An Efficient Key Mismatch Attack on NewHope

Yue Qin, Chi Cheng, *Member, IEEE*, and Jintai Ding

***Abstract*—The ring learning with errors based key exchange schemes like NewHope have attracted significant attention since they provide good alternatives for the widely used Diffie-Hellman key exchange in the quantum age. In CT-RSA 2019, Bauer et al. have analysed the case when the public key is reused in NewHope, and proposed a simple and elegant method, which is claimed to recover elements of the secret key ranges from -6 to 4. However, their recovery is incomplete in two aspects. First,through our experiments a significant part of secret elements ranges from -6 to 4 cannot be recovered. Second, for the elements not ranging from -6 to 4, they suggested that we can brute-force them. But for each secret key there are 10 elements cannot be recovered, and each of them has 6 possibilities, which means searching them in this way is not efficient. In this paper, we first improve Bauer et al.'s method to recover 99.2% of the elements ranging from -6 to 4. Then, inspired by Ding et al. 's key mismatch attack,we propose an efficient method which succeeds in recovering all the secret key elements ranging from -8 to 8 with a probability of 96.88%. To show the correctness and efficiency of our proposed method, we have also implemented our proposed key mismatch attack.**

***Index Terms*—Ring-LWE, quantum-safe, key reuse attacks, security analysis;**

I. Introduction

To solve the challenges quantum computers may bring to the current cryptosystem, the security community have proposed the so called post-quantum cryptography, which is a fast-growing research area and being in the standardization process by many standard bodies. Currently, the standardization process of post-quantum cryptography algorithms run by the NIST has completed the first round and the second round is scheduled to be held on August, 2019 [1]. As one of the most promising candidates for future post-quantum cryptography standard, the lattice-based cryptography, especially the ring learning with errors (Ring-LWE) based approaches, have attracted a lot of attention due to the provable security and high efficiency [2–4]. Among them, the Ring-LWE based key exchange schemes play a vital role due to the fact that they provide good alternatives for the Diffie-Hellman (DH) key exchange in the quantum age [5, 6].

To construct DH-like key exchange schemes whose hardness are based on the Ring-LWE problem, generally there are two methods. One is to use the error reconciliation mechanism, which means that one of the party needs to send additional information to help the other party to agree on an exactly same key. So far as we know the first paper proposing this idea was attributed to Ding, Xie, and Lin [7], and then an authenticated key exchange variant was proposed by Zhang et al. [8]. Peikert proposed a key encapsulation mechanism (KEM) using a similar error correction mechanism in [9], which is then reformulated by Bos et al. as a key exchange scheme and inserted into the Transport Layer Security (TLS) protocol [10].

Another Ring-LWE based key exchange scheme, the NewHope-Usenix [11], also attracts significant attention since Google has tested it in its browser Chrome to get real-world experiences about the deployment of the post-quantum cryptography. But the error reconciliation mechanism in the original NewHope-Usenix was so complex that later Alkim et al. proposed a simplified variant called the NewHope-simple [12], where the authors use the encryption-based approach to transfer the keys. In the submission to the competition of NIST's post-quantum cryptography, the submitted NewHope [13] was based on NewHope-simple, therefore in this paper we only consider the NewHope scheme with the encryption-based approach.

Note that in the widely used Internet standards, the key reuse mode is commonly used. For example, in the recently released standard TLS 1.3 [14], there exists a pre-shared key (PSK) mode in which the key can be reused. But the key reuse in lattice-based key exchange could cause much trouble. Generally, the key reuse attacks can be further divided into signal leakage attack and the key mismatch attack. The main cause of the signal leakage attack is that if the key is reused, the corresponding signal information used for exact key recovery reveals information about the secret key. On the other side, the key mismatch attack tries to recover the secret by querying a number of times whether the secret keys generated by the two parties are match or not. Recently, a series of key reuse attacks on the reconciliation based approaches have been proposed. Fluhrer first proposed the idea to exploit the leakage of secret keys of Ring-LWE based key exchange when one participant's public key is reused [15]. Later, Ding et al. has developed a key leakage attack on [7], where the reused keys leak information about the secret key [16]. In [17], a key mismatch attack was proposed on the one pass case of [7], without using the information leaked by the signal function. In [17], the attacker can determine the value and the sign of the private key by observing whether the final shared key between two parties matches. Specifically, the characteristics of the Mod_2 function in Ding et al.'s key exchange protocol can help judge the sign of the private key in a simple way. To thwart the proposed key leakage attack in case the public key is required to be reuse, in [18], a randomized method has been proposed. Another related work is [19], in which Liu et al. proposed a signal leakage attack against the reconciliation-based NewHope-Usenix key exchange protocol [11].

Unlike Ding's [7], Peikert's [9], and the NewHope-Usenix key exchange protocols, the NewHope key exchange submitted to the NIST [12] is based on the RLWE encryption rather than

Yue Qin and Chi Cheng are with School of Computer Science, China University of Geosciences, Wuhan, 430074, China. E-mail: chengchi@cug.edu.cn.

Jintai Ding is with the University of Cincinnati.

the reconciliation mechanism, and newly designed Encode and Compress functions are used. Therefore, these attacks proposed by Fluhrer [15], Ding et al. [16, 17], or Liu et al. [19] cannot be directly applied to the encryption-based NewHope key exchange protocol [13]. So a natural question arises, can we still use the key reuse attacks against the encryption-based approaches?

In this case, the main challenge for launching a key mismatch attack is that the Encode and Decode functions used in NewHope encodes fours positions together, which makes it much harder to recover the secret key as we cannot determine the value and the sign of the private key as that in [17]. Just recently, Bauer et al. have proposed a key mismatch attack on NewHope [20]. As we know, the coefficient of a secret key in NewHope belongs to the interval $\{-8,-7,\ldots,-1,0,1,\ldots,7,8\}$ due to the fact that they are selected from the centered binomial distribution ψ_8^n. The key observation of [20] is that in a 1024-bit long secret key, nearly 99% of the elements are in $\{-6,-5,\ldots,2,3,4\}$. From this observation, they have proposed a simple and elegant method, which is claimed to recover all the elements in $\{-6,-5,\ldots,2,3,4\}$.

However, their recovery is incomplete in two aspects. One is that through our experiments a significant part of secret elements in $\{-6,-5,\ldots,2,3,4\}$ cannot be recovered. The other is for the remaining 1% elements not belonging to $\{-6,-5,\ldots,2,3,4\}$, i.e. nearly 10 elements that are selected from $\{-8,-7,5,6,7,8\}$, they suggested brute-forcing them. As we can see, there are nearly $6^{10} \approx 6\times 10^7$ possibilities in this case, which is far from optimal. To solve these problems,first we improve Bauer et al.'s method to recover 99.2% of the elements in $\{-6,-5,\ldots,2,3,4\}$. Then, we proposed an efficient key mismatch attack against the NewHope key exchange protocol when the public key is reused, which can recover the elements that are selected from $\{-8,-7,5,6,7,8\}$ in an efficient way. Through in-depth analysis of the properties of the Decode function, we can notice that it can help us get some relationship between the four positions of the private key. Since in a targeted quadruplet, with a high probability there is only one element belongs to $\{-8,-7,5,6,7,8\}$, and the other 3 elements belong to $\{-6,-5,\ldots,2,3,4\}$. Therefore, we can first recover the three elements using the method in [20], then get the value of the remaining element. Finally, we have implemented the proposed attack against the NewHope key exchange, and the results show that other proposed method is rather efficient.

II. The Ring-LWE Problem and NewHope Key Exchange

Set $\mathbb{Z}_q$ the ring with all elements are integers modulo q, then $\mathbb{Z}_q[x]$ represents a polynomial ring, where all the polynomials in $\mathbb{Z}_q[x]$ are with coefficients selected from $\mathbb{Z}_q$. Then, we can define the polynomial ring $\mathcal{R}_q = \mathbb{Z}_q[x]/(x^n+1)$, in which for every polynomial $f(x) = a_0 + a_1x + \cdots + a_{n-1}x^{n-1} \in \mathbb{R}_q$, each coefficient $a_i \in \mathbb{Z}_q$ $(0 \le i \le n-1)$ and the polynomial additions and multiplications are operated modulo x^n+1. All polynomials are in bold, and we treat a polynomial $\mathbf{c} \in \mathcal{R}_q$ the same with its vector form $(\mathbf{c}[0],\cdots,\mathbf{c}[n-1])$, here $\mathbf{c}[i]$ $(0 \le i \le n-1)$ represents the ith coefficient of the polynomial $\mathbf{c}$. The operation $\lfloor x \rfloor$ represents the maximum integer not exceeding x, and $\lfloor x \rceil = \lfloor x + \frac{1}{2} \rfloor$.

The schemes based on Ring-LWE enjoy certain advantages due to the fact that there exists a quantum reduction which solves a hard problem in ideal lattices in the worst-case to solving a Ring-LWE problem in the average-case, as well as high efficiency even in resource-limited devices. Similar to the DH problems, there exist two versions of the Ring-LWE problem. The decision Ring-LWE is to distinguish the pair ($\mathbf{a}$, $\mathbf{as}+\mathbf{e}$) from randomly selected pair ($\mathbf{x}$, $\mathbf{y}$), where $\mathbf{a}$ is randomly sampled from $\mathcal{R}_q$ and $\mathbf{s}$, $\mathbf{e}$ are randomly selected according to a error distribution. Similarly, the search Ring-LWE is to recover $\mathbf{s}$ with the the above pair ($\mathbf{a}$, $\mathbf{as}+\mathbf{e}$).

Since in the submission to the competition of NIST's post-quantum cryptography, the submitted NewHope KEM was based on NewHope-simple, in the remaining of this paper we refer to the encryption based approach when we use NewHope. In NewHope, the polynomial ring $\mathcal{R}_q = \mathbb{Z}_q[x]/(x^n+1)$ is set with $q = 12289$ and $n = 1024$ or $n = 512$. The selected error distribution in NewHope is ψ_8^n, which is a centered binomial distribution with parameter 8, and can be easily sampled from computing $\sum_{i=1}^{8}(b_i - b_i')$. Here b_i and b_i' is randomly selected from $\{0,1\}$. The most important functions in the Newhope key exchange protocol are defined as follows.

Definition 1: The Encode function can map each bit in $\nu_{\mathbf{B}}' \in \{0,1\}^{256}$ to four bits in $\mathbf{k}$, which is for $i = 0,1,\ldots,255$,

$$\mathbf{k}[i] = \mathbf{k}[i+256] = \mathbf{k}[i+512] = \mathbf{k}[i+768] = \lfloor \frac{q}{2} \rfloor \nu_B'[i]. \quad (1)$$

Definition 2: The Decode function is the inverse of the Encode function, which can recover one bit of $\nu_A' \in \{0,1\}^{256}$ from four bits in $\mathbf{k}'$, i.e., $\nu_A' = \text{Decode}(\mathbf{k}')$ and

$$\nu_A'[i] = \begin{cases} 1 & \text{if } m < q, \\ 0 & \text{otherwise}, \end{cases} \quad (2)$$

where $m = \sum_{j=0}^{3} |\mathbf{k}'[i+256j] - \lfloor \frac{q}{2} \rfloor|$ for $i = 0,1,\ldots,255$.

Definition 3: The Compression function Compress: $\mathbb{Z}_q \to \mathbb{Z}_8$ is defined as $\bar{\mathbf{c}} = \text{Compress}(\mathbf{c})$ and for $i = 0, 1, \ldots, 1023$,

$$\bar{\mathbf{c}}[i] = \lfloor (\mathbf{c}[i]\cdot 8)/q \rceil \pmod 8. \quad (3)$$

Definition 4: The Decompression function Decompress: $\mathbb{Z}_8 \to \mathbb{Z}_q$ is the inverse of the Compression function $c' = \text{Decompress}(\bar{\mathbf{c}})$, which is for $i = 0,1,\ldots,1023$,

$$\mathbf{c}'[i] = \lfloor (\bar{\mathbf{c}}[i]\cdot q)/8 \rceil. \quad (4)$$

In Table I, we describe the details of the NewHope key exchange. Since in NewHope, the number-theoretic transform (NTT) is used to speedup the polynomial multiplication, which has nothing to do with security. To simplify the security analysis of NewHope, in Table I we use ordinary multiplication instead of NTT. To share a same key, the two participants Alice and Bob should share a common $\mathbf{a}$ in advance, which is randomly selected from $\mathcal{R}_q$. The NewHope key exchange protocol consists of three parts:

TABLE I
THE NEWHOPE KEY EXCHANGE

Common parameter:	$\mathbf{a} \leftarrow \mathcal{R}_q$	
Alice		**Bob**
$\mathbf{s}_A, \mathbf{e}_A \xleftarrow{\$} \psi_8^n$		
$\mathbf{P}_A \leftarrow \mathbf{a}\mathbf{s}_A + \mathbf{e}_A$	$\xrightarrow{\mathbf{P}_A}$	$\mathbf{s}_B, \mathbf{e}_B, \mathbf{e}'_B \xleftarrow{\$} \psi_8^n$
		$\mathbf{P}_B \leftarrow \mathbf{a}\mathbf{s}_B + \mathbf{e}_B$
		$\nu_B \xleftarrow{\$} \{0,1\}^{256}$
		$\nu'_B \leftarrow$ SHA3-256(ν_B)
		$\mathbf{k} \leftarrow$Encode(ν'_B)
		$\mathbf{c} \leftarrow \mathbf{P}_A\mathbf{s}_B + \mathbf{e}'_B + \mathbf{k}$
$\mathbf{c}' \leftarrow$ Decompress$(\bar{\mathbf{c}})$	$\xleftarrow{(\mathbf{P}_B, \bar{\mathbf{c}})}$	$\bar{\mathbf{c}} \leftarrow$ Compress$(\mathbf{c})$
$\mathbf{k}' = \mathbf{c}' - \mathbf{P}_B\mathbf{s}_A$		$S_{k_B} \leftarrow$ SHA3-256(ν'_B)
$\nu'_A \leftarrow$ Decode$(\mathbf{k}')$		
$S_{k_A} \leftarrow$ SHA3-256(ν'_A)		

(1) Alice selects $\mathbf{s}_A$ and $\mathbf{e}_A$ uniformly at random from ψ_8^n, and computes a public key $\mathbf{P}_A = \mathbf{a}\mathbf{s}_A + \mathbf{e}_A$. Then Alice will send $\mathbf{P}_A$ to Bob.

(2) After receiving $\mathbf{P}_A$ sent by Alice, Bob will select $\mathbf{s}_B$, $\mathbf{e}_B$ and $\mathbf{e}'_B$ uniformly at random from ψ_8^n, and compute a public key $\mathbf{P}_B = \mathbf{a}\mathbf{s}_B + \mathbf{e}_B$. Then Bob will choose ν_B randomly from $\{0,1\}^{256}$ and compute $\nu'_B \leftarrow$ SHA3-256(ν_B), $\mathbf{k} \leftarrow$Encode(ν'_B), $\mathbf{c} \leftarrow \mathbf{P}_A\mathbf{s}_B + \mathbf{e}'_B + \mathbf{k}$ and $\bar{\mathbf{c}} \leftarrow$ Compress$(\mathbf{c})$. Subsequently, Bob will send $\mathbf{P}_B$ and $\bar{\mathbf{c}}$ to Alice, and compute the shared key $S_{k_B} \leftarrow$ SHA3-256(ν'_B).

(3) When Alice receives the $\mathbf{P}_B$ and $\bar{\mathbf{c}}$ sent by Bob, she will calculate $\mathbf{c}' \leftarrow$ Decompress$(\bar{\mathbf{c}})$, $\mathbf{k}' = \mathbf{c}' - \mathbf{P}_B\mathbf{s}_A$, $\nu'_A \leftarrow$ Decode$(\mathbf{k}')$ and her shared key $S_{k_A} \leftarrow$ SHA3-256(ν'_A).

III. THE PROPOSED KEY MISMATCH ATTACK

In this section, we will use the key mismatch method to assess the security of the NewHope key exchange protocol when the public key is reused.

Algorithm 1: Oracle

```
Input: $\mathbf{P}_B, \bar{\mathbf{c}}, S_{k_B}$
Output: 1 or 0
$\mathbf{c}'$ = Decompress($\bar{\mathbf{c}}$);
$\mathbf{k}' = \mathbf{c}' - \mathbf{P}_B\mathbf{s}_A$ ;
$\nu'_A \leftarrow$ Decode($\mathbf{k}'$);
$S_{k_A} \leftarrow$ SHA3-256($\nu'_A$);
if $S_{k_A} = S_{k_B}$ then
    Return 1;
else
    Return 0;
end
```

In a key mismatch attack, the adversary $\mathcal{A}$ is an active adversary who plays the role of Bob, and we build an oracle $\mathcal{O}$ that simulates Alice in Table I. We assume that Alice's public key $\mathbf{P}_A$ is reused and $\mathcal{A}$ can query the oracle a number of times. In Algorithm 1 we describe how the oracle works. To be specific, $\mathcal{A}$ calculates $\mathbf{P}_B$, as well as $\bar{\mathbf{c}}$ and S_{k_B} generated by using a selected ν'_B. By receiving the input $(\mathbf{P}_B, \bar{\mathbf{c}}, S_{k_B})$, the oracle will use $\mathbf{P}_B$ and $\bar{\mathbf{c}}$ to calculate $\mathbf{c}'$, $\mathbf{k}'$, ν'_A, S_{k_A} and checks whether $S_{K_A} = S_{K_B}$ holds, if yes the oracle $\mathcal{O}$ will output 1 and 0 otherwise. Specifically, if $\mathcal{O}$ outputs 1, S_{k_A} and S_{k_B} match and $\nu'_A = \nu'_B$. If $\mathcal{O}$ outputs 0, S_{k_A} and S_{k_B} mismatch and $\nu'_A \neq \nu'_B$. We can see that the adversary can get useful information from the oracle by knowing whether the two keys S_{k_A} and S_{k_B} match or not, and further recover $\mathbf{s}_A$ using these information.

The main challenge in launching a key mismatch attack against the NewHope key exchange is that, 4 elements of $\mathbf{s}_A$, for example $\mathbf{s}_A[i]$, $\mathbf{s}_A[i+256]$, $\mathbf{s}_A[i+512]$, and $\mathbf{s}_A[i+768]$ are mixed, which makes it harder to decide each of them.

A. Bauer et al.'s method

In this subsection, we briefly introduce the Bauer et al.'s method in [20]. They used the key mismatch attack to recover Alice's private key $\mathbf{s}_A$ if Alice's public key $\mathbf{P}_A$ is reused. But they can only recover the private key in $S_2 = \{-6, -5, \ldots, 2, 3, 4\}$. First of all, the adversary $\mathcal{A}$ directly chooses $\nu'_B = (1, 0, \cdots, 0)$. If $\mathcal{A}$ wants to recover the quadruplet $(\mathbf{s}_A[k], \mathbf{s}_A[k+256], \mathbf{s}_A[k+512], \mathbf{s}_A[k+768])$, he will set his public key $\mathbf{P}_B = \lfloor \frac{q}{8} \rfloor x^{-k}$ and $\bar{\mathbf{c}} = \sum_{i=0}^{3} ((l_i + 4) \bmod 8)x^{256i}$, here each l_i increases from -4 to 3. Then he will send $(\mathbf{P}_B, \bar{\mathbf{c}}, S_{k_B})$ to the oracle $\mathcal{O}$. When $\mathcal{O}$ receives $(\mathbf{P}_B, \bar{\mathbf{c}}, S_{k_B})$, he will honestly calculate $\mathbf{c}', \mathbf{k}', \nu'_A$ and S_{k_A}. If $S_{k_A} = S_{k_B}$ he will return 1 and 0 otherwise. Finally, $\mathcal{A}$ will calculate the private key according to $\mathcal{O}$'s output. Since each quadruplet (l_0, l_1, l_2, l_3) corresponds to an output of $\mathcal{O}$, the adversary $\mathcal{A}$ can recover the elements of the private key if he can find outputs in a form like $1, 1, \cdots, 1, 0, \cdots, 0, 1, \cdots, 1$ as (l_0, l_1, l_2, l_3) changes. Here this kind of form is called a favourable case.

Specifically, if $\mathcal{A}$ wants to recover $\mathbf{s}_A[k+256i]$ in $\mathbf{s}_A$, he can first set each l_i $(i = 1, 2, 3)$ be randomly selected from -4 to 3, and then by letting $l_0 = -4$, the resulted output is a bit b_0. Next $\mathcal{A}$ can increase l_0 to -3, with the same l_i $(i = 1, 2, 3)$ the resulted output is another bit b_1. Repeating the above processes until l_0 becomes 3, there will be 8 bits b_i $(i = 0, 1, \cdots, 7)$. The above processes will be repeated with different l_i $(i = 1, 2, 3)$ until $\mathcal{A}$ can assure that $(b_0, b_1, \ldots, b_7)$ is a favorable case. Then the adversary $\mathcal{A}$ can recover the elements in S_1. First, he will find two special positions τ_1 and τ_2, when i increases from 1 to 6, if $b_{i-1} = 1$ and $b_i = 0$, then $\mathcal{A}$ sets $\tau_1 = i - 4$, else if $b_i = 0$ and $b_{i+1} = 1$, then $\mathcal{A}$ sets $\tau_2 = i - 4$. Subsequently, $\mathcal{A}$ will calculate $\tau = \tau_1 + \tau_2$, if τ is even, we can determine that $\mathbf{s}_A[k+256i] = \tau$, or $\mathbf{s}_A[k+256i] = 2\lfloor \frac{\tau}{2} \rfloor + 1$. $\mathcal{A}$ will repeat above processes until he recovers all the elements of $\mathbf{s}_A$ that in a specific interval.

B. Problems with Bauer et al.'s method

Then we implement Bauer et al.'s method to recover the coefficients belonging to $\mathrm{S}_2 = \{-6, -5, \ldots, 2, 3, 4\}$. To make our experiment more convincing, we use the code the designers of NewHope submitting to the NIST [13] to generate 1000 secret keys. Unfortunately, using Bauer et al.'s method we cannot even recover all the coefficients belonging to S_2 in every secret key. In other ways, in every 1024-bit long key, there are at least 78 coefficients in S_2 cannot be recovered.

In the proof of Proposition 1 of [20], by letting $q = 8s + 1$ and $f_j = (l_j + 4) \bmod 8$ $(j = 0, 1, 2, 3)$ they have

$$\begin{aligned} Decompress(\bar{\mathbf{c}})[256j] &= \left\lceil \frac{f_j \times q}{8} \right\rfloor \\ &= \left\lceil f_j \times s + \frac{1}{8} \right\rfloor \\ &= f_j \times s. \end{aligned} \tag{5}$$

However, the second equation is inaccurate, instead,

$$\begin{aligned} Decompress(\bar{\mathbf{c}})[256j] &= \left\lceil \frac{f_j \times q}{8} \right\rfloor \\ &= \left\lceil \frac{f_j \times (8s+1)}{8} \right\rfloor \\ &= \left\lceil f_j \times s + \frac{f_j}{8} \right\rfloor. \end{aligned} \tag{6}$$

Since $f_j = (l_j + 4) \bmod 8$ $(j = 0, 1, 2, 3)$ and l_j ranges from -4 to 3, equation 5 only holds when l_j ranges from -4 to -1. When l_j increases from 0 to 3, we can see that in Table II $\left\lceil \frac{f_j \times q}{8} \right\rfloor$ is not equal to $\left\lceil f_j \times s + \frac{1}{8} \right\rfloor$ nor $f_j \times s$. We can also observe from Table II that when l_j ranges from 0 to 3, $\left\lceil f_j \times s + \frac{f_j}{8} \right\rfloor$ is equal to $\left\lceil \frac{f_j \times q}{8} \right\rfloor$. This means that equation 6 is consistent with the outputs of the Decompress function.

TABLE II
DECOMPRESS FUNCTION'S DERIVATION RESULTS

l_j	$\left\lceil \frac{f_j \times q}{8} \right\rfloor$	$\left\lceil f_j \times s + \frac{1}{8} \right\rfloor$	$f_j \times s$	$\left\lceil f_j \times s + \frac{f_j}{8} \right\rfloor$
0	$\lceil 6144.5 \rfloor$	$\lceil 6144.125 \rfloor$	6144	$\lceil 6144.5 \rfloor$
1	$\lceil 7608.625 \rfloor$	$\lceil 7608.125 \rfloor$	7608	$\lceil 7608.625 \rfloor$
2	$\lceil 9216.75 \rfloor$	$\lceil 9216.125 \rfloor$	9216	$\lceil 9216.75 \rfloor$
3	$\lceil 10752.875 \rfloor$	$\lceil 10752.125 \rfloor$	10752	$\lceil 10752.875 \rfloor$

In the Bauer et al.'s implementation they directly use the results of $f_j \times s$ as the output of Decompress function instead of using the actual output of Decompress function, i.e., $\frac{f_j \times q}{8}$. Specifically, when l_j increases form -4 to 3, the outputs of $f_j \times s$ are 0, 1536, 3072, 4608, 6144, 7608, 9216 and 10752. But the actual outputs of $\frac{f_j \times q}{8}$ should be 0, 1536, 3072, 4608, 6145, 7609, 9217 and 10753. Although the gap between them is tiny, the resulted impact is huge, i.e., the calculation of $(b_0, \ b_1, \ldots, b_7)$ is inaccurate in [20]. In the following, we will use an example to further analyze it.

We randomly choose a secret key s_A, among which a quadruplet is $(0, 4, 3, 3)$, and 0 is the coefficient of the private key to be recovered. When we have a favorable case, (l_1, l_2, l_3) is set as $(-4, 0, -1)$.

- In Bauer et al.'s implementation and Equation 5, $Decompress(\bar{\mathbf{c}})[256j] = f_j \times s$ and $\bar{\mathbf{c}} = \sum_{j=0}^{3} f_j \cdot x^{256j}$, then

$$\begin{aligned} Decompress(\bar{\mathbf{c}}) =& f_0 \times s + 0 \times s \cdot x^{256} + 4 \times s \cdot x^{512} \\ &+ 3 \times s \cdot x^{768} \\ =& f_0 \times s + 6144 \cdot x^{512} + 4608 \cdot x^{768}. \end{aligned}$$

Since $\mathbf{P}_B = sx^{-k}$, $\mathbf{c}' \leftarrow$ Decompress($\bar{\mathbf{c}}$), $\mathbf{k}' = \mathbf{c}' - \mathbf{P}_B \mathbf{s}_A$, for $i = 0, 1, \ldots, 255$, we have

$$\begin{aligned} m &= \sum_{j=0}^{3} |\mathbf{k}'[i + 256j] - \lfloor \frac{q}{2} \rfloor| \\ &= \sum_{j=0}^{3} |(\mathbf{c}' - \mathbf{P}_B \mathbf{s}_A)[i + 256j] - \lfloor \frac{q}{2} \rfloor| \\ &= \sum_{j=0}^{3} |(Decompress(\bar{\mathbf{c}}) - \mathbf{P}_B \mathbf{s}_A)[i + 256j] - \lfloor \frac{q}{2} \rfloor| \\ &= \sum_{j=0}^{3} |(Decompress(\bar{\mathbf{c}}) - \mathbf{s}_A s x^{-k})[i + 256j] - \lfloor \frac{q}{2} \rfloor|. \end{aligned}$$

So, when l_0 increases from -4 to 3, the value of m and b are shown in Table III.

- In our analysis and Equation 6, since $Decompress(\bar{\mathbf{c}})[256j] = \left\lceil f_j \times s + \frac{f_j}{8} \right\rfloor$, we have

$$\begin{aligned} Decompress(\bar{\mathbf{c}}) =& \left\lceil f_0 \times s + \frac{f_0}{8} \right\rfloor + 0 \cdot x^{256} + \left\lceil 4 \times s + \frac{4}{8} \right\rfloor \cdot x^{512} \\ &+ \left\lceil 3 \times s + \frac{3}{8} \right\rfloor \cdot x^{768} \\ =& \left\lceil f_0 \times s + \frac{f_0}{8} \right\rfloor + 0 + \lceil 6144 + 0.5 \rfloor \cdot x^{512} \\ &+ \lceil 4608 + 0.375 \rfloor \cdot x^{768} \\ =& \left\lceil f_0 \times s + \frac{f_0}{8} \right\rfloor + 6145 \cdot x^{512} + 4608 \cdot x^{768}. \end{aligned}$$

In this case, when l_0 increases from -4 to 3, we set m in Decode function to m' and set b to b' . m' and b' are shown in Table III.

TABLE III

l_0	-4	-3	-2	-1	0	1	2	3
m	15360	13825	12289	10753	9217	10753	12289	13825
b	0	0	0	1	1	1	0	0
m'	15360	13824	12288	10752	9217	10753	12289	13825
b'	0	0	1	1	1	1	0	0

When $l_0 = -2$, the value of m in Table III is 12289 and the value of b is 0. $\tau_1 = -1, \tau_2 = 1, \tau = 0$, so we can recover s as 0. But in Table III, m' is 12288 and b' is 1. $\tau_1 = -2, \tau_2 = 1, \tau = -1$, the value of s we can recover is -1. In this case, we recover an incorrect s. The main reason for the error is that the value of $Decompress(\bar{\mathbf{c}})$ is different. According to the paper [13], we should set $Decompress(\bar{\mathbf{c}})[256j] = \left\lceil f_j \times s + \frac{f_j}{8} \right\rfloor$ instead of $f_j \times s$.

2) In Bauer et al.s method, there is only one kind of favorable case like $1, 1, \ \cdots, 1, \ 0, \ \cdots, \ 0, 1, \ \cdots, \ 1$. But we find there is another favorable case like $0, \ 0, \cdots, 0, \ 1, \ \cdots, \ 1, 0, \ \cdots, \ 0$.

3) In Bauer et al.s method, they try to directly recover s according to τ is odd or even. For example, if we want to recover 0, τ should be even, but in Table III we can see that τ is odd. So, we can not rely on τ is odd or even to determine s. We ran an experiment in which we generated 1000 private keys s and used Bauer et al.'s method to recover the elements

of each s in S_2. But for each recovered private key s, all elements in S_2 cannot be fully recovered, and at least 262 elements in S_2 cannot be recovered.

C. Our Improved Method

Algorithm 2: Find-τ

```
Input: $b$
Output: $\tau$
set $\tau_1$ = NULL, $\tau_2$ = NULL, $s$ = NULL ;
if $b[0] = 1$ then
    for $i := 1$ to $6$ do
        if $(b[i-1] = 1)$ and $(b[i] = 0)$ then
            $\tau_1 = i - 4$;
        end
        if $(b[i] = 0)$ and $(b[i+1] = 1)$ then
            $\tau_2 = i - 4$;
        end
    end
else if $b[0] = 0$ then
    for $i := 1$ to $6$ do
        if $(b[i-1] = 0)$ and $(b[i] = 1)$ then
            $\tau_1 = i - 4$;
        end
        if $(b[i] = 1)$ and $(b[i+1] = 0)$ then
            $\tau_2 = i - 4$;
        end
    end
$\tau = \tau_1 + \tau_2$;
if $\tau$ is odd then
    odd++;
    odd_$\tau = \tau$;
    count ++;
else if $\tau$ is even then
    even++;
    even_$\tau = \tau$;
    count ++;
else
    countinue;
end
Return $\tau$
```

For the two kinds of favourable cases, we just need to improve the method of calculate τ_1 and τ_2. If the favorable case in a form like $1, 1, \cdots, 1, 0, \cdots, 0, 1, \cdots, 1$, we will use Bauer et al.s method to to calculate τ_1 and τ_2. But if the favorable case in a form like $0, 0, \cdots, 0, 1, \cdots, 1, 0, \cdots, 0$. When i increase from 1 to 6, if $b_{i-1} = 0$ and $b_i = 1$, then we set $\tau_1 = i - 4$, else if $b_i = 1$ and $b_{i+1} = 0$, then we set $\tau_2 = i - 4$. The specific process of calculating τ_1 and τ_2 are shown in lines 2 to 19 of Algorithm 2.

According to Table III, we can find that if the private key s is 0, there are not only odd τ, but also even τ. So, we need to find another relationship between s and τ.

We made a statistical experiment using a s containing all the elements in [-6, 4]. In this experiment, (l_0, l_1, l_2, l_3) are full

Algorithm 3: Recover

```
Output: $s$ (the elements in $S_2$)
for $k := 0$ to $255$ do
    Set $\mathbf{P}_B = \lfloor \frac{q}{8} \rfloor x^{-k}$;
    for $j := 0$ to $3$ do
        Set odd = 0, even = 0, count = 0;
        while count < 50 do
            $(l_0, l_1, l_2, l_3) \leftarrow [-4, 3]^4$;
            set b[8] = 0;
            for $i := -4$ to $3$ do
                $l_j = i$;
                $\bar{\mathbf{c}} = \sum_{h=0}^{3}((l_h + 4) \bmod 8) x^{256*h}$;
                $b[i] = Oracle(\mathbf{P}_B, \bar{\mathbf{c}}, S_{k_B})$;
            end
            $t$ =Find-$\tau(b)$;
        end
        if odd >= even then
            temp$_s$= $\lfloor(\text{odd\_}\tau - 8)/2\rfloor * 2 + 1$;
            test(temp$_s$);
        else if even > odd then
            temp$_s$ = even_$\tau$ - 8;
            test(temp$_s$);

    end
end
$s[k + j * 256]$ = temp$_s$;
Return $s$
```

permutation from -4 to 3, then we count how many favorable cases have occurred for each element in s, and how many of the odd and even τ are respectively. The statistical results are shown in Table IV. We can observe that if s is odd, there are two cases, one case is that all τ are odd, and in another case there are both odd τ and even τ, moreover the number of odd τ must be more than the number of even τ. The same rule exists when s is even. So, we can recover s based on how many times odd τ and even τ appear in all favorable cases. This means that, we need to record the number of times the favorable cases, and the number of times the odd τ and even τ appear. The specific process of count the favorable cases, odd τ and even τ are shown in lines 21 to 31 of Algorithm 2. In order to improve efficiency, we will not full permutation (l_0, l_1, l_2, l_3), instead we only need 50 favorable cases for statistics. If the number of odd τ is more than the number of even τ, we will use odd_τ to recover s, otherwise use even_τ to recover s. The specific process of recover s are shown in Algorithm 3.

When we recover a s, we must check this value. Because we only counted 50 favorable cases, it is possible that the statistical results are wrong. Especially when $s = 3$, its odd τ and even τ appear very close, so it is particularly prone to statistical errors. For example, we could recover 3 to 2. Next, we will give an example to introduce how to judge whether the recovered value is correct, and how to correct it if the recover value is wrong. If we recover 3 to 2, the number of even τ must be more than odd τ, temp$_s$ in line 19 of Algorithm 3 is equal 2. And its even_τ = 2, odd_τ = 3 and $b[0] = 0$.

TABLE IV
THE DISTRIBUTION OF THE ELEMENTS IN A QUADRUPLET

s	odd τ	even τ	favorable cases	s	odd τ	even τ	favorable cases
0	0	2080	2080	1	1472	0	1472
	400	1656	2056		1344	512	1856
2	0	2048	2048	-1	2176	0	2176
	504	1328	1832		1808	320	2128
-2	0	2080	2080	3	1408	0	1408
	240	1824	2064		848	808	1656
4	0	2048	2048	-3	1408	0	1408
	520	1264	1784		1312	232	1544
-4	0	1952	1952	-5	1408	0	1408
	152	1792	1944		1160	296	1456
-6	0	2048	2048				
	136	1784	1920				

But if we recover 3 correctly, the number of odd τ must be more than even τ, temp$_s$ = 3, even_τ = -8, odd_τ = -5 and $b[0] = 1$. If we recover 2 correctly, the number of even τ must be more than odd τ, temp$_s$ = 2, even_τ = 2, odd_τ = -8 and $b[0] = 0$. So, when we recover a temp$_s$ = 2, we will check if even_τ = 2 and odd_τ = -8, then the recovered value is correct. Otherwise, if even_τ = 2 and odd_τ = 3, then the recovered value is wrong, we directly let temp$_s$ = 3.

TABLE V
THE DISTRIBUTION OF THE ELEMENTS IN A QUADRUPLET

S$_1$= $\{-8,-7,\ldots,-1,0,1,\ldots,7,8\}$
S$_2$= $\{-6,-5,\ldots,2,3,4\}$ S$_1$-S$_2$= $\{-8,-7,5,6,7,8\}$

4 elements in S$_1$		
100%		
4 elements in S$_2$	Others	
96.8%	3.2%	
	3 elements in S$_2$ 1 element in S$_1$-S$_2$	2 elements in S$_2$ 2 elements in S$_1$-S$_2$
	99%	1%

D. Our Observation

We set S$_1$= $\{-8,-7,\ldots,-1,0,1,\ldots,7,8\}$ and S$_2$= $\{-6,-5,\ldots,2,3,4\}$. In table V, we have analysed and listed the distribution of the elements in a quadruplet through our experiments. We have generated 10^6 keys following the centered binomial distribution, and then taken an average. We can see that all the elements are in set S$_1$, and the probability that all the elements of the quadruplet are in S$_2$ is 96.8%. From our observation, with high probability there is only one element in a quadruplet that belongs to S$_1-$S$_2$, while the other 3 elements are in S$_2$. Specifically, as shown in Table V, in the remaining 3.2% of the quadruplets, the probability that only one element not belonging to S$_2$ is 99%. Without loss of generality, we assume that $\mathbf{s}_A[i]$, $\mathbf{s}_A[i+256]$ and $\mathbf{s}_A[i+512]$ are in S$_2$ and $\mathbf{s}_A[i+768]$ is in S$_1-$S$_2$. Using our improved method in Algorithms 2 and 3, we can recover $\mathbf{s}_A[i]$, $\mathbf{s}_A[i+256]$ and $\mathbf{s}_A[i+512]$. Then, our remaining task is to determine the exact value of $\mathbf{s}_A[i+768]$.

E. The Complete Attack

To launch the attack, the adversary $\mathcal{A}$ will deliberately select the parameters $\mathbf{s}_B$ and $\mathbf{e}_B$ to calculate the public key $\mathbf{P}_B$, as well as the parameter ν'_B to calculate $\bar{\mathbf{c}}$. For each integer i in $0,1,\cdots,255$, if $\mathcal{A}$ wants to recover $\mathbf{s}_A[i], \mathbf{s}_A[i+256], \mathbf{s}_A[i+512], \mathbf{s}_A[i+768]$, he will choose $\mathbf{s}_B$ and $\mathbf{e}'_B$ to be 0 in $\mathbb{R}_q$, and an $\mathbf{e}_B$ with the coefficient vector that consists of all zeros, except that $\mathbf{e}_B[256] = h_1$, here h_1 increases from 0 to $q-1$. Instead of randomly selecting ν_B, the adversary $\mathcal{A}$ will directly set all elements of ν'_B as 0 except that $\nu'_B[i] = 1$.

As $\mathcal{A}$ sets $\mathbf{s}_B = \mathbf{0}$, correspondingly the public key is

$$\mathbf{P}_B = \mathbf{a}\mathbf{s}_B + \mathbf{e}_B = \mathbf{e}_B.$$

Next, $\mathcal{A}$ sets $\nu'_B = \mathbf{0}$ except that $\nu'_B[i] = 1$, according to the definition of the Encode function,

$$\begin{aligned}\mathbf{k} =& \mathbf{Encode}(\nu'_B)\\ =& \lfloor\frac{q}{2}\rfloor x^i + \lfloor\frac{q}{2}\rfloor x^{i+256} + \lfloor\frac{q}{2}\rfloor x^{i+512} + \lfloor\frac{q}{2}\rfloor x^{i+768},\end{aligned}$$

and the resulted $\mathbf{c} = \mathbf{P}_A\mathbf{s}_B + \mathbf{e}'_B + \mathbf{k} = \mathbf{k}$.

Then, since $\bar{\mathbf{c}}[i] = \lfloor(\bar{\mathbf{c}}[i]\cdot q)/8\rceil = 4$, according to the above analysis and the definition of the Compress function

$$\begin{aligned}\bar{\mathbf{c}} &= \mathbf{Compress}(\mathbf{c})\\ &= \mathbf{Compress}(\mathbf{k})\\ &= 4x^i + 4x^{i+256} + 4x^{i+512} + 4x^{i+768}.\end{aligned}$$

After that $\mathcal{A}$ will send $(\mathbf{P}_B, \bar{\mathbf{c}}, S_{k_B})$ to $\mathcal{O}$, who will then calculate

$$\begin{aligned}\mathbf{c}' =& \mathit{Decompress}(\bar{\mathbf{c}})\\ =& \lfloor\frac{q}{2}\rceil x^i + \lfloor\frac{q}{2}\rceil x^{i+256} + \lfloor\frac{q}{2}\rceil x^{i+512} + \lfloor\frac{q}{2}\rceil x^{i+768},\end{aligned} \tag{7}$$

as well as

$$\mathbf{k}' = \mathbf{c}' - \mathbf{P}_B\mathbf{s}_A = \mathbf{c}' - \mathbf{e}_B\mathbf{s}_A \tag{8}$$

and $S_{k_A} = \mathrm{SHA3}-256(\mathrm{Decode}(\mathbf{k}'))$.

In the following, we propose our method to recover the exact value of $\mathbf{s}_A[i]$ in an efficient way.

The adversary $\mathcal{A}$ chooses the parameters as described above, and the attack includes four steps. In step 1, the adversary $\mathcal{A}$ will use algorithm 2 to recover all the elements belong to S_2. In step 2, $\mathcal{A}$ will calculate $m_1 = |\mathbf{s}_A[i]|+|\mathbf{s}_A[i+256]|+|\mathbf{s}_A[i+512]|+|\mathbf{s}_A[i+768]|$. In step 3, $\mathcal{A}$ can get the absolute value of $\mathbf{s}_A[i]$ with m_1 , since $\mathcal{A}$ has recovered $\mathbf{s}_A[i+256]$, $\mathbf{s}_A[i+512]$ and $\mathbf{s}_A[i+768]$ using the improved method in Algorithms 2 and 3. Next $\mathcal{A}$ tries to decide the sign of $\mathbf{s}_A[i+768]$. In step 4, $\mathcal{A}$ will verify whether the private key he recovered is correct or not.

Next, we will briefly introduce Step 2, Step 3 and Step 4.

Step 2: In this step, adversary $\mathcal{A}$ wants to decide $m_1 = |\mathbf{s}_A[i]|+|\mathbf{s}_A[i+256]|+|\mathbf{s}_A[i+512]|+|\mathbf{s}_A[i+768]|$. First, $\mathcal{A}$ sets all the elements of $\mathbf{e}_B$ as $\mathbf{0}$, except $\mathbf{e}_B[256] = h_1$. From equations 7, 8 and $\lfloor\frac{q}{2}\rceil = 6145$, we have

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

Algorithm 4: Find m_1

```
Input: $i$
Output: $m_1$
for $h_1 := 0$ to $q - 1$ do
    $\mathbf{e}_B = \mathbf{0}$, set $\mathbf{e}_B[256] = h_1$;
    $\mathbf{P}_B = \mathbf{e}_B$;
    $\nu'_B = \mathbf{0}$, set $\nu'_B[i] = 1$ ;
    $\mathbf{k} \leftarrow$Encode$(\nu'_B)$ ;
    $\bar{\mathbf{c}} =$ Compress$(\mathbf{k})$;
    $S_{k_B} \leftarrow$ SHA3-256$(\nu'_B)$;
    $v =$ Oracle$(\mathbf{P}_B, \bar{\mathbf{c}}, S_{k_B})$ ;
    if $v = 1$ then
        $m_1 = \lfloor (q+2)/h_1 \rceil$;
        break;
    else
        continue;
    end
end
Return $m_1$
```

The last equation holds since $x^{1024} = -1$ in R_q. So, for $i = 0, 1, \ldots, 255$, according to the Decode function we have

$$\begin{aligned}
m &= \sum_{j=0}^{3} |\mathbf{k}'[i + 256j] - 6145| \\
&= |1 - (-\mathbf{s}_A[i+768]h_1)| + |1 - \mathbf{s}_A[i]h_1| \\
&\quad + |1 - \mathbf{s}_A[i+256]h_1| + |1 - (-\mathbf{s}_A[i+512]h_1)| \\
&= \mathbf{s}_A[i+768]h_1 + 1 + \mathbf{s}_A[i]h_1 - 1 \\
&\quad + \mathbf{s}_A[i+256]h_1 - 1 + \mathbf{s}_A[i+512]h_1 + 1 \\
&= (\mathbf{s}_A[i] + \mathbf{s}_A[i+256] + \mathbf{s}_A[i+512] \\
&\quad + \mathbf{s}_A[i+768])h_1.
\end{aligned}$$

Then the adversary let h_1 change from 1 to q, at the beginning $m < q$, $Decode(\mathbf{k}'[i]) = 1$ and the oracle $\mathcal{O}$ will output 1. When h_1 becomes larger, m also becomes larger, and when $m > q$, the output of $\mathcal{O}$ becomes 0. By recording the value of h_1 when the output of $\mathcal{O}$ changes, we can know that here m roughly equals q, and $\mathcal{A}$ can calculate $m_1 = \lfloor \frac{q}{h_1} \rceil$ by setting $m = m_1 h_1 = q$.

It should be noted that with $m_1 = |\mathbf{s}_A[i+256]| + |\mathbf{s}_A[i+512]| + |\mathbf{s}_A[i+768]|$, if $\mathcal{A}$ can determine that $\mathbf{s}_A[i] = 0$, then $\mathcal{A}$ will skip Step 3.

The main processes of Step 2 is shown in Algorithm 4.

Step 3: In this step, the adversary $\mathcal{A}$ will determine the sign of $\mathbf{s}_A[i]$. When $\mathcal{A}$ completes Step 1, if the value of $\mathbf{s}_A[i]$ is not in $[-6, 4]$, then $\mathbf{s}_A[i]$ will be recovered to an incorrect value, but its sign is still correct. So, we can directly determine the sign of $\mathbf{s}_A[i]$ according this. However, there are two special cases where the correct sign of $\mathbf{s}_A[i]$ is opposite to the value recovered in Step 1 when $\mathbf{s}_A[i] = 8$ or $\mathbf{s}_A[i] = -8$.

Step 4: The adversary $\mathcal{A}$ verifies whether the private key he recovered is correct by calculating the distribution of $\mathbf{P}_A - \mathbf{a}\mathbf{s}_A$. Since $\mathbf{a}$ and $\mathbf{P}_A$ are public, if $\mathcal{A}$ gets the correct private key, then the distribution is the same as that of $\mathbf{e}_A$, which follows the centered binomial distribution.

TABLE VI
ONE INSTANCE OF THE KEY MISMATCH ATTACK

Common parameter:	$\mathbf{a} \overset{\$}{\leftarrow} \mathcal{R}_q$	
Oracle		**Adversary**
$\mathbf{s}_A, \mathbf{e}_A \overset{\$}{\leftarrow} \psi_8^n$ Reused public key: $\mathbf{P}_A \leftarrow \mathbf{a}\mathbf{s}_A + \mathbf{e}_A$		
	$\xrightarrow{\mathbf{P}_A}$	$\mathbf{s}_B = \mathbf{0}, \mathbf{e}'_B = \mathbf{0}$ $\mathbf{e}_B = \mathbf{0}$, set $\mathbf{e}_B[0] = h_1, \mathbf{e}_B[512] = h_2$ $\mathbf{P}_B \leftarrow \mathbf{e}_B$ $\nu'_B = \mathbf{0}$, set $\nu'_B[i] = 1$ $\mathbf{k} \leftarrow$Encode(ν'_B) $\mathbf{c} \leftarrow \mathbf{P}_A\mathbf{s}_B + \mathbf{e}'_B + \mathbf{k}$
$\mathbf{c}' \leftarrow$ Decompress$(\bar{\mathbf{c}})$ $\mathbf{k}' = \mathbf{c}' - \mathbf{P}_B\mathbf{s}_A$ $\nu'_A \leftarrow$ Decode$(\mathbf{k}')$	$\xleftarrow{(\mathbf{P}_B, \bar{\mathbf{c}}, S_{k_B})}$	$\bar{\mathbf{c}} \leftarrow$ Compress$(\mathbf{c})$ $S_{k_B} \leftarrow$ SHA3-256(ν'_B)
$S_{k_A} \leftarrow$ SHA3-256(ν'_A)	$\xrightarrow{\text{0 or 1}}$	

IV. Experiments

In this section, we report the performance our implementations. All our implementations are done on a MacBook Air, which has a Intel Core i7 processor at 2.7 GHz and an 8 GB RAM. All experiments are in C, and all multiplications are not optimized.

First of all, we want to show the advantage of our proposed algorithm 2 in recovering coefficients belonging to $S_2 = \{-6, -5, \ldots, 2, 3, 4\}$. To make our experiment more convincing, we use the code the designers of NewHope submitting to the NIST [13] to generate $10,000$ secret keys. When we use our method as shown in algorithm 2, in 9688 keys we can recover all the coefficients, and in the remaining 312 keys that cannot be recovered. The probability of successfully recovering s is 96.88%. At the same time, we also found that the number of coefficients belonging to S_1-S_2 is between 2 and 19.

In our proposed method, first we implement our proposed algorithms 2 and 3 to recover the coefficients belonging to S_2. Then we will use algorithm 4 to calculate $m_1 = |\mathbf{s}_A[i]| + |\mathbf{s}_A[i+256]| + |\mathbf{s}_A[i+512]| + |\mathbf{s}_A[i+768]|$, then we can known the absolute value of the element that belonging to S_1-S_2. For example, if we do not know $\mathbf{s}_A[i+768]$, we can determine the absolute value $|\mathbf{s}_A[i+768]| = s_1 - |\mathbf{s}_A[i]| - |\mathbf{s}_A[i+256]| - |\mathbf{s}_A[i+512]|$. Subsequently, we will follow the Steps 3 aforementioned to decide the sign of $\mathbf{s}_A[i+768]$.

In Table VII, we report the average time consuming and average queries

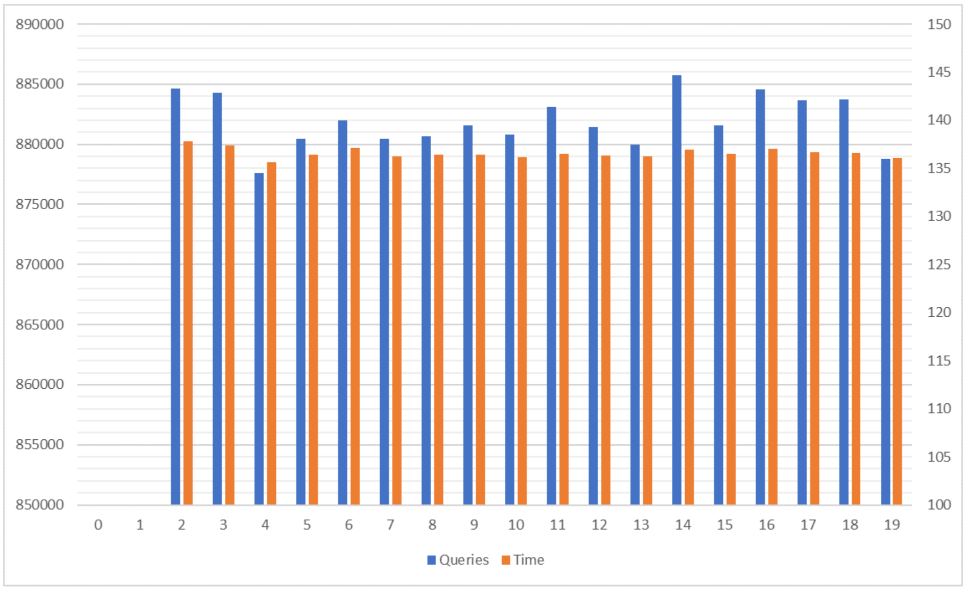
V. Conclusion

In this paper, we have analyzed the security of NewHope when the public key is reused. We developed Bauer et al.'s method and proposed a complete key mismatch attack on NewHope. Since these kinds of lattice-based key exchange schemes are widely believed to replace the DH key exchange in the quantum age, their resistance to misuse situations are of high importance. It is worth noting that the NewHope KEM submitted to NIST is CPA secure, which is then transformed into CCA-secure using Fujisaki-Okamoto transforma-

TABLE VII

	0	1	2	3	4	5	6	7	8	9
Queries	0	0	884624	884324	877610	880447	881960	880485	880701	881555
Time(ms)	0	0	137.77	137.37	135.65	136.42	137.19	136.231	136.41	136.45
	10	11	12	13	14	15	16	17	18	19
Queries	880801	883106	881461	879977	885730	881568	884556	883677	883706	878808
Time(ms)	136.12	136.52	136.35	136.26	136.97	136.53	137.01	136.67	136.59	136.12

Fig. 1.



tion. Therefore, the proposed key mismatch attack does not harm the NewHope designers' security goals. But our results show that when designers who base their approaches on the lattice-based key exchange should be careful to avoid the public key reuse, which is common in the design with DH key exchange approaches.

Acknowledgments

The work presented in this paper was supported in part by the National Natural Science Foundation of China under Grant no. 61672029.

References

[1] G. Alagic, J. Alperin-Sheriff, D. Apon, et al. "Status Report on the First Round of the NIST Post-Quantum Cryptography Standardization Process", *NIST Interagency/Internal Report (NISTIR)-8240*, 2019. https://nvlpubs.nist.gov/nistpubs/ir/2019/NIST.IR.8240.pdf, accessed 26 February 2019.

[2] O. Regev, "On lattices, learning with errors, random linear codes, and cryptography", *Journal of the ACM*, vol. 56, no. 6, pp. 34: 1- 34: 40, 2009.

[3] V. Lyubashevsky, C. Peikert and O. Regev, "On ideal lattices and learning with errors over rings", *Advances in Cryptology - EUROCRYPT 2010*, pp. 1-23, 2010.

[4] R. Lindner and C. Peikert, "Better key sizes (and attacks) for LWE-based encryption", *CT-RSA*, vol. 6558, pp. 319-339, 2011.

[5] L. Chen, S. Jordan, Y. Liu, D. Moody, R. Peralta, R. Perlner and D. Smith-Tone, *Report on post-quantum cryptography*, NISTIR 8105, available at http://nvlpubs.nist.gov/nistpubs/ir/2016/NIST.IR.8105.pdf, accessed 26 February 2019.

[6] C. Cheng, R. Lu, A. Petzoldt, and T. Takagi, "Securing the Internet of Things in a Quantum World", *IEEE Communications Magazine*, vol. 55, no. 2, pp. 116-120, 2017.

[7] J. Ding, X. Xie, and X. Lin, "A simple provably secure key exchange scheme based on the Learning with errors problem", *IACR Cryptology EPrint Archive Report 2012/688*, https://eprint.iacr.org/2012/688.pdf, accessed 26 February 2019.

[8] J. Zhang, Z. Zhang, J. Ding, M. Snook and ö. Dagdelen, "Authenticated key exchange from ideal lattices," In EUROCRYPT, pp. 719-751, April 2015.

[9] C. Peikert, "Lattice cryptography for the Internet", *in Proc. International Workshop on Post-Quantum Cryptography*, pp.197-219, 2014.

[10] J. W. Bos, C. Costello, M. Naehrig, and D. Stebila, "Post-quantum key exchange for the TLS protocol from the ring learning with errors problem", *IEEE Symposium on Security and Privacy,* pp. 553-570, 2015.

[11] E. Alkim, L. Ducas, T. Pöppelmann, and P. Schwabe, "Post-quantum key exchange-a new hope", *in Proc. USENIX Security Symposium*, pp. 327-343, 2016.

[12] E. Alkim, L. Ducas, T. Pöppelmann and P. Schwabe,

"NewHope without reconciliation", 2016. https://www.cryptojedi.org/papers/newhopesimple-20161217.pdf, accessed 17 February 2019.

[13] E. Alkim, R. Avanzi, J. Bos, L. Ducas, et al. "Newhope: Algorithm specification and supporting documentation", *Submission to the NIST Post-Quantum Cryptography Standardization Project*, 2018. https://newhopecrypto.org/data/NewHope_2018_12_02.pdf, accessed 27 February 2019.

[14] E. Rescorla, "The transport layer security (TLS) protocol version 1.3", *No. RFC 8446*. 2018. http://www.rfc-editor.org/info/rfc8446, accessed 26 February 2019.

[15] S. Fluhrer, "Cryptanalysis of ring-LWE based key exchange with key share reuse", *IACR Cryptology ePrint Archive Report 2016/085*, http://eprint.iacr.org/2016/085, accessed 18 February 2019.

[16] J. Ding, S. Alsayigh, R. Saraswathy, et al. "Leakage of signal function with reused keys in RLWE key exchange", *Communications (ICC), 2017 IEEE International Conference on*, pp.1-6, 2017.

[17] J. Ding, S. Fluhrer, and R. Saraswathy, "Complete attack on RLWE key exchange with reused keys, without signal leakage", *Australasian Conference on Information Security and Privacy*, pp. 467-486, 2018.

[18] X. Gao, J. Ding, L. Li, et al. "Practical randomized RLWE-based key exchange against signal leakage attack", *IEEE Transactions on Computers*, vol. 67, no. 11, pp. 1584-1593, 2018.

[19] C. Liu, Z. Zheng, and G. Zou, "Key Reuse Attack on NewHope Key Exchange Protocol", *International Conference on Information Security and Cryptology*, pp.163-176, 2018.

[20] A. Bauer, H. Gilbert1, G. Renault1, and M. Rossi, "Assessment of the Key-Reuse Resilience of NewHope," in the proceedings of CT-RSA 2019.