Hybrid Lattice Reduction and Meet in the Middle Resistant Parameter selection for NTRUEncrypt

Jeffrey Hoffstein, Nick Howgrave-Graham, Jill Pipher, Joseph H. Silverman, Wiliam Whyte

NTRU Cryptosystems, Inc.

Abstract. We present a new method for choosing parameter sets for the NTRUEncrypt public key cryptosystem. This modifies our original method of [5] to protect against the new hybrid meet in the middle and lattice reduction attack of [4]

1 Introduction and Notation

In [4] a new method was developed for locating an NTRUEncrypt private key. This method combined lattice reduction with a meet in the middle approach. The effect of this attack was to decrease the bit security of the private key, at the cost of increasing memory usage. For example, the original $N = 251, q = 197, d_f = 48, d_g = 125$ parameter set which in [5] was originally thought to have security on the order of 80 bits, was reduced to approximately 60.3 bits.

In this note we will assume familiarity with the details and notation of NTRUEncrypt The reader desiring further background should consult standard references such as [1, 2, 5].

The attack of [4] was particularly successful because for reasons of efficiency the private key f was chosen to be binary, and to consist of the smallest possible number of 1's consistent with combinatorial security restraints. With the new perspective of [4] we now see that it is necessary to enlarge the combinatorial search size of the space f is chosen from. Fortunately, only a small modification turns out to be necessary, and the impact on efficiency is not great.

In previous parameter choices for NTRUEncrypt we have often taken the polynomials f, g to be binary, that is, drawn at random from a space of polynomials whose coefficients consist only of 0 or 1. To increase the size of our combinatorial search spaces while having a minimal impact on efficiency, we shift our focus now to *trinay polynomials*. These are polynomials whose coefficients are all 0's, 1's and -1's. We set the notation:

$$\mathcal{T}_N = \{\text{trinary polynomials}\},$$
$$\mathcal{T}_N(d, e) = \left\{ \begin{array}{l} \text{trinary polynomials with exactly} \\ d \text{ ones and } e \text{ minus ones} \end{array} \right\}.$$

If N is fixed, we drop it from the notation and write $\mathcal{T}$ and $\mathcal{T}(d, e)$. A further advantage to using trinary polynomials is that q can be chosen to be a power of 2, a choice that makes reductions modulo q particularly easy.

Remark 1. No attacks are presently known that would take advantage of the fact that q would not be prime, as N must certainly be. However the truly conservative user might wish to restrict q to prime values.

NTRUEncrypt uses the *ring of truncated polynomials* (also sometimes called the *ring of convolution polynomials*)

$$\mathbf{Z}[X]/(X^N - 1).$$

We denote multiplication in this ring by $*$. At the final stage of the decryption process, a certain polynomial is computed:

$$a = p * r * g + m * f,$$

where $p = 3$ and $r, f, g, m \in \mathcal{T}_N$ are trinary polynomials. The decryption process succeeds if the maximum and minimum coefficients of a satisfy

$$\min a_j > -\frac{1}{2}q \qquad \text{and} \qquad \max a_j < \frac{1}{2}q.$$

In more efficient implementations of NTRU, the polynomial f is taken to have the form $f = 1 + p * F$, where $F \in \mathcal{T}_N$ is trinary, so now a has the form

$$a = p * (r * g + m * F) + m.$$

Remark 2. For the most part we will choose parameters consistent with the more efficient implementation. Sample parameter sets are given in Tables 1,3,5,6. Such parameters are also, of course, valid for the less efficient trinary implementation, although in this version the value of q can be decreased if desired. Using the implementation where f is trinary rather than of the form $f = 1 + p * F$ necessitates extra operations in the decryption process, but allows for a reduction in key size in exchange. This tradeoff is particularly pronounced when $q = 256$, and so we do provide a table of parameters in this case (Table 2).

If we let $b = r * g + m * F$, then decryption succeeds if the coefficients of b satisfy

$$\min b_j > -\frac{1}{p}\left(\frac{1}{2}q - 1\right) \qquad \text{and} \qquad \max b_j < \frac{1}{p}\left(\frac{1}{2}q - 1\right).$$

The probability that $\max b_j$ is larger than some given value K is exactly the quantity $P_3(K)$ that was studied in [3] and we will make extensive use of the formulas of [3] in our calculations. (For the trinary case we use the slightly modified probability formula for $P_4(K)$ given in [3].)

We will assume that each trinary polynomial r, g, m, F that we use is drawn from a space $\mathcal{T}_N(d, e)$ with

$$d, e \le d_f + 1 \le 1 + \lfloor N/3 \rfloor.$$

To establish parameters with bit security k we will only search among triples d_f, q, N such that for $r, g, m, F \in \mathcal{T}_N(d_f+1, d_f+1)$ the probability that decryption fails is less than 2^{-k}. For such r, g, m, F by symmetry, the max and min conditions on the coefficients have equal probability. Hence

$$\begin{aligned}\text{Prob(Decryption fails)} &\le 2\,\text{Prob}\left(\max b_j \ge \frac{1}{p}\left(\frac{1}{2}q-1\right)\right)\\ &\le 2P_3\left(\frac{1}{p}\left(\frac{1}{2}q-1\right)\right) \qquad (1)\end{aligned}$$

Clearly for any choice of $d, e \le d_f+1$, the probability of decryption failure will also be less than 2^{-k} for $r, g, m, F \in \mathcal{T}_N(d, e)$.

2 A review of the hybrid attack

In this section we will review the method of [4]. We will also make some substantial simplifications. These clarify the analysis by giving a somewhat undeserved advantage to an attacker. Thus the security estimates we arrive at will be quite conservative but will have the advantage of being relatively transparent.

The structure of the argument of [4] is simpler for the less efficient version of NTRU where the public key has the form $h \equiv f^{-1} * g \pmod q$ and $f, g \in \mathcal{T}_N(d, e)$. The rough idea is as follows. Suppose one is given N, q, d, e, h and hence implicitly an NTRUEncrypt public lattice L of dimension $2N$. The problem is to locate the short vector corresponding to the secret key (f, g). One first chooses $N_1 < N$ and removes a $2N_1$ by $2N_1$ lattice L_1 from the center of L. Thus the original matrix corresponding to L has the form

$$\left(\begin{array}{c|c} qI_N & 0 \\ \hline H & I_N \end{array}\right) = \left(\begin{array}{c|c|c} qI_{N-N_1} & 0 & 0 \\ \hline * & L_1 & 0 \\ \hline * & * & I_{N-N_1} \end{array}\right) \qquad (2)$$

and L_1 has the form

$$\left(\begin{array}{c|c} qI_{N_1} & 0 \\ \hline H_1 & I_{N_1} \end{array}\right). \qquad (3)$$

Here H_1 is a truncated piece of the circulant matrix H corresponding to h appearing in (2).

Let us suppose that an attacker must use a minimum of k_1 bits of effort to reduce L_1 until all N_1 of the q-vectors are removed. When this is done and L_1 is put in lower triangular form the entries on the diagonal will have values $\{q^{\alpha_1}, q^{\alpha_2}, \ldots, q^{\alpha_{2N_1}}\}$, where $\alpha_1 + \cdots + \alpha_{2N_1} = N_1$, and the α_i will come very close to decreasing linearly, with

$$1 \approx \alpha_1 > \cdots > \alpha_{2N_1} \approx 0.$$

That is to say, L_1 will roughly obey the geometric series assumption, or GSA. This reduction will translate back to a corresponding reduction of L, which when reduced to lower triangular form will have a diagonal of the form

$$\{q, q, \ldots, q, q^{\alpha_1}, q^{\alpha_2}, \ldots, q^{\alpha_{2N_1}}, 1, 1, \ldots, 1\}.$$

The key point here is that it requires k_1 bits of effort to achieve this reduction, with $\alpha_{2N_1} \approx 0$. If $k_2 > k_1$ bits are used then the situation can be improved to achieve $\alpha_{2N_1} = \alpha > 0$. As k_2 increases the value of α is increased.

In the previous work the following method was used to launch the meet in the middle attack. It was assumed that the coefficients of f are partitioned into two blocks. These are of size N_1 and $K = N - N_1$. The attacker guesses the coefficients of f that fall into the K block and then uses the reduced basis for L to check if his guess is correct. The main observation of [4] is that a list of guesses can be made about half the coefficients in the K block and can be compared to a list of guesses about the other half of the coefficients in the K block. With a probability $p_s(\alpha)$ a correct matching of two half guesses can be confirmed, where $p_s(0) = 0$ and $p_s(\alpha)$ increases monotonically with α. In [4] a value of $\alpha = 0.182$ was used with a corresponding probability $p_s(0.182) = 2^{-13}$. The probability $p_s(0.182)$ was computed by sampling and the bit requirement, k_2 was less than 60.3. In general, if one used k_2 bits of lattice reduction work to obtain a given $p_s(\alpha)$ (as large as possible), then the number of bits required for a meet in the middle search through the K block decreases as K decreases and as $p_s(\alpha)$ increases.

A very subtle point in [4] was the question of how to optimally choose N_1 and k_2. The objective of an attacker was to choose these parameters so that k_2 equaled the bit strength of a meet in the middle attack on K, given the $p_s(\alpha)$ corresponding to N_1. It is quite hard to make an optimal choice, but it is considerably easier to make this choice if one makes the simplifying assumption, in the attacker's benefit, that $p_s(\alpha) = 1$, even as α approaches 0.

We will make this simplifying assumption in this paper. Thus our question reduces to the problem of computing the complexity of the lattice reduction process required to remove N_1 of the q-vectors in the original lattice L. Once this is done the result is compared with the meet in the middle search on K. When these two bit strengths are equal the attackers task will be optimized. In the next section we examine the work required to remove q-vectors.

Remark 3. It should be stressed that the simplifying assumption we make is to the attacker's benefit. Thus the parameters we arrive at are stronger than claimed. In future work we hope to include a $p_s(\alpha)$ analysis.

3 Quantifying the removal of q-vectors

We performed a number of experiments applying the BKZ package in the NTL library of [6] to matrices of dimension $2z$ by $2z$ of the form

$$\left(\begin{array}{c|c} qI_z & 0 \\ \hline H_z & I_z \end{array}\right) \tag{4}$$

Here q would be fixed, z would be a number less than the number of q-vectors that we expected to remove at the security level we were investigating, and H_z would be a z by z portion of the circulant associated to an NTRU public key h (modulo q). We applied BKZ at block sizes 10 through 21 or 22 and ran between 10 and 100 repetitions of each block size experiment. For each block size b the initial number of q-vectors was z and we computed $N_1(b)$ equal to z minus the number of remaining q-vectors in the output matrix from the BKZ reduction. That is, $N_1(b)$ represented the number of q-vectors consumed by BKZ with block size b.

We discovered that the behavior of BKZ under these conditions was far more regular than its behavior when searching for the private key of an NTRU lattice. For example, we took $z = 150$ and $q = 1024$ and ran 100 tests (i.e. different H_z) on each block size 10 to 20, and 10 on block size 21. Here are values of standard deviation of time in bits vs block size b:

$\{\{10, 0.135426\}, \{11, 0.138498\}, \{12, 0.163593\}, \{13, 0.141772\}, \{14, 0.154801\}, \{15, 0.157524\},$

$\{16, 0.169264\}, \{17, 0.191727\}, \{18, 0.242162\}, \{19, 0.254833\}, \{20, 0.456409\}, \{21, 0.744065\}\}$

and here are values of standard deviation of $N_1(b)$ vs block size b.

$\{\{10, 1.21911\}, \{11, 1.27165\}, \{12, 1.27493\}, \{13, 1.10534\}, \{14, 1.09803\}, \{15, 1.12908\},$

$\{16, 1.10031\}, \{17, 1.17099\}, \{18, 1.02516\}, \{19, 0.970972\}, \{20, 0.993557\}, \{21, 0.632456\}\}.$

In fact, $N_1(b)$ appeared to increase linearly with b as Figure 1, a graph of the mean of $N_1(b)$, over all experiments at blocksize b, against b shows:

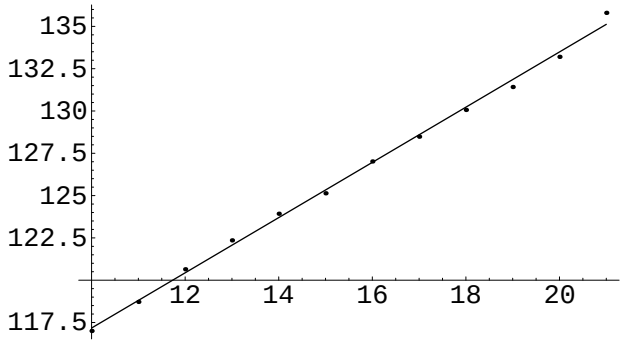

Fig. 1. Mean number of consumed q-vectors $N_1(b)$ against block size, $q = 1024$

The relevant data is:

$\{\{10, 117.\}, \{11, 118.718\}, \{12, 120.645\}, \{13, 122.355\}, \{14, 123.927\}, \{15, 125.136\},$

$\{16, 127.018\}, \{17, 128.482\}, \{18, 130.064\}, \{19, 131.418\}, \{20, 133.2\}, \{21, 135.8\}\}$

and the closest fit line in Figure 1 has equation

$$N_1(b) = 100.857 + 1.6316b. \tag{5}$$

As the number of bits required to compute a given block size b, for fixed z is $\mathcal{O}(b \log b)$ (with the implied constant depending on z and q) it's reasonable to guess that a plot of the mean value of $N_1(b)$ against bits will have a good fit to a $b \log b$ curve. In fact, our results give us the following list for the mean value of $N_1(b)$ paired with the mean number of bits required for the computation, as b varies from 10 to 21:

$$\{\{117., 38.6164\}, \{118.718, 38.8509\}, \{120.645, 39.0626\}, \{122.355, 39.3132\},$$
$$\{123.927, 39.5661\}, \{125.136, 39.8229\}, \{127.018, 40.1786\}, \{128.482, 40.5165\},$$
$$\{130.064, 40.9629\}, \{131.418, 41.5226\}, \{133.2, 42.4924\}, \{135.8, 44.937\}\}.$$

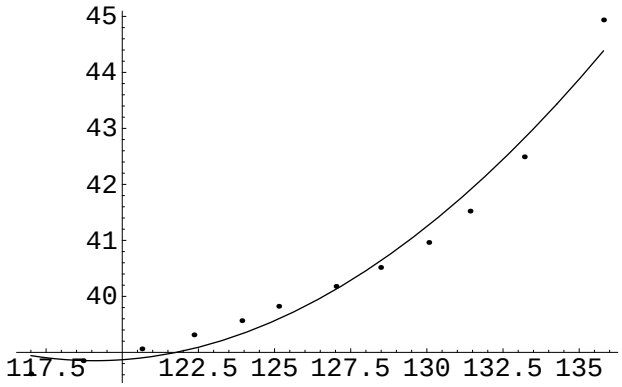

Fig. 2. Bits required to remove q-vectors from a 300 by 300 block, $q = 1024$, blocks 10 to 21

The equation of the curve in Figure 2 is

$$626.339 - 28.5139N_1 + 4.93334N_1 \log(N_1).$$

Notice that there is a reasonably nice fit of the points to the curve, but the behavior seems to alter as b increases. In particular, if we only plot the points corresponding to blocksizes 17 through 21 we obtain a considerably better fit:

The equation of the curve in Figure 3 is

$$2198.48 - 98.946N_1 + 16.918N_1 \log(N_1). \tag{6}$$

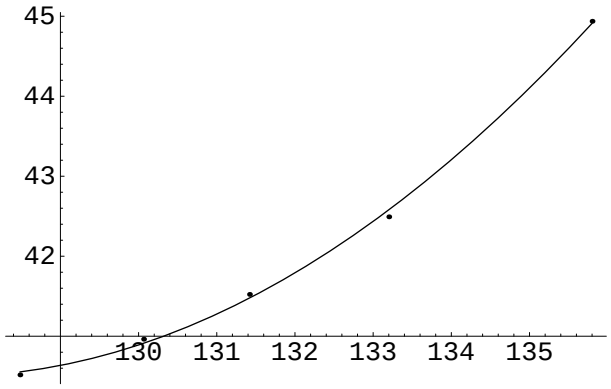

Fig. 3. Bits required to remove q-vectors from a 300 by 300 block, $q = 1024$, blocks 17 to 21

Notice in particular how the coefficient of the $N_1 \log(N_1)$ in (6) has increased from 4.9 to 16.9. It seems that the fundamentally exponential nature of this problem, at least when approached by the BKZ technique, becomes more apparent as the block size increases.

We have performed similar experiments for $q = 256$ and $q = 2048$ with similar results. Interestingly, as q increases the number of q-vectors removed by a given block size increases. For example, we present below the analogous results for $q = 256$. These were obtained by taking 10 samples each from block sizes 15 to 22.

Again $N_1(b)$ appears to increase linearly with b as Figure 4, a graph of the mean of $N_1(b)$, over all experiments at blocksize b, against b shows:

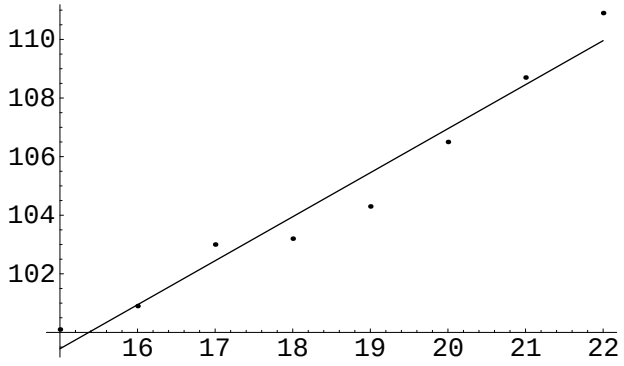

Fig. 4. Mean number of consumed q-vectors $N_1(b)$ against block size, $q = 256$

The relevant data is:

$$\{\{15, 100.1\}, \{16, 100.9\}, \{17, 103.0\}, \{18, 103.2\},$$

$$\{19, 104.3\}, \{20, 106.5\}, \{21, 108.7\}, \{22, 110.9\}\}$$

and the closest fit line in Figure 4 has equation

$$N_1(b) = 76.906 + 1.50238b. \tag{7}$$

Note that slope of (7) is 1.50238 versus the 1.63 slope of (5), indicating the harder time BKZ has removing q-vectors when q is smaller.

In the case $q = 256$ our results give us the following list for the mean value of $N_1(b)$ paired with the mean number of bits required for the computation, as b varies from 15 to 22:

$$\{\{100.1, 38.1727\}, \{100.9, 38.3585\}, \{103., 38.9201\}, \{103.2, 39.1194\},$$

$$\{104.3, 39.6493\}, \{106.5, 40.6388\}, \{108.7, 42.5249\}, \{110.9, 46.0674\}\}.$$

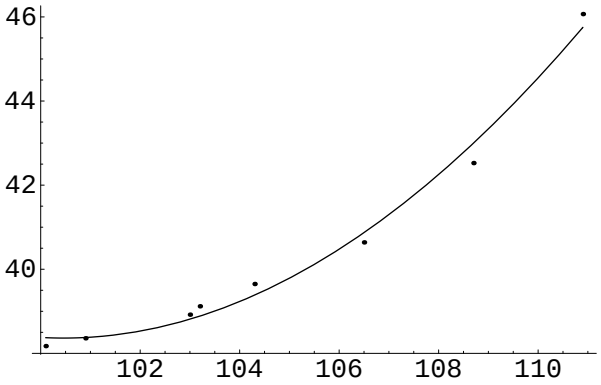

Fig. 5. Bits required to remove q-vectors from a 240 by 240 block, $q = 256$, blocks 15 to 22

The equation of the curve in Figure 5 is

$$1453.47 - 79.02N_1 + 14.0861N_1 \log(N_1).$$

Notice that there is a reasonably nice fit of the points to the curve, but the as in the case of $q = 1024$, the behavior seems to alter as b increases. In particular, if we only plot the points corresponding to blocksizes 20 through 22 we obtain a considerably better fit:

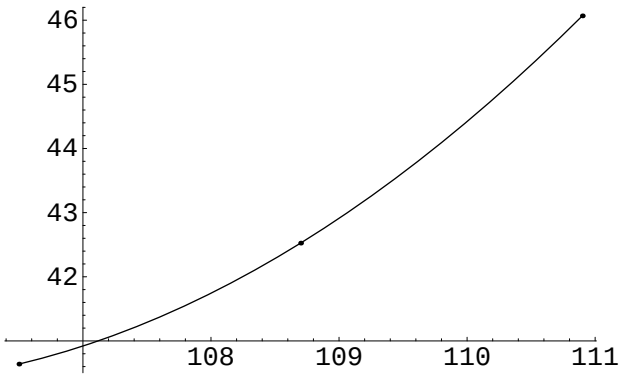

Fig. 6. Bits required to remove q-vectors from a 250 by 250 block, $q = 256$, blocks 20 to 22

The equation of the curve in Figure 6 is

$$3951.88 - 210.385N_1 + 37.201N_1 \log(N_1). \tag{8}$$

Notice in particular how the coefficient of the $N_1 \log(N_1)$ in (8) has increased from 14.0861 to 37.201. Again the fundamentally exponential nature of this problem, at least when approached by the BKZ technique, becomes more apparent as the block size increases.

In the case of $q = 2048$ we have performed 100 experiments at each of the block sizes 15 through 19. (We intend to increase this by the time this paper is released.) The results have the same form as $q = 256, q = 1024$ and the analogous extrapolation equation is

$$1450.46 - 60.3175N_1 + 10.1684N_1 \log(N_1). \tag{9}$$

Remark 4. A relevant question is: "How does the choice of z, that is, the size of the block that one takes from the center of the matrix, affect the block size and the number of bits required to remove a given number of q-vectors?". The most important thing to note is that as long as z is less than the number of q-vectors one intends to remove, the block size is independent of z. What does depend on z, though in a smaller way, is the number of bits necessary to accomplish a reduction at a given block size. However this method of extrapolation of the number of bits required to eliminate N_1 q-vectors will always give an underestimate if z is smaller than N_1. This is the case for all our estimates, and thus these bit estimates are all conservative.

We can now use equations (6), (8) and (9) to extrapolate the number of bits required to remove a given number N_1 of q-vectors. This is how the N_1 column was computed in Tables 1 - 6.

4 The meet in the middle search

Let us suppose now that an attacker has eliminated N_1 of the q-vectors. The number of coefficients of f that must be guessed is $K = N - N_1$. Suppose that f consists of d 1's, e -1's and the rest zeros. Then for some choice of d_1, d_2 with $0 \le d_1 \le d$ and $0 \le d_2 \le e$, the K coefficients will contain d_1 1's and d_2 -1's. An attacker must make a correct choice of d_1, d_2 before beginning his meet in the middle search. In fact, any rotation of f will suffice to locate the key, and so any rotation of f that has d_1 1's and d_2 -1's in K pre-selected locations will give a successful outcome to the attacker. To simplify the analysis and benefit the attacker we will assume that all N rotations of f have exactly d_1 1's and d_2 -1's in the K locations.

Thus if the attacker has guessed correctly, the meet in the middle size of the search space is

$$T(N_1, K, d_1, d_2) = \sqrt{\frac{\binom{K}{d_1}\binom{K-d_1}{d_2}}{N_1 + K}}. \tag{10}$$

Now we must consider the probability that for f (having exactly d 1's and e -1's) a random choice of K coefficients of f contains exactly d_1 1's and d_2 -1's. This is easily seen to be given by

$$P(N_1, K, d_1, d_2) = \frac{\binom{K}{d_1}\binom{N_1}{d-d_1}\binom{K-d_1}{d_2}\binom{N_1-d+d_1}{e-d_2}}{\binom{N_1+K}{d}\binom{N_1+K-d}{e}}. \tag{11}$$

The strength of a set of parameters $\{N_1, K, d, e\}$ is then bounded below by the minimum over all permissible pairs d_1, d_2 of the ratio T over P. We thus define the *bit strength* of the parameter set $\{N_1, K, d, e\}$ by

$$S(N_1, K, d, e) = \min \frac{\log\left(\frac{T(N_1,K,d_1,d_2)}{P(N_1,K,d_1,d_2)}\right)}{\log(2)}, \tag{12}$$

where $T(N_1, K, d_1, d_2)$ is given by (10), $P(N_1, K, d_1, d_2)$ is given by (11), and the minimum is taken over all pairs d_1, d_2 satisfying $0 \le d_1 \le d$ and

$$\max(0, d + e - N_1 - d_1) \le d_2 \le \max(\min(K - d_1, e), 0).$$

5 A strictly lattice based search for the key

Suppose we have located a triple d_f, q, N, with $N = N_1 + K$, such that the probability of decryption failure using the $f = 1 + 3F$ protocol is less than 2^{-k} (as measured by (1)), the strength against a meet in the middle attack (as measured by (12)) is at least k, and such that the number of bits required by the BKZ algorithm to remove N_1 of the q-vectors is at least k. There is still one more requirement that must be met.

It is possible that if d_f is very small compared to N and q that the time required for BKZ to locate the actual key might be less than 2^k. This is because

the characteristics of the BKZ algorithm when searching for an extremely small vector in a lattice (that is, very much shorter than the expected shortest vector) are different from the characteristics of the BKZ algorithm when reducing a generic lattice. In particular, suppose a target vector τ in an NTRU lattice is selected from $\mathcal{T}_N(d_f+1, d_f)$, so $\|\tau\| = 2d_f+1$. The size of the Gaussian expected shortest vector is

$$\sigma = \sqrt{\frac{Nq}{\pi e}}.$$

It has been discussed before, for example in [5], that the constant c defined by the relation

$$c = \frac{\|\tau\|}{\sqrt{2N}} \cdot \sigma$$

is useful for measuring the time and block size required for BKZ to locate τ. For a sequence of triples d_f, q, N such that N/q is held constant and c is held constant, the time required for BKZ to locate τ has been observed (but not proved) to be at least exponential in N as N increases. As c decreases, the time required for BKZ to succeed decreases, while remaining exponential in N. Thus we must consider the possibility that a particularly small value of d_f relative to N and q might make it unexpectedly easy for BKZ to locate the key.

In the new context of parameter generation that we are considering here, there is a different sequence of lattices which is a bit more convenient for our purposes in measuring and extrapolating the strength of the BKZ algorithm. We will find it useful to understand the properties of BKZ when applied to a sequence of NTRU lattices with increasing N, but with fixed q and d_f.

Proceeding in this way we have run a number of experiments that suggest that for fixed q, d_f the block size required by BKZ to locate the key increases at least linearly with N. For example, see the following graph of block size necessary to locate the key against N for $q = 1024, d_f = 29$

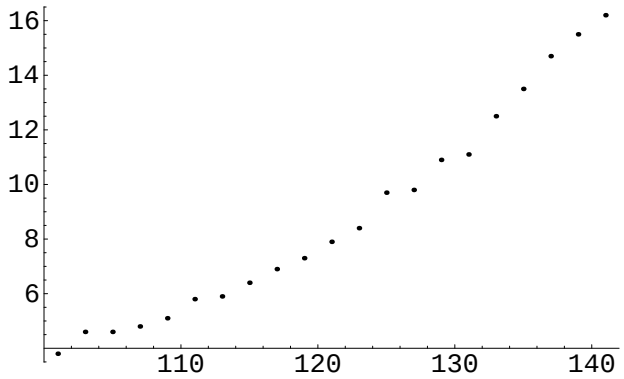

Fig. 7. Block size required to locate the private key for $q = 1024, d_f = 29$.

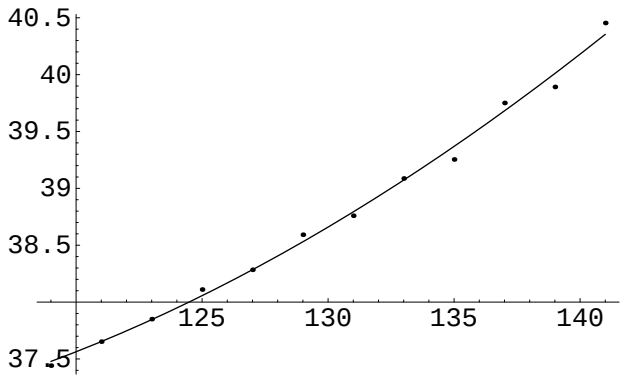

Fig. 8. Bits required to locate the private key for $q = 1024, d_f = 29$ (tail).

As the time in bits for BKZ to do a block reduction of size b is $\mathcal{O}(b \log b)$ it follows that the time in bits required for BKZ to locate the key in a lattice of dimension $2N$ should be at least $\mathcal{O}(N \log N)$. We thus take our experimental results and extrapolate by using the best fit to a curve of the form

$$\text{time in bits} = AN \log N + BN + C$$

that lies under the latter part of our data. We use the points coming from as large N and block sizes as we can experimentally reach in a reasonable amount of time. Our results suggest, that continuing to higher N and higher block sizes will always result in higher estimates of bits required for breaking times. For the example above, in Figure 8, the extrapolating equation is

$$93.2501 - 3.10353N + 0.551326N \log(N). \tag{13}$$

Remark 5. The time required by BKZ to locate a key for fixed q and N increases as d_f increases. Thus to determine that key recovery by BKZ will take at least k bits it suffices to have performed experiments with any $d'_f < d_f$ that predict greater than k bits. For example, in Table 5 the lowest d_f mentioned is $d_f = 29$, and the lowest N on the table is $N = 257$ (though associated with a *higher* d_f). Substituting $N = 257$ into (13) we see that key security is at least 81 bits for all N above 257 and d_f above 29. The next lowest N is $N = 313$, required for 112 of security (again with a higher d_f). Substituting $N = 313$ into (13) we see that key security is at least 113 bits for all N above 313 and d_f above 29. Continuing, $N = 347$ produces at least 135 bits of security, $N = 421$ produces at least 189 bits of security, and $N = 521$ produces at least 273 bits of security. This suffices for the entire $q = 1024$ table. In fact the key security requirement of this section must always be checked, but in all cases investigated so far it has turned out to be an easier requirement to satisfy (i.e. weaker) than the hybrid meet in the middle attack.

6 Balancing the requirements

Recall we have assumed that each trinary polynomial r, g, m, F that we use is drawn from a space $\mathcal{T}_N(d, e)$ with

$$d, e \leq d_f + 1 \leq 1 + \lfloor N/3 \rfloor.$$

Remark 6. To improve the clarity of the exposition the meet in the middle argument in the previous section was built around the case $f \in \mathcal{T}_N(d, e)$. However the case $f = 1 + 3 * F$, with $F \in \mathcal{T}_N(d, e)$ is argued the same way, with identical lower bounds on security.

For concreteness, we'll now set $d = d_f + 1, e = d_f$. To establish parameters with bit security k we will only search among triples d_f, q, N such that for all $r, g, m, F \in \mathcal{T}_N(d_f + 1, d_f)$ the probability that decryption fails, as guaranteed by (1), when using the $f = 1 + 3F$ protocol, is less than 2^{-k}. We will use the following procedure to choose a parameter triple which has minimal d_f:

- For a selection of q, for example $q = 1024, 256, 2048$, construct extrapolation equations such as (6), (8) and (9). Use these equations to determine, for each q, an associated N_1 such that BKZ reduction requires at least k bits of effort to remove N_1 of the q-vectors.
- For each q, N_1 pair choose any $d_0 \leq \lfloor q/24 \rfloor - 1$ and such that d_0 is at least as large as any d_f for which experiments have been run and extrapolation equations such as (13) have been constructed. The maximum possible width of a polynomial occurring in decryption is $24d_f + 2$, [3], and thus the probability of decryption failure for any choice of N corresponding to this d_0 is 0.
- Use (12) to determine the minimal K for which $N = N_1 + K$ is prime and

$$S(N_1, K, d_0 + 1, d_0) > k.$$

- Now verify via some version of (13) for the given q for which the experimental choices of d_f, N are smaller than the values being tested, that the time in bits required for BKZ to locate the key is greater than k. If so, then the triple d_0, q, N will satisfy the security requirements for bit strength k. This will be the choice for the given q with minimal $d_f = d_0$. If N is not sufficiently large to ensure k bits of security by (13) then replace N by the smallest prime N' for which this is true and set $K = N' - N_1$. It is easy to see that the requirement

$$S(N_1, K, d_0 + 1, d_0) > k$$

will still hold, and of course decryption failure probability will still be 0. Thus setting $d_f = d_0, N = N'$ will give us a decryption triple for q at security level k with minimal d_f.

Assuming we have found a triple d_f, q, N at security level k with minimal d_f, we may now, for $d'_f > d_f$ try to select a new $N' < N$. This will have the

effect of reducing key size. To accomplish this, we search among pairs d'_f, K', with $N' = N_1 + K'$, such that $d'_f \leq \lfloor (N_1 + K')/3 \rfloor$, $K' \leq K$ and N' is prime, that satisfy the requirements that the probability of decryption failure as given by (1), is less than 2^{-k}, and that the meet in the middle strength, given by (12) is at least k. Of these, the choice with $2d'_f N'$ minimized will be most efficient, with the smallest number of operations performed, while the choice with minimal N' will have the smallest key size.

7 Sample parameter generation

Consider the case $q = 1024, k = 80$. Referring to (6) we see that choosing $N_1 = 153$ will suffice to achieve 80 bits of security from the lattice reduction part of the meet in the middle approach. We next choose $d_0 = 24 < 41 = \lfloor q/24 \rfloor - 1$ and compute via (12) that $K = 394$ is the minimal K for which $N = 153 + 394 = 547$ is prime and $S(153, 394, 25, 24) > 80$. Thus $d_f = 24, N = 547$ will give 80 bits of security when $q = 1024$. For this particular choice, $2d_f N = 26256$. Increasing d_f and decreasing N we find that $2d_f N$ hits a minimum at $d_f = 29, N = 439$, with $2d_f N = 25462$. This gives our minimal operations triple for $k = 80$ and $q = 1024$, and we don't have to check decryption failure probability as $29 < 41$.

We now increase d_f by steps. For each d_f we decrease N, until we reach the minimal N for which $S(153, K, d_f + 1, d_f) > 80$ and the probability of decryption failure via (1) is less than 2^{-80}. For example, we might overshoot to $d_f = 85$ and compute via (12) that $K_1 = 103$ and $d_f = \lfloor (153 + 103)/3 \rfloor = 85$ satisfy $S(153, 103, 86, 85 > 80$. The next prime after $256 = 153 + 103$ is $N = 257$ and so we compute via (1) that for polynomials chosen from $\mathcal{T}_{257}(86, 86)$ the probability of a positive coefficient exceeding 168 is less than $2^{-83.2}$. Thus any value of q satisfying

$$\frac{1}{3}\left(\frac{1}{2}q - 1\right) \geq 168,$$

i.e, $q \geq 1011$ will suffice for our purposes, in particular the value $q = 1024$. Thus $d_f = 85, q = 1024, N = 257$ is a parameter triple giving at least 80 bits of security. In fact, tinkering a bit with this we find that we can reduce d_f to 77 while keeping $S(153, 104, 78, 77) > 80$ and so $d_f = 77, q = 1024, N = 257$ is a slightly more efficient parameter triple with the same key size giving at least 80 bits of security.

Following this procedure we arrive at the following tables:

References

1. J. Hoffstein, J. Pipher, J.H. Silverman, NTRU: A new high speed public key cryptosystem, Algorithmic Number Theory (ANTS III), Portland, OR, June 1998, Lecture Notes in Computer Science 1423, J.P. Buhler (ed.), Springer-Verlag, Berlin, 1998, 267–288

k	N	N_1	K	d_f	Lattice bits	$2d_fN$	bits in key
80	449	121	328	24	80	21552	3592
80	479	128	351	24	120	22992	3832
112	839	126	713	24	112	40272	6712
112	877	134	743	24	168	42096	7016
128	1223	129	1094	24	128	58704	9784
128	1231	136	1095	24	192	59088	9848
160	2273	133	2140	24	160	109104	18184
160	2357	141	2216	24	240	113136	18856
192	3967	136	3853	25	192	199450	31912
192	4111	145	3966	25	288	205550	32888
256	13963	142	13821	26	256	726076	111704
256	14303	153	14150	26	384	743756	114424

Table 1. Sample parameter choices, $q = 256$, $f = 1 + 3F$.

k	N	N_1	K	d_f	Lattice bits	$2d_fN$	bits in key
80	337	121	216	31	80	20894	2696
80	353	128	225	31	120	21886	2824
112	557	126	431	31	112	34534	4456
112	577	134	443	31	168	35774	4616
128	719	129	590	31	128	44578	5752
128	739	136	603	31	192	45818	5912
160	1193	133	1060	31	160	73966	9544
160	1229	141	1088	31	240	76198	9832
192	2017	136	1881	31	192	125054	16136
192	2089	145	1944	31	288	129518	16712
256	6247	142	6105	31	256	387314	49976
256	6449	153	6296	31	384	399838	51592

Table 2. Sample parameter choices, $q = 256$, $f =$ trinary.

2. J. Hoffstein, J.H. Silverman, Optimizations for NTRU, Public Key Cryptography and Computational Number Theory (Warsaw, Sept. 11–15, 2000), Walter de Gruyter, Berlin–New York, 2001, 77–88.
3. J. Hoffstein, J.H. Silverman, Provable Probability Bounds for NTRUEncrypt Convolution Wrap/Gap Events Technical report, NTRU Cryptosystems, July 2007. Report #022, available at `http://www.ntru.com`.
4. N. Howgrave-Graham A hybrid lattice-reduction and meet-in-the-middle attack against NTRU To appear in Crypto '07
5. N. Howgrave-Graham, J. H. Silverman, W. Whyte Choosing Parameter Sets for NTRUEncrypt with NAEP and SVES-3, Topics in cryptology—CT-RSA 2005, 118–135, Lecture Notes in Comput. Sci., 3376, Springer, Berlin, 2005. `http://www.ntru.com/cryptolab/articles.htm#2005_1`
6. V. Shoup *NTL: A Library for doing Number Theory*, Version 5.4, `http://www.shoup.net/ntl`

k	N	N_1	K	d_f	Lattice bits	$2d_fN$	bits in key
80	383	141	242	31	80	23746	3447
80	439	153	286	29	120	25462	3951
112	521	151	370	36	112	37512	4689
112	521	163	358	38	168	39596	4689
128	569	155	414	40	128	45520	5121
128	569	168	401	42	192	47796	5121
160	733	162	571	44	160	64504	6597
160	787	177	610	44	240	69256	7083
192	977	168	809	46	192	89884	8793
192	1039	184	855	46	288	95588	9351
256	1913	179	1734	47	256	179822	17217
256	2039	198	1841	47	384	191666	18351

Table 3. Sample parameter choices, $q = 512$, $f = 1 + 3F$, minimal operations.

k	N	N_1	K	d_f	Lattice bits	$2d_fN$	bits in key
160	521	162	359	59	160	61478	4689
160	557	177	380	59	240	65726	5013

Table 4. Sample parameter choices, $q = 512$, $f =$ trinary.

k	N	N_1	K	d_f	Lattice bits	$2d_fN$	bits in key
80	257	153	104	77	80	39578	2570
80	271	164	107	77	120	41734	2710
80	439	153	286	29	80	25462	4390
80	439	164	286	31	120	27218	4390
112	313	162	151	82	112	51332	3130
112	331	174	157	82	168	54284	3310
112	467	162	305	42	112	39228	4670
112	467	174	293	45	168	42030	4670
128	347	166	181	82	128	56908	3470
128	367	179	188	82	192	60188	3670
128	499	166	333	47	128	46906	4990
128	499	179	320	50	192	49900	4990
160	421	173	248	83	160	69886	4210
160	443	187	256	83	240	73538	4430
160	617	173	444	52	160	64168	6170
160	617	187	430	55	240	67870	6170
192	521	179	342	81	192	84402	5210
192	557	194	363	81	288	90234	5570
192	683	179	504	61	192	83326	6830
192	683	194	489	61	288	88790	6830
256	743	189	554	85	256	126310	7430
256	787	207	580	85	384	133790	7870
256	751	189	562	84	256	126168	7510
256	797	207	590	84	384	133896	7970

Table 5. Sample parameter choices, $q = 1024$, $f = 1 + 3F$.

k	N	N_1	K	d_f	Lattice bits	$2d_fN$	bits in key
80	277	173	104	83	80	45982	3047
80	461	173	288	31	80	28582	5071
80	461	188	273	34	120	31348	5071
112	331	186	145	100	112	66200	3641
112	569	186	383	39	112	44382	6259
112	569	202	367	42	168	47796	6259
128	359	191	168	101	128	72518	3949
128	631	191	440	42	128	53004	6941
128	631	208	423	45	192	56790	6941
160	409	200	209	116	160	94888	4499
160	661	200	461	54	160	71388	7271
160	661	219	461	58	240	76676	7271
192	461	208	253	131	192	120782	5071
192	761	208	553	61	192	92842	8371
192	761	229	532	65	288	98930	8371
256	563	223	340	156	256	175656	6193
256	883	223	660	80	256	141280	9713
256	883	247	660	86	384	151876	9713

Table 6. Sample parameter choices, $q = 2048$, $f = 1 + 3F$.