A Wrap Error Attack against NTRUEncrypt

Tommi Meskanen

Department of Mathematics, University of Turku
FIN-20014, Turku, Finland
and
Turku Centre for Computer Science TUCS
e-mail: tommes@utu.fi

Ari Renvall

Department of Mathematics, University of Turku
FIN-20014, Turku, Finland
e-mail: ariren@utu.fi

Turku Centre for Computer Science
TUCS Technical Report No 507
January 2003
ISBN 952-12-1119-9
ISSN 1239-1891

Abstract

We present a chosen plaintext attack on the NTRU encryption system. We assume that the attacker can detect wrap errors, that the blinding polynomial is generated from three parts (as specified in the standards) and that the attacker has a big database of carefully selected plaintexts. The attack is based on the fact that wrap errors occur more frequently if blinding polynomials with larger coefficients are used.

Keywords: cryptography, chosen plaintext attack, NTRU

TUCS Laboratory
Discrete Mathematics for Information Technology Laboratory

1 Introduction

The NTRU cryptosystem was first introduced in 1998 [HPS]. Since then, several minor modifications have been introduced, but the main ideas have remained the same. For different versions of NTRU see [NTRU]. At the moment NTRU is also being standardized, see [EESS] and [P1363.1]. We follow the system and notations as given in [EESS].

The peculiar property of NTRU is that decryption does not always succeed. There is a possibility that to decrypt one ciphertext several decryption attempts have to be made. It might even happen that decryption fails completely. Fortunately the probabilities of these events are so small that normally it does not matter too much. However, the security of an earlier version of NTRU was susceptible of this property. In [JJ] Jaulmes and Joux presented a chosen-ciphertext attack, where the attacker changed a legal cryptotext a little to observe if the decryption still succeeded. Based on that information the attacker then could quite easily find out information about the private key.

After attacks like this NTRU was changed. It now includes what they call the Fujisaki-Okamoto Self-Referential Technique to block cryptotexts that are not rightfully generated.

Our attack is somewhat similar to that of [JJ], as also we try to find the private key by observing whether the decryptor phases problems during decryption. However, we use only legally produced ciphertexts.

The paper is organized as follows. In chapter 2 we describe the NTRU encryption system. Special emphasis is put to features relevant from the point of view of our attack. Chapter 3 then describes the attack. An example of realistic size is given in chapter 4. In chapter 5 we mention one way to improve the attack and in chapter 6 we list some final remarks.

2 The NTRUEncrypt System

The basic objects of NTRU are polynomials in the ring $R = \mathbf{Z}[x]/(x^N - 1)$. These polynomials are frequently reduced modulo p and q, the small and large modulus. The large modulus q is an integer, so reduction modulo q is performed by reducing the coefficients of the polynomial modulo q. The small modulus p is a polynomial $p = 2+x$, and it is easy to see that representatives modulo p are (almost) exactly the binary polynomials in R.

Polynomials with "small" non-negative coefficients play an important role in NTRU. Let $a(x) = \sum a_i x^i$ and denote

$$\begin{aligned} T(d) =& \{a(x) \in R \mid a_i \geq 0, \ \sum a_i = d\}, \\ T_B(d) =& \{a(x) \in R \mid a_i \in \{0,1\}, \ \sum a_i = d\} \end{aligned}$$

Key generation

In NTRU N, p and q are system wide parameters, and thus public information.

The private key $f \in R$ is of the form $f = 1 + p * F$, where the standard gives two options for F. Either it is a randomly selected binary polynomial from $T_B(d_f)$, or $F = f_1 * f_2 + f_3$ with each $f_i \in T(d_{fi})$. From the point of view of our attack it does not matter which option is selected.

The public key h is obtained by selecting the polynomial g from $T_B(d_g)$ and computing $h = f^{-1} * g \pmod q$. The polynomial g is not needed after the key generation, and thus needs not to be remembered. Despite this, our attack aims to reveal g, after which the private key f can be computed from g and h.

The generation methods of f and g, as well as the values of d_f (or d_{fi}) and d_g are public information.

Encryption

NTRU is a probabilistic public key cryptosystem, hence one plaintext message has several possible encryptions.

Encryption of message m is performed by first selecting a *message representative* i and a *blinding polynomial* r, and then computing the ciphertext

$$e = i + r * p * h \pmod q.$$

The selection of i and r is performed by first choosing some random data b and then computing $i = \varphi(m, b)$ and $r = \rho(m, b)$. Without going into details, i will be a random looking binary polynomial. Also, the mapping φ is efficiently invertible: given i it is possible to compute m and b. There are options for the generation of r. Either it will be a binary polynomial from $T_B(d_r)$, or it will be computed as $r = r_1 * r_2 + r_3$, where each $r_i \in T(d_{ri})$. In any case r will be a polynomial from $T(d_r)$. Our attack will assume the latter option.

Decryption

Decryption starts by computing

$$a = f * e \pmod q.$$

It is easily verified that $a = \alpha \pmod q$, where

$$\alpha = f * i + p * r * g.$$

Moreover, since all polynomials involved are "small", it is very likely that all coefficients of α lie within a certain interval of length q. If this is the case, the decryptor can reduce the coefficients of a to this interval and obtain α. As $f = 1 \pmod p$, reduction modulo p then gives i.

However, it is possible that the decryptor will not get the correct α. Then there are two alternatives.

- The coefficients of a were reduced to a wrong interval of length q (*wrap failure*).
- The difference between the smallest and largest coefficient of α is larger than q (*gap failure*).

Whether either of these cases has happened is checked by re-encryption: from the obtained message representative i' one computes m' and b', and further $r' = \rho(m', b')$. Then from i' and r' one gets the encryption e'. If $e = e'$ then also $i = i'$ and decryption has succeeded. If $e \neq e'$, then either a wrap or a gap failure has occurred, or e was not a valid encryption at all. The decryptor must assume that the reason was a wrap failure. To recover (s)he reduces the coefficients of a to a different interval, until decryption succeeds (or he decides that it was not a wrap failure and quits).

The candidate for the proper interval is first selected as $[A, A+q)$, where A (or actually $A + q/2$) is the (expected) average coefficient of α. This is easily obtained if $\alpha(1)$ is known. Clearly $\alpha(1) = f(1)i(1) + p(1)r(1)g(1)$, where only $i(1)$ is unknown. But $\alpha(1) = a(1) \pmod q$, so $i(1) \pmod q$ can be computed. As i is a random binary polynomial, it is very probable that $N/2 - q/2 \leq i(1) < N/2 + q/2$. Making this assumption one gets the average decryption coefficient A and the first guess on the proper interval. A wrap (or gap) error occurs only if at least one coefficient differs by $q/2$ from the average.

There are at least two reason to adopt the "check back" decryption algorithm. Decryption always produces some candidate message representative

i'. If a wrap or gap failure has occurred, then i' is incorrect. Using the method above one can check whether this is the case. Secondly, this method guarantees that the encryptor can create a valid cryptotext e only if he knows the corresponding plaintext m. This *plaintext awareness* property is advantageous from the point of view of hindering some most powerful cryptographic attacks, such as adaptive chosen cryptotext attacks [Ble]. And specifically, it is a valid counter-measure against the reaction attack presented in [JJ].

3 The Attack

Our attack is based on the observation that blinding polynomials r with big coefficients generate wrap errors more frequently than polynomials with small coefficients. Therefore, by a careful selection of pairs (m, b) (which determine r), one can make the wrap probability rather big. And the probability gives us useful information on the private key.

Throughout this chapter we assume that the parameter set ees251ep1 of [EESS] is used. Thus, $N = 251$, $p = 2 + x$ and $q = 128$. The private key f is of the form $f = 1 + p * F$, where $F \in T(72)$ (and thus $f \in T(217)$). The polynomial g is binary with 72 1's: $g \in T_B(72)$.

We assume that the blinding polynomial is generated from three parts: $r = r_1 * r_2 + r_3$, where each $r_i \in T(8)$ (algorithm 3.3.3.2 in [EESS]). Most coefficients of r_i's are therefore 0, only few equal 1. Also a 2 (or bigger) is possible, but they are few and far between. It follows that also the coefficients of r are very small: $r \in T(72)$. Most coefficients are 0's and 1's (as the average is 72/251). However, also larger values appear; this time more frequently than in r_i's. The main tools in our attack are polynomials which have a suitably big number of "large" coefficients (≥ 4 or 5, for instance).

As noted earlier, the wrap failure occurs if at least one coefficient of $\alpha = f * i + p * r * g$ differs by $q/2$ from the average. Our goal is to

- Generate such pairs (m, b) that the wrap probability correlates with the coefficients of $p * g$;
- Increase the wrap probability to such a level that the differences in the probabilities are efficiently detectable.

These goals are achieved by generating pairs (m, b) such that the resulting blinding polynomial r has some large coefficients. In the following we denote $r = \sum r_i x^i$, $g = \sum g_i x^i$ and $\beta = p * g = \sum \beta_i x^i$. Clearly, $\beta_i \in \{0, 1, 2, 3\}$, and if we learn β_i then both g_i and g_{i-1} are revealed. For example, if $\beta_i = 3$

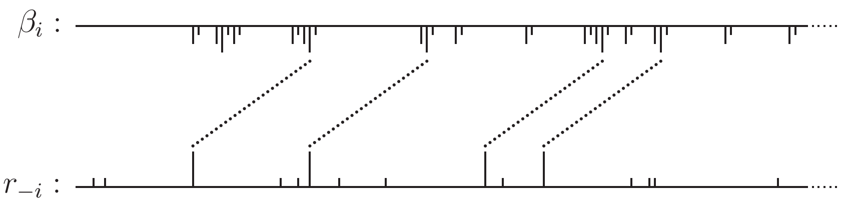

Figure 1: The four big coefficients of r match the big coefficients of β resulting a higher wrap error frequency.

then $g_i = g_{i-1} = 1$. Also, the 72 indexes i for which $\beta_i \geq 2$ are exactly those for which $g_i = 1$.

We increase the wrap probability by trying to make some coefficient of $\beta * r$ (and hence also α) exceptionally large. Suppose that $\beta_{u+i_j} = 3$ and that r_{v-i_j} are large for some indexes u, v and $i_1, \ldots, i_s$ (additions in sub indexes are reduced modulo N). Then all of the large coefficients r_{v-i_j} contribute partly to the same coefficient of $\beta * r$. More specifically, the coefficient of x^{u+v} in $\beta * r$ is at least $3 \cdot \sum_{j=1}^{s} r_{v-i_j}$. As the rest non-zero coefficients of r contribute to random terms in $\beta * r$, it is obvious that wrap failures become more frequent than normally.

The attack consists of four steps. The first step attempts to locate four 3's of β. After the second step we should have learned 15 3's of β. In the third step we spot the rest of β's coefficients that are at least 2, thus revealing g. The final step consists of counting the private key f from g and the public key.

In steps 1 and 2 we use blinding polynomials with four coefficients $r_i \geq 4$. For this purpose, denote by $\mathcal{M}^k_{4\times4}(i_0, \ldots, i_3)$ a set of k pairs (m, b) such that for some $u \in \{0, 1, \ldots, 250\}$ the coefficients $r_{u+i_j} \geq 4$ $(j = 0, 1, 2, 3)$, where $r = \rho(m, b)$. In step 3 we use blinding polynomials with one coefficient at least 5. A set of pairs (m, b) resulting in this kind of blinding polynomial is denoted by $\mathcal{M}_{1\times5}$.

All probabilities given below are estimates based on our implementations.

3.1 First step

Our first goal is to locate four big coefficients of $\beta = p * g$ (or actually in one of its cyclic shifts $x^u * \beta$). For this purpose we encrypt messages in the sets $\mathcal{M}^{1000}_{4\times4}(i_0, i_1, i_2, i_3)$. Clearly, the quadruples (i_0, i_1, i_2, i_3) corresponding to four 3's of β induce high wrap probability. In this case the large coefficients of r contribute to some term of $\beta * r$ by $4 \times 4 \times 3 = 48$, and therefore this term of α has a fair chance of exceeding the average by $q/2 = 64$, thus causing a wrap failure. Therefore the set $\mathcal{M}^{1000}_{4\times4}(i_0, i_1, i_2, i_3)$ that most frequently induces wrap failures most probably gives $\beta_{u-i_0} = \beta_{u-i_1} = \beta_{u-i_2} = \beta_{u-i_3} = 3$ (for some index u). The setup is depicted in figure 1.

Fortunately it is not necessary to go through all sets $\mathcal{M}^{1000}_{4\times4}(i_0, i_1, i_2, i_3)$. The average number of 3's in β is $72 \cdot \frac{71}{251} \approx 20$, and therefore a random guess of four indexes has a reasonable chance (roughly 1%) of hitting four 3's in one of β's rotations. Therefore, if we randomly select, say, 1000 quadruples and select the one that causes the most wrap failures, most probably we have found four 3's.

In our tests we made 1000 guesses for (i_0, i_1, i_2, i_3), generated 1000 encryptions for each such quadruple, and observed the number of wrap failures. The probability that this process found four 3's in β turned out to be 90%. In most of the erroneous cases one 3 was in fact 2. With 200 sets of 200 messages the probability of success was $\frac{1}{6}$ and with probability 40% we had found three 3's and a 2.

3.2 Second step

In the first step we learned i_0, i_1, i_2 and i_3 such that $\beta_{u-i_0} = \beta_{u-i_1} = \beta_{u-i_2} = \beta_{u-i_3} = 3$ for some u. Next we exploit these to find more large coefficients of β. More specifically, the goal is that after this step we have found 15 3's (or possibly some 2's among them) in β. Note that the probability that β has at least 15 3's is roughly 97%.

We encrypt the messages in the sets $\mathcal{M}^{1000}_{4\times4}(i_0, i_1, i_2, k)$, $\mathcal{M}^{1000}_{4\times4}(i_0, i_1, i_3, k)$, $\mathcal{M}^{1000}_{4\times4}(i_0, i_2, i_3, k)$ and $\mathcal{M}^{1000}_{4\times4}(i_1, i_2, i_3, k)$ for all $0 \leq k \leq 250$, $k \neq i_j$. The wrap probability depends in this case on the coefficient β_{u-k}: the higher the wrap probability, the bigger β_{u-k} (see figure 2). We make the assumption that the 11 k's with the highest wrap probabilities give 11 big coefficients β_{u-k} (3's and possibly a few 2's among them).

This approach has one problem. It is possible that, for example, for some $v \neq u$ the coefficients of $x^{v-i_0}, x^{v-i_1}, x^{v-i_2}$ in β are also 3's. In this case there is a good chance that β_{u-k} is selected to be big, although it is not. Fortunately we have a good chance of detecting such situations, as then the number of wrap errors generated by $\mathcal{M}^{1000}_{4\times4}(i_0, i_1, i_2, k)$'s differ from other three cases and are more frequent. If this is the case, we can replace $\mathcal{M}^{1000}_{4\times4}(i_0, i_1, i_2, k)$ by some $\mathcal{M}^{1000}_{4\times4}(i_0, i_1, i_4, k)$, where most probably $\beta_{u-i_4} = 3$ based on the other three cases.

If we had found four 3's in step 1 and used the sets of 1000 messages, the sum of 15 indices found in step 2 was in our tests always at least 41. On the other hand, if we had found three 3's and a 2, the sum of 15 indices was at least 41 in half of our tests and at least 37 with probability 90%.

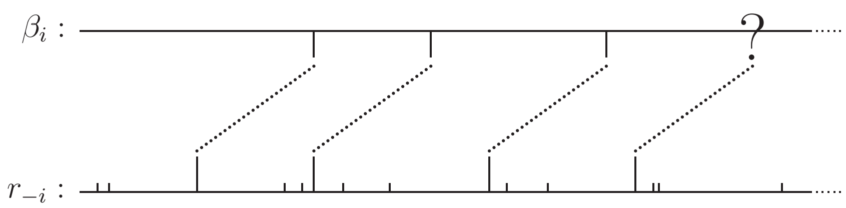

Figure 2: The four big coefficients of r match three known big coefficients of β and the position corresponding to k. The number of wrap errors at step 2 give us information about that coefficient.

Using the sets of 200 messages and four known 3's the sum of the 15 indices was at least 41 with probability 70% and always at least 37. With three 3's and one 2 the sum was at least 37 with probability $\frac{1}{3}$.

3.3 Third step

To start with, we have spotted 15 3's (or 2's) in $\beta * x^u$ for some u. Our strategy is to test the largeness of the remaining terms one at a time. We also test whether the already known big coefficients really are big.

Let β_{u-j} be the coefficient to be examined, and assume that $\beta_{u-i_0}, \ldots, \beta_{u-i_{14}}$ are known to be big. We use blinding polynomials r with one peak: $r_v \geq 5$ for some v. Moreover, we assume that $\sum_{\ell=0}^{14} r_{v-j-i_\ell} \geq 25$. (If $j = i_\ell$ then the term r_v is excluded from the sum.) Assuming that the average of known big coefficients β_{u-i_ℓ} is at least 2.5, then the coefficient of x^{u+v-j} in $r * \beta$ is at least $2.5 \cdot 25 + 5 \cdot \beta_{u-j}$. The consequence is that wrap failures are quite probable, and the probability is strongly influenced by β_{u-j}.

We test each index β_{u-j} by selecting and encrypting (say) 1000 pairs (m, b) such that the resulting blinding polynomial satisfies the conditions above. Then we sort the indexes $u - j$ according to the number of wrap failures. The 72 highest wrap failure rates gives us the 72 big β_{u-j}'s (those that equal 2 or 3). And, as already observed, these values reveal the polynomial $g * x^u$.

Our tests have shown that this step is the most accurate and can correct the possible errors we have made in the previous steps.

When the sum of the 15 indices of step 2 was 41 and we used sets of 1000 messages this step revealed β practically always. With sets of 200 messages we failed to spot in average four 1's in g. In the case where the sum of the 15 indices was 37 and we used sets of 1000 messages, we failed to spot in average one 1 in g.

It seems impractical to construct a sufficiently large database of pairs

(m, b) for every 16-tuple of indexes $u, i_0, \ldots, i_{14}$. Fortunately this is not necessary. It is sufficient to have a database of pairs (m, b) such that the resulting blinding polynomial has the required peak. If the peak value is ≥ 5, then roughly one out of 30 peak polynomials can be used to test β_u w.r.t. indexes $i_0, \ldots, i_{14}$. Hence, a database of 30000 pairs (m, b) that result in a peak polynomial gives roughly 1000 suitable pairs for each to be tested index.

3.4 Computing the private key f

Let us finally show how to compute the private key f from the polynomial g we already learned. Denote $\psi(x) = \frac{x^N - 1}{x - 1} = x^{250} + x^{249} + x^{248} + \cdots + 1$. As ψ decomposes into 5 prime factors of degree 50, with a very high probability $\gcd(h, \psi) = 1$ in $\mathbf{Z}_2[x]$, where $h = f^{-1} * g \pmod{128}$ is the public key. Thus using the Extended Euclidean algorithm we can find out the polynomials H_0, a and b such that

$$h * H_0 - a * \psi = 1 - 2b.$$

We also have

$$\begin{aligned}
h * H_0 &= 1 - 2b \pmod{\psi} \\
h * H_0 * (1 + 2b) &= 1 - 4b^2 \pmod{\psi} \\
h * H_0 * (1 + 2b)(1 + 4b^2) &= 1 - 16b^4 \pmod{\psi} \\
h * H_0 * (1 + 2b)(1 + 4b^2)(1 + 16b^4) &= 1 - 256b^8 \pmod{\psi}.
\end{aligned}$$

The polynomial $H = H_0 * (1 + 2b)(1 + 4b^2)(1 + 16b^4)$ is called the pseudo inverse of h in the ring $\mathbf{Z}_{128}[x]/(x^{251} - 1)$.

We have

$$g * H = f * f^{-1} * g * H = f * h * H = f \pmod{\psi, 128},$$

where f^{-1} is the inverse of f modulo $x^{251} - 1$ in $\mathbf{Z}_{128}[x]$. Therefore

$$f = g * H + c * \psi \pmod{x^{251} - 1, 128}$$

for some $c \in \mathbf{Z}_{128}$. From the condition

$$f(1) = g(1) \cdot H(1) + c \cdot \psi(1)$$

we obtain $c = (217 - 72 \cdot H(1)) \cdot 251^{-1} \pmod{128}$.

3.5 Generating suitable blinding polynomials

As mentioned, we assume that the blinding polynomial r is constructed from three parts: $r = r_1 * r_2 + r_3$, where each $r_i \in T(8)$. The parts r_i are generated (algorithm 3.3.3.2 of [EESS]) by selecting 8 random indexes $j_0, \ldots,$ j_7 (repetitions allowed) and setting $r_i = x^{j_0} + \ldots + x^{j_7}$. In theory the random indexes are not random, they are determined by the message m, random data b and some specified pseudo-random number generator. However, a similar distribution of r's is obtained via random selection (assuming the security of the used PRNG).

In the attack we needed two types of blinding polynomials. The following probabilities are obtained by generating blinding polynomials randomly.

- About one out of 10000 pairs (m, b) results in a blinding polynomial $r = \rho(m, b)$ with four coefficients at least four. Such polynomials are needed for about $8 \cdot 10^5$ quadruples (i_0, i_1, i_2, i_3). The computation of these seems to be the most demanding task of the attack. In fact, the whole database requires approximately 40 giga bytes of memory.

- Roughly one out of 250 pairs (m, b) results in a blinding polynomial $r = \rho(m, b)$ with one coefficient at least five.

It should be noted that if the parts r_i are required to be binary polynomials, it is a lot harder to generate r's with coefficients big enough. Therefore the construction of the databases takes a longer time, or alternatively one could modify the attack to use somewhat smaller blinding polynomials resulting in a lower success probability. Anyway, it would still be possible to apply the attack.

4 An example

In the next we show an example of this attack:

We attack a system with private key f, public key $h = f^{-1} * g \pmod{q}$. Just for reference, we list the coefficients of the polynomials f and $\beta =$

quadruple	wrap errors
21,159,204,205	11
37,53,60,224	10
44,67,88,166	10
68,75,112,221	10
9,118,226,236*	10
1,16,118,172	9
9,53,197,240*	9

Table 1: Step 1. List of best choices for the quadruple corresponding to four 3's. Quadruples marked with a star do not correspond to four 3's in β and choosing one of them would cause some problems for the attack.

$(2+x)*g$ from the smallest to the largest exponent:

f: 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 1, 0, 0, 0, 0, 2, 3, 3, 1, 0, 0, 0, 2, 1, 2, 1, 0, 0, 0, 0, 0, 0, 2, 3, 3,
5, 2, 0, 0, 0, 0, 2, 1, 0, 0, 0, 0, 0, 2, 3, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 1, 0,
0, 0, 0, 0, 0, 2, 1, 0, 0, 2, 3, 3, 1, 2, 1, 2, 3, 3, 3, 3, 1, 0, 0, 2, 5, 4, 1, 0, 0, 0, 0, 0, 2, 1, 2, 3,
1, 0, 0, 2, 1, 2, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 1, 0, 2, 1, 0, 2, 1, 0, 0, 2, 1, 2, 3, 1, 0, 0, 0, 0, 2,
1, 0, 0, 0, 2, 1, 4, 2, 0, 2, 3, 1, 0, 4, 4, 1, 0, 2, 1, 0, 0, 0, 2, 1, 2, 1, 0, 0, 0, 2, 1, 2, 1, 0, 0, 0,
0, 0, 0, 0, 2, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 1, 0, 0, 2, 1, 0, 0, 0, 0, 0, 4, 4, 3, 1, 0, 0, 2, 1, 2, 3,
3, 1, 0, 2, 1, 0, 0, 0, 0, 2, 1, 0, 2, 1, 2, 3, 1, 0, 0, 2, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 1, 0

β: 0, 0, 0, 0, 2, 1, 2, 1, 0, 2, 1, 2, 3, 3, 3, 1, 2, 1, 0, 0, 0, 2, 1, 2, 1, 0, 0, 2, 1, 2, 1, 0, 2, 1, 0, 0,
0, 2, 1, 2, 1, 0, 0, 2, 1, 0, 0, 2, 1, 0, 2, 3, 1, 0, 0, 2, 3, 3, 3, 3, 1, 0, 0, 2, 1, 2, 3, 1, 0, 0, 0, 0,
0, 2, 1, 0, 0, 2, 3, 1, 2, 3, 1, 2, 1, 0, 2, 1, 0, 0, 0, 2, 1, 0, 2, 1, 0, 0, 2, 1, 0, 2, 1, 2, 1, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 2, 1, 0, 0, 0, 2, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 1, 0, 2, 1, 2, 1, 0, 0, 2, 1,
2, 3, 3, 1, 0, 0, 0, 0, 2, 1, 2, 3, 3, 3, 1, 0, 0, 0, 0, 0, 2, 1, 2, 1, 0, 0, 0, 0, 0, 0, 0, 0, 2, 3, 1, 0,
0, 0, 0, 2, 3, 1, 2, 1, 0, 0, 0, 0, 0, 0, 0, 2, 3, 1, 0, 2, 3, 1, 0, 0, 0, 0, 0, 0, 0, 0, 2, 1, 0, 0, 0, 2,
1, 2, 1, 0, 2, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 1, 0, 0, 0, 0, 2, 1, 0, 2, 1, 0, 0, 0, 0, 0, 0, 2, 1

Step 1: We randomly select 1000 quadruples and sets $\mathcal{M}_{4\times4}^{1000}(i_0, i_1, i_2, i_3)$. After all one million encrypted and decrypted messages the quadruples that generated the most wrap errors are listed in table 1. We select the best quadruple $(21, 159, 204, 205)$ that resulted in 11 wrap errors. We guess that for some index u we have $\beta_{u-21} = \beta_{u-159} = \beta_{u-204} = \beta_{u-205} = 3$. The attacker does not know for sure if this is true, but we see that it is for $u = 217$.

Step 2: Next, for indexes $k \in \{0, 1, \ldots, 80\}$, we encrypt messages in the sets $\mathcal{M}_{4\times4}^{1000}(21, 159, 204, k)$, $\mathcal{M}_{4\times4}^{1000}(21, 159, 205, k)$, $\mathcal{M}_{4\times4}^{1000}(21, 204, 205, k)$ and $\mathcal{M}_{4\times4}^{1000}(159, 204, 205, k)$ and observe the number of wrap errors. Results can be seen in table 2. We have included (in the last column) the coefficients

k	wrap errors	
0	4+2+2+7=15	2
1	1+1+7+3=12	1
2	2+2+4+7=15	2
3	0+1+3+0=4	0
4	0+0+1+3=4	0
5	1+1+1+1=4	0
6	1+1+1+3=6	1
7	6+0+2+7=15	2
8	0+1+2+7=10	0
9	1+1+1+2=5	0
10	0+1+2+2=5	0
11	2+0+0+2=4	0
12	2+0+0+2=4	0
13	0+0+2+2=4	0
14	0+1+2+1=4	0
15	0+0+4+4=8	0
16	2+5+5+2=14	1
17	8+5+6+6=25	3
18	2+7+6+6=21	2
19	1+1+3+5=10	0
20	2+4+1+9=16	1
21	n/a	3
22	8+2+13+4=27	2
23	2+2+5+1=10	0
24	2+0+0+3=5	0
25	2+2+1+1=6	0
26	1+0+4+2=7	0

k	wrap errors	
27	0+2+2+0=4	0
28	2+3+4+3=12	0
29	1+0+2+4=7	0
30	1+0+3+3=7	1
31	4+4+7+4=19	2
32	1+2+3+6=12	1
33	6+8+6+7=27	3
34	2+3+9+1=15	2
35	1+1+3+1=6	0
36	0+0+4+2=6	0
37	0+0+1+1=2	0
38	0+1+1+4=6	0
39	0+4+2+6=12	1
40	10+5+10+15=40	3
41	3+3+4+10=20	2
42	1+2+7+9=19	0
43	0+0+4+3=7	0
44	1+0+2+6=9	0
45	1+0+2+2=5	0
46	0+2+0+1=3	0
47	0+1+1+3=5	0
48	0+2+0+4=6	0
49	1+1+2+2=6	0
50	2+2+3+4=11	1
51	7+1+3+5=16	2
52	2+0+10+2=14	1
53	1+1+7+5=14	2

k	wrap errors	
54	0+0+2+0=2	0
55	0+0+0+3=3	0
56	0+0+1+5=6	0
57	1+1+2+0=4	0
58	2+1+2+1=6	0
59	1+4+3+1=9	1
60	4+7+12+7=30	3
61	8+8+11+5=32	3
62	4+9+21+14=48	3
63	1+3+12+1=17	2
64	1+1+3+0=5	1
65	0+2+5+5=12	2
66	2+0+3+4=9	0
67	0+1+0+7=8	0
68	1+2+3+0=6	0
69	0+1+0+0=1	0
70	0+3+6+1=10	1
71	6+6+4+8=24	3
72	6+9+7+10=32	3
73	4+5+9+2=20	2
74	1+2+3+1=7	1
75	0+4+5+6=15	2
76	3+0+9+1=13	0
77	0+1+0+2=3	0
78	5+3+4+5=17	1
79	1+5+3+2=11	2
80	3+3+3+3=12	1

Table 2: Step 2. Wrap error counts for the first 81 k's. In the last column there is the corresponding coefficient of x^{217-k} in β.

β_{217-k}. Of course, the attacker does not know them (yet), but it clearly demonstrates the correlation between them and the number of wrap errors induced. We add the 11 k's with largest wrap error rates to the four indexes picked in step 1. The 15 indexes are then

$$17, 18, 21, 22, 33, 40, 41, 60, 61, 62, 71, 72, 159, 204, 205.$$

Note that we only examined about one third of the indexes, and therefore $\beta_{217-k} < 3$ for some picked k's. But it does not matter, as step 3 still finds all big coefficients of $\beta * x^{217}$. Naturally more reliable results are obtained if we go through all indexes.

Step 3: The next task is to find 1000 suitable pairs $(m, b) \in \mathcal{M}_{1\times 5}$ for each index $k \in \{0, 1, \ldots, 250\}$ (as described earlier). Again, we observe the number of wrap errors encountered (table 3). We make the guess that the

72 largest wrap counts – anything over 80 here – correspond to 2's and 3's in some rotation of β. If this guess is correct, we have a rotation of the polynomial g. In our case the guess is indeed true: we have learned $g * x^{217}$. However, the attacker must proceed to the next step to check the validity of the obtained g.

Step 4: Let g' be the polynomial found in step 3. The final task is to compute f' such that $f'^{-1} * g' = h \pmod{q}$, where h is the public key. Applying the method described earlier we obtain $f' = f * x^{217}$, and the key is broken.

In this example we have not used the actual messages that generate useful blinding polynomials. Instead, for steps 1 and 2 we have selected random binary polynomials with weight 56 and added 4 to required 4 coefficients. For step 3 we have selected random binary polynomials with weight 45, added 5 to one coefficient and distributed the rest of the weight, 22, randomly between the specified 15 coefficients.

5 A method to create more wrap errors

The attack we have described would be more efficient if wrap failures occurred more frequently. And, in fact, we can make this happen. As noted, the method to generate the message representative i from the message m and randomness b is invertible. Therefore we can select messages m such that, for example, $i(1) > N/2 + q/2$. The consequence is that now the average coefficient of α, i.e. A, is computed to be too small (by 110 if $N = 251$), so that wrap errors are inevitable. But the decryptor is still able to correct the error if she tries to vary A as stated in the standard. If A is computed to be too small, the number of corrections needed depends on the largest coefficient of α.

Our described attack exploited the connection of the largest coefficient of α and a guess on the polynomial g. Then, a large coefficient was detected by a wrap error, which occurred quite rarely. Using this new approach we get information on the largest coefficient in every decryption process, provided that we can find out how many corrections to A were made before success. This information can be obtained by measuring the time spent in decrypting a message.

We expect (but have not tested), that this sketched modification of our attack should be significantly more effective.

Although not specified in the standards, NTRU recommends [HNPSW] that the polynomial g does not have any two consecutive coefficients equal to 1. Therefore no 3 can occur in $p * g$, which makes our basic attack less

k	errors	
0	105	2
1	46	1
2	85	2
3	19	0
4	16	0
5	27	0
6	46	1
7	101	2
8	43	0
9	9	0
10	16	0
11	14	0
12	19	0
13	29	0
14	15	0
15	19	0
16	72	1
17	192	3
18	140	2
19	39	0
20	45	1
21	208	3
22	131	2
23	25	0
24	13	0
25	23	0
26	21	0
27	18	0
28	20	0
29	20	0
30	44	1
31	93	2
32	59	1
33	216	3
34	105	2
35	23	0
36	23	0
37	18	0
38	19	0
39	52	1
40	217	3
41	129	2

k	errors	
42	48	0
43	40	0
44	24	0
45	13	0
46	18	0
47	17	0
48	36	0
49	20	0
50	44	1
51	125	2
52	62	1
53	105	2
54	22	0
55	14	0
56	25	0
57	19	0
58	24	0
59	53	1
60	210	3
61	222	3
62	228	3
63	116	2
64	38	1
65	100	2
66	23	0
67	16	0
68	18	0
69	22	0
70	47	1
71	201	3
72	202	3
73	116	2
74	46	1
75	107	2
76	30	0
77	18	0
78	48	1
79	101	2
80	45	1
81	91	2
82	26	0
83	53	1

k	errors	
84	91	2
85	48	0
86	41	0
87	49	0
88	23	0
89	22	0
90	21	0
91	17	0
92	15	0
93	26	0
94	13	0
95	44	1
96	101	2
97	30	0
98	21	0
99	11	0
100	43	1
101	108	2
102	27	0
103	16	0
104	19	0
105	23	0
106	21	0
107	20	0
108	19	0
109	21	0
110	16	0
111	8	0
112	24	0
113	44	1
114	94	2
115	60	1
116	101	2
117	25	0
118	41	1
119	117	2
120	29	0
121	16	0
122	42	1
123	104	2
124	26	0
125	41	1

k	errors	
126	93	2
127	13	0
128	11	0
129	18	0
130	60	1
131	97	2
132	22	0
133	54	1
134	105	2
135	55	1
136	200	3
137	122	2
138	52	1
139	197	3
140	137	2
141	26	0
142	16	0
143	50	1
144	110	2
145	18	0
146	14	0
147	19	0
148	14	0
149	27	0
150	70	1
151	197	3
152	117	2
153	51	1
154	108	2
155	40	0
156	19	0
157	43	1
158	203	3
159	199	3
160	214	3
161	205	3
162	113	2
163	17	0
164	22	0
165	63	1
166	223	3
167	116	2

k	errors	
168	30	0
169	38	1
170	118	2
171	26	0
172	25	0
173	68	1
174	116	2
175	30	0
176	22	0
177	47	1
178	97	2
179	46	1
180	94	2
181	22	0
182	23	0
183	17	0
184	53	1
185	116	2
186	37	0
187	50	1
188	98	2
189	52	1
190	121	2
191	25	0
192	19	0
193	72	1
194	121	2
195	67	1
196	109	2
197	32	0
198	19	0
199	16	0
200	33	1
201	88	2
202	55	1
203	208	3
204	221	3
205	227	3
206	136	2
207	40	1
208	115	2
209	28	0

k	errors	
210	42	1
211	93	2
212	49	1
213	95	2
214	34	0
215	29	0
216	23	0
217	23	0
218	42	1
219	81	2
220	29	0
221	19	0
222	14	0
223	13	0
224	22	0
225	29	0
226	45	1
227	92	2
228	22	0
229	48	1
230	105	2
231	40	0
232	24	0
233	23	0
235	13	0
235	50	1
236	91	2
237	28	0
238	12	0
239	19	0
240	18	0
241	11	0
242	14	0
243	18	0
244	18	0
245	10	0
246	19	0
247	49	1
248	95	2
249	27	0
250	56	1

Table 3: Step 3. Wrap error counts for all k's. In the last column there is the corresponding coefficient of x^{217-k} in β.

effective. However, it seems that the above modified approach would still work.

The starting point for this modification of the attack was that it is easy to find messages with representatives of large weight. There are also several other attacks that are based on this property, see [NP], [HHKH], [Pro].

6 Remarks and Conclusion

We conclude this article by listing some remarks on the attack.

The starting assumption in our attack is that wrap failures are detectable for the attacker. An obvious method for this is to measure the time decryption takes. To make the attack impossible, one could make the decryption algorithm constant time, as if wrap failures occurred every time. A more efficient way would be to simulate wrap failures every now and then. Also, the decryption machinery could count the number of wrap failures it faces. If they occur too frequently, the key should be changed.

The counter-measure we would like to recommend is to always require that the blinding polynomial r is binary. Specifically r should not be constructed from two or more "smaller" parts.

The generation of the required databases $\mathcal{M}_{4\times 4}$ and $\mathcal{M}_{1\times 5}$ can (and must) be done off-line. The workload of generating these sets could be divided to the whole internet, for example. The same library can be used to break any NTRU key.

Our attack is by no means optimized; most probably similar ideas can be used to extract g more efficiently. For example, one could use blinding polynomials with different patterns or even larger coefficients. Also we have not taken into account that the coefficient of β obey a certain pattern. For example, a 3 is always preceded by another 3 or a 2. And finally, it is not necessary to get g exactly. If we have an pretty close approximation of it, we can use some lattice reduction methods to extract f (and g).

We should also mention that although the attack is designed against a particular set of parameters, it can be modified to beat efficiently any parameter set, as long as one can generate blinding polynomials with large coefficients. The complexity seems to be linear with respect to N, the degree of polynomials. However, the number of quadruples increases more rapidly, so construction of the databases becomes more troublesome.

References

[Ble] Daniel Bleichenbacher, Chosen Ciphertext Attacks against Protocols Based on the RSA Encryption Standard PKCS#1, Advances in Cryptology - Crypto '98, Springer-Verlag, 1992.

[EESS] Efficient Embedded Security Standard (EESS) #1: Draft 4, Consortium for Efficient Embedded Security, March, 2002.

[HHKH] Daewan Han, Jaewoo Han, Daesung Kwon and Jin Hong, A New Chosen Ciphertext Attack against NTRU, Unpublished manuscript.

[HPS] Jeffrey Hoffstein, Jill Pipher and Joseph H. Silverman, NTRU: A Ring-Based Public Key Cryptosystem, Algorithmic Number Theory (ANTS III), Portland, OR, June 1998, J.P. Buhler (ed.), Lecture Notes in Computer Science 1423, Springer-Verlag, Berlin, 1998, 267-288.

[HNPSW] Nick Howgrave-Graham, Phong Q. Nguyen, David Pointcheval, Ari Singer and William Whyte, Padding Schemes and Decryption Failures in NTRUEncrypt, Unpublished manuscript.

[P1363.1] IEEE P1363.1. Standard Specification for Public-Key Cryptographic Techniques Based on Hard Problems over Lattices.

[JJ] Éliane Jaulmes and Antoine Joux, A Chosen-Ciphertext Attack against NTRU, Advances in Cryptology - Crypto 2000, Lecture Notes in Computer Science 1880, Springer-Verlag, Berlin, 2000, 20-35.

[NP] Phong Q. Nquyen and David Pointcheval, Analysis and Improvements of NTRU Encryption Paddings, Advances in Cryptology - CRYPTO 2002, Lecture Notes in Computer Science 2442, Springer-Verlag, Berlin, 2002, 210-225.

[NTRU] www.ntru.com.

[Pro] John Proos, Imperfect Decryption and an Attack on the NTRU Encryption Scheme, Technical report, available at http://www.cacr.math.uwaterloo.ca/tech_reports.html.

Turku Centre for Computer Science
Lemminkäisenkatu 14
FIN-20520 Turku
Finland

http://www.tucs.fi

University of Turku
- **Department of Information Technology**
- **Department of Mathematics**

Åbo Akademi University
- **Department of Computer Science**
- **Institute for Advanced Management Systems Research**

Turku School of Economics and Business Administration
- **Institute of Information Systems Science**