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

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

[1] China University of Geosciences, Wuhan, 430074, China
{qy52hz,chengchi}@cug.edu.cn
[2] Key Laboratory of Mathematics Mechanization, Academy of Mathematics and Systems Science, Chinese Academy of Sciences
[3] Institute of Information Engineering, Chinese Academy of Sciences
[4] Tsinghua University, Beijing
jintai.ding@gmail.com

Abstract. Most submitted lattice-based key encapsulation mechanisms (KEMs) on the second or third round list of the NIST standardization follow a similar structure: First a CPA secure scheme is constructed, which is then converted to a CCA secure one. The research of the key reuse attacks against the CPA secure ones is important in two folds: First, it is an important part of the cryptographic assessment of the ongoing NIST standardization. Secondly, it helps the design of CCA-secure authenticated key exchange directly from LWE, without FO transform. There have been a number of key mismatch attacks on these CPA secure versions when the public key is reused. However, a unified method to evaluate their resilience under key mismatch attacks is still missing. Since the key index of the efficiency of these attacks is the number of queries (matches and mismatches) needed to successfully mount such an attack, in this paper, we propose and develop a systematic approach to find the 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 certain Shannon entropy. The approach means that one cannot find a better attack with fewer queries than this lower bound. The introduction of the optimal BRT approach enables us to understand why, for some schemes, there is a big gap between the theoretical bounds and practical attacks, in terms of the number of queries needed. This further leads us to improve the existing attacks. Especially, we can reduce the needed queries against Frodo640 by 71.99% , LAC256 by 82.81%, and Newhope1024 by 97.44%.

1 Introduction

The Diffie-Hellman (DH) key exchange [21] and its Elliptic Curve counterpart have played a fundamental role in many standards, such as Transport Layer Se-

curity (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 [29], 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 [39]. 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 [17]. There are 17 public key encryption (PKE) or key encapsulation mechanism (KEM) candidates in the second round [2], among which 9 are based on lattice [1]. On the third-round list, there are still 3 lattice-based KEMs out of the 4 finalists [40].

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 transformations such as the Fujisaki-Okamoto transformation [26]. We have to point out that there is no security guarantee on the CPA secure ones when the public key is reused. However, it is an important part of the cryptographic assessment of these candidates to understand their key-reuse resilience in even misuse situations. Therefore, the line of research focusing on the key reuse attacks against the CPA secure ones has been actively studied.

There are two kinds of key reuse attacks. 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 the key mismatch attack, which launches the attack by simply knowing whether the shared two keys match or not. In 2015, Kirkwood et al. from the US National Security Agency (NSA) announced that there may exist key reuse attacks against lattice-based post-quantum key exchange protocols, without giving any details [34]. Later, Fluhrer showed in [25] that if the public key of the Ring Learning with Errors (RLWE) based key exchange is reused, then this protocol could be broken. In [22], Ding, Alsayigh and Saraswathy first launched signal leakage attacks to the key exchange protocol in [24] by using the leaked information about the secret key from the signal messages. Then, Liu, Zheng and Zou proposed a signal leakage attack [36] against the reconciliation-based NewHope-Usenix key exchange protocol [6].

The idea of key mismatch attack was first proposed by Ding, Fluhrer and Saraswathy [23] against the one-pass case of the protocol in [24]. 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 [43]. In [41], Okada, Wang, and Takagi improved the method in [43] to further reduce the number of queries. The work of [44] gave a similar key mismatch attack on

Kyber. In [27] a key mismatch attack was proposed against LAC, requiring up to 8 queries for each coefficient.

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, Băetu et al. tried to answer this problem, but most of their result is related to the first-round candidates [9]. In a recent work of Huguenin-Dumittan and Vaudenay [33], 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. The main contributions of this paper include:

- We propose a unified method to find lower bounds for all the lattice-based NIST candidates. 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 bound. Further analysis shows that the calculation of these bounds becomes essentially the computation of certain Shannon entropy, which means that one cannot find a better attack with fewer queries.
- 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, Round5, and Saber [41,33,43]. 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 [33] and [41], we have improved attacks against Frodo640 and LAC256 with 71.99% and 82.81% reduced number of queries, and 97.44% reduced queries against Newhope1024.
- 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 reduce the number of queries needed.

2 Preliminaries

2.1 Lattice-based key encapsulation mechanisms

In [19], 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 [24], 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 [42] Peikert proposed a KEM using a similar error correction mechanism, and then in [15] 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 Frodo [4], NewHope [5,3], LAC [37], Kyber [14,7], Threebears [30], Round5 [8], Saber [20] NTRU [16] and NTRU Prime[13]. Just recently, NIST [40] 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: Frodo 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 [45], Lyubashevsky-Peikert-Regev [38], and lattice-based key exchange scheme proposed by Ding, Xie and Lin [24]. The other is NTRU [31] and NTRUprime [12].

In Table 1 we present the meta structure of the CPA-secure KEMs in the first category of the NIST second-round candidates. Although there is no CPA-secure version in the Kyber KEM and the authors have warned 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.

Let $\mathcal{R}$ be a ring. For example, in NewHope $\mathcal{R} = \mathbb{Z}_q[x]/(x^n + 1)$, where $\mathbb{Z}_q$ denotes the ring of integers modulo a prime q. Throughout the paper matrices and vectors are in bold. In the key generation algorithm KEM.Gen(), Alice first generates $\mathbf{a} \in \mathcal{R}$ using a predefined seed. The coefficients of secret $\mathbf{s}_A$ and error $\mathbf{e}_A$ are randomly selected from $\mathcal{R}$ following a distribution χ. Generally, χ is chosen to be the discrete Gaussian distribution or the central binomial distribution. Different KEMs have different sampling parameters, so the range of coefficients of private key and errors could also be different, which significantly affects the number of queries needed in launching the key reuse attack.

The output P_A is calculated using $\mathbf{a} \circ \mathbf{s}_A + \mathbf{e}_A$. Here the notation $\circ$ denotes an operation that is determined by the hardness assumption of these candidates. For example, in Frodo $\circ$ means matrix multiplication while in NewHope it is the

Table 1: The structure of CPA-secure KEM

Alice		**Bob**
1. ▷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.$K_A \leftarrow \mathbf{H}(\mathbf{m}'\|\|(P_B, \bar{\mathbf{c}}))$		

polynomial multiplication. The input of the encryption algorithm KEM.Enc is a random binary string $\mathbf{m}$ and the public key P_A, and the output is $(P_B, \bar{\mathbf{c}})$, where P_B is Bob's public key and $\bar{\mathbf{c}}$ is the ciphertext. The decryption algorithm KEM.Dec takes $(P_B, \bar{\mathbf{c}})$ and the secret key $\mathbf{s}_A$ as input, and outputs $\mathbf{m}'$.

Another important factor in affecting the needed number of queries is whether the Encode/Decode and Compress/Decompress functions are employed or not. Some KEMs do not encode the message as Step 3.4 in Table 1 but just let $\mathbf{k} = \mathbf{m}$, such as Kyber, whereas some other KEMs do to decrease the decryption failure, such as Newhope512 and LAC use D-2 lattice code, and Newhope1024 uses D-4 lattice code. Below we list the Encode algorithm that encodes $\mathbf{m}$ of length λ into $\mathbf{k}$ with length N, and Decode algorithm that recovers one bit of $\mathbf{m}$ according to the v-bit coefficient in $\mathbf{k}$, when D-v lattice code is used where v is 2 or 4.

The Compress/Decompress function can also be viewed as a coefficient-wise modulus switching. That is, it can change the coefficient from module q to module p. Newhope, Kyber, Frodo, Saber and Round5 use this method. For the Compress function, each coefficient of $\mathbf{c}$ can be converted to the new module p from the existing module q by multiplying p and then performing rounding division by q:

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

The Decompress function operates in an opposite way:

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

Algorithm 1 The Encode and Decode functions

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

◇ **Decode**$(\mathbf{k}, v)$
Input: $\mathbf{k} \leftarrow \{0, \frac{q-1}{2}\}^{v\lambda}, v$
Output: $\mathbf{m}'$
for $i = 0$ *to* $\lambda - 1$ **do**
  s=0
  $s+ = \sum_{j=0}^{v-1} |\mathbf{K}[i \cdot v + j] - \frac{q-1}{2}|$
  if $s < \frac{v \cdot q}{4}$ **then**
    $\mathbf{m}'[i] = 1$
  else
    $\mathbf{m}'[i] = 0$
  end if
end for
Return $\mathbf{m}'$

2.2 Key mismatch attacks

In a key mismatch attack, Alice's public key P_A is reused. The adversary $\mathcal{A}$ impersonate 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

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

◇ **key mismatch attack**
Input: Alice's pk P_A and Oracle $\mathcal{O}$
Output: 0 or 1
$\mathbf{s}'_A \leftarrow \mathcal{A}^{\mathcal{O}}(P_A)$
if $\mathbf{s}'_A = \mathbf{s}_A$ **then**
  Return 1
else
  Return 0
end if

3 Optimal bound of 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 $\mathbf{s}_A$ (see Section 4.2 for more details).

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 key 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 recovering 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 recovers Alice's secret key $\mathbf{s}_A$ one coefficient block by one coefficient block. 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, in Kyber, the coefficients of $\mathbf{s}_A$ are drawn from $\{-2, -1,\ 0,\ 1,\ 2\}$. Since there are no Encode/Decode functions, we let $\mathbf{S} = \{-2, -1,\ 0,\ 1,\ 2\}$. In LAC, the coefficients of $\mathbf{s}_A$ are selected from $\{-1,\ 0,\ 1\}$. Since D-2 is used, a coefficient block contains 2 coefficients of $\mathbf{s}_A$. Therefore, we let $\mathbf{S}$ be $\{(0,0), (0,1), (1,0), (-1,0), (0,-1), (1,1), (-1,-1), (1,-1), (-1,1)\}$.

For every $\mathbf{S}_i$, denote by P_i the probability that $\mathbf{S}_i$ happens when $\mathbf{s}_A$ is generated from the distribution χ. 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, by repeatedly accessing the Oracle with properly selected parameters, the adversary gets a sequence of returned values from the Oracle. We use $\bar{s}$ to represent the sequence. For example, by accessing $\mathcal{O}$ five times, the adversary $\mathcal{A}$ gets five returned values, which may be $\bar{s} = 00001$. The length of $\bar{s}$ is represented by $l_{\bar{s}}$, which also denotes the number of times the adversary accesses the Oracle.

The adversary aims to decide each coefficient block of the secret key, which can be any element in $\mathbf{S}$. Let Q_i represent the number of queries needed to recover each $\mathbf{S}_i$, which must be the length for some $\bar{s}$, then the average number

of queries required to recover one $\mathbf{S}_i$ is:

$$E(\mathbf{S}) = \sum_{i=0}^{n-1} P_i Q_i. \tag{1}$$

Our goal is to minimize $E(\mathbf{S})$ by running over the set of all possible Q_i's. Note that each Q_i corresponds to a bit string $\bar{s}_i$ returned by the Oracle and it is sufficient for us to determine $\mathbf{S}_i$ from $\bar{s}_i$ in the attack. Hence, the constraint for Q_i's is at least every Q_i is equipped with a (w.l.o.g. only one) bit string $\bar{s}_i$ and $\bar{s}_i$ can not be a prefix of $\bar{s}_j$ for $i \neq j$.

We next compute the optimal value $E(\mathbf{S})$ under this constraint and then the bound E(#Queries) can be calculated by multiplying $\min E(\mathbf{S})$ by the number of unknowns in the secret key, which is obviously a lower bound for the key mismatch attack.

Binary recovering tree. Define the BRT T associated with $\mathbf{S} = \{\mathbf{S}_0, \mathbf{S}_1, \cdots, \mathbf{S}_{n-1}\}$ as below: it is a binary tree with n leaf nodes and 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. Then for every i, the path from the root node to the leaf node $\mathbf{S}_i$ implies a bit string $\bar{s}_i$ consisting of the nodes on the path, which yields to a value Q_i that is the length of $\bar{s}_i$, also known as the depth $\mathrm{depth}_T(\mathbf{S}_i)$ of leaf node $\mathbf{S}_i$. This means that a BRT will yield a feasible solution $\{Q_i\}$ for computing $E(\mathbf{S})$. On the other hand, it is obvious that we can construct a binary recovering tree linking any feasible Q_i with the corresponding $\bar{s}_i$.

Hence, we can transform the problem of finding the optimal value of $E(\mathbf{S})$ to the problem of finding a binary recovering tree associated with $\mathbf{S}$ to minimize

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

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 [32,35]. 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).

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. Further, we can have 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 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, the above calculated result is the bound for the minimum average number of queries. Moreover, set* $H(\boldsymbol{S})$ *the Shannon entropy for* $\boldsymbol{S}$*, then we have*

$$H(\boldsymbol{S}) \leq \min\ \mathrm{E}(\mathbf{S}) \leq H(\boldsymbol{S}) + 1 \tag{3}$$

Proof. Our first result comes from the facts in Section 5.8 of [18]. That is, it is impossible to find any other code with a lower expected length than the code constructed by Huffman coding. Furthermore, from Theorem 5.4.1 [18], we have $H(\mathbf{S}) \leq \min\ \mathrm{E}(\mathbf{S}) \leq H(\mathbf{S}) + 1$.

3.2 Lower bounds for key mismatch attacks on NIST candidates

Lower bounds for Kyber In this subsection, we take Kyber as an example to show how to find the optimal BRT to get the bound. Kyber 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 $\mathbf{S}_4 = -2$.

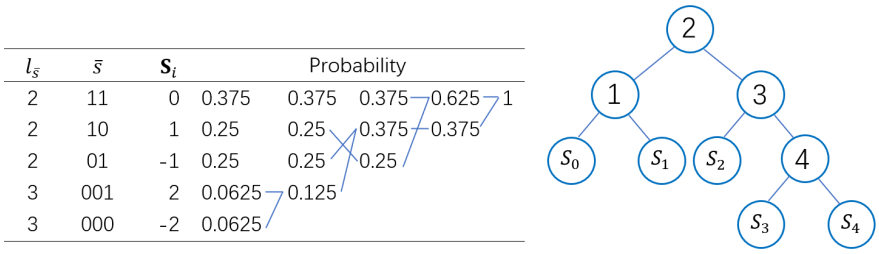

Fig. 1: Finding the optimal BRT for Kyber by using Huffman coding

As shown in Fig. 1, 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. 1. The corresponding $\bar{s}$ represents how to encode each $\mathbf{S}_i$, while $l_{\bar{s}}$ is the code length.

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

calculated as

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

We can see that this is in accordance with our proposed Theorem 1. Similarly, we can get the bounds for Kyber512, Kyber768, and Kyber1024, which are 1088, 1632, and 2176, respectively.

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 $[-8, 8]$.

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 $\mathbf{S}_i$. To simplify the calculation, we only keep those $\mathbf{S}_i$ with occurrence probability greater than or equal to 0.000001. So here we let $n = 241$. 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, we also keep those $\mathbf{S}_i$ with occurrence probability greater than 0.000001, which results in $n = 16,481$. Similarly, we have E(#Queries)= 3103 for Newhope1024.

Lower bounds for NIST Candidates In Table 2, we present the lower bounds for key mismatch attacks against the following second or third round NIST candidates: Newhope, Kyber, LAC, Frodo, Saber, and Round5. 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 and in known practical attacks (Italic). For other NIST candidate KEMs, we report their results in Table 13 in the Appendix.

From Table 2, we can see that for some KEMs, there is a huge gap in terms of number of queries between the theoretical bound and practical attacks. For example, so far the best key mismatch attack against NewHope is given in [41], in which the number of queries for Newhope1024 is $233,803$, while our bound is just 3103. Since we have built an optimal BRT for each KEM, in the following we can use it to help design the practical attacks.

4 The improved practical key mismatch attacks on NIST candidates

4.1 The improved practical attacks on Kyber

Kyber. Kyber is on the third-round list of NIST competition, and is regarded as one of the most promising one for the final standard. In Kyber all the se-

Table 2: Key mismatch attacks against lattice-based NIST KEMs. For each scheme, we give the ranges of coefficients and the number of unknowns, as well as whether the Encode/Decode and Compress/Decompress are employed or not. We also report the minimum average number of queries in our proposed bounds, as well as other existing results (Italic). Here "-" means no result is given.

Schemes	$\mathbf{s}_A$ & $\mathbf{e}$ Ranges	Encode Decode	Comp Decomp	Unknowns	E(#Queries) Bounds/*Existing*
Newhope512	[-8,8]	✓	✓	512	1568/-
Newhope1024				1024	3103/*233,803*[41]
Kyber512	[-2,2]	/	✓	512	1088/*1401*[44]
Kyber768				768	1632/*1855*[44]
Kyber1024				1024	2176/*2475*[44]
LightSaber	[-5,5]	/	✓	512	1412/*2048*[33]
Saber	[-4,4]			768	1986/-
FireSaber	[-3,3]			1024	2432/-
Frodo640	[-12,12]	/	✓	5120	18,227/*65536*[33]
Frodo976	[-10,10]			7808	25,796/-
Frodo1344	[-6,6]			10,752	27,973/-
LAC128	[-1,1]	✓	/	512	553/*1024*[28]
LAC192				1024	1106/*2048*[28]
LAC256				1024	1398/*8192*[28]
Round5 R5ND_1	[-1,1]	/	✓	618	722/-
Round5 R5ND_3				786	1170/-
Round5 R5ND_5				1018	1446/-

cret keys and error vectors are sampled from a centered binomial distribution $\mathbf{B}_\eta$. Here $\eta = 2$ and $\mathbf{B}_\eta$ is generated using $\sum_{i=1}^{\eta}(a_i - b_i)$, where a_i and b_i are randomly sampled from $\{0, 1\}$. The CPA-secure KEM of Kyber also consists of three parts: the key generation algorithm Kyber.KEM.Gen(), the encryption algorithm Kyber.KEM.Enc() and the decryption algorithm Kyber.KEM.Dec().

In Kyber.KEM.Gen(), 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 $q = 3329$. The coefficients of $\mathbf{a}$ are polynomials in R_q. Another parameter k is set to be 2, 3 or 4, which is in accordance with the three different security levels. That is, Kyber-512, Kyber-768, and Kyber-1024, respectively. Then, she samples $\mathbf{s}_A$ and $\mathbf{e}_A$ from $\mathbf{B}_\eta$ to calculate $\mathbf{P}_A = \mathbf{a} \circ \mathbf{s}_A + \mathbf{e}_A$. The output is the key pair $(\mathbf{s}_A, \mathbf{P}_A)$.

The inputs to the encryption algorithm Kyber.KEM.Enc() are the public key $\mathbf{P}_A$ and a random binary string $\mathbf{m}$. In this part, Bob generates the same matrix $\mathbf{a}$ and samples $\mathbf{s}_B, \mathbf{e}_B \in R_q^k$ and $\mathbf{e}'_B \in R_q$ from $\mathbf{B}_\eta$. Next, he calculates $\mathbf{P}_B = \mathbf{a}^T \circ \mathbf{s}_B + \mathbf{e}_B$, $\mathbf{v}_B = \mathbf{P}_A^T \circ \mathbf{s}_B + \mathbf{e}'_B + \mathbf{Decompress}_q(\mathbf{m}, 2)$, $\mathbf{c}_1 = \mathbf{Compress}_q(\mathbf{P}_B, 2^{d_{\mathbf{P}_B}})$ and $\mathbf{c}_2 = \mathbf{Compress}_q(\mathbf{v}_B, 2^{d_{\mathbf{v}_B}})$. In the end, he calculates the shared key as $K_B \leftarrow \mathbf{H}(\mathbf{m}||(\mathbf{P}_B, (\mathbf{c}_1, \mathbf{c}_2)))$, where $\mathbf{H}$ represents the hash function. The resulted outputs of Kyber.KEM.Enc() are $\mathbf{P}_B$ and $(\mathbf{c}_1, \mathbf{c}_2)$. The parameters $d_{\mathbf{P}_B}$ and $d_{\mathbf{v}_B}$ vary in different security levels of Kyber. Specifically, $d_{\mathbf{P}_B}$ is set as 10, 10 and 11, corresponding to Kyber-512, Kyber-768 and Kyber-1024, respectively. While the corresponding values of $d_{\mathbf{v}_B}$ are 3, 4, and 5.

For the decryption algorithm Kyber.KEM.Dec(), the inputs are $\mathbf{P}_B$, $(\mathbf{c}_1, \mathbf{c}_2)$ and Alice's secret key $\mathbf{s}_A$. Then Alice 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 $\mathbf{m}' = \mathbf{Compress}_q(\mathbf{v}_A - \mathbf{s}_A^T \circ \mathbf{u}_A, 2)$. After that she calculates the shared key as $K_A \leftarrow \mathbf{H}(\mathbf{m}'||(\mathbf{P}_B, (\mathbf{c}_1, \mathbf{c}_2)))$. In this part, the output is $\mathbf{m}'$.

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 $\mathbf{s}_A[0]$.

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 $\mathbf{c}_2[0] = h$.

With the $\mathbf{P}_B$, $(\mathbf{c}_1, \mathbf{c}_2)$, and K_B received from the adversary, the Oracle calculates

$$\begin{aligned}\mathbf{m}'[0] &= \mathbf{Compress}_q((\mathbf{v}_A - \mathbf{s}_A^T\mathbf{u}_A)[0], 1)\\ &= \mathbf{Compress}_q(\mathbf{v}_A[0] - (\mathbf{s}_A^T\mathbf{u}_A)[0], 1)\\ &= \left\lceil \frac{2}{q}\left(\mathbf{v}_A[0] - (\mathbf{s}_A^T\mathbf{u}_A)[0]\right)\right\rfloor \bmod 2.\end{aligned} \tag{5}$$

Since $\mathbf{v}_A[0] = \left\lceil \frac{q}{32}h \right\rfloor$ and $(\mathbf{s}_A^T\mathbf{u}_A)[0] = \mathbf{s}_A^T[0]\mathbf{u}_A[0] = \mathbf{s}_A^T[0]\left\lceil \frac{q}{32} \right\rfloor$, it holds that

$$\mathbf{m}'[0] = \left\lceil \frac{2}{q}\left(\left\lceil \frac{q}{32}h \right\rfloor - \mathbf{s}_A^T[0]\left\lceil \frac{q}{32} \right\rfloor\right)\right\rfloor \bmod 2. \tag{6}$$

Therefore, by letting $h = 8$, we have the following result: If $\mathbf{s}_A^T[0] \in [-2, -1]$, $\mathbf{m}'[0] = 1$, the Oracle outputs 1. Otherwise, if $\mathbf{s}_A^T[0] \in [0, 2]$, $\mathbf{m}'[0] = 0$, the Oracle outputs 0. In this way, we can distinguish which subinterval $\mathbf{s}_A^T[0]$ belongs to by only one query. Similar to the above process, $\mathcal{A}$ needs to select the appropriate h according to Table 3.

Table 3: 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 4: $\mathbf{S}_i$ 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

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 number of queries as possible. Table 3 shows how to choose h and how the States change according to the output of Oracle in Kyber512, Kyber768 and Kyber1024, respectively. The key mismatch attack always starts from State 1, and then the choice of h in the next State depends on the current Oracle 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.

Next, we show how the adversary $\mathcal{A}$ selects h until he can determine $\mathbf{s}_A[0]$ in Kyber1024.

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 4. 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 in as fewer queries as possible. We can also observe that the way we find $\bar{s}$ is similar to the Huffman coding [32].

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

For completeness, Table 3 shows how to choose h and how the States change according to the output of Oracle in Kyber512, Kyber768 and Kyber1024, respectively.

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, Kyber769 and Kyber512 are 2365.44, 1774.08 and 1182.72, respectively. Compared with the bound in Table 2, there is only a gap less than 9%.

In [44], 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, the improved practical key mismatch attack on Kyber we proposed is better than that in [44].

4.2 The improved practical attacks on Newhope

Newhope. There are also three parts in Newhope's CPA-secure KEM: Newhope.KEM.Gen(), Newhope.KEM.Enc() and Newhope.KEM.Dec().

In Newhope.KEM.Gen(), Alice generates a polynomial $\mathbf{a}$ in $\mathcal{R}_q$. 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. Then, Alice samples $\mathbf{s}_A$ and $\mathbf{e}_A$ to calculate $\mathbf{P}_A = \mathbf{a} \circ \mathbf{s}_A + \mathbf{e}_A$. The output is the keypair $(\mathbf{s}_A, \mathbf{P}_A)$.

In Newhope.KEM.Enc(), the inputs are Alice's public key $\mathbf{P}_A$ and a random binary string $\mathbf{m}$. Bob first generates the same $\mathbf{a}$ in $\mathcal{R}_q$, and then selects $\mathbf{s}_B$, $\mathbf{e}_B$, $\mathbf{e}'_B$ to calculate $\mathbf{P}_B = \mathbf{a} \circ \mathbf{s}_B + \mathbf{e}_B$. Next, he continues to calculate $v_b = \mathbf{H}_1(\mathbf{m})$, $\mathbf{k} = \text{Encode}(v_b)$, $\mathbf{c} = \mathbf{P}_A \circ \mathbf{s}_B + \mathbf{e}'_B + \mathbf{k}$ and $\bar{\mathbf{c}} = \text{Compress}(\mathbf{c})$. In the end, he calculates the shared key $K_B \leftarrow \mathbf{H}_2(v_b || (\mathbf{P}_B, \bar{\mathbf{c}}))$, here $\mathbf{H}_1$ and $\mathbf{H}_2$ are hash functions. The output is $(\mathbf{P}_B, \bar{\mathbf{c}})$.

In Newhope.KEM.Dec(), the inputs are Alice's secret key $\mathbf{s}_A$ and $(\mathbf{P}_B, \bar{\mathbf{c}})$. Alice directly calculates $\mathbf{c}' = \text{Decompress}(\bar{\mathbf{c}})$, $\mathbf{K}' = \mathbf{c}' - \mathbf{P}_B \circ \mathbf{s}_A$, $v_A = \text{Decode}(\mathbf{K}')$ and the shared key $K_A \leftarrow \mathbf{H}_2(v_A || (\mathbf{P}_B, \bar{\mathbf{c}}))$. In this part the output is v_A.

Improved practical attack 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 $(\mathbf{P}_B, \bar{\mathbf{c}})$ and K_B. The Oracle honestly executes Newhope.KEM.Dec() to get K_A. Then, he compares K_A and K_B. If they are equal, it returns 1, otherwise it returns 0.

So far as we know, the best practical key mismatch attack Newhope1024 given in [41] needs 233,803 queries, while the bound we give is 3103. There is a huge gap between the theory and practice. In the following, we further improve the parameter choices of [43] by the techniques of BRT, and finally reduce the needed queries dramatically.

In a key mismatch attack, we assume that Alice's public key $\mathbf{P}_A$ is always reused, and the secret key $\mathbf{s}_A$ of Alice is the Adversary $\mathcal{A}$'s target. Therefore, in order to achieve this target, $\mathcal{A}$ needs to select appropriate parameters.

Recall that Newhope1024 uses D-4 technology, so $\mathcal{A}$ can recover the value of four $\mathbf{s}_A$ at a time. We assume that $\mathcal{A}$ wants to recover $\mathbf{s}_A[i], \mathbf{s}_A[i+256], \mathbf{s}_A[i+512], \mathbf{s}_A[i+768]$.

At first, $\mathcal{A}$ sets v_b as $\{1, 0, \cdots, 0\}$, rather than randomly selecting $\mathbf{m}$ and computing $v_b \leftarrow \mathbf{H}_1(\mathbf{m})$.

Table 5: The choice of t and the States

	State 1	State 2	State 3	State 4	State 5	State 6	State 7	State 8	State 9	State 10
t_1	2047	3071	2457	4095	6143	12286	12289	1535	1755	1365
$\mathcal{O} \to 0$	State 8	State 3	$s_1 = 6$	$s_1 = 4$	$s_1 = 3$	$s_1 = 2$	$s_1 = 1$	State 10	$s_1 = 8$	State 11
$\mathcal{O} \to 1$	State 2	State 4	$s_1 = 5$	State 5	State 6	State 7	$s_1 = 0$	State 9	$s_1 = 7$	$s_1 = 9$

	State 11	State 12	State 13	State 14	State 15	State 16	State 17	State 18	State 19	State 20
t_1	1228	1117	1024	945	877	819	768	722	682	646
$\mathcal{O} \to 0$	State 12	State 13	State 14	State 15	State 16	State 17	State 18	State 19	State 20	$s_1 = 20$
$\mathcal{O} \to 1$	$s_1 = 10$	$s_1 = 11$	$s_1 = 12$	$s_1 = 13$	$s_1 = 14$	$s_1 = 15$	$s_1 = 16$	$s_1 = 17$	$s_1 = 18$	$s_1 = 19$

Step 1: In this step, $\mathcal{A}$ wants to get $s_1 = |\mathbf{s}_A[i]| + |\mathbf{s}_A[i+256]| + |\mathbf{s}_A[i+512]| + \mathbf{s}_A[i+768]|$.

$\mathcal{A}$ selects $\mathbf{s}_B$,$\mathbf{e}_B$, $\mathbf{e}'_B$ to be $\mathbf{0}$ except letting $\mathbf{e}_B[512] = t_1$. With the calculated $\mathbf{P}_B = \mathbf{a} \circ \mathbf{s}_B + \mathbf{e}_B = \mathbf{e}_B$, $\mathcal{A}$ goes on computing $\mathbf{k}$, $\mathbf{c}$ and $\bar{\mathbf{c}}$. To launch the attack, $\mathcal{A}$ needs to select appropriate t_1. We show the selections of t_1 and the States in Table 5. It can be seen that $\mathcal{A}$ keeps on selecting different t_1 and goes to next State according to the output of the Oracle. This process is repeated until he can determine s_1 based on the current t_1.

Next, we explain how to choose t_1 by using the idea of optimal BRT in details. When the Oracle receives $(\mathbf{P}_B, \bar{\mathbf{c}}, S_{k_B})$ from $\mathcal{A}$, it calculates

$$\begin{aligned}\mathbf{k}' =&\mathbf{c}' - \mathbf{P}_B \cdot \mathbf{s}_A = \mathbf{c}' - (\mathbf{a} \cdot \mathbf{s}_B + \mathbf{e}_B) \cdot \mathbf{s}_A = \mathbf{c}' - \mathbf{e}_B \cdot \mathbf{s}_A \\ =&[6145 - (-\mathbf{s}_A[i+512] \cdot t_1)] \cdot x^i + [6145 - (-\mathbf{s}_A[i+768] \cdot t_1)] \cdot x^{i+256} \\ &+ [6145 - \mathbf{s}_A[i] \cdot t_1] \cdot x^{i+512} + [6145 - \mathbf{s}_A[i+256] \cdot t_1] \cdot x^{i+768}.\end{aligned} \tag{7}$$

First, according to the Decode function in Algorithm 1, we have

$$\begin{aligned}s =&|6145 - (-\mathbf{s}_A[i+512] \cdot t_1) - 6144| + |6145 - (-\mathbf{s}_A[i+768] \cdot t_1) - 6144| \\ &+ |6145 - \mathbf{s}_A[i] \cdot t_1 - 6144| + |6145 - \mathbf{s}_A[i+256] \cdot t_1 - 6144| \\ =&|1 + \mathbf{s}_A[i+512] \cdot t_1| + |1 + \mathbf{s}_A[i+768] \cdot t_1| + |1 - \mathbf{s}_A[i] \cdot t_1| \\ &+ |1 - \mathbf{s}_A[i+256] \cdot t_1| \\ =&(|\mathbf{s}_A[i]| + |\mathbf{s}_A[i+256]| + |\mathbf{s}_A[i+512]| + |\mathbf{s}_A[i+768]|) \cdot t_1 + D.\end{aligned} \tag{8}$$

Here D is an integer which satisfies that $D = 0, \ \pm 2, \ \pm 4$. In [43], the authors let t_1 change from 0 to q, at the beginning $s < q$ and the Oracle outputs 1. As t_1 becomes larger, when $s \geq q$, the Oracle outputs 0 and the adversary records the current t_1. The recorded t_1 is then used to calculate $s_1 = \left\lfloor \frac{q}{t_1} \right\rceil$. But in this way, thousands of queries are needed to determine s_1. Therefore, we use our BRT method to solve this problem and greatly reduce the number of queries.

Since $s_1 = |\mathbf{s}_A[i]| + |\mathbf{s}_A[i+256]| + |\mathbf{s}_A[i+512]| + |\mathbf{s}_A[i+768]|$, $s_1 \in [0, 32]$. To simplify the problem we only consider $s_1 \in [0, 20]$, omitting those s_1 with a occurrence probability less than 0.000001. We observe that doing so has little impact on the E(#Queries). To find $s_1 = \left\lfloor \frac{q}{t_1} \right\rceil$, we divide the interval $[0, 20]$ into two subintervals $[0, s_1]$ and $[s_1 + 1, 20]$. If we can set t_1 in the interval $\left(\left\lfloor \frac{q}{s_1 + \frac{1}{2}} \right\rfloor, \left\lceil \frac{q}{s_1 - \frac{1}{2}} \right\rceil \right]$, then all the elements in $[0, s_1]$ correspond to $s > q$ and the Oracle outputs 0. At the same time, the elements in $[s_1 + 1, 20]$ correspond to $s \leq q$ and the Oracle outputs 0. Thus we can distinguish $[0, s_1]$ and $[s_1 + 1, 20]$ based on the outputs of the Oracle.

Table 6: $\mathbf{S}_i$ and its corresponding $l_{\bar{s}}$ & $l'_{\bar{s}}$ on Newhope

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
$\mathbf{S}_i$	6	5	7	4	8	9	3	10	2	11	12	1	13	14	0	15	16	17	18	19	20
$l_{\bar{s}}$	3	3	3	3	3	3	4	4	5	5	5	6	7	8	9	10	11	12	13	14	15
$l'_{\bar{s}}$	3	3	3	3	3	3	4	4	5	5	6	6	7	8	6	9	10	11	12	13	14

Specifically, by letting $\mathbf{S}_i$ ($i = 0, \ 1, \ \ldots, \ n-1$) represent all the possible value of s_1 and $n = 21$, we first sort the occurrence probabilities of s_1 in the descending order. Since $s_1 = 6$ occurs with the highest probability, we judge whether s_1 belongs to $[0, 6]$ or $[7, 20]$. Here the corresponding interval of t_1 is

$\left(\left\lfloor \frac{q}{6+\frac{1}{2}} \right\rfloor, \left\lceil \frac{q}{6-\frac{1}{2}} \right\rceil\right] = (1890, 2235]$. Therefore we select $t_1 = 2047 \in (1890, 2235]$ in State 1. Similar to the method in Section 3.2, we can find an optimal BRT for s_1. We list all the results in Table 6 but omitting the $\bar{s}$. We also show the length $l_{\bar{s}}$ for each $\mathbf{S}_i$ in the table, and the theoretical $E(\mathbf{S})$ is 3.3773. In addition, we can get the length $l'_{\bar{s}}$ that recovers s_1 in the practical attack, as shown in in Table 6. The $E(\mathbf{S})$ corresponding to the practical attack is 3.3854, which is almost the same as the theoretical $E(\mathbf{S})$.

Step 2: In this step, $\mathcal{A}$ checks how many 0s exist in $\mathbf{S}_A[i + 256 * j]$ ($j = 0, 1, 2, 3$), and then he tries to obtain the possible absolute values of them.

Step 2.1 We first introduce how $\mathcal{A}$ selects the parameters and calculates s_{21}, s_{22} and s_{23}, these three results will be used in **Step 2.2** and **Step 2.3**.

Similar to **Step 1**, $\mathcal{A}$ selects $\mathbf{s}_B$, $\mathbf{e}_B$ and $\mathbf{e}'_B$ to be $\mathbf{0}$, except for two special positions: $\mathbf{e}_B[j_1] = t_{21}$ and $\mathbf{e}_B[j_2] = t_2$, where $j_1, j_2 \in \{0, 256, 512, 768\}$ and $j_1 \neq j_2$. And then, he calculates $\mathbf{P}_B$, $\mathbf{k}$, $\mathbf{c}$ and $\bar{\mathbf{c}}$ the same way as that in **Step 1**.

If he sets $\mathbf{e}_B[0] = t_{21}$ and $\mathbf{e}_B[256] = t_2$, the result of s in Equation (8) becomes

$$\begin{aligned} s =& |1 - (\mathbf{s}_A[i] \cdot t_{21} - \mathbf{s}_A[i + 768] \cdot t_2)| \\ &+ |1 - (\mathbf{s}_A[i + 256] \cdot t_{21} + \mathbf{s}_A[i] \cdot t_2)| \\ &+ |1 - (\mathbf{s}_A[i + 512] \cdot t_{21} + \mathbf{s}_A[i + 256] \cdot t_2)| \\ &+ |1 - (\mathbf{s}_A[i + 768] \cdot t_{21} + \mathbf{s}_A[i + 512] \cdot t_2)|. \end{aligned} \tag{9}$$

Here, if $\mathbf{s}_A[i] > 0, \mathbf{s}_A[i + 256] > 0, \mathbf{s}_A[i + 512] > 0, \mathbf{s}_A[i + 768] > 0$, and t_{21} is much great than t_2. Then,

$$\begin{aligned} s =& (\mathbf{s}_A[i] \cdot t_{21} - \mathbf{s}_A[i + 768] \cdot t_2 - 1) + (\mathbf{s}_A[i + 256] \cdot t_{21} + \mathbf{s}_A[i] \cdot t_2 - 1) \\ &+ (\mathbf{s}_A[i + 512] \cdot t_{21} + \mathbf{s}_A[i + 256] \cdot t_2 - 1) \\ &+ (\mathbf{s}_A[i + 768] \cdot t_{21} + \mathbf{s}_A[i + 512] \cdot t_2 - 1) \\ =& (\mathbf{s}_A[i] + \mathbf{s}_A[i + 256] + \mathbf{s}_A[i + 512] + \mathbf{s}_A[i + 768])(t_{21} + t_2) \\ &- 2 \cdot t_2 \cdot \mathbf{s}_A[i + 768] - 4 \\ =& s_1 \cdot (t_{21} + t_2) - 2 \cdot t_2 \cdot \mathbf{s}_A[i + 768] - 4. \end{aligned} \tag{10}$$

By selecting appropriate t_{21} and t_2, $\mathcal{A}$ can get

$$\mathbf{s}_A[i + 768] = \frac{s_1 \cdot (t_{21} + t_2) - 4 - q}{2 \cdot t_2}. \tag{11}$$

For other cases, i.e. at least one of $\mathbf{s}_A[i + 256 * j] = 0$ or $\mathbf{s}_A[i + 256 * j] < 0$ ($j = 0, 1, 2, 3$), we can analyze them in the same way.

In Equations (9) and (10), there exists a constant -4. We find that the constant can only be selected from $\{\pm 4, \pm 2, 0\}$. In order to eliminate these constants, we set t_2 to be 100.

Different positions of j_1 and j_2 in $\mathbf{e}_B$ correspond to different results. But, if the positions of j_1 and j_2 in $\mathbf{e}_B$ are fixed, only the sign of $\mathbf{s}_A[i + 256 * j]$ ($j = 0, 1, 2, 3$) is different, then we can get similar result in Equation (11). There are a total of 12 cases, among which in Step 2, we only need three of them.

(1) $\mathcal{A}$ sets $\mathbf{e}_B[0] = t_{21}$ and $\mathbf{e}_B[512] = t_2 = 100$. By selecting proper t_{21}, together with the Oracle's output, he can have

$$s_{21} = \left\lfloor \frac{s_1 * (t_{21} + 100) - q}{200} \right\rceil .$$

(2) $\mathcal{A}$ sets $\mathbf{e}_B[0] = t_{22}$ and $\mathbf{e}_B[768] = t_2 = 100$, which results in

$$s_{22} = \left\lfloor \frac{s_1 * (t_{22} + 100) - q}{200} \right\rceil .$$

(3) Similarly, $\mathcal{A}$ sets $\mathbf{e}_B[256] = t_{23}$ and $\mathbf{e}_B[0] = t_2 = 100$, which gives

$$s_{23} = \left\lfloor \frac{s_1 * (t_{23} + 100) - q}{200} \right\rceil .$$

Here, s_{21} is always the sum of two different terms in $\mathbf{s}_A[i + 256 * j]$ ($j = 0, 1, 2, 3$), while s_{22} and s_{23} are the sum of one or three different terms.

Step 2.2: $\mathcal{A}$ checks whether there is a 0 in the $\mathbf{S}_A[i+256*j]$ ($j = 0, 256, 512, 768$) based on s_{21}, s_{22} and s_{23}.

- If $s_1 = 0$, then $\mathbf{S}_A[i + 256 * j] = 0$ ($j = 0, 1, 2, 3$).
- If $s_1 \neq 0$, then
  - If $s_{21} \neq 0$,
    - If $s_{22} + s_{23} = s_1$, then there is no 0 in the four $\mathbf{S}_A[i + 256 * j]$.
    - When $s_{22} + s_{23} \neq s_1$, if $s_{22} = s_{23} = 0$, there are two 0s in the tuple, else there is one 0 in the tuple.
  - If $s_{21} = 0$,
    - If $s_{22} = s_{23} = 0$, there are three 0s in the tuple, else there are two 0s in the tuple.

Step 2.3: $\mathcal{A}$ determines the four possible absolute values of $\mathbf{S}_A[i + 256 * j]$ ($j = 0, 1, 2, 3$).

- If there are three 0s in the tuple, we can directly know the non-zero value.
- If there are two 0s in the tuple, from the above analysis there are two cases, and we assume that the two non-zero values are x_1 and x_2.
  - When $s_1 \neq 0$, $s_{21} \neq 0$ and $s_{22} = s_{23} = 0$, we observer that $s_{21} = |x_1| + 0$ or $s_{21} = |x_2| + 0$. So, one of the non-zero value is equal to s_{21}, and the other non-zero value can be calculated as $s_1 - s_{21}$.
  - When $s_1 \neq 0$, $s_{21} = 0$ and either $s_{22} \neq 0$ or $s_{23} \neq 0$, from our observation the possible values of (s_{22}, s_{23}) could be $(x_1, 0)$, $(0, x_1)$, $(x_2, 0)$ and $(0, x_2)$. If $s_{22} = 0$, then $x_1 = s_{23}$. Subtracting s_{23} from s_1 we get x_2.
- If there is one 0 in the tuple, we assume that the three non-zero values are x_1, x_2 and x_3. According to our observation, we list the possible values of s_{21}, s_{22} and s_{23} in Table 7.
  - If $s_{23} = 0$, then $x_1 = s_{21}$, $x_2 = s_{22} - s_{21}$ and $x_3 = s_1 - x_1 - x_2$.
  - If $s_{23} \neq 0$, then $x_1 = s_{22}$, $x_2 = s_{23}$ and $x_3 = s_1 - x_1 - x_2$.

Table 7: The possible values of s_{21}, s_{22} and s_{23}

s_{21}	x_3+0	x_3+0	x_2+0	x_1+0
s_{22}	x_2+x_3+0	x_2	x_1+x_2+0	x_2
s_{23}	0	x_1	0	x_3

Table 8: The possible values of s_{21}, s_{22}, s_{23} and s_{24}

s_{21}	x_3+x_4	x_2+x_3	x_1+x_4	x_1+x_2
s_{22}	$x_2+x_3+x_4$	$x_1+x_2+x_3$	$x_1+x_3+x_4$	$x_1+x_2+x_4$
s_{23}	x_1	x_4	x_2	x_3
s_{24}	x_4	x_3	x_1	x_2
s_{21}	x_2+x_3	x_3+x_4	x_1+x_2	x_1+x_4
s_{22}	x_4	x_1	x_3	x_2
s_{23}	$x_1+x_2+x_3$	$x_2+x_3+x_4$	$x_1+x_2+x_4$	$x_1+x_3+x_4$
s_{24}	$x_1+x_2+x_4$	$x_1+x_2+x_3$	$x_1+x_3+x_4$	$x_2+x_3+x_4$

– If there is no 0 in the tuple, we assume that the four non-zero values are x_1, x_2, x_3 and x_4. In this case, we need to select $\mathbf{e}_B[0]=t_{24}$ and $\mathbf{e}_B[256]=100$, and get $s_{24}=\left\lfloor \frac{s_1*(t_{24}+100)-q}{200} \right\rceil$. According to our observation, we list the possible values of s_{21}, s_{22}, s_{23} and s_{24} in Table 8.
  - If $s_{22}>s_{24}$, then $x_1=s_{23}$, $x_2=s_{24}$, $x_3=s_{21}-s_{24}$ and $x_4=s_1-x_1-x_2-x_3$.
  - If $s_{22}\leq s_{24}$, then $x_1=s_{22}$, $x_2=s_{23}-s_{21}$, $x_3=s_{24}-x_1-x_2$ and $x_4=s_1-x_1-x_2-x_3$.

At the end of **Step 2**, $\mathcal{A}$ knows the four possible absolute values of the tuple.

Step 3. In this step, $\mathcal{A}$ wants to determine the exact value and the sign of each $\mathbf{S}_A[i+256*j]$ $(j=0,1,2,3)$. For example, $\mathcal{A}$ tries to recover $\mathbf{S}_A[i]$. First, he selects $\mathbf{P}_B=\left\lfloor \frac{q}{8} \right\rfloor x^{-i}$ and $\bar{\mathbf{c}}=\sum_{j=0}^{3}((l_j+4) \pmod 8)x^{256j}$ $(l_j=-4,-3,\cdots,3)$.

Then the Oracle calculates $v_A=\text{Decode}(\mathbf{K}')=\text{Decode}(\text{Decompress}(\bar{\mathbf{c}}-\mathbf{P}_B\mathbf{S}_A))$. According to the Decode function in Algorithm 1, we have

$$s=\sum_{j=0}^{3}\left|\text{Decompress}(\bar{\mathbf{c}}-\mathbf{P}_B\mathbf{S}_A)[i+256j]-\left\lfloor\frac{q}{2}\right\rfloor\right|. \tag{12}$$

If $s<q$ then $v_A[i]=1$, and the Oracle outputs 1, otherwise it outputs 0.

We let

$$f(\mathbf{S}_A[i])=\left|\left\lfloor(l_0+4)*\frac{q}{8}\right\rceil-\left\lfloor\frac{q}{16}\right\rfloor*\mathbf{S}_A[i]-\left\lfloor\frac{q}{2}\right\rfloor\right|+v, \tag{13}$$

where

$$v=\sum_{j=1}^{3}\left|\left\lfloor(l_j+4)*\frac{q}{8}\right\rceil-\left\lfloor\frac{q}{16}\right\rfloor*\mathbf{S}_A[i+256j]-\left\lfloor\frac{q}{2}\right\rfloor\right|. \tag{14}$$

Then we set $l_0 = -4$, now Equation (13) becomes

$$f(\mathbf{S}_A[i]) = \left| - \left\lfloor \frac{q}{16} \right\rfloor * \mathbf{S}_A[i] - \left\lfloor \frac{q}{2} \right\rfloor \right| + v. \tag{15}$$

In the following, we select the proper v to help decide $\mathbf{S}_A[i]$, here we also use our aforementioned BRT method. We let $f'(\mathbf{S}_A[i]) = - \left| \left\lfloor \frac{q}{16} \right\rfloor * \mathbf{S}_A[i] - \left\lfloor \frac{q}{2} \right\rfloor \right|$ and when $\mathbf{S}_A[i]$ changes from -8 to 8 the results of $f'(i)$ are shown in Table 9. Here, we also list $q - f'(\mathbf{S}_A[i])$ in this table, so that we can select v more conveniently. Specifically, if we want to know $\mathbf{S}_A[i]$ belongs to [-8,-1] or [0,8], we can select $v \in [q - f'(-1), q - f'(0)) = \left[\left\lfloor \frac{9q}{16} \right\rfloor + 2, \left\lfloor \frac{5q}{8} \right\rfloor + 1 \right)$. In this way, if $\mathbf{S}_A[i] \in [-8, -1]$, $f < q$ and the Oracle outputs 1, otherwise $f > q$ and the Oracle outputs 0.

Table 9: The value of f' and $q - f'$ when $\mathbf{S}_A[i] \in [-8, 8]$

$\mathbf{S}_A[i]$	-8	-7	-6	-5	-4	-3	-2	-1	0
f'	1	$\lfloor \frac{q}{16} \rfloor$ -1	$\lfloor \frac{q}{8} \rfloor$ -1	$\lfloor \frac{3q}{16} \rfloor$ -1	$\lfloor \frac{q}{4} \rfloor$ -1	$\lfloor \frac{5q}{16} \rfloor$ -1	$\lfloor \frac{3q}{8} \rfloor$ -1	$\lfloor \frac{7q}{16} \rfloor$ -1	$\lfloor \frac{q}{2} \rfloor$
$q - f'$	q-1	$\lfloor \frac{15q}{16} \rfloor + 2$	$\lfloor \frac{7q}{8} \rfloor + 2$	$\lfloor \frac{13q}{16} \rfloor + 2$	$\lfloor \frac{3q}{4} \rfloor + 2$	$\lfloor \frac{11q}{16} \rfloor + 2$	$\lfloor \frac{5q}{8} \rfloor + 2$	$\lfloor \frac{9q}{16} \rfloor + 2$	$\lfloor \frac{q}{2} \rfloor + 1$
$\mathbf{S}_A[i]$	1	2	3	4	5	6	7	8	
f'	$\lfloor \frac{9q}{16} \rfloor$	$\lfloor \frac{5q}{8} \rfloor$	$\lfloor \frac{11q}{16} \rfloor$	$\lfloor \frac{3q}{4} \rfloor$	$\lfloor \frac{13q}{16} \rfloor$	$\lfloor \frac{7q}{8} \rfloor$	$\lfloor \frac{15q}{16} \rfloor$	q-1	
$q - f'$	$\lfloor \frac{7q}{16} \rfloor + 1$	$\lfloor \frac{3q}{8} \rfloor + 1$	$\lfloor \frac{5q}{16} \rfloor + 1$	$\lfloor \frac{q}{4} \rfloor + 1$	$\lfloor \frac{3q}{16} \rfloor + 1$	$\lfloor \frac{q}{8} \rfloor + 1$	$\lfloor \frac{q}{16} \rfloor + 1$	1	

For example, if we want to decide whether $\mathbf{S}_A[i]$ is 0 or ± 3, first we must find $l_j (j = 1, 2, 3)$ such that $v = \lfloor \frac{q}{2} \rfloor$ in Equation (14). Since

$$\begin{aligned} f(-3) &= \left\lfloor \frac{5q}{16} \right\rfloor - 1 + \left\lfloor \frac{q}{2} \right\rfloor < q, \\ f(0) &= \left\lfloor \frac{q}{2} \right\rfloor + \left\lfloor \frac{q}{2} \right\rfloor < q, \\ f(3) &= \left\lfloor \frac{11q}{16} \right\rfloor + \left\lfloor \frac{q}{2} \right\rfloor > q, \end{aligned} \tag{16}$$

the Oracle outputs 1 when $f(\mathbf{S}_A[i]) < q$, we know that $\mathbf{S}_A[i] = 0$ or -3. If $f(\mathbf{S}_A[i]) > q$, the Oracle outputs 0, and we know that $\mathbf{S}_A[i] = 3$. Second, we must find $l_j (j = 1, 2, 3)$ such that $v = \lfloor \frac{11q}{16} \rfloor$ in Equation (14). Now

$$\begin{aligned} f(-3) &= \left\lfloor \frac{5q}{16} \right\rfloor - 1 + \left\lfloor \frac{11q}{16} \right\rfloor < q, \\ f(0) &= \left\lfloor \frac{q}{2} \right\rfloor + \left\lfloor \frac{11q}{16} \right\rfloor > q. \end{aligned} \tag{17}$$

In this turn, if Oracle outputs 1, we can determine $\mathbf{S}_A[i] = -3$. Otherwise, $\mathbf{S}_A[i] = 0$.

If $\mathcal{A}$ wants to determine $\mathbf{S}_A[i+256]$, he selects $\mathbf{P}_B = \left\lfloor \frac{q}{8} \right\rfloor x^{-i}$ and $\bar{\mathbf{c}} = \sum_{j=0}^{3}((l_j+4) mod\ 8)x^{256j}(l_j = -4, -3, \cdots, 3)$. Then he sets $l_1 = -4$, and according Table 9 to calculate an appropriate v in Equation (14) as

$$v = \sum_{j=0\&j\neq 1}^{3} \left| \left\lfloor (l_j+4) * \frac{q}{8} \right\rceil - \left\lfloor \frac{q}{16} \right\rfloor * \mathbf{S}_A[i+256j] - \left\lfloor \frac{q}{2} \right\rfloor \right|. \tag{18}$$

He can determine $\mathbf{S}_A[i+256]$ based on Oracle's output. Next, he recovers $\mathbf{S}_A[i+512]$ and $\mathbf{S}_A[i+768]$ in the same way.

Finally, the adversary repeats **Steps 1-3**, until he can determine all the coefficients in $\mathbf{s}_A$.

Further analysis and more details In this subsection, we analyze the results of our improved practical key mismatch attack on Newhope1024. First, we show how to use the BRT method to optimize the attack.

With an optimal BRT, we know the ideal way to recover the secret key with the smallest E(#Queries), that is, to find appropriate parameters to make the Oracle output 0 or 1, so that $\mathbf{S}_i$ belongs to the ideal subtree. The application of our BRT method on Kyber is relatively simple. For Newhope we have to recover 4 coefficients at the same time, but we can still apply the BRT method to our attack.

In **Step 1**, we need to select a proper t_1 to obtain the accurate s_1. First, we calculate the occurrence probability of each s_1, and then sort these probabilities in the descending order. For example, $s_1 = 6$ occurs with the highest probability. In the optimal BRT, $s_1 = 4, 5, 6, 7, 8, 9$ corresponds to the minimum number of queries 3. By setting the appropriate parameter t_1, we divide [0,20] into two subintervals based on the first output of the Oracle. From our calculations, the E(#Queries) obtained by dividing [0,20] into [0,6] and [7,20] is smaller than other divisions. The following operations are performed in a similar way.

In **Step 2**, we also need an appropriate t_{2i} to calculate s_{2i} corresponding to s_1. The process is similar to Section 3.2, and we let $n = s_1$. We must emphasize that, different s_1 results in different optimal BRT, and the corresponding E(#Queries) is also different.

Table 10: choices of t_1 and the States

	State 1	State 2	State 3	State 4	State 5	State 6
t_1	2100	2135	2065	2035	2000	1965
$\mathcal{O} \to 0$	State 3	$s_2 = 5$	State 4	State 5	State 6	$s_2 = 0$
$\mathcal{O} \to 1$	State 2	$s_2 = 6$	$s_2 = 4$	$s_2 = 3$	$s_2 = 2$	$s_2 = 1$

Table 11: $\mathbf{S}_i$ and its corresponding $\bar{s}$, $l_{\bar{s}}$ when $s_1 = 6$

i	0	1	2	3	4	5	6
$\mathbf{S}_i$	4	5	3	6	2	1	0
$\bar{s}$	01	10	001	11	0001	00001	00000
$l_{\bar{s}}$	2	2	3	2	4	5	5

For example, if $s_1 = 6$, then $\mathbf{S}_i$ is restricted to be selected from $\{0, 1, 2, 3, 4, 5, 6\}$. Here $n = 7$, and the resulted number of queries is 2.110. Table 10 depicts how to choose t_1 and which State to go upon the output of the Oracle. In Table 11, we show the $\mathbf{S}_i$ and its corresponding $\bar{s}$ and $l_{\bar{s}}$. Similar to the method in Section 3.2, we can successfully get s_{2i} using messages in this Table.

For each $s_1 \in [0, 20]$, we use the similar method in Section 3.2 to get the corresponding $\bar{s}$, $l_{\bar{s}}$ and the needed number of queries. Similar to the case $s_1 = 6$, each $s_1 \in [0, 20]$ is related to two tables. To simplify the calculation of s_{2i}, we only keep those with an occurrence probability greater than 0.000001. The remaining s_{2i} is between 0 and 17. Here we consider the worst case, and n is always set as 18. The corresponding number of needed queries is 3.197. Through our experiments, the probability that all elements in the tuple are not 0 is 0.417. Therefore, the average number of queries in this step can be calculated as $(0.417 * 4 + 0.583 * 3) * 3.197 = 10.923$.

In **Step 3**, we mainly want to determine the exact value of each $\mathbf{s}_A[i+256*j]$ $(j = 0, 1, 2)$. The challenge is that the possible absolute values may not be continuous, but we can still use the BRT method to solve this problem. For example, if there is no 0 in the tuple, $\mathbf{s}_A[i]$ may be one of $\{-x_4, -x_3, -x_2, -x_1, x_1, x_2, x_3, x_4\}$. Here $|x_4| < |x_3| < |x_2| < |x_1|$, and $\Pr(\pm x_4) < \Pr(\pm x_3) < \Pr(\pm x_2) < \Pr(\pm x_1)$. We first consider weather $\mathbf{s}_A[i] = \pm x_1$ or not. And we only need to find a v between $[q - f'(-x_1), q - f'(x_1))$ according to Table 9. If the Oracle's output is 0, we know it belongs to $\{x_1, x_2, x_3, x_4\}$. By repeating the above operations, we can finally determine $\mathbf{s}_A[i]$ with the smallest number of queries.

We still consider the worst case in this step. When there is no 0 in the tuple, if we want to determine x_1, x_2, x_3 and x_4, according to our BRT method, x_i needs $5 - i$ queries, for each $i = 1,\ 2,\ 3,\ 4$. Similarly, in this step 9 queries are required.

In a word, for each coefficient the needed number of queries is 23.308. Recall that we recover a quadruplet at one time, and there are 1024 unknown coefficients in a secret key $\mathbf{S}_A$. Therefore, in total we need 5966.848 queries to completely recover $\mathbf{S}_A$.

The key mismatch attack on Newhope1024 are also given in [43] and [41]. The parameters selection in [43] and [41] are similar, and they can only recover one coefficient in the secret key $\mathbf{s}_A$ at a time, while we are able to recover four coefficients together. This is one of the reasons why our proposed key mismatch attack performs much better. In addition, in [43] to recover an exact coefficient, the authors have to run 50 favorable cases, which costs more than 800 queries. The work of [41] has proposed an improved method to reduce the needed queries. However, more than 200 queries are still needed. In our proposed key mismatch attack, by applying the BRT method to each step, we only need 3-4 queries on average to obtain the desired results. This is another reason why we need much fewer queries.

Similarly, we can also improve the key mismatch attack on Frodo, Round5, Saber and Threebears. 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.

5 Comparisons

In Table 12, we compare the minimum average number of queries in our improved practical attacks (Bold) with other existing attacks (Italic). We can see that our improved attacks on Kyber is slightly better than that in [44], since for Kyber the gap between the theory and practice is limited. For Frodo640 and LightSaber, significant improvements have been made, compared to the results in [33]. It is worth noting that we greatly reduce the needed number of queries in key mismatch attacks against Newhope1024, from the reported 233, 803 queries in [41] to 5967.

Table 12: Comparisons of our improved key mismatch (Bold) attacks and the existing attacks (Italic), where "-" means no result is given.

Schemes	E(#Queries)		
	Existing attacks	This work	Lower bounds
Newhope512	-	**1707**	1568
Newhope1024	*233,803* [41]	**5967**	3103
Kyber512	*1401* [44]	**1184**	1088
Kyber768	*1855* [44]	**1776**	1632
Kyber1024	*2475* [44]	**2369**	2176
LightSaber	*2048* [33]	**1460**	1412
Saber	-	**2091**	1986
FireSaber	-	**2624**	2432
Frodo640	*65536* [33]	**18,360**	18,227
Frodo976	-	**26,079**	25,796
Frodo1344	-	**29,374**	27,973
LAC128	*1024* [28]	**640**	553
LAC192	*2048* [28]	**1280**	1106
LAC256	*8192* [28]	**1408**	1398
Round5 R5ND_1	-	**1184**	722
Round5 R5ND_3	-	**1380**	1170
Round5 R5ND_5	-	**1822**	1446

It can be seen that our improved attacks approach the bound in most cases. There is still a relatively big gap for Newhope1024 although we recover four coefficients in the secret key $\mathbf{s}_A$ at a time, and the BRT method is applied to each step. The reason is that in practice we need to decide the coefficients in order, but these coefficients in the optimal BRT jumps from one to another.

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 is 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.

6 Conclusion

In this paper, we have developed a unified method to calculate the minimum number of required requires 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 in each step, our improved attack can significantly reduce the needed number of queries. Especially, we can reduce the needed queries against Newhope1024 by 97.44%. An interesting problem is whether our proposed method applies to the similar attacks against other post-quantum cryptosystem such as HQC, which also advance to the third round.

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. 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.
15. 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)
16. 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)
17. 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)
18. Cover, T.M.: Elements of information theory. John Wiley & Sons (1999)
19. 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)
20. 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.
21. Diffie, W., Hellman, M.: New directions in cryptography. IEEE transactions on Information Theory **22**(6), 644–654 (1976)
22. 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)
23. 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)
24. 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.
25. Fluhrer, S.R.: Cryptanalysis of ring-lwe based key exchange with key share reuse. IACR Cryptology ePrint Archive (2016), http://eprint.iacr.org/2016/085.
26. Fujisaki, E., Okamoto, T.: Secure integration of asymmetric and symmetric encryption schemes. In: Annual International Cryptology Conference. pp. 537–554. Springer (1999)
27. 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.
28. Greuet, A., Montoya, S., Renault, G.: Attack on lac key exchange in misuse situation. Cryptology ePrint Archive, Report 2020/063 (2020)
29. Gyongyosi, L., Imre, S.: A survey on quantum computing technology. Computer Science Review **31**, 51–71 (2019)
30. 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.
31. 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)
32. Huffman, D.A.: A method for the construction of minimum-redundancy codes. Proceedings of the IRE **40**(9), 1098–1101 (1952)
33. 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.
34. Kirkwood, D., Lackey, B.C., McVey, J., Motley, M., Solinas, J.A., Tuller, D.: Failure is not an option: standardization issues for post-quantum key agreement. In: Workshop on Cybersecurity in a Post-Quantum World (2015)
35. Knuth, D.E.: The art of computer programming, vol. 3. Pearson Education (1997)
36. 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)
37. 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.
38. 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)

39. 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.
40. 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
41. Okada, S., Wang, Y., Takagi, T.: Improving key mismatch attack on newhope with fewer queries. IACR Cryptol. ePrint Arch. **2020**, 585 (2020)
42. Peikert, C.: Lattice cryptography for the internet. In: International workshop on post-quantum cryptography. pp. 197–219 (2014)
43. 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)
44. 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.
45. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. Journal of the ACM **56**(6), 1–40 (2009)

A Bounds for other candidates

Table 13: Key mismatch attacks against Three Bears, NTRU, and NTRU Prime. 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 minimum average number of queries in our proposed bounds.

Schemes	$\mathbf{s}_A$ & $\mathbf{e}$ Ranges	Encode Decode	Comp Decomp	Unknowns	E(#Queries) Bounds
BabyBear	[-1,1]			320	520
MamaBear	[-2,2]	/	✓	320	680
PapaBear	[-3,3]			320	738
ntruhps2048509	[-1,1]	/	/	509	849
ntruhps2048677				677	1129
ntruhps4096821				821	1369
ntruhrss701				701	1183
sntrup653	[-1,1]	/	/	653	1126
sntrup761				761	1397
sntrup857				857	1574
ntrulpr653				653	1162
ntrulpr761				761	1379
ntrulpr857				857	1553

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 Saber, Frodo, LAC and Round5.

B.1 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 14.

Table 14: 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 14 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 14) 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 14 and repeat operations in steps 2 and 3 until we can know which of $\{\mathbf{S}_1, \mathbf{S}_3, \mathbf{S}_5\}$ is $\mathbf{s}_A[0]$.

B.2 Improved key mismatch attacks on Frodo

There are three versions of Frodo, 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 15.

Next, we introduce how to use h_i in Table 15 to recover $\mathbf{s}_A[0]$.

Table 15: Selection of h_i in practical key mismatch attacks on Frodo

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	$2^{12}-6$	$2^{12}-7$	$2^{12}-8$	$2^{12}-9$	$2^{12}-10$	$2^{12}-11$

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 15. 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 15 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 $\mathbf{s}_A[0]$.

B.3 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.

According to the above process, we can calculate the E(#Queries) in LAC128 and LAC256, respectively.

B.4 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 $\mathcal{O} \to 1$: $\mathbf{s}_A[0] = \mathbf{S}_1$.

All the above results are summarized in Table 12.