Small Leaks Sink a Great Ship: An Evaluation of Key Reuse Resilience of PQC Third Round Finalist NTRU-HRSS

Xiaohan Zhang, Chi Cheng, Yue Qin, and Ruoyu Ding

China University of Geosciences, Wuhan, 430074, China
{chengchi}@cug.edu.cn

Abstract. NTRU is regarded as an appealing finalist due to its long history against all known attacks and relatively high efficiency. In the third round of NIST competition, the submitted NTRU cryptosystem is the merger of NTRU-HPS and NTRU-HRSS. In 2019, Ding et al. have analyzed the case when the public key is reused for the original NTRU scheme. However, NTRU-HRSS selects coefficients in an arbitrary way, instead of fixed-weight sample spaces in the original NTRU and NTRU-HPS. Therefore, their method cannot be applied to NTRU-HRSS. To address this problem, we propose a full key mismatch attack on NTRU-HRSS. Firstly, we find a longest chain which helps us in recovering the following coefficients. Next, the most influential interference factors are eliminated by increasing the weight of targeted coefficients. In this step, we adaptively select the weights according to the feedbacks of the oracle to avoid errors. Finally, experiments show that we succeed in recovering all coefficients of the secret key in NTRU-HRSS with a success rate of 93.6%. Furthermore, we illustrate the trade-off among the success rate, average number of queries, and average time. Particularly, we show that when the success rate is 93.6%, it has the minimum number of queries at the same time.

Keywords: Post-quantum cryptography · Lattice based cryptography · NTRU · Public key reuse · Key mismatch attack.

1 Introduction

Under the threat of rapid development of quantum computers [15], the current public key algorithms which base their security on number theoretic problems will no longer be safe. For example, the RSA and DH algorithms relying on integer factorization and discrete logarithm problems could be broken as shown in Shor's pioneer paper [27]. To thwart attacks from quantum computers, the cryptography community has prompted to look for a new cryptosystem, which is called post-quantum cryptography (PQC) [7].

The National Institute of Standards and Technology (NIST) has started a project to select and evaluate PQC algorithms against both classical and quantum computers ever since 2016 [21]. The second round NIST PQC standardization process has been completed on July, 2020 [1]. On the finalists, the lattice-based public key encryption (PKE) or key encapsulation mechanism (KEM) algorithms draw significant attention, since there are 3 out of 4 candidates, KYBER [3], SABER [12], and NTRU [6]. NIST aims to standardize at most one of them when the third round ends. Among them, NTRU has been regarded as a compelling one due to its long history against known attacks and relatively high efficiency [1].

Currently, it can be noted that in the widely adopted Internet standards, the key reuse mode is commonly used. For example, in the released Transport Layer Security (TLS) 1.3 [26], the key pair in the pre-shared key mode is reused. However, key reuse may cause attacks in lattice-based key exchange [18]. In general, key reuse attacks can be divided into signal leakage attacks [8, 13] and key mismatch attacks [10]. The main reason for the signal leakage attack is that if the key is reused, the relevant signal information used for key recovery leaks the relevant information of the secret key. Meanwhile, the key mismatch attack is to query the two communication parties whether the shared keys match or not, analyzing the feedbacks to recover the secret key.

Nowadays, a series of key mismatch attacks have been successively proposed. In [10], a key mismatch attack was proposed by Ding et al. on the one pass case of [11], in which no information leaked by the signal function was used. Later, Bauer et al. [5] proposed a key mismatch attack against a PQC second round candidate NewHope [2]. In [23], Qin et al. showed that the recovery in Bauer et al. was incomplete, and then proposed an effective key recovery scheme. Okada et al. followed closely, and in [22] they further improved Qin et al.'s method. Later, the work of [14] gave a key mismatch attack on another PQC second round candidate LAC. In [24], a key mismatch attack is proposed against Kyber. Băetu et al. proposed a classical key mismatch attack as well as a quantum key recovery [4]. In [25] Qin et al. gave a systematic approach to find bounds of key mismatch attacks against all the NIST candidate KEMs.

Unlike the protocols based on Ring Learning with Errors (RLWE) problem [20] or Modular Learning with Errors (MLWE) problem [19], the NTRU cryptosystem submitted to the NIST [6] is operated in a different polynomial ring which modulo $x^n - 1$. Therefore, these attacks proposed by Ding et al. [10], Qin et al. [23] or Okada et al. [22] cannot be directly applied to the NTRU cryptosystem [6]. The main reason is that NTRU lacks the structure of affine transformation, making it difficult to recover the secret key using the previous method.

In 2019, Ding et al. [9] proposed a key mismatch attack on the original NTRU scheme [16]. As we know, the coefficients of the secret key in NTRU belong to $\{-1, 0, 1\}$ and there is a longest chain of consequent coefcients that consists of either consecutive 1s or consecutive -1s of a secret key at least. With the longest chain, they proposed an elegant method, which is claimed to recover all the coefficients in the secret key.

However, their method cannot be directly applied to the current whole NTRU cryptosystem. The NTRU cryptosystem submitted to the third round of NIST competition [6] is a merger of NTRU-HPS [16] and NTRU-HRSS [17]. One of the most important differences is that they compute on cyclotomic polynomials $\mathbf{\Phi}_1 = x - 1$ and $\mathbf{\Phi}_n = x^{n-1} + x^{n-2} + \cdots + 1$, instead of $x^n - 1$ in the original NTRU. Another important difference is that NTRU-HPS is similar to the original NTRU scheme, which still selects coefficients from fixed-weight sample spaces. While NTRU-HRSS selects coefficients in an arbitrary way. Therefore, Ding et al.'s method still works for NTRU-HPS, but cannot be applied to NTRU-HRSS.

Contributions. In this paper, we propose a complete key mismatch attack on NTRU-HRSS. The main contributions of this paper include:

1. We investigate the resilience of the NTRU-HRSS KEM under a misuse case: we assume that the same key is reused for multiple key establishments and an attacker can use a key mismatch oracle.
2. Unlike the direct recovery of secret key in Ding et al's method, we recover the product of a secret key and a cyclotomic polynomial, which is also the reason why Ding et al.'s method cannot be applied directly. Specifically, we first find a longest chain. After that, we increase the weight of targeted coefficients to eliminate the most influence of disturbances. Considering that the introduction of weight may lead to the errors in the recovery, we adaptively select the weights according to the feedbacks of the oracle to avoid errors.
3. As verified by the experiments, our improved method can recover all the coefficients in the secret key with a probability of 93.6%. Moreover, by evaluating the trade-off between the success rate and average number of queries, we can achieve minimum number of queries with a success rate of 93.6% at the same time.

Organization of this paper. In Section 2, we introduce the basic notions and describe the NTRU-HRSS KEM. In Section 3, we propose a complete key mismatch attack on NTRU-HRSS KEM. We give the experimental results and illustrate the trade-off among the success rate, average number of queries, average time in Section 4. Finally, the conclusion is given in Section 5.

2 Preliminaries

2.1 Notations

In NTRU, n, p and q are coprime integers. We denote the i-th cyclotomic polynomial by $\mathbf{\Phi}_i$. Specifically, $\mathbf{\Phi}_n = x^{n-1} + x^{n-2} + \cdots + 1$, $\mathbf{\Phi}_1 = x - 1$, and $\mathbf{\Phi}_1\mathbf{\Phi}_n = x^n - 1$. $\mathbb{Z}_q$ represents the integer ring modulo q. Let $\mathbb{Z}_q[x]$ represent a polynomial ring, in which all polynomial coefficients are selected from $\mathbb{Z}_q$.

We further define the polynomial rings $\mathcal{R}_q = \mathbb{Z}_q[x]/(x^n - 1)$, $\mathcal{R}'_q = \mathbb{Z}_q[x]/(\mathbf{\Phi}_n)$, and $\mathcal{R}'_p = \mathbb{Z}_p[x]/(\mathbf{\Phi}_n)$. Here, all polynomials are in bold. A polynomial $\mathbf{P}$ in $\mathcal{R}_q$ is of degree at most $n - 1$ with coefficients in $\mathbb{Z}_q$. If a polynomial $\mathbf{P}'$ belongs to $\mathcal{R}'_q$, it is a polynomial of degree of $n - 2$ with coefficients $\mathbf{P}'[i]$ belonging to the set $\mathbb{Z}_q$, where $\mathbf{P}'[i]$ $(0 \leq i \leq n - 2)$ represents the ith coefficient of the polynomial $\mathbf{P}'$. $\mathbf{P}$ and $\mathbf{P}'$ can also be represented as a vector with n and $n - 1$

coordinates, respectively. For a real number x, the operation $\lceil x \rceil$ represents the smallest integer not less than x.

NTRU was originally presented as a probabilistic public key encryption (PPKE) scheme in [16]. In the third round NTRU submission, PPKE is replaced with the deterministic public key encryption (DPKE) scheme and all aspects of the designs are unified except for the use of fixed-weight sampling. Specifically, the probability of occurrence of -1, 0 and 1 are $\frac{85}{256}$, $\frac{86}{256}$, $\frac{85}{256}$, and we can easily sample them from $(\sum_{i=0}^{7} 2^i b_i) \bmod 3$. Here b_i is randomly selected from $\{0, 1\}$.

2.2 NTRU-HRSS KEM

Table 1. The CPA version of the NTRU-HRSS KEM

Alice		**Bob**
$\mathbf{f} \in \mathbf{F}_+$, $\mathbf{g} \in \mathbf{F}_+$		
$\mathbf{f}_q \leftarrow \mathbf{f}^{-1} \in \mathcal{R}'_q$		
$\mathbf{f}_p \leftarrow \mathbf{f}^{-1} \in \mathcal{R}'_p$		
$\mathbf{h} \leftarrow p\mathbf{g}\mathbf{\Phi}_1\mathbf{f}_q \in \mathcal{R}_q$		
$\mathbf{h}_q \leftarrow \mathbf{h}^{-1} \in \mathcal{R}'_q$	$\xrightarrow{\mathbf{h}}$	$\mathbf{r} \in \mathcal{R}'_p$, $\mathbf{m} \in \mathcal{R}'_p$
$\mathbf{a} \leftarrow \mathbf{c}\mathbf{f} \in \mathcal{R}_q$	$\xleftarrow{\mathbf{c}}$	$\mathbf{c} \leftarrow \mathbf{r}\mathbf{h} + \mathrm{Lift}(\mathbf{m}) \in \mathcal{R}_q$
$\mathbf{m}' \leftarrow \mathbf{a}\mathbf{f}_p \in \mathcal{R}'_p$		
$\mathbf{r}' \leftarrow (\mathbf{c} - \mathrm{Lift}(\mathbf{m}'))\mathbf{h}_q \in \mathcal{R}'_q$		
if $(\mathbf{r}', \mathbf{m}') \in \mathcal{R}'_p \times \mathcal{R}'_p$		
Return$(\mathbf{r}', \mathbf{m}', 0)$;		
else		
Return$(0, 0, 1)$;		

The most important definitions in the NTRU-HRSS KEM are shown as below.

Definition 1. *The Lift function Lift:* $\mathcal{R}'_p \to \mathcal{R}_q$ *is defined as* $\boldsymbol{P'}$=Lift($\boldsymbol{m}$),

$$\boldsymbol{P'} = \boldsymbol{m}(\mathbf{\Phi}_1^{-1} \bmod (p, \mathbf{\Phi}_n))\mathbf{\Phi}_1. \tag{1}$$

Definition 2. *Non-negative-correlation:*

$$\mathbf{F}_+ = \{\mathbf{P} \in \mathcal{R}'_p : \langle x\mathbf{P}, \mathbf{P} \rangle \geq 0\}. \tag{2}$$

In NTRU-HRSS, $n = 701$, $q = 8192$, and $p = 3$ are employed. It consists of three parts:

(1) Alice selects $\mathbf{f}$ and $\mathbf{g}$ uniformly at random from $\mathbf{F}_+$. Then she computes the inverses of $\mathbf{f}$ in $\mathcal{R}'_q$ and $\mathcal{R}'_p$ as $\mathbf{f}_q$ and $\mathbf{f}_p$. Next she computes the public key $\mathbf{h} \leftarrow p\mathbf{g}\mathbf{\Phi}_1\mathbf{f}_q$ and the inverse of $\mathbf{h}$ in $\mathcal{R}'_q$. Finally, she sends $\mathbf{h}$ to Bob.

(2) After receiving $\mathbf{h}$, Bob selects $\mathbf{r}$ and the shared key $\mathbf{m}$ uniformly at random from $\mathcal{R}'_p$. Then he calculates the ciphertext $\mathbf{c} \leftarrow \mathbf{rh} + \mathrm{Lift}(\mathbf{m})$. Subsequently, he sends $\mathbf{c}$ to Alice.

(3) When Alice receives $\mathbf{c}$, she calculates $\mathbf{a} \leftarrow \mathbf{cf}$, $\mathbf{m}' \leftarrow \mathbf{af}_p$ and $\mathbf{r}' \leftarrow (\mathbf{c} - \mathrm{Lift}(\mathbf{m}'))\mathbf{h}_q$. In the end, she checks whether $(\mathbf{r}', \mathbf{m}')$ in message space or not.

2.3 The Key Mismatch Attack Oracle $\mathcal{O}$

Algorithm 1: The Oracle $\mathcal{O}$

```
Input: $\mathbf{c}, \mathbf{m}$
Output: 1 or 0
1 $\mathbf{a} \leftarrow \mathbf{cf} \in \mathcal{R}_q$
2 $\mathbf{m}' \leftarrow \mathbf{af}_p \in \mathcal{R}'_p$
3 $\mathbf{r}' \leftarrow (\mathbf{c} - \mathrm{Lift}(\mathbf{m}'))\mathbf{h}_q) \in \mathcal{R}'_q$
4 if $(\boldsymbol{r}', \boldsymbol{m}') \in \mathcal{R}'_p \times \mathcal{R}'_p$ and $\boldsymbol{m} = \boldsymbol{m}'$ then
5     Return 1;
6 else
7     Return 0;
```

In the process of key mismatch attack on NTRU-HRSS, Alice is an honest server and the adversary $\mathcal{A}$ acts as Bob. For convenience, we build the Oracle $\mathcal{O}$ that plays the role of Alice. In addition, we suppose that public key $\mathbf{h}$ is reused and $\mathcal{A}$ has access to the Oracle $\mathcal{O}$ many times. The inputs of oracle $\mathcal{O}$ are $\mathbf{c}$ and $\mathbf{m}$. Afterwards, $\mathcal{O}$ calculates $\mathbf{a}$, $\mathbf{m}'$ and $\mathbf{r}'$ as depicted in Algorithm 1. Next, $\mathcal{O}$ first checks whether $(\mathbf{r}', \mathbf{m}')$ in the message space and then checks whether $\mathbf{m} = \mathbf{m}'$ holds. If both of them are yes, $\mathcal{O}$ outputs 1, and 0 otherwise. That is, when $\mathcal{O}$ outputs 1, $\mathbf{m}$ and $\mathbf{m}'$ match, otherwise $\mathbf{m}$ and $\mathbf{m}'$ mismatch. By observing the outputs of $\mathcal{O}$, $\mathcal{A}$ can get information about the secret key.

2.4 Why Ding et al.'s Method cannot be applied to NTRU-HRSS

To illustrate why Ding et al.'s method cannot be applied to NTRU-HRSS, we suppose that $\mathcal{A}$ directly uses Ding et al.'s method. Then $\mathcal{A}$ first needs to find a longest chain that consists of either consecutive 1s or consecutive -1s of consequent coefcients of the secret key g.

To launch this attack, $\mathcal{A}$ sets $\mathbf{m}$ as $\mathbf{0}$ and $\mathbf{r}[0] = \cdots = \mathbf{r}[l-1] = \lceil \frac{q}{2pl} \rceil$, where l ranges from 2 to $n-1$. Then, $\mathcal{A}$ calculates $\mathbf{c}$. Then $\mathbf{c}$ and $\mathbf{m}$ are sent to the oracle $\mathcal{O}$.

By receiving the inputs $\mathbf{c}$ and $\mathbf{m}$, $\mathcal{O}$ first calculates

$$\begin{aligned}
\mathbf{a} &\equiv \mathbf{cf} && \pmod q \\
&\equiv \mathbf{rhf} + \mathrm{Lift}(\mathbf{m})\mathbf{f} && \pmod q \\
&\equiv \mathbf{rhf} && \pmod q \\
&\equiv p\mathbf{rg}\mathbf{\Phi}_1\mathbf{f}_q\mathbf{f} && \pmod q.
\end{aligned} \tag{3}$$

Since $\mathbf{f}_q \in \mathcal{R}'_q$, $\mathbf{f}_q\mathbf{f} = (1 + \mathrm{t}\mathbf{\Phi}_n)$, where $t \in \mathbb{Z}$. Further, we have

$$\begin{aligned}
\mathbf{a} &\equiv p\mathbf{rg}\mathbf{\Phi}_1(1 + \mathrm{t}\mathbf{\Phi}_n) && \pmod q \\
&\equiv p\mathbf{rg}\mathbf{\Phi}_1 && \pmod q \\
&\equiv p\mathbf{rg}(x-1) && \pmod q \\
&\equiv p\mathbf{rg}x - p\mathbf{rg} && \pmod q
\end{aligned} \tag{4}$$

Next, for $i = 0, 1, \ldots n-1$, $\mathcal{O}$ gets

$$\mathbf{a}[i] \equiv \begin{cases} p(\mathbf{rg})[n-1] - p(\mathbf{rg})[0] & \pmod q \text{ if } i = 0, \\ p(\mathbf{rg})[i-1] - p(\mathbf{rg})[i] & \pmod q \text{ otherwise.} \end{cases} \tag{5}$$

Similarly, since $\mathbf{r} = (\mathbf{r}[0], \ldots, \mathbf{r}[l-1])$=$(\lceil \frac{q}{2pl} \rceil, \ldots, \lceil \frac{q}{2pl} \rceil)$, Equation (5) can be transformed as

$$\mathbf{a}[i] \equiv \begin{cases} p\lceil \dfrac{q}{2pl} \rceil(\mathbf{g}[n-l] - \mathbf{g}[0]) & \pmod q \text{ if } i = 0, \\ p\lceil \dfrac{q}{2pl} \rceil(\mathbf{g}[(i-l) \bmod n] - \mathbf{g}[i]) & \pmod q \text{ otherwise.} \end{cases} \tag{6}$$

The basic idea of Ding et al.'s method is to find a longest chain of length k to help recovering all the coefficients in $\mathbf{g}$. To achieve this goal, $\mathbf{a}[i]$ should be related to $k+1$ coefficients. In original NTRU, the number of 1s and -1s is equal in $\mathbf{g}$, i.e. $\mathbf{g} \equiv 0 \pmod{\mathbf{\Phi}_1}$, then $\mathbf{a}[i]$ is related to $k+1$ coefficients. But in NTRU-HRSS, the distribution of $\mathbf{g}$ is modified and $\mathbf{g} \not\equiv 0 \pmod{\mathbf{\Phi}_1}$, then $\mathbf{a}[i]$ is only related to two coefficients of $\mathbf{g}$ as depicted in Equation (6). Hence, we cannot use this method to recover the longest chain of $\mathbf{g}$, let alone the remaining coefficients of $\mathbf{g}$. Therefore, Ding et al.'s method cannot be applied directly.

3 Our Proposed Attack

In this section, we propose an attack on NTRU-HRSS KEM. Firstly, we introduce the parameter choices of the adversary, then propose improvements in two following subsections, finally describe the complete attack.

3.1 Parameter Choices of the Adversary

We recover $\mathbf{G} = \mathbf{g}\mathbf{\Phi}_1$ instead of $\mathbf{g}$ in Ding et al.'s method. We say that $\mathcal{A}$ succeeds if he recovers any equivalent of $\mathbf{G}$, which is denoted as $\mathbf{G}'$.

The equivalent of G. $\mathbf{G}'$ can differ from $\mathbf{G}$ by a sign $s \in \{-1, 1\}$ and a shifting of its coefficients. The relationship between $\mathbf{G}'$ and $\mathbf{G}$ is shown as follows, for some integer $v \in \mathbb{N}$,

$$\begin{aligned}\mathbf{G}' &= s\sum_{i=0}^{n-1}\mathbf{G}[(i+v) \bmod n]x^i \\ &= sx^v\sum_{i=0}^{n-1}\mathbf{G}[i]x^i \\ &= sx^v\mathbf{G}.\end{aligned} \tag{7}$$

Now Equation (4) becomes $\mathbf{a} \equiv p\mathbf{r}\mathbf{G} \pmod q$.
And for $i = 0, 1, \ldots, n-1$, $t \in \mathbb{Z}$, we get

$$\mathbf{a}[i] = \begin{cases} p(\mathbf{rG})[i] & \text{if } p(\mathbf{rG})[i] \in \left[-\frac{q}{2}, \frac{q}{2}\right], \\ p(\mathbf{rG})[i] - tq & \text{otherwise.} \end{cases} \tag{8}$$

Then we have

$$\begin{aligned}\mathbf{m}'[i] &\equiv (\mathbf{a}\mathbf{f}_p)[i] && \pmod p \\ &\equiv (\mathbf{a}[0]\mathbf{f}_p[i] + \cdots \mathbf{a}[n-1]\mathbf{f}_p[(i+1) \bmod n]) && \pmod p.\end{aligned} \tag{9}$$

Next, if all $p(\mathbf{rG})[i] \in \left[-\frac{q}{2}, \frac{q}{2}\right]$, the corresponding $\mathbf{m}'[i]$ is equal to

$$\begin{aligned}\mathbf{m}'[i] &\equiv p(\mathbf{rG})[0]\mathbf{f}_p[i] + \cdots p(\mathbf{rG})[n-1]\mathbf{f}_p[(i+1) \bmod n] \pmod p \\ &\equiv 0 \pmod p.\end{aligned} \tag{10}$$

Otherwise, if there is one $p(\mathbf{rG})[j] \notin \left[-\frac{q}{2}, \frac{q}{2}\right]$, for $j \in [0, n-1]$, then

$$\mathbf{m}'[i] \equiv -tq\mathbf{f}_p[(i-j) \bmod n] \not\equiv 0 \pmod p. \tag{11}$$

Since we set $\mathbf{m} = 0$, Equation(11) means that $\mathbf{m}' \neq \mathbf{m}$ and the corresponding $\mathcal{O}$ outputs 0. Therefore, in order to recover the coefficients of $\mathbf{G}$, we only need to make $p(\mathbf{rG})[j] \notin [-\frac{q}{2}, \frac{q}{2}]$ by setting the proper coefficients of $\mathbf{r}$. Then, according to the output of $\mathcal{O}$, we can recover $\mathbf{G}$.

3.2 Finding a Longest Chain

After that, the remaining problem is how to recover $\mathbf{G}$ according to the output of $\mathcal{O}$. In NTRU-HRSS, the most crucial issue is finding a longest chain. To illustrate this issue, we first discuss the range of coefficients in $\mathbf{G}$.

$$\begin{aligned}\mathbf{G} &= \mathbf{g}\mathbf{\Phi}_1 \\ &= \mathbf{g}(x-1) \\ &= (\mathbf{g}[0]x + \cdots + \mathbf{g}[n-1]x^n) - (\mathbf{g}[0] + \cdots + \mathbf{g}[n-1]x^{n-1}) \\ &= (\mathbf{g}[n-1] - \mathbf{g}[0]) + (\mathbf{g}[0] - \mathbf{g}[1])x + \cdots + (\mathbf{g}[n-2] - \mathbf{g}[n-1])x^{n-1},\end{aligned} \tag{12}$$

and for $i = 0, 1, \dots n-1$,

$$\mathbf{G}[i] = \begin{cases} \mathbf{g}[n-1] - \mathbf{g}[0] & \text{if } i = 0, \\ \mathbf{g}[i-1] - \mathbf{g}[i] & \text{otherwise.} \end{cases} \tag{13}$$

According to Equation (13), the sum of z consecutive coefficients in $\mathbf{G}$ is

$$\mathbf{G}[i] + \cdots + \mathbf{G}[i+z-1] = \begin{cases} \mathbf{g}[n-1] - \mathbf{g}[z-1] & \text{if } i = 0, \\ \mathbf{g}[i-1] - \mathbf{g}[i+z-1] & \text{otherwise,} \end{cases} \tag{14}$$

where $z \in [2, n]$.

For $i \in [0, n-1]$, $\mathbf{g}[i] \in [-1, 1]$, according to Equation (13), $\mathbf{G}[i] \in [-2, 2]$. And according to Equation (14), we can get $(\mathbf{G}[i] + \cdots + \mathbf{G}[i+z-1]) \in [-2, 2]$.

Specifically, when $z = 2$, $(\mathbf{G}[i] + \mathbf{G}[i+1]) \in [-2, 2]$. By simply adding the two consecutive coefficients, we can draw the first conclusion:

(1) In $\mathbf{G}$, the tuples {(1,2), (2,1), (-1,-2), (-2,-1), (-2,-2), (2,2)} do not exist.

When $z = 3$, $(\mathbf{G}[i] + \mathbf{G}[i+1] + \mathbf{G}[i+2]) \in [-2, 2]$. Therefore, we can similarly conclude that:

(2) There are at most two consecutive 1's or -1's in $\mathbf{G}$.

By denoting a chain $(2, -2, \dots, 2 * (-1)^{k-1})$ with length k the k-chain, we have the following result.

Theorem 1. *When $k \leq 6$, we can find a longest chain of the form $(2, -2, \dots, 2 * (-1)^{k-1})$ in $\boldsymbol{G}$ with a high probability.*

Proof. From the second observation, we cannot find a longest chain consisting of consecutive 1's or -1's. Therefore, a longest k-chain consists of consecutive coefficients as $(2, -2, \dots, 2 * (-1)^{k-1})$ in $\mathbf{G}$.

According to Equation (13), for $i = 0, 1, \dots n-1$, we note that when $\mathbf{G}[i] = 2$, it should be the case that $\mathbf{g}[i-1] = 1$, $\mathbf{g}[i] = -1$. Corresponding to this, when $\mathbf{G}[i] = -2$, we have $\mathbf{g}[i-1] = -1$, $\mathbf{g}[i] = 1$. Thus, the occurrence probability of $(2, -2, \dots, 2 * (-1)^{k-1})$ in $\mathbf{G}$ is the same as that of $(-1, 1, \dots, (-1)^{k-1})$ in $\mathbf{g}$. As we mentioned above, the probabilities of occurrences of -1, 0 and 1 are $\frac{85}{256}$, $\frac{86}{256}$, $\frac{85}{256}$, respectively.

Let $\mathcal{X}$ denote the event that a k-chain $(2, -2, \dots, 2 * (-1)^{k-1})$ occurs in $\mathbf{G}$. To calculate the average number of times a k-chain occurs in $\mathbf{G}$, we try to get the corresponding expectation $E_k(\mathcal{X})$. By dividing $\mathcal{X}$ into the subevents $\mathcal{X}_i$ $i = 0, \dots, n-k$, where each $\mathcal{X}_i$ denotes the event that a k-chain $(2, -2, \dots, 2 * (-1)^{k-1})$ occurs in the i-th position of $\mathbf{G}$. Since $E_k(\mathcal{X}_i) = (\frac{85}{256})^k$ and recall that n =701, from the property of Expectation, we have

$$E_k(\mathcal{X}) = \sum_{i=0}^{n-k} E_k(\mathcal{X}_i) = (\frac{85}{256})^k * (702 - k). \tag{15}$$

In Table 2, we show the relationship between k and $E_k(\mathcal{X})$, where k is selected from the set {2,3,4,5,6,7}. As we can see when $k = 6$, $E_k(\mathcal{X})$ is near 1.

Table 2. The average number of times a k-chain occurs in $\mathbf{G}$

k	2	3	4	5	6	7
$E_k(\mathcal{X})$	77.171	25.587	8.483	2.813	0.933	0.309

The results show that there is at least a longest chain that consists of consecutive coefficients such as $(2, -2, \ldots, 2 * (-1)^{k-1})$ in $\mathbf{G}$ when $k = 6$. Afterwards, we use the chain as an anchor to recover $\mathbf{G}$.

3.3 The Selection of Parameter r

In addition, in order to recover $\mathbf{G}$, the adversary $\mathcal{A}$ directly sets all coefficients of $\mathbf{r}$ as 0 except for the first few coefficients of $\mathbf{r}$. In NTRU-HRSS, while setting $\mathbf{r}$, we need to increase the weight of targeted coefficients.

As we stated above, the coefficients of $\mathbf{G}$ range from -2 to 2, and there are many disturbances to prevent us from recovering coefficients of $\mathbf{G}$ correctly. We take two adjacent coefficients of $\mathbf{G}$ as an illustration, including $5^2 = 25$ tuples. According to the first conclusion above, the tuples of the set {(1,2), (2,1), (-1,-2), (-2,-1), (-2,-2), (2,2)} do not exist in $\mathbf{G}$. Then, we can classify the remaining tuples in accordance with the summation of the two adjacent coefficients in Table 3. The tuples of equal summation interfere with each other's recovery. Concretely, when the summation is -2, the tuples (-2,0) and (0,-2) disturb the recovery of the tuple (-1,-1). Therefore, we need to make the sum of these tuples unequal to eliminate the disturbance by increasing the weight of some coefficients. Obviously, increasing the weight of 0 is useless, and increasing the weight of 1 or -1 is ineffective since 2 or -2 can get the double weight. Finally, we can only increase the weight of coefficients 2 and -2, which has proved to be effective.

Table 3. Different summation of the two adjacent coefficients

Summation	Tuple1	Tuple2	Tuple3	Tuple4	Tuple5
-2	(-2,0)	(0,-2)	(-1,-1)		
-1	(-2,1)	(1,-2)	(-1,0)	(0,-1)	
0	(-2,2)	(2,-2)	(-1,1)	(1,-1)	(0,0)
1	(-1,2)	(2,-1)	(0,1)	(1,0)	
2	(0,2)	(2,0)	(1,1)		

Additionally, the tuples of different summation can also interfere each other's recovery in Table 4. To recover the coefficients of $\mathbf{G}$ correctly, when $\mathbf{G}[i]$=0, if $\mathbf{r}[i-1] \geq 0$, we set $\mathbf{r}[i] > 0$, and if $\mathbf{r}[i-1] < 0$, we set $\mathbf{r}[i] < 0$.

For convenience, we define some symbols. Let num_1 denote the number of recovered coefficients with absolute value of 1 in $\mathbf{G}$, then num_2 denote the number of recovered coefficients with absolute value of 2 in $\mathbf{G}$ and w is the

Table 4. Interference between tuples of different summation

Recovered tuple	Interference tuple1	Interference tuple2
(-2,0)	(-2,2)	(-2,1)
(0,-2)	(1,-2)	(2,-2)
(2,0)	(2,-2)	(2,-1)
(0,2)	(-1,2)	(-2,2)

weight of whose absolute value is 2 in recovered coefficients of $\mathbf{G}$. G_s denotes the weighted sum of recovered coefficients in $\mathbf{G}$, which can be computed as $G_s = 2 * num_2 * w + 1 * num_1$. r_u denotes the unit value of $\mathbf{r}$ and $r_u > 0$.

If w is too large, the mismatch appears prematurely, and if it is too small, it is not enough to recover the target coefficients. To take a balance, we set the initial value of w to 4.

3.4 The Full Attack

In this subsection, we introduce our method to recover $\mathbf{G}$. Recall that we suppose the length of a longest chain that consists of consecutive coefficients such as $(2, -2, \ldots, 2 * (-1)^{k-1})$ in $\mathbf{G}$ is k.

The key mismatch attack consists of three steps. And the adversary $\mathcal{A}$ always sets $\mathbf{m}$ as $\mathbf{0}$ in each step.

Step 1: In this step, the adversary $\mathcal{A}$ recovers $(\mathbf{G}[0], \cdots, \mathbf{G}[k-1])$ and decides the value of k. For this purpose, he needs to find a longest chain in $\mathbf{G}$.

The parameter selections of $\mathbf{r}$ is shown as below.

For $l \geq 2$, $0 \leq i \leq l-1$, $\mathcal{A}$ sets $\mathbf{r} = (\mathbf{r}[0], \ldots, \mathbf{r}[l-1], 0, \ldots, 0)$,

$$\mathbf{r}[i] = \begin{cases} \left\lceil \dfrac{q}{2p * 2l} \right\rceil & \text{if } i \text{ is even,} \\ -\left\lceil \dfrac{q}{2p * 2l} \right\rceil & \text{if } i \text{ is odd.} \end{cases} \tag{16}$$

Since $\mathbf{a} \equiv p\mathbf{r}\mathbf{G} \pmod q$, for $i = 0, 1, \ldots n-1$,

$$\begin{aligned} \mathbf{a}[i] &\equiv p(\mathbf{r}[0]\mathbf{G}[i] + \cdots \mathbf{r}[n-1]\mathbf{G}[(i+1) \bmod n]) \pmod q \\ &\equiv p(\mathbf{r}[0]\mathbf{G}[i] + \cdots \mathbf{r}[l-1]\mathbf{G}[(i-l+1) \bmod n]) \pmod q. \end{aligned} \tag{17}$$

Note that when $l \leq k$, $|\mathbf{a}[i]| \equiv p * \left\lceil \frac{q}{2p*2l} \right\rceil * 2l > \frac{q}{2}$, which means $\mathcal{O}$ outputs 0, and when $l = k+1$, $|\mathbf{a}[i]| \equiv p * \left\lceil \frac{q}{2p*2(k+1)} \right\rceil * (2k+1) < \frac{q}{2}$, which means $\mathcal{O}$ outputs 1. Therefore, when $\mathcal{O}$ outputs 1, $\mathcal{A}$ can get $k = l - 1$.

Step 2: The adversary $\mathcal{A}$ has recovered $(\mathbf{G}[0], \ldots, \mathbf{G}[k-1])$ and he needs to recover $\mathbf{G}[k]$ in this step.

Step 2.1: The adversary $\mathcal{A}$ judges whether $(\mathbf{G}[k], \mathbf{G}[k+1])$ is (0,2) or (0,-2).

```
Algorithm 2: Find-$w$-1
    Input: $num_1, num_2$
    Output: $w$
1   Set $w = 4$, $G_s$ = NULL, $temp$ = NULL;
2   for $i := 1$ to 3 do
3       $G_s = 2 * (num_2 + 1) * w + num_1$;
4       $temp = \lceil \frac{q}{G_s * 2p} \rceil$;
5       if $temp * (G_s - 2 * w) * p \geq \frac{q}{2}$ or $\lceil \frac{\frac{q}{2} - temp * (G_s - 2 * w) * p}{temp * p} \rceil < 2 * w$ then
6           $w = w - 1$;
7       end
8       else
9           break;
10      end
11  end
12  Return $w$
```

```
Algorithm 3: Find-$w$-2
    Input: $num_1, num_2$
    Output: $w$
1   Set $w = 4$, $G_{s_1}$ = NULL, $G_{s_2}$ = NULL;
2   for $i := 1$ to 3 do
3       $G_{s_1} = 2 * num_2 * w + num_1 + 1$;
4       $G_{s_2} = 2 * num_2 * w + num_1 + 2$;
5       if $\lceil \frac{q}{G_{s_1} * 2p} \rceil = \lceil \frac{q}{G_{s_2} * 2p} \rceil$ or $\lceil \frac{q}{G_{s_1} * 2p} \rceil * (G_{s_1} - 1) * p \geq \frac{q}{2}$ then
6           $w = w - 1$;
7       end
8       else
9           break;
10      end
11  end
12  Return $w$
```

First of all, $\mathcal{A}$ sets the proper w in Algorithm 2. In this step, when $\mathcal{O}$ outputs 0, which demonstrates that $(\mathbf{G}[k], \mathbf{G}[k+1])$ is (0,2) or (0,-2). Hence, $\mathcal{A}$ needs to keep $\mathcal{O}$ output 1 before adding the tuple (0,2) or (0,-2). Also, the absolute value of the sum of two coefficients in the tuple must be 2. Otherwise, $\mathcal{A}$ needs to decrease w.

Then $\mathcal{A}$ computes the value of G_s. Since the recovered coefficients of $\mathbf{G}$ are $(\mathbf{G}[0], \mathbf{G}[1], \cdots, \mathbf{G}[k-1])$, and $(\mathbf{G}[k], \mathbf{G}[k+1])$ is (0,2) or (0,-2). Therefore, $num_2 = k+1$, $num_1 = 0$, $G_s = 2*(k+1)*w$.

Next, $\mathcal{A}$ sets $\mathbf{r} = (\mathbf{r}[0], \ldots, \mathbf{r}[k-1], \mathbf{r}[k], \mathbf{r}[k+1], 0, \ldots, 0)$. For $0 \le i \le k-1$,

$$\mathbf{r}[i] = \begin{cases} \left\lceil \dfrac{q}{2p*G_s} \right\rceil * w & \text{if } \mathbf{G}[i] = 2, \\ -\left\lceil \dfrac{q}{2p*G_s} \right\rceil * w & \text{if } \mathbf{G}[i] = -2. \end{cases} \tag{18}$$

Specifically, $\mathcal{A}$ sets $\mathbf{r}[k]$ and $\mathbf{r}[k+1]$ as follows.

(1) When $\mathbf{G}[k-1] = -2$, $(\mathbf{G}[k], \mathbf{G}[k+1])$=(0,2), $\mathbf{r}[k] = \left\lceil \frac{q}{2p*G_s} \right\rceil$, $\mathbf{r}[k+1] = \left\lceil \frac{q}{2p*G_s} \right\rceil * w$. If $\mathcal{O}$ outputs 0, $\mathcal{A}$ recovers $(\mathbf{G}[k], \mathbf{G}[k+1])$ as (0,2). Otherwise $(\mathbf{G}[k], \mathbf{G}[k+1])$ isn't (0,2).

(2) When $\mathbf{G}[k-1] = 2$, $(\mathbf{G}[k], \mathbf{G}[k+1])$=(0,-2), $\mathbf{r}[k] = -\left\lceil \frac{q}{2p*G_s} \right\rceil$, $\mathbf{r}[k+1] = -\left\lceil \frac{q}{2p*G_s} \right\rceil * w$. If $\mathcal{O}$ outputs 0, $\mathcal{A}$ recovers $(\mathbf{G}[k], \mathbf{G}[k+1])$ as (0,-2). Otherwise $(\mathbf{G}[k], \mathbf{G}[k+1])$ isn't (0,-2).

When $(\mathbf{G}[k], \mathbf{G}[k+1])$ is neither (0,2) nor (0,-2), according to Equation (13),

$$\mathbf{G}[k] = \mathbf{g}[k-1] - \mathbf{g}[k], \tag{19}$$

and $\mathbf{g}[k-1]$ is known, so the adversary $\mathcal{A}$ recovers $\mathbf{G}[k]$ in {-1,0} or {1,0} in turn. Specifically, when $\mathbf{G}[k-1] = 2$, $\mathbf{g}[k-1] = -1$, $\mathbf{g}[k] \in \{-1, 0, 1\}$, according to Equation (19), $\mathbf{G}[k] \in \{0, -1, -2\}$. Since the length of a longest chain that consists of consecutive coefficients such as $(2, -2, \ldots, 2*(-1)^{k-1})$ in $\mathbf{G}$ is k, then $\mathbf{G}[k] \neq -2$, $\mathbf{G}[k] \in \{-1, 0\}$. Similarly, when $\mathbf{G}[k-1] = -2$, $\mathbf{G}[k] \in \{1, 0\}$.

Step 2.2: When $(\mathbf{G}[k], \mathbf{G}[k+1])$ is neither (0,2) nor (0,-2), $\mathcal{A}$ recovers $\mathbf{G}[k]$ in {-1,0} or {1,0} in turn.

Firstly, $\mathcal{A}$ sets the proper w according to Algorithm 3. Then $\mathcal{A}$ computes the value of G_s, since the recovered coefficients of $\mathbf{G}$ are $(\mathbf{G}[0], \mathbf{G}[1], \cdots, \mathbf{G}[k-1])$, and $\mathbf{G}[k] \in \{-1, 0\}$ or $\{1, 0\}$. Therefore, $num_2 = k$, $num_1 = 1$, $G_s = 2k*w+1$. Next, $\mathcal{A}$ sets $\mathbf{r} = (\mathbf{r}[0], \ldots, \mathbf{r}[k-1], \mathbf{r}[k], 0, \ldots, 0)$. For $0 \le i \le k-1$, he sets according to Equation (18). Afterward, we discuss the parameter selection of $\mathbf{r}[k]$ in cases {-1,0} and {1,0}, respectively.

(1) When $\mathbf{G}[k-1] = 2$, i.e. $\mathbf{G}[k] \in \{-1, 0\}$, $\mathbf{r}[k] = -\left\lceil \frac{q}{2p*G_s} \right\rceil$. If the output of $\mathcal{O}$ related to the selection of $\mathbf{r}$ is 0, $\mathbf{G}[k] = -1$, otherwise $\mathbf{G}[k] = 0$.

(2) When $\mathbf{G}[k-1] = -2$, then $\mathbf{G}[k] \in \{1, 0\}$, $\mathbf{r}[k] = \left\lceil \frac{q}{2p*G_s} \right\rceil$. If $\mathcal{O}$'s output associated with the selection of $\mathbf{r}$ is 0, $\mathbf{G}[k] = 1$, otherwise $\mathbf{G}[k] = 0$.

Step 3: Suppose that $\mathcal{A}$ has recovered $(\mathbf{G}[0], \dots, \mathbf{G}[k-1], \mathbf{G}[k], \cdots, \mathbf{G}[k+t-1])$, then $\mathcal{A}$ needs to recover $\mathbf{G}[k+t]$, where $t \in [1, n-k-1]$.

Recall that $(\mathbf{G}[0], \dots, \mathbf{G}[k-1])$ is denoted as a k-chain, we denote a chain $(\mathbf{G}[0], \cdots, \mathbf{G}[k-1], \mathbf{G}[k], \cdots, \mathbf{G}[z'-1])$ with length z' the z'-chain, which is the extension of the k-chain. Here $\mathbf{G}[k], \cdots, \mathbf{G}[z'-1]$ can be arbitrary coefficients and z' is a fixed number. Through experiments we find that by setting $z = 15$ we can get the best results.

Step 3.1: $\mathcal{A}$ needs to determine whether $(\mathbf{G}[k+t], \mathbf{G}[k+t+1])$ is in the set {(0,2), (2,0), (0,-2), (-2,0)}.

Firstly, $\mathcal{A}$ selects the proper w in Algorithm 2 and computes the value of G_s according to $G_s = 2 * num_2 * w + num_1$. Then, according to Equation (13),

$$\mathbf{G}[k+t] = \mathbf{g}[k+t-1] - \mathbf{g}[k+t], \tag{20}$$

and $\mathbf{g}[k+t-1]$ is known, thus $\mathcal{A}$ can recover $(\mathbf{G}[k+t], \mathbf{G}[k+t+1])$ in the set {(0,2), (2,0)} or {(0,-2), (-2,0)}, respectively.

Specifically, when $\mathbf{g}[k+t-1] = -1$, $(\mathbf{G}[k+t], \mathbf{G}[k+t+1])$ is in the set {(0,-2), (-2,0)}. When $\mathbf{g}[k+t-1] = 1$, $(\mathbf{G}[k+t], \mathbf{G}[k+t+1])$ is in {(0,2), (2,0)}.

After that, $\mathcal{A}$ sets $\mathbf{r} = (\mathbf{r}[0], \dots, \mathbf{r}[z'-1], 0, \dots, 0, \mathbf{r}[k+t], \mathbf{r}[k+t+1], 0, \dots, 0)$. For $0 \le i \le z'-1$,

$$\mathbf{r}[i] = \begin{cases} \left\lceil \dfrac{q}{2p * G_s} \right\rceil * w * \dfrac{\mathbf{G}[i]}{2} & \text{if } |\mathbf{G}[i]| = 2, \\ \left\lceil \dfrac{q}{2p * G_s} \right\rceil * \mathbf{G}[i] & \text{if } |\mathbf{G}[i]| = 1, \\ \left\lceil \dfrac{q}{2p * G_s} \right\rceil & \text{if } \mathbf{G}[i] = 0 \text{ and } \mathbf{r}[i-1] \ge 0, \\ -\left\lceil \dfrac{q}{2p * G_s} \right\rceil & \text{if } \mathbf{G}[i] = 0 \text{ and } \mathbf{r}[i-1] < 0. \end{cases} \tag{21}$$

And $\mathcal{A}$ sets $\mathbf{r}[k+t]$ and $\mathbf{r}[k+t+1]$ as follows.

(1) When $\mathbf{g}[k+t-1] = -1$, $(\mathbf{G}[k+t], \mathbf{G}[k+t+1])$ is in {(0,-2), (-2,0)}. Firstly, $\mathcal{A}$ sets $\mathbf{r}[k+t]$ and $\mathbf{r}[k+t+1]$ as:

1) (-2,0): $\mathbf{r}[k+t] = -\left\lceil \frac{q}{2p*G_s} \right\rceil * w$, $\mathbf{r}[k+t+1] = -\left\lceil \frac{q}{2p*G_s} \right\rceil$,

2) (0,-2): $\mathbf{r}[k+t] = -\left\lceil \frac{q}{2p*G_s} \right\rceil$, $\mathbf{r}[k+t+1] = -\left\lceil \frac{q}{2p*G_s} \right\rceil * w$.

$\mathcal{A}$ recovers $(\mathbf{G}[k+t], \mathbf{G}[k+t+1])$ as (-2,0) if $\mathcal{O}$'s output associated with the first choice of $\mathbf{r}$ is 0, otherwise $\mathcal{A}$ continues to set in the order. If the output of $\mathcal{O}$ associated with the second choice of $\mathbf{r}$ is 0, $\mathcal{A}$ recovers $(\mathbf{G}[k+t], \mathbf{G}[k+t+1])$ as (0,-2). Finally, if $\mathcal{O}$ does not output 0, which demonstrates $(\mathbf{G}[k+t], \mathbf{G}[k+t+1])$ is neither (-2,0) nor (0,-2).

(2) When $\mathbf{g}[k+t-1] = 1$, $(\mathbf{G}[k+t], \mathbf{G}[k+t+1])$ is in {(0,2), (2,0)}. At the outset, $\mathcal{A}$ sets $\mathbf{r}[k+t]$ and $\mathbf{r}[k+t+1]$ as:

1) (0,2): $\mathbf{r}[k+t] = \left\lceil \frac{q}{2p*G_s} \right\rceil$, $\mathbf{r}[k+t+1] = \left\lceil \frac{q}{2p*G_s} \right\rceil * w$,

2) (2,0): $\mathbf{r}[k+t] = \left\lceil \frac{q}{2p*G_s} \right\rceil * w$, $\mathbf{r}[k+t+1] = \left\lceil \frac{q}{2p*G_s} \right\rceil$.

Similarly, $\mathcal{A}$ sets $\mathbf{r}[k+t]$ and $\mathbf{r}[k+t+1]$ in the order and when $\mathcal{O}$ outputs 0, $\mathcal{A}$ recovers $\mathbf{G}[k+t], \mathbf{G}[k+t+1]$ as the corresponding tuple. In the end, if $\mathcal{O}$ does not output 0, $(\mathbf{G}[k+t], \mathbf{G}[k+t+1])$ is neither (0,2) nor (2,0).

Step 3.2: When $(\mathbf{G}[k+t], \mathbf{G}[k+t+1])$ isn't in {(0,2), (2,0), (0,-2), (-2,0)}, $\mathcal{A}$ recovers $\mathbf{G}[k+t]$ in {0,-1,-2}, {1,0,-1} or {2,1,0} in turn.

Table 5. The two outputs of $\mathcal{O}$ corresponding to $\mathbf{G}[k+t]$ in three sets

$\mathbf{G}[k+t]$ \ $\mathcal{O}$ / Set	(0,0)	(0,1)	(1,1)	(1,0)
{0,-1,-2}	-2	-1	0	
{1,0,-1}	1	1	0	-1
{2,1,0}	2	1	0	

Specifically, when $\mathbf{g}[k+t-1] = -1$, $\mathbf{g}[k+t] \in \{-1, 0, 1\}$, according to Equation (20), $\mathbf{G}[k+t] \in \{0, -1, -2\}$. When $\mathbf{g}[k+t-1] = 0$, $\mathbf{G}[k+t] \in \{1, 0, -1\}$. And when $\mathbf{g}[k+t-1] = 1$, $\mathbf{G}[k+t] \in \{2, 1, 0\}$.

Next, we discuss the parameter selections of $\mathbf{r}$ in cases {0,-1,-2}, {1,0,-1} and {2,1,0}, respectively. The two outputs of $\mathcal{O}$ corresponding to $\mathbf{G}[k+t]$ are shown in Table 5.

(1) When $\mathbf{g}[k+t-1] = -1$, then $\mathbf{G}[k+t] \in \{0, -1, -2\}$. $\mathcal{A}$ first selects w according to Algorithm 3. Then $\mathcal{A}$ computes $G_s = 2 * num_2 * w + num_1 + 1$. Next, $\mathcal{A}$ sets $\mathbf{r} = (\mathbf{r}[0], \ldots, \mathbf{r}[z'-1], 0, \ldots, 0, \mathbf{r}[k+t], 0, \ldots, 0)$, for $0 \leq i \leq z'-1$, he sets according to Equation(21) and $\mathbf{r}[k+t] = -\left\lceil \frac{q}{2p*G_s} \right\rceil$.

Afterward, $\mathcal{A}$ sets $\mathbf{r} = (\mathbf{r}[0], \ldots, \mathbf{r}[z'-1], 0, \ldots, 0, \mathbf{r}[k+t], 0, \ldots, 0]$, for $0 \leq i \leq z'-1$,

$$\mathbf{r}[i] = \begin{cases} \left\lceil \dfrac{q}{2p*(G_s+1)} \right\rceil * w * \dfrac{\mathbf{G}[i]}{2} & \text{if } |\mathbf{G}[i]| = 2, \\ \left\lceil \dfrac{q}{2p*(G_s+1)} \right\rceil * \mathbf{G}[i] & \text{if } |\mathbf{G}[i]| = 1, \\ \left\lceil \dfrac{q}{2p*(G_s+1)} \right\rceil & \text{if } \mathbf{G}[i] = 0 \text{ and } \mathbf{r}[i-1] \geq 0, \\ -\left\lceil \dfrac{q}{2p*(G_s+1)} \right\rceil & \text{if } \mathbf{G}[i] = 0 \text{ and } \mathbf{r}[i-1] < 0. \end{cases} \tag{22}$$

and $\mathbf{r}[k+t] = -\left\lceil \frac{q}{2p*(G_s+1)} \right\rceil$.

If the only output of $\mathcal{O}$ related to the first choice of $\mathbf{r}$ is 0, $\mathbf{G}[k+t] = -1$. And if both of the two outputs of $\mathcal{O}$ are 0, $\mathbf{G}[k+t] = -2$. Then if both of the two outputs of $\mathcal{O}$ are 1, $\mathbf{G}[k+t] = 0$.

(2) When $\mathbf{g}[k+t-1] = 1$, $\mathbf{G}[k+t] \in \{2, 1, 0\}$. Similarly, $\mathcal{A}$ first selects the proper w and calculates G_s. Then, for $0 \leq i \leq z'-1$, $\mathcal{A}$ sets $\mathbf{r}$ according to

Equation (21) and $\mathbf{r}[k+t] = \left\lceil \frac{q}{2p*G_s} \right\rceil$. Subsequently, for $0 \le i \le z'-1$, $\mathcal{A}$ sets $\mathbf{r}$ according to Equation(22) and $\mathbf{r}[k+t] = \left\lceil \frac{q}{2p*(G_s+1)} \right\rceil$. If the only $\mathcal{O}'$s output associated with the first choice of $\mathbf{r}$ is 0, $\mathbf{G}[k+t] = 1$. And if $\mathcal{O}$ outputs 0 twice, $\mathbf{G}[k+t] = 2$. Then if $\mathcal{O}$ outputs 1 twice, $\mathbf{G}[k+t] = 0$.

(3) When $\mathbf{g}[k+t-1] = 0$, $\mathbf{G}[k+t] \in \{1, 0, -1\}$. $\mathcal{A}$ similarly selects w according to Algorithm 3 except judging whether $\lceil \frac{q}{G_{s_1}*2p} \rceil = \lceil \frac{q}{G_{s_2}*2p} \rceil$ holds. Then $\mathcal{A}$ computes the value of G_s by $G_s = 2*num_2*w + num_1 + 1$. Next, $\mathcal{A}$ sets $\mathbf{r} = (\mathbf{r}[0], \ldots, \mathbf{r}[z'-1], 0, \ldots, 0, \mathbf{r}[k+t], 0, \ldots, 0)$, for $0 \le i \le z'-1$. After that, he sets $\mathbf{r}[k+t] = -\left\lceil \frac{q}{2p*G_s} \right\rceil$.

If the only $\mathcal{O}'$s output related to the first choice of $\mathbf{r}$ is 0, $\mathbf{G}[k+t] = 1$. And if $\mathcal{O}$ only outputs 0 on the second choice of $\mathbf{r}$, $\mathbf{G}[k+t] = -1$. Then if $\mathcal{O}$ outputs 1 twice, $\mathbf{G}[k+t] = 0$. In addition, if $\mathcal{O}$ outputs 0 twice, which demonstrates there are at least two chains of length z' in $\mathbf{G}$. In order to recover the unique $\mathbf{G}$, $\mathcal{A}$ simply assigns $\mathbf{G}[k+t]$ as 1, and he sets $\mathbf{r}[k+t]$ in recovering the remaining coefficients of $\mathbf{G}$ later.

Finally, $\mathcal{A}$ repeats **Step 3** until all the coefficients of $\mathbf{G}$ are recovered.

4 Experiments and Analysis

In this section, we introduce our experiments and the results show the correctness and efficiency of our proposed attack. We run our code on an Intel Xeon E5-2620 at 2.1 GHz and a 64 GB RAM and our code is made public[1].

To make our experiments more convincing, we generate 10,000 secret keys using the code submitted to NIST [6] by the designers of NTRU, and then take an average. At first, we succeed in launching key mismatch attacks on the ntruhps2048509, ntruhps2048677 and ntruhps4096821 by using Ding et al.s method. Experiments demonstrate the correctness of the method, and we can indeed get an equivalent of secret key in NTRU-HPS with almost 100% probability.

Recall that z' denotes the length of z'-chain, and w is the weight of whose absolute value is 2 in recovered coefficients of $\mathbf{G}$. In NTRU-HRSS, the parameters $(n, q, p) = (701, 8192, 3)$. Then we implement our proposed method to recover all coefficients of the secret key in NTRU-HRSS, where we try different z' in the set {5,7,9,11,13,15,17,19}.

The results are shown in Table 6, where we illustrate the trade-off among the success rate, average number of queries, and average time. We also represent the relationship between the success rate and z' in Figure 1. It is notable that our proposed attack can achieve the success rate of 93.6% when $z' = 15$. The results show that among 10,000 secret keys, all coefficients of 9360 secret keys can be recovered. Meanwhile, when $z' = 15$, it also represents the least number of queries. Therefore, we choose $z' = 15$ in the final.

[1] `https://github.com/AHaQY/Key-Mismatch-Attack-on-NIST-KEMs/tree/master/ntruhrss701_key_mismatch_attack`

Table 6. Performance comparison when increasing the value of z'

z'	Success rate(%)	Average #queries	Average time (s)
5	50.0	1884	12.652
7	70.0	1879	11.744
9	90.0	1855	11.471
11	92.4	1866	11.586
13	92.9	1856	11.853
15	93.6	1844	11.983
17	92.7	1867	11.858
19	90.0	1945	11.965

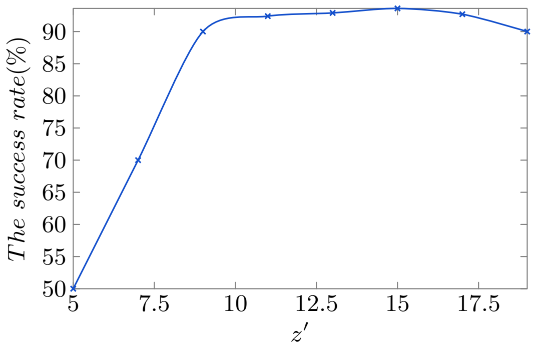

Fig. 1. The relationship between z' and the success rate

And when $z' > 15$, we need to set a smaller weight w, which prevents us from distinguishing some coefficients. Thus, the success rate continues decreasing as depicted in Figure 1. In addition, when the weight $w = 1$, we cannot continue decreasing w as required in Algorithm 2 or Algorithm 3. Otherwise the value of w is 0, which is the reason why the success rate cannot be 100%.

5 Conclusion

In this paper, we propose a key mismatch attack on NTRU-HRSS KEM. Furthermore, we illustrate the trade-off among the success rate, average number of queries, and the average running time. As a result, we can achieve minimum number of queries with a success rate of 93.6% at the same time. NTRU-HRSS KEM submitted to NIST is CCA-secure, so our proposed key mismatch attack does not harm the NTRU-HRSS designers' security goals. However, in view of the aspect of efficiency in the real life, the CPA version of NTRU-HRSS may be used with the public key repeatedly used. Therefore, in order to avoid the leakage of secret key, we need to take some countermeasures, such as regularly updating the key or adding a system to detect the user's query operation.

Bibliography

[1] Alagic, G., Alperin-Sheriff, J., Apon, D., Cooper, D., Dang, Q., Kelsey, J., Liu, Y.K., Miller, C., Moody, D., Peralta, R., et al.: Status report on the second round of the nist post-quantum cryptography standardization process. Tech. rep. (2020), `https://nvlpubs.nist.gov/nistpubs/ir/2020/NIST.IR.8309.pdf`

[2] Alkim, E., Avanzi, R., Bos, J.W., Ducas, L., de la Piedra, A., Pöppelmann, P.S.T., Stebila, D.: Newhope. Submission to the NIST Post-Quantum Cryptography standardization project, Round 2 (2019)

[3] Avanzi, R., Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J., Schwabe, P., Seiler, G., Stehlé, D.: Kyber: Algorithm specifications and supporting documentation, version 2.0, nist pqc round 2. Tech. rep., en. Tech. rep (2019)

[4] Băetu, C., Durak, F.B., Huguenin-Dumittan, L., Talayhan, A., Vaudenay, S.: Misuse attacks on post-quantum cryptosystems. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 747–776. Springer (2019)

[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] Chen, C., Danba, O., Hoffstein, J., Hulsing, A., Rijneveld, J., Schanck, J.M., Schwabe, P., Whyte, W., Zhang, Z.: Ntru: algorithm specifications and supporting documentation (2019)

[7] Chen, L., Jordan, S., Liu, Y.K., Moody, D., Peralta, R., Perlner, R., Smith-Tone, D.: Report on post-quantum cryptography. US Department of Commerce, National Institute of Standards and Technology (2016)

[8] 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)

[9] Ding, J., Deaton, J., Schmidt, K., Vishakha, Zhang, Z.: A simple and efficient key reuse attack on ntru cryptosystem (2019), `https://eprint.iacr.org/2019/1022`

[10] 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)

[11] 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`.

[12] DAnvers, J.P., Karmakar, A., Roy, S.S., Vercauteren, F.: Saber: Mod-lwr based kem (round 2 submission). Tech. rep., Tech. Rep (2019)

[13] Fluhrer, S.R.: Cryptanalysis of ring-lwe based key exchange with key share reuse. IACR Cryptology ePrint Archive (2016), `http://eprint.iacr.org/2016/085`.

[14] Greuet, A., Montoya, S., Renault, G.: Attack on lac key exchange in misuse situation. IACR Cryptology ePrint Archive **2020**, 63 (2020)

[15] Gyongyosi, L., Imre, S.: A survey on quantum computing technology. Computer Science Review **31**, 51–71 (2019)

[16] Hoffstein, J., Pipher, J., Silverman, J.H.: Ntru: A ring-based public key cryptosystem. In: International Algorithmic Number Theory Symposium. pp. 267–288. Springer (1998)

[17] Hülsing, A., Rijneveld, J., Schanck, J.M., Schwabe, P.: Ntru-hrss-kem: algorithm specifications and supporting documentation (2017)

[18] Kirkwood, D., Lackey, B.C., McVey, J., Motley, M., Solinas, J.A., Tuller, D.: Failure is not an option: Standardization issues for post-quantum key agreement. In: Workshop on Cybersecurity in a Post-Quantum World. p. 21 (2015)

[19] Langlois, A., Stehl, D.: Worst-case to average-case reductions for module lattices (2015)

[20] Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. In: Cryptology - EUROCRYPT. pp. 1–23 (2010)

[21] Moody, D.: Post-quantum cryptography standardization: Announcement and outline of nist's call for submissions (2016)

[22] Okada, S., Wang, Y., Takagi, T.: Improving key mismatch attack on newhope with fewer queries (2020), `https://eprint.iacr.org/2020/585`.

[23] Qin, Y., Cheng, C., Ding, J.: A complete and optimized key mismatch attack on nist candidate newhope. In: European Symposium on Research in Computer Security. pp. 504–520. Springer (2019)

[24] Qin, Y., Cheng, C., Ding, J.: An efficient key mismatch attack on the nist second round candidate kyber. IACR Cryptology ePrint Archive **2019**, 1343 (2019)

[25] Qin, Y., Cheng, C., Zhang, X., Pan, Y., Hu, L., Ding, J.: A systematic approach and analysis of key mismatch attacks on cpa-secure lattice-based nist candidate kems. Cryptology ePrint Archive, Report 2021/123 (2021), `https://eprint.iacr.org/2021/123`

[26] Rescorla, E.: The transport layer security (tls) protocol version 1.3. Tech. rep. (2018), `http://www.rfc-editor.org/info/rfc8446`.

[27] Shor, P.W.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM review **41**(2), 303–332 (1999)