NTRU Cryptosystems Technical Report Report # 004, Version 2: A Meet-In-The-Middle Attack on an NTRU Private Key

Nick Howgrave-Graham, Joseph H. Silverman, William Whyte

NTRU Cryptosystems,
5 Burlington Woods,
Burlington, MA 02144

Abstract. In this report we describe a meet-in-the-middle attack on an NTRU private key. If the private key is chosen from a sample space with 2^M elements, then the security level of the cryptosystem is no more than $2^{M/2}$. We also describe variants of this attack applicable to product form NTRU keys.

Acknowledgement. We would like to thank Andrew Odlyzko for the original description of a meet-in-the-middle attack on NTRU private keys.

1 A Meet-In-The-Middle Attack on Random Binary Keys

1.1 Algorithm

The NTRU cryptosystem is described in [4] and subsequent papers. Here we give only a brief outline.

We begin with some notation:

- N, d, q Integer parameters used to create an NTRU cryptosystem. To make the explanation clearer, we will assume N and d are even; the modifications for odd values are easy. We also assume that q is a power of 2; the modification for other values is also easy.
- f The private key, chosen consisting of d ones and $N - d$ zeros.
- g Used to form the public key, chosen with binary coefficients.
- h The public key $h \equiv f^{-1}g \pmod{q}$, where multiplication is defined as convolution multiplication. For more details of this process, see [4, 2]
- k Integer chosen by the attacker so that 2^k is larger than $\binom{N/2}{d/2}$ (say by factor of 100).

The idea is to search for f in the form $f_1||f_2$, where f_1 and f_2 are each of length $N/2$ with $d/2$ ones and "$||$" denotes concatenation, using the property that

$$\begin{aligned} f * h &= g \pmod q \\ \Rightarrow (f_1||f_2) * h &= g \pmod q \\ \Rightarrow f_1 * h &= g - f_2 * h \pmod q \\ \Rightarrow (f_1 * h)_i &= \{0,1\} - (f_2 * h)_i \pmod q \forall i \end{aligned}$$

where the a_i notation denotes the ith entry in a.

In fact, although f itself may not have the property that half its ones fall in the first $N/2$ entries, we know that there is at least one rotation of f which has this property[1] and that any rotation of f will be effective as the private key.

The steps in the attack are as follows:

Enumerate f_1 — Enumerate the vectors f_1. (These are of length $N/2$, but we identify them with the length-N vectors formed by appending $N/2$ zeroes.) This takes $\binom{N/2}{d/2}$ steps. We put each f_1 into a "bin" based on the most significant bit of the first k coordinates of $f_1 * h \pmod q$. Each bin is then referenced by $\{0,1\}^k$, and there are 2^k bins, of which about $\binom{N/2}{d/2}$ will be occupied. (To be precise, the fraction of occupied bins will be about $e^{-\binom{N/2}{d/2}/2^k}$, and some bins will contain multiple f_1s).

Enumerate f_2 — Enumerate the vectors f_2, which also takes $\binom{N/2}{d/2}$ steps. (These vectors are of length $N/2$, but we identify them with the length-N vectors formed by *pre*pending $N/2$ zeroes.) Check each f_2 to see if it corresponds to an occupied bin. Here, we know that if we have the correct f_1 and f_2, then $(f_1 * h)_i = \{0,1\} - (f_2 * h)_i \pmod q \forall i$. We therefore check for occupation not merely the bin given by the most significant bits of the first k coefficients of $-f_2 h \pmod q$, but also the bins given by the flips of all those most significant bits that would be changed by adding 1 to the corresponding coefficient of $-f_2 h \pmod q$.

As an example, take $N = 4$ and $q = 8$.

- If $f_1 * h \pmod q = [7,2,3,5]$, then f_1 is stored in the bin marked [1001].
- If $-f_2 * h \pmod q = [6,2,1,5]$, then f_2 is checked against only the bin [1001].
- If $-f_2 * h \pmod q = [7,2,3,5]$, then f_2 is checked against the bins [1001], [0001], [1011], [0011].

[1] Proof: Let $D = d/2$. Say f has $D + a$ ones in the first $N/2$ entries, $D - a$ in the second. Rotating f by one position can only change the number of ones in the first $N/2$ entries by 0, 1 or -1. After $N/2$ rotations by one position, the first $N/2$ entries will have $D - a$ ones in them. Therefore, at some point, the number of ones in the first $N/2$ entries must have been exactly D.

Search for matches — When f_2 hits an occupied bin, take the (length-$N/2$) f_1 from the bin and form the candidate value for f as $f_1||f_2$. Check if $f * h \pmod q$ is binary. If it is, terminate and return f. Otherwise, proceed to the next f_2. If the bin contains more than one f_1, perform this check for each f_1 in the bin.

1.2 Analysis of the Algorithm: Running Time and Memory

Let τ_c be the time for a convolution, ie the time to calculate $f_1 * h \pmod q$. The time to calculate $f * h \pmod q$ will be no more than $2\tau_c$. Let τ_l be the time for a lookup, ie the time to find the contents of bin i, or to write to bin i, given i. We will use these quantities to get upper bounds for the running time of the algorithm.

The expected time to run the first part of the attack, enumerating f_1, will be no more than

$$\tau_1 = \binom{N/2}{d/2}(\tau_c + \tau_l).$$

The expected time to run the second part, enumerating f_2 and performing the check, will be no more than

$$\begin{aligned}\tau_2 = \#(f_2) * \\ &(\tau_c + \\ &\text{(Expected Different Bins per } f_2) * \tau_l + \\ &\text{(Expected Hits per } f_2) * \tau_c) \\ = \binom{N/2}{d/2}&\left(\tau_c + \frac{2k}{q}\tau_l + \frac{\binom{N/2}{d/2}}{2^k}\tau_c\right).\end{aligned}$$

By increasing k, we can decrease the expected running time of this step, at the cost of increasing memory use.

The amount of memory required, μ, is highly dependent on the storage and retrieval algorithms used. For example, memory need not be allocated for a bin before it is used if the bins are held in a linked list structure; the resulting reduction in memory required will be offset by the increased amount of time required to add bins and to retrieve the data from the bins. However, taking μ_f to be the size of one stored f_1 plus header information, and μ_o to be the overhead required for the storage infrastructure, we can say

$$\mu \approx \binom{N/2}{d/2}\mu_f + \mu_o.$$

It is probable that μ_o increases with k, but not exponentially with k, and that μ_f increases with k, but not faster than k.

1.3 Improvements

Can we reduce these requirements further? We note that clever scheduling of the enumeration of the f_1, f_2s will enable the attacker to calculate almost every

$f_1 * h \pmod q$ by adding one rotation of h to and subtracting one rotation of h from the previous value of $f_1 * h$ (and similarly for f_2). This will reduce the intial τ_c term in τ_1, τ_2 to about $2\tau_c/(d/2)$.

We also note that if instead of storing only f_1 in the first stage of the attack, the attacker stores $(f_1, f_1 * h \bmod q)$, then it is not necessary to calculate $f * h \bmod q$ in the second stage of the attack: the attacker already knows $-f_2 * h \bmod q$, and can calculate $f_1 * h - f_2 * h \bmod q$ by a single subtraction, taking time approximately τ_c/d.

Finally, we note that the figures above assume there is only one possible $(f_1 || f_2)$ that gives a rotation of f with $d/2$ ones in each of the first. In fact, we have run experiments showing that the number of rotations of f of the correct form is typically more than $\sqrt{N}$. We may use this to improve the algorithm as follows: instead of searching first on f_1, then on f_2, search on them simultaneously, storing each f_1 in a single bin and each f_2 in approximately $(2N/q)$ bins. If there are r rotations of the correct form, we expect a collision between an f_1 and an f_2 corresponding to the same rotation after we have picked approximately $1/\sqrt{r}$ of all of the f_1, f_2 that correspond to a substring of any correct rotation of f. The expected running time becomes

$$\begin{aligned}\tau_2 = \sum_i (\tau_c + \\ &(\text{Expected Different Bins per } f_1) * \tau_l + \\ &(\text{Expected Different Bins per } f_2) * \tau_l + \\ &(\text{Expected Hits on picking } i\text{th } f_1) * \tau_c) + \\ &(\text{Expected Hits on picking } i\text{th } f_2) * \tau_c) \\ \approx \frac{\binom{N/2}{d/2}}{\sqrt{r}} \left(\tau_c + \left(1 + \frac{2N}{q}\right)\tau_l\right) + \frac{C}{2^k}\sum_i (\text{Hits})_i \ .\end{aligned}$$

By choosing k such that 2^k is large relative to $\binom{N/2}{d/2}/\sqrt{r}$, we can reduce the number of false positives such that the time used to check them is a small fraction of the time taken to perform the enumeration. This allows us to ignore the second term above. The running time and the storage are then constant multiples of

$$\frac{\binom{N/2}{d/2}}{\sqrt{r}} \ .$$

The value of r will vary between private keys, but it will certainly be no bigger than N. Our final estimate of the running time and storage space required for this method is therefore

$$\frac{\binom{N/2}{d/2}}{\sqrt{N}} \ .$$

1.4 Alternative Algorithms

We next consider alternative approaches to the one outlined above.

For example, an attacker may choose to assume that a run of z zeroes occurs at the start of one rotation of f. We know that z will be at least $\lceil N/df \rceil - 1$, and typically it could be much more than this.

The attacker enumerates randomly through the f_1s which have $d/2$ ones and length $N - z$. In order to succeed, he must pick f_1', f_1'', such that $f_1' + f_1'' = f$. We can use a birthday paradox like argument to estimate the probability of this happening, as follows. Each f_1 picked defines a "dual", $f - f_1$. The "collisions" of interest do not arise from picking a given f_1 twice, but from picking both an f_1 and its dual. However, since each f_1 defines a single dual, the chance of a collision with a dual is the same as the chance of a collision with an f_1.

There are

$$\binom{d}{d/2}$$

substrings of length $d/2$ contained in a single rotation of f. We expect to have to pick the square root of this number before getting a collision. The expected running time of this approach is therefore

$$\frac{\binom{N-z}{d/2}}{\sqrt{\binom{d}{d/2}}} \ .$$

Depending on the expected value of z, this may be more effective than the method outlined above. For example, the parameter sets recommended in [2] have

$$N = 251, \quad d = 72 \ .$$

Assuming that $z = 20$, the first method above gives an estimated running time of 2^{100}, the second a time of 2^{106}. If d were 47 and z were 30, the estimated running times would be 2^{79} and 2^{81} respectively. However, note that clever scheduling of the enumeration algorithm in the second method may further reduce its running time.

1.5 Recommendations: Binary Keys

We have described the best known techniques for meet-in-the-middle search on binary keys. Additional refinements to these techniques may be possible. Our recommendation is that, as a rule of thumb, τ_c and τ_l are taken to be 1 operation, μ_f is taken to be $O(N)$, and μ_o is taken to be 0, giving the security limits:

$$\text{Running time: } \frac{\binom{N/2}{d/2}}{\sqrt{N}}$$

$$\text{Required space: } \frac{\binom{N/2}{d/2}}{\sqrt{N}}$$

The parameter sets recommended in [2] give some margin of safety above these limits, to allow for minor improvements in these techniques. To be precise:

$$N = 251, d = 72 \Rightarrow \text{running time} = 2^{100} \ .$$

2 Application to Other Forms of Keys

The paper [3] describes the efficiency gains possible by taking NTRU private keys to have a form other than random binary with d ones. For example, they may be of the form

$$f = f_1 * f_2$$

or

$$f = f_1 * f_2 + f_3.$$

In the case of the first form, the meet-in-the-middle attack consists of letting f_1 run over its whole sample space and then, for each value of f_1, splitting f_2 into f_2' and f_2'' and looking for "almost collisions" in the lists of polynomials

$$f_1 * f_2' * h \pmod q \quad \text{and} \quad - f_1 * f_2'' * h \pmod q \ .$$

Let f_1, f_2 have df_1, df_2 ones respectively. We can speed up the search time for f_1 by noting that there will always be a rotation of f_1 such that the first $(\lceil N/df_1 \rceil - 1)$ coefficients are one and the second entry is zero. We can speed up the search time for f_2 by noting that any rotation of $f_1 * f_2$ will serve as the private key, and so we can search for f_2', f_2'' as two length-$N/2$ vectors with df_2 ones each. Thus the search time will be approximately equal to

$$\tau \sim \binom{N - \lceil N/df_1 \rceil}{df_1 - 1} \cdot \binom{N/2}{df_2/2} \ .$$

If $df_1 \neq df_2$, an attacker will choose to perform the full enumeration on whichever of f_1, f_2 has fewer ones, and will perform the meet-in-the-middle part of the search on the other vector.

In the case of the second form, the meet-in-the-middle attack consists of looking for "almost collisions" in the lists of polynomials

$$f_1 * f_2 * h \pmod q \quad \text{and} \quad - f_3 * h \pmod q \ .$$

Here, the relative rotation of f_3 to $f_1 * f_2$ is important. The time to enumerate $f_1 * f_2$ will be approximately

$$\tau_{f_1 f_2} \sim \binom{N - \lceil N/df_1 \rceil}{df_1 - 1} \cdot \binom{N - \lceil N/df_2 \rceil}{df_2 - 1} \ ,$$

and the time to enumerate f_3, which cannot be speeded up by selecting a rotation, will be

$$\tau_{f_3} \sim \binom{N}{df_3} \ .$$

Note that if $df_3 \lesssim df_2$, the attacker can transfer some ones from the $f_1 * f_2$ side to the f_3 side, and search for collisions in the lists

$$f_1 * f_2' * h \pmod q \quad \text{and} \quad f_1 * f_2'' * h - f_3 * h \pmod q \ ,$$

choosing df_2' and df_2'' appropriately such that the expected running time becomes approximately

$$\tau \sim \binom{N - \lceil N/df_1 \rceil}{df_1 - 1} \cdot \sqrt{\binom{N - \lceil N/df_2 \rceil}{df_2 - 1}\binom{N}{df_3}} \,.$$

If $\binom{N-\lceil N/df_2\rceil}{df_2-1} \leq \binom{N}{df_3} \leq \binom{N-\lceil N/df_1\rceil}{df_1-1}\binom{N-\lceil N/df_2\rceil}{df_2-1}$, there does not appear to be a way to transfer work between the two sides. In this case, the running time will be dominated by the $f_1 * f_2$ term, resulting in:

$$\tau \sim \binom{N - \lceil N/df_1 \rceil}{df_1 - 1} \cdot \binom{N - \lceil N/df_2 \rceil}{df_2 - 1} \,, \tag{1}$$

If $df_3 \gtrsim (df_1 + df_2)$, the attacker can transfer some ones from the f_3 side to the $f_1 * f_2$ side. In this case, the expected running time becomes approximately

$$\tau \sim \sqrt{\binom{N - \lceil N/df_1 \rceil}{df_1 - 1}\binom{N - \lceil N/df_2 \rceil}{df_2 - 1}\binom{N}{df_3}} \,.$$

For the previously recommended parameter sets $N = 251, df_1 = df_2 = df_3 = 8$, Equation 1 gives an estimated work factor of 2^{82}.

Other suggested parameter sets have taken f to be of the form $1 + pF$, where F is binary or takes one of the product forms described above. These will increase running time by a factor of about N.

References

1. L. Babai, 'On Lovász' lattice reduction and the nearest lattice point problem', *Combinatorica*, **6** (1986), 1–13.
2. Consortium for Efficient Embedded Security, *Efficient Embedded Security Standard #1*, available from `http://www.ceesstandards.org`.
3. J. Hoffstein and J. H. Silverman. Optimizations for NTRU. In Publickey Cryptography and Computational Number Theory. DeGruyter, 2000. Available from `http://www.ntru.com`.
4. J. Hoffstein, J. Pipher, J.H. Silverman, *NTRU: A new high speed public key cryptosystem*, Algorithmic Number Theory (ANTS III), Portland, OR, June 1998, Lecture Notes in Computer Science 1423, J.P. Buhler (ed.), Springer-Verlag, Berlin, 1998, 267–288

Comments and questions concerning this technical report should be addressed to `techsupport@ntru.com`

Additional information concerning NTRU Cryptosystems and the NTRU Public Key Cryptosystem are available at `www.ntru.com`

NTRU is a trademark of NTRU Cryptosystems, Inc.

The NTRU Public Key Cryptosystem is subject to U.S. and worldwide patents.

The contents of this technical report are copyright June 20, 2003 by NTRU Cryptosystems, Inc.