A Complete and Optimized Key Mismatch Attack on NIST Candidate NewHope

Yue Qin[1], Chi Cheng[1], and Jintai Ding[2]

[1] China University of Geosciences, Wuhan, 430074, China
{qy52hz,chengchi}@cug.edu.cn
[2] University of Cincinnati, Cincinnati, 45219, USA
jintai.ding@gmail.com

Abstract. In CT-RSA 2019, Bauer et al. have analyzed the case when the public key is reused for the NewHope key encapsulation mechanism (KEM), a second-round candidate in the NIST Post-quantum Standard process. They proposed an elegant method to recover coefficients ranging from -6 to 4 in the secret key. We repeat their experiments but there are two fundamental problems. First, even for coefficients in [-6,4] we cannot recover at least 262 of them in each secret key with 1024 coefficients. Second, for the coefficient outside [-6,4], they suggested an exhaustive search. But for each secret key on average there are 10 coefficients that need to be exhaustively searched, and each of them has 6 possibilities. This makes Bauer et al.'s method highly inefficient. We propose an improved method, which with 99.22% probability recovers all the coefficients ranging from -6 to 4 in the secret key. Then, inspired by Ding et al.'s key mismatch attack, we propose an efficient strategy which with a probability of 96.88% succeeds in recovering all the coefficients in the secret key. Experiments show that our proposed method is very efficient, which completes the attack in about 137.56 ms using the NewHope parameters.

Keywords: Post-quantum cryptography · Key exchange · Ring learning with errors · Key mismatch attack.

1 Introduction

Currently, the standardization process of post-quantum cryptography algorithms run by the NIST has completed the first round and the second round workshop is scheduled to be held on August, 2019 [1]. As one of the most promising candidates for future post-quantum cryptography standard, the ring learning with errors (Ring-LWE) based approaches have attracted a lot of attention due to the provable security and high efficiency [13, 15, 17].

To construct DH-like key exchange schemes whose hardness are based on the Ring-LWE problem, the key breakthrouigh is to use the error reconciliation mechanism, which means that one party needs to send additional information to help the other party agree on an exactly same key. The first paper proposing this idea was attributed to Ding, Xie, and Lin [10]. Then, an authenticated key exchange variant was proposed by Zhang et al. [19]. Peikert proposed a key encapsulation mechanism (KEM) using a tweaked error correction mechanism in [16], which is then reformulated by Bos et al. as a key exchange scheme and inserted into the Transport Layer Security (TLS) protocol [6]. Later, a further tweaked Ring-LWE based key exchange scheme, the NewHope-Usenix [4], 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 [3], where the authors use the encryption-based approach to transfer the keys but the key idea of reconciliation of sending interval information is again deployed in the name of Encode. In the submission to the competition of NIST's post-quantum cryptography, the submitted NewHope [2] was based on NewHope-simple, and 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 TLS 1.3 [18], 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 the key reuse attacks. 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 shared keys generated by the two parties 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 [11]. Later, Ding et al. has developed a key leakage attack on [10], where the reused keys leak information about the secret key [7]. In [9], a key mismatch attack was proposed on the one pass case of [10], without using the information leaked by the signal function. To thwart the proposed key leakage attack in case the public key is required to be reused, in [12] a randomized method has been proposed. Another related work is [14], in which Liu et al. proposed a signal leakage attack against the reconciliation-based NewHope-Usenix key exchange protocol [4].

Unlike the DH-like key exchange protocols, the NewHope KEM submitted to the NIST [3] is based on the encryption rather than the reconciliation mechanism, and newly designed Encode and Compress functions are used. Therefore, these attacks proposed by Fluhrer [11], Ding et al. [7–9], or Liu et al. [14] cannot be directly applied to the encryption-based NewHope key exchange protocol [2]. The main challenge for launching a key mismatch attack is that the Encode and Decode functions in NewHope deal with four coefficients together, which makes it hard to recover the secret key using the previous methods.

In CT-RSA 2019, Bauer et al. have proposed a key mismatch attack on NewHope [5]. As we know, the coefficients of the secret key in NewHope belong to $[-8, 8]$ due to the fact that they are selected from the centered binomial distribution ψ_8^n. The key observation of Bauer et al. is that in a secret key with 1024 coefficients, 99.22% of them lie in $[-6, 4]$. From this observation, they have proposed an elegant method, which is claimed to recover all the coefficients belonging to $[-6, 4]$ in the key.

However, their recovery is first incomplete. Through our experiments, for each secret key with 1024 coefficients there are at least 262 coefficients in $[-6, 4]$ but cannot be recovered using their method. Second, for the coefficients outside $[-6, 4]$, i.e. those selected from $\{-8, -7, 5, 6, 7, 8\}$, they suggested an exhaustive search. But for each secret key on average there are 10 coefficients that need to be exhaustively searched, and each of them has 6 possibilities. The resulted $6^{10} \approx 6 \times 10^7$ possibilities make Bauer et al.'s method highly inefficient.

After analyzing the cause of the incomplete recovery, we propose an improved method, which with 99.22% probability can recover all the coefficients ranging from -6 to 4 in the secret key. Then, inspired by Ding et al.'s key mismatch attack, we propose an efficient strategy which with a probability of 96.88% succeeds in recovering all the coefficients belonging to $[-8, 8]$ in the secret key. Recall that in NewHope four coefficients are encoded at a time. Through in-depth analysis of the properties of the Decode function, we notice that it can help us find the sum of the 4 coefficients. Since in a targeted quadruplet, there is a 96.88% probability that only one coefficient belongs to $\{-8, -7, 5, 6, 7, 8\}$, and the other 3 coefficients belong to $[-6, 4]$. The key idea of our strategy is that we can first recover the 3 coefficients using our improved method, then recover the remaining coefficient since the sum of the 4 coefficients is known. Experiments show that our proposed method is very efficient, which completes the attack in about 137.56 ms using the NewHope parameters.

2 The Ring-LWE Problem and NewHope KEM

Set $\mathbb{Z}_q$ the ring with all coefficients 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 $f(x) = a_0 + a_1 x + \cdots + a_{n-1}x^{n-1} \in \mathbb{R}_q$, each coefficient $a_i \in \mathbb{Z}_q$ $(0 \leq i \leq n-1)$ 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 $\mathbf{c}[i]$ $(0 \leq i \leq n-1)$ 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 KEM 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 $i = 0, 1, \ldots, 255$,*

$$\mathbf{k}[i] = \mathbf{k}[i+256] = \mathbf{k}[i+512] = \mathbf{k}[i+768] = \left\lfloor \frac{q}{2} \right\rfloor \nu'_B[i]. \tag{1}$$

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., $\nu'_A =$ Decode($\mathbf{k}'$) and*

$$\nu'_A[i] = \begin{cases} 1 & \text{if } m < q, \\ 0 & \text{otherwise}, \end{cases} \tag{2}$$

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 $\bar{\mathbf{c}} =$ Compress($\mathbf{c}$) and for $i = 0, 1, \ldots, 1023$,*

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

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

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

In Table 1, we describe the details of the NewHope KEM. Since in NewHope, the number-theoretic transform (NTT) is used to speed up the polynomial multiplication, which has nothing to do with security. To simplify the security analysis of NewHope, in Table 1 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:

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

Table 1. The NewHope KEM

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)		

(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).

3 The Proposed Key Mismatch Attack

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

Algorithm 1: Oracle

```
Input: P_B, c̄, S_{k_B}
Output: 1 or 0
c' = Decompress(c̄);
k' = c' − P_B s_A ;
ν'_A ← Decode(k');
S_{k_A} ← SHA3-256(ν'_A);
if S_{k_A} = S_{k_B} then
    Return 1;
else
    Return 0;
```

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 1. 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 KEM is that, 4 coefficients 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 encoded and decoded together, which makes it hard to decide each of them.

3.1 Bauer et al.'s method

In this subsection, we briefly introduce Bauer et al.'s method in [5]. 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. Set $\mathrm{S}_1 = \{-8, -7, \ldots, -1, 0, 1, \ldots, 7, 8\}$ and $\mathrm{S}_2 = \{-6, -5, \ldots, 2, 3, 4\}$. Their basic idea is to recover all the coefficients in S_2. First of all, the adversary $\mathcal{A}$ directly chooses $\nu'_B = (1, 0, \cdots, 0)$. If $\mathcal{A}$ wants to recover the quadruplet $(\mathbf{s}_A[i], \mathbf{s}_A[i+256], \mathbf{s}_A[i+512], \mathbf{s}_A[i+768])$, he will set his public key $\mathbf{P}_B = \lfloor \frac{q}{8} \rfloor x^{-i}$ and $\bar{\mathbf{c}} = \sum_{j=0}^{3} ((l_j + 4) \bmod 8) x^{256j}$, here each l_j ranges 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 coefficients of the private key if he can find outputs in a form like $1, \cdots, 1,\ 0, \cdots, 0,\ 1, \cdots, 1$ as (l_0, l_1, l_2, l_3) changes. Here this kind of form is called a favorable case.

Specifically, if $\mathcal{A}$ wants to recover $\mathbf{s}_A[i]$ in $\mathbf{s}_A$, he can first set each l_j $(j = 1, 2, 3)$ 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_j $(j = 1, 2, 3)$ the resulted output is another bit b_1. Repeating the above processes until l_0 becomes 3, there will be 8 bits b_j $(j = 0, 1, \cdots, 7)$. The above processes will be repeated with different l_j $(j = 1, 2, 3)$ until $\mathcal{A}$ finds a favorable case. Then the adversary $\mathcal{A}$ can recover the coefficients in S_2 by recording the positions where 1 changes to 0 and 0 goes to 1 in the favorable case. $\mathcal{A}$ will repeat the above processes until he recovers all the coefficients of $\mathbf{s}_A$ that belongs to S_2.

We have generated 1,000 secret keys and repeated the experiments using Bauer et al.'s method. Unfortunately, even for coefficients in [-6,4] we cannot recover at least 262 of them in each secret key with 1024 coefficients. What makes the situation worse is that in some cases the recovered coefficients are wrong and we cannot detect these cases using Bauer et al.'s method. Another problem is that, for the coefficients outside [-6,4], they suggested an exhaustive search. But for each secret key on average there are 10 coefficients that need to be exhaustively searched, and each of them has 6 possibilities. This makes Bauer et al.'s method highly inefficient.

Algorithm 2: Find-τ

```
Input: b
Output: τ
set τ = NULL, τ1 = NULL, τ2 = NULL ;
if b[0] = 1 then
    for i := 1 to 6 do
        if (b[i − 1] = 1) and (b[i] = 0) then
            τ1 = i − 4;
        if (b[i] = 0) and (b[i + 1] = 1) then
            τ2 = i − 4;
    end
else if b[0] = 0 then
    for i := 1 to 6 do
        if (b[i − 1] = 0) and (b[i] = 1) then
            τ1 = i − 4;
        if (b[i] = 1) and (b[i + 1] = 0) then
            τ2 = i − 4;
    end
τ = τ1 + τ2;
if τ > 0  and  b[0] = 1 then
    τ = τ − 8;
else if τ <= 0  and  b[0] = 1 then
    τ = τ + 8;
if τ is odd and τ1 ≠ NULL and τ2 ≠ NULL then
    odd_number = odd_number +1;
    odd_τ = τ;
else if τ is even and τ1 ≠ NULL and τ2 ≠ NULL then
    even_number = even_number +1;
    even_τ = τ;
else
    τ = NULL;
end
Return τ;
```

Table 2. The relationship between τ and $\mathbf{s}_A$[i]∈[-6,4]

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

Algorithm 3: Find-s-in-S_2

Output: $\mathbf{s}$ (the coefficients 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_number = 0, even_number = 0, count = 0;
    while count < 50 **do**
      $(l_0, l_1, l_2, l_3) \leftarrow [-4, 3]^4$; $b[8] \leftarrow 0$;
      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-$\tau(b)$;
      if $t \neq$ NULL **then**
        count = count +1
    end
    if odd_number >= even_number **then**
      temp$_s$= $\lfloor$(odd_τ)/2$\rfloor * 2 + 1$;
      test(temp$_s$);
    else if even_number > odd_number **then**
      temp$_s$ = even_τ;
      test(temp$_s$);
    end
  end
end
$\mathbf{s}[k + j * 256]$ = temp$_s$;
Return s

3.2 Our Improved Method

In this subsection, we propose an improved method to recover the coefficients in S_2.

First in Algorithm 2 we propose how to calculate τ_1 and τ_2, which paly an important role in our following recovery. We can also determine whether $\mathbf{b} = (b_0, \ldots, b_7)$ is a favorable case or not through the calculated τ_1 and τ_2. In Bauer et al.'s method, there is only one kind of favorable case in the form $1, \cdots, 1,\ 0, \cdots, 0,\ 1, \cdots, 1$. In this case, we use Bauer et al.'s method to calculate τ_1 and τ_2, which records the positions where 1 goes to 0 and 0 changes to 1, respectively. Through experiments, we find that there is another favorable case in the form $0,\ 0, \cdots, 0, 1, \cdots, 1,\ 0, \cdots, 0$. In this case, we use τ_1 and τ_2 to record the positions where 0 goes to 1 and 1 changes to 0, respectively. The precise definition of τ_1 and τ_2 can be found in Algorithm 2. If the output of Algorithm 2 is NULL, there is no favorable case, otherwise we can find a favorable case.

In Table 2, we show the relationship between $\tau = \tau_1 + \tau_2$ and $\mathbf{s}_A[\mathrm{i}]$ in S_2. In Bauer et al.'s method, they assume that the value of τ is either even or odd. But our experiments show that there exist both even and odd τs, and this is also the reason why Bauer et al.'s method cannot recover the coefficients completely.

Then, in Algorithm 3 we propose how to recover all the coefficients in S_2. The main idea is that we repeat the processes in Algorithm 2 until we find enough favorable cases. Of course if we can find more favorable cases, then the recovery of coefficients can be more exact, but this needs more time and more queries. To take a balance, in Algorithm 3, we try to get 50 favorable cases. Next, we can use the data collected in these 50 favorable cases to recover the coefficients in S_2.

We use odd-number and even-number to record the times the odd and even τ occurs, and the corresponding values of τ are stored in odd_τ and even_τ, respectively. We can see from Table 2 that if the coefficient $\mathbf{s}_A[i]$ is odd, then odd-number is larger than the even-number, and vice versa. Therefore, if even-number is larger than the odd-number, the corresponding coefficient $\mathbf{s}_A[\mathrm{i}]$ is calculated as $\mathbf{s}_A[i] = \text{even\_}\tau$. Otherwise, we calculate it as $\mathbf{s}_A[i] = \lfloor(\text{odd\_}\tau)/2\rfloor * 2 + 1$.

Table 3. $s_A[i]$ and the possible τs

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

Since we only get 50 favorable cases, there may exist the case one coefficient is recovered to be another coefficient. For example when $\mathbf{s}_A[i] = 3$, the corresponding odd-number and even-number are close. So if the recovered coefficient is 3, we need to eliminate the case that we recover 4 to be 3. In order to solve this problem, we generate 1000 secret keys, and record the possible values of τ for each coefficient between −6 and 4. As shown in Table 3, the corresponding τs can help us decide which one is correct. For example, when $\mathbf{s}_A[i] = 4$, the possible values of τ are 3 and 4, but if $\mathbf{s}_A[i] = 3$, the corresponding values of τ are 2 and 3. Since 3 is odd, the recovered 3 must be calculated by $\lfloor (\text{odd\_}\tau)/2 \rfloor * 2 + 1$. We can know that odd_$\tau = 3$, and odd-number must be bigger than the even-number. We can see that in the two cases the odd_τs are the same, but the even_τs are different, so we can distinguish them according to the value of even_τ. Specifically, if even_$\tau = 2$ we can determine that the recovered coefficient is correct. But if even_$\tau = 4$, we make sure that the recovered coefficient is wrong, which should be 4. Similarly we can correct most of the errors using this method, and finally with a high probability we can recover all the coefficients in S_2.

3.3 The Complete Attack

Table 4. The distribution of the coefficients in a quadruplet

$S_1 = \{-8, -7, \ldots, -1, 0, 1, \ldots, 7, 8\}$
$S_2 = \{-6, -5, \ldots, 2, 3, 4\}$ S_1-$S_2 = \{-8, -7, 5, 6, 7, 8\}$

<table>
<tr><td colspan="3">4 coefficients in S_1</td></tr>
<tr><td colspan="3">100%</td></tr>
<tr><td>4 coefficients in S_2</td><td colspan="2">Others</td></tr>
<tr><td rowspan="3">95.84%</td><td colspan="2">4.16%</td></tr>
<tr><td>3 coefficients in S_2
1 coefficient in S_1-S_2</td><td>2 coefficients in S_2
2 or more coefficients in S_1-S_2</td></tr>
<tr><td>98.50%</td><td>1.50%</td></tr>
</table>

After recovering all the coefficients that belongs to S_2, the remaining problem is how to recover the coefficients in $S_1 - S_2$? In table 4, we have analyzed and listed the distribution of the coefficients 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 coefficients are in set S_1, and the probability that all the coefficients of the quadruplet are in S_2 is 95.84%. In the remaining 4.16% quadruplets, there is at least 1 coefficient that belongs to $S_1 - S_2$. Our key observation is that, with 98.50% probability there is only 1 coefficient that belongs to $S_1 - S_2$, while the other 3 coefficients are in S_2 in the remaining quadruplets.

Without loss of generality, we assume that $\mathbf{s}_A[i+256]$, $\mathbf{s}_A[i+512]$ and $\mathbf{s}_A[i+768]$ are in S_2 and $\mathbf{s}_A[i]$ is in $S_1 - S_2$. Using our improved method in Algorithms 2 and 3, we can recover $\mathbf{s}_A[i+256]$, $\mathbf{s}_A[i+512]$ and $\mathbf{s}_A[i+768]$. Then, our strategy is that if we can compute the sum of these four coefficients, we can recover $\mathbf{s}_A[i]$ by eliminating $\mathbf{s}_A[i+256]$, $\mathbf{s}_A[i+512]$ and $\mathbf{s}_A[i+768]$ from the sum. In the following, we describe 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 $\mathbf{0}$ in $\mathbb{R}_q$, and an $\mathbf{e}_B$ of which coefficients are all zero, except that $\mathbf{e}_B[512] = h_1$. Here h_1 increases from 0 to $q-1$. Instead of randomly selecting ν_B to calculate ν'_B, the adversary $\mathcal{A}$ will directly set all coefficients of ν'_B as 0 except that $\nu'_B[i] = 1$.

As $\mathcal{A}$ sets $\mathbf{s}_B = \mathbf{0}$, correspondingly now the public key is $\mathbf{P}_B = \mathbf{a}\mathbf{s}_B + \mathbf{e}_B = \mathbf{e}_B$. According to the definition of the Encode function, we have

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

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

Then, since $\bar{\mathbf{c}}[i] = \lfloor (\mathbf{c}[i] \cdot q)/8 \rceil \bmod 8$, if $\mathbf{c}[i] = \lfloor \frac{q}{2} \rfloor$, then $\bar{\mathbf{c}}[i] = 4$, according to the above analysis and the definition of the Compress function

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

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

$$\mathbf{c}' = Decompress(\bar{\mathbf{c}}) = \left\lfloor \frac{q}{2} \right\rceil x^i + \left\lfloor \frac{q}{2} \right\rceil x^{i+256} + \left\lfloor \frac{q}{2} \right\rceil x^{i+512} + \left\lfloor \frac{q}{2} \right\rceil x^{i+768}, \tag{5}$$

as well as

$$\mathbf{k}' = \mathbf{c}' - \mathbf{P}_B \mathbf{s}_A = \mathbf{c}' - \mathbf{e}_B \mathbf{s}_A. \tag{6}$$

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

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

Algorithm 4: Find-m_1

Input: i
Output: m_1
for $h_1 := 0$ *to* $q-1$ **do**
  $\mathbf{e}_B = \mathbf{0}$, set $\mathbf{e}_B[512] = h_1$; $\mathbf{P}_B = \mathbf{e}_B$;
  $\nu'_B = \mathbf{0}$, set $\nu'_B[i] = 1$;
  $\mathbf{k} \leftarrow$Encode(ν'_B); $\bar{\mathbf{c}} =$ Compress$(\mathbf{k})$;
  $S_{k_B} \leftarrow$ SHA3-256(ν'_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
Return m_1

The adversary $\mathcal{A}$ chooses the parameters as described above, and the complete attack consists of four steps.

Algorithm 5: Full-recovery

Output: s' (All the coefficients in S_1)
$\mathbf{s}' \leftarrow$ Find-s-in-S$_2$();
for $i := 0$ *to* 255 **do**
  for $j := 0$ *to* 3 **do**
    if $\mathbf{s}'[i + 256 * j] < -6$ or $\mathbf{s}'[i + 256 * j] > 4$ **then**
      break;
  end
  $m_1 =$ Find-m_1(i);
  for $k := 0$ *to* 3 **do**
    if $k \neq j$ **then**
      $m_1 = m_1$ - $|\mathbf{s}'[i + 256 * k]|$;
  end
  if $\mathbf{s}'[i + 256 * j] < 0$ **then**
    $\mathbf{s}'[i + 256 * j] = -m_1$;
  else
    $\mathbf{s}'[i + 256 * j] = m_1$;
end
Return $\mathbf{s}'$

Step 1: In this step, the adversary $\mathcal{A}$ uses our improved method in algorithm 2 to recover all the coefficients belonging to S_2.
Step 2: In this step, the 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 coefficients of $\mathbf{e}_B$ as $\mathbf{0}$, except $\mathbf{e}_B[512] = h_1$. From equations 5, 6 and $\lfloor \frac{q}{2} \rceil = 6145$, we have

$$\begin{aligned}
\mathbf{k}' &= \mathbf{c}' - \mathbf{e}_B \mathbf{s}_A \\
&= [6145 - (-\mathbf{s}_A[i + 512]\mathbf{e}_B[512])]x^i + [6145 - (-\mathbf{s}_A[i + 768]\mathbf{e}_B[512])]x^{i+256} \\
&\quad + (6145 - \mathbf{s}_A[i]\mathbf{e}_B[512])x^{i+512} + (6145 - \mathbf{s}_A[i + 256]\mathbf{e}_B[512])x^{i+768} \\
&= [6145 - (-\mathbf{s}_A[i + 512]h_1)]x^i + [6145 - (-\mathbf{s}_A[i + 768]h_1)]x^{i+256} \\
&\quad + (6145 - \mathbf{s}_A[i]h_1)x^{i+512} + (6145 - \mathbf{s}_A[i + 256]h_1)x^{i+768}.
\end{aligned}$$

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

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

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. As h_1 increases, correspondingly m also increases until it reach the point that $m \geq q$. Now 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 $\mathcal{A}$ tries to determine the sign of $\mathbf{s}_A[i]$. In Step 1, if $\mathbf{s}_A[i]$ is outside $[-6, 4]$, then $\mathbf{s}_A[i]$ will be recovered to an incorrect value, but its sign is correct. So, we can directly determine the sign of $\mathbf{s}_A[i]$ according to this. There are only two special cases when $\mathbf{s}_A[i] = 8$ or $\mathbf{s}_A[i] = -8$ then the correct sign of $\mathbf{s}_A[i]$ is opposite to that recovered in Step 1.

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 should follow the centered binomial distribution.

4 Experiments

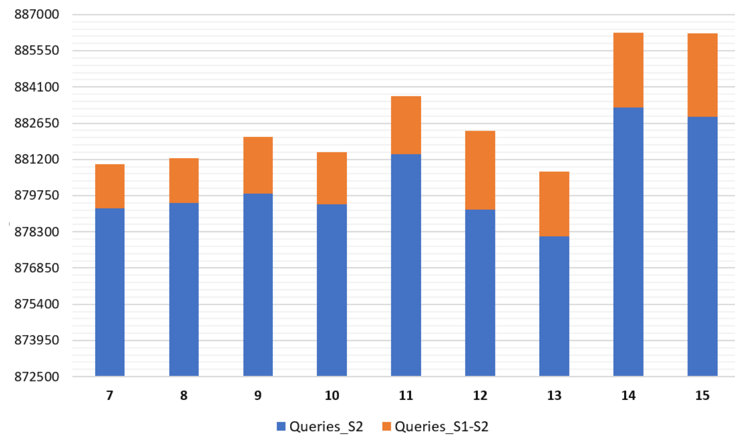
In this section, we show the efficiency of our proposed attack. All our implementations are done on a MacBook Air, which is equipped with a Intel Core i7 processor at 2.7 GHz and an 8 GB RAM.

First of all, we want to show the advantage of our proposed algorithm 2 in recovering 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 [2] to generate 1000 secret keys. Then we implement Bauer et al.'s method to recover the coefficients belonging to S_2. Unfortunately, using Bauer et al.'s method we cannot even recover all the coefficients belonging to S_2 in every secret key. In other words, in every secret key with 1024 coefficients, there are at least 262 coefficients in S_2 that cannot be recovered.

On the other side, when we use our method as shown in algorithm 2, in 992 keys we can recover all the coefficients belonging to S_2, and in the remaining 8 keys there are at most 2 coefficients that cannot be recovered. Then, by using algorithms 2 and 4 together we can recover all the coefficients belong to $\mathrm{S}_1 = \{-8, -7, \ldots, 6, 7, 8\}$. In our experiment, we also generate $1,000$ secret keys. The result is, in 969 keys we recover all the coefficients in S_1. Thus the probability of successfully recovering the whole secret key is 96.9%.

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 $m_1 = |\mathbf{s}_A[i]| + |\mathbf{s}_A[i+256]| + |\mathbf{s}_A[i+512]| + |\mathbf{s}_A[i+768]|$, and get the absolute value of the coefficient that belonging to S_1-S_2. For example, if we do not know $\mathbf{s}_A[i]$, we can have $|\mathbf{s}_A[i]| = m_1 - |\mathbf{s}_A[i+256]| - |\mathbf{s}_A[i+512]| - |\mathbf{s}_A[i+768]|$. Finally, we will follow the Steps 3 to decide the sign of $\mathbf{s}_A[i]$ and verify whether the recovered is correct using the method in Step 4.

Fig. 1. Comparison of queries between different T



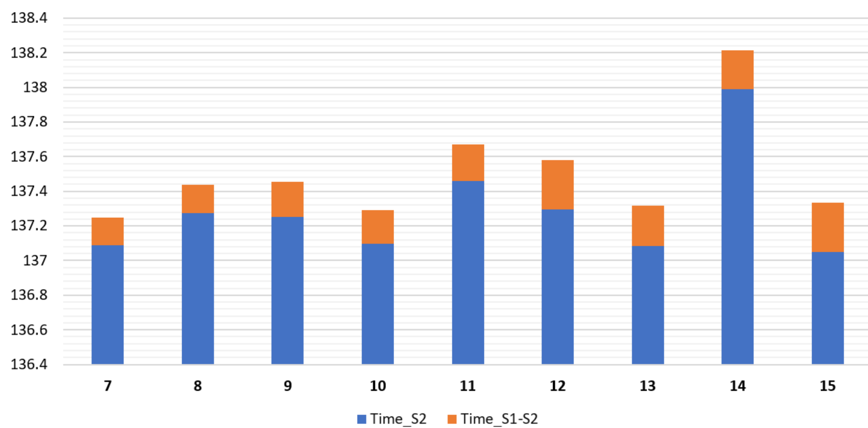
From the above experiments, we also find that in each secret key with 1024 coefficients, the most possible number of coefficients that belongs S_1-S_2 is between 7 and 15. In the following, we set T the number of coefficients in S_1-S_2.

Table 5. Queries needed in recovering coefficients in S_2 and S_1-S_2

T	7	8	9	10	11	12	13	14	15
Queries_S_2	879,246	879,458	879,829	879,396	881,418	879,181	878,118	883,281	882,896
Queries_S_1-S_2	1,764	1,795	2,269	2,094	2,319	3,167	2,583	2,988	3,346
Total	881,010	881,254	882,098	881,490	883,738	882,348	880,701	886,269	886,242

In Figure 1, we report the average number of queries for recovering coefficients in S_2 and S_1-S_2 when T ranges from 7 to 15. The specific queries is given in Table 5. We can see that the number of queries used in recovering coefficients in S_2 is almost 365 times more than the number of queries required to recover the coefficients in S_1-S_2. The reason is when recovering a coefficient in S_2, we need to find 50 favorable cases, which need a large number of queries. We can also observe that as T increases from 7 to 15, the average number of queries for recovering coefficients in S_2 is between $878,118$ and $883,281$. It does not increase a lot as T increases. This is because when we recover coefficients in S_2, we need to randomly generate (l_0, l_1, l_2, l_3) to get the favorable cases. Since the number of favorable cases is fixed at 50, the number of queries is almost the same. On average the number of needed queries is $879,725$. On the other side, as T increases, the number of queries for recovering coefficients in S_1-S_2 will increase. When we recover a coefficient in S_1-S_2, we need to use the Algorithm

Fig. 2. The average time (ms) between different T



4. Larger T means that there are more coefficients that cannot be recovered by Algorithm 2, and more queries are needed.

Table 6. Average Time (ms) needed in recovering coefficients in S_2 and S_1-S_2

T	7	8	9	10	11	12	13	14	15
Time_S_2	137.08	137.27	137.25	137.09	137.45	137.29	137.08	137.99	137.04
Time_S_1-S_2	0.16	0.16	0.20	0.19	0.20	0.28	0.23	0.22	0.28
Time	137.24	137.43	137.45	137.29	137.66	137.58	137.31	138.21	137.33

When T increases from 7 to 15, the average time for recovering coefficients in S_2 and S_1-S_2 is shown in Figure 2, and the specific data is given in Table 6. We can see that the time required to recover coefficients in S_2 occupies 99% of the total time, since a lot of time is spent on looking for the 50 favorable cases when we recover the coefficient in S_2. We can also observe that as T increases, the average time for recovering coefficients in S_2 is between 136 ms and 138 ms, which is almost the same due our above analysis.

Compared with using an exhaustive research to find coefficients in S_1-S_2, our proposed method is much more efficient. In the exhaustive search experiment the best strategy is to search each element in the order $\{5, 6, 7, -7, 8, -8\}$. Then, we can verify whether the recovered private key is correct by calculating the distribution of $\mathbf{e}'_A = \mathbf{P}_A - \mathbf{a}\mathbf{s}_A$. If we get a correct private key, then the distribution of $\mathbf{e}'_A$ is the same as that of $\mathbf{e}_A$, which follows the centered binomial distribution. As an example, when $T = 12$, if we use an exhaustive search the required time is about 1.91 hours. From this perspective, our proposed attack is very efficient.

5 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 and efficient 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 transformation. 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.

Bibliography

[1] Alagic, G., Alagic, G., Alperin-Sheriff, J., Apon, D., Cooper, D., Dang, Q., Liu, Y.K., Miller, C., Moody, D., Peralta, R., et al.: Status Report on the First Round of the NIST Post-Quantum Cryptography Standardization Process. US Department of Commerce, National Institute of Standards and Technology (2019), https://nvlpubs.nist.gov/nistpubs/ir/2019/NIST.IR.8240.pdf. Last accessed 26 February 2019

[2] Alkim, E., Avanzi, R., Bos, J., Ducas, L., de la Piedra, A., Pöppelmann, T., Schwabe, P., Stebila, D.: Newhope: Algorithm specification and supporting documentation. submission to the nist post-quantum cryptography standardization project, 2017, https://newhopecrypto.org/data/NewHope_2018_12_02.pdf. Last accessed 27 February 2019

[3] Alkim, E., Ducas, L., Pöppelmann, T., Schwabe, P.: Newhope without reconciliation. IACR Cryptology ePrint Archive **2016**, 1157 (2016), https://www.cryptojedi.org/papers/newhopesimple-20161217.pdf. Last accessed 17 February 2019

[4] Alkim, E., Ducas, L., Pöppelmann, T., Schwabe, P.: Post-quantum key exchangea new hope. In: 25th USENIX Security Symposium (USENIX Security 16). pp. 327–343 (2016)

[5] Bauer, A., Gilbert, H., Renault, G., Rossi, M.: Assessment of the key-reuse resilience of newhope. In: Cryptographers Track at the RSA Conference. pp. 272–292. Springer (2019)

[6] Bos, J.W., Costello, C., Naehrig, M., Stebila, D.: Post-quantum key exchange for the tls protocol from the ring learning with errors problem. In: 2015 IEEE Symposium on Security and Privacy. pp. 553–570. IEEE (2015)

[7] Ding, J., Alsayigh, S., Saraswathy, R., Fluhrer, S., Lin, X.: Leakage of signal function with reused keys in rlwe key exchange. In: 2017 IEEE International Conference on Communications (ICC). pp. 1–6. IEEE (2017)

[8] Ding, J., Cheng, C., Qin, Y.: A simple key reuse attack on lwe and ring lwe encryption schemes as key encapsulation mechanisms (kems). Cryptology ePrint Archive, Report 2019/271 (2019), https://eprint.iacr.org/2019/271. Last accessed 21 April 2019

[9] Ding, J., Fluhrer, S., Rv, S.: Complete attack on rlwe key exchange with reused keys, without signal leakage. In: Australasian Conference on Information Security and Privacy. pp. 467–486. Springer (2018)

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

[11] Fluhrer, S.R.: Cryptanalysis of ring-lwe based key exchange with key share reuse. IACR Cryptology ePrint Archive **2016**, 85 (2016), http://eprint.iacr.org/2016/085. Last accessed 18 February 2019

[12] Gao, X., Ding, J., Li, L., Liu, J.: Practical randomized rlwe-based key exchange against signal leakage attack. IEEE Transactions on Computers **67**(11), 1584–1593 (2018)
[13] Lindner, R., Peikert, C.: Better key sizes (and attacks) for lwe-based encryption. In: Cryptographers Track at the RSA Conference. pp. 319–339. Springer (2011)
[14] Liu, C., Zheng, Z., Zou, G.: Key reuse attack on newhope key exchange protocol. In: International Conference on Information Security and Cryptology. pp. 163–176. Springer (2018)
[15] Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. In: Cryptology - EUROCRYPT. pp. 1–23 (2010)
[16] Peikert, C.: Lattice cryptography for the internet. In: International workshop on post-quantum cryptography. pp. 197–219 (2014)
[17] Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. Journal of the ACM (JACM) **56**(6), 34:1–40 (2009)
[18] Rescorla, E.: The transport layer security (tls) protocol version 1.3. Tech. rep. (2018), `http://www.rfc-editor.org/info/rfc8446`. Last accessed 26 February 2019
[19] Zhang, J., Zhang, Z., Ding, J., Snook, M., Dagdelen, Ö.: Authenticated key exchange from ideal lattices. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 719–751. Springer (2015)