Optimizations for NTRU

Jeffrey Hoffstein, Joseph Silverman

Abstract. In this note we describe a variety of methods that may be used to increase the speed and efficiency of the NTRU public key cryptosystem.

1991 Mathematics Subject Classification: 94A60, 11T71.

1. An Overview of NTRU

The NTRU Public Key Cryptosystem is based on ring theory and relies for its security on the difficulty of solving certain lattice problems. In this section we will briefly review the properties of NTRU that are relevant to the topics in this paper. For further details and a security analysis of NTRU, see [HPS,S1,S2].

A general formulation of the NTRU Public Key Cryptosystem uses a ring R and two (relatively prime) ideals p and q in R. A rough outline of the key creation, encryption, and decryption processes is as follows:

- **Key Creation**
  Bob creates a public key h by choosing elements $f, g \in R$, computing the mod q inverse f_q^{-1} of f, and setting
  $$h \equiv f_q^{-1} * g \pmod{q}.$$
  Bob's private key is the element f. Bob also precomputes and stores the mod p inverse f_p^{-1} of f.
- **Encryption**
  In order to encrypt a plaintext message $m \in R$ using the public key h, Alice selects a random element $r \in R$ and forms the ciphertext
  $$e \equiv r * h + m \pmod{q}.$$
- **Decryption**
  In order to decrypt the ciphertext e using the private key f, Bob first computes
  $$a \equiv f * e \pmod{q}.$$

He chooses $a \in R$ to satisfy this congruence and to lie in a certain prespecified subset R_a of R. He next does the mod p calculation

$$f_p^{-1} * a \pmod{p},$$

and the value he computes is equal to m modulo p.

Remark. In practice the elements f, g, r, and m are taken from certain large prespecified subsets R_f, R_g, R_r, and R_m of R. These sets (and R_a) are chosen large enough so that it is infeasible for an attacker to find f or g from h or to find r or m from e, while at the same time they are chosen so that the decryption process works.

The general NTRU cryptosystem described above makes no specification of the ring R or its subsets R_f, R_g, R_m, R_r, R_a. A standard implemention of NTRU uses the ring of convolution polynomials

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

The sets R_f, R_g, R_m, R_r are sets of "small" polynomials, which means that their coefficients are chosen to be small, generally either binary $\{0, 1\}$ or trinary $\{-1, 0, 1\}$ and possibly with a specified number of nonzero coefficients. A typical choice for (p, q) is $(3, 128)$, and in order for decryption to work properly it is necessary that the coefficients of the polynomial

$$p * r * g + m * f \tag{1}$$

lie in an interval of length at most q. That is, the difference between the largest coefficient and the smallest coefficient should not exceed q. In this case we will say that the polynomial is *narrow* mod q. If p, q, and the sets R_f, R_g, R_m, R_r are chosen appropriately, then the polyomial (1) will be narrow mod q for most choices of f, g, m, r.

2. Guarding Against Chosen Ciphertext Attacks

There are various ways to construct adaptive chosen ciphertext attacks against the NTRU cryptosystem, see for example [HS2,JJ]. Padding techniques of Fujisaki and Okamoto [FO1,FO2] can be used to prevent such attacks. Further, if the NTRU plaintext is not scrambled in some way, then there may be dictionary attacks against part or all of the plaintext. An adaptation for NTRU of the Fujisaki-Okamoto method with addded message scrambling works as follows (see [HS3] for details).

Let M be Alice's plaintext. She chooses a collection of random bits R and uses an all-or-nothing transformation on the concatenation $M \| R$. For example, she could split each of M and R into equal size pieces $M = M_1 \| M_2$ and $R = R_1 \| R_2$

and then use hash functions $\mathcal{H}_1$ and $\mathcal{H}_2$ to compute

$$m_1 = (M_1 \| R_1) \oplus \mathcal{H}_1(M_2 \| R_2) \qquad \text{and} \qquad m_2 = (M_2 \| R_2) \oplus \mathcal{H}_2(m_1).$$

(Here $\oplus$ denotes bitwise exclusive or.) The NTRU plaintext m is set equal to $m = m_1 \| m_2$. The NTRU "random" element r is also computed using a hash function

$$r = \mathcal{H}_3(M \| R).$$

(Of course, r is no longer random. We have moved the randomness into the plaintext as the bit string R.)

Alice computes the ciphertext $e \equiv r * h + m \pmod q$ is as usual. When Bob decrypts the message, he recovers the plaintext m, and then he inverts the all-or-nothing transformation to recover M and R. However, before accepting the plaintext M as valid, he recomputes $r = \mathcal{H}_3(M \| R)$ and checks that e is acually equal to $r * h + m \pmod q$. If this is not true, then he rejects the message. This prevents Alice from sending Bob specially constructed ciphertexts e whose decryption might allow her to deduce information about Bob's private key.

It is important to note that Alice has no control over the "random" element r. So if she sends Bob a purported ciphertext e, then the quantity $r * h$ that Bob computes will be essentially a random element modulo q. If R/qR is sufficiently large, there is little chance of choosing particular values for r and mounting an adaptive attack.

The cost of guarding against chosen ciphertext attacks is that Bob needs to reencrypt the message. (Both Bob and Alice also need to compute some hash functions, but in practice, these take very little time.) Thus Bob needs to compute the extra product $r * h \bmod q$. However, Bob's interest in computing this product is simply to compare it with $e - m$. It may be possible to do this without computing the full product. For example, if NTRU is implemented with a convolution polynomial ring $R = \mathbf{Z}[X]/(X^N - 1)$, then Bob might compute only k coefficients of $r * h$ and just compare those k coefficients to $e - m$. For an appropriate choice of k, he can be fairly confident that if the k coefficients match, then all of the coefficients will match. The probability of any particular coefficient of $r * h$ matching the corresponding coefficient of $e - m$ is approximately $1/q$. For a typical value such as $q = 128$, it would suffice to check (say) 12 coefficients to achieve a spoofing probability of under 2^{-84}.

3. Invertibility of f Modulo p

In practical situations, the most time consuming part of the NTRU encryption and decryption processes is multiplication in the ring R. The NTRU encryption process involves one product $r * h \bmod q$, while decryption requires the computation of two products $f * e \bmod q$ and $f_p^{-1} * a \bmod p$. Similarly, the most time consuming part of the NTRU key creation process is the computation of the inverses f_p^{-1} and f_q^{-1}.

In order to speed both the key creation and the the decryption processes, the user can choose the element f to have the form

$$f = 1 + p * f_1 \quad \text{with } f_1 \in R.$$

Notice that an f of this form has the property that $f_p^{-1} = 1$, so it is not necessary to compute the inverse modulo p, and the second multiplication in the decryption process disappears.

Remark. It is important to verify that the specified set of elements of the form $f = 1+p*f_1$ provides the desired level of security. In practice, this means that one must assume that an attacker can use his search methods on the set of f_1's. In particular, the lattice search methods described in [HPS,S1,S2] can be easily modified to search for the pair (f_1, g), so one must analyze these alternative lattices.

Remark. In practice, the prime p is the "small" prime and the prime q is the "large" prime (e.g., $(p,q) = (2,127)$ or $(3,128)$). The performance gain using $f = 1+p*f_1$ comes from eliminating an inversion modulo p in key creation and a multiplication modulo p in decryption, so this may not seem significant. However, it may happen that operations modulo p are as costly as operations modulo q, even if p is smaller than q. To take a specific example, if $p = 3$ and $q = 128$, then basic operations modulo q are no more difficult on a computer than those modulo p. [In terms of computational efficiency, the ideal situation would be to take $p = 2$ and $q = 128$, but then $e \equiv m \pmod{p}$, so the system becomes insecure. This is why it is important that the ideals p and q be relatively prime in the ring R.]

Remark. If the coefficients of the polynomial

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

lie in an interval of length greater than q, then the decryption process will not work properly. For an appropriate choice of parameters, this will happen very rarely. If we take f in the form $f = 1 + p * f_1$, then

$$a = m + p * (g * r + m * f_1),$$

so we see that coefficients that "wrap" (i.e., that lie outside the chosen mod q interval) will each affect a single coefficient of the decryption. This means that if a small amount of error correction is put into m, then it will be possible to correct messages that have a small amount of wrapping. Although this cannot be recommended for high security applications, since redundancy in m might open some lines of statistical attacks on long transcripts of messages, it could be useful in moderate security situations where one wants to virtually eliminate the possiblity of a wrapping decryption failure.

4. Taking p to be a Polynomial

If f is taken in the form $f = 1 + p * f_1$, then the polynomial

$$p * r * g + m * f$$

that is being formed during the decryption process becomes

$$p * (r * g + m * f_1) + m, \qquad (2).$$

It is clear that as p becomes larger, there is more likelihood that the polynomial will be too wide. On the other hand, there are many computational advantages to taking q to be a power of 2, which would seem to force p to be at least 3.

However, as the description of NTRU makes clear, there is no particular reason that p needs to be an integer. It could instead be a polynomial, as long as the ideals generated by p and q are relatively prime in the ring R. Notice that this condition on p and q is equivalent to requiring that the elements p, q, and $X^N - 1$ generate the unit ideal in $\mathbf{Z}[X]$.

There is a second issue related to the practical matter of retrieving the plaintext m. The plaintext polynomial $m(X)$ needs to be small, say with binary or trinary coefficients; yet the decryption process only recovers $m(X)$ modulo p. In other words, we recover the image of $m(X)$ in the quotient ring

$$\frac{R}{pR} = \frac{\mathbf{Z}[X]}{(X^N - 1, p)}.$$

For example, if $m(X)$ has binary coefficients, then there are 2^N possible values for $m(X)$, and we want to ensure that the map

$$\{\text{binary polynomials } m(X)\} \longrightarrow R/pR \qquad (3)$$

is injective. Even more than this, we need to know how to invert this map in order to recover m.

We also want the coefficients of p to be very small, since we need the polynomial (2) to be narrow. A natural candidate for p would be a polynomial of the form $p = X^k \pm 1$. Unfortunately, the elements

$$X^k \pm 1 \text{ and } X^N - 1 \text{ and } 128$$

are not relatively prime in the ring $\mathbf{Z}[X]$.

A second natural candidate is a polynomial of the form $p = X^k \pm X^j \pm 1$. These polynomials usually satisfy the relative primality condition, but the quotient R/pR will not be large enough and the reduction map (3) will not be injective. For example, if $p = X^2 + X + 1$ and N is a large prime, then R/pR will have approximately 1.618^N elements. (Here $1.618\ldots$ is the golden ratio $(1 + \sqrt{5})/2$.)

We thus turn to the polynomial

$$p = X + 2. \qquad (4)$$

(The polynomial $X - 2$ works similarly, but is slightly less efficient.) We will assume that N is odd and $N \geq 7$. We begin by verifying that the polynomial $p = X + 2$

satisfies the relative primality condition. We write

$$X^N - 1 = X^N + 2^N - 2^N - 1 = (X^N + 2^N) - 128 \cdot 2^{N-7} - 1. \tag{5}$$

Notice that $X^N + 2^N = (X+2)u(X)$ with $u(X) = X^{N-1} - 2X^{N-2} + \cdots + 2^{N-1}$, so we can rewrite (5) as

$$1 = (X+2)u(X) - (X^N - 1) - 128 \cdot 2^{N-1}.$$

This shows that the three elements $X+2$, $X^N - 1$, and 128 generate the unit ideal in $\mathbf{Z}[X]$.

Next we consider the quotient ring R/pR when $p = X + 2$. An elementary calculation shows that this ring has $2^N + 1$ elements. We will prove in Section 5 that the map (3) is injective and we will describe a fast algorithm for inverting it. This suggests that taking $p = X + 2$ and $q = 128$ provides for an efficient implementation of the NTRU public key cryptosystem. A standard security analysis of the underlying lattice problem shows that $N = 251$ provides a security level comparable to that given by RSA with 1024 bit keys.

5. Working With p Equal To $X + 2$

In this section we will prove that binary plaintext messages $m(X)$ can be uniquely recovered if they are known modulo $X+2$. We begin by proving that distinct $m(X)$ give distinct residue classes. For this section, we fix the notation

$$R = \frac{\mathbf{Z}[X]}{(X^N - 1)}, \qquad R_m = \{\text{binary polynomials in } R\}, \qquad p = p(X) = X + 2.$$

Proposition. (a) *The evaluation map*

$$R/p(X)R \longrightarrow \mathbf{Z}/(2^N+1)\mathbf{Z}, \qquad a(X) \longmapsto a(-2),$$

is an isomorphism.
(b) *The map*

$$R_m \longrightarrow R/p(X)R$$

is one-to-one.

Proof. The evaluation map $a(X) \mapsto a(-2)$ gives an isomorphism $\mathbf{Z}[X]/(X+2) \cong \mathbf{Z}$. The element $X^N - 1$ in the lefthand ring corresponds to the integer $(-2)^N - 1 = -(2^N + 1)$ in $\mathbf{Z}$. (Note we are assuming that N is odd.) Hence the quotient of $\mathbf{Z}[X]/(X+2)$ by the ideal $(X^N - 1)$ (which is the same as $R/p(X)R$) is isomorphic to the quotient of $\mathbf{Z}$ by $2^N + 1$. This proves (a).

Using the identification $R/p(X)R \cong \mathbf{Z}/(2^N+1)\mathbf{Z}$ from (a), we need to show that if $m(X)$ and $m'(X)$ are binary polynomials satisfying

$$m(-2) \equiv m'(-2) \pmod{2^N + 1},$$

then $m(X) = m'(X)$. We first observe that if $m(X)$ is a binary polynomial, then the largest possible value of $m(-2)$ is

$$1 + (-2)^2 + (-2)^4 + \cdots + (-2)^{N-1} = \frac{2^{N+1} - 1}{3},$$

and the smallest (i.e., most negative) possible value of $m(-2)$ is

$$(-2)^1 + (-2)^3 + (-2)^5 + \cdots + (-2)^{N-2} = -\frac{2^N - 2}{3}.$$

The difference of these two limits is $2^N - 1$, so we see that the congruence class of $m(-2)$ modulo $2^N + 1$ determines exactly the value of $m(-2)$. It remains to show that different binary polynomials have different values at $X = -2$.

Suppose that $m(X)$ and $m'(X)$ are binary polynomials satisfying $m(-2) = m'(-2)$. Let $A(X) = m(X) - m'(X) \neq 0$ and suppose that $A(X) \neq 0$. Then we can write

$$A(X) = aX^k + bX^{k+1} + \cdots + cX^{N-1} \quad \text{with } a \neq 0.$$

We know that every coefficient of $A(X)$ is -1, 0, or 1, so in particular $a = \pm 1$. On the other hand, we also know that $A(-2) = 0$, so

$$0 = A(-2) \equiv a(-2)^k \pmod{2^{k+1}},$$

which implies that 2 divides a. This contradiction shows that $A(X) = 0$, so $m(X) = m'(X)$, which concludes the proof of the proposition. $\square$

Theorem. *Let N be an odd integer and let $a(X) = a_0 + a_1X + \cdots + a_{N-1}X^{N-1} \in \mathbf{Z}[X]$ be a polynomial. The algorithm given below transforms $a(X)$ into a polynomial $b(X) \in \mathbf{Z}[X]$ with the following properties:*

- $\deg b(X) \leq N - 1$.
- *$b(X)$ is a binary polynomial, that is, its coefficients are all 0 or 1 (with one exception noted below).*
- $b(-2) \equiv a(-2) \pmod{2^N + 1}$.

The one exception occurs when

$$a(-2) \equiv \frac{2^{N+1} + 2}{3} \pmod{2^N + 1},$$

in which case one can take $b(X) = 2 + X^2 + X^4 + \cdots + X^{N-1}$. The algorithm terminates after its main loop has been executed no more than $2N + 1$ times.

Algorithm. In the following description, all indices on coefficients of a are treated modulo N. Thus $a_N = a_0$, $a_{N+1} = a_1$, etc.

```
replace each $a_i$ with its congruence class modulo $2^N + 1$.
set $i = 0$.
LOOP:
write $a_i = u + 2v$ with $u = 0$ or 1.
set $a_i = u$, $a_{i+1} = a_{i+1} + v$, $a_{i+2} = a_{i+2} + v$.
set $i = i + 1$.
```

```
    let $w = \min\{\lfloor a_i/2 \rfloor, a_{i+1}\}$.
    set $a_i = a_i - 2w$ and $a_{i+1} = a_{i+1} - w$.
    if $i < N$ then go to LOOP.
    if $a_i \notin \{0,1\}$ then go to LOOP.
    if $a_{i+1} \notin \{0,1\}$ and $i < 2N$ then set $i = i+1$ and go to LOOP.
    if $i < 2N$ then return the polynomial $a(X)$.
    else return the polynomial $2 + X^2 + X^4 + \cdots + X^{N-1}$.
```

Proof. The algorithm has two main operations. In the first operation, three consecutive coefficients α, β, γ of $a(X)$ are altered by the rule

$$\textbf{Rule I}: \quad (\alpha,\beta,\gamma) \longrightarrow (u, \beta+v, \gamma+v), \qquad \text{where } \alpha = u + 2v.$$

In the second operation, two consecutive coefficients α, β are altered by the rule

$$\textbf{Rule II}: \quad (\alpha,\beta) \longrightarrow (\alpha - 2w, \beta - w), \qquad \text{where } w = \min\{\lfloor \alpha/2 \rfloor, \beta\}.$$

We first observe that these rules do not change the value of $a(X)$ at $X = -2$, since for Rule I we have

$$\begin{aligned}\alpha\cdot(-2)^i + \beta\cdot(-2)^{i+1} &+ \gamma\cdot(-2)^{i+2} \\ &= (u+2v)\cdot(-2)^i + \beta\cdot(-2)^{i+1} + \gamma\cdot(-2)^{i+2} \\ &= u\cdot(-2)^i + (\beta+v)\cdot(-2)^{i+1} + (\gamma+v)\cdot(-2)^{i+2}\end{aligned}$$

and for Rule II we have

$$\alpha\cdot(-2)^i + \beta\cdot(-2)^{i+1} = (\alpha-2w)\cdot(-2)^i + (\beta-w)\cdot(-2)^{i+1}.$$

Also note that both rules leave all coefficients positive.

The invariance of the value of $a(-2)$ under the two substitution rules is not quite true if $i = N-2$ or $i = N-1$, since then the indices on some of the coefficients are reduced modulo N. For example, if $i = N-2$, then Rule I changes the value of $a(-2)$ as follows:

$$\begin{aligned}\alpha\cdot(-2)^{N-2} + \beta\cdot(-2)^{N-1} &+ \gamma\cdot(-2)^0 \\ &\to u\cdot(-2)^{N-2} + (\beta+v)\cdot(-2)^{N-1} + (\gamma+v)\cdot(-2)^0 \\ &= \alpha\cdot(-2)^{N-2} + \beta\cdot(-2)^{N-1} + \gamma\cdot(-2)^0 + v(1+2^N).\end{aligned}$$

Thus the value of $a(-2)$ changes, but its congruence class modulo 2^N+1 does not change.

Similarly, if $i = N-1$, then the following formulas show that both rules change the value of $a(-2)$ by a multiple of 2^N+1:

$$\begin{aligned}\alpha\cdot(-2)^{N-1} + \beta\cdot(-2)^0 &+ \gamma\cdot(-2)^1 \\ &\longrightarrow u\cdot(-2)^{N-1} + (\beta+v)\cdot(-2)^0 + (\gamma+v)\cdot(-2)^1 \\ &= \alpha\cdot(-2)^{N-1} + \beta\cdot(-2)^0 + \gamma\cdot(-2)^1 - v(1+2^N).\end{aligned}$$

$$\begin{aligned}\alpha\cdot(-2)^{N-1} + \beta\cdot(-2)^0 &\longrightarrow (\alpha-2w)\cdot(-2)^{N-1} + (\beta-w)\cdot(-2)^0 \\ &= \alpha\cdot(-2)^{N-1} + \beta\cdot(-2)^0 - w(1+2^N).\end{aligned}$$

We have now verified that repeated applications of Rules I and II do not change the value of $a(-2) \bmod 2^N + 1$, and they also leave all coefficients nonnegative. We next study the effect of the rules on the coefficients.

Consider first Rule I as applied to three consecutive coefficients (α, β, γ) of $a(X)$. If $\alpha \in \{0, 1\}$, then Rule I has no effect; and in any case after Rule I is applied, the value of α will be either 0 or 1.

Next consider Rule II as applied to two consecutive coefficients (α, β) of $a(X)$. If $\alpha \in \{0, 1\}$, then Rule II has no effect, since $w = 0$. In any case, after Rule II is applied, we have either $\alpha \in \{0, 1\}$ or else $\beta = 0$, depending on which of $\alpha/2$ and β is smaller.

Now consider what happens on each iteration through the main loop of the algorithm. We first apply Rule I to three consecutive coefficients (a_i, a_{i+1}, a_{i+2}), replacing them with coefficients $(\epsilon_i, a'_{i+1}, a'_{i+2})$ satisfying $\epsilon_i \in \{0, 1\}$, and then we apply Rule II to the pair of consecutive coefficients (a'_{i+1}, a'_{i+2}). There are two possible outcomes. We end up with either

$$(\epsilon_i, a''_{i+1}, 0) \qquad \text{or} \qquad (\epsilon_i, \epsilon_{i+1}, a''_{i+2}),$$

with $\epsilon_i, \epsilon_{i+1} \in \{0, 1\}$. In particular, the i^{th} coefficient of $a(X)$ becomes 0 or 1, and at most the next two coefficients are altered. Thus after the main loop has been iterated N times, every coefficient of $a(X)$ will have been changed to 0 or 1, but the first two coefficients a_0 and a_1 may have been subsequently changed to some other value when we applied the two rules to a_{N-2} and a_{N-1}. However, after the rules have been applied to a_{N-1}, at most one of a_0 and a_1 can be nonbinary.

So after N iterations of the main loop, we are in the situation where at most one a_i is nonbinary. We may also assume that the nonbinary coefficient satisfies $0 \le a_i < 2^N + 1$, since it is always permissible to reduce the coefficients of $a(X)$ modulo $2^N + 1$. (In practice, the nonbinary value will be much smaller than $2^N + 1$ if the original coefficients of $a(X)$ were not too large.) Further, each additional pass through the loop will leave at most one coefficient of $a(X)$ nonbinary

Let (α, β, γ) be consecutive coefficients of $a(X)$ with α the nonbinary coefficient. As noted above, after applying the Rule I we get $(\epsilon, \beta', \gamma')$, and when we then apply Rule II, we get either $(\epsilon, \beta'', 0)$ or $(\epsilon, \epsilon', \gamma'')$, where$\epsilon, \epsilon' \in \{0, 1\}$. We are going to estimate the size of β'' and γ'' and show that they are considerably smaller than α.

The application of Rule I says $\beta' = \beta + v$ and $\gamma' = \gamma + v$, where $\alpha = \epsilon + 2v$. Note that $v \le \alpha/2$. Now suppose that Rule II yields $(\epsilon, \beta'', 0)$. This occurs if $\gamma' \le \beta'/2$ and we have $\beta'' = \beta' - 2\gamma'$. Then

$$\beta'' = \beta' - 2\gamma' \le \beta' = \beta + v \le \beta + \alpha/2 \le 1 + \alpha/2, \quad \text{since } \beta \in \{0, 1\}.$$

Similarly, if Rule II yields $(\epsilon, \epsilon', \gamma'')$, then $\beta'/2 \le \gamma'$, we have $\gamma'' = \gamma' - \beta'$, and

$$\gamma'' = \gamma' - \beta' \le \gamma' = \gamma + v \le \gamma + \alpha/2 \le 1 + \alpha/2, \quad \text{since } \gamma \in \{0, 1\}.$$

To recapitulate, we have shown that after N iterations of the main loop, there is at most one nonbinary coefficient in $a(X)$, call it A. Further, each subsequent iteration of the main loop yields a polynomial with at most one nonbinary coefficient, call it B, satisfying

$$0 \leq B \leq A/2 + 1.$$

We know that the initial nonbinary coefficient is at most 2^N, so it takes at most $N+1$ iterations to ensure that the sole nonbinary coefficient is at most 2. (Notice if $A = 2$, the bound for B is 2.)

Finally, we need to consider what happens if we obtain a polynomial with with one coefficient equal to 2 and the other coefficients all binary. If we at the effect of Rule I on this coefficient and the two subsequent coefficients, we find four possibilities:

$$(2,0,0) \to (0,1,1), \quad (2,0,1) \to (0,1,2), \quad (2,1,0) \to (0,0,0), \quad (2,1,1) \to (0,0,1).$$

Thus $a(X)$ becomes binary unless three consecutive coefficients equal $(2,0,1)$, in which case they are permuted to $(0,1,2)$. Thus repeated application of Rule I will eventually eliminate the 2, unless the coefficients of $a(X)$ following the 2 look like $0,1,0,1,0\dots,0,1$, in which case after at most N iterations $a(X)$ will become the polynomial $2 + X^2 + X^4 + \cdots + X^{N-1}$.

This completes the proof of the theorem except for the assertion that the value of the exceptional polynomial at $X = -2$ satisfies

$$2 + 4 + 4^2 + \cdots + 4^{(N-1)/2} \equiv \frac{2^{N+1}+2}{3} \pmod{2^N + 1}.$$

We leave the proof of this congruence, valid for all odd integers N, as an exercise for the reader. □

Remark. If the coefficients of $a(X)$ are positive and small compared to 2^N, then a more careful analysis of the size of the coefficients shows that after the first N iterations of the algorithm, the sole nonbinary coefficient A satisfies

$$A \leq 2 \max_{0 \leq i < N} a_i.$$

(The 2 can certainly be improved. We leave the proof of this formula to the reader.) Hence except in the exceptional case, the main loop of the algorithm is actually executed no more than

$$N + 1 + \log_2 \max_i a_i$$

times.

6. Low Hamming Weight Products

In this section we briefly describe a method for speeding up the encryption and decryption processes through the use of products of low Hamming weight polynomials. For further details and a discussion of similar constructions for Diffie-Hellman over $\mathrm{GF}(2^n)$ and on Koblitz elliptic curves, see [HS1].

The most time consuming part of NTRU encryption is computation of the product $r(X) * h(X) \bmod q$, and the most time consuming part of NTRU decryption is computation of the product $f(X) * e(X) \bmod q$. The polynomials $h(X)$ and $e(X)$ have coefficients that are more or less randomly distributed modulo q, while one normally take $r(X)$ and $f(X)$ to have binary (i.e., 0 or 1) or trinary (i.e., -1, 0, or 1) coefficients.

For concreteness, suppose that $r(X)$ is a binary polynomial with d ones. Then computation of the product $r(X) * h(X) \bmod q$ requires approximately dN operations, where one operation is an addition and a remainder modulo q. We further require that the set of $r(X)$ polynomials be sufficiently large so that an attacker cannot find $r(X)$ by an exhaustive search, and there is also a lattice norm requirement on the number of nonzero coefficients in $r(X)$. The search space for $r(X)$ has essentially $\binom{N-1}{d-1}$ elements, since the rotation $X^i r(X)$ is really equivalent to $r(X)$. For information about the lattice norm requirement, see [HPS,S1,S2].

It is possible to significantly reduce the computational requirements by taking $r(X)$ to be a product of polynomials with fewer ones. Thus suppose that we write $r(X) = r_1(X)r_2(X)$, where r_1 and r_2 are binary polynomials with d_1 and d_2 ones respectively. Then $r(X)$ will have approximately d_1d_2 ones. (In practice, it will have have a few twos and an occasional three mixed in with the ones, but that will not affect matters very much.) Notice that the computation of the product

$$r(X) * h(X) = r_1(X) * \big(r_2(X) * h(X)\big)$$

requires only $(d_1 + d_2)N$ operations, so the computational complexity is proportional to the sum of d_1 and d_2. On the other hand, the search space for the pair of polynomials (r_1, r_2) has size approximately $\binom{N-1}{d_1-1}\binom{N-1}{d_2-1}$, so is proportional to the product of the r_1 search space and the r_2 search space. (In practice, there may be a meet-in-the-middle search that reduces the size of the search space, see [HS1] for details.) Further, the number of nonzero coefficients in $r_1(X)r_2(X)$ is essentially the product d_1d_2. Thus one might say that using a product $r = r_1r_2$ requires computation proportional to the sum $d_1 + d_2$ while giving security proportional to the product d_1d_2. In rough terms, this explains why one obtains significant performance gains without changing the level of security.

Similar remarks apply to the private key polynomial $f(X)$; or, if $f(X)$ is taken in the form $f(X) = 1 + p * f'(X)$, to the polynomial $f'(X)$. For example, one might take $f(X)$ to have the form

$$f(X) = 1 + p * (f_1(X) * f_2(X) + f_3(X)),$$

where f_1, f_2, f_3 are quite sparse binary polynomials. Further, to help reduce decryption failures caused by coefficients in the polynomial (1) spreading further than q, one might further choose f_1, f_2, f_3 so that the polynomial $f_1 * f_2 + f_3$ is purely binary, that is, has only 0 and 1 coefficients. In principle this will cut down somewhat on the size of the search space, but in practice there does not appear to be any way to determine if a triple (f_1, f_2, f_3) is in the desired space without actually computing at least part of the quantity $f_1f_2+f_3$. Thus the effective search space remains the collection of all triples (f_1, f_2, f_3) having the specified number of nonzero coefficients.

Acknowledgements. The authors would like to thank Jill Pipher for her assistance, Eliane Jaulmes and Antoine Joux for their work on adaptive attacks against NTRU, E. Fujisaki and T. Okamoto for correspondence related to their work on protection against general adaptive attacks, and Rafal Nessel for his suggestions and corrections to the original version of this paper.

References

[FO1] E. Fujisaki, T. Okamoto, Secure integration of asymmetric and symmetric encryption schemes, *Advances in Cryptology—CRYPTO '99*, Lecture Notes in Computer Science 1666, Springer-Verlag, 1999, 537–554

[FO2] E. Fujisaki, T. Okamoto, How to Enhance the Security of Public-Key Encryption at Minimum Cost, IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, Vol.E83-A, No.1, Special Issue on Cryptography and Information Security (January 2000)

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

[HS1] J. Hoffstein, J.H. Silverman, Small Hamming Weight Products in Cryptography, preprint, September 2000.

[HS2] J. Hoffstein, J.H. Silverman, Reaction Attacks Against the NTRU Public Key Cryptosystem, NTRU Technical Report #015, August 1999, `www.ntru.com`

[HS3] J. Hoffstein, J.H. Silverman, Protecting NTRU Against Chosen Ciphertext and Reaction Attacks, NTRU Technical Report #016, June 2000, `www.ntru.com`

[JJ] E. Jaulmes, A. Joux, A chosen-ciphertext attack against NTRU, in *Proceedings of CRYPTO 2000*, Lecture Notes in Computer Science, Springer-Verlag.

[S1] J.H. Silverman, Estimated Breaking Times for NTRU Lattices, NTRU Technical Report #012, `<www.ntru.com>`.

[S2] J.H. Silverman, Dimension-Reduced Lattices, Zero-Forced Lattices, and NTRU Public Key Cryptosystem, NTRU Technical Report #013, `<www.ntru.com>`.

NTRU Cryptosystems, Inc., 5 Burlington Woods, Burlington, MA 01803, USA.
`jhoffstein@ntru.com, jsilverman@ntru.com`