

# Reduced Memory Meet-in-the-Middle Attack against the NTRU Private Key

Christine van Vredendaal

## ABSTRACT

NTRU is a public-key cryptosystem introduced at ANTS-III. The two most used techniques in attacking the NTRU private key are meet-in-the-middle attacks and lattice-basis reduction attacks. In the 2007 CRYPTO paper “A Hybrid Lattice-Reduction and Meet-in-the-Middle Attack Against NTRU” both techniques are combined and it is pointed out that the largest obstacle to attacks is the memory capacity that is required for the meet-in-the-middle phase.

In this paper an algorithm is presented that applies low-memory techniques to find ‘golden’ collisions to Odlyzko’s meet-in-the-middle attack against the NTRU private key. Several aspects of NTRU secret keys and the algorithm are analysed. The running time of the algorithm with a maximum storage capacity of  $w$  is estimated and experimentally verified. Experiments indicate that decreasing the storage capacity by a factor  $c$  increases the running time by a factor  $\sqrt{c}$ .

## 1. Introduction

NTRU [12] is a ring-based public key cryptosystem that so far has survived quantum attacks (e.g. [27, 9]). Therefore it is one of the proposals to become standard in a post-quantum-computer world. There has been much research into which parameters to choose for NTRU [16, 10, 11] to keep up with new attacks and new proposals for the structure of the private key.

Odlyzko’s meet-in-the-middle attack is one of those attacks which attempts to recover the NTRU private key from the public key. In a CRYPTO 2007 paper [14] by Nick Howgrave-Graham it is said regarding this attack:

“Odlyzko’s attack on the ees251ep6 parameter set will require too many operations ( $2^{95.8}$  modular additions) and/or too much storage ( $2^{94}$  bits) to be feasible, and hence the parameter set is more than adequate for a  $k = 80$  security level. Of these two constraints the storage requirement is by far the larger obstacle in today’s hardware.”

The requirement of  $2^{94}$  bits of storage would mean that at a conservative estimate of a \$25 1TB Hard Drive, this attack would cost over 50 quadrillion dollars.

In this paper we show that it is possible to apply low-memory search techniques to find a golden collision to the meet-in-the-middle attacks against the NTRU private key. Using these techniques we can achieve a factor  $c$  reduction in the required space of this attack at the cost of a factor of approximately  $\sqrt{c}$  in the running time.

---

2000 *Mathematics Subject Classification* 94A60 (primary), 11T71, 11Y16, 68Q25 (secondary).

This work was supported by the Netherlands Organisation for Scientific Research (NWO) under grant 639.073.005. Calculations were carried out on the Saber cluster of the Cryptographic Implementations group at Technische Universiteit Eindhoven. Date: 22 February 2016.

### 1.1. Roadmap

In Section 2 we describe the theory of NTRU and the corresponding key recovery problem. We then explain how a meet-in-the-middle attack can be mounted to find the NTRU private key. We end this section with explaining how a low-memory search for collisions works as was described in [25].

In Section 3 we describe how we can use this collision search in the meet-in-the-middle attack to achieve the reduction in the required memory.

In Section 4 we analyze some mathematical aspects of the attack. We look at the number of rotations of a key that have the correct form to be found by the attack and the probability that one of those rotations cannot be detected by the collision search. This section is ended with a heuristic for the estimated running time of the attack.

In Section 5 we verify the analysis by running simulated attacks implemented in the computer algebra system Sage [7].

Lastly, in Section 6, we discuss open questions about this work such as optimizations of the attack and applicability to other key forms and the hybrid attack of [14].

## 2. Preliminaries

### 2.1. The NTRU Cryptosystem

NTRU [12] is a ring-based public key cryptosystem and a proposed alternative to RSA and ECC. The original proposal, which we will work with for the remainder of this paper, works over the ring  $\mathcal{R} = \mathbb{Z}_q[X]/(X^n - 1)$  in which elements can be represented by vectors in  $\mathbb{Z}_q^n$  and multiplications with powers of  $X$  are cheap rotations. The integers  $q$  and  $n$  are system parameters.

Keys in this system are generated as follows. The user chooses two random small polynomials  $f, g \in \mathcal{R}$ . If we suppose that elements  $\mathcal{R}$  are represented by coefficients in the interval  $[-q/2, q/2)$ , then small in this context means that the coefficients of  $f$  and  $g$  are much smaller than the average  $q/4$ . The cryptosystem has additional parameters  $d_f$  and  $d_g$  determining how small they are by setting the private key  $f$  to have  $d_f$  coefficients equal to 1 and the rest equal to 0 and  $g$  to have  $d_g$  coefficients equal to 1 and the remainder equal to 0. In this paper we will assume  $d = d_f = d_g$  although they can be chosen differently.

The user then computes the public key as follows:

$$h \equiv f^{-1}g.$$

Note that since  $f, g \in \mathcal{R}$ , the result is also in  $\mathcal{R}$  and, as is common for NTRU, we will assume the coefficients to be centralized to fall in the interval  $[-q/2, q/2)$ . In the remainder of the paper we will assume the same for operations involving elements of  $\mathcal{R}$ . We will denote ring elements by regular typesetting  $f$  and the corresponding vectors in  $(\mathbb{Z}_q)^n$  by  $\mathbf{f}$ . For  $q$  we choose a prime smaller than the odd prime parameter  $n$ . Again, many variations are possible in  $n$ ,  $q$ , the form of  $f$  and  $g$  and the ring that is chosen and we conjecture the attack work on all variants. Some of these variations will be discussed in Section 6, but in the next sections we will assume the cryptosystem as is described above.

### 2.2. Odlyzko's Meet-in-the-Middle Attack

The original description of a meet-in-the-middle attack on NTRU is attributed to Andrew Odlyzko and first described in [15]. We will follow the description in [14] and use its notation for the remainder of this paper.

The goal of a general meet-in-the-middle attack is to find specific elements  $x, x'$  in a search space of which it is known that  $F_1(x) = F_2(x')$  for some functions  $F_1, F_2$ . There may be more solutions to the equation, also known as *collisions*, but usually a specific solution is sought. This solution is called the *golden collision* [26]. In NTRU this idea can be applied by splitting the search space of the possible private keys into two parts and searching through them separately to find the full private key. Recall that the private key  $f$  is a polynomial of degree  $n - 1$ . Searching for the key  $f$  is split up by looking for polynomials  $f_1$  in the polynomials of degree at most  $n/2 - 1$  ( $\mathcal{F}_1$ ) and polynomials  $f_2$  which only contain terms of degree between  $n/2$  and  $n - 1$  ( $\mathcal{F}_2$ ). For  $f = f_1 + f_2$  it holds that

$$\begin{aligned} h &= (f_1 + f_2)^{-1}g \\ f_1 \cdot h &= g - f_2 \cdot h. \end{aligned} \tag{2.1}$$

This implies that the correct pair of  $f_1$  and  $f_2$  will only differ by a binary vector after multiplication with  $h$ . All rotations of the keys  $f$  and  $g$  will be a solution to this equation. Therefore the mean value theorem implies the search can be restricted to  $f_1, f_2$  with (almost) equal number of ones. Without the  $g$  term the search would boil down to a simple collision search. We are however not searching for a collision, but a *near-collision*.

To search for near-collisions an auxiliary function  $a(x)$  is needed. This function takes a vector of length  $n$  and in each coordinate  $x_i$  returns  $\mathbb{1}(x_i > 0)$ . If  $g$  does not cause the coordinates of  $-f_2 \cdot h$  to change sign, i.e.  $a(-f_2 \cdot h) \neq a(-f_2 \cdot h + g)$ , we have that  $a(f_1 h) = a(-f_2 h)$ .

The attack then works as follows. We generate elements  $f_i$  from the sets  $\mathcal{F}_1$  and  $\mathcal{F}_2$  and store them in a hash table according to their auxiliary function values, or *addresses*,  $a_i$  until a collision is found. For colliding values  $f_i, f_j$  we compute if  $(f_i + f_j) \cdot h$  is binary. If it is we almost surely found the private key  $f$ .

There is a probability that  $g$  does change the sign of  $-f_2 h$  (see more on this in Section 4.1). This can be taken into account by storing it in each address that could be achieved by all possible values of  $g$ .

The expected number of loops before a collision is found is

$$L = \frac{1}{\sqrt{n}} \binom{n}{d/2} \binom{d}{d/2}^{-1/2}. \tag{2.2}$$

If we assume the computation of  $f_1 h$  and  $f_2 h$  require  $nd/2$  operations each and that each  $f_1, f_2$  takes at least  $\binom{n}{d/2}$  space to store, then this attack requires expected time  $ndL/2$  and expected space  $\binom{n}{d/2}L$ .

Note that this does not take the possibility of  $g$  changing the address into account. If we do this the space complexity needs to be multiplied by the average number of addresses that  $f_i h$  can have after adding a sparse, binary  $g$ .

### 2.3. Parallel Collision Search

Parallel collision search [24] is a method to search for colliding values  $x, x'$  in the function values  $F(x), F(x')$  for a given function  $F : S \rightarrow S$ . Its techniques are based on Pollard rho [21] and try to find the collisions by creating deterministic sequences. In [25] and subsequently [26] the application of this technique to, among others, meet-in-the-middle attacks was shown. We will follow [26]’s description of the attack.

Let  $\mathcal{D}$  be a nonempty subset of  $S$  and let  $\theta = |\mathcal{D}|/|S|$ . We call this set of points the distinguished points. We then create sequences in  $S$ , also called *trails*, by picking a random point  $x_0 \in S$  and computing  $x_i = F(x_{i-1})$  until a point  $x_t \in \mathcal{D}$  is detected. Note that the expected value of  $t$  is  $\frac{1}{\theta}$ . For each trail we store the triples  $(x_0, x_t, t)$ . Whenever we find two

triples  $(x_0, x_t, t)$ ,  $(x'_0, x'_t, t')$  with  $x_t = x'_t$  and  $x_0 \neq x'_0$  we have found a collision. These trails can be re-run from their starting values to find the steps  $x_i \neq x_j$  for which  $F(x_i) = F(x_j)$ . It can then be checked if this is the golden collision we were looking for.

This method can fail in one of two ways. It can happen that a sequence gets stuck in a cycle of which none of the points are distinguished. This risk can be managed by setting a maximal sequence length after which it is highly likely that a distinguished point has been found, for instance  $20/\theta$ . The second type of hazard is called a *Robin Hood*. This is whenever one trail hits the starting point of another trail. Such an occurrence does not lead to a collision, but can be easily detected. The occurrence is also highly unlikely.

Assuming there is one golden collision, the following (heuristic) running time for the attack is given.

**HEURISTIC 1.** (From [26]) Let  $F : S \rightarrow S$  and  $w \geq 2^{10}$  the number of triples  $(x_0, x_t, t)$  that can be stored. Then the (conjectured) optimum proportion of distinguished points is  $\theta \approx 2.25\sqrt{w/|S|}$ , and one should generate about  $10w$  trails per version of  $F$ . The expected number of iterations of  $F$  required to complete a meet-in-the-middle attack using these parameters is  $(2.5|S|^{3/2}/w^{1/2})r$ , and the expected number of memory accesses is  $4.5|S|$ .

With ‘a version of  $F$ ’ we mean that the function needs to be varied to increase the probability of success and  $r$  is the time required for a function iteration. This attack time can be linearly reduced with the number of processors used.

Also discussed in [26] is the application to meet-in-the-middle attacks. We deviate slightly from their exposition, but the idea and heuristic running times remain the same. We define two functions  $F_1 : S_1 \rightarrow S$  and  $F_2 : S_2 \rightarrow S$  and we wish to find  $x, x'$  such that  $F_1(x) = F_2(x')$ . Assume without loss of generality that  $|S_1| \geq |S_2|$ .

To apply parallel collision search to meet-in-the-middle attacks, we construct a *single* function  $F$  which has identical domain and codomain to do the collision search on. This function is constructed from the functions  $F_1$  and  $F_2$  as follows. Let  $I$  be a large enough interval of integers  $[0, \dots, I)$  ( $|I| \geq |S_1|$ ) and let  $h_1 : I \rightarrow S_1$  and  $h_2 : I \rightarrow S_2$  functions which map elements of the interval onto elements of respectively  $S_1$  and  $S_2$ . There are many options for these functions but they should map to each element in the codomain with (almost) uniform probability.

Now let  $G : S \rightarrow I \times \{0, 1\}$  be a mapping that maps elements from the codomain of the  $F_i$  to elements of  $I$  and a bit selector. A hash function in which the most significant bit is split off is a good example for  $G$ . The function  $F(x, i) = G(F_{i+1}(h_{i+1}(x)))$  is now a function of equal domain and codomain in which collision search can be performed. Note that even though  $|I|$  can be as large as we want, only  $|S| = |S_1| + |S_2|$  different values of the interval will be used.

The (almost) uniform codomain stipulation, as well the requirement for a large interval  $I$ , makes finding a surjective  $G$  improbable and it can occur that none of the paths lead to the golden collision. A solution for this is to vary  $F$ . For instance different hash functions for  $G$  could be used, or the  $h_i$  could be randomized.

Using Heuristic 1 and  $\theta = 2.25\sqrt{w/|S|}$  we get a slightly overestimated runtime of  $(2.5\sqrt{|S|^3/w})t$  with memory  $w$ . In [26] it is noted that with a small variation of the described attack, it can also be run in  $(7|S_1|\sqrt{|S_2|/w})t$ . Although this is an improvement when  $S_2$  is much smaller than  $S_1$ , we will focus on cases where  $1 < |S_1|/|S_2| < 2$ .

### 3. The Reduced Memory Meet-in-the-Middle Algorithm

In this section we describe how to apply low-memory search for golden collisions to NTRU.

Recall the secret keys are denoted by  $f$  and  $g$ , the public key  $h \equiv f^{-1}g \pmod{q}$  and  $f = f_1 + f_2$ . To apply collision techniques we first need a function  $F$ , of equal domain and codomain, in which  $f_1$  and  $f_2$  collide.

Let  $H$  be a hash function. This can be any function with codomain larger than the keyspace of the NTRU private key that is being attacked. We will denote the codomain of this hash function by  $\mathcal{I} = I \times \{0, 1\}$ . If the space  $\mathcal{F}_1$  is the space of possibilities for  $f_1$  and  $\mathcal{F}_2$  the space of possibilities for  $f_2$ , then  $|I| > \max\{|\mathcal{F}_1|, |\mathcal{F}_2|\}$ . There is an easy surjective function that maps elements from an interval  $I$  to elements in either of the spaces  $\mathcal{F}_1$  and  $\mathcal{F}_2$ .

$$I_i : I \rightarrow \mathcal{F}_i \\ x \rightarrow \text{detcomb}(x \bmod |\mathcal{F}_i|),$$

where  $i = 1, 2$  and  $\text{detcomb}$  is a function that deterministically assigns the integer  $x$  to a vector  $\mathbf{f}_i$  in  $\{0, 1\}^n$  which corresponds to a ring element  $f_i \in \mathcal{F}_i$ . The problem of assigning deterministic indices to binary sequences of length  $n$  and weight  $d$  is well known in combinatorial literature (e.g. [18]). An algorithm for the application of data compression can be found in [20, 6, 22, 5]

Note that if  $|\mathcal{F}_1| = |\mathcal{F}_2|$ , then elements in both sets have a one-to-one correspondence by division by  $x^{n/2}$ . This can be useful for efficient implementation.

We define

$$F_1 : [0, I) \rightarrow \{0, 1\}^n \\ x \rightarrow a(I_1(x) \cdot h) \\ F_2 : [0, I) \rightarrow \{0, 1\}^n \\ x \rightarrow a(-I_2(x) \cdot h),$$

where as before  $a$  is an address function and

$$G : \{0, 1\}^n \rightarrow \mathcal{I} \\ x \rightarrow (H(x) \bmod |I|) \times \text{MSB}(H(x)),$$

where  $\text{MSB}$  denotes the most significant bit. Note that the address function  $a$ , which is independent of  $i$  could also be moved to the  $G$  function. Also note that in theory we want to map uniformly onto  $I$ . In practice, taking  $|I| = \max\{|\mathcal{F}_1|, |\mathcal{F}_2|\}$  there is a large risk of  $H$  mapping to the same values after reduction modulo  $|I|$ . To avoid these ‘fake’ collisions we took  $I = |\mathcal{F}_1| \cdot |\mathcal{F}_2|$ , which seems to solve this problem. The function we will use for the collision search is now defined as

$$F : \mathcal{I} \rightarrow \mathcal{I} \\ (x, i) \rightarrow G(F_{i+1}(x))$$

The function  $F$  can now be used for a collision attack as follows. We define a distinguishing property  $\mathcal{D}$  on  $I$  and create trails starting from  $x_0 \in \mathcal{I}$  chosen at random. We run trails until they reach a distinguished point in  $x_t \in \mathcal{D}$  and then store the triple  $(x_0, x_t, t)$  in a hashlist, where  $t$  is the number of function iterations until the distinguished point was reached. When the memory is full we do not append new values to the hashlist, but instead replace a random triple. Upon a collision  $(x_0, x_t, t), (x'_0, x'_t, t')$  the trails are re-run to find the collision point (this can be done efficiently by storing  $t$  and  $t'$ ). It is then checked if it is a golden collision, and if it is, the key  $f = f_1 + f_2$  is returned. If a collision is non-golden, we replace  $(x_0, x_t, t)$  with

$(x'_0, x'_t, t')$  in the hashlist. After producing  $10w$  points we delete the hashlist and start from scratch after randomizing  $H$  by appending a seed to the addresses that are hashed.

#### 4. Analysis

This section will present analysis of the new attack of which aspects are also applicable to Odlyzko's meet-in-the-middle attack. We will end with an estimated running time for our new reduced memory meet-in-the-middle attack. To ease the readability of the analysis, this section we will assume that  $n$  and  $d$  are even, such that  $|\mathcal{F}_1| = |\mathcal{F}_2| = \binom{n/2}{d/2} = |S|/2$ . All statements can be generalized to odd cases by adding  $\lfloor \cdot \rfloor$  and  $\lceil \cdot \rceil$  in the appropriate places.

##### 4.1. Success Probability

As mentioned before, the meet-in-the-middle attack will work if  $g$  does not cause the addresses of the parts to change sign. To solve this problem we could store each sample  $f_1$  into all boxes it could fall into if  $g$  were to change the address. In the attack we use however this will not be practical. To this end we analyse the probability that  $g$  changes the sign.

**LEMMA 1.** *Suppose  $f$  and  $g$  are randomly chosen of degree  $n - 1$  with  $d$  coefficients set to 1. Under the assumption that the public key  $h$  is uniformly distributed over  $\mathcal{R}$ , the probability that  $g$  will change the address of  $-f_2h$  in Equation (2.1) is  $(1 - \frac{d}{nq})^n \approx e^{-\frac{d}{q}}$ .*

*Proof.* If  $g$  is randomly chosen with  $d$  ones, then the probability that an entry of  $g$  can change a sign is  $d/n$ . If we assume that the public key  $h$  is uniformly distributed over  $\mathcal{R}$ , then each entry of  $h$  is uniformly distributed over  $[0, q]$ . Multiplying  $f_2$  and  $h$  can be seen as summing over  $d/2$  rotations of  $h$ . Since the rotations are uniform we get a sum of uniform distributions modulo  $q$ , which is again uniform. Therefore we assume that  $-f_2 \cdot h$ , where  $\mathbf{f}_2$  has  $d/2$  ones, is uniformly distributed over  $\mathcal{R}$  and so we have a probability of  $\frac{d}{nq}$  for one specific entry to change sign and thus probability

$$(1 - \frac{d}{nq})^n \approx e^{-\frac{d}{q}}$$

of having no sign change. □

If  $q \approx 2/3n$  and  $d \approx n/3$  then this boils down to a probability of  $\exp^{-1/2} \approx 0.61$ .

Because of the mean value theorem we always have a rotation of  $\mathbf{f}$  that has  $d/2$  ones in its first  $n/2$  coordinates. To get a better estimate of our running times we first present a heuristic which estimates how many rotations have this form.

**HEURISTIC 2.** *Let  $f$  be a polynomial of degree  $n - 1$  with binary coefficient vector  $\mathbf{f}$ . Define a correct form of  $f$  as one of its  $n$  rotations  $x^i f$  that has  $d/2$  ones in the first  $n/2$  coordinates of  $\mathbf{f}^1$ . Then the average number of correct forms for  $f$  can be approximated by*

$$\frac{n\sqrt{2}}{\sqrt{\pi d}}. \tag{4.1}$$

*Proof.* Let  $f$  be of degree  $n - 1$  with  $d$  coordinates equal to 1. Then the probability that  $f$  has  $d/2$  ones in its first  $n/2$  coordinates is

$$\Pr[X = d/2] = \frac{\binom{n/2}{d/2}^2}{\binom{n}{d}}. \quad (4.2)$$

Then because each  $\mathbf{f}$  has  $n$  rotations, if we assume independence, we have an approximate number of  $n\Pr[X = d/2]$  rotations. This is not an equality, because we neglect the cases in which  $f$  has overlapping rotations. We can approximate this quantity as follows

$$\begin{aligned} n\Pr[X = d/2] &\approx \frac{n\binom{n/2}{d/2}^2}{\binom{n}{d}} \\ &\approx n \cdot \left( \frac{\left(\frac{n/2}{d/2} - 1/2\right)^{d/2} e^{d/2}}{\sqrt{\pi d}} \right)^2 \cdot \frac{\sqrt{2\pi d}}{\left(\frac{n}{d} - 1/2\right)^d e^d} \\ &= \frac{n\sqrt{2\pi d}}{\pi d} \\ &= \frac{n\sqrt{2}}{\sqrt{\pi d}}. \end{aligned}$$

Here the first approximation applies (4.2) and the second approximation is due to Stirling's Approximation.  $\square$

As [15] discovered experimentally, this is more than  $\sqrt{n}$  rotations for common parameters.

#### 4.2. Performance

We can now estimate how many collisions we need to find before we find a golden collision.

**LEMMA 2.** *Under the assumption that Heuristic 2 is correct, when mounting a meet-in-the-middle attack like described in Section 2.2 we expect to find a golden collision after  $\frac{|S|}{2} \cdot \left(e^{\frac{d}{q}} \cdot \frac{\sqrt{\pi d}}{\sqrt{2n}}\right)^{1/2}$  collisions.*

*Proof.* The number of unordered pairs of elements in  $S$  is close to  $\frac{|S|^2}{2}$ . For each of these pairs the probability that they map to the same value in  $S$  by  $F$  is  $\frac{1}{|S|}$ . This means that the total number of collisions is approximately equal to  $\frac{|S|^2}{2}$ . By Heuristic 2 and Lemma 1 on average  $e^{-\frac{d}{q}} \cdot \frac{n\sqrt{2}}{\sqrt{\pi d}}$  of them are golden. By combining this with the birthday paradox, the lemma follows.  $\square$

The next lemma shows, as [25] argues intuitively, that analysis using a full memory does not suffice.

**LEMMA 3.** *Let  $\theta$  be the fraction of distinguished points and assume they are distributed uniformly over  $S$ . Furthermore assume each one has equal probability of being reached. Then if paths are run until the memory of  $w$  points is full, an expected*

$$\frac{1 - e^{-\frac{w}{\theta^2|S|}}}{1 - e^{-\frac{1}{\theta^2|S|}}}$$

*collisions have been found.*

*Proof.* Suppose there are  $i$  distinguished points in the memory. The average walk length is  $1/\theta$ , so the expected number of points covered by these  $i$  distinguishing points is  $i/\theta$ . Let  $X_i$  be a new path. The probability that this path does *not* hit any of the paths leading up to the distinguished points is then

$$\left(1 - \frac{i}{\theta|S|}\right)^{1/\theta} \approx e^{-\frac{i}{\theta^2|S|}}.$$

This gives an expected  $e^{\frac{i}{\theta^2|S|}}$  collisions at the point where  $i$  memory elements are filled. Therefore when the allotted memory is full the expected number of collisions that have occurred is

$$\sum_{i=0}^{w-1} \left(e^{\frac{i}{\theta^2|S|}}\right)^i = \frac{1 - e^{\frac{w}{\theta^2|S|}}}{1 - e^{\frac{1}{\theta^2|S|}}}.$$

□

Of course the above lemma does not take into account that in reality some paths are more likely to be hit than others. It should however serve as a good estimate. A similar analysis implies that if the memory is full with  $w$  points then a new collision is found after every  $\frac{\theta|S|}{w}$  paths. Using the above lemmas we can now derive a heuristic running time and memory requirement for the reduced memory meet-in-the-middle attack.

**HEURISTIC 3.** *Let  $w$  be the number of triples  $(x_0, x_t, t)$  for which there is available memory. Let  $\mathcal{D}$  be a set of distinguished points with  $|\mathcal{D}|/|S| = \theta = \alpha\sqrt{w/|S|} = 2.25\sqrt{w/|S|}$ .*

*Then the algorithm is expected to run in*

$$L^* = 5r\sqrt{\frac{2\binom{n}{d}\binom{n/2}{d/2}}{nw}}, \quad (4.3)$$

*operations, where  $r$  is the number of operations needed for a function evaluation of  $F$ .*

*Proof.* As was noted in [25] a simple analysis leads to a cost per detected collision of  $\alpha\sqrt{\frac{|S|}{w}} + 2/\alpha\sqrt{|S|/w}$ . In Lemma 2 we proved that we need on average  $\gamma = \frac{|S|}{2} \cdot \left(e^{\frac{d}{q}} \cdot \frac{\sqrt{\pi d}}{\sqrt{2n}}\right)^{1/2}$  collisions before we find a golden one, which means a runtime of  $\gamma \cdot (\alpha\sqrt{\frac{|S|}{w}} + 2/\alpha\sqrt{\frac{|S|}{w}})$  function iterations if the memory is full.

There are multiple problems with this heuristic time. If the memory is not yet full, the running time per collision is longer. In Lemma 3 we saw that a number of collisions are already found during the filling of memory. On top of this, not all distinguished points have the same probability of being hit.

Therefore we use the experimentally found formula of [25] that for  $\theta = 2.25\sqrt{w/|S|}$  the expected running time to find a golden collision is slightly overestimated as  $2.5r\sqrt{\frac{|S|^3}{w}}$ . This is however an estimate for the case where there is only one golden collision. In Lemmas 2 and 1 it was shown that on average  $\frac{n\binom{n/2}{d/2}^2}{\binom{n}{d}}e^{-\frac{d}{q}}$  golden collisions can be found. We get an estimated number of function evaluations of

$$2.5 \left( \frac{e^{-\frac{d}{q}} n \binom{n/2}{d/2}^2}{\binom{n}{d}} \right)^{-1/2} \sqrt{\frac{(2 \binom{n/2}{d/2})^3}{w}} = 5 \sqrt{\frac{2 \binom{n}{d} \binom{n/2}{d/2}}{n w e^{-\frac{d}{q}}}},$$

which concludes our proof.  $\square$

The cost for each function iteration of  $F$  will be slightly higher than the cost of  $F'$  in each loop of the classic meet-in-the-middle attack. We argue that it will not make a large difference:

- Both  $F$  and  $F'$  compute a product of  $nd/2$  operations.
- Both  $F$  and  $F'$  compute an address by checking  $n$  bits in  $n$  operations
- $F$  first computes detcomb of an integer. The function performs at most  $n/2$  steps and at each step there is a comparison, a multiplication and a division. We therefore require a computational cost of  $1.5n$ . Since the random sampling of an element in  $F'$  will also incur at least  $n$  operations, it only makes a small difference.
- In  $F$  a hash function  $H$  is evaluated, which is not used in  $F'$ . The number of operations this requires depends on the hash function that is used and can be optimized based on the sample space.

As for the storage costs, each triple  $(x_0, x_t, t)$  has the following storage requirements.

$$\begin{aligned} \text{Total Storage} &= \text{storage}(x_0) + \text{storage}(x_d) + \text{storage}(t) \\ &= |\mathcal{F}| + |\mathcal{F}|\theta + \left\lceil \frac{20}{\theta} \right\rceil \\ &= |S| + \alpha \sqrt{w|S|} + \frac{\sqrt{|S|}}{\alpha \sqrt{w}} \end{aligned}$$

The total storage per triple is therefore approximately  $2 \log_2 \binom{n/2}{d/2}$  bits. Therefore if we have  $b$  bits of memory available, we should choose  $w \approx \frac{b}{2 \log_2 \binom{n/2}{d/2}}$ .

## 5. Results

We implemented tests and the attack in the computer-algebra system Sage [7] and ran it on a single core of an AMD FX-8350 Vishera 4.0GHz CPU of the Saber cluster [1].

### 5.1. Rotations

First we verified the correctness of Heuristic 2. Counting the number of correct rotations of 100000 random keys, we observed that the number is independent of  $q$ . For the dependence on  $n$  and  $d$  we graph the results in Figure 1.

We see that the experimental results of the number of rotations of the right form converge to the theoretic estimate.

### 5.2. Probability of failure

Next we verified the validity of Lemma 1. For  $n = 37$  we generated 100000 random keys  $f$  and  $g$  for different values of  $q$  and  $d$ . We then checked if  $g - f_2h$  had a different address from  $f_1h$ . The results are displayed in Figures 2 and 3.

We see that Lemma 1 predicts the probability well and that the assumption that the coefficients of  $-\mathbf{f}_2\mathbf{h}$  can be modeled as uniformly distributed is realistic. We also observed that for larger  $n$ , the value of  $n$  no longer significantly influences the probability.

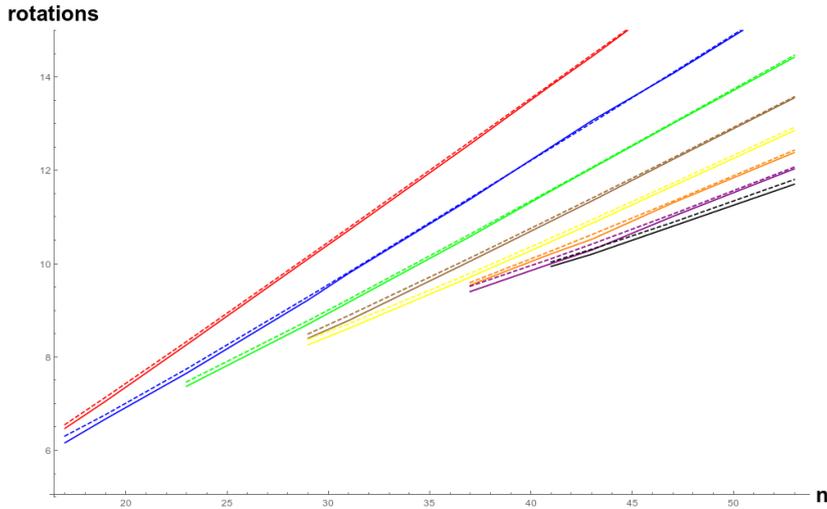


FIGURE 1. Theoretical (dashed) and experimental average (solid) of “good” rotations of 100000 random secret keys. The results for setting the parameter  $d$  equal to 6, 8, 10, 12, 14, 16, 18 and 20 are shown in respectively red, blue, green, brown, yellow, orange, purple and black.

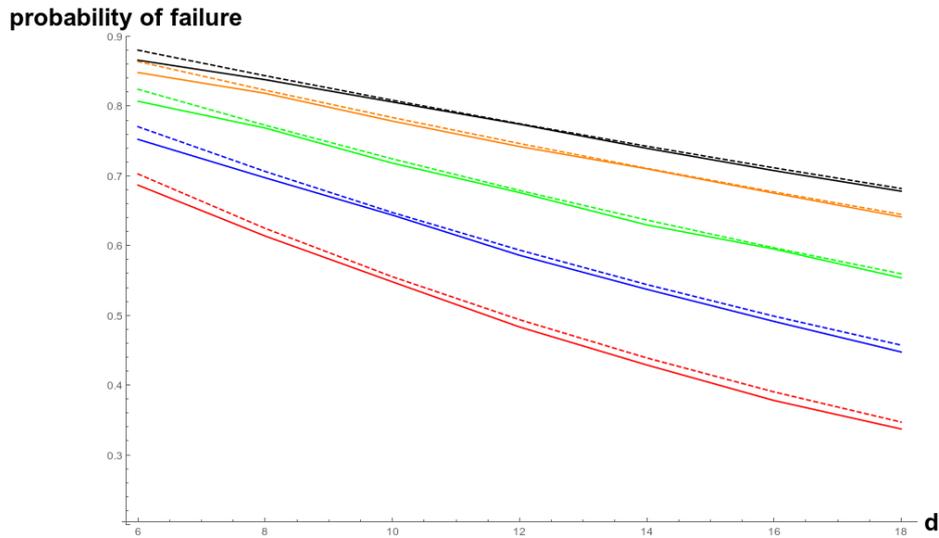


FIGURE 2. Theoretical (dashed) and experimental average (solid) of the probability of  $g$  not changing the address of 100000 random secret keys with  $n = 37$  and varying  $d$ . The results for setting  $q$  equal to 17, 23, 31, 41 and 47 are shown in respectively red, blue, green, orange and black.

### 5.3. Performance of the Algorithm

Although on average there are multiple rotations of a key  $f$  that have  $d/2$  ones in each half of its vector  $\mathbf{f}$ , it can happen that a chosen key has only a very small number of correct rotations. If this is the case it can happen that our algorithm does not find the key at all, because  $g$  might change all possible addresses.

In the experiments to measure the performance of the attack algorithm we picked a prime  $n$ ,  $q \approx (2/3)n$ ,  $d \approx (1/3)n$  and varied the number of triples  $w$  that could be stored. As advised in [25] we took the number of distinguished points as  $D = 1/\theta$  with  $\theta = 2.25\sqrt{w/|\mathcal{F}_1 + \mathcal{F}_2|} =$

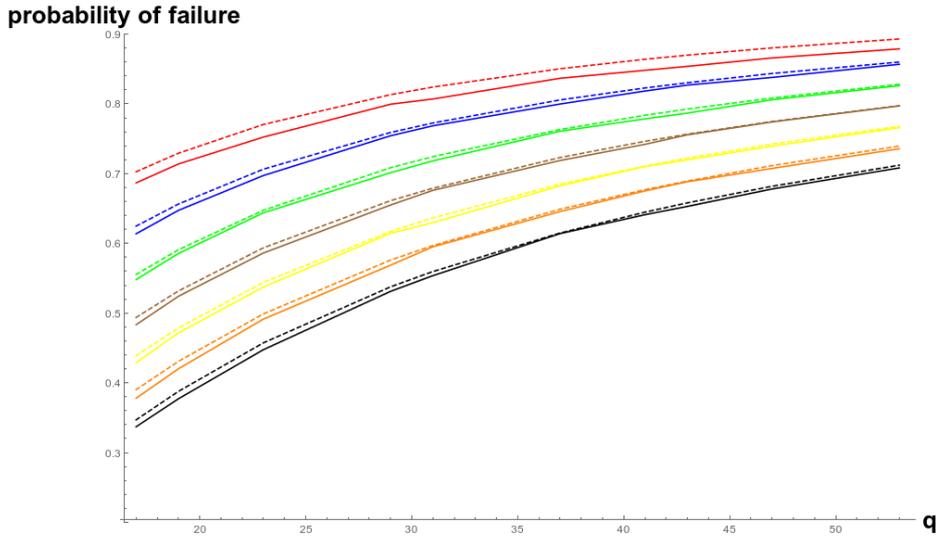


FIGURE 3. Theoretical(dashed) and experimental average(unbroken) of the probability of  $g$  not changing the address of 100000 random secret keys with  $n = 37$  and varying  $q$ . The results for setting the parameter  $d$  equal to 6, 8, 10, 12, 14, 16 and 18 are shown in respectively red, blue, green, brown, yellow, orange and black.

$2.25\sqrt{w/\left(\binom{\lfloor n/2 \rfloor}{d/2} + \binom{\lceil n/2 \rceil}{d/2}\right)}$  and randomized the step function  $F$  after every  $10w$  distinguished points. If a distinguished point was to be added to a full memory, we deleted a random entry. We ran the algorithm until at least 20 times the value predicted by Heuristic 3.

We looked at the number of evaluations of  $F$  and the success probability of the algorithm. On each set of parameters we ran 1000 attacks, each time taking a fresh random  $f$  and  $g$ . The results are given in Table 1.

We see that there is a high variance in the results; the expectation is that the first version of  $F$  will find the solution, if it does not then it takes on average  $10w|\mathcal{D}|$  steps to get another chance. If our algorithm uses approximately the same memory as required on average by the classic meet-in-the-middle attack, it uses more iterations. We also see that a decrease of factor 4 in memory capacity leads to an increased the runtime by a factor 2, which we expected from Equation 4.3. Note that this increase is limited by the search space. Increasing the allotted memory above  $|\mathcal{F}_1 + \mathcal{F}_2|$  is no longer useful. We see this by allowing the storage of  $2^{16}$  triples for the  $n = 37$  set and  $2^{18}$  and for the  $n = 43$  set, which surpasses the size of the search space.

Although there is a bit of fluctuation in the success rate with varying  $w$ , this seems to be due to the sample size. The values do seem to correlate with the probability of  $g$  not changing the address of a single rotation. Equation 1 gives values of 0.625, 0.591, 0.592, 0.617 for the parameters of respectively  $n = 23, 31, 37, 43$ . Although this is not very strong evidence, it leads us to believe that this probability will remain similar as long as  $q \approx 2d$ .

## 6. Open Questions

**Generalization of Key Forms** In this paper we assumed that the system uses binary form keys. This is the most basic version of the NTRU scheme, but other forms have been proposed. For varying reasons other proposals include ternary polynomials [16, 10, 11], sampling keys from a discrete Gaussian distribution [23], keys with a set number of ones, minus ones, twos and minus twos [8] and so called product form polynomials [13].

n	q	d	w	Success	Experimental	L* (Eq. 4.3)	L (Eq 2.2)
23	17	8	2 <sup>4</sup>	0.53	12.82	12.85	7.79
23	17	8	2 <sup>6</sup>	0.50	11.61	11.85	7.79
23	17	8	2 <sup>8</sup>	0.53	10.56	10.85	7.79
31	19	10	2 <sup>6</sup>	0.44	16.02	16.48	10.91
31	19	10	2 <sup>8</sup>	0.45	14.59	15.48	10.91
31	19	10	2 <sup>10</sup>	0.45	14.60	14.48	10.91
31	19	10	2 <sup>12</sup>	0.42	13.44	13.48	10.91
37	23	12	2 <sup>10</sup>	0.50	18.70	18.38	13.62
37	23	12	2 <sup>12</sup>	0.46	17.59	17.38	13.62
37	23	12	2 <sup>14</sup>	0.49	16.13	16.38	13.62
37	23	12	2 <sup>16</sup>	0.46	15.99	15.38	13.62
43	29	14	2 <sup>14</sup>	0.55	20.57	20.26	16.36
43	29	14	2 <sup>16</sup>	0.55	18.95	19.26	16.36
43	29	14	2 <sup>18</sup>	0.56	18.68	18.26	16.36

TABLE 1. Table with the results of 1000 trials for each set of parameters. For each set the experimental successrate and ( $\log_2$ ) runtime (in number of loops), as well as the theoretic ( $\log_2$ ) runtime ( 4.3) and the classic meet-in-the-middle ( $\log_2$ ) runtime and storage capacity ( 2.2) are given.

This attack will work for these forms. Although we chose to attack the keys by splitting them in the middle, the only requirement is that the key space  $\mathcal{F}$  can be rewritten as a decomposition  $\mathcal{F} = \mathcal{F}_1 \oplus \mathcal{F}_2$ . This can also be done for the other forms. It does have to be taken into account that for some forms there might be fewer rotations to take advantage of; in these cases the running time will increase accordingly.

**Re-keying** Given that the attack has a significant probability of failing, it would be possible to adapt the key generation algorithm of NTRU such that none of the rotations of  $f$  and  $g$  will cause addresses to collide. However, since we can choose any  $d/2$ -sized set of positions for  $\mathcal{F}_1$  and split the keyspace into  $\mathcal{F}_1$  and  $\mathcal{F} \setminus \mathcal{F}_1 = \mathcal{F}_2$ , the attack will still work.

Also the attack could be adapted to succeed with probability 1. As discussed in Section 2.2 we can store the keys in all possible addresses that can occur by  $g$  changing the sign of  $-f_2h$ . In our attack, we could add a few extra bits to  $I$  that would not only map it to a polynomial, but also select a deterministic option out of its possible addresses. If the expected number of addresses that can occur is  $A$ , then we could add  $\log_2(2A)$  bits to  $I$ . This would increase the search space by a factor  $2A$  and therefore also increase the expected running time by  $(2A)^{3/2}$ , but it would remove the possibility of failure.

**Optimization** Many optimizations are possible to increase the speed of the algorithm further. The values for  $\alpha$ , the number of distinguished points to try before switching functions as well as the method to replace distinguished points are taken from [25], but could be optimized

for this particular attack. To remove the constraint of good rotations, the attack can be adapted to choose random polynomials of  $d/2$  ones are chosen instead of splitting the space into two disjoint parts. Also, parallelization could be easily achieved by using a central server that collects the distinguished points and thus lets multiple cores give a linear speed-up in the runtime. Furthermore, as is for instance suggested in [2], it might be better to not replace the distinguished points randomly when the memory is full, but instead replace the “less-used” points. The goal of this paper was however not to implement an optimized attack, but to show that the amount of memory that is used can be varied and in particular that this does not constitute a bottleneck.

**Hybrid Attack** The hybrid lattice-basis reduction and meet-in-the-middle attack [14] mentioned in the introduction works by taking the NTRU lattice and applying lattice-basis reduction to part of the matrix and then applying a meet-in-the-middle approach to the remaining key space. Although we did not write down the details of such an attack, it should be possible to apply the reduced memory algorithm of this paper to reduce the memory requirement for the hybrid attack. The difference is that the guess for the structure of the key and the fact that a good guess of the key might not be detected by a rounding algorithm must be considered.

A related class of cryptographic schemes that have so far withstood quantum attacks are based on the Learning with Errors problem (see e.g. [19, 8, 3] and [17]). It has recently been shown that the hybrid attack applies to LWE with binary error as well [4]. This reduced memory variation of the attack should be taken into account for parameter considerations of NTRU and LWE-based schemes.

For the ees251ep6 parameter set of NTRU, [14] estimated a storage cost of  $2^{65.6}$  after applying the lattice reduction phase of the hybrid attack with  $2^{56.7}$  loops of the algorithm. We conjecture that if the attack had to be run with only 1TB storage available, then using a similar attack as described in this paper could be mounted at the expense of only a factor  $\sim 2^{13}$  more running time. We leave the details of the attack for possible future research, but this paper’s goal is to show that the expense of storage capacity is not as unambiguous as portrayed in [14].

### References

1. Daniel J. Bernstein. The Saber cluster, 2014. <http://blog.cr.yp.to/20140602-saber.html>.
2. Daniel J. Bernstein and Tanja Lange. Computing small discrete logarithms faster. In Steven D. Galbraith and Mridul Nandi, editors, *Progress in Cryptology - INDOCRYPT 2012, 13th International Conference on Cryptology in India, Kolkata, India, December 9-12, 2012. Proceedings*, volume 7668 of *Lecture Notes in Computer Science*, pages 317–338. Springer, 2012.
3. Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. *SIAM J. Comput.*, 43(2):831–871, 2014.
4. Johannes Buchmann, Florian Göpfert, Rachel Player, and Thomas Wunderer. On the hardness of LWE with binary error: Revisiting the hybrid lattice-reduction and meet-in-the-middle attack. *IACR Cryptology ePrint Archive*, 2016:089, 2016.
5. Thomas M. Cover. Enumerative source encoding. *IEEE Transactions on Information Theory*, 19(1):73–77, 1973.
6. L.D. Davisson. Comments on “Sequence time coding for data compression”. *Proceedings of the IEEE*, 54(12):2010–2010, Dec 1966.
7. The Sage Developers. *Sage Mathematics Software (Version 6.9)*, 2015. <http://www.sagemath.org>.
8. Léo Ducas, Alain Durmus, Tancrede Lepoint, and Vadim Lyubashevsky. Lattice signatures and bimodal gaussians. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 40–56. Springer, 2013.
9. Scott Fluhrer. Quantum cryptanalysis of NTRU. *IACR Cryptology ePrint Archive*, 2015:676, 2015.
10. Philip S. Hirschhorn, Jeffrey Hoffstein, Nick Howgrave-Graham, and William Whyte. Choosing NTRU-Encrypt parameters in light of combined lattice reduction and MITM approaches. In Michel Abdalla, David Pointcheval, Pierre-Alain Fouque, and Damien Vergnaud, editors, *Applied Cryptography and*

- Network Security, 7th International Conference, ACNS 2009, Paris-Rocquencourt, France, June 2-5, 2009. Proceedings*, volume 5536 of *Lecture Notes in Computer Science*, pages 437–455, 2009.
11. Jeffrey Hoffstein, Jill Pipher, John M. Schanck, Joseph H. Silverman, William Whyte, and Zhenfei Zhang. Choosing parameters for NTRUEncrypt. *IACR Cryptology ePrint Archive*, 2015:708, 2015.
  12. Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. NTRU: A ring-based public key cryptosystem. In Joe Buhler, editor, *Algorithmic Number Theory, Third International Symposium, ANTS-III, Portland, Oregon, USA, June 21-25, 1998, Proceedings*, volume 1423 of *Lecture Notes in Computer Science*, pages 267–288. Springer, 1998.
  13. Jeffrey Hoffstein and Joseph H. Silverman. Random small Hamming weight products with applications to cryptography. *Discrete Applied Mathematics*, 130(1):37–49, 2003.
  14. Nick Howgrave-Graham. A hybrid lattice-reduction and meet-in-the-middle attack against NTRU. In Alfred Menezes, editor, *Advances in Cryptology - CRYPTO 2007, 27th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2007, Proceedings*, volume 4622 of *Lecture Notes in Computer Science*, pages 150–169. Springer, 2007.
  15. Nick Howgrave-Graham, Joseph H Silverman, and William Whyte. A meet-in-the-middle attack on an NTRU private key. Technical report, NTRU Cryptosystems, June 2003. Report, 2003.
  16. Nick Howgrave-Graham, Joseph H. Silverman, and William Whyte. Choosing parameter sets for NTRUEncrypt with NAEP and SVES-3. *IACR Cryptology ePrint Archive*, 2005:45, 2005.
  17. Adeline Langlois, San Ling, Khoa Nguyen, and Huaxiong Wang. Lattice-based group signature scheme with verifier-local revocation. In Hugo Krawczyk, editor, *Public-Key Cryptography - PKC 2014 - 17th International Conference on Practice and Theory in Public-Key Cryptography, Buenos Aires, Argentina, March 26-28, 2014. Proceedings*, volume 8383 of *Lecture Notes in Computer Science*, pages 345–361. Springer, 2014.
  18. Dick H. Lehmer. Teaching combinatorial tricks to a computer. In *Proc. Sympos. Appl. Math., Vol. 10*, pages 179–193. American Mathematical Society, Providence, R.I., 1960.
  19. Richard Lindner and Chris Peikert. Better key sizes (and attacks) for LWE-based encryption. In Aggelos Kiayias, editor, *Topics in Cryptology - CT-RSA 2011 - The Cryptographers' Track at the RSA Conference 2011, San Francisco, CA, USA, February 14-18, 2011. Proceedings*, volume 6558 of *Lecture Notes in Computer Science*, pages 319–339. Springer, 2011.
  20. T.J. Lynch. Sequence time coding for data compression. *Proceedings of the IEEE*, 54(10):1490–1491, Oct 1966.
  21. John M. Pollard. Monte Carlo methods for index computation (mod  $p$ ). *Mathematics of Computation*, 32:918–924, 1978.
  22. J. Schalkwijk. An algorithm for source coding. *Information Theory, IEEE Transactions on*, 18(3):395–399, May 1972.
  23. Damien Stehlé and Ron Steinfeld. Making NTRU as secure as worst-case problems over ideal lattices. In Kenneth G. Paterson, editor, *Advances in Cryptology - EUROCRYPT 2011 - 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, Estonia, May 15-19, 2011. Proceedings*, volume 6632 of *Lecture Notes in Computer Science*, pages 27–47. Springer, 2011.
  24. Paul C. van Oorschot and Michael J. Wiener. Parallel collision search with application to hash functions and discrete logarithms. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, and Ravi S. Sandhu, editors, *CCS '94, Proceedings of the 2nd ACM Conference on Computer and Communications Security, Fairfax, Virginia, USA, November 2-4, 1994.*, pages 210–218. ACM, 1994.
  25. Paul C. van Oorschot and Michael J. Wiener. Improving implementable meet-in-the-middle attacks by orders of magnitude. In Neal Koblitz, editor, *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings*, volume 1109 of *Lecture Notes in Computer Science*, pages 229–236. Springer, 1996.
  26. Paul C. van Oorschot and Michael J. Wiener. Parallel collision search with cryptanalytic applications. *J. Cryptology*, 12(1):1–28, 1999.
  27. Hong Wang, Zhi Ma, and ChuanGui Ma. An efficient quantum meet-in-the-middle attack against NTRU-2005. *Chinese Science Bulletin*, 58(28):3514–3518, 2013.

C. van Vredendaal  
Department of Mathematics and Computer  
Science  
Technische Universiteit Eindhoven  
P.O. Box 513, 5600 MB Eindhoven, The  
Netherlands

c.v.vredendaal@tue.nl