot v + j] - \frac{q-1}{2}| < \frac{v \cdot q}{4}$ **then**
9: $\mathbf{m}'[i] = 1$
10: **else**
11: $\mathbf{m}'[i] = 0$
12: **end if**
13: **end for**
14: **Return** $\mathbf{m}'$

and the decompress function operates in an opposite way:

$$\mathrm{Decompress_q}(\bar{\mathbf{c}}[\mathrm{i}], \mathrm{p}) = \lceil \bar{\mathbf{c}}[\mathrm{i}] \cdot \mathrm{q/p} \rfloor.$$

Next, we describe the MLWE-based Kyber in details.

Kyber. Kyber is on the third-round list of the NIST competition, and regarded as one of the most promising ones for the final standard. In Kyber the authors have warned about the harm of key reuse, but in practice there may still be some users who ignore the warnings and try to create one. So it is reasonable to assume that Kyber has a CPA-secure version to evaluate its key reuse resilience.

Fig. 2 shows pseudo-code for a possible instantiation of the CPA-secure Kyber, which directly invokes the three functions of Kyber.CPAPKE in [7]: Kyber.CPAPKE.KeyGen(), Kyber.CPAPKE.Enc() and Kyber.CPAPKE.Dec().

In Kyber.CPAPKE.KeyGen(), Alice first generates a matrix $\mathbf{a} \in R_q^{k \times k}$. Here R_q represents the ring $\mathbb{Z}_q[x]/(x^N + 1)$, where $N = 256$ and $q = 3329$. Another parameter k is set to be 2, 3 or 4, which is in accordance with the three different security levels. That is, Kyber512, Kyber768, and Kyber1024, respectively. In Kyber all the secret keys and error vectors are sampled from a centered binomial distribution $\mathbf{B}_\eta$. In Kyber512 $\eta = 3$, and in Kyber768 and Kyber1024 $\eta = 2$. Here $\mathbf{B}_\eta$ is generated using $\sum_{i=1}^{\eta}(a_i - b_i)$, where a_i and b_i are independently randomly sampled from $\{0, 1\}$.

NewHope. Similarly, we present a CPA-secure version of NewHope in Fig. 3, which also includes three parts. Here $\mathcal{R}_q$ is the residue ring $\mathbb{Z}_q[x]/(x^N + 1)$ with $N = 512$ in NewHope512 and 1024 in NewHope1024. The parameter q is always set as 12289.

Fig. 2: The CPA version of Kyber

Alice		**Bob**
1. ▷ Kyber. CPAPKE. KeyGen()		
1.1 Generate matrix $\mathbf{a} \in \mathcal{R}_q^{k \times k}$		
1.2 Sample $\mathbf{s}_A, \mathbf{e}_A \in \mathbf{B}_\eta^k$		2. $\mathbf{m} \xleftarrow{\$} \{0,1\}^{256}$
1.3 $\mathbf{P}_A \leftarrow \mathbf{a} \circ \mathbf{s}_A + \mathbf{e}_A$		3. ▷ Kyber. CPAPKE. Enc$(\mathbf{P}_A, \mathbf{m})$
1.4 Output: $(\mathbf{s}_A, \mathbf{P}_A)$	$\xrightarrow{\mathbf{P}_A}$	3.1 Generate matrix $\mathbf{a} \in \mathcal{R}_q^{k \times k}$
		3.2 Sample $\mathbf{s}_B, \mathbf{e}_B \in \mathbf{B}_\eta^k$, $\mathbf{e}'_B \in \mathbf{B}_\eta$
		3.3 $\mathbf{P}_B \leftarrow \mathbf{a}^T \circ \mathbf{s}_B + \mathbf{e}_B$
5. ▷ Kyber. CPAPKE. Dec$(\mathbf{s}_A, \mathbf{P}_B, \mathbf{c}_1, \mathbf{c}_2)$		3.4 $\mathbf{v}_B \leftarrow \mathbf{P}_A^T \circ \mathbf{s}_B + \mathbf{e}'_B + \mathbf{Decompress}_q(\mathbf{m}, 2)$
5.1 $\mathbf{u}_A \leftarrow \mathbf{Decompress}_q(\mathbf{c}_1, 2^{d_{\mathbf{P}_B}})$		3.5 $\mathbf{c}_1 \leftarrow \mathbf{Compress}_q(\mathbf{P}_B, 2^{d_{\mathbf{P}_B}})$
5.2 $\mathbf{v}_A \leftarrow \mathbf{Decompress}_q(\mathbf{c}_2, 2^{d_{\mathbf{v}_B}})$	$\xleftarrow{(\mathbf{P}_B, \mathbf{c}_1, \mathbf{c}_2)}$	3.6 $\mathbf{c}_2 \leftarrow \mathbf{Compress}_q(\mathbf{v}_B, 2^{d_{\mathbf{v}_B}})$
5.3 $\mathbf{m}' \leftarrow \mathbf{Compress}_q(\mathbf{v}_A - \mathbf{s}_A^T \circ \mathbf{u}_A, 2)$		3.7 Output: $(\mathbf{c}_1, \mathbf{c}_2)$
5.4 Output: $\mathbf{m}'$		4. $K_B \leftarrow \mathbf{H}(\mathbf{m} \| \| (\mathbf{P}_B, (\mathbf{c}_1, \mathbf{c}_2)))$
6. $K_A \leftarrow \mathbf{H}(\mathbf{m}' \| \| (\mathbf{P}_B, (\mathbf{c}_1, \mathbf{c}_2)))$		

Fig. 3: The CPA version of NewHope

Alice		**Bob**
1. ▷ NewHope. CPAPKE. KeyGen()		
1.1 Generate matrix $\mathbf{a} \in \mathcal{R}_q$		
1.2 Sample $\mathbf{s}_A, \mathbf{e}_A \in \mathbf{B}_8$		2. $\mathbf{m} \xleftarrow{\$} \{0,1\}^{256}$
1.3 $\mathbf{P}_A \leftarrow \mathbf{a} \circ \mathbf{s}_A + \mathbf{e}_A$		3. ▷ NewHope. CPAPKE. Enc$(\mathbf{P}_A, \mathbf{m})$
1.4 Output: $(\mathbf{s}_A, \mathbf{P}_A)$	$\xrightarrow{\mathbf{P}_A}$	3.1 Generate matrix $\mathbf{a} \in \mathcal{R}_q$
		3.2 Sample $\mathbf{s}_B, \mathbf{e}_B, \mathbf{e}'_B \in \mathbf{B}_8$
		3.3 $\mathbf{P}_B \leftarrow \mathbf{a} \circ \mathbf{s}_B + \mathbf{e}_B$
		3.4 $v_b \leftarrow \mathbf{H}_1(\mathbf{m})$
5. ▷ NewHope. CPAPKE. Dec$(\mathbf{s}_A, \mathbf{P}_B, \bar{\mathbf{c}})$		3.5 $\mathbf{k} \leftarrow$Encode(v_b)
5.1 $\mathbf{c}' \leftarrow$ Decompress$(\bar{\mathbf{c}})$		3.6 $\mathbf{c} \leftarrow \mathbf{P}_A \circ \mathbf{s}_B + \mathbf{e}'_B + \mathbf{k}$
5.2 $\mathbf{k}' \leftarrow \mathbf{c}' - \mathbf{P}_B \circ \mathbf{s}_A$	$\xleftarrow{(\mathbf{P}_B, \bar{\mathbf{c}})}$	3.7 $\bar{\mathbf{c}} \leftarrow$ Compress$(\mathbf{c})$
5.3 $v_A \leftarrow$ Decode$(\mathbf{k}')$		3.8 Output: $(\mathbf{P}_B, \bar{\mathbf{c}})$
5.4 Output: v_A		4. $K_B \leftarrow \mathbf{H}_2(v_b \| \| (\mathbf{P}_B, \bar{\mathbf{c}}))$
6. $K_A \leftarrow \mathbf{H}_2(v_A \| \| (\mathbf{P}_B, \bar{\mathbf{c}}))$		

2.2 Model of key mismatch attacks

In a key mismatch attack, Alice's public key P_A is reused. The adversary $\mathcal{A}$ impersonates as Bob to recover the secret key of Alice with the help of an Oracle that can decide if the two shared keys match or not.

More precisely, to show how the attack works, we build an Oracle $\mathcal{O}$ that simulates Alice's KEM.Dec part. As shown in Algorithm 2, the Oracle $\mathcal{O}$'s input P includes the parameters P_B, $\bar{c}$ chosen by the adversary and the shared key K_B. The output of $\mathcal{O}$ is 1 or 0. To be specific, with the received P_B, $\bar{c}$, $\mathcal{O}$ calls the function $\mathrm{Dec}(P)$ and gets the shared key K_A as the return. If the shared keys K_A and K_B match, $\mathcal{O}$ outputs 1, otherwise the output is 0.

Algorithm 2 The Oracle and key mismatch attack

$\diamond$ **Oracle** $\mathcal{O}(P)$
Input: $P := (P_B, \bar{c}, K_B)$
Output: 0 or 1
1: $K_A \leftarrow \mathrm{KEM.Dec}(P_B, \bar{c})$
2: **if** $K_A = K_B$ **then**
3: **Return** 1
4: **else**
5: **Return** 0
6: **end if**

$\diamond$ **key mismatch attack**
Input: Alice's P_A and Oracle $\mathcal{O}$
Output: 0 or 1
7: $\mathbf{s}'_A \leftarrow \mathcal{A}^{\mathcal{O}}(P_A)$
8: **if** $\mathbf{s}'_A = \mathbf{s}_A$ **then**
9: **Return** 1
10: **else**
11: **Return** 0
12: **end if**

3 Lower bounds for the average number of queries for the key mismatch attacks

For the key mismatch attacks on lattice-based KEMs, the adversary $\mathcal{A}$'s goal is to recover each coefficient of Alice's secret key $\mathbf{s}_A$ by accessing the oracle $\mathcal{O}$ multiple times.

For simplicity, we assume the adversary recovers Alice's secret key $\mathbf{s}_A$ one coefficient block by one coefficient block. A coefficient block can be either one coefficient of $\mathbf{s}_A$ or a subset of all the coefficients of $\mathbf{s}_A$. Usually, for KEMs that do not employ Encode/Decode functions, such as Kyber, a coefficient block is set to be only one coefficient. For KEMs that employ Encode/Decode functions, such as NewHope, a coefficient block contains v coefficients of $\mathbf{s}_A$ where v is defined as in Algorithm 1, since one coefficient relates to v coefficients of $\mathbf{s}_A$.

Note that the number of the queries to the oracle is obviously a key index to evaluate the efficiency of the attack. In fact, even in practice, the bottleneck of the efficiency of the attacks is also to determine if the two shared keys match or not. Therefore, it is important to indicate the optimal lower bound of the number of queries to mount a mismatch attack successfully.

3.1 Lower bound by optimal binary recovery tree

In this subsection, we describe how to find the bounds of key mismatch attacks, which can be regarded as a problem of finding a binary tree with minimum weighted depth.

Recall that the adversary $\mathcal{A}$ recovers Alice's secret key $\mathbf{s}_A$ one coefficient block by one coefficient block, where a coefficient block can be either one coefficient of $\mathbf{s}_A$ or several coefficients of $\mathbf{s}_A$. Let $\mathbf{S} = \{\mathbf{S}_0, \mathbf{S}_1, \cdots, \mathbf{S}_{n-1}\}$ be the set of all the possible values for one coefficient block. For example, the coefficients of $\mathbf{s}_A$ in Kyber are drawn from $\{-2, -1,\ 0,\ 1,\ 2\}$. Since there are no Encode/Decode functions, we try to recover the coefficients of $\mathbf{s}_A$ one by one and hence $\mathbf{S} = \{-2, -1,\ 0,\ 1,\ 2\}$. In LAC, the coefficients of $\mathbf{s}_A$ are selected from $\{-1,\ 0,\ 1\}$. Since D-2 lattice is used to encode, we would like to recover every coefficient block which contains 2 coefficients of $\mathbf{s}_A$ due to the decryption, which yields that $\mathbf{S} = \{(0,0), (0,1), (1,0), (-1,0), (0,-1), (1,1), (-1,-1), (1,-1), (-1,1)\}$.

For any coefficient block $\mathbf{s}_A^b$ of $\mathbf{s}_A$, denote by P_i the probability that $\mathbf{s}_A^b = \mathbf{S}_i$ where $\mathbf{s}_A$ is generated from the distribution χ, that is, $P_i = \text{Prob}(\mathbf{s}_A^b = \mathbf{S}_i | \mathbf{s}_A \leftarrow \chi)$ for $i = 0, 1, \cdots, n-1$. Without loss of generality, we assume that $P_0 \geq P_1 \geq \cdots \geq P_{n-1}$. Then, it holds that $\sum_{i=0}^{n-1} P_i = 1$.

In a key mismatch attack, the adversary $\mathcal{A}$ needs to query the Oracle with properly selected parameters for several times to recover every coefficient block, which may be $\mathbf{S}_i$ with probability P_i. Denote by Q_i the number of queries $\mathcal{A}$ needs to determine the coefficient block when it is exactly $\mathbf{S}_i$. Then the average (expected) number of queries required to recover one coefficient block is obviously:

$$E_{\mathcal{A}}(\mathbf{S}) = \sum_{i=0}^{n-1} P_i Q_i.$$

Our goal is to minimize $E_{\mathcal{A}}(\mathbf{S})$ by running over the set of all possible attack strategies under our model.

Binary recovery tree. Our key idea to get a lower bound of minimum of $E(S)$ is to associate every attack with a binary recovery tree (BRT).

Define the BRT associated with $\mathbf{S} = \{\mathbf{S}_0, \mathbf{S}_1, \cdots, \mathbf{S}_{n-1}\}$ as below: it is a rooted binary tree with a root node and n leaf nodes, where every $\mathbf{S}_i$ occupies a leaf node. For every node that has child nodes, denote by 1 its left child node and by 0 its right child node.

Note that to recover any coefficient block for any attack, the adversary $\mathcal{A}$ can get a binary sequence of returned values from the Oracle. Denote by $\bar{s}_i$ the corresponding returned binary sequence when the coefficient block is exactly $\mathbf{S}_i$. It is obvious that each coefficient block $\mathbf{S}_i$ can be recovered by a unique binary sequence $\bar{s}_i$ and for any $i \neq j$, $\bar{s}_i$ must not be the prefix of $\bar{s}_j$. Otherwise, it would not suffice to identify $\mathbf{S}_i$ uniquely. This means that we can construct a BRT $T_{\mathcal{A}}$ associated with $\mathbf{S} = \{\mathbf{S}_0, \mathbf{S}_1, \cdots, \mathbf{S}_{n-1}\}$, where for every i, the binary string consisting of the nodes on the path from the root node to the leaf node

$\mathbf{S}_i$ is exactly the binary sequence $\bar{s}_i$. The length of $\bar{s}_i$ is of course Q_i as defined above, also known as the depth $\text{depth}_{T_{\mathcal{A}}}(\mathbf{S}_i)$ of leaf node $\mathbf{S}_i$. Then

$$E_{\mathcal{A}}(\mathbf{S}) = \sum_{i=0}^{n-1} P_i Q_i = \sum_{i=0}^{n-1} P_i \cdot \text{depth}_{T_{\mathcal{A}}}(\mathbf{S}_i).$$

It seems still hard to find the minimum of $E_{\mathcal{A}}(\mathbf{S})$ since we should consider all the binary recovery trees corresponding to the possible attacks under our model. However, it presents an obvious way to compute a lower bound of the minimum, just by enlarging the set of BRTs corresponding to the attacks to the set of all the possible BRTs.

Then, we can transform the problem of finding the lower bound of the optimal value of $E_{\mathcal{A}}(\mathbf{S})$ to the problem of finding a binary recovery tree to minimize

$$E(\mathbf{S}) = \sum_{i=0}^{n-1} P_i \cdot \text{depth}_T(\mathbf{S}_i).$$

We call the tree with the minimum weighted depth, i.e. $\min E(\mathbf{S})$, the optimal BRT. Therefore, it is enough to construct an optimal BRT to find the lower bound for recovering the secret key with fewest number of queries.

A well known method to find the optimal binary recovery tree is the Huffman coding [36,38]. The basic idea of Huffman coding is to combine two symbols with the lowest probabilities in each step. Specifically, we first find the two $\mathbf{S}_i$'s with the lowest probabilities, for example, P_{n-1} and P_{n-2}. Then the problem has transformed into solving the problem with $n-1$ weights $\{P_0, P_1, \ldots, P_{n-3}, P_{n-2}+P_{n-1}\}$. By repeating this process, we can finally solve the problem and find the optimal BRT to get $\min E(\mathbf{S})$ in time $O(n \log n)$, as well as the E(#Queries).

Therefore, our proposed method for calculating the bound can be summarized as follows: First, list $\mathbf{S}_0, \mathbf{S}_1, \ldots, \mathbf{S}_{n-1}$ and their corresponding probabilities $\{P_0, P_1, \ldots, P_{n-1}\}$ in the descending order. Then, construct the optimal BRT using Huffman coding. The constructed optimal BRT leads us to the $\min E(\mathbf{S})$ and the E(#Queries). The process of building the Huffman code to obtain the corresponding $\min E(\mathbf{S})$ is shown in Algorithm 3.

To prove our main theorem, we first present the following lemma, which is a special case of the famous Kraft inequality (See Theorem 5.2.2, [20]).

Lemma 1. *(Kraft equality) For any $n \geq 1$, $(\text{depth}_T(\boldsymbol{S}_0), \cdots, \text{depth}_T(\boldsymbol{S}_{n-1}))$ is the sequence of depths in a rooted binary tree if and only if*

$$\sum_{i=0}^{n-1} 2^{-\text{depth}_T(\boldsymbol{S}_i)} = 1. \tag{1}$$

Further, we obtain the following result.

Theorem 1. *In our key mismatch attack model, the proposed method finds bounds for minimum average number of queries in launching the key mismatch*

Algorithm 3 Huffman codes

◇ **Building a Huffman Tree**
Input: $P_0, \cdots, P_{n-1}$
Output: HuffTree T
for $i = 0 \to n-1$ **do**
  Insert leafnode $T[i]$
  $T[i].weight = P[i]$
end for
for $i = 0 \to n-1$ **do**
  for $j = 0 \to n+i-1$ **do**
    Find two nodes x_1 and x_2 with the smallest weight and no parent
  end for
  Combine x_1 and x_2, and insert the new node into $T[n+i]$
end for

◇ **Huffman Coding**
Input: HuffTree T
Output: Huffman code C
E$(S) = 0$
for $i = 0 \to n-1$ **do**
  $C[i].length = 0$
  $j = i$
  while T[j].parent exist **do**
    if $T[j].lchild = j$ **then**
      $C[i].code[C[i].length] = 0$
    else
      $C[i].code[C[i].length] = 1$
    end if
    $C[i].length++$
    $j = T[j].parent$
  end while
  E$(S) +=$ $C[i].length * T[i].weight$
end for

attacks. To be precise, given $\boldsymbol{S} = \{\boldsymbol{S}_0, \boldsymbol{S}_1, \cdots, \boldsymbol{S}_{n-1}\}$ *and its corresponding probabilities* $\{P_0, P_1, \cdots, P_{n-1}\}$ *in each lattice-based KEM,* $\min E(\boldsymbol{S})$ *calculated by the optimal BRT is a lower bound for the minimum average number of queries. Moreover, set* $H(\boldsymbol{S})$ *the Shannon entropy for* $\boldsymbol{S}$*, then we have*

$$H(\boldsymbol{S}) \leq \min\ \mathrm{E}(\mathbf{S}) < H(\boldsymbol{S}) + 1.$$

Proof. Our first result comes from the facts in Section 5.8 of [20]. That is, it is impossible to find any other code with a lower expected length than the code constructed by Huffman coding. To obtain the min E(**S**), we use the Lagrange multipliers. From Lemma 1, we let

$$L = \sum_{i=0}^{n-1} P_i \cdot \mathrm{depth}_T(\mathbf{S}_i) + \lambda(\sum_{i=0}^{n-1} 2^{-\mathrm{depth}_T(\mathbf{S}_i)} - 1).$$

By differentiating with respect to $\mathrm{depth}_T(\mathbf{S}_i)$ and letting the derivative be 0, we have

$$\frac{\partial L}{\partial \mathrm{depth}_T(\mathbf{S}_i)} \ = \ P_i - \lambda \cdot 2^{-\mathrm{depth}_T(\mathbf{S}_i)} \log_e 2 \ = \ 0.$$

That is $2^{-\mathrm{depth}_T(\mathbf{S}_i)} = \frac{P_i}{\lambda \log_e 2}$. Substituting this into Equation (1), we obtain $\sum_{i=0}^{n-1} \frac{P_i}{\lambda \log_e 2} = 1$. Thus we have $\lambda \log_e 2 = 1$, which leads to $P_i = 2^{-\mathrm{depth}_T(\mathbf{S}_i)}$. Therefore, the optimum solution occurs when $\mathrm{depth}_T(\mathbf{S}_i) = \lceil -\log_2 P_i \rceil$. Here

$\lceil x \rceil$ means the smallest integer greater than or equal to x, due to the fact that $\text{depth}_T(\mathbf{S}_i)$ should be integers. Since $x \leq \lceil x \rceil < x + 1$, we then conclude that $H(\mathbf{S}) \leq \min\ \mathrm{E}(\mathbf{S}) < H(\mathbf{S}) + 1$.

In [9], it has been proved that $H(\mathbf{S}) \leq \min\ \mathrm{E}(\mathbf{S})$. From our perspective, this can be easily obtained from the optimality of Huffman codes.

Remark 1. One may have the idea that it is safe to implement the CPA-secure version and reuse the keys fewer times than the proposed bound. In fact it is still dangerous to do so, even reusing the key far below the bound. First of all, our bound is on the average number of needed queries, which means that there may exist attacks with fewer number of queries for certain keys. Secondly, what we talk about is recovering the full key, but obviously the recovery of the partial key also leaks information about the key, significantly decreasing the bit-security. Therefore, it is still not safe to reuse the keys in a CPA-secure KEM.

3.2 Lower bounds for key mismatch attacks on NIST candidates

Lower bounds for Kyber In this subsection, we take Kyber1024 as an example to show how to find the optimal BRT to get the bound. Kyber1024 uses centered binomial distribution $\mathcal{B}_\eta$ with $\eta = 2$ and has no Encode/Decode functions, which means $S = \{-2, -1, 0, 1, 2\}$. We set $\mathbf{S}_0 = 0$, $\mathbf{S}_1 = 1$, $\mathbf{S}_2 = -1$, $\mathbf{S}_3 = 2$ and $\mathbf{S}_4 = -2$.

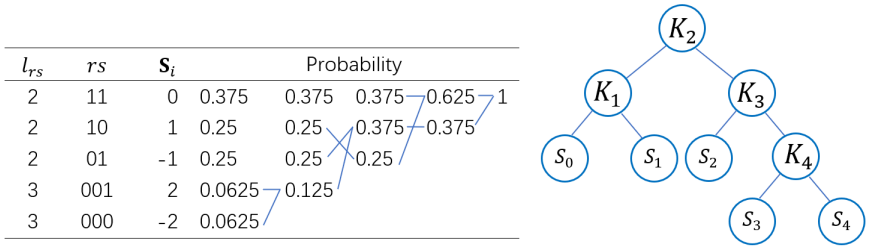

Fig. 4: Finding the optimal BRT for Kyber1024 by using Huffman coding

As shown in Fig. 4, we first list the occurrence probabilities of $\mathbf{S}_i$ in the descending order. Since $\mathbf{S}_3$ and $\mathbf{S}_4$ occur with the smallest probabilities, we create a subtree that contains them as leaf nodes. By repeatedly doing so, finally we can get an optimal BRT as also shown in Fig. 4. The corresponding $\bar{s}$ represents how to encode each $\mathbf{S}_i$, while $l_{\bar{s}}$ is the code length.

The resulting $\min E(\mathbf{S}) = 2.125$, which is the minimum number of queries needed for recovering each coefficient. Note that the Shannon entropy $H(\mathbf{S})$ is

$$H(\mathbf{S}) = \sum_{i=0}^{4} P_i \log \frac{1}{P_i} = 2.03,$$

which is in accordance with our Theorem 1. Hence, the bounds for recovering the full private key of Kyber768 and Kyber1024 with $\eta = 2$ are 1632 and 2176, respectively. Similarly, it can be concluded that the bound is 1216 for Kyber512 with $\eta = 3$.

Lower bounds for NewHope. One of the main differences between Kyber and NewHope is that Kyber does not use Encode/Decode functions, while NewHope uses both Encode/Decode and Compress/Decompress functions. In NewHope, the secret key is sampled from centered binomial distribution $\mathbf{B}_\eta$ with parameter $\eta = 8$, so the coefficients of the secret key are integers in $[-8, 8]$.

Table 1: Lower bounds for key mismatch attacks on lattice-based NIST KEMs.

Schemes	$\mathbf{s}_A$ & $\mathbf{e}$ Ranges	Encode Decode	Comp Decomp	Unknowns	E(#Queries) Bounds
NewHope512	[-8,8]	✓	✓	512	1568
NewHope1024				1024	3127
Kyber512	[-3,3]			512	1216
Kyber768	[-2,2]	/	✓	768	1632
Kyber1024				1024	2176
LightSaber	[-5,5]			512	1412
Saber	[-4,4]	/	✓	768	1986
FireSaber	[-3,3]			1024	2432
Frodo640	[-12,12]			5120	18,227
Frodo976	[-10,10]	/	✓	7808	25,796
Frodo1344	[-6,6]			10,752	27,973
NTRU hps4096821	[-1,1]	/	/	821	1369
NTRU hrss701				701	1183
NTRU Prime sntrup857				857	1574
NTRU Prime ntrulpr857				857	1553

Recall that NewHope512 uses D-2 Encode/Decode functions, while in NewHope-1024 D-4 Encode/Decode functions are used. Therefore, in NewHope512, $\mathbf{S}_i = (s_{i,1}, s_{i,2})$ where $s_{i,1},\ s_{i,2} \in [-8, 8]$. In total there are 289 possibilities about each $\mathbf{S}_i$. So here we let $n = 289$. Then, we can also build the optimal BRT for NewHope512 using Huffman coding, and the $\min E(\mathbf{S}) = 6.124$. Since we can recover two coefficients in $\mathbf{s}_A$ at one time, the resulted E(#Queries)=1568. For NewHope1024, there are a total of $83,521$ possible $\mathbf{S}_i$, that is, $n = 83,521$. Similarly, we have E(#Queries)= 3127 for NewHope1024.

Lower bounds for other NIST Candidates. Similarly, we can obtain bounds for other LWE-based KEMs as well as NTRU and NTRU Prime in the second category. In Table 1, we present the lower bounds for key mismatch attacks

against the following second or third round NIST candidates: NewHope, Kyber, FrodoKEM, Saber, NTRU and NTRU Prime. For every candidate, we report the ranges of $\mathbf{s}_A$ & $\mathbf{e}$ and the number of unknowns, and whether the Encode/Decode and Compress/Decompress functions are employed (✓) or not (/). We also report the minimum average number of queries in our proposed bounds. For other NIST candidate KEMs, we report their results in Table 7 in Appendix A.

4 Improved key mismatch attacks on NIST candidates

We would like to point out that for some KEMs, there is still a huge gap in terms of number of queries between our theoretical bound and practical attacks, such as Frodo640. Since we have built an optimal BRT for each KEM, in the following we show how the optimal BRT helps us improve the practical attacks.

4.1 Improved practical attacks on Kyber

We take Kyber1024 as an example to show how to launch the practical key mismatch attack. First, we build an Oracle that simulates Alice's Kyber.KEM.Dec(), the same as that in Algorithm 2. The inputs of the oracle $\mathcal{O}$ are $\mathbf{P}_B$, $(\mathbf{c}_1, \mathbf{c}_2)$ and K_B.

In a key mismatch attack, Alice's public key $\mathbf{P}_A$ is reused, and the goal of the adversary $\mathcal{A}$ is to recover Alice's secret key $\mathbf{s}_A$. Therefore, $\mathcal{A}$ needs to choose the appropriate parameters $\mathbf{P}_B$ and $(\mathbf{c}_1, \mathbf{c}_2)$ to access $\mathcal{O}$, so that he can determine $\mathbf{s}_A$ based on $\mathcal{O}$'s return. Without loss of generality, assume that $\mathcal{A}$ wants to recover $\mathbf{s}_A[0]$. We next show the basic idea of our attack.

First of all, $\mathcal{A}$ selects a 256-bit $\mathbf{m}$ as (1,0, $\cdots$, 0). Then he sets $\mathbf{P}_B = \mathbf{0}$, except $\mathbf{P}_B[0] = \lceil \frac{q}{32} \rfloor$. After calculating $\mathbf{c}_1 = \mathbf{Compress}_q(\mathbf{P}_B, 2^{d_{\mathbf{P}_B}})$, $\mathcal{A}$ sets $\mathbf{c}_2 = \mathbf{0}$, except that $\mathbf{c}_2[0] = h$, where h will be determined later.

With $(\mathbf{c}_1, \mathbf{c}_2)$, the Oracle calculates $\mathbf{u}_A = \mathbf{Decompress}_q(\mathbf{c}_1, 2^{d_{\mathbf{P}_B}})$, $\mathbf{v}_A = \mathbf{Decompress}_q(\mathbf{c}_2, 2^{d_{\mathbf{v}_B}})$ and

$$\mathbf{m}'[0] = \mathbf{Compress}_q((\mathbf{v}_A - \mathbf{s}_A^T\mathbf{u}_A)[0], 1) = \left\lceil \frac{2}{q} \left(\mathbf{v}_A[0] - (\mathbf{s}_A^T\mathbf{u}_A)[0] \right) \right\rfloor \bmod 2.$$

Since $\mathbf{v}_A[0] = \lceil \frac{q}{32} h \rfloor$ and $(\mathbf{s}_A^T\mathbf{u}_A)[0] = \mathbf{s}_A^T[0]\mathbf{u}_A[0] = \mathbf{s}_A^T[0] \lceil \frac{q}{32} \rfloor$, it holds that $\mathbf{m}'[0] = \left\lceil \frac{2}{q} \left(\lceil \frac{q}{32} h \rfloor - \mathbf{s}_A^T[0] \lceil \frac{q}{32} \rfloor \right) \right\rfloor \bmod 2$.

Therefore, it allows us to determine $\mathbf{s}_A^T[0]$ by choosing proper value for h. For example, by letting $h = 8$, we have the following result: If $\mathbf{s}_A^T[0] \in [-2, -1]$, $\mathbf{m}'[0] = 1$, then the oracle will output 1. Otherwise, if $\mathbf{s}_A^T[0] \in [0, 2]$, $\mathbf{m}'[0] = 0$, then the oracle outputs 0. In this way, we can distinguish which subinterval (or subset) $\mathbf{s}_A^T[0]$ belongs to by only one query. Similarly, by choosing a different value for h, we may determine another subinterval that $\mathbf{s}_A^T[0]$ belongs to. Once the intersection of the determined subintervals has only one element, we can determine the value of $\mathbf{s}_A^T[0]$ exactly. However, our goal is to query the Oracle as few as possible, which asks us to choose h more carefully.

From the optimal BRT in Section 3.2, to approach the bound we need to determine $\mathbf{S}_i$ with high occurrence probability with as few numbers of queries as possible. In fact, this also suggests us the ideal way to choose h, that is, choosing h such that the oracle outputs different values when $\mathbf{s}_A^T[0]$ belongs to different sets of descendants, left or right, for every node in the optimal BRT. Of course, such h may not exist. However, the optimal BRT does reveal some clues.

Following the optimal BRT, we show how to choose h for every State in our improved attack and how the States change according to the output of the Oracle in Kyber512, Kyber768 and Kyber1024, respectively in Table 2. Our key mismatch attack always starts from State 1, and then the choice of h in the next State depends on the current Oracle's output. In each State, when the adversary gets a returned value from the Oracle, he can narrow the range of $\mathbf{s}_A[0]$ until the exact value of $\mathbf{s}_A[0]$ is determined.

Table 2: The choice of h and the States

		State 1	State 2	State 3	State 4
Kyber512	h	2	3	4	1
	$\mathcal{O} \to 0$	State 2	State 3	$\mathbf{s}_A[0] = 2$	$\mathbf{s}_A[0] = -1$
	$\mathcal{O} \to 1$	State 4	$\mathbf{s}_A[0] = 0$	$\mathbf{s}_A[0] = 1$	$\mathbf{s}_A[0] = -2$
Kyber768	h	4	5	6	3
	$\mathcal{O} \to 0$	State 2	State 3	$\mathbf{s}_A[0] = 2$	$\mathbf{s}_A[0] = -1$
	$\mathcal{O} \to 1$	State 4	$\mathbf{s}_A[0] = 0$	$\mathbf{s}_A[0] = 1$	$\mathbf{s}_A[0] = -2$
Kyber1024	h	8	9	10	7
	$\mathcal{O} \to 0$	State 2	State 3	$\mathbf{s}_A[0] = 2$	$\mathbf{s}_A[0] = -1$
	$\mathcal{O} \to 1$	State 4	$\mathbf{s}_A[0] = 0$	$\mathbf{s}_A[0] = 1$	$\mathbf{s}_A[0] = -2$

Table 3: $\mathbf{S}_i$ and its corresponding $\bar{s}$, $l_{\bar{s}}$

i	0	1	2	3	4
$\mathbf{S}_i$	0	1	-1	2	-2
$\bar{s}$	01	001	10	000	11
$l_{\bar{s}}$	2	3	2	3	2

As an example, we show how the adversary $\mathcal{A}$ determines $\mathbf{s}_A[0]$ for Kyber1024 in details.

1. The key mismatch attack starts from State 1, and $\mathcal{A}$ sets $h = 8$ first. Then $\{\mathbf{S}_0, \mathbf{S}_1, \mathbf{S}_2, \mathbf{S}_3, \mathbf{S}_4\}$ can be divided into two parts based on the returned value of the first oracle:
   - If $\mathcal{O} \to 0$: $\mathbf{s}_A[0]$ belongs to $\{\mathbf{S}_0, \mathbf{S}_1, \mathbf{S}_3\}$, and goes to State 2.
   - If $\mathcal{O} \to 1$: $\mathbf{s}_A[0]$ belongs to $\{\mathbf{S}_2, \mathbf{S}_4\}$, and State 4 will be executed.
2. If $\mathcal{A}$ comes to State 2, he goes on setting $h = 9$:
   - If $\mathcal{O} \to 0$: $\mathbf{s}_A[0]$ belongs to $\{\mathbf{S}_1, \mathbf{S}_3\}$, then goes to State 3.
   - If $\mathcal{O} \to 1$: $\mathcal{A}$ can determine $\mathbf{s}_A[0] = \mathbf{S}_0 = 0$.
3. In State 3, $\mathcal{A}$ sets $h = 10$:
   - If $\mathcal{O} \to 0$: $\mathcal{A}$ determines $\mathbf{s}_A[0] = \mathbf{S}_3 = 2$.
   - If $\mathcal{O} \to 1$: $\mathcal{A}$ determines $\mathbf{s}_A[0] = \mathbf{S}_1 = 1$.
4. When $\mathcal{A}$ is in State 4, he sets $h = 7$:
   - If $\mathcal{O} \to 0$: $\mathcal{A}$ finds that $\mathbf{s}_A[0] = \mathbf{S}_2 = -1$.
   - If $\mathcal{O} \to 1$: $\mathcal{A}$ finds that $\mathbf{s}_A[0] = \mathbf{S}_4 = -2$.

Based on the above process, we can construct $\bar{s}$, $l_{\bar{s}}$ for $\{\mathbf{S}_0, \mathbf{S}_1, \mathbf{S}_2, \mathbf{S}_3, \mathbf{S}_4\}$, as shown in Table 3. For example, if $\mathbf{s}_A[0] = \mathbf{S}_1 = 1$, we come to State 1 first, and the oracle outputs 0. Then we go to State 2 and the oracle outputs 0. Now we are in State 3 and the output is 1. Therefore we can get $\bar{s} = 001$. We can see that in this way we decide $\mathbf{S}_i$ with larger occurrence probability by as fewer

queries as possible. We can also observe that the way we find $\bar{s}$ is similar to our optimal BRT.

Similarly, to recover $\mathbf{s}_A[i]$ when $i \neq 0$, $\mathcal{A}$ only needs to set $\mathbf{P}_B = 0$ except $\mathbf{P}_B[n-i] = -\lceil \frac{q}{32} \rfloor$ at first.

Now we can calculate the average number of queries needed to recover each coefficient in $\mathbf{s}_A$ as $\frac{3}{8} \times 2 + \frac{1}{4} \times (2+3) + \frac{1}{16} \times (2+3) = 2.31$. Therefore, the corresponding numbers of average queries needed in Kyber1024 and Kyber768 are 2365.44, 1774.08 respectively. Similarly, we can get the average number of queries on Kyber512, which is 1312.06. Compared with the bound in Table 1, there is only a gap less than 9%.

In [49], the authors proposed three different methods to perform key mismatch attacks on Kyber. For their best method, the queries are 2475, 1855 and 1401. Therefore, our improved practical key mismatch attack on Kyber is better than that in [49].

4.2 Improved key mismatch attacks on Saber

There are three versions of Saber, the LightSaber, Saber, and FireSaber. Here we take the attack on FireSaber as an example. The attacks on LightSaber and Saber are similar. The adversary chooses $P_B = h$ and $c_m = k$, and the selection of each h_i/k_i ($i = 1, \ldots, 10$ in LightSaber; $i = 1, \ldots, 8$ in Saber; $i = 1, \ldots, 6$ in FireSaber) is shown in Table 4.

Table 4: Selection of h_i/k_i in the practical key mismatch attacks on Saber

i	1	2	3	4	5	6	7	8	9	10
LightSaber	2/60	1/69	1/35	1/23	0/50	0/40	2/30	2/20	2/15	2/12
Saber	4/28	3/37	3/36	3/18	3/12	4/27	4/13	4/9		
FireSaber	17/7	16/2	16/4	8/125	4/95	2/76				

The following procedure shows how to use h_i/k_i in Table 4 to recover $\mathbf{s}_A[0]$.

1. We set $h = h_1$ and $k = k_1$ first, then $\mathbf{S}_i$ ($i = 0, \ldots, 6$) can be divided into two parts based on the returned value of the first Oracle:
   - If $\mathcal{O} \to 0$: $\mathbf{s}_A[0]$ belongs to $\{\mathbf{S}_1, \mathbf{S}_3, \mathbf{S}_5\}$, and turn to step 4.
   - If $\mathcal{O} \to 1$: $\mathbf{s}_A[0]$ belongs to $\{\mathbf{S}_0, \mathbf{S}_2, \mathbf{S}_4, \mathbf{S}_6\}$, then step 2 and step 3 will be executed.
2. If the oracle returns 1 when we set $h = h_1$ and $k = k_1$, then we set $h = h_2$ and $k = k_2$:
   - If $\mathcal{O} \to 0$: We can determine $\mathbf{s}_A[0] = \mathbf{S}_0$.
   - If $\mathcal{O} \to 1$: $\mathbf{s}_A[0]$ belongs to $\{\mathbf{S}_2, \mathbf{S}_4, \mathbf{S}_6\}$, and go to step 3.
3. Next, we select different parameters $h = h_3$, $k = k_3$ and $h = h_4$, $k = k_4$ (the specific values of h_i/k_i are shown in Table 4) and repeat operations in step 2 until we can know which of $\{\mathbf{S}_2, \mathbf{S}_4, \mathbf{S}_6\}$ is equal to $\mathbf{s}_A[0]$.

4. Similarly, we select different parameters $h = h_5$, $k = k_5$ and $h = h_6$, $k = k_6$ in Table 4 and repeat operations in steps 2 and 3 until we can know which of $\{\mathbf{S}_1, \mathbf{S}_3, \mathbf{S}_5\}$ is $\mathbf{s}_A[0]$.

4.3 Improved key mismatch attacks on FrodoKEM

There are three versions of FrodoKEM, the Frodo640, Frodo976, and Frodo1344. Here we take the attack on Frodo1344 as an example. The attacks on Frodo640 and Frodo976 are similar. In Frodo1344, $\mathbf{S}_i \in [-6, 6]$, the selection of h_i ($i \in [0, 12]$) is shown in Table 5.

Table 5: Selection of h_i in practical key mismatch attacks on FrodoKEM

i	1	2	3	4	5	6
h_i	2^{12}	$2^{12}-2$	$2^{12}-1$	$2^{12}-3$	$2^{12}-4$	$2^{12}-5$
i	7	8	9	10	11	12
h_i	$2^{12}-6$	$2^{12}-7$	$2^{12}-8$	$2^{12}-9$	$2^{12}-10$	$2^{12}-11$

Next, we introduce how to use h_i in Table 5 to recover $\mathbf{s}_A[0]$.

1. We set $h = h_1$ first, then $\mathbf{S}_i (i \in [0, 12])$ can be divided into two parts based on the returns value of the first Oracle:
   - If $\mathcal{O} \to 0$: $\mathbf{s}_A[0]$ belongs to $\{\mathbf{S}_0, \mathbf{S}_2, \mathbf{S}_4, \mathbf{S}_6, \mathbf{S}_8, \mathbf{S}_{10}, \mathbf{S}_{12}\}$, and then step 2 and step 3 will be executed.
   - If $\mathcal{O} \to 1$: $\{\mathbf{S}_1, \mathbf{S}_3, \mathbf{S}_5, \mathbf{S}_7, \mathbf{S}_9, \ \mathbf{S}_{11}\}$
2. If the oracle returns 0 when we set $h = h_1$, then we set $h = h_2$:
   - If $\mathcal{O} \to 0$: We can determine $\mathbf{s}_A[0] = \mathbf{S}_0$.
   - If $\mathcal{O} \to 1$: $\mathbf{s}_A[0]$ belongs to $\{\mathbf{S}_2, \mathbf{S}_4, \mathbf{S}_6, \mathbf{S}_8, \mathbf{S}_{10}, \mathbf{S}_{12}\}$, then we will proceed step 3.
3. Next, we select different parameter $h = h_2, \cdots, h_7$ (the specific values of h_i are shown in Table 5. Repeat operations in step 2 until we can know which of $\{\mathbf{S}_2, \mathbf{S}_4, \mathbf{S}_6, \mathbf{S}_8, \mathbf{S}_{10}, \mathbf{S}_{12}\}$ is $\mathbf{s}_A[0]$.
4. Similarly, we select different parameter $h = h_8, \cdots, h_{12}$ in Table 5 and repeat operations in steps 2 and 3 until we can know which of $\{\mathbf{S}_1, \mathbf{S}_3, \mathbf{S}_5, \mathbf{S}_7, \mathbf{S}_9, \mathbf{S}_{11}\}$ is $\mathbf{s}_A[0]$.

4.4 Improved practical attacks on other NIST candidates

Similarly, we can also improve the key mismatch attacks on NewHope, LAC, and Round5. The details are given in Appendix B, where we show how the adversary chooses the parameters in each scheme, and how to determine $\mathbf{s}_A$ according to the returns of the oracle.

An interesting question is, can we construct an attack to force the Oracle to output the string that is exactly the same as suggested by the optimal BRT? In

this way, certainly we can find an optimal practical attack that reaches the theoretical lower bound. Unfortunately, due to the restriction of concrete schemes, we may not find such parameters to launch the attack since they may not exist at all. For example, if we want to achieve the lower bounds against Kyber1024 using Huffman coding, we need to select the parameter $\mathbf{K}_2$ according to Fig. 4, in this way the range of the secret key is divided into two sub-intervals: $\{0, 1\}$ and $\{-1, 2, -2\}$. However, in our improved practical attacks, the parameter $\mathbf{K}_2$ we choose can only divide the range of the secret key into two adjacent sub-intervals, namely $\{-2, -1\}$ and $\{0, 1, 2\}$, or $\{-2, -1, 0\}$ and $\{1, 2\}$. This is the reason why the number of queries needed in our improved practical attacks is close to the bound, but not exactly the same.

5 Improved side-channel assisted chosen ciphertexts attacks on CCA-secure NIST KEM candidates

As described above, the CPA-secure KEM candidates are vulnerable to key reuse attacks. However, it is well known that the NIST candidates are CCA-secure by applying some well-known transformation such as FO transformation [29]. To be specific, FO transform mainly consists of two parts. First, Alice decrypts Bob's ciphertext $\bar{c}$ to obtain m' and a seed by calling KEM.CPA.Dec. Then she re-encrypts m' and the seed to get c'. If $\bar{c} = c'$, she continues to calculate the shared key, otherwise she rejects the ciphertext $\bar{c}$. This mechanism of decrypting and then re-encrypting in the CCA-secure KEM protects the validity of the ciphertext, returning failure when an invalid ciphertext is detected. Thus Alice always rejects these malicious chosen ciphertexts and the adversary cannot gain any meaningful information, which also means that our attacks above will not work when these cryptosystems are correctly deployed. However, at CHES 2020, Ravi et al. [50] showed that chosen ciphertexts attacks on CCA-secure NIST candidate KEMs can also be launched with the help of side channel information. Therefore, our proposed method can be directly used to further improve the efficiency of these attacks.

Ravi et al.'s key observation is that, we can use the side channel information to bypass the restrictions of FO transform to obtain useful match or mismatch information about decryption outputs of chosen ciphertexts, making it possible to successfully attack CCA-secure cryptosystems. In other words, Ravi et al.'s chosen ciphertext attack is almost the same as the key mismatch attack, except that the adversary can actively know whether the shared message matches or not by physically accessing to devices performing decapsulation.

In Ravi et al.'s side-channel attack (SCA), they mainly utilize Welch's t-test based template approach [30], which consists of two stages. The first is the preprocessing stage including how to generate a template for each class, while the second stage involves the template matching operation. In the first stage, we need to collect 50 measurements of $\mathcal{T} = \mathcal{T}_0 \cup \mathcal{T}_1$. Here, $\mathcal{T}_0$ and $\mathcal{T}_1$ correspond to the failure and success of KEM.CCA.Dec(), respectively. To get $\mathcal{T}_0$, we directly set $\mathbf{m}' = \mathbf{0}$ instead of calling the decryption part KEM.CPA.Dec() in

Algorithm 4 The Oracle and SCA-assisted chosen ciphertext attack

◇ **Oracle** $\mathcal{O}_s(P, m_0, m_1)$
Input: $P := (\mathbf{c}_1, \mathbf{c}_2)$, m_0, m_1
Output: 0 or 1
1: $\mathcal{W} \leftarrow$ SCA(KEM.CCA.Dec(P))
2: **if** $\Gamma_0 \geq \Gamma_1$ **then Return** 1
3: **else** **Return** 0
4: **end if**

◇ **Chosen ciphertext attack**
Input: Alice's P_A and Oracle $\mathcal{O}_s$
Output: 0 or 1
5: $\mathbf{s}'_A \leftarrow \mathcal{A}^{\mathcal{O}_s}(P_A)$
6: **if** $\mathbf{s}'_A = \mathbf{s}_A$ **then Return** 1
7: **else** **Return** 0
8: **end if**

KEM.CCA.Dec(), and then collect the corresponding 50 measurements. Similarly, we can get $\mathcal{T}_1$ by setting $\mathbf{m}' = \{1, 0, \ldots, 0\}$. Then we calculate their respective means denoted as $m_0 = (\sum_{i=1}^{50} \mathcal{T}_0[i])/50$ and $m_1 = (\sum_{i=1}^{50} \mathcal{T}_1[i])/50$. In the second stage, according to the results of m_0 and m_1, when we collect a wave $\mathcal{W}$ from KEM.CCA.Dec(), we can distinguish which class the wave $\mathcal{W}$ belongs to. Specifically, we need to compute the sum-of-squared difference Γ_* of the wave $\mathcal{W}$ with m_* as follows:

$$\Gamma_0 = (\mathcal{W} - m_0)^T \cdot (\mathcal{W} - m_0), \Gamma_1 = \quad (\mathcal{W} - m_1)^T \cdot (\mathcal{W} - m_1).$$

If $\Gamma_0 \geq \Gamma_1$, $\mathcal{W}$ belongs to Γ_1, otherwise it belongs to Γ_0. When $\mathcal{W}$ belongs to Γ_0, we know that $\mathbf{m}' = \mathbf{0}$, which is the same as the principle of the mismatch situation in the key mismatch attack aforementioned. Similarly, if $\mathcal{W}$ belongs to Γ_1, it is consistent with the match situation in key mismatch attack.

Based on the above analysis, we can build an Oracle $\mathcal{O}_s$ that simulates Alice's KEM.CCA.Dec part, which is depicted in Algorithm 4. In the following we take Kyber1024 as an example to show our detailed attacks. But we need to emphasize that our method is applicable to other CCA-secure NIST candidates. For Kyber1024 we choose the parameters $(\mathbf{c}_1, \mathbf{c}_2)$ exactly the same as listed in Table 2, to launch our chosen ciphertext attack. Here we show how the adversary $\mathcal{A}$ determines $\mathbf{s}_A[0] = 0$ with only 2 queries, and the rest are similar.

In the pre-processing stage, $\mathcal{A}$ collects two sets of 50 measurements in advance and computes their respective means. Then $\mathcal{A}$ gets two means m_0 and m_1. For the chosen ciphertext attack stage, starting from State 1 in Table 2, $\mathcal{A}$ sets $\mathbf{P}_B = 0$ except $\mathbf{P}_B[0] = \lceil \frac{q}{32} \rfloor$. After computing $\mathbf{c}_1 = \mathbf{Compress}_q(\mathbf{P}_B, 2^{d_{\mathbf{P}_B}})$, $\mathcal{A}$ sets $\mathbf{c}_2 = \mathbf{0}$, except that $\mathbf{c}_2[0] = 2$ at the first time. If the first output of $\mathcal{O}_s$ is 0, then State 1 switches to State 2. Next, $\mathcal{A}$ sets $\mathbf{c}_2[0] = 3$, if the second output of $\mathcal{O}_s$ is 1 and $\mathcal{A}$ can determine $\mathbf{s}_A[0] = 0$.

In summary, our improved attack can be applied to attack CCA-secure NIST KEM candidates just as Ravi et al.'s chosen ciphertexts attack.

However, Ravi et al. had to brute-forcedly select the parameters, which is not efficient for secret key with larger coefficients. Therefore, our proposed optimal BRT approach can be directly used to select better parameters and significantly reduce the needed number of queries with high efficiency. Specifically, Ravi et

al. only gave the detailed description of attacks against Kyber512 in the second round, where the secret key $\mathbf{s}_A$ is sampled from centered binomial distribution $\mathbf{B}_\eta$ with $\eta = 2$. Thus, in their attack the needed queries for each coefficient is 5. Since $n = 256$, $k = 2$, the total number of queries for Kyber512 is 2560. In order to make a fair comparison with their results, we also apply our improved attack to the second round of Kyber. By adopting our proposed optimal BRT approach, we only need 1182.72 queries on average, reducing the number of queries by 53.79% correspondingly. Secondly, in Ravi et al.'s attack, to retrieve coefficient -2 the selected parameters $(\mathbf{c}_1, \mathbf{c}_2) = (415, 3)$. Through our analysis 415 is too large, which is the reason why their attack cannot succeed with a 100% probability.

In our paper we consider Kyber in the third round, where the private key of Kyber512 ranges from -3 to 3. All the results can be found at Table 6.

Similarly, we can also improve Ravi et al.'s method with our improved key mismatch attacks on other NIST candidates. In [50], for NewHope512, and NewHope1024, the total number of queries is 6945 and 26624, respectively. According to our results in Table 6, we reduce the number of queries for NewHope512 and NewHope1024 by 76.1% and 88.06%, respectively.

In [54], there is another interesting side-channel attack, namely the fault-injection attack, against the NIST KEMs. We find that their main idea is to construct a plaintext-checking oracle by injecting a fault first and then recover the private key by employing the key mismatch attack directly. Hence, we believe our results can also be applied to improve their attacks.

6 Experiments

In this section, we conduct experiments on the above improved attacks to confirm their correctness and efficiency. All our improved key mismatch attacks are implemented on a desktop equipped with two 3 GHz Intel Xeon E5-2620 CPUs and a 64 GB RAM. Our code is based on the C reference implementations of the NIST candidates, and we have made it public[8]. Note that first our attack is against the CPA-secure KEMs for Kyber, we directly call the Kyber.CPAPKE.KeyGen() to launch the attack. For schemes like Saber and FrodoKEM, we remove the FO transform in their CCA version. Since the improved key mismatch attacks and the SCA-assisted selection ciphertext attack share similar processes on the NIST candidate KEMs, as shown in Algorithm 5, we use Kyber1024 as an example to illustrate the details of these attacks.

In the experiment we generate 1000 different secret keys $\mathbf{s}_A$ and recover them separately. We use *queries* to represent the number of times the adversary needs to access the oracle to recover a complete $\mathbf{s}_A$. The experimental results given in Table 6 are the average number of times the adversary needs to access the oracle to recover these 1000 different $\mathbf{s}_A$.

In Table 6, we present our experimental results. For each scheme, we list the lower bound of the minimum average number of queries by our BRT method,

[8] https://github.com/AHaQY/Key-Mismatch-Attack-on-NIST-KEMs

Algorithm 5 Pseudocode of improved key mismatch attack on Kyber1024

◇ **Generate reused key pair**
$(\mathbf{s}_A, \mathbf{P}_A) \leftarrow$ Kyber.CPA.Gen()
◇ **Recover** $\mathbf{s}_A$
Set $\mathbf{m} = \mathbf{0}$, except $\mathbf{m}[0] = 1$
Set $queries = 0$
for $i = 0 \rightarrow 3$ **do**
  for $j = 0 \rightarrow 255$ **do**
    Set $\mathbf{P}_B = \mathbf{0}$
    if $j = 0$ **then**
      $\mathbf{P}_B[0] = \lceil \frac{q}{32} \rfloor$
    else
      $\mathbf{P}_B[256 - j] = -\lceil \frac{q}{32} \rfloor$
    end if
    $\mathbf{c}_1 = \mathbf{Compress}_q(\mathbf{P}_B, d_{\mathbf{P}_B})$
    Set $round = 0$
    while $round < 4$ **do**
      $round$ ++
      Set $\mathbf{c}_2 = \mathbf{0}$, except $\mathbf{c}_2[0] = h$
      $t = \mathcal{O}(\mathbf{c}_1, \mathbf{c}_2)$
      $queries$ ++
      if $\mathcal{A}$ recovers $\mathbf{s}_A[i][j]$ **then**
        Break
      else Continue
      end if
    end while
    if $round == 4$ **then**
      Cannot recover $\mathbf{s}_A[i][j]$
    end if
  end for
end for

Table 6: Key mismatch attacks against lattice-based NIST KEMs.

Schemes	E(#Queries)			
	Lower Bounds	**Our improved attacks**		*Existing*
		Theory	Experiments	
Kyber512	1216	**1312**	1311	*1401* (Round 2)[49]
Kyber768	1632	**1774**	1777	*1855*[49]
Kyber1024	2176	**2365**	2368	*2475*[49]
LightSaber	1412	**1460**	1476	*2048*[37]
Saber	1986	**2091**	2095	-
FireSaber	2432	**2642**	2622	-
Frodo640	18,227	**18,329**	18,360	*65,536*[37]
Frodo976	25,796	**26,000**	26,078	-
Frodo1344	27,973	**29,353**	29,378	-
NewHope512	1568	**1660**	1660	-
NewHope1024	3127	**3180**	3180	*3197*[52]

the expected number of queries for our improved attacks (Bold), the average number of queries for our improved attacks in our experiments, as well as the number of queries of other existing results (Italic). We use "-" to mean that no result is given.

It can be seen that our improved attacks approach the lower bound in most cases and our experiments almost perfectly match the theoretical results in our improved attack. That is, the difference between the improved attack and our experiments is less than 1.2%. As we can see in Table 6 , the experimental results of our improved approach are very close to the theoretical bounds. In general, there is less than 8.2% gap between our experiments and the theoretical bounds.

Compared with other existing attack, we can see that our improved attack on Kyber is slightly better than that in [49], since for Kyber the gap between the lower bounds and practice is small. For Frodo640 and LightSaber, we have reduced the number of queries by 71.99 % and 27.93%, respectively, compared to the results in [37]. Our result on NewHope1024 is slightly better than that of [52]. For LAC256, we greatly decrease the number of queries in comparison with the work of Wang et al. [53]. Using our improved method, the results of LAC128 and LAC192 are also better than the current results [31,53]. The details are shown in Appendix B.

7 Conclusion and discussions

In this paper, we have developed a unified method to calculate the minimum number of required queries in launching key mismatch attacks against lattice-based NIST candidate KEMs. The bound is calculated through constructing an optimal BRT, which is further used to guide us in improving the practical attacks. By using BRT method, our improved attack can significantly reduce the needed number of queries. An interesting problem is whether our proposed method applies to the similar attacks against other post-quantum cryptosystems such as HQC, which also advance to the third round of NIST's PQC standardization progress.

From the analysis of our proposed attacks, we find that the ranges of the coefficients in the secret key and their corresponding probabilities, as well as the employment of Encode/Decode functions are the most important factors in evaluating their key mismatch resilience. More specifically, the larger the range of the coefficients, the more queries are needed. For example, neither Kyber nor Saber use the Encode/Decode functions, and their number of unknowns are the same, the only difference is the range of their coefficients in secret keys. The range of coefficients in Saber is larger than that of Kyber, which leads to more queries in recovering Saber's secret key.

The occurrence probabilities corresponding to the coefficients are another factor. For example, for LAC192 and LAC256, the only difference between them is the occurrence probabilities corresponding to the coefficients. More specifically, in LAC192 the occurrence probability of 0 is greater than that of 0 in LAC256, and the probability of other coefficients is less than that in LAC256. This results

in larger number of queries needed to recover the secret keys of LAC256 than that in LAC192. Whether or not the Encode/Decode functions are used also affects the number of queries needed. NewHope512 and NewHope1024 use D-2 and D-4 functions, respectively, which allows them to recover two and four coefficients at the same time. This also greatly reduces the number of queries needed to recover the coefficients. However, we need to emphasize that these factors only increase complexities of launching the key mismatch attack, but cannot stop the attack.

Acknowledgments

Chi Cheng is the corresponding author. The authors would like to thank Michael Naehrig, Muyan Shen, and the anonymous reviewers for their kind help. The research in this paper was partially supported by the National Natural Science Foundation of China (NSFC) under Grant no.s 62172374, 61672029, and 61732021, and Guangxi Key Laboratory of Trusted Software (no. KX202038). Y. Pan was supported by National Key Research and Development Program of China (No. 2018YFA0704705) and NSFC (No. 62032009). J. D. would like to thank CCB Fintech Co. Ltd for partially sponsoring the work with grant No. KT2000040.

References

1. Ajtai, M.: Generating hard instances of lattice problems. In: Proceedings of the twenty-eighth annual ACM symposium on Theory of computing. pp. 99–108. ACM (1996)
2. Alagic, G., Alagic, G., Alperin-Sheriff, J., Apon, D., Cooper, D., Dang, Q., Liu, Y.K., Miller, C., Moody, D., Peralta, R., et al.: Status Report on the First Round of the NIST Post-Quantum Cryptography Standardization Process. US Department of Commerce, National Institute of Standards and Technology (2019), https://nvlpubs.nist.gov/nistpubs/ir/2019/NIST.IR.8240.pdf.
3. Alkim, E., Avanzi, R., Bos, J., Ducas, L., de la Piedra, A., Pöppelmann, T., Schwabe, P., Stebila, D.: Newhope: Algorithm specification and supporting documentation - version 1.03 (2019), https://newhopecrypto.org/data/NewHope_2019_07_10.pdf.
4. Alkim, E., Bos, J., Ducas, L., Longa, P., Mironov, I., Naehrig, M., Nikolaenko, V., Peikert, C., Raghunathan, A., Stebila, D.: Frodokem learning with errors key encapsulation: Algorithm specification and supporting documentation. In: Submission to the NIST post-quantum project (2019) (2019), https://frodokem.org/files/FrodoKEM-specification-20190702.pdf.
5. Alkim, E., Ducas, L., Pöppelmann, T., Schwabe, P.: Newhope without reconciliation. IACR Cryptology ePrint Archive (2016), https://www.cryptojedi.org/papers/newhopesimple-20161217.pdf.
6. Alkim, E., Ducas, L., Pöppelmann, T., Schwabe, P.: Post-quantum key exchange–a new hope. In: 25th USENIX Security Symposium (USENIX Security 16). pp. 327–343 (2016)

7. Avanzi, R., Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J.M., Schwabe, P., Seiler, G., Stehlé, D.: 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.
8. Baan, H., Bhattacharya, S., Fluhrer, S., Garcia-Morchon, O., Laarhoven, T., Player, R., Rietman, R., Saarinen, M.J.O., Tolhuizen, L., Torre-Arce, J.L., et al.: Round5: merge of round2 and hila5 algorithm specification and supporting documentation. In: Submission to the NIST post-quantum project (2019) (2019), https://round5.org/Supporting_Documentation/Round5_Submission.pdf.
9. Băetu, C., Durak, F.B., Huguenin-Dumittan, L., Talayhan, A., Vaudenay, S.: Misuse attacks on post-quantum cryptosystems. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 747–776. Springer (2019)
10. Bauer, A., Gilbert, H., Renault, G., Rossi, M.: Assessment of the key-reuse resilience of newhope. In: Cryptographers' Track at the RSA Conference. pp. 272–292. Springer (2019)
11. Bernstein, D.J., Bruinderink, L.G., Lange, T., Panny, L.: Hila5 pindakaas: on the cca security of lattice-based encryption with error correction. In: International Conference on Cryptology in Africa. pp. 203–216. Springer (2018)
12. Bernstein, D.J., Chuengsatiansup, C., Lange, T., van Vredendaal, C.: Ntru prime: reducing attack surface at low cost. In: International Conference on Selected Areas in Cryptography. pp. 235–260. Springer (2017)
13. Bernstein, D.J., Chuengsatiansup, C., Lange, T., van Vredendaal, C.: Ntru prime: round 2. In: Submission to the NIST post-quantum project (2019) (2019), https://ntruprime.cr.yp.to/nist/ntruprime-20190330.pdf.
14. Bindel, N., Stebila, D., Veitch, S.: Improved attacks against key reuse in learning with errors key exchange. IACR Cryptology EPrint Archive (2020), https://eprint.iacr.org/2020/1288.pdf.
15. Bleichenbacher, D.: Chosen ciphertext attacks against protocols based on the rsa encryption standard pkcs# 1. In: Annual International Cryptology Conference. pp. 1–12. Springer (1998)
16. Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J.M., Schwabe, P., Seiler, G., Stehlé, D.: Crystals-kyber: a cca-secure module-lattice-based kem. In: 2018 IEEE European Symposium on Security and Privacy (EuroS&P). pp. 353–367. IEEE (2018), https://eprint.iacr.org/2017/634.
17. Bos, J.W., Costello, C., Naehrig, M., Stebila, D.: Post-quantum key exchange for the tls protocol from the ring learning with errors problem. In: 2015 IEEE Symposium on Security and Privacy. pp. 553–570. IEEE (2015)
18. Chen, C., Danba, O., Hoffstein, J., Hülsing, A., Rijneveld, J., Schanck, J.M., Schwabe, P., Whyte, W., Zhang, Z.: Ntru algorithm specifications and supporting documentation. Submission to the NIST post-quantum project (2019) (2019)
19. Chen, L., Jordan, S., Liu, Y.K., Moody, D., Peralta, R., Perlner, R., Smith-Tone, D.: Report on post-quantum cryptography. US Department of Commerce, National Institute of Standards and Technology (2016)
20. Cover, T.M.: Elements of information theory. John Wiley & Sons (1999)
21. Cramer, R., Shoup, V.: Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. SIAM Journal on Computing **33**(1), 167–226 (2003)
22. D'Anvers, J.P., Karmakar, A., Roy, S.S., Vercauteren, F.: 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.
23. D'Anvers, J.P., Tiepelt, M., Vercauteren, F., Verbauwhede, I.: Timing attacks on error correcting codes in post-quantum schemes. In: Proceedings of ACM Workshop on Theory of Implementation Security Workshop. pp. 2–9 (2019)
24. Diffie, W., Hellman, M.: New directions in cryptography. IEEE transactions on Information Theory **22**(6), 644–654 (1976)
25. Ding, J., Alsayigh, S., Saraswathy, R., Fluhrer, S., Lin, X.: Leakage of signal function with reused keys in rlwe key exchange. In: 2017 IEEE International Conference on Communications (ICC). pp. 1–6. IEEE (2017)
26. Ding, J., Branco, P., Schmitt, K.: Key exchange and authenticated key exchange with reusable keys based on rlwe assumption. IACR Cryptology EPrint Archive (2020), https://eprint.iacr.org/2019/665.pdf.
27. Ding, J., Fluhrer, S., Rv, S.: Complete attack on rlwe key exchange with reused keys, without signal leakage. In: Australasian Conference on Information Security and Privacy. pp. 467–486. Springer (2018)
28. Ding, J., Xie, X., Lin, X.: A simple provably secure key exchange scheme based on the learning with errors problem. IACR Cryptology EPrint Archive (2012), https://eprint.iacr.org/2012/688.pdf.
29. Fujisaki, E., Okamoto, T.: Secure integration of asymmetric and symmetric encryption schemes. In: Annual International Cryptology Conference. pp. 537–554. Springer (1999)
30. Gilbert Goodwill, B.J., Jaffe, J., Rohatgi, P., et al.: A testing methodology for side-channel resistance validation. In: NIST non-invasive attack testing workshop. vol. 7, pp. 115–136 (2011)
31. Greuet, A., Montoya, S., Renault, G.: Attack on lac key exchange in misuse situation. IACR Cryptology EPrint Archive (2020), https://eprint.iacr.org/2020/063.
32. Gyongyosi, L., Imre, S.: A survey on quantum computing technology. Computer Science Review **31**, 51–71 (2019)
33. Hall, C., Goldberg, I., Schneier, B.: Reaction attacks against several public-key cryptosystem. In: Information and Communication Security. pp. 2–12. Springer (1999)
34. Hamburg, M.: Post-quantum cryptography proposal: Threebears. In: Submission to the NIST post-quantum project (2019) (2019), https://www.shiftleft.org/papers/threebears/threebears-spec.pdf.
35. Hoffstein, J., Pipher, J., Silverman, J.H.: Ntru: A ring-based public key cryptosystem. In: International Algorithmic Number Theory Symposium. pp. 267–288. Springer (1998)
36. Huffman, D.A.: A method for the construction of minimum-redundancy codes. Proceedings of the IRE **40**(9), 1098–1101 (1952)
37. Huguenin-Dumittan, L., Vaudenay, S.: Classical misuse attacks on nist round 2 pqc. In: International Conference on Applied Cryptography and Network Security. pp. 208–227. Springer (2020), https://link.springer.com/chapter/10.1007/978-3-030-57808-4_11.
38. Knuth, D.E.: The art of computer programming, vol. 3. Pearson Education (1997)
39. Liu, C., Zheng, Z., Zou, G.: Key reuse attack on newhope key exchange protocol. In: International Conference on Information Security and Cryptology. pp. 163–176. Springer (2018)

40. Lu, X., Liu, Y., Jia, D., Xue, H., He, J., Zhang, Z., Liu, Z., Yang, H., Li, B., Wang, K.: Lac: Lattice-based cryptosystems algorithm specification and supporting documentation. In: Submission to the NIST post-quantum project (2019) (2019), https://eprint.iacr.org/2018/1009.pdf.
41. Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 1–23. Springer (2010)
42. Mehlhorn, K.: Nearly optimal binary search trees. Acta Informatica **5**(4), 287–295 (1975)
43. Menezes, A., Ustaoglu, B.: On reusing ephemeral keys in diffie-hellman key agreement protocols. International Journal of Applied Cryptography **2**(2), 154–158 (2010)
44. Moody, D.: 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.
45. Moody, D., Alagic, G., Apon, D.C., Cooper, D.A., Dang, Q.H., Kelsey, J.M., Liu, Y.K., Miller, C.A., Peralta, R.C., Perlner, R.A., 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
46. Okada, S., Wang, Y., Takagi, T.: Improving key mismatch attack on newhope with fewer queries. IACR Cryptol. ePrint Arch. **2020**, 585 (2020)
47. Peikert, C.: Lattice cryptography for the internet. In: International workshop on post-quantum cryptography. pp. 197–219 (2014)
48. Qin, Y., Cheng, C., Ding, J.: A complete and optimized key mismatch attack on nist candidate newhope. In: European Symposium on Research in Computer Security. pp. 504–520. Springer (2019)
49. Qin, Y., Cheng, C., Ding, J.: An efficient key mismatch attack on the nist second round candidate kyber. IACR Cryptology EPrint Archive (2019), https://eprint.iacr.org/2019/1343.
50. Ravi, P., Roy, S.S., Chattopadhyay, A., Bhasin, S.: Generic side-channel attacks on cca-secure lattice-based pke and kems. IACR Transactions on Cryptographic Hardware and Embedded Systems pp. 307–335 (2020)
51. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. Journal of the ACM **56**(6), 1–40 (2009)
52. Vacek, J., Václavek, J.: Key mismatch attack on newhope revisited. Tech. rep., Cryptology ePrint Archive, Report 2020/1389 (2020)
53. Wang, K., Zhang, Z., Jiang, H.: Key recovery under plaintext checking attack on lac. In: International Conference on Provable Security. pp. 381–401. Springer (2020)
54. Xagawa, K., Ito, A., Ueno, R., Takahashi, J., Homma, N.: Fault-injection attacks against nist's post-quantum cryptography round 3 kem candidates. IACR Cryptology EPrint Archive (2021), https://ia.cr/2021/840
55. Zhang, X., Cheng, C., Ding, R.: Small leaks sink a great ship: An evaluation of key reuse resilience of pqc third round finalist ntru-hrss. ICICS2021 (Accepted) (2021), https://ia.cr/2021/168

A Bounds for other candidates

We list our lower bounds for key mismatch attacks against LAC, Round5 and Three Bears in Table 7. For each scheme, we give the ranges of coefficients, number of unknowns, and whether the Encode/Decode and Compress/Decompress are employed or not. We also report the average number of queries in our proposed bounds.

Table 7: Key mismatch attacks against LAC, Round5 and Three Bears

Schemes	$\mathbf{s}_A$ & $\mathbf{e}$ Ranges	Encode Decode	Comp Decomp	Unknowns	E(#Queries) Bounds
LAC128				512	553
LAC192	[-1,1]	✓	/	1024	1106
LAC256				1024	1398
Round5 R5ND_1				618	722
Round5 R5ND_3	[-1,1]	/	✓	786	1170
Round5 R5ND_5				1018	1446
BabyBear	[-1,1]			320	520
MamaBear	[-2,2]	/	✓	320	680
PapaBear	[-3,3]			320	738

B Improved practical key mismatch attacks

In this section, according to the proposed bounds, we discuss how to launch the practical key mismatch attacks on NewHope, LAC and Round5.

B.1 Improved key mismatch attacks on NewHope

In a key mismatch attack on NewHope, we build an Oracle $\mathcal{O}$ to simulate the process of NewHope.KEM.Dec(). The inputs of $\mathcal{O}$ are $(\mathbf{P}_B, \overline{\mathbf{c}})$ and K_B. The Oracle honestly executes the decryption to get K_A. Then it, compares K_A and K_B, if they are equal, it returns 1, otherwise it returns 0.

As far as we know, the best practical key mismatch attack on NewHope1024 given in [52] needs 3197 queries, which is still higher than our theoretical bound 3127. Based on it, we propose a method that can further decrease the number of queries to 3180. Here we take NewHope1024 as an example to show how to launch the attack.

Main idea. Our improved attack method is on the basis of Mehlhorn's Rule II in Nearly Optimal Binary Search Tree [42], i.e., for every given probability range, we always select the root in a way that the differences between sums of weights of its left subtree and right subtree are as small as possible.

In a key mismatch attack, we assume that Alice's public key $\mathbf{P}_A$ is always reused, and the adversary $\mathcal{A}$'s target is to get the secret key $\mathbf{s}_A$ of Alice. In order to achieve the target, $\mathcal{A}$ needs to set proper parameters.

Recall that NewHope1024 uses D-4 encoding, which means 4 coefficients $\mathbf{s}_A[i]$, $\mathbf{s}_A[i+256]$, $\mathbf{s}_A[i+512]$, $\mathbf{s}_A[i+768]$ are operated at a time. We assume that the adversary $\mathcal{A}$ wants to recover the *i-th* quadruplet, then he needs to properly select v_b, and parameters $(\mathbf{P}_B, \overline{\mathbf{c}})$.

In our improved attack, in Step 1 by precomputing the probabilities of all the quadruplets, along with the outputs of Oracle corresponding to selected parameters $(\mathbf{P}_B, \bar{\mathbf{c}})$ and all the quadruplets, $\mathcal{A}$ can choose the proper parameters which relate each quadruplet to a leaf node in a binary tree. Finally, in Step 2 by repeatedly querying the Oracle and getting the corresponding sequence of returned values, $\mathcal{A}$ can decide the quadruplets.

Step1: The pre-computation phase. In this step, the adversary $\mathcal{A}$ needs to compute the probabilities of all the quadruplets, along with the outputs of Oracle corresponding to selected parameters $(\mathbf{P}_B, \overline{\mathbf{c}})$ and all the quadruplets which is denoted as O_A, and constructs a corresponding binary recovery tree.

In the following, we construct the corresponding nearly optimal binary recovery tree T. Here, a nearly optimal binary tree means a binary tree in which the sum of the probabilities of quadruplets of the left subtree and the right subtree should be as equal as possible. We require T to be nearly optimal since in this way we can recursively divide all the possible quadruplets into almost equal two parts with lower time complexity.

We set the sum of the probabilities of quadruplets of a non-leaf node's left subtree and the right subtree as p_0 and p_1, respectively. The nearly optimal binary recovery tree T should satisfy the following properties.

1. For each non-leaf node, its corresponding p_0 and p_1 should be as equal as possible.

2. For each non-leaf node, if the Oracle returns 0, it corresponds to the left subtree of the current node, otherwise it corresponds to its right subtree.

First, we traverse the precomputing O_A to find one appropriate parameter $P = (\mathbf{P}_B, \overline{\mathbf{c}})$ which satisfies the above two properties. The construction of tree T starts from the root node with index $i = 0$. After obtaining the appropriate parameter P, we insert the root node and P into the 0-*th* position in T. Then we recursively build the left subtree and the right subtree for the root node, respectively. Finally, all the possible quadruplets are stored in the leaf nodes, and parameters $\mathbf{P}_B$ and $\overline{\mathbf{c}}$ are stored in the non-leaf nodes.

Step2: The recovery phase. In this step, the adversary $\mathcal{A}$ tries to decide the quadruplet according to the precomputed binary tree T.

We show how the adversary $\mathcal{A}$ decides the *i-th* quadruplet in Algorithm 6. Specifically, $\mathcal{A}$ first starts from the root node of the precomputed binary tree T, and sets P as the parameters stored in the root node. Then, he accesses the Oracle, if it returns 0, $\mathcal{A}$ accessed the left subtree of the root node, otherwise $\mathcal{A}$

Algorithm 6 Determining each quadruplet

Input: T
Output: the quadruplet
1: Set $node = T.root$
2: **while** $node$ is not a leaf node **do**
3: Set P the parameter stored in the $node$
4: $v = \text{Oracle}(P)$
5: **if** $v = 0$ **then**
6: $node = node.leftnode$
7: **else**
8: $node = node.rightnode$
9: **end if**
10: **end while**
11: **Return** the quadruplet stored in the $node$

accessed the right subtree of the root node. Next, $\mathcal{A}$ repeats the following two steps until the current node he accesses is a leaf node.

1. $\mathcal{A}$ judges whether the current node is a leaf node, and if it is, he directly returns the value of the quadruplet stored in the node. Otherwise, he sets P as the parameters stored in the current node, and accesses the Oracle again.

2. If the Oracle returns 0, he sets $node = node.leftnode$, otherwise $node = node.rightnode$.

Parameter choices: The total number of queries depend on the precomputed binary tree T. Recall that in Step1, when we construct the binary tree T, we need to compute the probabilities of all the quadruplets and O_A, the latter is associated with selected parameters $(\mathbf{P}_B, \overline{\mathbf{c}})$, thus the selected parameters determines the number of queries.

Table 8: The relationship between g, success probability of Hypothesis 1, and average number of queries on NewHope1024

g	[0,383]	[384,512]	[384,534]	[384,768]	[384,819]
Success probability (%)	100	99.999	99.999	94.577	85.811
E(#Queries)	3574.953	3179.215	3206.605	3174.853	3174.085

Hypothesis 1. *The adversary* $\mathcal{A}$ *sets* $v_b = \{1, 0, 0, \ldots, 0\}$, $\mathbf{P}_B = gx^{-i}$, $\overline{\mathbf{c}} = \sum_{j=0}^{3} ((l_j + 4) \bmod 8)\, x^{256j}$. *Then the goal of* $\mathcal{A}$ *is to choose* $(\mathbf{P}_B, \overline{\mathbf{c}})$ *such that* $v_a = Decode(Decompress(\overline{\mathbf{c}} - \mathbf{P}_B \circ \mathbf{s}_A)) = \{b, 0, 0, \ldots, 0\}$, *where* $b \in \{0, 1\}$.

Moreover, while selecting parameters $(\mathbf{P}_B, \overline{\mathbf{c}})$, $\mathcal{A}$ needs to guarantee that Hypothesis 1 holds with nearly 100% probability. Otherwise when the output of Oracle is 0, $\mathcal{A}$ does not know whether the mismatch is due to the 0-th position or other positions.

In order to get the best parameter, we traverse and compute the success rate of Hypothesis 1 through the whole value interval of g, and show the relationship

among $g \in [0, 819]$, the success probability of Hypothesis 1 and the average number of queries in Table 8 above. Considering the success probability and the number of queries, we finally decide the optimal interval of parameter g, i.e. $[384, 512]$. In Step2, while selecting parameter $g \in [384, 512]$, the average number of queries needed by the adversary to get each quadruplet is 12.41881, and there are 256 unknown quadruplets in a secret key $\mathbf{s}_A$. Therefore, in total we need 3179.21536 queries to completely recover $\mathbf{s}_A$.

B.2 Improved key mismatch attacks on LAC

Although there are three versions of LAC with different security levels, the parameters in the proposed key mismatch attacks are the same. In the attack, the adversary needs to modify three parameters: $\mathbf{e}_B[0]$, $\mathbf{e}'_B[vb-1]$ and $\mathbf{e}'_B[2vb-1]$. Here, $vb = l_v = 400$, and l_v is a parameter set in LAC. And next we will show how to recover $\mathbf{s}_A[0]$.

1. We set $\mathbf{e}_B[0] = 124$, $\mathbf{e}'_B[vb-1] = 1$ and $\mathbf{e}'_B[2vb-1] = 1$ first, then $\{\mathbf{S}_0, \mathbf{S}_1, \mathbf{S}_2, \mathbf{S}_3, \mathbf{S}_4, \mathbf{S}_5, \mathbf{S}_6, \mathbf{S}_7, \mathbf{S}_8\}$ can be divided into two parts based on the returned value of the first Oracle:
   - If $\mathcal{O} \to 0$: $\mathbf{s}_A[0]$ belongs to $\{\mathbf{S}_3, \mathbf{S}_4, \mathbf{S}_5, \mathbf{S}_6, \mathbf{S}_7, \mathbf{S}_8\}$, next step 2, step 3 and step 5 will be executed.
   - If $\mathcal{O} \to 1$: $\mathbf{s}_A[0]$ belongs to $\{\mathbf{S}_0, \mathbf{S}_1, \mathbf{S}_2\}$, then go to step 4 and step 5.
2. If the oracle returns 0 in step 1, then we set $\mathbf{e}_B[0] = 124$, $\mathbf{e}'_B[vb-1] = 0$ and $\mathbf{e}'_B[2vb-1] = 0$:
   - If $\mathcal{O} \to 0$: $\mathbf{s}_A[0]$ belongs to $\{\mathbf{S}_5, \mathbf{S}_6, \mathbf{S}_7, \mathbf{S}_8\}$, next step 3 will be proceeded.
   - If $\mathcal{O} \to 1$: $\mathbf{s}_A[0]$ belongs to $\{\mathbf{S}_3, \mathbf{S}_4\}$, and next turn to step 5.
3. If the oracle returns 0 in step 2, then we set $\mathbf{e}_B[0] = 63$, $\mathbf{e}'_B[vb-1] = 63$ and $\mathbf{e}'_B[2vb-1] = 63$:
   - If $\mathcal{O} \to 0$: $\mathbf{s}_A[0]$ belongs to $\{\mathbf{S}_7, \mathbf{S}_8\}$, next go to step 5.
   - If $\mathcal{O} \to 1$: $\mathbf{s}_A[0]$ belongs to $\{\mathbf{S}_5, \mathbf{S}_6\}$, next go to step 5.
4. If the oracle returns 1 in step 1, then we set $\mathbf{e}_B[0] = 125$, $\mathbf{e}'_B[vb-1] = 0$ and $\mathbf{e}'_B[2vb-1] = 0$:
   - If $\mathcal{O} \to 0$: $\mathbf{s}_A[0]$ belongs to $\{\mathbf{S}_1, \mathbf{S}_2\}$, then turn to step 5.
   - If $\mathcal{O} \to 1$: We can determine $\mathbf{s}_A[0] = \mathbf{S}_0$.
5. Similarly we only need to distinguish the two coefficients in $\{\mathbf{S}_7, \mathbf{S}_8\}$, $\{\mathbf{S}_5, \mathbf{S}_6\}$, $\{\mathbf{S}_3, \mathbf{S}_4\}$, and $\{\mathbf{S}_1, \mathbf{S}_2\}$. As long as the appropriate parameters are selected, only one query is needed.

B.3 Improved key mismatch attacks on Round5

Round5 does not use D-2 Encode/Decode functions. Although there are three different versions of Round5 R5ND with different security levels, their attack process is the same, except that the parameters $P_B = h$ ($h = h_1$ or h_2) chosen by the adversary are different. Specifically, the adversary selects h_1/h_2 as 44/-44, 120/-120 and 144/113, and the process of recovering $\mathbf{s}_A[0]$ is shown as follows.

1. We set $h = h_1$ first, then $\{\mathbf{S}_0, \mathbf{S}_1, \mathbf{S}_2\}$ can be divided into two parts based on the returned value of the first Oracle:
   - If $\mathcal{O} \to 0$: We can determine $\mathbf{s}_A[0] = \mathbf{S}_2$.
   - If $\mathcal{O} \to 1$: $\mathbf{s}_A[0]$ belongs to $\{\mathbf{S}_0, \mathbf{S}_1\}$.
2. When $h = h_1$, if the oracle returns 0 then we go on setting $h = h_2$:
   - If $\mathcal{O} \to 0$: $\mathbf{s}_A[0] = \mathbf{S}_0$.
   - If $\mathcal{O} \to 1$: $\mathbf{s}_A[0] = \mathbf{S}_1$.