NTRU: A Ring-Based Public Key Cryptosystem

Jeffrey Hoffstein, Jill Pipher, Joseph H. Silverman

Abstract. We describe NTRU, a new public key cryptosystem. NTRU features reasonably short, easily created keys, high speed, and low memory requirements. NTRU encryption and decryption use a mixing system suggested by polynomial algebra combined with a clustering principle based on elementary probability theory. The security of the NTRU cryptosystem comes from the interaction of the polynomial mixing system with the independence of reduction modulo two relatively prime integers p and q.

Contents

0. Introduction
1. Description of the NTRU algorithm
   1.1. Notation
   1.2. Key Creation
   1.3. Encryption
   1.4. Decryption
   1.5. Why Decryption Works
2. Parameter Selection
   2.1. Notation and a norm estimate
   2.2. Sample spaces
   2.3. A Decryption Criterion
3. Security Analysis
   3.1. Brute force attacks
   3.2. Meet-in-the-middle attacks
   3.3. Multiple transmission attacks
   3.4. Lattice based attacks
4. Practical Implementations of NTRU
   4.1. Specific Parameter Choices
   4.2. Lattice Attacks — Experimental Evidence
5. Additional Topics
   5.1. Improving Message Expansion
   5.2. Theoretical Operating Specifications
   5.3. Other Implementation Considerations
   5.4. Comparison With Other PKCS's
6. Appendix

§0. Introduction

There has been considerable interest in the creation of efficient and computationally inexpensive public key cryptosystems since Diffie and Hellman [3] explained how such systems could be created using one-way functions. Currently, the most widely used public key system is RSA, which was created by Rivest, Shamir and Adelman in 1978 [9] and is based on the difficulty of factoring large numbers. Other systems include the McEliece system [8] which relies on error correcting codes, and a recent system of Goldreich, Goldwasser, and Halevi [4] which is based on the difficulty of lattice reduction problems.

In this paper we describe a new public key cryptosystem, which we call the NTRU system. The encryption procedure uses a mixing system based on polynomial algebra and reduction modulo two numbers p and q, while the decryption procedure uses an unmixing system whose validity depends on elementary probability theory. The security of the NTRU public key cryptosystem comes from the interaction of the polynomial mixing system with the independence of reduction modulo p and q. Security also relies on the (experimentally observed) fact that for most lattices, it is very difficult to find extremely short (as opposed to moderately short) vectors.

We mention that the presentation in this paper differs from an earlier, widely circulated but unpublished, preprint [6] in that the analysis of lattice-based attacks has been expanded and clarified, based largely on the numerous comments received from Don Coppersmith, Johan Håstad, and Adi Shamir in person, via email, and in the recent article [2]. We would like to take this opportunity to thank them for their interest and their help.

NTRU fits into the general framework of a probabilistic cryptosystem as described in [1] and [5]. This means that encryption includes a random element, so each message has many possible encryptions. Encryption and decryption with NTRU are extremely fast, and key creation is fast and easy. See Section 5 for specifics, but we note here that NTRU takes $O(N^2)$ operations to encrypt or decrypt a message block of length N, making it considerably faster than the $O(N^3)$ operations required by RSA. Further, NTRU key lengths are $O(N)$, which compares well with the $O(N^2)$ key lengths required by other "fast" public keys systems such as [8, 4].

§1. Description of the NTRU algorithm

§1.1. Notation. An NTRU cryptosystem depends on three integer parameters (N, p, q) and four sets $\mathcal{L}_f$, $\mathcal{L}_g$, $\mathcal{L}_\phi$, $\mathcal{L}_m$ of polynomials of degree $N-1$ with integer coefficients. Note that p and q need not be prime, but we will assume that $\gcd(p, q) = 1$, and q will always be considerably larger than p. We work in the ring $R = \mathbb{Z}[X]/(X^N - 1)$. An element $F \in R$ will be written as a polynomial or a vector,

$$F = \sum_{i=0}^{N-1} F_i x^i = [F_0, F_1, \dots, F_{N-1}].$$

We write $\circledast$ to denote multiplication in R. This *star multiplication* is given explicitly as a cyclic convolution product,

$$F \circledast G = H \quad \text{with} \quad H_k = \sum_{i=0}^{k} F_i G_{k-i} + \sum_{i=k+1}^{N-1} F_i G_{N+k-i} = \sum_{i+j \equiv k \pmod{N}} F_i G_j.$$

When we do a multiplication modulo (say) q, we mean to reduce the coefficients modulo q.

Remark. In principle, computation of a product $F \circledast G$ requires N^2 multiplications. However, for a typical product used by NTRU, one of F or G has small coefficients, so the computation of $F \circledast G$ is very fast. On the other hand, if N is taken to be large, then it might be faster to use Fast Fourier Transforms to compute products $F \circledast G$ in $O(N \log N)$ operations.

§**1.2. Key Creation.** To create an NTRU key, Dan randomly chooses 2 polynomials $f, g \in \mathcal{L}_g$. The polynomial f must satisfy the additional requirement that it have inverses modulo q and modulo p. For suitable parameter choices, this will be true for most choices of f, and the actual computation of these inverses is easy using a modification of the Euclidean algorithm. We will denote these inverses by F_q and F_p, that is,

$$F_q \circledast f \equiv 1 \pmod{q} \qquad \text{and} \qquad F_p \circledast f \equiv 1 \pmod{p}. \tag{1}$$

Dan next computes the quantity

$$h \equiv F_q \circledast g \pmod{q}. \tag{2}$$

Dan's public key is the polynomial h. Dan's private key is the polynomial f, although in practice he will also want to store F_p.

§**1.3. Encryption.** Suppose that Cathy (the encrypter) wants to send a message to Dan (the decrypter). She begins by selecting a message m from the set of plaintexts $\mathcal{L}_m$. Next she randomly chooses a polynomial $\phi \in \mathcal{L}_\phi$ and uses Dan's public key h to compute

$$e \equiv p\phi \circledast h + m \pmod{q}.$$

This is the encrypted message which Cathy transmits to Dan.

§**1.4. Decryption.** Suppose that Dan has received the message e from Cathy and wants to decrypt it using his private key f. To do this efficiently, Dan should have precomputed the polynomial F_p described in Section 1.1.

In order to decrypt e, Dan first computes

$$a \equiv f \circledast e \pmod{q},$$

where he chooses the coefficients of a in the interval from $-q/2$ to $q/2$. Now treating a as a polynomial with integer coefficients, Dan recovers the message by computing

$$F_p \circledast a \pmod{p}.$$

Remark. For appropriate parameter values, there is an extremely high probability that the decryption procedure will recover the original message. However, some parameter choices may cause occasional decryption failure, so one should probably include a few check bits in each message block. The usual cause of decryption failure will be that the message is improperly centered. In this case Dan will be able to recover the message by choosing the coefficients of $a \equiv f \circledast e \pmod q$ in a slightly different interval, for example from $-q/2 + x$ to $q/2 + x$ for some small (positive or negative) value of x. If no value of x works, then we say that we have *gap failure* and the message cannot be decrypted as easily. For well-chosen parameter values, this will occur so rarely that it can be ignored in practice.

§**1.5. Why Decryption Works.** The polynomial a that Dan computes satisfies

$$\begin{aligned} a \equiv f \circledast e &\equiv f \circledast p\phi \circledast h + f \circledast m \pmod q \\ &= f \circledast p\phi \circledast F_q \circledast g + f \circledast m \pmod q \quad \text{from (2)}, \\ &= p\phi \circledast g + f \circledast m \pmod q \quad \text{from (1)}. \end{aligned}$$

Consider this last polynomial $p\phi \circledast g + f \circledast m$. For appropriate parameter choices, we can ensure that (almost always) all of its coefficients lie between $-q/2$ and $q/2$, so that it doesn't change if its coefficients are reduced modulo q. This means that when Dan reduces the coefficients of $f \circledast e$ modulo q into the interval from $-q/2$ to $q/2$, he recovers *exactly* the polynomial

$$a = p\phi \circledast g + f \circledast m \quad \text{in} \quad \mathbb{Z}[X]/(X^N - 1).$$

Reducing a modulo p then gives him the polynomial $f \circledast m \pmod p$, and multiplication by F_p retrieves the message $m \pmod p$.

§2. Parameter Selection

§**2.1. Notation and a norm estimate.** We define the *width* of an element $F \in R$ to be

$$|F|_\infty = \max_{1 \le i \le N}\{F_i\} - \min_{1 \le i \le N}\{F_i\}.$$

As our notation suggests, this is a sort of L^∞ norm on R. Similarly, we define a *centered* L^2 *norm* on R by

$$|F|_2 = \left(\sum_{i=1}^N (F_i - \bar{F})^2\right)^{1/2}, \qquad \text{where} \quad \bar{F} = \frac{1}{N}\sum_{i=1}^N F_i.$$

(Equivalently, $|F|_2 / \sqrt{N}$ is the standard deviation of the coefficients of F.) The following proposition was suggested to us by Don Coppersmith.

Proposition. *For any $\varepsilon > 0$ there are constants $\gamma_1, \gamma_2 > 0$, depending on ε and N, such that for randomly chosen polynomials $F, G \in R$, the probability is greater than $1 - \varepsilon$ that they satisfy*

$$\gamma_1 \left|F\right|_2 \left|G\right|_2 \leq \left|F \circledast G\right|_\infty \leq \gamma_2 \left|F\right|_2 \left|G\right|_2 .$$

Of course, this proposition would be useless from a practical veiwpoint if the ratio γ_2/γ_1 were very large for small ε's. However, it turns out that even for moderately large values of N and very small values of ε, the constants γ_1, γ_2 are not at all extreme. We have verified this experimentally for a large number of parameter values and have an outline of a theoretical proof.

§**2.2. Sample spaces.** The space of messages $\mathcal{L}_m$ consists of all polynomials modulo p. Assuming p is odd, it is most convenient to take

$$\mathcal{L}_m = \left\{ m \in R : m \text{ has coefficients lying between } -\frac{1}{2}(p-1) \text{ and } \frac{1}{2}(p-1) \right\}.$$

To describe the other samples spaces, we will use sets of the form

$$\mathcal{L}(d_1, d_2) = \left\{ F \in R : \begin{array}{c} F \text{ has } d_1 \text{ coefficients equal } 1, \\ d_2 \text{ coefficients equal } -1, \text{ the rest } 0 \end{array} \right\}.$$

With this notation, we choose three positive integers d_f, d_g, d and set

$$\mathcal{L}_f = \mathcal{L}(d_f, d_f - 1), \quad \mathcal{L}_g = \mathcal{L}(d_g, d_g), \quad \text{and} \quad \mathcal{L}_\phi = \mathcal{L}(d, d).$$

(The reason we don't set $\mathcal{L}_f = \mathcal{L}(d_f, d_f)$ is because we want f to be invertible, and a polynomial satisfying $f(1) = 0$ can never be invertible.) Notice that $f \in \mathcal{L}_f$, $g \in \mathcal{L}_g$, and $\phi \in \mathcal{L}_\phi$ have L^2 norms

$$\left|f\right|_2 = \sqrt{2d_f - 1 - N^{-1}}, \qquad \left|g\right|_2 = \sqrt{2d_g}, \qquad \left|\phi\right|_2 = \sqrt{2d}.$$

Later we will give values for d_f, d_g, d which allow decryption while maintaining various security levels.

§**2.3. A Decryption Criterion.** In order for the decryption process to work, it is necessary that

$$\left|f \circledast m + p\phi \circledast g\right|_\infty < q.$$

We have found that this will virtually always be true if we choose parameters so that

$$\left|f \circledast m\right|_\infty \leq q/4 \qquad \text{and} \qquad \left|p\phi \circledast g\right|_\infty \leq q/4,$$

and in view of the above Proposition, this suggests that we take

$$\left|f\right|_2 \left|m\right|_2 \approx q/4\gamma_2 \qquad \text{and} \qquad \left|\phi\right|_2 \left|g\right|_2 \approx q/4p\gamma_2 \tag{3}$$

for a γ_2 corresponding to a small value for ε. For example, experimental evidence suggests that for $N = 107$, $N = 167$, and $N = 503$, appropriate values for γ_2 are 0.35, 0.27, and 0.17 respectively.

§3. Security Analysis

§3.1. Brute force attacks. An attacker can recover the private key by trying all possible $f \in \mathcal{L}_f$ and testing if $f \circledast h \pmod q$ has small entries, or by trying all $g \in \mathcal{L}_g$ and testing if $g \circledast h^{-1} \pmod q$ has small entries. Similarly, an attacker can recover a message by trying all possible $\phi \in \mathcal{L}_\phi$ and testing if $e - \phi \circledast h \pmod q$ has small entries. In practice, $\mathcal{L}_g$ will be smaller than $\mathcal{L}_f$, so key security is determined by $\#\mathcal{L}_g$, and individual message security is determined by $\#\mathcal{L}_\phi$. However, as described in the next section, there is a meet-in-the-middle attack which (assuming sufficient storage) cuts the search time by the usual square root. Hence the security level is given by

$$\begin{pmatrix}\text{Key}\\ \text{Security}\end{pmatrix} = \sqrt{\#\mathcal{L}_g} = \frac{1}{d_g!}\sqrt{\frac{N!}{(N-2d_g)!}}$$

$$\begin{pmatrix}\text{Message}\\ \text{Security}\end{pmatrix} = \sqrt{\#\mathcal{L}_\phi} = \frac{1}{d!}\sqrt{\frac{N!}{(N-2d)!}}.$$

§3.2. Meet-in-the-middle attacks. Recall that an encrypted message looks like $e \equiv \phi \circledast h + m \pmod q$. Andrew Odlyzko has pointed out that there is a meet-in-the-middle attack which can be used against ϕ, and we observe that a similar attack applies also to the private key f. Briefly, one splits f in half, say $f = f_1 + f_2$, and then one matches $f_1 \circledast e$ against $-f_2 \circledast e$, looking for (f_1, f_2) so that the corresponding coefficients have approximately the same value. Hence in order to obtain a security level of (say) 2^{80}, one must choose f, g, and ϕ from sets containing around 2^{160} elements. (For further details, see [13].)

§3.3. Multiple transmission attacks. If Cathy sends a single message m several times using the same public key but different random ϕ's, then the attacker Betty will be able to recover a large part of the message. Briefly, suppose that Cathy transmits $e_i \equiv \phi_i \circledast h + m \pmod q$ for $i = 1, 2, \ldots, r$. Betty can then compute $(e_i - e_1) \circledast h^{-1} \pmod q$, thereby recovering $\phi_i - \phi_1 \pmod q$. However, the coefficients of the ϕ's are so small that she recovers exactly $\phi_i - \phi_1$, and from this she will recover many of the coefficients of ϕ_1. If r is even of moderate size (say 4 or 5), Betty will recover enough of ϕ_1 to be able to test all possibilities for the remaining coefficients by brute force, thereby recovering m. Thus multiple transmission are not advised without some further scrambling of the underlying message. We do point out that even if Betty decrypts a single message in this fashion, this information will not assist her in decrypting any subsequent messages.

§3.4. Lattice based attacks. The object of this section is to give a brief analysis of the known lattice attacks on both the public key h and the message m. We begin with a few words concerning lattice reduction. The goal of lattice reduction is to find one or more "small" vectors in a given lattice. In theory, the smallest vector can be found by an exhaustive search, but in practice this is not possible if the dimension is large. The LLL algorithm of Lenstra-Lenstra-Lovász [7], with various improvements due to Schnorr and others, [10, 12, 11] will find relatively small vectors in polynomial time, but even LLL will take a long time to find the smallest vector provided that the smallest vector is not too much smaller than the expected length of the smallest vector. We will make these observations more precise below.

§*3.4.1. Lattice attack on an NTRU private key.* Consider the $2N$-by-$2N$ matrix composed of four N-by-N blocks:

$$\left(\begin{array}{cccc|cccc} \alpha & 0 & \cdots & 0 & h_0 & h_1 & \cdots & h_{N-1} \\ 0 & \alpha & \cdots & 0 & h_{N-1} & h_0 & \cdots & h_{N-2} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \alpha & h_1 & h_2 & \cdots & h_0 \\ \hline 0 & 0 & \cdots & 0 & q & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 & 0 & q & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & q \end{array}\right)$$

(Here α is a parameter to be chosen shortly.) Let L be the lattice generated by the rows of this matrix. The determinant of L is $q^N \alpha^N$.

Since the public key is $h = g \circledast f^{-1}$, the lattice L will contain the vector $\tau = (\alpha f, g)$, by which we mean the $2N$ vector consisting of the N coefficients of f multiplied by α, followed by the N coefficients of g. By the gaussian heuristic, the expected size of the smallest vector in a random lattice of dimension n and determinant D lies between

$$D^{1/n}\sqrt{\frac{n}{2\pi e}} \quad \text{and} \quad D^{1/n}\sqrt{\frac{n}{\pi e}}.$$

In our case, $n = 2N$ and $D = q^N \alpha^N$, so the expected smallest length is larger (but not much larger) than

$$s = \sqrt{\frac{N\alpha q}{\pi e}}.$$

An implementation of a lattice reduction algorithm will have the best chance of locating τ, or another vector whose length is close to τ, if the attacker chooses α to maximize the ratio $s/|\tau|_2$. Squaring this ratio, we see that an attacker should choose α so as to maximize

$$\frac{\alpha}{\alpha^2 |f|_2^2 + |g|_2^2} = \left(\alpha |f|_2^2 + \alpha^{-1} |g|_2^2\right)^{-1}.$$

This is done by choosing $\alpha = |g|_2 / |f|_2$. (Note that $|g|_2$ and $|f|_2$ are both public quantities.)

When α is chosen in this way, we define a constant c_h by setting $|\tau|_2 = c_h s$. Thus c_h is the ratio of the length of the target vector to the length of the expected shortest vector. The smaller the value of c_h, the easier it will be to find the target vector. Substituting in above, we obtain

$$c_h = \sqrt{\frac{2\pi e \,|f|_2\,|g|_2}{Nq}}.$$

For a given pair (f, g) used to set up the cryptosystem, c_h may be viewed as a measure of how far the associated lattice departs from a random lattice. If c_h is close to 1, then L will resemble a random lattice and lattice reduction methods will have a hard time finding a short vector in general, and finding τ in particular. As c_h decreases, lattice reduction algorithms will have an easier time finding τ. Based on the limited evidence we have obtained, the time required appears to be (at least) exponential in N, with a constant in the exponent proportional to c_h.

§*3.4.2. Lattice attack on an NTRU message.* A lattice attack may also be directed against an individual message m. Here the associated lattice problem is very similar to that for h, and the target vector will have the form $(\alpha m, \phi)$. As before, the attacker should balance the lattice using $\alpha = |\phi|_2 \,/\, |m|_2$, which leads to the value

$$c_m = \sqrt{\frac{2\pi e \,|m|_2\,|\phi|_2}{Nq}}.$$

This constant c_m gives a measure of the vulnerability of an individual message to a lattice attack, similar to the way c_h does for a lattice attack on h. An encrypted message is most vulnerable if c_m is small, and becomes less so as c_m gets closer to 1.

In order to make the attacks on h and m equally difficult, we want to take $c_m \approx c_h$, or equivalently, $|f|_2\,|g|_2 \approx |m|_2\,|\phi|_2$. For concreteness, we will now restrict to the case that $p = 3$; other values may be analyzed similarly. For $p = 3$, an average message m will consist of $N/3$ each of 1, 0 and -1, so $|m|_2 \approx \sqrt{2N/3}$. Similarly, ϕ consists of d each of 1 and -1, with the rest 0's, so $|\phi|_2 = \sqrt{2d}$. Thus we will want to set

$$|f|_2\,|g|_2 \approx \sqrt{4Nd/3}.$$

This can be combined with the decryption criterion (3) to assist in choosing parameters.

§*3.4.3. Lattice attack on a spurious key.* Rather than trying to find the private key f, an attacker might use the lattice described above (in Section 3.4.1) and try to find some other short vector in the lattice, say of the form $\tau' = (\alpha f', g')$. If this vector is short enough, then f' will act as a decryption key. More precisely, if it turns out that with high probability,

$$f' \circledast e \equiv p\phi \circledast g' + m \circledast f' \pmod{q}$$

satisfies $|p\phi \circledast g' + m \circledast f'|_\infty < q$, then decryption will succeed; and even if this width is $2q$ or $3q$, it is possible that the message could be recovered via error-correcting techniques, especially if several such τ''s could be found. This idea, which is due to Coppersmith and Shamir, is described in [2]. However experimental evidence suggests that the existence of spurious keys does not pose a security threat. See Section 4.2 for a further discussion of this point.

§4. Practical Implementations of NTRU

§4.1. Specific Parameter Choices. We will now present three distinct sets of parameters which yield three different levels of security. The norms of f and g have been chosen so that decryption failure occurs with probability less than $5 \cdot 10^{-5}$(based on extensive computer experimentation).

Case A: Moderate Security

The Moderate Security parameters are suitable for situations in which the intrinsic value of any individual message is small, and in which keys will be changed with reasonable frequency. Examples might include encrypting of television, pager, and cellular telephone transmissions.

$$(N, p, q) = (107, 3, 64)$$
$$\mathcal{L}_f = \mathcal{L}(15, 14), \quad \mathcal{L}_g = \mathcal{L}(12, 12), \quad \mathcal{L}_\phi = \mathcal{L}(5, 5) \quad \text{(i.e., } d = 5\text{)}.$$

(In other words, f is chosen with 15 1's and 14 -1's, g is chosen with 12 1's and 12 -1's, and ϕ is chosen with 5 1's and 5 -1's.) These give key sizes

$$\text{Private Key} = 340 \text{ bits} \quad \text{and} \quad \text{Public Key} = 642 \text{ bits},$$

and (meet-in-the-middle) security levels

$$\text{Key Security} = 2^{50} \quad \text{and} \quad \text{Message Security} = 2^{26.5}.$$

(We note again that meet-in-the-middle attacks require large amounts of computer storage; for straight search brute force attacks, these security levels should be squared.) Substituting the above values into the appropriate formulas yields lattice values

$$c_h = 0.257, \quad c_m = 0.258, \quad \text{and} \quad s = 0.422q.$$

Case B: High Security

$$(N, p, q) = (167, 3, 128)$$
$$\mathcal{L}_f = \mathcal{L}(61, 60), \quad \mathcal{L}_g = \mathcal{L}(20, 20), \quad \mathcal{L}_\phi = \mathcal{L}(18, 18) \quad \text{(i.e., } d = 18\text{)}$$
$$\text{Private Key} = 530 \text{ bits} \quad \text{and} \quad \text{Public Key} = 1169 \text{ bits}$$
$$\text{Key Security} = 2^{82.9} \quad \text{and} \quad \text{Message Security} = 2^{77.5}$$
$$c_h = 0.236, \quad c_m = 0.225, \quad \text{and} \quad s = 0.296q.$$

Case C: Highest Security

$$(N, p, q) = (503, 3, 256)$$

$$\mathcal{L}_f = \mathcal{L}(216, 215), \quad \mathcal{L}_g = \mathcal{L}(72, 72), \quad \mathcal{L}_\phi = \mathcal{L}(55, 55) \quad \text{(i.e., } d = 55\text{)}$$

$$\text{Private Key} = 1595 \text{ bits} \quad \text{and} \quad \text{Public Key} = 4024 \text{ bits}$$

$$\text{Key Security} = 2^{285} \quad \text{and} \quad \text{Message Security} = 2^{170}$$

$$c_h = 0.182, \quad c_m = 0.160, \quad \text{and} \quad s = 0.0.365q.$$

§**4.2. Lattice Attacks — Experimental Evidence.** In this section we describe our preliminary analysis of the security of the NTRU Public Key Cryptosystem from attacks using lattice reduction methods. It is based on experiments which were performed using version 1.7 of Victor Shoup's implementation of the Schnorr,Euchner and Hoerner improvements of the LLL algorithm, distributed in his NTL package at `http://www.cs.wisc.edu/~shoup/ntl/`. The NTL package was run on a 200 M Hz Pentium Pro with a Linux operating system.

This algorithm has several parameters that can be adjusted to give varying types of results. In general the LLL algorithm can be tuned to either find a somewhat short point in a small amount of time or a very short point in a longer time. The key quantity is the constant c_h (or c_m) described above. It is somewhat easier to decrypt messages if these constants are small, somewhat harder if they are close to 1. The idea is to choose a compromise value which makes decryption easy, while still making it difficult for LLL to work effectively.

The following tables give the time required for LLL to find either the target $(\alpha f, g)$ or a closely related vector in the lattice L of 3.4.1 for various choices of q, c_h and dimension N. As will be elaborated on further in the Appendix, the algorithm seems to find either a vector of the correct length, or one considerably too long to be useful for decryption. Even if it were to find a spurious key of length somewhat longer than the target, as discussed by Coppersmith and Shamir in [2], it appears that the time required to find such a key would not be significantly less than that required to find the true target.

We have chosen parameters so that $c_m \approx c_h$. (So the time required to break an individual message should be on the same order as the time required to break the public key). In all cases we found that when N gets sufficiently large the algorithm fails to terminate, probably because of accumulated round off errors. The tables end roughly at this point.

In this version of LLL there are three parameters that can be fine tuned to optimize an attack. The tables give typical running times to break a key pair for the most optimal choices of parameters we have found to date. The two columns give results for two different floating point versions of the program, QP1 offering higher precision. We then use this information to extrapolate running times for larger values of N, assuming the algorithm were to terminate.

	FP		QP1	
	N	time (secs)	N	time (secs)
	75	561	75	1604
Case A	80	1493	80	3406
q=64	85	2832	85	5168
c=0.26	90	4435	88	11298
	92	7440	90	16102
	94	12908	95	62321
	96	28534	96	80045
	98	129938	98	374034
			100	183307
	N	time (secs)	N	time (secs)
	75	600	75	3026
Case B	80	953	80	5452
q=128	85	1127	85	8171
c=0.23	90	3816	90	20195
	95	13588	95	57087
			100	109706
	N	time (secs)	N	time (secs)
	75	547	75	2293
Case C	80	765	78	3513
q=256	85	1651	81	3453
c=0.18	90	2414	84	5061
	95	2934	87	6685
	100	7471	90	9753
	102	8648	93	16946
			96	19854
			99	30014
			102	51207
			105	75860
			108	145834

We will write $t(N)$ for the time in seconds necessary to break a public key corresponding to a parameter N. When we graph $\log t(N)$ against N, the examples we have done seem to indicate that the graph has a positive slope with a small positive concavity. This would indicate that $t(N)$ grows at least exponentially with N, and possibly even with $N \log N$. To extrapolate out to higher values of N, we have taken the information we have and approximated a lower bound for the slope of $\log t(N)$ against N. This gives the following rough estimates for

$t(N)$ in seconds using FP:

$$\begin{aligned} t(N) &> 12908 \exp[(0.396)(N-94)] && \text{(Moderate Security)} \\ t(N) &> 13588 \exp[(0.291)(N-95)] && \text{(High Security)} \\ t(N) &> 2414 \exp[(0.10)(N-92)] && \text{(Highest Security)} \end{aligned}$$

The running times for QP1 are longer for small N, but yield a better exponential constant, so for QP1 we obtain:

$$\begin{aligned} t(N) &> 80045 \exp[(0.207)(N-96)] && \text{(Moderate Security)} \\ t(N) &> 8171 \exp[(0.17315)(N-85)] && \text{(High Security)} \\ t(N) &> 30014 \exp[(0.17564)(N-99)] && \text{(Highest Security)} \end{aligned}$$

These lower bounds yield the following estimates for the time necessary to break the different levels of NTRU security using QP1 running on one 200 MHz Pentium Pro:

Type	Level	q	c	N	Time (seconds)
$QP1$	Moderate	64	0.26	107	$780,230$ (9 days)
$QP1$	High	128	0.23	167	$1.198 \cdot 10^{10}$ (380 years)
$QP1$	Highest	256	0.18	503	$1.969 \cdot 10^{35}$ ($6.2 \cdot 10^{27}$ years)

A more detailed analysis and description of the lattice experiments is given in the Appendix.

§5. Additional Topics

§**5.1. Improving Message Expansion.** The NTRU PKCS's for the sample parameters presented in Section 4.1 have moderate message expansions. However, as the principal use for PKCS's is the exchange of a private key in a single message block this is not a significant problem. It may be worth mentioning, though, that there is a simple way that the NTRU technique can be used to convey a very long message, with an expansion of only 1-1 after the first mesage block.

With this approach, the first encrypted message e_1 that Cathy sends is decrypted as a sequence of 1's, 0's and -1's (taking $p = 3$) and interpreted as a ϕ_1 for the next message block. The next encrypted message block is $\phi_1 \circledast e_1 + m_1$, where m_1 is the first block of the actual message. As Dan knows ϕ_1, he can recover $m_1 \bmod q$ exactly. The next encrypted message block Cathy sends is $e_2 = \phi_2 \circledast e_1 + m_2$, where Cathy derived ϕ_2 from m_1 by squaring m_1 and reducing it mod 3. Dan can now recover ϕ_2 as he knows m_1, and hence can derive m_2 mod q from e_2. This can continue for a message of arbitrary length.

§**5.2. Theoretical Operating Specifications.** In this section we consider the theoretical operating characteristics of the NTRU PKCS. There are three integer parameters (N, p, q), four sets $\mathcal{L}_f, \mathcal{L}_g, \mathcal{L}_\phi, \mathcal{L}_m$ determined respectively by integers d_f, d_g, d, p as described in Sections 1.1 and 2.2. The following table summarizes the NTRU PKCS operating characteristics in terms of these parameters.

Plain Text Block	$N \log_2 p$ bits
Encrypted Text Block	$N \log_2 q$ bits
Encryption Speed*	$O(N^2)$ operations
Decryption Speed	$O(N^2)$ operations
Message Expansion	$\log_p q$-to-1
Private Key Length	$2N \log_2 p$ bits
Public Key Length	$N \log_2 q$ bits

* Precisely, $4N^2$ additions and N divisions by q with remainder

§5.3. Other Implementation Considerations. We briefly mention some additional factors which should be considered when implementing NTRU.

(1) It is important that $\gcd(q, p) = 1$. Although in principle NTRU will work without this requirement, in practice having $\gcd(q, p) > 1$ will decrease security. At the extreme range, if $p|q$, then the encrypted message e satisfies $e \equiv m \pmod{p}$, so it is completely insecure.

(2) We want most f's to have inverses modulo p and modulo q, since otherwise it will be hard to create keys. A first necessary requirement is that $\gcd(f(1), pq) = 1$, but if this fails for some chosen f, the code creator can instead use, say, $f(X) + 1$ or $f(X) - 1$. Assuming $\gcd(f(1), pq) = 1$, virtually all f's will have the required inverses if we take N to be a prime and require that for each prime P dividing p and q, the order of P in $(\mathbb{Z}/N\mathbb{Z})^*$ is large, say either $N - 1$ or $(N - 1)/2$. For example, this will certainly be true if $(N - 1)/2$ is itself prime (i.e., N is a Sophie Germain prime). Examples of such primes include 107, 167 and 503.

§5.4. Comparison With Other PKCS's. There are currently a number of public key cryptosystems in the literature, including the system of Rivest, Shamir, and Adelman (RSA [9]) based on the difficulty of factoring, the system of McEliece [8] based on error correcting codes, and the recent system of Goldreich, Goldwasser, and Halevi (GGH [4]) based on the difficulty of finding short almost-orthogonalized bases in a lattice.

The NTRU system has some features in common with McEliece's system, in that $\circledast$-multiplication in the ring R can be formulated as multiplication of matrices (of a special kind), and then encryption in both systems can be written as a matrix multiplication $E = AX + Y$, where A is the public key. A minor difference between the two systems is that for an NTRU encryption, Y is the message and X is a random vector, while the McEliece system reverses these assignments. But the real difference is the underlying trap-door which allows decryption. For the McEliece system, the matrix A is associated to an error correcting (Goppa) code, and decryption works because the random contribution is small enough to be "corrected" by the Goppa code. For NTRU, the matrix A

is a circulant matrix, and decryption depends on the decomposition of A into a product of two matrices having a special form, together with a lifting from mod q to mod p.

As far as we can tell, the NTRU system has little in common with the RSA system. Similarly, although the NTRU system must be set up to prevent lattice reduction attacks, its underlying decryption method is very different from the GGH system, in which decryption is based on knowledge of short lattice bases. In this aspect, GGH actually resembles the McEliece system, since in both cases decryption is performed by recognizing and eliminating a small random contribution. Contrasting this, NTRU eliminates a much larger random contribution via divisibility (i.e., congruence) considerations.

The following table compares some of the theoretical operating characteristics of the RSA, McEliece, GGH, and NTRU cryptosystems. In each case the number N represents a natural security/message length parameter.

	NTRU	RSA	McEliece	GGH
Encryption Speed$^{(1,2)}$	N^2	N^2	N^2	N^2
Decryption Speed$^{(3)}$	N^2	N^3	N^2	N^2
Public Key	N	N	N^2	N^2
Private Key	N	N	N^2	N^2
Message Expansion$^{(4)}$	varies	1–1	2–1	1–1

$^{(1)}$ NTRU encryption requires only additions and shifts, no other multiplications
$^{(2)}$ RSA encryption is $O(N^3)$ unless small encryption exponents are used.
$^{(3)}$ Asymptotically, NTRU encryption and decryption are $O(N \log N)$ using FFT.
$^{(4)}$ For NTRU, see Section 5.1.

We have made some preliminary timing comparisons between NTRU and RSA, using information available from RSA's web page. The NTRU program we used was written in C and not optimized for speed.

The main uses to which PKCS's are applied are the exchange of secret keys and short messages. Also, RSA, ECC and NTRU all work in units of "message blocks," and any message block in any of these systems is large enough to hold a secret key of very high security, or a short message. Thus for comparison purposes, in the following we interpreted a key encryption or decryption in a PKCS to be the process of encrypting or decrypting one message block. Numbers given for encryption and decryption are message blocks processed per second.

The information is summarized in the following tables:

Security Level	Encrypt (blks/sec)	Decrypt (blks/sec)	Create key (sec)
Moderate	1818	505	0.1080
High	649	164	0.1555
Highest	103	19	0.8571

NTRU: 75 MHz Pentium, running MSDOS

Security Level	Encrypt (blks/sec)	Decrypt (blks/sec)	Create key (sec)
Moderate	16666	2273	0.0079
High	4762	724	0.0184
Highest	730	79	0.1528

NTRU: 200 MHz Pentium Pro, running Linux

Security Level	Encrypt (blks/sec)	Decrypt (blks/sec)	Create key (sec)
512 bit	370	42	0.45
768 bit	189	15	1.5
1024 bit	116	7	3.8

RSA: 90MHz Pentium

Security Level	Encrypt (blks/sec)	Decrypt (blks/sec)	Create key (sec)
512 bit	1020	125	0.26
768 bit	588	42	0.59
1024 bit	385	23	1.28

RSA: 255 MHz Digital AlphaStation

Comparing NTRU and RSA on the Pentium 75 and 90 platforms, adjusting for clock speed, and comparing the moderate NTRU security level to 512 bit RSA security level, we find that NTRU is 5.9 times faster at encryption, 14.4 times faster at decryption and 5.0 times faster at key creation. Similarly comparing the highest NTRU security level to the 1024 bit RSA security level, NTRU is the same speed at encryption, 3.2 times faster at decryption, and 5.3 times faster at key creation.

The 200 MHz Pentium pro and the 256 MHz Digital Alpha are sufficiently different that there is no obvious way to precisely compare one to the other. But simply comparing the raw numbers it is interesting to note that in spite of the slower clock speed, NTRU comes out 16, 18 and 33 times faster at encryption, decryption and key creation at moderate security, and 2, 3 and 8 times faster at high security.

For related timings of ECC, we refer to Certicom's published report: "Certicom Releases Security Builder 1.2 Performance Data" According to their report (available at http://www.certicom.com/secureb.htm), on a Pentium platform ECC takes 4.57 times as long as RSA to encrypt a message block, and 0.267 times as long to decrypt a message block. Thus compared to RSA, ECC wins by a factor of about 4 when decrypting, but loses by a factor of 4 when encrypting.

Acknowledgments. We would like to thank Don Coppersmith, Johan Håstad, Hendrik Lenstra Jr., Bjorn Poonen, Adi Shamir, Claus Schnorr and Benne de

Weger for their help with lattice reduction methods, Philip Hirschhorn for his assistance in implementing NTRU and doing LLL testing, Victor Shoup for his NTL package, Martin Mohlenkamp for several enlightening conversations about this package, Andrew Odlyzko for pointing out the meet-in-the-middle attack and other helpful suggestions, Mike Rosen for his help with polynomial inverses, and Dan Lieman for his assistance in all phases of this project. In particular, our analysis of lattice-based attacks is an amalgamation of the suggestions of Don Coppersmith, Johan Håstad, and Adi Shamir, combined with some thoughts of our own, although we stress that any oversights or errors in this analysis are entirely of our own devising.

References

1. M. Blum, S. Goldwasser, *An efficient probabilistic public-key encryption scheme which hides all partial information*, Advances in Cryptology: Proceedings of CRYPTO 84, Lecture Notes in Computer Science, vol. 196, Springer-Verlag, 1985, pp. 289–299.
2. D. Coppersmith, A. Shamir, *Lattice attacks on NTRU*, Preprint, April 5, 1997; presented at Eurocrypt 97.
3. W. Diffie, M.E. Hellman, *New directions in cryptography*, IEEE Trans. on Information Theory **22** (1976), 644–654.
4. O. Goldreich, S. Goldwasser, S. Halevi, *Public-key cryptosystems from lattice reduction problems*, MIT – Laboratory for Computer Science preprint, November 1996.
5. S. Goldwasser and A. Micali, *Probabilistic encryption*, J. Computer and Systems Science **28** (1984), 270–299.
6. J. Hoffstein, J. Pipher, J.H. Silverman, *NTRU: A new high speed public key cryptosystem*, Preprint; presented at the rump session of Crypto 96.
7. A.K. Lenstra, H.W. Lenstra, L. Lovśz, *Factoring polynomials with polynomial coefficients*, Math. Annalen **261** (1982), 515–534.
8. R.J. McEliece, *A public-key cryptosystem based on algebraic coding theory*, JPL Pasadena, DSN Progress Reports **42–44** (1978), 114–116.
9. R.L. Rivest, A. Shamir, L. Adleman, *A method for obtaining digital signatures and public key cryptosystems*, Communications of the ACM **21** (1978), 120–126.
10. C.P. Schnorr, *Block reduced lattice bases and successive minima*, Combinatorics, Probability and Computing **3** (1994), 507–522.
11. C.P. Schnorr, M. Euchner, *Lattice basis reduction: improved practical algorithms and solving subset sum problems*, Mathematical Programing **66** (1994), 181-199.
12. C.P. Schnorr, H.H. Hoerner, *Attacking the Chor Rivest cryptosystem by improved lattice reduction*, Proc. EUROCRYPT 1995, Lecture Notes in Computer Science 921, Springer-Verlag, 1995, pp. 1–12.
13. J.H. Silverman, *A Meet-In-The-Middle Attack on an NTRU Private Key*, preprint.

§6. Appendix - Some remarks on the impementation of the Schnorr-Euchner improvements of LLL

The LLL algorithm produces, from a given basis for a lattice, a reduced basis whose first vector is guaranteed to be relatively short. Part of this procedure involves minimizing the length of linear combinations of basis vectors, taking "blocks" of two at a time. If one minimized the length of linear combinations of basis vectors, taking as a block the entire basis, then an actual shortest vector could be found, but the time to produce it would be exponential in the dimension. One of Schnorr and Euchner's improvements (see [10, 11, 12] was to add an extra degree of flexibility. They minimize over blocks of vectors of size greater than two, but less than the dimension. This results in shorter vectors than are generally found by the original LLL algorithm, i.e with block size equal 2, but causes an increase in running time which is exponential in the block size.

In NTL 1.7 the blocksize β can be chosen, as well as a second parameter p which Schnorr and Hoerner introduced. This is intended to moderate the increase in running time as β increases. The "pruning" parameter p halts the minimization process when the probability of finding a shorter vector than already found within a given block falls below a prescribed value which depends on p. This probability is computed via the gaussian volume heuristic, the validity of which depends on the randomness of the lattice.

There is a third parameter δ which is allowed to vary between 0.5 and 1.0. This parameter determines how frequently a certain recursive operation is performed. The program recommends setting $\delta = .99$, and we have followed this recommendation.

In our experiments we varied the choice of c_h and of the blocksize β and pruning factor p. We never observed, even for larger values of β, a noticeable improvement from the pruning procedure and finally set $p = 0$, so the pruning procedure was not called.

The following tables give a more complete set of information which includes the choice of β and the ratio of the smallest vector found to the target vector. We observed that for small values of β the algorithm would fail to find a vector useful for decryption. In fact it would most likely produce a q-vector, that is to say a vector with a single coordinate equal to q and the rest all zero. The initial basis for L contains N of these vectors, which are in fact not much longer than the length $s = \sqrt{N\alpha q/\pi e}$ of the shortest expected vector. As β increased, the smallest vector found would continue to be a q-vector until a certain threshold was passed, which depended on N and c_h. (Increasing with N, decreasing with c_h). After this threshold, if the algorithm terminated it would usually succeed in finding the target vector. On some occasions it would find a vector slightly smaller than a q-vector and then at the next blocksize succeed in finding the target. The general pattern is that for fixed c_h the blocksize would have to increase with N in order for the algorithm to succeed in finding the target. At slightly smaller blocksizes the time required would be on the same order as the time required to find the target but the vector found — either the q-vector or slightly smaller — would be useless for decryption purposes.

In Table 1 timings are given for a lattice corresponding to $c_h = 0.26$ with $|f|_2 = |g|_2$. This is the equivalent to the moderate security lattice attack, but the balancing of f and g makes it possible to work with smaller integers and the NTL program runs, with some exceptions, more efficiently. Notice that the necessary blocksize increases monotonically with N. In the Tables 2, 3 and 4, timings are given for moderate, high and highest security. These are again formed with $|f|_2 = |g|_2$, and the moderate security table is a repeat to give some idea of the variation that occurs. Finally, Table 5 is formed with $|f|_2$ and $|g|_2$ taking the same ratio as in the actual encryption procedure. The $\alpha = 0.9097$ indicates that the lattice has been balanced to optimize the chances of an attacker. Note that the times are roughly the same as the equivalent situation in Tables 1 and 2, but timing deteriorates very substantially at $N = 98$. Notice some curiously short timings at $N = 90$ in Tables 2 and 5. These occurred when the algorithm terminated after locating a particular short vector: $(f', f' \circledast h)$, with $f' = (1, -1, 1, -1, 1, \ldots)$. The value of $f' \circledast h$ is then $(k, -k, k, \ldots)$, for some k, with k taking the value 1 or -1 with probability $2/q$. If this happens, $(f', f' \circledast h)$ is short, but as f' is highly non-invertible it is useless for decryption purposes.

N	Block size	Running time (sec)	Actual Total Norm	Smallest Norm Found	Ratio of found to actual
75	6	1910	6.32	6.32	1.0
80	4	1823	6.48	64.00	9.9
80	6	2731	6.78	64.00	9.4
80	8	3285	6.48	64.00	9.9
80	10	3663	6.63	6.63	1.0
85	4	2091	6.93	64.00	9.2
85	6	3661	6.78	64.00	9.4
85	8	5012	6.93	64.00	9.2
85	10	5497	6.78	64.00	9.4
85	12	7438	6.93	64.00	9.2
85	14	7433	7.07	7.07	1.0
90	4	3382	6.93	64.00	9.2
90	6	3305	6.78	64.00	9.4
90	8	5910	6.78	64.00	9.4
90	10	7173	6.78	64.00	9.4
90	12	7367	6.78	64.00	9.4
90	14	12182	6.93	64.00	9.2
90	16	16102	6.78	6.78	1.0
90	18	18920	6.93	6.93	1.0
95	4	3019	7.21	64.00	8.9
95	6	4434	7.07	64.00	9.1
95	8	7707	7.07	64.00	9.1
95	10	9449	7.35	64.00	8.7
95	12	11308	7.21	64.00	8.9
95	14	14520	7.21	64.00	8.9
95	16	22348	7.07	64.00	9.1
95	18	23965	7.21	64.00	8.9
95	20	81028	7.07	64.00	9.1
95	22	62321	7.35	7.35	1.0
100	4	4020	7.21	64.00	8.9
100	6	6307	7.07	64.00	9.1
100	8	9225	7.07	64.00	9.1
100	10	11109	7.07	64.00	9.1
100	12	13381	7.07	64.00	9.1
100	14	19096	7.21	64.00	8.9
100	16	23850	7.07	64.00	9.1
100	18	40670	7.21	50.99	7.1
100	20	72130	7.21	64.00	8.9
100	22	444773	7.21	7.21	1.0

Table 1: BKZ-QP1 with $Q = 64$**,** $c = 0.26$**,** $\delta = 0.99$**, and prune** $= 0$

N	Block size	Running time (sec)	Actual Total Norm	Smallest Norm Found	Ratio of found to actual
75	4	1797	6.16	64.00	10.4
75	6	1604	6.48	6.48	1.0
80	6	2776	6.78	64.00	9.4
80	8	3406	6.63	6.63	1.0
85	8	4614	6.93	64.00	9.2
85	10	5898	6.78	64.00	9.4
85	12	7536	6.93	64.00	9.2
85	14	8106	7.21	64.00	8.9
85	16	5168	6.78	6.78	1.0
88	16	11298	6.93	6.93	1.0
90	16	12987	6.93	64.00	9.2
90	18	2	6.78	13.42	2.0
95	18	25908	7.21	64.00	8.9
95	19	36754	7.21	64.00	8.9
95	20	59664	7.21	64.00	8.9
96	20	80045	7.07	7.07	1.0
98	20	75365	7.21	64.00	8.9
98	22	374034	7.07	7.07	1.0
100	22	183307	7.07	7.07	1.0

Table 2: BKZ-QP1 with $Q = 64$, $c = 0.26$, $\delta = 0.99$, **and prune** $= 0$

N	Block size	Running time (sec)	Actual Total Norm	Smallest Norm Found	Ratio of found to actual
75	2	1067	8.00	128.00	16.0
75	4	2699	8.00	121.90	15.2
75	6	3244	8.12	121.04	14.9
75	8	3026	7.87	7.87	1.0
80	8	6022	8.37	124.54	14.9
80	10	5452	8.12	8.12	1.0
85	10	10689	8.37	124.26	14.9
85	12	8171	8.37	8.37	1.0
90	12	15304	8.60	128.00	14.9
90	14	17802	8.83	126.60	14.3
90	16	20195	8.60	8.60	1.0
95	16	31338	9.17	128.00	14.0
95	18	54490	8.94	128.00	14.3
95	20	57087	8.83	8.83	1.0
100	20	109706	9.17	9.17	1.0

Table 3: BKZ-QP1 with $Q = 128$, $c = 0.23$, $\delta = 0.99$, **and prune** $= 0$

N	Block size	Running time (sec)	Actual Total Norm	Smallest Norm Found	Ratio of found to actual
75	4	2293	8.60	8.60	1.0
75	20	1930	8.72	8.72	1.0
78	4	3513	8.94	12.25	1.4
81	4	3422	9.38	221.22	23.6
81	6	3453	9.17	9.17	1.0
84	6	5061	9.17	9.17	1.0
87	6	6685	9.38	9.38	1.0
90	6	7085	9.49	256.00	27.0
90	8	9753	9.59	9.59	1.0
93	8	11900	9.90	254.55	25.7
93	10	14671	9.80	237.58	24.2
93	12	16946	9.70	9.70	1.0
96	12	22684	9.80	231.59	23.6
96	14	19854	9.90	9.90	1.0
99	14	30014	10.00	10.00	1.0
102	14	30817	10.20	239.62	23.5
102	16	64718	10.39	223.64	21.5
102	18	51207	10.39	10.39	1.0
105	18	81336	10.58	244.38	23.1
105	20	75860	10.30	10.30	1.0
108	20	197697	10.30	255.87	24.9
108	22	145834	10.30	10.30	1.0

Table 4: BKZ-QP1 with $Q = 256$, $c = 0.18$, $\delta = 0.99$, **and prune** $= 0$

N	Block size	Running time (sec)	Actual Total Norm	Smallest Norm Found	Ratio of found to actual
75	2	808	6000.00	64000.0	10.7
75	4	1895	6000.00	64000.0	10.7
75	6	2363	6000.00	7857.87	1.3
80	6	3582	6164.41	6164.78	1.0
85	6	5412	6324.56	64000.0	10.1
85	8	7252	6324.56	64000.0	10.1
85	10	8633	6324.56	64000.0	10.1
85	12	10074	6324.56	64000.0	10.1
85	14	12371	6324.56	64000.0	10.1
85	16	17729	6324.56	64000.0	10.1
85	18	16095	6324.56	6630.40	1.0
90	18	4	6480.74	12820.5	2.0
95	18	37998	6633.25	64000.0	9.6
95	20	43108	6633.25	64000.0	9.6
95	22	200195	6633.25	6900.34	1.0
96	22	240563	6633.25	64000.0	9.6
96	24	68054	6633.25	6779.54	1.0
98	24	1369730	6782.33	6852.89	1.0

Table 5: BKZ-QP1 with $Q = 64$**,** $c = 0.26$**,** $\alpha = 0.9097$**,** $\delta = 0.99$**, and prune** $= 0$

Jeffrey Hoffstein, Mathematics Department, Box 1917, Brown University, Providence, RI 02912 USA. ⟨jhoff@ntru.com⟩, ⟨jhoff@math.brown.edu⟩

Jill Pipher, Mathematics Department, Box 1917, Brown University, Providence, RI 02912 USA. ⟨jpipher@ntru.com⟩, ⟨jpipher@math.brown.edu⟩

Joseph H. Silverman, Mathematics Department, Box 1917, Brown University, Providence, RI 02912 USA. ⟨jhs@ntru.com⟩, ⟨jhs@math.brown.edu⟩