A Hybrid Lattice-Reduction and Meet-in-the-Middle Attack Against NTRU

Nick Howgrave-Graham

NTRU Cryptosystems, Inc.
nhowgravegraham@ntru.com

Abstract. To date the NTRUEncrypt security parameters have been based on the existence of two types of attack: a meet-in-the-middle attack due to Odlyzko, and a conservative extrapolation of the running times of the best (known) lattice reduction schemes to recover the private key. We show that there is in fact a continuum of more efficient attacks between these two attacks. We show that by combining lattice reduction and a meet-in-the-middle strategy one can reduce the number of loops in attacking the NTRUEncrypt private key from $2^{84.2}$ to $2^{60.3}$, for the $k = 80$ parameter set. In practice the attack is still expensive (dependent on ones choice of cost-metric), although there are certain space/time tradeoffs that can be applied. Asymptotically our attack remains exponential in the security parameter k, but it dictates that NTRUEncrypt parameters must be chosen so that the meet-in-the-middle attack has complexity 2^k even after an initial lattice basis reduction of complexity 2^k.

1 Introduction

It is well known that the closest vector problem (CVP) can be solved efficiently in the case that the given point in space is very close to a lattice vector [7,18]. If this CVP algorithm takes time t and a set S has the property that it includes at least one point $v_0 \in S$ which is very close to a lattice vector, then clearly v_0 can be found in time $O(|S|t)$ by exhaustively enumerating the set S. We show that if the points of S can be represented as $S = S' \oplus S'$, i.e. for every $(v, v') \in S \times S'$ there exists an $v'' \in S'$ such that $v = v' + v''$, then there are conditions under which there is actually an efficient meet-in-the-middle algorithm on this space to find the point v_0 in time $O(|S|^{1/2}t)$.

We can translate this CVP result to a result about lattice basis reduction by defining the set S to be some linear combinations of the last $n-m$ rows of a given basis $\{b_1 \ldots, b_n\}$, and then using the CVP algorithm on the elements of S and the basis $\{b_1, \ldots, b_m\}$. We note that a similar approach is taken by Schnorr in [21] for reducing generic lattices with the SHORT algorithm. Schnorr also suggests that "birthday" improvements might be possible for his method (generalizing results from [24]) but concludes that, in general, storage requirements may be prohibitive.

In this paper we show that, in the case of searching for the NTRUEncrypt private key, meet-in-the-middle techniques are indeed possible. We show that

A. Menezes (Ed.): CRYPTO 2007, LNCS 4622, pp. 150–169, 2007.

Odlyzko's storage ideas may be generalized to remain efficient even when used after lattice reduction, and we optimize the set S for the structure of the NTRU-Encrypt private key.

1.1 Roadmap

In section 2 we describe the key recovery problem behind NTRUEncrypt, and we explain the best known attacks against it. We introduce the following question regarding its parameters: "for a given N and q and security parameter k, how low can d_f be?": this is the fundamental mathematical question that our paper addresses, and it ultimately shows that d_f cannot be as low as previously thought. This is important because d_f is one of the factors that govern the efficiency of NTRUEncrypt, and so from a parameter generation point of view there is a practical desire to keep it as low as possible.

In section 3 we give a brief summary of the theory lattices including the usefulness of triangularization of lattice bases. In section 3.1 we discuss the practical consequences of running lattice reduction schemes on the NTRU public basis.

In section 4 we explain the mathematics behind the new hybrid technique, and in section 5 we analyze the cost of the technique in theory and in practice.

As with many meet-in-the-middle techniques the storage requirements of our technique are considerable. In section 6 we discuss methods to lessen these requirements at the cost of increasing the running time.

In section 7 we discuss possible generalizations of our work in more generic lattice situations, and give conclusions in section 8.

2 The NTRU Cryptosystem

NTRUEncrypt was invented in 1996, and was first published in [10]. It is based in the ring $\mathcal{R} = \mathbb{Z}[X]/(X^N - 1, q)$ whose elements can be represented by vectors of length N with integer entries modulo q. To aid exposition we will differentiate between a vector representation $a \in V_N(\mathbb{Z})$ and a ring representation $\mathsf{a} \in \mathcal{R}$ by the use of the $\LaTeX$ fonts shown. The NTRUEncrypt private key is two "binary" vectors $f, g \in V_N(\{0,1\})$ with d_f and d_g ones respectively, and the remaining entries zero[1]. The NTRUEncrypt public key is $\mathsf{h} = \mathsf{g}/\mathsf{f}$ in the ring $\mathcal{R}$, where h is typically viewed as h, a vector of length N with integer entries modulo q.

There are many good descriptions of how the NTRU cryptosystem works [9,10,11,14], but in this paper we directly take on the problem of recovering the private key from the public information, so we do not need to delve into details of encryption and decryption. Out of interest we note that the encryption and decryption algorithms are both very efficient operations (both encryption and decryption are $O(k^2)$ in the security parameter k), and all known attacks against NTRUEncrypt are exponential in the security parameter k (including

[1] Other sets of small vectors are possible for the set of NTRUEncrypt private keys, but this is the one we will initially concentrate on.

the one demonstrated in this paper). Another potential upside of NTRUEncrypt is its apparent resistance to attack by quantum computers. The downsides to NTRUEncrypt are that the public key-size and ciphertext size are both slightly large, and that there is expansion in encryption (a raw N-bit plaintext (after padding) is encrypted to a $(N \log_2 q)$-bit ciphertext) so NTRUEncrypt lacks some of the nice properties that an encryption-permutation allows.

The parameter choices for N, q, d_f, d_g have undergone several changes since the invention of NTRUEncrypt due to both progress in cryptanalysis [11], and fine tuning of the parameters for efficiency reasons [14]. The currently recommended choices for $k = 80$ bit security are $N = 251$, $q = 197$, $d_f = 48$, $d_g = 125$. This parameter set is known as `ees251ep6` in the IEEE P1363.1 draft standard [16].

The attack demonstrated in this paper is applicable, to some degree, to all the NTRUEncrypt parameter sets since its invention. Unfortunately it is most effective on the currently recommended parameter sets because d_f has been lowered considerably for efficiency reasons.

2.1 Lattice Attacks Against NTRU

The recovery of the NTRUEncrypt private key from public information can be posed as a lattice problem. This was known by the inventors of NTRU [10], and further explored in [5].

From the definition of $\mathsf{h} = \mathsf{g}/\mathsf{f}$, it is clear that there is an length-$(2N)$ integer vector (k, f) such that

$$(k, f) \begin{pmatrix} qI & 0 \\ H & I \end{pmatrix} = (g, f), \tag{1}$$

where H is a circulant matrix generated from h, i.e. $H_{i,j} = h_{i+j \bmod N}$. Note that the vector/matrix multiplication fH respects the multiplication fh in the ring $\mathbb{Z}[X]/(X^N - 1)$, and the k part of the vector corresponds to the reduction of each coefficient modulo q. The $(2N) \times (2N)$ basis $((qI, 0), (H, I))$ is referred to as the *NTRU public lattice basis.*

The discriminant of the NTRU public lattice basis is clearly q^N, whilst the (g, f) vector has size $(d_f + d_g)^{1/2}$. The Gaussian heuristic therefore suggests that there are no smaller vectors in the lattice than (g, f) and so lattice reduction might be used to find it. We note that the NTRU public lattice basis does not contain just one small vector (g, f) but all N "rotations" $(g^{(i)}, f^{(i)})$ where $g^{(i)}$, $f^{(i)}$ correspond to $\mathsf{f}^{(i)} = \mathsf{f}X^i$ and $\mathsf{g}^{(i)} = \mathsf{g}X^i$ respectively, for $i = 0, \ldots, N-1$, since $(\mathsf{f}X^i)\mathsf{h} = \mathsf{g}X^i$ in the ring $\mathcal{R}$.

Although the rotations of the (g, f) vectors are the smallest vectors in the NTRU lattice, the best (known) direct lattice reduction techniques find it hard to recover any of these vectors in practice. Indeed typically lattice reduction methods appear to be fully exponential in the security parameter k. For the lattice family to which the `ees251ep6` parameter set belongs, it is stated in [14,16] that lattice reduction has a complexity of at least

$$R = 2^{0.4245N - 3.44} \tag{2}$$

for $N > 120$, to find any vector smaller that a q-vector. For $N = 251$ this corresponds to time of $2^{103.1}$ to directly find a (g, f) rotation from the public basis.

2.2 Odlyzko's Meet-in-the-Middle Attack on NTRU

NTRU parameter sets have always been secure against a meet-in-the-middle attack discovered by Odlyzko, which is described in [15].

The idea is that if f_1 and f_2 are such that $f = f_1 + f_2$ then the entries of $\mathsf{x}_1 = \mathsf{f}_1\mathsf{h}$ and $\mathsf{x}_2 = -\mathsf{f}_2\mathsf{h}$ differ only by 0 or 1 mod q, since $(\mathsf{f}_1 + \mathsf{f}_2)\mathsf{h} = \mathsf{g}$ and g is binary.

Assuming f has d_f ones and d_f is even, then the attack progresses by sampling a binary ring element f_1 with $d_f/2$ ones, and computing $\mathsf{x}_1 = \mathsf{f}_1\mathsf{h}$.

The vector x_1 corresponding to x_1 is of length N with entries satisfying $-q/2 < (x_1)_i \leq q/2$. For each index i of x_1 we determine a bit β_i, where $\beta_i = 1$ if $(x_1)_i > 0$ and 0 otherwise. We can therefore determine an N-bit string from x_1, namely $a_1 = \beta_1 : \beta_2 : \ldots : \beta_N$, which we call an "address" or "label'. Let $\overline{\beta} = 1 - \beta$ denote the complement of a bit β, and let $\overline{a}$ denote the component-wise complement of a bit-string a. The element f_1 is stored in two "boxes": one with address a_1, and one with address $\overline{a_1}$.

The meet-in-the-middle technique carries on sampling f_1 as above, and storing them in boxes dependent on the x_1. If two binary elements f_1 and f_2 are sampled such that $\mathsf{f}_1 + \mathsf{f}_2 = \mathsf{f}$ then one can hope that the a_1 corresponding to $\mathsf{x}_1 = \mathsf{f}_1\mathsf{h}$ is the same as the $\overline{a_2}$ corresponding to $\mathsf{x}_2 = \mathsf{f}_2\mathsf{h}$, since $\mathsf{x}_1 = -\mathsf{x}_2 + \mathsf{g}$. This will only be the case if the entries of g do not cause af the entries of x_1 to "change sign", but this technicality can be dealt with by either simply accepting the probability of the occurrence, or by storing the f_1 in more boxes if the x_1 have coefficients that may change sign. These approaches are discussed further in [15] and later in this report.

In this introduction we will assume that whenever $\mathsf{f}_1 + \mathsf{f}_2 = \mathsf{f}$ then with certainty $a_1 = \overline{a_2}$, i.e. sampling f_1, f_2 such that $\mathsf{f}_1 + \mathsf{f}_2 = \mathsf{f}$ can be detected by a collision in a box. For any collisions we can retrieve the f_1, f_2 stored in the box, and check if $(\mathsf{f}_1 + \mathsf{f}_2)\mathsf{h}$ is binary: if so we have found a very small vector in the NTRU public basis; undoubtedly[2] one of the rotations of (g, f).

To estimate the complexity of this attack, let V denote the set of f_1 which are actually a subset of the ones of some rotation of f. Assuming the rotations have a small number of intersections we see that $|V| \approx N\binom{d_f}{d_f/2}$, and we can expect a collision in the set of such f_1 after $O(|V|^{1/2})$ samples. The probability of sampling from this set is $|V|/\binom{N}{d_f/2}$, so the expected number of loops of the algorithm before a collision is

$$L = \frac{1}{\sqrt{N}}\binom{N}{d_f/2}\binom{d_f}{d_f/2}^{-1/2}. \tag{3}$$

For the `ees251ep6` parameter set this turns out to be $2^{84.2}$.

[2] It will almost certainly be a (g, f) rotation because of the way we performed the search, although it is worth noting that discovering any short enough vector is tantamount to breaking NTRUEncrypt, as observed in [5].

2.3 Choosing NTRUEncrypt Parameters

NTRU are typically conservative when choosing parameters, so the true lattice security (when considering BKZ attacks only) is probably significantly higher than equation 2 suggests, due to the upward concavity of the observed running times. Similarly, ensuring that the number of loops, L, given by equation 3 is greater than 2^{80} is conservative for two reasons:

- There are hidden computational costs per loop, e.g. Odlyzko's attack requires summing together $d_f/2$ vectors of length N and reducing their coefficients modulo q. If we count "one addition modulo q" as an "intrinsic operation" then this cost could arguably add $\log_2(Nd_f/2)$ bits of security.
- The storage requirements of Odlyzko's attack is slightly greater than the number loops given by equation 2, since we may need to store the f_1's in several boxes per loop (on average 8, say). Also the f_1's take at least $\log_2 \binom{N}{d_f/2}$ bits to store.

Thus one might conclude that, in practice, 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 given today's hardware.

Although NTRU have been conservative in their parameter choices, this is with respect to the best known attacks. In this paper we demonstrate a new class of attack that may cause NTRU to re-evaluate their parameter sets. Indeed, as a piece of mathematics, the contribution of this paper can be summed up as an improved answer to the question "for a fixed N and q and a security level k, how low can d_f go?": we show that d_f cannot be as low as previously thought.

To gauge the practicality of our attack, we examine how it changes the running time and storage requirements of Odlyzko's attack in section 5, and discuss methods to make the storage requirements more feasible (at the cost of extra computation) in section 6.

This paper is not about suggesting new parameter sets which would require a large amount of analysis to justify well, however we do mention techniques that can mitigate our attack in section 8. When it does come to choosing new NTRU parameters we advocate the methodology outlined in [19], i.e. for a fixed PC architecture working out how much time it takes to break a symmetric key algorithm (e.g. DES), and how much time it takes to break a small NTRUEncrypt example, and then extrapolating the two to work out when NTRUEncrypt will require the same amount of work as an 80-bit symmetric algorithm (and similarly for higher k-bit security levels).

The above methodology (of comparison with an exhaustive search on a DES symmetric key) essentially benchmarks the PC in a standard way, and for example, allows one to argue how much security $2^{95.8}$ modular additions truly gives (it is certainly less than a bit security level of $k = 95.8$).

3 Lattice Basis Representation and Lattice Reduction

We take a row-oriented view of matrices and allow some flexibility between basis representations and matrix representations, e.g. we call a *matrix* BKZ-reduced if the *rows* of the matrix form a BKZ-reduced basis [22].

For a thorough grounding on lattices see [3,4], however for our purposes the following will suffice: for a given basis $\mathcal{B} = \{b_1, \ldots, b_n\}$ of $\mathbb{R}^n$ a lattice is defined to be the set of points

$$\mathcal{L} = \left\{ y \in \mathbb{R}^n \;\middle|\; y = \sum_{i=1}^{n} a_i b_i,\ a_i \in \mathbb{Z} \right\}$$

Clearly many bases will generate the same set of lattice points; indeed if we represent a basis $\mathcal{B}$ by a matrix B with rows $\{b_1, \ldots, b_n\}$ then it is exactly the rows of UB for any $U \in GL_n(\mathbb{Z})$ that generate these points.

However it is often convenient to give ourselves even more freedom with matrix representations of bases in that one can consider bases of isomorphic lattices too[3].

Definition 1. *Two lattices $\mathcal{L}, \mathcal{L}'$ are called isomorphic if there is a length-preserving bijection $\phi : \mathcal{L} \to \mathcal{L}'$ satisfying $\phi(x + y) = \phi(x) + \phi(y)$.*

In terms of matrix representations this means that if the rows of B form a basis for a lattice $\mathcal{L}$ then the rows of $B' = UBY$ where $U \in GL_n(\mathbb{Z})$ and Y is orthonormal, form a basis for an isomorphic lattice $\mathcal{L}'$, even though the rows of B' do not necessarily generate the same *points* of $\mathcal{L}$.

The point of allowing the extra freedom of post-multiplying by an orthonormal matrix is that if (for some reason) one can find an integer vector u such that uB' is small, then $uU^{-1}B$ is also small, i.e. solving lattice problems in an isomorphic lattice can help solve them in the original lattice. It is worth noting that this freedom also allows one to always consider lower triangular lattice bases by forming Y from the Gram-Schmidt procedure[4]. Explicitly $T_{i,j} = \mu_{i,j}|b_j^*|$ where $\mu_{i,j} = \langle b_i, b_j^* \rangle / |b_j^*|^2$ for $1 \leq j < i \leq n$ and $\mu_{i,i} = 1$.

Given that there are many bases of the same lattice $\mathcal{L}$, there is a significant amount of research around defining which bases are "more reduced" than others, and generating efficient algorithms to produce such bases [22,8,21]. The most commonly used reduction scheme in cryptography is BKZ [22] and its efficient implementation in the number theory library NTL [23].

Lattice reduction typically transforms a basis $\{b_1, \ldots, b_n\}$ so that the size of the Gram-Schmidt vectors b_i^* do not decrease "too quickly". This allows one to

[3] This phenomenon is usually explained through the language of quadratic forms, but such a presentation typically misses the concreteness of the isomorphic lattice bases, which we prefer in this report.

[4] It is worth saying that mathematicians do not always apply this transformation because some non-lower triangular lattice bases naturally have integer entries (as opposed to general real entries), and putting a lattice in lower triangular form can force the use of square roots of rational numbers (or real approximations) in this case.

prove an approximation factor between the size of the first vector b_1 and the size of the smallest vector in the lattice λ_1 (which is normally bounded by the size of b_n^*). Thus lattice reduction can be used to solve the (approximate) shortest vector problem (SVP).

Another well-studied lattice problem is the (approximate) closest vector problem (CVP): one is given an arbitrary point in space $y \in \mathbb{R}^n$ and the problem is to find the closest lattice point to this point (or more generally a lattice point within a radius of a multiple of λ_1). We make use of the following simple CVP-algorithm when the lattice basis is given by the rows of a lower triangular $(n) \times (n)$ matrix T (as explained above a basis can always be represented this way, and this avoids explicit use of b_i^*). We remark that this algorithm has a long history; it is sometimes called "weak reduction" or "size reduction" of the vector y against the basis T and is an essential component of lattice reduction techniques, however is usually referred to as Babai's nearest plane algorithm from the analysis in [2].

Algorithm 1. weakly reducing y against T

```
1: $x \leftarrow y$
2: for $i = n$ down to 1 do
3:     let $u_i$ to be the nearest integer to $x_i/T_{i,i}$
4:     $x \leftarrow x - u_i T_i$
5: end for
6: return the reduced vector $x$
```

The following lemma (first shown in [7]) shows that if a point in space is "particularly close" to a lattice vector then it can be recovered by algorithm 1.

Lemma 1 (Furst, Kannan). *Assume $y = uT + x$ for some $u \in V_n(\mathbb{Z})$, $x \in V_n(\mathbb{R})$ and a lower triangular $T \in M_n(\mathbb{R})$. If the entries of x satisfy*

$$-T_{i,i}/2 < \quad x_i \quad \leq T_{i,i}/2 \tag{4}$$

for $1 \leq i \leq n$, then x can be recovered by algorithm 1.

Proof. It is simple to confirm that the "error" vector x does not change any of the "rounding" computations of u_i in step 3 of algorithm 1.

3.1 Reducing the NTRU Public Basis

The NTRU public basis is given[5] in equation 1. The state of a partially-reduced NTRU lattice can be expressed well by plotting the $\log_q |b_i^*|$ for $i = 1, \ldots, 2N$, as done in [9]. Figure 1 shows the various states of reduction of an NTRU public basis with the `ees251ep6` parameter set.

[5] We note that this basis description is slightly different from the original description in [10], but we prefer putting the small q-vectors first.

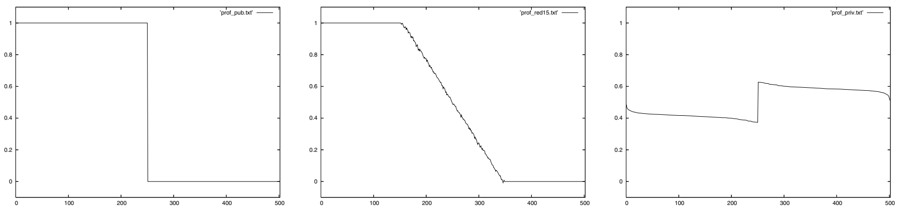

Fig. 1. A representation of lattice reduction on NTRU lattices via plotting $\log_q |b_i^*|$. The left figure is the public basis, the middle figure is after reduction with BKZ with blocksize 15, the right is the fully reduced private basis.

As can be seen from the middle graph of figure 1, the first few b_i^* vectors of a partially-reduced NTRU basis typically remain q-vectors, whilst the last b_i^* vectors typically satisfy $|b_i^*| = 1$. We note the (approximate) symmetry of these graphs can be made exact by using the symplectic lattice reduction techniques of [9]. Notice the central region of a partially-reduced NTRU basis is approximately linear in the log scale, i.e. it obeys the geometric series assumption (GSA) as defined in [21].

Given that the first and last vectors are untouched, reducing an NTRU basis can be speeded up by extracting a suitable lower triangular submatrix B', reducing this, and putting the basis back in a lower triangular form[6]. If the public basis is represented by B, then this transformation can be written $UBY = T$, where the structure of U, B, Y and T are shown below:

$$\left(\begin{array}{c|c|c} I_r & 0 & 0 \\ \hline 0 & U' & 0 \\ \hline 0 & 0 & I_{r'} \end{array}\right) \left(\begin{array}{c|c|c} qI_r & 0 & 0 \\ \hline * & B' & 0 \\ \hline * & * & I_{r'} \end{array}\right) \left(\begin{array}{c|c|c} I_r & 0 & 0 \\ \hline 0 & Y' & 0 \\ \hline 0 & 0 & I_{r'} \end{array}\right) = \left(\begin{array}{c|c|c} qI_r & 0 & 0 \\ \hline * & T' & 0 \\ \hline * & * & I_{r'} \end{array}\right). \tag{5}$$

The (partially-reduced) matrix T also has N small vectors given by $(g^{(i)}, f^{(i)})$ Y, where $(g^{(i)}, f^{(i)})$ are the original small vectors corresponding to $\mathsf{f} = \mathsf{f}X^i$, $\mathsf{g} = \mathsf{g}X^i$ for $i = 1, \ldots, N$. From the structure of Y we see that these small vectors have binary entries for the first r entries and the last r' entries, and only the middle entries are affected by Y'.

Let $m = 2N - r'$ denote the number of vectors in the first two "blocks", and let $\{b'_1, \ldots, b'_{m-r}\}$ denote the rows of B'. Our strategy to recover the NTRUEncrypt private key is to pick a submatrix B' such that $(b'_{m-r})^*$ can be made reasonably large (so that lemma 1 may be usefully employed), whilst at the same time making m reasonably large so that the last r' entries of $(g^{(i)}, f^{(i)})Y$ can either be guessed, or (less-restrictively) have a meet-in-the-middle attack mounted on them.

We remark that the standard way to ensure $(b'_{m-r})^*$ is large, is to try to minimize the first vector in the dual matrix of B', as described in [6,12,13,20].

[6] This is completely akin to the treatment of blocks in a block reduction scheme.

4 The Hybrid Lattice-Reduction and Meet-in-the-Middle Method

Let T be as defined in equation 5, and let u, v, s be such that

$$(u|v)T = (s|v) = (g^{(i)}, f^{(i)})Y,$$

for some $i = 1, \ldots, 2N$ and where u, s are of length m, and v is of length $r' = 2N - m$.

We start by showing that an algorithm that enumerates all possible v is enough to recover $(u|v)$, and then show that there is actually a meet-in-the-middle algorithm to recover the same information. We note that knowledge of $(u|v)$ is clearly equivalent to knowledge of $(s|v)$ and (g, f).

Lemma 2. *The vector $(0|v)T - (0|v)$ is a distance of $|s|$ away from a lattice point of T.*

Proof. We know

$$\begin{aligned}(0|v)T - (0|v) &= (u|v)T - (u|0)T - (0|v)\\ &= (s|0) - (u|0)T.\end{aligned}$$

Corollary 1. *If s if "small enough" to satisfy the conditions of lemma 1 then it can be found by algorithm 1.*

In our analysis we always ensure that s satisfies the conditions of lemma 1 for a large proportion[7] of the rotations $(g^{(i)}, f^{(i)})$. In principle this condition could be slightly relaxed by using the methods of [18,21], at the cost of doing some extra "searching".

We use the output of algorithm 1 to determine a number of "addresses" for "boxes" to store meet-in-the-middle data in to. As mentioned in section 2.2 there are slight complications in working out which boxes to store information in to increase the probability of good collisions. In our analysis we always ensure that r (the number of initial q-vectors untouched) is large enough so that the storage requirements of the meet-in-the-middle attack is less that 2^r. This way we can use Odlyzko's storage strategy directly without having to consider i for which $T_{i,i} < q$. We note that the following definition could be generalized to handle the case when $|x_i| \leq T_{i,i}/2$ and the error on the x_i is non-constant, unknown but small (rather than the fixed value 1), but this is presently unnecessary.

Definition 2. *For a fixed integer r, and any vector $(x|0)$ with entries satisfying $-q/2 < x_i \leq q/2$ for $1 \leq i \leq r$ we define an associated set, $\mathcal{A}_x^{(r)}$, of r-bit integer "addresses" where $\mathcal{A}_x^{(r)}$ contains every r-bit integer a satisfying both of the following properties:*

[7] This probability depends on the form of T and the effect of Y so can be checked easily.

- *bit* $a_i = 1$ *for all indices* i, $1 \le i \le r$, *such that* $x_i > 1$, *and*
- *bit* $a_i = 0$ *for all indices* i, $1 \le i \le r$, *such that* $x_i \le 0$.

Example 1. To help explain definition 2 we do a simple example with $r = 10$, $q = 11$, and

$$x = (2, 3, -4, -1, 1, 5, -3, -2, 0, 1).$$

In this case there are 2 entries satisfying $x_i = 1$, so

$$\begin{aligned}\mathcal{A}_x^{(r)} = \{&1100010000_2, 1100010001_2\\ &1100110000_2, 1100110001_2\}.\end{aligned}$$

Lemma 3. *When the first* r *entries of* $(x|0)$ *are random integers modulo* q *and independent of each other, then the expected size of* $\mathcal{A}_x^{(r)}$ *is* 2^z *where*

$$z = \sum_{j=0}^{r} j \binom{r}{j} \frac{(q-1)^{r-j}}{q^r}.$$

Proof. Each $x_i = 1$ doubles the entries of $\mathcal{A}_x^{(r)}$ (one with bit $a_i = 0$ and the other with bit $a_i = 1$), so the number of expected entries in $\mathcal{A}_x^{(r)}$ is 2^z where z is the number of expected 1's in the first r entries of x.

Assuming the entries of x are random modulo q and independent of each other, the probability that x has j ones in its first r entries is

$$p_j = \binom{r}{j} \frac{(q-1)^{r-j}}{q^r},$$

and the expected value is therefore given by $z = \sum_{j=0}^{r} j p_j$.

Example 2. In the case $r = 159$ and $q = 197$ then $p_0 \approx 0.45, p_1 \approx 0.36, p_2 \approx 0.14, p_3 \approx 0.04$, so $z \approx 0.36 + 2(0.14) + 3(0.04) \approx 0.76$, and the probability that $z > 3$ is very low (so in practice we discard such x since they are costly to store).

Lemma 4. *If the first* r *entries of a vector* s *are binary, and* $-q/2 < x_i - s_i \le q/2$ *for* $1 \le i \le r$, *then the set* $\mathcal{A}_x^{(r)} \cap \mathcal{A}_{x-s}^{(r)}$ *is non-empty.*

Proof. If the first r entries of a vector s are binary then the sign of the first r entries of $x - s$ are unchanged whenever $x_i > 1$ or $x_i \le 0$. If $0 < x_i \le 1$ then the sign does change but in that case $\mathcal{A}_x^{(r)}$ contained addresses with both choices of bit a_i.

The meet-in-the-middle attack is described in algorithm 2. To analyze its properties we create the following definition.

Definition 3. *A vector* v_1 *of length* $r' = 2N - m$ *is called* s-admissible *if the* x_1, u_1 *gotten from algorithm 1 satisfy:*

$$\begin{aligned}(0|v_1)T - (0|v_1) &= (x_1|0) - (u_1|0)T, \text{ and}\\ (0|v_1)T - (s|v_1) &= (x_1 - s|0) - (u_1|0)T,\end{aligned} \tag{6}$$

i.e. the subtraction of $(s|0)$ *does not affect the multiple of* T *taken away during algorithm 1.*

Algorithm 2. meet-in-the-middle on v

```
loop
    guess a binary vector $v_1$ of length $r' = 2N - m$ with $c$ ones
    use algorithm 1 to calculate $x_1, u_1$ such that $(0|v_1)T - (0|v_1) = (x_1|0) - (u_1|0)T$
    store $v_1$ in the boxes addressed by $a$, for every $a \in \mathcal{A}_{x_1}^{(r)} \cup \mathcal{A}_{-x_1}^{(r)}$
    if there is already a value $v_2$ stored in any of the above boxes then
        let $v = v_1 + v_2$ and use algorithm 1 to calculate $x, u$ such that $(0|v)T - (0|v) = (x|0) - (u|0)T$
        if $(g|f) = (x|v)Y^{-1}$ is binary then
            return $f, g$
        end if
    end if
end loop
```

Lemma 5. *If a vector v_1 is s-admissible, then the vector $v_2 = v - v_1$ is also s-admissible.*

Proof. We have

$$\begin{aligned}(0|v - v_1)T - (0|v - v_1) &= (0|v)T - (0|v) - (0|v_1)T + (0|v_1)\\ &= (s|0) - (u|0)T - (x_1|0) + (u_1|0)T\\ &= (s - x_1|0) - (u - u_1|0)T\end{aligned}$$

and

$$(0|v - v_1)T - (s|v - v_1) = (-x_1|0) - (u - u_1|0)T.$$

Theorem 1. *Let v_1, v_2 be two s-admissible vectors such that $v_1 + v_2 = v$. If x_i, u_i are gotten from applying algorithm 1 to $(0|v_i)T - (0|v_i)$ for $i = 1, 2$, then $x_1 + x_2 = s$.*

Proof. We know

$$\begin{aligned}(0|v_1)T - (s|v_1) &= (x_1 - s|0) - (u_1|0)T\\ (0|v_2)T - (0|v_2) &= (x_2|0) - (u_2|0)T,\end{aligned}$$

where $-T_{i,i}/2 < (x_1)_i - s_i \leq T_{i,i}/2$, $-T_{i,i}/2 < (x_2)_i \leq T_{i,i}/2$, so summing these equations yields

$$\begin{aligned}(0|v)T - (s|v) &= (x_1 + x_2 - s|0) - (u_1 + u_2|0)T\\ (u|0)T &= (x_1 + x_2 - s|0) - (u_1 + u_2|0)T\\ (u - u_1 - u_2|0)T &= (x_1 + x_2 - s|0).\end{aligned}$$

Thus $(x_1)_m + (x_2)_m - s_m = 0$ modulo $T_{m,m}$, but in fact we can deduce $(x_1)_m + (x_2)_m - s_m = 0$ over the integers because of the size restrictions on $(x_1)_m$ and $(x_2)_m - s_m$ (two real numbers modulo $T_{m,m}$ cannot be as large as $2T_{m,m}$). This implies $u_m = (u_1)_m + (u_2)_m$, and given that one can then re-apply a similar argument to coefficients $(m - 1), \ldots, 1$ to realize $x_1 + x_2 = s$, and $u_1 + u_2 = u$.

Theorem 2. *If v_1, v_2 are s-admissible such that $v_1+v_2=v$ and they are chosen in separate loops of algorithm 2 then there exists a box which contains both v_1 and v_2.*

Proof. Since v_1 and v_2 are both admissible and $v=v_1+v_2$ then by theorem 1 we know $x_1+x_2=s$. We know v_2 is contained in all the boxes addressed by $a \in \mathcal{A}^{(r)}_{-x_2} = \mathcal{A}^{(r)}_{x_1-s}$. But v_1 is stored in all the boxes addressed by $a \in \mathcal{A}^{(r)}_{x_1}$ so by lemma 4 there is at least one box which contains both v_1 and v_2.

Remark 1. The problem of estimating the probability that a vector v_1 chosen in step 2 of algorithm 2 is s-admissible can be modelled by the problem of calculating the probability distribution of a coordinate ϵ of a point obtained by multiplying a binary vector times an orthonormal matrix[8]. In particular, the square of a coordinate of the image of a binary vector with d 1's and $m-r-d$ 0's after multiplication by an orthonormal matrix will have $|\epsilon| < \sqrt{d}$ and expected value $E(\epsilon^2) = d/(m-r)$. Denote by $p_d(\delta, \delta')$ the probability that $\delta \leq |\epsilon| < \delta'$ and choose $0 = \delta_1 < \delta_2 < \cdots < \delta_K = \sqrt{d}$. Let $T_{m,m} = q^\alpha$ and assume that the GSA holds and that the density function associated to p_d is decreasing for $\delta \geq \delta_2$. When $i > r$, the ith coefficient of v_1 is $x_i + s_i$ where x_i is uniformly distributed in $[-T_{i,i}/2, T_{i,i}/2]$, where $T_{i,i} = q^{e_i}$. Let's approximate the density function of s_i by a step function on intervals $[\delta_k, \delta_{k+1}]$. In practice, the probabilities $p_d(\delta_k, \delta_{k+1})$ are obtained experimentally, for the choices of the partition by δ_k, but the precise values aren't relevant for the validity of the formula below. More precisely, for each i, $i \geq r+1$ the factors on the right hand side below represent lower and upper approximations to an integral which involves the convolution of the density functions of x_i and s_i.The factor $(1-1/q)^{r/2}$ is present because for $i \leq r$, x_i is a uniform random variable in $[-q/2, q/2)$ and s_i takes on the values $\{0,1\}$ each with probability 1/2. Under these assumptions, and given these approximations, the following "theoretical" computation provides a reality check for a probability determined experimentally. This is the probability p_s that a vector v_1 chosen in step 2 of algorithm 2 is s-admissible. In particular, we have

$$p_s > \left(1 - \frac{1}{q}\right)^{r/2} \prod_{i=r+1}^{m} \left(\sum_{k=1}^{K-1} \left(1 - \frac{\delta_{k+1}}{q^{e_i}}\right) p_d(\delta_k, \delta_{k+1}) \right)$$

$$p_s < \left(1 - \frac{1}{q}\right)^{r/2} \prod_{i=r+1}^{m} \left(\sum_{k=1}^{K} \left(1 - \frac{\delta_k}{q^{e_i}}\right) p_d(\delta_k, \delta_{k+1}) \right)$$

Here $e_i = ((\alpha-1)i + (m-\alpha r))/(m-r)$.

Remark 2. The interval for p_s given by the above inequality comes reasonably close to calculations of p_s obtained by direct sampling and testing (given T and Y). For example, taking the parameters of the $N = 251$ example in the table,

[8] That is, we are modelling Y' as a random orthonormal matrix, which can be approximated by applying the Gram-Schmidt procedure (with normalization) to a random matrix.

with $\alpha = 0.3$, computations show that $2^{-5.95} < p_s < 2^{-6.82}$, while sampling directly gave $p_s = 2^{-6.7}$.

Lemma 6. *The probability that a vector v_1 sampled in step 2 of algorithm 2 is such that $v = v_1 + v_2$, for some v_2 with c ones, is given by*

$$p_h = w\binom{2c}{c}\binom{2N-m}{c}^{-1},$$

where w is the number of rotations of $(g|f)$ resulting in $2c$ distinct ones in v.

Proof. This is just the ratio of the sizes of the respective sets, assuming no intersections of the rotations.

Theorem 3. *The expected number of loops of algorithm 2 before (f, g) is returned is estimated by*

$$L^* = \binom{2N-m}{c}\left(p_s w\binom{2c}{c}\right)^{-1/2},$$

where w is the number of rotations of $(g|f)$ resulting in $2c$ ones in v.

Proof. Let V denote the set of s-admissible vectors v_1 with c ones, such that $v - v_1$ also has c ones. Assuming independence the probability of choosing an element of V in step 2 of algorithm 2 is $p_s p_h$ so we can expect to draw from V about 1 in every $(p_s p_h)^{-1}$ samples. Again assuming independence the size of the set V is $|V| = p_s w\binom{2c}{c}$, and after about $|V|^{1/2}$ samples of the set V we can expect to have sampled a v_1 and a v_2 such that $v_1 + v_2 = v$.

Remark 3. Although the number L^* seems the most natural measure of the cost of the hybrid method, it does ignore the change in the cost per loop in going from Odlyzko's attack to the hybrid attack. We previously estimated the inner-loop cost of Odlyzko's attack as $Nd_f/2$, whereas the corresponding cost of the hybrid scheme is $m^2c/2$ modular additions.

5 Results

To determine the practicality of our attack we have implemented it fully on a small example, and have done a thorough analysis for the `ees251ep6` parameter set, namely: examples with $m = 302$ and $m = 325$ based on actual lattice reduction data, and an example with $m = 344$ based on extrapolated data.

5.1 A Small Example

Algorithm 2 has been fully implemented for a small example with $N = 53$, $q = 37$, $d_f = d_g = 16$. In this example Odlyzko's meet-in-the-middle attack should have taken $2^{20.1}$ loops whereas algorithm 2 has $2^{13.1}$ loops.

With such a small example lattice reduction can often recover the NTRUEncrypt private key, so care was taken to avoid this. We extracted the lower triangular submatrix from rows/columns 24 to 76 inclusive, and LLL-reduced this basis (it took a few seconds using NTL on a 2GHz laptop with 1GB of RAM running Cygwin on a Windows XP platform).

This left the last $r' = 30$ to launch the meet-in-the-middle attack on. We assumed there were 8 ones in these last 30 entries, which was true for 11 of the 57 rotations of (g, f).

We chose many combinations of $c = 4$ ones from these last $r' = 30$, storing the choices in boxes dependent on the output of algorithm 1. After $2^{13.1}$ loops we successfully found a rotation of the (g, f) vector.

On average we stored information in roughly 4 boxes per loop, so the storage complexity was $2^{15.1}$.

5.2 `ees251ep6` with $m = 302$

In the first of the experiments of the `ees251ep6` parameter set we extracted the lower triangular submatrix from rows/columns 160 to 300 inclusive, and BKZ-reduced this basis with blocksize 15. The form of the matrix T is shown in the left side of figure 2.

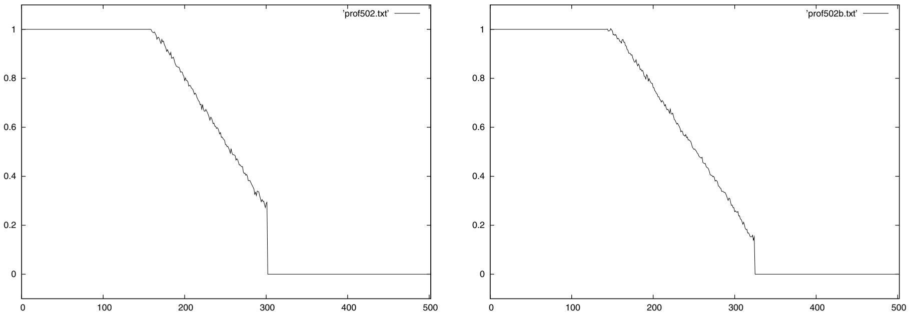

Fig. 2. $\log_{197} |b_i^*|$ for $i = 1, \ldots, 502$. Left: $m = 302$, Right: $m = 325$.

This left the last $r' = 200$ to launch the meet-in-the-middle attack on. We assumed there were 34 ones in these last 200 entries, which was true for 2 of the 251 rotations of (g, f).

We chose many combinations of $c = 17$ ones from these last $r' = 200$, storing the choices in boxes dependent on the output of algorithm 1. Our analysis predicts algorithm 2 will find the NTRUEncrypt private key after $2^{66.9}$ loops.

5.3 A Table of Results with an Extrapolation

From an initial analysis of BKZ lattice running times, the connection between running time and number of q-vectors removed appears to be

$$y = 2378.28 - 132.652x + 23.7371x \log x \tag{7}$$

where it takes 2^y time to remove x q-vectors, and we assume $x > 98$. Out of interest we note that the accuracy of this equation would imply that equation 2 is a severe underestimate of the lattice security of the `ees251ep6` parameter set against BKZ attacks.

Given equation 7 it seems reasonable to assume that in less time than it takes to do $2^{76.2}$ modular additions, one can hope to perform lattice reduction with $r = 136$ and $m = 344$. If this is indeed the case then the security of the `ees251ep6` parameter set is at most that given by $2^{76.2}$ modular additions and a storage requirement of $2^{65.6}$. A stronger initial lattice reduction would improve both of these figures.

N	q	d_f	m	β	t (secs)	r	c	α	p_s	w	L	L^*	#adds	#store
53	37	16	76	2	5	23	4	0.35	$2^{-6.3}$	11	$2^{20.1}$	$2^{13.1}$	$2^{26.6}$	2^{19}
107	67	32	151	15	360	36	7	0.235	$2^{-8.6}$	2	$2^{44.0}$	$2^{28.3}$	$2^{44.6}$	$2^{36.2}$
251	197	48	302	15	780	159	17	0.287	$2^{-6.8}$	9	$2^{84.2}$	$2^{66.9}$	$2^{86.4}$	$2^{76.2}$
251	197	48	325	22	48727	144	14	0.182	2^{-13}	4	$2^{84.2}$	$2^{60.3}$	$2^{79.8}$	$2^{69.4}$
251	197	48	344	$*$	$*$	136	12	0.106	$2^{-20.4}$	4	$2^{84.2}$	$2^{56.7}$	$2^{76.2}$	$2^{65.6}$

The extrapolation was done assuming the GSA [21], which gives the relation

$$\alpha = \frac{2N - m - r}{m - r},$$

where α is as defined in remark 1 and corresponds to the "height of the cliffs" in figure 2, and we modelled Y' as an $(m - r) \times (m - r)$ random orthonormal matrix (these assumptions fit very well with the data from real examples).

We remark that, for a given m, the parameter c was chosen to be minimal such that a randomly chosen f has probability ≥ 0.4 of having at least one rotation with $2c$ ones in the last $2N - m$ entries. The parameter w holds the expected number of such rotations, given that f has at least one such rotation.

The storage complexity "#store" is measured in bits and was assumed to be $8L^* \log_2 \binom{2N-m}{c}$, i.e. that an average of 8 boxes per loop were used to store the v_1's. β denotes the blocksize used in BKZ.

Note that "#adds" count the number of *modular additions* and does not correspond to *bit-security* (see section 2.3 for a further discussion on this distinction). The usefulness of this measure is that it shows the factor of improvement over existing attacks.

6 Lessening Storage Requirements

The storage requirements of the extrapolated data point for the `ees251ep6` parameter set corresponds to a total of $2^{65.6}$ bits. Although this is significantly better than the storage requirements for Odlyzko's attack (2^{94} bits), it is still very expensive given today's hardware. In this section we discuss how to reduce this requirement to a more manageable figure, e.g. a total of $2^{53.6}$ bits of storage[9].

[9] At the time of writing a 1TB hard drive costs about $400, so this amount of storage can be had for approximately $500,000.

The idea is to perform the attack as before, but now we assume we know more about the structure of f. For example in the `ees251ep6` parameter set after reducing the first $m = 344$ rows, we assume there is a rotation of f satisfying:

$\leftarrow 93 \rightarrow$	$\leftarrow 108 \rightarrow$	$\leftarrow D = 50 \rightarrow$
22 ones	$2c' = 20$ ones	$c'' = 6$ ones

When f is randomly chosen with 48 ones, there is a probability of 0.4 that there will be a rotation of f satisfying this pattern (if f is not of this structure the method will end in failure, and another common form should be chosen). Given that there is at least one rotation of f of this form, the expected number of rotations of this form is $w = 3$.

The attacker[10] chooses a fixed vector v_0 of length $r' = 2N - m$ with c'' of the last D entries set to 1, and the remaining entries 0.

We now explain how to slightly modify algorithm 2 to solve CVP with the point $(0|v_0)T$ rather than SVP. In step 2 the algorithm should guess a vector v_1' of length $r' = 2N - m$ such that $v_1 = v_1' + v_0$ has c ones (so v_1' has $c' = 11$ ones in the first $r' - D$ entries, assuming the above form of f).

In step 3 we then calculate the x_1, u_1, and x_1', u_1' corresponding to both v_1 and v_1' respectively. In step 4 we store v_1' in the boxes addressed by a for every $a \in \mathcal{A}^{(r)}_{x_1} \cup \mathcal{A}^{(r)}_{-x_1} \cup \mathcal{A}^{(r)}_{x_1'} \cup \mathcal{A}^{(r)}_{-x_1'}$.

In step 6 we let $v = v_1' + v_2' + v_0$ and apply algorithm 1 as before. It is easy to confirm that theorem 2 still holds where we now have $v_1 = v_1' + v_0$ and $v_2 = v_2'$, so if these are s-admissible then there will be a collision in a box and v can be recovered.

The cost per loop of the modified algorithm is roughly twice the running time and storage of algorithm 2.

The number of loops of the modified algorithm is

$$L^{\#} = \binom{2N - m - D}{c'} \left(p_s w \binom{2c'}{c'} \right)^{-1/2},$$

and the cost per loop is $m^2 c'$ modular additions, and there are $\binom{D}{c''}$ choices for v_0. Thus the total work done is $2^{89.2}$ modular additions, which is still better than Odlyzko's attack $(2^{95.8})$ but now we have brought the storage down to $8L^{\#} \log_2 \binom{2N-m-D}{c'} = 2^{53.6}$ bits, which is a factor of $2^{40.4}$ less than Odlyzko's attack.

7 Generalizations

As mentioned in the introduction, the idea of this paper is really about achieving a meet-in-the-middle technique when one has a set $S = S' \oplus S'$ which contains a vector v_0 which is close to a lattice point of a well-reduced lattice basis. We

[10] Or multiple attackers, since this part is totally parallelizable.

have seen how the idea can be applied to NTRUEncrypt, but there is also hope it can be applied in more generic lattice situations.

To place the approximate-SVP problem on a basis $\mathcal{B} = \{b_1, \ldots, b_n\}$ in to the above framework one can split the basis in to two parts: $\mathcal{B}_1 = \{b_1, \ldots, b_m\}$ and $\mathcal{B}_2 = \{b_{m+1} \ldots, b_n\}$. The set S' can then be generated by linear combinations of $\mathcal{B}_2$ which are small in the space orthogonal to $\mathcal{B}_1$. In [21] Schnorr proposes that $\mathcal{B}_2$ be sampled with the `SHORT` algorithm[11], but many other approaches are possible[12], and indeed other approaches may result in shorter vectors (in the space orthogonal to $\mathcal{B}_1$).

There are several competing criteria dictating what value of m, $1 \leq m \leq n$ one should use: m should be small enough to ensure:

- S' is large enough: An important criteria for the technique to work is that there should be some vector $v_0 \in S = S' \oplus S'$ which, when projected in to the space generated by $\mathcal{B}_1$, is close to a lattice point of $\mathcal{B}_1$. In the case of NTRUEncrypt the structure of the private key guarantees this, but in a more generic lattice situation one must ensure that S' is large enough for there to be a reasonable probability of this being true.
- b_m^* is large enough: For the technique to work m must be chosen such that lattice reduction is likely to b_m^* large enough (with respect to the closeness of v_0 to a lattice point of $\mathcal{B}_1$) for the s-admissible probability to be non-negligible.

However there are also reasons for making m large:

- There need to be enough boxes to store the multiple of $\mathcal{B}_2$ in to: If m is too low there will be too many collisions and the technique will not work. Interestingly this means that lattices resulting from subset sum problems do not seem good candidates for this approach (they typically only contain one non-trivial column).
- Sampling from $\mathcal{B}_2$ should not take too long.

There is hope that the parameter m, and the amount of initial lattice reduction can be fine tuned to optimize this meet-in-the-middle approach, and possibly improve on Kannan's exhaustive search algorithm [17]. Asymptotically we know that Kannan's algorithm will be beaten by sieving techniques [1], but in relatively low dimensions there may be a space for a hybrid algorithm out-performing both existing techniques. We leave the investigation of this idea to future research.

As a final generalization we also note that algorithm 1 is not essential to the method, indeed the s-admissible probability can be improved by using a better CVP algorithm than Babai's closest plane algorithm (e.g. mixing Babai's CVP (which is essentially blocksize 1) with searching in higher blocksizes $2, 3, \ldots$), but this means a more costly CVP approach has to be performed for each loop, so

[11] This can be seen as a slightly modified version of Babai's nearest plane algorithm which takes a binary auxiliary "error" vector as input, and makes a slight error in rounding wherever this vector has a non-zero entry.

[12] For example an exhaustive search of small vectors within some bound.

care should be taken to keep it relatively efficient[13]. When using such a higher blocksize CVP approach it is possible to calculate the addresses for the boxes from the multiple of the rows take away in the CVP process rather than the sign of the x_i.

8 Conclusions

We have demonstrated a new class of attack on the NTRU cryptosystem: one where there is an initial amount of lattice reduction, followed by a generalized meet-in-the-middle procedure.

One way this result can be viewed is as a large strengthening of the result in [5]. In that paper it was shown that lattice reduction sufficient to retrieve a vector of size less that q could be used to break NTRUEncrypt; in this paper we show that far less lattice reduction is needed to mount a successful attack.

With regards to the `ees251ep6` parameter set, we have performed lattice reduction to a sufficient degree to make our method 2^{16} times quicker than Odlyzko's attack, whilst at the same time requiring a factor of $2^{24.6}$ less storage. However, due to the original conservative choice of NTRUEncrypt parameters there still remains a substantially hard problem to recover the NTRUEncrypt private key (primarily due to the storage requirements).

We extrapolated lattice running times to make our method $2^{19.6}$ times quicker than Odlyzko's attack, and requiring a factor of $2^{28.4}$ less storage, but the storage requirements were still substantial.

We have thus modified the attack to require a factor of $2^{40.4}$ less memory, and take time about one hundredth of that of Odlyzko's attack. This is therefore the most practical attack on NTRUEncrypt since its inception in 1996. Progress in lattice reduction will improve our results (both the running time and storage requirements), and so should be factored in if choosing new parameters.

Our attack is still exponential in the security parameter k, so it does not "break" NTRUEncrypt in an asymptotic sense. However to avoid this attack it is imperative to chose parameters so that the meet-in-the-middle attack has complexity 2^k even after an initial lattice basis reduction of complexity 2^k. We also note that when choosing parameters it seems overly-cautious to allow the attacker up to 2^k storage, especially for security levels $k > 80$, but some realistic model of the attackers' storage capabilities should be made.

We observe that in order to defend against this attack it is probably a good idea to "thicken" the NTRUEncrypt private vector (g, f), i.e. to set $d_f = d_g \approx N/2$, or preferably to use a "trinary" vector (g, f) with -1's, 0's, and 1's, to make meet-in-the-middle attacks substantially harder without increasing N considerably.

Acknowledgements. The author would very much like to thank Jeff Hoffstein and Phil Hirschhorn for verifying the analysis in this paper, and Jill Pipher and William Whyte for several useful and stimulating conversations. Jeff and Jill also gave great help with remark 1.

[13] For example, it could be only used where the b_i^* are small.

References

1. Ajtai, M., Kumar, R., Sivakumar, D.: A sieve algorithm for the shortest lattice vector problem. In: Proc. of 29th STOC, pp. 284–293. ACM Press, New York (1997)
2. Babai, L.: On Lovász lattice reduction and the nearest lattice point problem. Combinatorica 6, 1–13 (1986)
3. Cassels, J.W.S.: An introduction to the geometry of numbers, Springer-Verlag, Reprint of the 1st ed. Berlin Heidelberg New York, Corr. 2nd printing 1971, 1997, VIII (1959)
4. Conway, J.H., Sloane, N.J.A.: Sphere Packings, Lattices and Groups. Grundlehren der mathematischen Wissenschaften, 290 (1993)
5. Coppersmith, D., Shamir, A.: Lattice Attacks on NTRU. In: Fumy, W. (ed.) EUROCRYPT 1997. LNCS, vol. 1233, pp. 52–61. Springer, Heidelberg (1997)
6. Fincke, U., Pohst, M.: Improved methods for calculating vectors of short length in a lattice, including a complexity analysis. Math. Comp. 44, 463–471 (1985)
7. Furst, M.L., Kannan, R.: Succinct certificates for almost all subset sum problems. SIAM Journal on Computing 1989, 550–558
8. Gama, N., Howgrave-Graham, N., Nguyen, P.Q.: Rankin's Constant and Blockwise Lattice Reduction. In: Dwork, C. (ed.) CRYPTO 2006. LNCS, vol. 4117, pp. 112–130. Springer, Heidelberg (2006)
9. Gama, N., Howgrave-Graham, N., Nguyen, P.Q.: Symplectic Lattice Reduction and NTRU. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 233–253. Springer, Heidelberg (2006)
10. Hoffstein, J., Pipher, J., Silverman, J.H.: NTRU: A Ring-Based Public Key Cryptosystem. In: Buhler, J.P. (ed.) Algorithmic Number Theory. LNCS, vol. 1423, pp. 267–288. Springer, Heidelberg (1998)
11. Howgrave-Graham, N., Nguyen, P.Q., Pointcheval, D., Proos, J., Silverman, J.H., Singer, A., Whyte, W.: The Impact of Decryption Failures on the Security of NTRU Encryption. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 226–246. Springer, Heidelberg (2003)
12. Howgrave-Graham, N.: Computational Mathematics Inspired by RSA, PhD. Thesis, University of Bath (1998)
13. Howgrave-Graham, N.: Finding Small Roots of Univariate Modular Equations Revisited. IMA Int. Conf. pp. 131–142 (1997)
14. Howgrave-Graham, N., Silverman, J.H., Whyte, W.: Choosing Parameter Sets for NTRUEncrypt with NAEP and SVES-3. In: Menezes, A.J. (ed.) CT-RSA 2005. LNCS, vol. 3376, pp. 118–135. Springer, Heidelberg (2005), `http://www.ntru.com/cryptolab/articles.htm#2005_1`
15. Howgrave-Graham, N., Silverman, J.H., Whyte, W.: A Meet-In-The-Middle Attack on an NTRU Private Key, `http://www.ntru.com/cryptolab/tech_notes.htm#004!`
16. W. Whyte, (ed.) IEEE P1363, 1/D9 Draft Standard for Public-Key Cryptographic Techniques Based on Hard Problems over Lattices
17. Kannan, R.: Improved algorithms for integer programming and related lattice problems. In: Proc. of the 15th Symposium on the Theory of Computing (STOC 1983), pp. 99–108. ACM Press, New York (1983)
18. Philip, N.: Finding the closest lattice vector when it's unusually close. In: Proceedings, ACM-SIAM Symposium on Discrete Algorithms, pp. 937–941. ACM, New York (2000)

19. Lenstra, A., Verheul, E.: Selecting Cryptographic Key Sizes. Journal of Cryptology 14(4), 255–293 (2001)
20. Pohst, M.: On the computation of lattice vectors of minimal length, successive minima and reduced bases with applications. ACM SIGSAM Bull. 15, 37–44 (1981)
21. Schnorr, C.P.: Lattice Reduction by Random Sampling and Birthday Methods. In: Alt, H., Habib, M. (eds.) STACS 2003. LNCS, vol. 2607, pp. 145–156. Springer, Heidelberg (2003)
22. Schnorr, C.P., Euchner, M.: Lattice Basis Reduction: Improved Practical Algorithms and Solving Subset Sum Problems. Mathematical Programming 66, 181–191 (1994)
23. Shoup, V.: NTL: A Library for doing Number Theory, Version 5.4, `http://www.shoup.net/ntl`
24. Wagner, D.: A Generalized Birthday Problem. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 288–303. Springer, Heidelberg (2002), `http://www.cs.berkeley.edu/daw/papers`