NTRU: A PUBLIC KEY CRYPTOSYSTEM

NTRU Cryptosystems, Inc. (www.ntru.com)
Jeff Hoffstein
Daniel Lieman
Jill Pipher
Joseph H. Silverman

Contents

0. INTRODUCTION

This purpose of this paper is to submit the NTRU public key cryptosystem for consideration for inclusion into the P1363A standard. NTRU was originally presented by Jeffrey Hoffstein in the rump session at CRYPTO '96, and was published in [HPS] in 1998. Since that time, NTRU Cryptosystems, Inc. has issued a number of technical reports. In some cases, these reports have described amplifications on the techniques in [HPS] in order to deal with new attacks (e.g., to deal with issues of plaintext awareness). In other cases, these reports have described fast algorithms for carrying out some of the computations required by NTRU. In this document, we have freely copied from [HPS] and other NTRU Cryptosystems, Inc. publications whenever appropriate. Although we have attempted to make this present document self-contained, at times we defer to the original documents and other references for detailed explanations and discussions. NTRU Cryptosystems, Inc. has also prepared documentation for NTRU in the P1363 format; this present paper does not include that documentation, since it is not structured in a manner appropriate for the P1363A call for submissions. In keeping with the format specified for P1363A submissions, we defer until Section 2 a discussion of the advantages of NTRU.

1. DESCRIPTION OF NTRU

NTRU is based on the algebraic structures of certain polynomial rings. The "hard problem" on which NTRU is based is the Short Vector Problem (finding a short vector in a lattice); this is discussed in much more detail in Section 3.6.

1.1 Notation. Before we proceed, we set some notation. The following are all part of the domain parameters for an implementation of NTRU.

n The dimension of the poynomial ring used in NTRU. (The polynomials will have degree $n - 1$.)

p A positive integer specifying a ring $\mathbb{Z}/p\mathbb{Z}$ over which the coefficients of a certain product of polynomials will be reduced during the encryption and decryption processes

q A positive integer specifying a ring $\mathbb{Z}/q\mathbb{Z}$ over which the coefficients of a certain product of polynomials will be reduced during the encryption and decryption processes; also used in the construction of the public key.

k A security parameter which controls resistance to certain types of attacks, including plaintext awareness.

d_f The distribution of the coefficients of the polynomial f, below (f is part of the private key).

d_g The distribution of the coefficients of the polynomial g, below (g is used to construct the public key).

d_r The number of 1s and -1s used in a certain random polynomial r, below, in the encryption process.

We will also use the following notation.

f A polynomial in $\mathbb{Z}[X]/(x^n - 1)$.

f_p A polynomial in $\mathbb{Z}[X]/(p, X^n - 1)$ (this is part of the private key). This polynomial is obtained by reducing the coefficients of f modulo p.

f_q A polynomial in $\mathbb{Z}[X]/(q, X^n - 1)$. This polynomial is obtained by reducing the coefficients of f modulo q.

$\mathcal{L}_f$ The set of polynomials in $\mathbb{Z}[X]/(x^n - 1)$ whose coefficients satisfy d_f

g A polynomial in $\mathbb{Z}[X]/(q, X^n - 1)$ (used with f_q to construct the public key).

$\mathcal{L}_g$ The set of polynomials in $\mathbb{Z}[X]/(x^n - 1)$ whose coefficients satisfy d_g.

$\mathcal{L}_r$ The set of polynomials in $\mathbb{Z}[X]/(x^n - 1)$ whose coefficients satisfy d_r.

f_p^{-1} The inverse of f_p in $\mathbb{Z}[X]/(p, X^n - 1)$.

f_q^{-1} The inverse of f_q in $\mathbb{Z}[X]/(q, X^n - 1)$.

h The public key, a polynomial in $\mathbb{Z}[X]/(q, X^n - 1)$.

r A polynomial in $\mathbb{Z}[X]/(q, X^n - 1)$ (used with h to encode a message).

m The plaintext message, a polynomial in $\mathbb{Z}[X]/(p, X^n - 1)$.

e The encrypted message, a polynomial in $\mathbb{Z}[X]/(q, X^n - 1)$.

G A *generating function* (defined below).

H A *hashing function* (defined below).

Throughout this paper, 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,

$$f \circledast g = h \quad \text{with} \quad h_k = \sum_{i=0}^{k} f_i g_{k-i} + \sum_{i=k+1}^{n-1} f_i g_{n+k-i} = \sum_{i+j \equiv k \bmod n} f_i g_j.$$

When we do a multiplication modulo (say) q, we mean to reduce the coefficients modulo q, so the result lies in $\mathbb{Z}[X]/(q, X^n - 1)$.

Remark. The naive computation of a product $f \circledast S$ requires n^2 multiplications. However, in a typical product used by NTRU, one of f or g has small coefficients that are all 0's and ± 1's, so $f \circledast g$ may be computed extremely rapidly. Further, for large values of n one may choose n to be highly divisible by 2, in which case the convolution product can be computed in $O(n \log n)$ operations by using Fast Fourier Transforms.

In addition to this convolution product, there are two other operations we need to define on rings of polynomials. These are a *generating function* and a *hashing function*. They are required in order to build a digital envelope into the NTRU protocol. We first let

$$\mathcal{P}_p(n) = \{\text{polynomials of degree at most } n - 1 \text{ with mod } p \text{ coefficients}\},$$

and we will write

$$[g]_p = \begin{cases} g \text{ with its coefficients reduced} \\ \text{modulo } p \text{ into the range } (-p/2, p/2]. \end{cases}$$

We may now describe more precisely what we mean by a generating function G and a hashing function H,

$$G : \mathcal{P}_p(N) \longrightarrow \mathcal{P}_p(N) \qquad \text{and} \qquad H : \mathcal{P}_p(N) \times \mathcal{P}_p(N) \longrightarrow \mathcal{P}_p(K).$$

These should be easy to compute, highly non-linear, and unpredictable. There are numerous examples of such functions, constructed out of shifts and other primitive operations, in the literature .

The NTRU PKC digital envelope depends on the choice of the functions G and H, and on an integer k. The probability of forging a valid ciphertext will be p^{-k}.

Remark. The original presentation of NTRU [HPS] did not suggest the use of a digital envelope (i.e., in the present discussion, both G and H would be functions which, no matter what the input, produce an output of 0). This provides an insecure digital envelope, as described in [NT7] (cf. [BKS]).

1.2 Key Creation. To create an NTRU key, Bob randomly chooses 2 polynomials $f \in \mathcal{L}_f$ and $g \in \mathcal{L}_g$. 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 (see [NT9]), and the actual computation of these inverses is easy using a modification of the Euclidean algorithm (see [NT1, NT14] for details). As noted above, we will denote these inverses by f_q^{-1} and f_p^{-1}, that is,

$$(1) \qquad\qquad f_q^{-1} \circledast f \equiv 1 \mod q \qquad \text{and} \qquad f_p^{-1} \circledast f \equiv 1 \mod p.$$

Bob next computes the quantity

$$(2) \qquad\qquad\qquad h \equiv p f_q^{-1} \circledast g \mod q.$$

Bob's public key is the polynomial h. Bob's private key is the polynomial f, although in practice he will also want to store f_p^{-1}. For an extremely efficient algorithm to compute f_p^{-1} and f_q^{-1}, please see [NT14]; for an efficient algorithm for multiplication, please see [NT10].

1.3 Encryption. We now describe how Alice wraps and sends a message to Bob using Bob's NTRU public key h. Alice chooses her plaintext m from the set

$$m \in \mathcal{P}_p(n - k).$$

She also chooses a random polynomial $r \in \mathcal{L}_r$. She computes

$$e \equiv r \circledast h + \left[m + H(m, [r \circledast h]_p)X^{n-k} + G([r \circledast h]_p) \right]_p \pmod{q}.$$

Alice then sends e to Bob.

1.4 Decryption. Suppose that Bob has received the message e from Alice and wants to decrypt it using his private key f. To do this efficiently, Bob should have precomputed the polynomial f_p^{-1} described in Section 1.1.

In order to decrypt e, Bob first computes the temporary polynomial a by

$$a \equiv f \circledast e \mod q,$$

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, Bob computes the temporary polynomial $t \in \mathbb{Z}[X]/(p, X^n - 1)$ by

$$t = f_p^{-1} \circledast a \quad (\mathrm{mod}\ p),$$

further computes the two temporary quantities

$$b \equiv e - t \quad (\mathrm{mod}\ p) \qquad \text{and} \qquad c \equiv t - G(b) \quad (\mathrm{mod}\ p),$$

and then writes c in the form

$$c = c' + c'' X^{n-k} \quad \text{with } \deg(c') < n - k \text{ and } \deg(c'') < k.$$

(Note that the quantity b is supposed to play the role of $[r * h]_p$.) Finally, he compares the quantities

$$c'' \qquad \text{and} \qquad H(c', b).$$

If they are the same, he accepts c' as a valid decryption. Otherwise he rejects the message as invalid.

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 Bob will be able to recover the message by choosing the coefficients of $a \equiv f \circledast e \mod q$ 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 Bob computes satisfies

$$
\begin{aligned}
a &\equiv f \circledast e \\
&\equiv f \circledast r \circledast h + f \circledast \big[m + H(m, [r \circledast h]_p) X^{n-k} + G([r \circledast h]_p) \big]_p \ (\mathrm{mod}\ q) \\
&= f \circledast pr \circledast f_q^{-1} \circledast g + f \circledast \big[m + H(m, [r \circledast h]_p) X^{n-k} + G([r \circledast h]_p) \big]_p \mod q \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{from (1),} \\
&= pr \circledast g + f \circledast \big[m + H(m, [r \circledast h]_p) X^{n-k} + G([r \circledast h]_p) \big]_p \mod q \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{from (2).}
\end{aligned}
$$

Consider this last polynomial. 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 Bob reduces the coefficients of $f \circledast e$ modulo q into the interval from $-q/2$ to $q/2$, he recovers *exactly* the polynomial

$$a = pr \circledast g + f \circledast \big[m + H(m, [r \circledast h]_p) X^{n-k} + G([r \circledast h]_p) \big]_p \quad \text{in} \quad R.$$

Reducing a modulo p then gives him the polynomial

$$f \circledast \left[m + H(m, [r \circledast h]_p)X^{n-k} + G([r \circledast h]_p) \right]_p \quad \text{in} \quad R,$$

and then multiplying by f_p^{-1} produces

$$t = m + H(m, [r \circledast h]_p)X^{n-k} + G([r \circledast h]_p) \quad \text{in} \quad \mathbb{Z}[X]/(p, X^n - 1).$$

Thus when Bob computes $b = e - t$ above, he is really recovering $b = r \circledast h$. Therefore his computation of c yields

$$c = m + H(m, [r \circledast h]_p)X^{n-k}.$$

Accordingly, c' is the original message m, and c'' should match up with the hash $H(m, [r \circledast h]_p) = H(m, b)$, as noted above.

1.6 Parameter choices - notation and a norm estimate. We define the *width* of an element $f \in R$ to be

$$|f|_\infty = \max_{0 \le i \le n-1}\{f_i\} - \min_{0 \le i \le n-1}\{f_i\}.$$

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

$$|f|_2 = \left(\sum_{i=0}^{n-1}(f_i - \bar{f})^2 \right)^{1/2}, \qquad \text{where} \quad \bar{f} = \frac{1}{n}\sum_{i=0}^{n-1} f_i.$$

(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*

$$\gamma_1 |f|_2 |g|_2 \le |f \circledast g|_\infty \le \gamma_2 |f|_2 |g|_2 \,.$$

Of course, this proposition would be useless from a practical viewpoint 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.

1.7 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

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

$$\left. \text{and has degree at most } n - k - 1 \right\}.$$

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

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

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

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

(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 $r \in \mathcal{L}_r$ have L^2 norms

$$|f|_2 = \sqrt{2d_f - 1 - n^{-1}}, \qquad |g|_2 = \sqrt{2d_g}, \qquad |r|_2 = \sqrt{2d_r}.$$

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

1.8 A Decryption Criterion. To ease notation, we let

$$m' = \left[m + H(m, [r \circledast h]_p)X^{n-k} + G([r \circledast h]_p) \right]_p$$

be the polynomial used by Alice for encryption. (That is, $e \equiv r \circledast h + m' \mod q$.) In order for the decryption process to work, it is necessary that

$$|f \circledast m' + pr \circledast g|_\infty < q.$$

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

$$|f \circledast m'|_\infty \leq q/4 \qquad \text{and} \qquad |pr \circledast g|_\infty \leq q/4;$$

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

(3) $$|f|_2 \, |m|_2 \approx q/4\gamma_2 \qquad \text{and} \qquad |r|_2 \, |g|_2 \approx q/4p\gamma_2$$

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

2. Attributes and Advantages of NTRU

Compared to other public key cryptosystems at roughly equivalent levels of security, NTRU offers:

- more efficient encryption and decryption, in both hardware and software implementations;
- much faster key generation, allowing the use of "disposable" keys (because keys are computationally "cheap" to create).

2.1 Theoretical Operating Specifications. In this section we consider the theoretical operating characteristics of the NTRU PKCS. There are four integer parameters (n, p, q, k), four sets $\mathcal{L}_f, \mathcal{L}_g, \mathcal{L}_r, \mathcal{L}_m$ determined respectively by integers d_f, d_g, d_r, p as described in Sections 1.1 and 1.7. The following table summarizes the NTRU PKCS operating characteristics in terms of these parameters.

Plain Text Block	$(n - k) \log_2 p$ bits
Encrypted Text Block	$n \log_2 q$ bits
Encryption Speed*	$O(n^2)$ operations
Decryption Speed	$O(n^2)$ operations
Message Expansion**	$\frac{n}{n-k} \log_p q$-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
** May be reduced, see section 4

2.2 Comparison With Other PKCS's. There are currently a number of public key cryptosystems in the literature, including:

- the RSA cryptosystem of Rivest, Shamir, and Adelman [RSA] based on the difficulty of factoring large integers;
- the ElGamal cryptosystem, based on the difficulty of solving the discrete logarithm problem in the multiplicative group of a field;
- various elliptic curve cryptosystems (ECC), based on the difficulty of solving the discrete logarithm problem in the group of points on an elliptic curve;
- the McEliece cryptosystem [MCEL] based on error correcting codes;
- lattice based cryptosystems such as those of Atjai-Dwork [AD] and Goldreich, Goldwasser, and Halevi [GHH], 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 ⊛-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 that 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 AD and GGH cryptosystems, in which decryption is based on knowledge of short lattice bases. In this aspect, the AD and GGH systems 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.

Another difference of vital practical importance between AD/GHH and NTRU is the ratio of key size to lattice dimension. For a lattice of dimension D, both AD and GHH use a key that is a (special) basis for the lattice, hence has size $O(D^2)$. This means that keys become impractically large for lattices of dimension a few hundred, while it turns out that lattices of lower dimension are susceptible to lattice reduction attacks. An NTRU key for a lattice of dimension D has size only $O(D \log D)$, hence NTRU remains practical for lattices of high dimension.

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/block size 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.

Preliminary security and timing comparisons between NTRU and RSA are given in Table 1. Public key cryptosystems are primarily used either to exchange a secret key or a short message (e.g., a credit card number). RSA and NTRU work in units of message blocks, and in either system, a single message block is large enough to hold a short message or a secret key of very high security. Thus for comparison purposes we measure the time to encrypt and decrypt a single message block.

System	Security (MIPS yrs)	Key Size (bits)	Create Key (millisecs)	Encrypt (blks/sec)	Decrypt (blks/sec)
RSA 512	$4.00 \cdot 10^5$	512	260	2441	122
NTRU 167	$2.08 \cdot 10^6$	1169	4.0	5941	2818
RSA 1024	$3.00 \cdot 10^{12}$	1024	1280	932	22
NTRU 263	$4.61 \cdot 10^{14}$	1841	7.5	3676	1619
RSA 2048	$3.00 \cdot 10^{21}$	2048	4195	310	3
RSA 4096	$2.00 \cdot 10^{33}$	4096	—	—	—
NTRU 503	$3.38 \cdot 10^{35}$	4024	17.3	1471	608

Comparison of NTRU and RSA

Notes for Table 1:

(1) "NTRU n" refers to an implementation of the NTRU public key cryptosystem using the domain parameter n, that is, using polynomials of degree $n - 1$ in the ring $\mathbb{Z}[X]/(x^n - 1)$. See Section 3.7 for details.

(2) Security is measured in MIPS-years required to break the system. Note that all security times are estimates, based on extrapolation of the breaking time for lower security levels. The security figures for NTRU are from [HPS] and [NT12], see also section 3. The security figures for RSA are from IEEE P1363 (draft 9), Standard Specifications of Public Key Cryptography, Section D.4.3.4.

(3) Key size refers to public key size in bits. NTRU private keys are shorter than their public keys. RSA public and private keys are the same size.

(4) NTRU encryption, decryption, and key creation performed using Tao Group's Tumbler implementation of the NTRU algorithm, programmed in C and running on a 300 MHz Pentium II operating under Linux.

(5) RSA key creation done on a 255 MHz Digital AlphaStation.

(6) RSA encryption/decryption programmed in Microsoft Visual C++ 5.0 (optimized for speed, Pentium Pro code generation), and run on a Pentium II 266MHz machine under Windows NT 4.0. RSA encryption uses exponent 17 to increase speed. See `<http://www.eskimo.com/~weidai/benchmarks.txt>` for details.

(7) 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.

3. Security Considerations

In what follows, we will assume that the desired level of "high" security against offline attacks is 2^{80}.

3.1 Security Analysis. RSA may be attacked by factoring, so to be secure, it must use integers sufficiently large so as to make factoring infeasible. Similarly, NTRU may be attacked by algorithms that find short vectors in a lattice, so it must use parameters sufficiently large so as to make it infeasible to find short vectors. We begin by describing some more elementary attacks, then we give a detailed description of the lattice attacks.

3.2 Brute force attacks. An attacker can recover the private key by trying all possible $f \in \mathcal{L}_f$ and testing if $f \circledast h \mod q$ has small entries, or by trying all $g \in \mathcal{L}_g$ and testing if $g \circledast h^{-1} \mod q$ has small entries. Similarly, an attacker can recover a message by trying all possible $r \in \mathcal{L}_r$ and testing if $e - r \circledast h \mod q$ 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 $\#\mathcal{L}_r$. 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

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

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

3.3 Meet-in-the-middle attacks. Recall that we denote an encrypted message by e. Andrew Odlyzko has pointed out that there is a meet-in-the-middle attack that can be used against r, 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 r from sets containing around 2^{160} elements. (For further details, see [NT4].)

3.4 Multiple transmission attacks. This attack only applies when the hash function H and generating function G are both identically 0, as they were in the original article introducing NTRU [HPS]. If Alice sends a single message m several times using the same public key but different random r's, then the attacker Eve will be able to recover a large part of the message. Briefly, suppose that Alice transmits $e_i \equiv r_i \circledast h + m \mod q$ for $i = 1, 2, \ldots, l$. Eve can then compute $(e_i - e_1) \circledast h^{-1} \mod q$, thereby recovering $r_i - r_1 \mod q$. (Actually, h^{-1} does not exist, see below.) The coefficients of the r's are so small that she recovers exactly $r_i - r_1$, and from this Eve will recover many of the coefficients of r_1. If l is of even moderate size (say 4 or 5), Eve will recover enough of r_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 Eve decrypts a single message in this fashion, this information will not assist her in decrypting any subsequent messages.

As noted above, h does not have an inverse modulo q. However, it does have an easily computed "pseudo-inverse" with the property that $a \circledast h \circledast h^{-1} \equiv a \mod q$ for all polynomials a satisfying $a(1) = 0$. Eve can use this pseudo-inverse.

3.5. Semantic security. As formulated in this note, the NTRU cryptosystem is not entirely semantically secure, it leaks approximately $\log_2(q)$ bits. Recall that the encrypted message has the form $e \equiv r \circledast h + m' \mod q$. The construction of h implies that $h(1) \equiv 0 \mod q$, and hence that $e(1) \equiv m'(1) \mod q$. Thus an attacker recovers the value of $m'(1) \mod q$. If desired, this semantic insecurity can be eliminated by reserving one coefficient of m' and setting this "check coefficient" so that $m'(1) \equiv 0 \mod q$.

3.6 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 [LLL], with various improvements due to Schnorr and others [SCHN, SCHO, SCEU], will find

moderately 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, but for full details, see [HPS] and [NT12].

3.6.1 Lattice attack on an NTRU private key. Consider the $2n$-by-$2n$ matrix composed of four n-by-n blocks:

$$
\begin{pmatrix}
\alpha & 0 & \cdots & 0 & h_0 & h_1 & \cdots & h_{n-1} \\
0 & \alpha & \cdots & 0 & h_{N-1} & h_0 & \cdots & h_{n-2} \\
\vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & \alpha & h_1 & h_2 & \cdots & h_0 \\
\hline
0 & 0 & \cdots & 0 & q & 0 & \cdots & 0 \\
0 & 0 & \cdots & 0 & 0 & q & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & 0 & 0 & 0 & \cdots & q
\end{pmatrix}
$$

(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 experimental evidence we have obtained, the time required to find τ or a similarly short vector appears to be (at least) exponential in n, with a constant in the exponent proportional to c_h.

3.6.2 Lattice attack on an NTRU message. A lattice attack may also be directed against an individual message m, or more precisely, against the digital envelope

$$m' = \big[m + H(m, [r \circledast h]_p)X^{n-k} + G([r \circledast h]_p)\big]_p$$

containing the message m. Here the associated lattice problem is very similar to that for h, and the target vector will have the form $(\alpha m', r)$. As before, the attacker should balance the lattice using $\alpha = |r|_2 / |m'|_2$, which leads to the value

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

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, $|f|_2 |g|_2 \approx |m'|_2 |r|_2$. For concreteness, we will now restrict to the case $p = 3$; other values may be analyzed similarly. For $p = 3$, an average m' will consist of $N/3$ each of 1, 0 and -1, so $|m'|_2 \approx \sqrt{2N/3}$. Similarly, r consists of d each of 1 and -1, with the rest 0's, so $|r|_2 = \sqrt{2d}$. 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.

3.6.3 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 3.6.1) 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,

$$f' \circledast e \equiv pr \circledast g' + m' \circledast f' \mod q$$

satisfies $|pr \circledast g' + m' \circledast f'|_\infty < q$, 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 [CS]. However experimental evidence suggests that the existence of spurious keys does not pose a security threat, because lattice reduction algorithms either terminate with a useless "q-vector" of the form $(0, 0, \ldots, 0, q, 0 \ldots, 0)$, or they find the actual target vector. See Section 4.2 of [HPS] for a further discussion of this point.

3.6.4 Experimental results. We briefly report on the experiments described in more detail in [HPS] and [NT12]. These experiment used Victor Shoup's NTL implementation of the LLL algorithm with improvements due to Schnorr, Euchner and Hoerner. The NTL package is available at [SHOU]. The program was run using increasing block sizes until it found the target vector (or a vector slightly longer than the target vector). An important observation from the experiments is that (at least for the NTRU lattices), it appears that LLL generally either finds a vector of the exact correct length, or it finds one that is considerably too long to be useful for decryption. Thus the idea of Coppersmith and Shamir [CS] to exploit vectors a little longer than the target vector to attack NTRU, while very interesting as a theoretical remark, does not appear to be of practical significance. In practice, LLL generally seems to terminate with a q-vector (i.e., a vector with one coordinate equal to q and the rest 0) until a sufficiently large block size is used, at which time it finds the target vector. Further, the necessary block size appears to increase more-or-less linearly with the dimension, while the running time of LLL appears to increase exponentially with the block size. Based on the experiments in [HPS] and [NT12], we give in the next table the extrapolated breaking times for the specific parameter choices described in Section 3.7 below. Note that these values are only estimates that may vary due to the particular definition of MIPS-year and to the difficulty of estimating actual processor utilization. All experiments were run on a 400 MHz Celeron machine.

System	T (seconds)	T (MIPS-years)
NTRU 167	$1.638 \cdot 10^{11}$	$2.077 \cdot 10^{6}$
NTRU 263	$3.634 \cdot 10^{19}$	$4.607 \cdot 10^{14}$
NTRU 503	$2.663 \cdot 10^{40}$	$3.375 \cdot 10^{35}$

Estimated Breaking Times

3.6.5 Zero-forced lattices. Alexander May [MAY] has described an improved method of applying LLL to NTRU lattices by exploiting the large number of zeros in f and g. Briefly, his idea is to guess a certain number of (consecutive) indices in the target vector that are equal to 0 and use this to effectively reduce the dimension of the lattice. Since various rotations of the target vector also serve as target vectors (due to the circulant nature of the lattice matrix), the probability of guessing a pattern of a moderate number of zeros is reasonably high, but decreases very rapidly if one attempts to find a pattern with a large number of zeros. May's paper [MAY] contains some theoretical results and data from his experiments. A reformulation and slight strengthening of the method, with further analysis, are given in [NGU] and [NT13]. May also describes a second attack, also analyzed in [NT13], in which one merely discards some of the coordinates of the target vector. The final conclusion is that the new attacks only marginally affect the security levels of the recommended NTRU parameter sets ($N = 167$, 263, and 503), but that the new lattices can be helpful for smaller parameters (e.g., $N = 107$). We refer the reader to [NT13], [MAY], and [NGU] for further details.

3.7 Practical Implementations of NTRU

We will now present three distinct sets of parameters that 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 encryption of television, pager, and cellular telephone transmissions.

$$(N, p, q, k) = (167, 3, 128, 49)$$
$$\mathcal{L}_f = \mathcal{L}(61, 60), \quad \mathcal{L}_g = \mathcal{L}(20, 20), \quad \mathcal{L}_r = \mathcal{L}(18, 18),$$

In other words, f is chosen with 61 1's and 60 -1's (i.e., $d_f = 61$), g is chosen with 20 1's and 20 -1's, (i.e., $d_g = 12$), and r is chosen with 18 1's and 18 -1's (i.e., $d_r = 18$). These give key and message sizes

$$\text{Private Key} = 530 \text{ bits}, \quad \text{Public Key} = 1169 \text{ bits}, \quad \text{and} \quad \text{Plaintext} = 187 \text{ bits},$$

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

$$\text{Key Security} = 2^{82.9} \quad \text{and} \quad \text{Message Security} = 2^{77.5}.$$

(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

$$c_h = 0.236, \quad c_m = 0.225, \quad \text{and} \quad s = 0.296q.$$

Case B: High Security

$$(N, p, q, k) = (263, 3, 128, 52)$$
$$\mathcal{L}_f = \mathcal{L}(50, 49), \quad \mathcal{L}_g = \mathcal{L}(24, 24), \quad \mathcal{L}_r = \mathcal{L}(16, 16)$$
$$\text{Private Key} = 834 \text{ bits}, \quad \text{Public Key} = 1841 \text{ bits}, \quad \text{and} \quad \text{Plaintext} = 335 \text{ bits},$$
$$\text{Key Security} = 2^{110.6} \quad \text{and} \quad \text{Message Security} = 2^{82.1}$$
$$c_h = 0.187, \quad c_m = 0.195, \quad \text{and} \quad s = 0.409q.$$

Case C: Highest Security

$$(N, p, q, k) = (503, 3, 256, 107)$$
$$\mathcal{L}_f = \mathcal{L}(216, 215), \quad \mathcal{L}_g = \mathcal{L}(72, 72), \quad \mathcal{L}_r = \mathcal{L}(55, 55)$$
$$\text{Private Key} = 1595 \text{ bits}, \quad \text{Public Key} = 4024 \text{ bits}, \quad \text{and} \quad \text{Plaintext} = 628 \text{ bits},$$
$$\text{Key Security} = 2^{285} \quad \text{and} \quad \text{Message Security} = 2^{170},$$
$$c_h = 0.182, \quad c_m = 0.160, \quad \text{and} \quad s = 0.0.365q.$$

The parameter sets described in this section are summarized in the follwoing table.

	N	p	q	k	d_f	d_g	d_r
NTRU167	167	3	128	49	61	20	18
NTRU263	263	3	128	52	50	24	16
NTRU503	503	3	256	107	216	72	55

NTRU Parameter Sets

4. Known Limitations and Disadvantages

The NTRU PKCS's for the sample parameters presented in Section 3.7 have moderately large message expansions. However, as the principal use for PKCS's is the exchange of a private key in a single message block this is generally not a significant problem. It may be worth mentioning, though, that there is a simple masking technique that can be used to signficantly reduce message expansion. With this approach, Alice sends a pair of polynomials (e_1, e_2). The first polynomial is the encryption $e_1 \equiv r_1 \circledast h + r_2 \mod q$, where r_2 is a randomly chosen polynomials with all coefficients equal to -1, 0 and 1. The second polynomial is $e_2 \equiv r_2 \circledast h + m' \mod q$, where m' is the plaintext message in a suitable digital envelope modulo q. Since Bob can decrypt e_1 to recover r_2, he is able to recover m' modulo q. In other words, at the cost of doubling the length of the encrypted message to $2n \log_2 q$ bits, Alice is able to send to Bob a $n \log_2 q$ bits of information. In principle, this reduces message expansion to 2-to-1, although the use of a digital envelope will naturally increase message expansion.

5. Intellectual Property Issues

The NTRU public key cryptosystem is patent pending. All rights have been assigned to NTRU Cryptosystems, Inc., of Rhode Island. If NTRU is incorporated into the P1363A standard, NTRU Cryptosystems, Inc. will issue a letter of assurance confirming that its policy has always been to license NTRU in a "reasonable and non-discriminatory" manner, and that NTRU Cryptosystems, Inc. will continue to license NTRU in this way. In the event any text from this submission or other NTRU Cryptosystems, Inc. supplied documents is incorporated into the P1363A standard, NTRU Cryptosystems, Inc. will execute any necessary copyright transfers.

NTRU is a trademark of NTRU Cryptosystems, Inc.

own, although we stress that any oversights or errors in this analysis are entirely of our own devising.

REFERENCES

[AD]   M. Ajtai, C. Dwork, *A public-key cryptosystem with worst case/average case equivalence*, Proc. 29th ACM Symposium on Theory of Computing, pp. 284–293.

[BKS]   D. Bleichenbacher, B. Kaliski, J. Staddon, *Recent results on PKCS#1: RSA encryption standard*, RSA Laboratories' Bulletin, Number 7, June 26, 1998..

[CS]   D. Coppersmith, A. Shamir, *Lattice attacks on NTRU*, Preprint, April 5, 1997; presented at Eurocrypt 97.

[GGH]   O. Goldreich, S. Goldwasser, S. Halevi, *Public-key cryptosystems from lattice reduction problems*, MIT – Laboratory for Computer Science preprint, November 1996.

[HPS]   J. Hoffstein, J. Pipher and J. Silverman, *NTRU: A ring based public key cryptosystem*, Algorithmic Number Theory (ANTS III), Portland, OR, June 1998, J.P. Buhler (ed.), Lecture Notes in Computer Science **1423**, 267–288.

[LLL]   A.K. Lenstra, H.W. Lenstra, L. Lovśz, *Factoring polynomials with polynomial coefficients*, Math. Annalen **261**, 515–534.

[MCEL]   R.J. McEliece, *A public-key cryptosystem based on algebraic coding theory*, JPL Pasadena, DSN Progress Reports **42–44** (1978), 114–116.

[MAY1]   A. May, *Cryptanalysis of NTRU*, preprint, February 1999 (a portion of a Master's thesis).

[MAY2]   A. May, *New lattice attacks on NTRU*, preprint, April 1999.

[NGU]   P. Nguyen, *Lattice attacks on NTRU, revisited*, preprint, March 1999.

[RSA]   R.L. Rivest, A. Shamir, L. Adleman, *A method for obtaining digital signatures and public key cryptosystems*, Communications of the ACM **21**, 120–126.

[SCHN]   C.P. Schnorr, *Block reduced lattice bases and successive minima*, Combinatorics, Probability and Computing **3**, 507–522.

[SCEU]   C.P. Schnorr, M. Euchner, *Lattice basis reduction: improved practical algorithms and solving subset sum problems*, Mathematical Programing **66**, 181-199.

[SCHO]   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, pp. 1–12.

[SHOU]   V. Shoup, *NTL — A Number Theory Library*, `http://www.cs.wisc.edu/~shoup/ntl/`.

[NT1]   J.H. Silverman, *NTRU: Pseudo-code implementation*, NTRU Cryptosystems Technical Report **1**, available at http://www.ntru.com.

[NT4]   J.H. Silverman, *A Meet-In-The-Middle Attack on an NTRU Private Key*, NTRU Cryptosystems Technical Report **4**, available at http://www.ntru.com.

[NT7]   Joseph H. Silverman, *Plaintext awareness and the NTRU PKCS*, NTRU Cryptosystems Technical Report **7**, available at http://www.ntru.com.

[NT9]   Joseph H. Silverman, *Invertibility in Truncated Polynomial Rings*, NTRU Cryptosystems Technical Report **9**, available at http://www.ntru.com.

[NT10]   Joseph H. Silverman, *High-Speed Multiplication of (Truncated) Polynomials*, NTRU Cryptosystems Technical Report **10**, available at http://www.ntru.com.

[NT12]   Joseph H. Silverman, *Estimated Breaking Times for NTRU Lattices*, NTRU Cryptosystems Technical Report **12**, available at http://www.ntru.com.

[NT13]   Joseph H. Silverman, *Dimension-Reduced Lattices, Zero-Forced Lattices, and the NTRU Public Key Cryptosystem*, NTRU Cryptosystems Technical Report **13**, available at http://www.ntru.com.

[NT14]   Joseph H. Silverman, *Almost Inverses and Fast NTRU Key Creation*, NTRU Cryptosystems Technical Report **14**, available at http://www.ntru.com.