Improving Key Mismatch Attack on NewHope with Fewer Queries

Satoshi Okada[1], Yuntao Wang[2], and Tsuyoshi Takagi[1]

[1] Graduate School of Information Science and Technology, The University of Tokyo
okada-satoshi323@g.ecc.u-tokyo.ac.jp takagi@mist.i.u-tokyo.ac.jp
[2] School of Information Science, Japan Advanced Institute of Science and Technology
y-wang@jaist.ac.jp

Abstract. NewHope is a lattice cryptoscheme based on the Ring Learning With Errors (Ring-LWE) problem, and it has received much attention among the candidates of the NIST post-quantum cryptography standardization project. Recently, key mismatch attacks on NewHope have been proposed, where the adversary tries to recover the server's secret key by observing the mismatch of the shared key from chosen queries. At CT-RSA 2019, Bauer et al. first proposed a key mismatch attack on NewHope, and then at ESORICS 2019, Qin et al. proposed an improved version with success probability of 96.9% using about 880,000 queries. In this paper, we further improve their key mismatch attacks on NewHope. First, we reduce the number of queries by adapting the terminating condition to the response from the server using an early abort technique. Next, the success rate of recovering the secret key polynomial is raised by setting a deterministic condition for judging its coefficients. We also improve the method of generating queries. Furthermore, the search range of the secret key in Qin et al.'s attack is extended without increasing the number of queries. As a result, about 73% of queries can be reduced compared with Qin et al.'s method under the success rate of 97%. Moreover, we analyze the trade-off between the number of queries and the success rate. In particular, we show that a lower success rate of 20.9% is available by further reduced queries of 135,000, simultaneously.

Keywords: PQC, Ring-LWE, Key Mismatch Attack, NewHope.

1 Introduction

The current public-key cryptosystems based on the hardness of the factorization problem or the discrete logarithm problem can be broken by quantum computers in polynomial time [17]. For this reason, it is urgent to develop post-quantum cryptography (PQC) which is secure against the threat of quantum computers. PQC is being standardized by the National Institute of Standards and Technology (NIST) [1]. There, lattice-based cryptography is one of the most promising categories, and NewHope is one of the lattice-based key exchange candidates selected in the second round of the NIST PQC standardization project. The security of NewHope [2] is based on the difficulty of the underlying Ring-LWE problem [12]. Comparing to the typical LWE problem [15], the Ring-LWE based cryptoschemes enjoy smaller key sizes that benefit from its ring structure. On the other hand, some potential demerits from the ring structure may be maliciously used by attackers, thus more careful cryptanalysis of these cryptoschemes is required.

Nowadays it is common to reuse keys in Internet communications, so as to improve the performance of the protocols. For example, TLS 1.3 [16] adopts the pre-shared key (PSK) mode where the server is allowed to reuse the same secret key and public key in intermittent communication with the clients, in order to reduce the procedure of handshakes. Meanwhile, such protocols may have a risk of leakage of the server's secret key when the adversary has enough chances to send queries to the honest server and get correct responses from it. There is a kind of key mismatch attack on the Ring-LWE based key exchange protocols. As its name implies, the key mismatch attack generally works as follows: an adversary sends chosen ciphertexts to the server, and recovers the server's secret key by observing a match or mismatch of a common key. In particular, there are mainly two key mismatch attacks on NewHope [4,13] which take advantage of the property that the secret key of NewHope is a polynomial constructed with integer coefficients sampled from -8 to 8 in a key-reuse scenario.

The first key mismatch attack on NewHope was proposed by Bauer et al. [4] at CT-RSA 2019, which can recover the secret coefficients belonging to the interval $[-6, 4]$. However, the success rate of recovering coefficients in $[-6, 4]$ was not so high. Bauer et al. also reported that the coefficients belonging to $\{-8, -7, 5, 6, 7, 8\}$ can be recovered by the brute-force attack, nevertheless, the computational complexity is as large as $6^{11} \simeq 2^{39}$ due to the fact that about 11.16 coefficients of 1024 ones are belonging to $\{-8, -7, 5, 6, 7, 8\}$ on average in one secret key.

Furthermore, Qin et al. [13] improved Bauer et al.'s attack at ESORICS 2019 so that the coefficients in $[-6, 4]$ can be successfully recovered with a high rate of 99.22%, and the others in $\{-8, -7, 5, 6, 7, 8\}$ can be recovered with fewer queries than the brute-force attack. As a result, the rate of recovering the secret key correctly achieves 96.88%. However, the attack proposed by Qin et al. requires a large number of 880,000 queries for recovering a secret key, which makes the attack not efficient. Besides, some specific patterns of secret keys can not be recovered successfully in this attack.

1.1 Our Contributions

In this paper, we further improve Qin et al.'s attack to reduce the number of queries, and evaluate its relationship with the success rate of recovering secret keys. First, we introduce an early abort technique to reduce the number of queries. Namely, we set an appropriate query stop condition according to the response (i.e. match or mismatch with the common key) from the server. Then, to raise the success rate of the attack, we propose a deterministic condition when judging the secret polynomial's coefficients; and we improve the method of generating queries sent by the adversary. Moreover, we observe that without increasing the number of queries, the attack of Qin et al. on the secret key coefficients in $[-6, 4]$ can be extended to a wider range of $[-6, 7]$. Since only 0.28 coefficients on average belonging to the remaining set of $\{-8, -7, 8\}$ in one secret key, we decide to perform a brute-force attack. As a result, to achieve almost the same success rate of 97%, the number of queries is reduced to about 230,000 which is 73% less than the cost claimed in Qin et al's method. Furthermore, the recovery success rate can be improved to 100.0% experimentally in our method. Simultaneously, by evaluating the relationship between the success rate and the number of queries, we can further reduce the number of queries to 135,000 with 20.9% success rate.

1.2 Related Works

A number of key recovery attacks have been developed to Ring-LWE based cryptography, under the assumption of a key reusing scenario. Generally, they are divided into two types: the signal leakage attacks with exploiting the flaws of the signal function [5,8,11], and the key mismatch attacks taking advantage of constructing the final shared key. In this work, we focus on the latter key mismatch attacks, as we already introduced two previous works of [4,13] above. Besides, in ACISP 2018, Ding et al. [7] proposed a general key mismatch attack model for Ring-LWE based key exchange scheme without using the signal leakage. Recently, there are also some key mismatch attacks on several specific lattice-based cryptographic schemes. For instance, in 2020, Greuet et al. [10] proposed the mismatch attack on LAC which is a Ring-LWE based cryptoscheme but with small key size. In 2019, Qin et al. [14] applied their attack on the Module-LWE based Kyber as well. And Ding et al. [6] analyzed the NTRU cryptoscheme by adapting the key mismatch attack to it. Especially, the mismatch attack using the quantum algorithm was proposed by Băetu et al. [3] in Eurocrypt 2019.

1.3 Roadmap

We recall the NewHope cryptoscheme and its relevant functions in Section 2. Then we introduce the previous works of mismatch attacks on NewHope in Section 3, including the methods proposed by Bauer et al [4] and its improvement by Qin et al [13], respectively. In Section 4, we propose our mismatch attack which is evidently improving Qin et al.'s attack. We give our experimental results, and show the trade-off between the number of queries and the success rate in Section 5. Finally, we conclude our work in Section 6.

2 Preliminaries

In this section, we introduce the algebraic definitions and notations used in NewHope. Next, we show the outline of NewHope's protocol, including several important functions being used in it.

pre-shared key $\mathbf{a}$		
Alice		Bob
$\mathbf{s}_A, \mathbf{e}_A \overset{\$}{\leftarrow} \psi_8^n$		
$\mathbf{P}_A \leftarrow \mathbf{a}\mathbf{s}_A + \mathbf{e}_A$	$\xrightarrow{\mathbf{P}_A}$	$\mathbf{s}_B, \mathbf{e}_B, \mathbf{e}'_B \overset{\$}{\leftarrow} \psi_8^n$
		$\mathbf{P}_B \leftarrow \mathbf{a}\mathbf{s}_B + \mathbf{e}_B$
		$\nu_B \overset{\$}{\leftarrow} \{0,1\}^{256}$
		$\nu'_B \leftarrow$ SHA3-256 (ν_B)
		$\mathbf{k} \leftarrow \texttt{Encode}\,(\nu'_B)$
		$\mathbf{c} \leftarrow \mathbf{P}_A\mathbf{s}_B + \mathbf{e}'_B + \mathbf{k}$
$\mathbf{c}' \leftarrow \texttt{Decompress}(\overline{\mathbf{c}})$	$\xleftarrow{(\mathbf{P}_B, \overline{\mathbf{c}})}$	$\overline{\mathbf{c}} \leftarrow \texttt{Compress}(\mathbf{c})$
$\mathbf{k}' = \mathbf{c}' - \mathbf{P}_B\mathbf{s}_A$		$S_{k_B} \leftarrow$ SHA3-256 (ν'_B)
$\nu'_A \leftarrow \texttt{Decode}\,(\mathbf{k}')$		
$S_{k_A} \leftarrow$ SHA3-256 (ν'_A)		

Fig. 1. NewHope key exchange protocol

Set $\mathbb{Z}_q$ the integer remainder ring modulo q, and $\mathbb{Z}_q[x]$ represents a polynomial ring whose coefficients are sampled from $\mathbb{Z}_q$. We also denote the residue ring of $\mathbb{Z}_q[x]$ modulo $(x^n + 1)$ by $\mathcal{R}_q = \mathbb{Z}_q[x]/(x^n + 1)$. Bold letters such as $\mathbf{P}, \mathbf{s}$ refer to elements in $\mathcal{R}_q$. We also use vector notation for polynomials in this paper, e.g. the vector notation for $\mathbf{a} \left(= \sum_{i=0}^{n-1} a_i x^i\right) \in \mathcal{R}_q$ is $(a_0, a_1, \cdots, a_{n-2}, a_{n-1})$. $\mathbf{a}[i]$ represents the coefficient of x^i in the polynomial, and the corresponding i-th element of the vector as well. For a real number x, $\lfloor x \rfloor$ represents the largest integer no larger than x and $\lfloor x \rceil = \lfloor x + \frac{1}{2} \rfloor$. For the sake of convenience, we set $s = \lfloor q/8 \rfloor$ where q is the integer modulus in NewHope.

We denote by ψ_8 a binomial distribution with a standard deviation of 8, and its element is sampled by calculating $\sum_{i=1}^{8} (b_i - b'_i)$. Here, b_i and b'_i are sampled from $\{0, 1\}$ uniformly at random. Let ψ_8^n be the polynomial set whose each coefficient is sampled from ψ_8. In the figures and algorithms, the notation $\overset{\$}{\leftarrow} \mathcal{D}$ means randomly sampling an element from distribution (or set) $\mathcal{D}$.

Ring-LWE Problem: Let χ be a distribution on $\mathcal{R}_q$. For randomly sampled polynomials $\mathbf{s}, \mathbf{e} \overset{\$}{\leftarrow} \chi, \mathbf{a} \overset{\$}{\leftarrow} \mathcal{R}_q$, the set of $(\mathbf{a}, \mathbf{b} = \mathbf{a}\mathbf{s} + \mathbf{e} \in \mathcal{R}_q)$ is called as *ring LWE sample*. The *ring learning with errors* (Ring-LWE) problem is to find the secret polynomial $\mathbf{s}$ (and the error $\mathbf{e}$ simultaneously) from a given Ring-LWE sample of $(\mathbf{a}, \mathbf{b})$.

2.1 NewHope Key Exchange Protocol

An outline of the NewHope key exchange protocol is shown in Figure 1. Here we omit the procedures that are not directly related to the key mismatch attack, such as NTT (Number Theoretic Transform) being used to speed up polynomial multiplication. NewHope aims to securely exchange a shared key between Alice and Bob and it executes the below three steps. Note that the public polynomial $\mathbf{a}$ is shared in advance, which is sampled from $\mathcal{R}_q$ uniformly at random. The security of NewHope is based on the hardness of the Ring-LWE problem, where χ is the distribution of ψ_8^n.

1. Alice randomly samples a secret key $\mathbf{s}_A$ and an error $\mathbf{e}_A$ from ψ_8^n. Then, she calculates the public key $\mathbf{P}_A = \mathbf{a}\mathbf{s}_A + \mathbf{e}_A$ using the previously shared $\mathbf{a}(\in \mathcal{R}_q)$, and sends $\mathbf{P}_A$ to Bob. From the public key $\mathbf{P}_A$ and the previously shared polynomial $\mathbf{a}$, it is difficult to obtain information about the secret key $\mathbf{s}_A$ thanks to the hardness of Ring-LWE problem.
2. Bob selects $\mathbf{s}_B$, $\mathbf{e}_B$ and $\mathbf{e}'_B$ from ψ_8^n and computes the public key $\mathbf{P}_B = \mathbf{a}\mathbf{s}_B + \mathbf{e}_B$. Then, Bob chooses a 256-bit long bit string ν_B that is the basis of the shared key S_{k_B}, and hashes it by calculating $\nu'_B = \text{SHA3-256}\,(\nu_B)$. Subsequently, he computes $\mathbf{k} = \mathtt{Encode}\,(\nu'_B)\,, \quad \mathbf{c} = \mathbf{P}_A\mathbf{s}_B + \mathbf{e}'_B + \mathbf{k}, \quad \overline{\mathbf{c}} =$ Compress($\mathbf{c}$) and sends $(\mathbf{P}_B, \overline{\mathbf{c}})$ to Alice. The shared key S_{k_B} is obtained by calculating $S_{k_B} = \text{SHA3-256}\,(\nu'_B)$.
3. When Alice receives $(\mathbf{P}_B, \overline{\mathbf{c}})$, she calculates $\mathbf{k}' = \mathbf{c}' - \mathbf{P}_B\mathbf{s}_A = \mathbf{e}_A\mathbf{s}_B - \mathbf{e}_B\mathbf{s}_A + \mathbf{e}'_B + \mathbf{k}$. Alice can get ν'_A equal to ν'_B with high probability by computing $\mathtt{Decode}\,(\mathbf{k}')$ because the coefficients of $\mathbf{e}_A\mathbf{s}_B - \mathbf{e}_B\mathbf{s}_A + \mathbf{e}'_B$ are small. Then, she also gains a shared key $S_{k_A} = \text{SHA3-256}\,(\nu'_A)$.

In NewHope, $q = 12289$ and $n = 512$ or 1024 are employed. NewHope512 and NewHope1024 refer to the case of $n = 512$ and $n = 1024$, respectively. In the five security levels defined by NIST, NewHope512 is at the lowest level (level 1), and NewHope1024 is at the highest level (level 5) [1]. In this paper, we deal with the higher secure NewHope1024.

2.2 The Functions Used in NewHope

We simply review four functions being used in NewHope (Figure 1): $\mathtt{Compress}(\mathbf{c})$, $\mathtt{Decompress}(\overline{\mathbf{c}})$, $\mathtt{Encode}(\nu'_B)$, and $\mathtt{Decode}(\mathbf{k}')$.

The `Compress` function (Algorithm 1) and the `Decompress` function (Algorithm 2) perform coefficient-wise modulus switching between modulus q and 8. By compressing $\mathbf{c} \in \mathcal{R}_q$, the total size of coefficients becomes smaller; thereby the transmission cost is lower.

The function `Encode` (Algorithm 3) takes a 256-bit string ν'_B as an input and maps each bit to four coefficients in $\mathbf{k} \in \mathcal{R}_q$: $\mathbf{k}[i]$, $\mathbf{k}[i + 256]$, $\mathbf{k}[i + 512]$, and $\mathbf{k}[i + 768]$ (for $i = 0 \cdots 255$). In contrast, The function `Decode` (Algorithm 4) restores each bit of $\nu'_A \in \{0, 1\}^{256}$ from four coefficients in $\mathbf{k}' \in \mathcal{R}_q$. Namely, $\nu'_A[i] = 1$ if the summation of the four coefficients is smaller than q, and $\nu'_A[i] = 0$ otherwise.

3 Key Mismatch Attack on NewHope

In this section, we first explain a general model of a key mismatch attack on NewHope. Then we recall the attacks proposed by Bauer et al. [4] and Qin et al. [13], respectively.

Algorithm 1: `Compress`($\mathbf{c}$)

```
  Input: $\mathbf{c} \in \mathcal{R}_q$
  Output: $\bar{\mathbf{c}} \in \mathcal{R}_8$
1 for $i \leftarrow 0$ to 255 do
2     $\bar{\mathbf{c}}[i] \leftarrow \lfloor (\mathbf{c}[i] \cdot 8)/q \rceil \pmod 8$
3 Return $\bar{\mathbf{c}}$
```

Algorithm 2: `Decompress`($\bar{\mathbf{c}}$)

```
  Input: $\bar{\mathbf{c}} \in \mathcal{R}_8$
  Output: $\mathbf{c}' \in \mathcal{R}_q$
1 for $i \leftarrow 0$ to 255 do
2     $\mathbf{c}'[i] \leftarrow \lfloor (\bar{\mathbf{c}}[i] \cdot q)/8 \rceil$
3 Return $\mathbf{c}'$
```

Algorithm 3: `Encode`(ν'_B)

```
  Input: $\nu'_B \in \{0,1\}^{256}$
  Output: $\mathbf{k} \in \mathcal{R}_q$
1 $\mathbf{k} \leftarrow 0$
2 for $i \leftarrow 0$ to 255 do
3     for $j \leftarrow 0$ to 3 do
4         $\mathbf{k}[i + 256j] \leftarrow 4s \cdot \nu'_B[i]$
5 Return $\mathbf{k}$
```

Algorithm 4: `Decode`($\mathbf{k}'$)

```
  Input: $\mathbf{k}' \in \mathcal{R}_q$
  Output: $\nu'_A \in \{0,1\}^{256}$
1 $\nu'_A \leftarrow 0$
2 for $i \leftarrow 0$ to 255 do
3     $m \leftarrow \sum_{j=0}^{3} |\mathbf{k}'[i + 256j] - 4s|$
4     if $m < q$ then
5         $\nu'_A[i] \leftarrow 1$
      else
6         $\nu'_A[i] \leftarrow 0$
7 Return $\nu'_A$
```

3.1 The General Model

In the model of the key mismatch attack, we assume that Alice is an honest server and Bob plays the role of an adversary in Figure 1. An adversary sends a query including $(\mathbf{P}_B, \bar{\mathbf{c}}, S_{k_B})$ to the server. Then, the server calculates the shared key S_{k_A} and returns whether S_{k_A} and S_{k_B} match or mismatch. Here, the server is set to reuse the same secret key and honestly respond to any number of queries.

For the sake of convenience, we build an oracle $\mathcal{O}$ (Algorithm 5) to simulate the behavior of the server in this paper. The oracle outputs 1 if $S_{k_A} = S_{k_B}$ and outputs 0 otherwise. Changing the formats of queries sent to the oracle, the adversary can get information about $\mathbf{s}_A$ by observing the responses.

3.2 Bauer et al.'s Method

Bauer et al. proposed a method for recovering the coefficients in $[-6, 4]$ of the secret key $\mathbf{s}_A$. To recover the coefficient $\mathbf{s}_A[i]$, the adversary forges the following query and send it to the oracle.

$$\begin{cases} \nu'_B = (1, 0, \cdots, 0) \\ \mathbf{P}_B = \dfrac{s}{2} x^{-i'} \qquad (i' \equiv i \pmod{256}) \\ \bar{\mathbf{c}} = \displaystyle\sum_{j=0}^{3} (l_j + 4)\, x^{256j} \qquad (l_j \in [-4, 3]) \end{cases}$$

Algorithm 5: $\texttt{Oracle}(\mathbf{P}_B, \overline{\mathbf{c}}, S_{k_B})$

```
    Input: $\mathbf{P}_B, \overline{\mathbf{c}}, S_{k_B}$
    Output: 1 or 0
1   $\mathbf{c}' \leftarrow \texttt{Decompress}(\overline{\mathbf{c}})$
2   $\mathbf{k}' \leftarrow \mathbf{c}' - \mathbf{P}_B \mathbf{s}_A$
3   $\nu'_A \leftarrow \texttt{Decode}\,(\mathbf{k}')$
4   $S_{k_A} \leftarrow$ SHA3-256 $(\nu'_A)$
5   if $S_{k_A} = S_{k_B}$ then
6       Return 1
    else
7       Return 0
```

In this case, $\mathbf{k}'$ is calculated as follows:

$$\begin{aligned}\mathbf{k}' &= \mathbf{c}' - \mathbf{P}_B \mathbf{s}_A \\ &= \sum_{j=0}^{3} \left(\texttt{Decompress}(\overline{\mathbf{c}})[256j] - \frac{s}{2}\mathbf{s}_A[i' + 256j] \right) x^{256j} \\ &\quad + \sum_{\substack{k \not\equiv i \\ (\text{mod } 256)}} \left(0 - \frac{s}{2}\mathbf{s}_A[k] \right) x^k . \end{aligned} \tag{1}$$

In ν'_A $(= \text{Decode}(\mathbf{k}'))$, all elements except $\nu'_A[0]$ are calculated from the second term of Equation (1) and they become 0 with high probability. Therefore, the key mismatch corresponds to the mismatch between $\nu'_B[0] (= 1)$ and $\nu'_A[0]$, which depends on selected l_j $(j = 0, 1, 2, 3)$. The adversary fixes l_j other than $l_{\lfloor \frac{i}{256} \rfloor}$ to a random value and observes the change of the oracle's output by increasing $l_{\lfloor \frac{i}{256} \rfloor}$ from -4 to 3. If the adversary gets a string of outputs like $1, \cdots, 1, 0, \cdots, 0, 1, \cdots, 1$, he can calculate an estimated value τ of $\mathbf{s}_A[i]$ from the two possible values $\tau_1 < \tau_2$. The oracle's output goes from 1 to 0 at point $l_{\lfloor \frac{i}{256} \rfloor} = \tau_1$ and then from 0 to 1 at point $l_{\lfloor \frac{i}{256} \rfloor} = \tau_2$. Here such a form of outputs is called a favorable case. Please refer to [4] for the details of the attack.

3.3 Qin et al.'s Method

Qin et al. pointed out the low success rate of the attack proposed by Bauer et al. They proposed the following improvements to Bauer et al.'s attack on recovering the coefficients in $[-6, 4]$. In their method, they used the same way to generate the queries as Bauer et al.'s. However, they observed that a form of outputs like $0, \cdots, 0, 1, \cdots, 1, 0, \cdots, 0$ is also a favorable case. Besides, they indicated that Bauer et al.'s attack algorithm is not deterministic. For example, in the case of $\mathbf{s}_A[i] = 2$, the adversary may get a value of the incorrect 1 along with the correct 2. Since the attack algorithm is probabilistic, the more times the attacks are applied, the higher success probability can be achieved. However, the number of queries increases accordingly. To take a balance between the cost and the success rate, Qin et al. decided to collect 50 τs for each coefficient and recover the coefficient from the breakdown of them.

Additionally, they proposed a new attack for coefficients in $\{-8,-7,5,6,7,8\}$. However, this attack is conditional and its success rate is smaller than 11.2%. We analyze the details of their attack and show its drawbacks in Appendix A.

4 Our Improved Method

We propose an improved method to increase the success rate of key recovery and reduce the number of queries. In this section, we first describe the improvements in three parts, and finally introduce the overall attack flow.

4.1 Improvement on the Construction of Queries

We focus on the point that there are some secret key patterns that cannot be recovered in Qin et al.'s and Bauer et al.'s attacks. In their attacks, when recovering the coefficients in $[-6,4]$, an adversary sets $\nu'_B = (1,0,\cdots,0)$. In this case, depending on the pattern of $\mathbf{s}_A$, some elements except for $\nu'_A[0]$ become unexpected value of 1 where $\nu'_A = \texttt{Decode}(\mathbf{k}')$. Due to this, the oracle keeps returning 0 because $\nu'_A \neq \nu'_B$ regardless of the value of $\nu'_A[0]$. Therefore, a key mismatch attack will never be established.

To solve this problem, we propose a new query construction. In our method, when an adversary wants to recover the coefficient $\mathbf{s}_A[i]$, he directly sets the query like

$$\begin{cases} \mathbf{P}_B = \dfrac{s}{2} \\ \overline{\mathbf{c}} = \displaystyle\sum_{j=0}^{3} (l_j+4)\, x^{i'+256j} \qquad (l_j \in [-4,3],\ i' \equiv i \pmod{256}). \end{cases}$$

The oracle receives it and calculates

$$\begin{aligned} \mathbf{k}' &= \mathbf{c}' - \mathbf{P}_B \mathbf{s}_A \\ &= \sum_{j=0}^{3} \left(\texttt{Decompress}(\overline{\mathbf{c}})[256j] - \frac{s}{2}\mathbf{s}_A[i'+256j] \right) x^{256j} \\ &\quad + \sum_{\substack{k \not\equiv 0 \\ \pmod{256}}} \left(0 - \frac{s}{2}\mathbf{s}_A\right)[k] x^k. \end{aligned} \tag{2}$$

Then, the oracle gets $\nu'_A = (\texttt{Decode}(\mathbf{k}'))$, where $\nu'_A[i']$ is calculated from the first term of Formula (2) and other elements are from the second term. Here, if ν'_B meets two conditions such as

$$\begin{cases} \nu'_B = \texttt{Decode}(0 - \dfrac{s}{2}\mathbf{s}_A), & (3) \\ \nu'_B[i'] = 1, & (4) \end{cases}$$

Algorithm 6: `Find`$-\nu'_{\mathtt{B}}()$

```
Output: $\nu'_B \in \{0,1\}^{256}$
1  $\mathbf{P}_B \leftarrow \frac{s}{2}$, $\overline{\mathbf{c}} \leftarrow 0$
2  $\nu'_B \leftarrow (0, \cdots, 0)$
3  $S_{k_B} \leftarrow$ SHA3-256 $(\nu'_B)$
4  if $\mathcal{O}(\mathbf{P}_B, \overline{\mathbf{c}}, S_{k_B}) = 1$ then
5      Return $\nu'_B$
6  for $i \leftarrow 0$ to 255 do
7      $\nu'_B \leftarrow (0, \cdots, 0)$
8      $\nu'_B[i] \leftarrow 1$
9      $S_{k_B} \leftarrow$ SHA3-256 $(\nu'_B)$
10     if $\mathcal{O}(\mathbf{P}_B, \overline{\mathbf{c}}, S_{k_B}) = 1$ then
11         Return $\nu'_B$
12 for $i \leftarrow 0$ to 255 do
13     for $j \leftarrow i + 1$ to 255 do
14         $\nu'_B \leftarrow (0, \cdots, 0)$
15         $\nu'_B[i] \leftarrow 1$, $\nu'_B[j] \leftarrow 1$
16         $S_{k_B} \leftarrow$ SHA3-256 $(\nu'_B)$
17         if $\mathcal{O}(\mathbf{P}_B, \overline{\mathbf{c}}, S_{k_B}) = 1$ then
18             Return $\nu'_B$
19 Terminate the entire program
```

the value returned by the oracle (matching or mismatching between the two shared keys) can be reduced to the value of $\nu'_A[i']$.

In Algorithm 6, we show how to set ν'_B to satisfy the above two conditions. Here we decide to find ν'_B that meets Equation (3) by the exhaustive search. First, $\mathbf{P}_B$ and $\overline{\mathbf{c}}$ are set as $\mathbf{P}_B = \frac{s}{2}, \overline{\mathbf{c}} = 0$. Then, ν'_B is set to be an 256-bit string in the following orders: (Ⅰ) all elements are 0, (Ⅱ) except for one 1, all elements are 0, and (Ⅲ) except for two 1s, all elements are 0. Simultaneously, the adversary sends a query $(\mathbf{P}_B, \overline{\mathbf{c}}, S_{k_B})$ to the oracle. When the oracle returns 1, this algorithm stops and returns ν'_B. In contrast, if the oracle does not return 1 even after examining all of the above patterns (Ⅰ)(Ⅱ)(Ⅲ), the algorithm can judge that the secret key $\mathbf{s}_A$ cannot be recovered successfully, and the whole program is terminated. The reason why we only deals with three patterns (Ⅰ)(Ⅱ)(Ⅲ) is as follows.

1. The case that ν'_B includes three or more 1s appears with probability of only 0.003% (see Table 1).
2. For the case of ν'_B with three or more 1s, the exhaustive search needs much more queries than Qin et al.'s method. For instance, it takes ${}_{256}\mathrm{C}_3 = 2,763,520$ queries when searching ν'_B with three 1s.

By performing Algorithm 6, ν'_B that satisfies the Equation (3) is set. Then, when querying the oracle, an adversary has to additionally set $\nu'_B[i'] = 1$ to meet Equation (4).

Table 1. Distribution of secret key $\mathbf{s}_A$

the number of 1 included in $\texttt{Decode}(0 - \frac{s}{2}\mathbf{s}_A)$	secret key $\mathbf{s}_A$
0	94.547%
1	5.302%
2	0.148%
3 or more	0.003%

4.2 Extending the Search Range of Secret Key

After setting a proper query, an adversary changes $l_{\lfloor \frac{i}{256} \rfloor}$ from -4 to 3 analogous to Qin et al.'s method. Then, he calculates an estimated value τ of the coefficient $\mathbf{s}_A[i]$ by observing the oracle's outputs. Qin et al. claimed that this attack was valid on coefficients of $\mathbf{s}_A[i]$ in [-6,4]. However, we point out that the attack can be applied to a wider range such as [-6,7], without any additional queries.

It is clear that the oracle's output is relative to $\nu'_A[i']$ whose value depends on the size of m comparing with the size of q (Algorithm 4). Remark that m is calculated by

$$\begin{aligned} m &= \sum_{j=0}^{3} |\mathbf{k}'[i' + 256j] - 4s| \\ &\approx \sum_{j=0}^{3} \left| (l_j + 4)\, s - \frac{s}{2}\mathbf{s}_A[i' + 256j] - 4s \right| \\ &= \sum_{j=0}^{3} \left| l_j - \frac{1}{2}\mathbf{s}_A[i' + 256j] \right| s. \end{aligned}$$

Moreover, all l_j ($j \in \{0,1,2,3\} \backslash \{\lfloor \frac{i}{256} \rfloor\}$) are fixed at random values. From these facts, we can conclude that the string of oracle's outputs depends on the change of

$$u = |\mathbf{k}'[i] - 4s|\,. \tag{5}$$

Meanwhile, there are two kinds of favorable cases such as $0, \cdots, 0,\ 1, \cdots, 1,\ 0, \cdots, 0$ and $1, \cdots, 1, 0, \cdots, 0, 1, \cdots, 1$. Next, we study the condition of u for meeting favorable cases from the oracle.

We explain the case of $1, \cdots, 1, 0, \cdots, 0, 1, \cdots, 1$ here, and the analysis for the case of $0, \cdots, 0,\ 1, \cdots, 1,\ 0, \cdots, 0$ is similar due to symmetry. For example, an adversary wants to recover $\mathbf{s}'_A[i] = -7$. He randomly fixes l_j ($j \in \{0,1,2,3\} \backslash \{\lfloor \frac{i}{256} \rfloor\}$) first. Then, we assume that he gets $v = 6s$ which is fixed by the computation of

$$\begin{aligned} v &= m - u \\ &= \sum_{i' + 256j \neq i} |\mathbf{k}'[i' + 256j] - 4s|\,. \end{aligned}$$

In this case, when $l_{\lfloor \frac{i}{256} \rfloor}$ varies, the behavior of m used in `Decode` function and the oracle's output is shown in Table 2.

Table 2. The behavior of m and the oracle's output($\mathbf{s}'_A[i] = -7$)

$l_{\lfloor \frac{i}{256} \rfloor}$	-4	-3	-2	-1	0	1	2	3
$u(= m - v)$	$0.5s$	$0.5s$	$1.5s$	$2.5s$	$3.5s$+1	$3.5s$	$2.5s$	$1.5s$
the oracle's output	1	1	1	0	0	0	0	1

We can conclude that if u monotonically increases and then monotonically decreases, the oracle's outputs become $1, \cdots, 1, 0, \cdots, 0, 1, \cdots, 1$. On the contrary, due to the symmetry, if u monotonically decreases and then monotonically increases, the oracle's outputs become $0, \cdots, 0, 1, \cdots, 1, 0, \cdots, 0$.

Table 3. The behavior of u (Formula 5) corresponding to parameter $l_{\lfloor \frac{i}{256} \rfloor}$ and $\mathbf{s}_A[i]$

u \ $l_{\lfloor \frac{i}{256} \rfloor}$ \ $\mathbf{s}_A[i]$	-4	-3	-2	-1	0	1	2	3
-8	0	s	$2s$	$3s$	$4s$	$3s$	$2s$	s
-7	$0.5s$	$0.5s$	$1.5s$	$2.5s$	$3.5s$+1	$3.5s$	$2.5s$	$1.5s$
-6	s	0	s	$2s$	$3s$+1	$4s$	$3s$	$2s$
-5	$1.5s$	$0.5s$	$0.5s$	$1.5s$	$2.5s$+1	$3.5s$+1	$3.5s$	$2.5s$
-4	$2s$	s	0	s	$2s$+1	$3s$+1	$4s$	$3s$
-3	$2.5s$	$1.5s$	$0.5s$	$0.5s$	$1.5s$+1	$2.5s$+1	$3.5s$+1	$3.5s$
-2	$3s$	$2s$	s	0	s+1	$2s$+1	$3s$+1	$4s$
-1	$3.5s$	$2.5s$	$1.5s$	$0.5s$	$0.5s$+1	$1.5s$+1	$2.5s$+1	$3.5s$+1
0	$4s$	$3s$	$2s$	s	1	s+1	$2s$+1	$3s$+1
1	$3.5s$+1	$3.5s$	$2.5s$	$1.5s$	$0.5s$-1	$0.5s$+1	$1.5s$+1	$2.5s$+1
2	$3s$+1	$4s$	$3s$	$2s$	s-1	1	s+1	$2s$+1
3	$2.5s$+1	$3.5s$+1	$3.5s$	$2.5s$	$1.5s$-1	$0.5s$-1	$0.5s$+1	$1.5s$+1
4	$2s$+1	$3s$+1	$4s$	$3s$	$2s$-1	s-1	1	s+1
5	$1.5s$+1	$2.5s$+1	$3.5s$+1	$3.5s$	$2.5s$-1	$1.5s$-1	$0.5s$-1	$0.5s$+1
6	s+1	$2s$+1	$3s$+1	$4s$	$3s$-1	$2s$-1	s-1	1
7	$0.5s$+1	$1.5s$+1	$2.5s$+1	$3.5s$+1	$3.5s$-1	$2.5s$-1	$1.5s$-1	$0.5s$-1
8	1	s+1	$2s$+1	$3s$+1	$4s$-1	$3s$-1	$2s$-1	s-1

Furthermore, we show how u changes according to $l_{\lfloor \frac{i}{256} \rfloor}$ for each value of $\mathbf{s}_A[i] \in [-8, 8]$ in Table 3. As we can see, for each $\mathbf{s}_A[i]$, u always monotonically increases and then decreases or monotonically decreases and then increases. Therefore, the estimated value τ of each $\mathbf{s}_A[i] \in [-8, 8]$ can be obtained theoretically.

Table 4. $\mathbf{s}_A[i]$ and possible τs, outputs of Algorithm 7

$\mathbf{s}_A[i]$	-8	-7		-6		-5		-4		-3		-2		-1		0		1		2		3		4		5		6		7		8	
τ	8	8	-7	-7	-6	-6	-5	-5	-4	-4	-3	-3	-2	-2	-1	-1	0	0	1	1	2	2	3	3	4	4	5	5	6	6	7	7	8

Algorithm 7: `Find`$-\tau(b)$

```
Input: $b \in \{0,1\}^8$
Output: $\tau \in [-7,8]$ or NULL
1  $\tau, \tau_1, \tau_2 \leftarrow$ NULL
2  for $i \leftarrow 0$ to 6 do
3      if $b[i] \neq b[i+1]$ then
4          if $\tau_1 =$ NULL then
5              $\tau_1 \leftarrow i - 4$
6          else if $\tau_2 =$ NULL then
7              $\tau_2 \leftarrow i - 3$
8  if $\tau_1 \neq$ NULL and $\tau_2 \neq$ NULL then
9      $\tau = \tau_1 + \tau_2$
10     if $\tau > 0$ and $b[0] = 1$ then
11         $\tau = \tau - 8$
12     else if $\tau \leq 0$ and $b[0] = 1$ then
13         $\tau = \tau + 8$
14 Return $\tau$
```

Next, we consider how to determine $\mathbf{s}_A[i]$ from τs. We invite Qin et al.'s algorithm (Algorithm 7) to calculate τ from the string of oracle's outputs b. Note that the output τ is expanded from -7 to 8, where τ belongs to [-6,4] in Qin et al.'s paper. We show the relationship between possible τs and $\mathbf{s}_A[i]$ in Table 4. In particular, three values of $\mathbf{s}_A[i] \in \{-8, -7, 8\}$ correspond to $\tau = 8$. Namely, when $\tau = 8$ appears, an adversary has to determine $\mathbf{s}_A[i]$ from {-8,-7,8}. Thus, we conduct this attack only against $\mathbf{s}_A[i] \in [-6, 7]$ in our improved method.

4.3 Early Abort Technique for Terminating Queries

In Qin et al.'s method, 50 τs are collected for each coefficient and $\mathbf{s}_A[i]$ is determined from the breakdown of them. For example, if 35 τs out of 50 are 3 and 15 τs are 2, then $\mathbf{s}_A[i] = 3$. However, this probabilistic process is inefficient. In our algorithm, we set a deterministic condition for judging $\mathbf{s}_A[i]$. As shown in Table 4, two types of τ correspond to unique value of $\mathbf{s}_A[i]$ in [-6,7]. Therefore, we can terminate collecting τ when two different types appear. We apply this condition at Step 20 in Algorithm 8.

Algorithm 8: `Partial−recovery`(p)

```
Input: $p \in \mathbb{N}$
Output: $\mathbf{s}'_A \in \mathcal{R}_q$
1  $\mathbf{P}_B \leftarrow \frac{s}{2}$
2  $\nu'_B \leftarrow$ Find−$\nu'_B()$
3  for $i' \leftarrow 0$ to 255 do
4      $\nu'_B[i'] \leftarrow 1$
5      $S_{k_B} \leftarrow$ SHA3-256 $(\nu'_B)$
6      for $j \leftarrow 0$ to 3 do
7          $count \leftarrow 0$
8          $t, temp \leftarrow$ NULL
9          while $count < p$ do
10             $(l_0, l_1, l_2, l_3) \xleftarrow{\$} [-4, 3]^4$
11             $b \leftarrow []$
12             for $l_j \leftarrow -4$ to 3 do
13                 $\overline{\mathbf{c}} \leftarrow \sum_{k=0}^{3} (l_k + 4)\, x^{i'+256k}$
14                 $b$.append $(\mathcal{O}(\mathbf{P}_B, \overline{\mathbf{c}}, S_{k_B}))$
15             $t \leftarrow$ Find−$\tau(b)$
16             if $t \neq$ NULL then
17                 $count = count + 1$
18                 if $temp =$ NULL then
19                     $temp \leftarrow t$
20                 else if $temp \neq t$ then
21                     break
22         if $temp \neq 8$ and $t \neq 8$ then
23             if $temp = t$ then
24                 $\mathbf{s}'_A[i' + 256j] \leftarrow temp$
               else
25                 $\mathbf{s}'_A[i' + 256j] \leftarrow \max(temp, t)$
           else
26             $\mathbf{s}'_A[i' + 256j] \leftarrow$ NULL
27 Return $\mathbf{s}'_A$
```

4.4 Our Proposed Algorithm

We first propose Algorithm 8 to recover the coefficient of $\mathbf{s}_A$ in [-6,7]. To launch this attack, an adversary sets $\mathbf{P}_B = \frac{s}{2}$, $\nu'_B =$ `Find`$-\nu'_B(), \nu'_B[i'] = 1$,and $\overline{\mathbf{c}} = \sum_{k=0}^{3} (l_k + 4)\, x^{i'+256k}$ according to the coefficient $\mathbf{s}_A[i' + 256j]$ that he wants to recover. Then, the queries including $(\mathbf{P}_B, \overline{\mathbf{c}}, S_{k_B})$ are sent to the oracle. When $l_{i'}$ varies from -4 to 3, the outputs from the oracle are appended to the array b (Step 14). Then the adversary gets τs from `Find`$-\tau(b)$. He keeps sending queries until he gets two different values of τ or he gets same τs for p times. We execute our experiments with different p and analyze the performance of the algorithm.

Algorithm 9: `Full−recovery`(p)

```
    Input: $p \in \mathbb{N}$
    Output: $\mathbf{s}'_A \in \mathcal{R}_q$
 1  $\mathbf{s}'_A \leftarrow$ Partial-recovery$(p)$
 2  $\mathbf{s}_B, \mathbf{e}_B, \mathbf{e}'_B \xleftarrow{\$} \psi_8^n$
 3  $\mathbf{P}_B \leftarrow \mathbf{a}\mathbf{s}_B + \mathbf{e}_B$
 4  $\nu_B \xleftarrow{\$} \{0,1\}^{256}$
 5  $\mathbf{k} \leftarrow$ Encode$(\nu'_B)$
 6  $\nu'_B \leftarrow$ SHA3-256$(\nu_B)$
 7  $S_{k_B} \leftarrow$ SHA3-256$(\nu'_B)$
 8  $index \leftarrow []$
 9  for $i \leftarrow 0$ to $1024$ do
10      if $\mathbf{s}'_A[i] = \text{NULL}$ then
11          $index$.append$(i)$
12  for $list \in \{-8, -7, 8\}^{index.\text{length}()}$ do
13      for $j \leftarrow 0$ to $index$.length$() - 1$ do
14          $\mathbf{s}'_A[index[j]] = list[j]$
15      $\mathbf{P}_A \leftarrow \mathbf{a}\mathbf{s}'_A$
16      $\mathbf{c} \leftarrow \mathbf{P}_A\mathbf{s}_B + \mathbf{e}'_B + \mathbf{k}$
17      $\bar{\mathbf{c}} \leftarrow$ Compress$(\mathbf{c})$
18      if $\mathcal{O}(\mathbf{P}_B, \bar{\mathbf{c}}, S_{k_B}) = 1$ then
19          break
20  Return $\mathbf{s}'_A$
```

If $\tau = 8$, this algorithm returns NULL without determining the value of the target coefficient (Step 26). Besides, if two different τs are obtained, the larger one is adopted (Step 25), otherwise $\mathbf{s}'_A[i' + 256j] = \tau$.

Finally, we propose Algorithm 9 to recover the entire secret key $\mathbf{s}_A$. Taking account of the fact that averagely only 0.28 coefficients in {-8, -7, 8} are included in one secret key, so we decide to perform an exhaustive search on them. First, by running `Partial-recovery`(p), an adversary gets $\mathbf{s}'_A$ whose coefficients in [-6,7] can be recovered. Next, he sets the public key $\mathbf{P}_B$, the secret key $\mathbf{s}_B$, and the error polynomial $\mathbf{e}_B, \mathbf{e}'_B$ in the same way as Bob does in Figure 1. ν_B is set random bit string, because it has no significant meaning in this exhaustive attack. Simultaneously, S_{k_B} is calculated by hashing ν_B. Finally, he exhaustively substitutes values in {-8,-7,8} for $\mathbf{s}'_A[i]$ where is NULL (Step 14). Originally, $\mathbf{P}_A$ sent from Alice is used when Bob calculates $\bar{\mathbf{c}}$. However, the goal of this attack is to check whether $\mathbf{s}'_A = \mathbf{s}_A$, thus $\mathbf{P}_A = \mathbf{a}\mathbf{s}'_A$ is used (Step 15). The queries generated by the above procedure are sent to the oracle repeatedly and when the oracle outputs 1, this program stops and returns $\mathbf{s}'_A$.

5 Our Experiments

At first, we implement NewHope1024-CPA-KEM [1] with parameters $(n, q) = (1024, 12289)$. Then, we generate 1000 secret keys $\mathbf{s}_A$ randomly. Our goal is to recover $\mathbf{s}_A$ by using our proposed Algorithm 9, where we try different parameters p in the set of $\{5, 10, 12, 15, 20, 30, 50\}$. All the algorithms and oracles are implemented by `Python3`. Polynomial calculation is implemented using `poly1d` of numerical calculation library `NumPy`. We run the programs on Intel Xeon Skylake Gold 6130 with CPUs at 2.1GHz.

Table 5. The success rate and the number of queries when increasing the value of p

The value of p	5	10	12	15	20	30	50
The success rate (%)	0	20.9	52.3	80.7	97.4	100.0	100.0
The number of queris	78,648	135,602	155,610	185,789	233,803	327,659	512,435

The experimental results are shown in Table 5, where we illustrate the trade-off between the number of required queries and the success rate. In addition, we also represent the data in Figure 2. It is notable that the attack can achieve 100.0% success rate when $p \geq 30$. Meanwhile, it indicates that decreasing p reduces not only the number of queries but also decreases the success rate. Specifically, if we sacrifice the success rate to 20.9%, we can reduce the number of queries to 135,602.

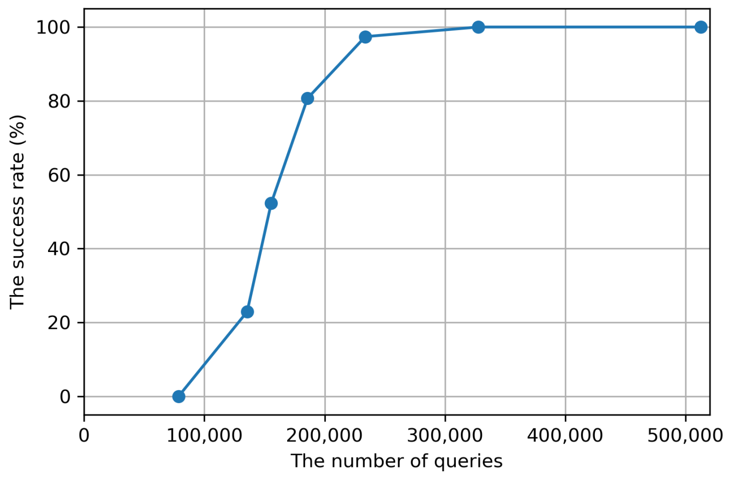

Fig. 2. The relationship between the number of queries and the success rate (each dot from left to right corresponds to the value of $p = 5, 10, \ldots, 50$)

Table 6. Comparison of experimental results of key mismatch attacks

	Qin et al. [13]	Our attack ($p = 20$)
The success rate (%)	96.9	97.4
The number of queries	879,725	**233,803**

Moreover, a comparison with Qin et al.'s experimental results is summarized in Table 6. In the case of $p = 20$, the success rate is almost the same as that of Qin et al.'s, but the number of queries is reduced by about 73%.

6 Conclusion

In this paper, we improved Qin et al's key mismatch attack on NewHope, and evaluated the trade-off between the number of queries and the success rate of recovering the secret keys. As a result, the number of required queries can be reduced by about 73% with almost the same success rate as that in Qin et al.'s attack. Moreover, our attack can achieve 100.0% success rate.

The current key mismatch attacks, including ours, Qin et al.'s, and Bauer et al.'s, are feasible under the assumption that the server honestly responses to freely chosen queries and continues to reuse the secret key. For this reason, they are not applicative to the CCA-secure NewHope using Fujisaki-Okamoto transformation [9]. However, considering the aspect of efficiency in practical use, there may be some cases where only CPA-secure NewHope is used without updating the server's secret key for a time. Then it is necessary to take some countermeasures to prevent the leakage of secret information, such as updating keys periodically or setting a detection system to malicious queries.

Acknowledgments Acknowledgments We thank Dr. Atsushi Takayasu for his helpful comments on this work. This work was supported by JSPS KAKENHI Grant Number 19F19378, JST CREST Grant Number JPMJCR14D6, Japan.

References

1. US Department of Commerce, National Institute of Standards and Technology. Post-Quantum Cryptography, 2019. `http://csrc.nist.gov/projects/post-quantum-cryptography/`.
2. E. Alkim, L. Ducas, T. Pöppelmann, and P. Schwabe. Post-quantum key exchange - A New Hope. In *25th USENIX Security Symposium, USENIX Security 16, August 10-12, 2016, Proceedings*, pages 327–343, 2016.
3. C. Băetu, F. B. Durak, L. Huguenin-Dumittan, A. Talayhan, and S. Vaudenay. Misuse attacks on post-quantum cryptosystems. In *EUROCRYPT 2019, May 19-23, 2019, Proceedings, Part II*, pages 747–776, 2019.
4. A. Bauer, H. Gilbert, G. Renault, and M. Rossi. Assessment of the key-reuse resilience of NewHope. In *The Cryptographers' Track at the RSA Conference 2019,CT-RSA 2019, March 4-8, 2019, Proceedings*, pages 272–292, 2019.

5. J. Ding, S. Alsayigh, R. V. Saraswathy, S. R. Fluhrer, and X. Lin. Leakage of signal function with reused keys in RLWE key exchange. In *IEEE International Conference on Communications, ICC 2017, May 21-25, 2017,Proceedings*, pages 1–6, 2017.
6. J. Ding, J. Deaton, K. Schmidt, Vishakha, and Z. Zhang. A simple and practical key reuse attack on NTRU cryptosystem. *IACR Cryptology ePrint Archive*, 2020:1022, 2019. `http://eprint.iacr.org/2019/1022`.
7. J. Ding, S. R. Fluhrer, and S. RV. Complete attack on RLWE key exchange with reused keys, without signal leakage. In *Information Security and Privacy - 23rd Australasian Conference, ACISP 2018, July 11-13, 2018, Proceedings*, pages 467–486, 2018.
8. S. R. Fluhrer. Cryptanalysis of ring-LWE based key exchange with key share reuse. *IACR Cryptology ePrint Archive*, 2016:85, 2016. `http://eprint.iacr.org/2016/085`.
9. E. Fujisaki and T. Okamoto. Secure integration of asymmetric and symmetric encryption schemes. *J. Cryptology*, 26(1):80–101, 2013.
10. A. Greuet, S. Montoya, and G. Renault. Attack on LAC key exchange in misuse situation. *IACR Cryptology ePrint Archive*, 2020:63, 2020. `http://eprint.iacr.org/2020/063`.
11. C. Liu, Z. Zheng, and G. Zou. Key reuse attack on NewHope key exchange protocol. In *Information Security and Cryptology, ICISC 2018, November 28-30, 2018, Revised Selected Papers*, pages 163–176, 2018.
12. V. Lyubashevsky, C. Peikert, and O. Regev. On ideal lattices and learning with errors over rings. In *EUROCRYPT 2010, May 30 - June 3, 2010, Proceedings*, pages 1–23, 2010.
13. Y. Qin, C. Cheng, and J. Ding. A complete and optimized key mismatch attack on NIST candidate newhope. In *24th European Symposium on Research in Computer Security,ESORICS 2019,September 23-27, 2019, Proceedings, Part II*, pages 504–520, 2019.
14. Y. Qin, C. Cheng, and J. Ding. An efficient key mismatch attack on the NIST second round candidate Kyber. *IACR Cryptology ePrint Archive*, 2019:1343, 2019. `http://eprint.iacr.org/2019/1343`.
15. O. Regev. On lattices, learning with errors, random linear codes, and cryptography. *J. ACM*, 56(6):34:1–34:40, 2009.
16. E. Rescorla. The transport layer security (TLS) protocol version 1.3. Technical report. `http://www.rfc-editor.org/info/rfc8446`.
17. P. W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput.*, 26(5):1484–1509, 1997.

Appendix A An Observation of Qin et al.'s Attack.

A.1 Qin et al.'s method [13]

Qin et al. analyzed the distribution of the quadruplet ($\mathbf{s}_A[i]$,$\mathbf{s}_A[i+256]$, $\mathbf{s}_A[i+512]$, and $\mathbf{s}_A[i+768]$). They show that with 98.50% probability, there are 3 coefficients in $\{-6,-5,\ldots,2,3,4\}$ and 1 coefficient is in {-8,-7,5,6,7,8} in each quadruplet (Table 4, [13]). In the following, we call this quadruplet Ξ for convenience sake.

Assume that an adversary wants to recover $\mathbf{s}_A[i]\,(\in \{-8,-7,5,6,7,8\})$ after he recovered other 3 coefficients, $\mathbf{s}_A[i+256]$, $\mathbf{s}_A[i+512]$, and $\mathbf{s}_A[i+768]$ by their attack for the coefficients in [-6,4]. He sets each polynomials with parameter h_1 as follows:

$$\begin{cases} \mathbf{s}_B, \mathbf{e}'_B = 0 \\ \mathbf{e}_B = h_1 x^{512}\,(h_1 \in \mathbb{Z}_q) \\ \nu'_B = (0,\cdots,0,1,0,\cdots,0)(\text{only } i\text{-th element is } 1). \end{cases}$$

After receiving the query, the server has

$$\begin{aligned} \mathbf{k}' &= \sum_{j=0}^{3}(4s+1)x^{i+256j} - h_1 x^{512}\mathbf{s}_A \\ &= \sum_{j=0}^{3}(4s+1)x^{i+256j} - \big(-h_1\mathbf{s}_A[512] - \cdots - h_1\mathbf{s}_A[1023]x^{511} \\ &\quad + h_1\mathbf{s}_A[0]x^{512} + \cdots + h_1\mathbf{s}_A[511]x^{1023}\big)\,. \end{aligned}$$

Finally, the parameter m used in `Decode` function to calculate $\nu'_A[i]$ is like

$$\begin{aligned} m &= \sum_{j=0}^{3}|\mathbf{k}'[i+256j]-4s| \\ &\approx |1+\mathbf{s}_A[i+512]h_1| + |1+\mathbf{s}_A[i+768]h_1| + |1-\mathbf{s}_A[i]h_1| + |1-\mathbf{s}_A[i+256]h_1| \\ &\approx (|\mathbf{s}_A[i]| + |\mathbf{s}_A[i+256]| + |\mathbf{s}_A[i+512]| + |\mathbf{s}_A[i+768]|)\,h_1. \end{aligned}$$

It sets $\nu'_A[i]=1$ if this $m \geq q$, and $\nu'_A[i]=0$ otherwise. If $\nu'_A[\ell]=0(\forall \ell \neq i)$, the oracle's output corresponds to the relationship between m and q. In other words, if h_1 is so small that $m<q$, the oracle keeps returning 1, but only after h_1 becomes large enough such that $m \geq q$, it returns 0. The adversary varies the value of h_1 from 1 to 12289. At the point that the oracle's output changes from 1 to 0, the following equation holds.

$$q \approx (|\mathbf{s}_A[i]| + |\mathbf{s}_A[i+256]| + |\mathbf{s}_A[i+512]| + |\mathbf{s}_A[i+768]|)\,h_1$$

At this time, the adversary have

$$|\mathbf{s}_A[i]| + |\mathbf{s}_A[i+256]| + |\mathbf{s}_A[i+512]| + |\mathbf{s}_A[i+768]| \approx \frac{q}{h_1}.$$

He knows 3 coefficients except for $\mathbf{s}_A[i]$ in the quadruplet. Therefore, he can calculate $|\mathbf{s}_A[i]|$ from Equation A.1. Furthermore he can determine $\mathbf{s}_A[i]$ from the absolute value and the information on the sign of $\mathbf{s}_A[i]$ which is obtained by the attack on the coefficients in [-6,4].

A.2 The Condition for Qin et al.'s Method

In this subsection, we consider the conditions for the success of Qin et al.'s attack. Hereinafter, we define m_i as m used in `Decode` function to calculate $\nu'_A[i]$. As shown above,

$$\begin{aligned} m_i &= \sum_{j=0}^{3} |\mathbf{k}'[i+256j] - 4s| \\ &\approx (|\mathbf{s}_A[i]| + |\mathbf{s}_A[i+256]| + |\mathbf{s}_A[i+512]| + |\mathbf{s}_A[i+768]|)\, h_1. \end{aligned}$$

Meanwhile, for all $\ell \neq i$, we have

$$\begin{aligned} m_\ell &= \sum_{j=0}^{3} |\mathbf{k}'[\ell+256j] - 4s| \\ &\approx 16s - (|\mathbf{s}_A[\ell]| + |\mathbf{s}_A[\ell+256]| + |\mathbf{s}_A[\ell+512]| + |\mathbf{s}_A[\ell+768]|)\, h_1 \end{aligned}$$

when h_1 is small. In this method, an adversary can calculate $\mathbf{s}_A[i]$ only when $\nu'_A[i]$ changes from 1 to 0 and all other elements of ν'_A keep 0s. Namely, until m_i becomes no smaller than q, m_ℓ ($\forall \ell \neq i$) must be smaller than q. Therefore, in Qin et al.'s attack, the following inequality is required for all ℓ.

$$\sum_{j=0}^{3} |\mathbf{s}_A[i+256j]| > \sum_{j=0}^{3} |\mathbf{s}_A[\ell+256j]|. \tag{6}$$

A.3 The Success Rate of Qin et al.'s Method

We show the distribution of the maximum value of $\sum_{j=0}^{3} |\mathbf{s}_A[\ell'+256j]|$ $(0 \leq \ell' \leq 255)$ in Table 7. We can see that $\max_{\ell'}(\sum_{j=0}^{3} |\mathbf{s}_A[\ell'+256j]|)$ is larger than 12 with (almost) 100% for every $\mathbf{s}_A$. In Table 8, we also show the distribution of Ξ about its summation ($= \sum_{j=0}^{3} |\mathbf{s}_A[i+256j]|$).

Table 7. The distribution of $\max_{\ell'}(\sum_{j=0}^{3} |\mathbf{s}_A[\ell'+256j]|)$

$\max_{\ell'}(\sum_{j=0}^{3} \lvert\mathbf{s}_A[\ell'+256j]\rvert)$	$\mathbf{s}_A$
12	5%
13	25%
14	34%
15	22%
16	10%
17	3%
18	1%

Table 8. The distribution of Ξ about $\sum_{j=0}^{3} |\mathbf{s}_A[i+256j]|$

$\sum_{j=0}^{3} \lvert\mathbf{s}_A[i+256j]\rvert$	Ξ
5	0.6%
6	3.4%
7	8.8%
8	14.8%
9	18.3%
10	18.1%
11	14.7%
12	10.1%
13	6.0%
14	3.1%
15	1.3%
16 or larger	0.8%

We can estimate the success rate of Qin et al.'s attack by using Inequality (6) and Table 8. For instance, if $\max_{\ell}(\sum_{j=0}^{3} |\mathbf{s}_A[\ell+256j]|) = 12$, $\mathbf{s}_A[i]$ can be calculated only if $\sum_{j=0}^{3} |\mathbf{s}_A[i+256j]|$ is 13 or larger. In this case, the success rate of recovering $\mathbf{s}_A[i]$ is 11.2%(= 6.0% + 3.1% + 1.3% + 0.8%). We show the relationship between the success rate and the values of $\max_{\ell}(\sum_{j=0}^{3} |\mathbf{s}_A[\ell+256j]|)$ in Table 9. We can conclude that the success rate of Qin et al.'s method for recovering coefficients in {-8,-7,5,6,7,8} is not so high.

Table 9. The relationship between $\max_{\ell}(\sum_{j=0}^{3} |\mathbf{s}_A[\ell+256j]|)$ and the success rate

$\max_{\ell}(\sum_{j=0}^{3} \lvert\mathbf{s}_A[\ell+256j]\rvert)$	The success rate
12	11.2%
13	6.2%
14	2.1%
15	0.8%