NTRU Enhancements 1

This part of the tutorial describes some of the special techniques which can be used to speed up NTRU operations. Before reading this tutorial, you should have read the introductory tutorial page.

The final tutorial in this series explains another enhancement to NTRU in which we choose our small vectors in a slightly different way, permitting even greater speed improvements

1. REVIEW

The NTRU Cryptosystem is parameterized by three values, *N*, *p* and *q*. All objects are univariate polynomials of degree *N*, which are multiplied using the convolution product rule. *p* and *q* are moduli; multiplications and additions are generally followed by reduction mod *p* or mod *q*. We use the following notation:

f	A small polynomial, part of the private key.
f_p	The inverse of f mod *p*.
f_q	The inverse of f mod *q*.
g	A small polynomial, used in generating the public key.
h	The public key. $h = f_q * g \bmod q$.
m	The message, a small polynomial.
r	The random blinding value, used when encrypting. A small polynomial.
e	The encrypted message.
a	The partially decrypted message. $a = f * e \bmod q$.

2. CHOOSING THE FORM OF *F*

As the discussion in Section 2 of the introductory tutorial makes clear, **f** must have the following properties:

1. **f** is invertible mod *p*.
2. **f** is invertible mod *q*.
3. **f** is small.

In the previous examples, we guaranteed that **f** was small by use of the d_f parameter. We guaranteed that it was invertible mod *p* and *q* because, during the key generation process, we threw f away if the inverse didn't exist.

In commercial applications, we use an alternative way of choosing **f**. We take

$$\mathbf{f} = 1 + p\mathbf{F},$$

where **F** is a small polynomial. This choice means that **f** is equal to 1 mod *p*, which has the following advantages:

- **f** is always invertible mod *p* (in fact, $\mathbf{f}^{-1} = 1 \bmod p$). This speeds up key generation, because we don't have to explicitly calculate the inverse mod *p*.
- Because $\mathbf{f}^{-1} = 1 \bmod p$, we no longer have to carry out the second polynomial multiplication when decrypting. This speeds up decryption considerably, as it now only requires one multiplication, not two. It also means that we don't have to store $\mathbf{f}_p = \mathbf{f}^{-1} \bmod p$ as part of the private key.

3. TAKING *P* = 2 + X

All of the small polynomials that we've described to date have coefficients which are small (since we always choose *p* to be small). The success of decryption depends on the coefficients of **a** being unchanged when they're reduced modulo *q*. Clearly, the smaller the coefficients of **f**, **g**, **m**, and **r** are, the smaller the coefficients of **a** will be in general (see Section 5 of the introductory tutorial for a reminder of why this is true). So if we can reduce the size of *p*, we make it easier to pick parameter sets such that decryption will succeed.

We can't pick *p* to be 2, because *p* and *q* must be relatively prime, but there is nothing that requires *p* to be an integer. All that is needed is for *p* and *q* to be relatively prime in the ring **R**. (This is the same as saying that the three element X^N-1, *p*, *q* generate the unit ideal in the ring **Z**[X].) Thus we may take *p* to be a small polynomial (so from now on we denote it by **p**), such as

$$\mathbf{p} = 2 + X.$$

When **p** is chosen in this form, it is more natural to use binary polynomials (with coefficients 0, 1) instead of the trinary ones (with coefficients +1, -1, 0) that we use if *p* = 3. This makes encoding messages for encryption much simpler -- instead of converting them from a standard binary encoding to a trinary encoding, we can simply use the ordinary, binary form. On the other hand, it makes it a little bit more complicated to recover the message polynomial **m** from its value mod **p**.

For more details on what it means to reduce a polynomial mod 2+X in this context, see the references. For the moment, we'll simply state that, given a polynomial **d** of degree *N* with coefficients mod *q*, it is (almost) always possible to find a polynomial **m** of degree *N* or less, with *binary* coefficients, such that

$$\mathbf{d}(-2) = \mathbf{m}(-2) \pmod{2^N + 1}.$$

This polynomial **m** is what we mean when we refer to "**d** reduced mod *q*"

4. CENTERING THE POLYNOMIAL A

Taking **m** to be binary has a further consequence, which means that we have to take a little more care when decrypting. Recall that when he decrypts, Bob is calcuating the following value:

a = f*e (modulo *q*)	
= f*(r*h + m) (modulo *q*)	[since e = r*h + m (modulo *q*)]
= $f*(r*p*f_q*g + m)$ (modulo *q*)	[since $h = p*f_q*g$ (modulo *q*)]
= p*r*g + f*m (modulo *q*)	[since $f*f_q = 1$ (modulo *q*)]

Using the parameters given in the introductory tutorial, **r**, **g** and **m** are centered around zero (in that they have equal numbers of +1s and -1s), and **f** is nearly centered around zero (in that **f** had one more +1 than -1). Another way of putting this is to say that

$$\mathbf{r}(1) = \mathbf{g}(1) = \mathbf{m}(1) = 0;$$
$$\mathbf{f}(1) = 1.$$

(Remember that **P** (1), for any polynomial **P**, is simply the sum of the coefficients of **P**.)

Decryption works because all the coefficients of **p*****r*****g** + **f*****m** naturally lie within the range (-*q*/2, *q*/2]. If we define reducing mod *q* as reducing into this range, we leave all the coefficients unchanged.

Now that we're using binary polynomials (and taking **f** to have the form 1 + **p*****F**), the values of **r**, **g**, **m** and **f**are no longer centered at zero. The polynomials involved are still small, and all their coefficients should still lie within *q* of each other, but they won't lie within the specific range (-*q*/2, *q*/2].

Why does this matter? For the sake of argument, let's take **p** = 3, *q* = 32. Say that one of the coefficients of**p*****r*****g** + **f*****m** has the value 18. When we reduce this mod **p**, we should get 0. But if we're taking reduction mod *q* to be reduction into the range (-15, 16), we will replace the value 18 with -14 before reducing mod **p**. On reduction, we get -14 mod **p** = 1, which is the wrong answer. (This is just another way of saying that, because **p** and *q* are relatively prime, for any value *a*,

a mod **p** ? *a* + *q* mod **p**).

So before we can carry out the reduction modulo **p** when we're decrypting, we have to work out what the true range of the coefficients of **p*****r*****g** + **f*****m** is likely to be. Of course, we don't know how many 1s and 0s there are in **m** before we decrypt it; but we can extract it from a using the following method.

1. Set *I* = $\mathbf{f}_q$ (1) . (**a** (1) - **p** (1) . **r** (1) . **g** (1)) (mod *q*)
2. Set *Avg* = (**p** (1) . **r** (1) . **g** (1) + *I* **f** (1)) / *N*
3. The expected range of the coefficients will be between *Avg* - *q*/2 and *Avg* + *q*/2. *Avg* will generally not be an integer, so the actual expected range will be the *q* integers that lie between *Avg* - *q*/2 and*Avg* + *q*/2.

By reducing the coefficients of **f** * **e** into this range, we can be confident that the reduction mod **p** will proceed correctly.

5. ADVANCED TOPICS EXAMPLE 1

Parameters

In the following sections we'll work through an example of the NTRU cryptosystem using two of the advanced techniques described above. We will use the following values for N, *q* and **p**:

N *q* **p**

Small Illustration Parameters11322+X

and take **f** to have the form **f** = 1 + **p*****F**.

Key Generation

Bob wants to generate an NTRU keypair following the basic principles of NTRU, but using the efficiency improvements outlined above.

To generate a key, Bob first generates a small binary vector **F**. We need to specify that **F** is "small" using the quantity d_F:

- **F** has d_F of its coefficients equal to 1; all of the rest of its coefficients are equal to 0.

Here we take $d_F = 4$.

Bob chooses a polynomial **F** with four 1's. Suppose he chooses

$$\mathbf{F} = 1 + X^4 + X^7 + X^9.$$

He now calculates the private polynomial **f** = 1 + (2 + X) * **F**. This gives him:

$$\mathbf{f} = 3 + X + 2X^4 + X^5 + 2X^7 + X^8 + 2X^9 + X^{10}.$$

Bob stores **f** as his private key.

Now Bob must calculate his public key. First he calculates $\mathbf{f}_q$, the inverse of **f** mod *q*. This turns out to be:

$$\mathbf{f}_q = -7 - 5X + 12X^2 - 3X^3 - 2X^4 + 6X^5 + 13X^6 - 10X^7 - 8X^8 - 8X^9 - 15X^{10}.$$

Next, he generates **g**, which is another small random polynomial. In this case, we'll make sure **g** is small by requiring it to have d_g coefficients equal to 1, and setting the other coefficients to zero. For purposes of this tutorial, we'll take d_g to be 5. Bob generates a **g** with five 1's and six 0's. Let's say he gets:

$$\mathbf{g} = 1 + X + X^4 + X^6 + X^{10}.$$

Finally, Bob generates the public key **h** using the formula

$$\mathbf{h} = \mathbf{p} * \mathbf{f}_q * \mathbf{g}.$$

This gives him

$$\mathbf{h} = 15 + 11X + 9X^2 - 14X^3 - 12X^4 + 12X^5 - 7X^6 - 12X^7 - 13X^8 - 8X^9 - 2X^{10}.$$

Bob makes **h** publicly available as his public key.

Encryption

Now Alice wants to encrypt a message for Bob using the NTRU cryptosystem. To review, she uses the message **m**, a randomly chosen small polynomial **r**, and Bob's public key **h** to compute the polynomial

$$\mathbf{e} = \mathbf{r}*\mathbf{h} + \mathbf{m} \text{ (modulo } q).$$

Let's assume she wants to encrypt the binary message 01010100111. This converts to the binary polynomial

$$\mathbf{m} = X + X^3 + X^5 + X^8 + X^9 + X^{10}.$$

She generates a small binary polynomial **r**. We'll make sure **r** is small by requiring it to have d_r coefficients equal to 1, and setting the other coefficients to zero. For purposes of this tutorial, we'll take d_r to be 4. Alice generates a **r** with four 1's and seven 0's. Let's say she gets:

$$\mathbf{r} = 1 + X^3 + X^4 + X^8.$$

Now she calculates the encrypted message **e**.

$$e = r*h + m \ (\text{modulo } q)$$
$$= 8 + 11X + 11X^2 - 7X^3 + 2X^4 - 12X^5 + 12X^6 - 8X^7 + 3X^8 + 9X^9 - 11X^{10}.$$

Alice transmits this message to Bob.

Decryption

Bob has received the message

$$\mathbf{e} = 8 + 11X + 11X^2 - 7X^3 + 2X^4 - 12X^5 + 12X^6 - 8X^7 + 3X^8 + 9X^9 - 11X^{10}$$

from Alice. He wants to decrypt this message using NTRU decryption. To review, he will do this by using his private key **f** to obtain

$$\mathbf{a} = \mathbf{f} * \mathbf{e} \bmod q$$

and then reducing the result mod **p** to obtain **d**, the decrypted ciphertext, which should be equal to **m**. (Previously, Bob would also have had to multiply **d** by the inverse of **f** mod **p**, but we have chosen **f** so that its inverse mod **p** is equal to 1. This final step therefore becomes trivial multiplication by 1.)

So, first Bob calculates **a** = **f** * **e** mod q. This gives him

$$\mathbf{a} = 7 + 14X + 10X^2 + 15X^3 + 14X^4 + 13X^5 + 10X^6 + 11X^7 + 15X^8 + 14X^9 + 15X^{10}.$$

Ordinarily, Bob should calculate the centering value of **a**. In this case, we'll omit this step; the values are clustered so tightly in the range 7 to 15 that no recentering is necessary. The next example shows a case where we need to calculate the centering value to get the correct decrypted message.

Now he reduces mod **p**, where **p** = 2 + X. In other words, he finds a binary polynomial **d** which has the property that

$$\mathbf{d}(-2) = \mathbf{a}(-2) \pmod{2^N + 1}.$$

This polynomial is

$$\mathbf{d} = X + X^3 + X^5 + X^8 + X^9 + X^{10}.$$

This can easily be confirmed, as follows:

- **a** (-2) = 10971
- $2^N + 1 = 2049$
- **a** (-2) mod (2^N + 1) = 726
- **d** (-2) = 726

Finally, Bob converts the polynomial **d** into the binary message 01010100111. This is the message Alice sent to him, which he has therefore successfully decrypted.

6. ADVANCED TOPICS EXAMPLE 2

In this example, we'll show a case where the choice of centering value makes a difference to the success of decryption. We use the usual parameters:

N q **p**
11322+X

To save space, we'll represent the polynomials in vector form. So instead of writing

$$X + X^3 + X^5 + X^8 + X^9 + X^{10}.$$

we'll write

[0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1].

Key Generation

Bob generates a keypair using the same values of d_F, d_g as in the previous example. He obtains:

F = [1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0]
f = [3, 1, 0, 0, 2, 1, 2, 3, 1, 0, 0]
$\mathbf{f}_q$= [14, 4, -1, -5, 10, 9, 6, 13, 4, 3, 12]
g = [0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0]
h = [16, 9, -3, 8, -2, -1, 16, 4, -4, 8, -8]

Bob stores **f** as his private key, and makes **h** publicly available as his public key.

Encryption

Now Alice wants to encrypt the message **m** for Bob. She picks a random small polynomial **r** using the value of d_r from the previous example, and calculates

r***h** + **m** (modulo *q*).

The values she gets are:

m = [0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1]
r = [0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0]
e= [-12, 9, -2, 6, 13, -1, 4, -11, -5, -3, -12]

Alice transmits the encrypted message **e** to Bob.

Decryption

Bob has received the message

e= [-12, 9, -2, 6, 13, -1, 4, -11, -5, -3, -12]

from Alice. He decrypts by calculating

a = **f** * **e** mod *q*

and then reducing the result mod **p** to obtain **d**. The value of **a** that he calculates is:

a (before reduction) = [-23, 12, -19, -50, -23, -22, -47, -49, 17, 12, 10]

a (mod q) = [9, 12, 13, 14, 9, 10, -15, 15, -15, 12, 10]

But when he reduces it mod **p**, he gets

m' = [0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0]

instead of the real message Alice sent, which was

m = [0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1]

This is because the partially decrypted message **a** is incorrectly centered. (In fact, this is obvious on inspection -- all of the coefficients of **a** lie in the range 9 to 15, except for two coefficients which have the value -15.)

So Bob must calculate the recentering value and recenter **a**. He uses the method given above to calculate *I*,

$$
\begin{aligned}
I &= \mathbf{f}_q(1) \,.\, (\mathbf{a}(1) - \mathbf{p}(1) \,.\, \mathbf{r}(1) \,.\, \mathbf{g}(1)). \\
&= 69 \,.\, (74 - 3 \,.\, 4 \,.\, 5) \\
&= 69 \,.\, 14 \\
&= 6 \pmod{32}
\end{aligned}
$$

(remember that **a** (1) is just the sum of the coefficients of **a**, so that **r** (1) and **g** (1) are simply d_r and $d_r$respectively). now Bob calculates *Avg*, obtaining

$$
\begin{aligned}
Avg &= (\, \mathbf{p}(1) \,.\, \mathbf{r}(1) \,.\, \mathbf{g}(1) + I\, \mathbf{f}(1)) / N \\
&= (\, 3 \,.\, 4 \,.\, 5 + 6 \,.\, 13) / 11 \\
&= (60 + 78) / 11 \\
&= 12.5454...
\end{aligned}
$$

Bob moves the coefficients of **a** to fall in the range (*Avg* - 16, *Avg* + 16) = (-3, 28). This converts **a** to

a (mod q) = [9, 12, 13, 14, 9, 10, 17, 15, 17, 12, 10]

On reducing this mod **p**, he obtains the correct message

m = [0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1].

Next Steps

The final tutorial in this series explains another enhancement to NTRU in which we choose our small vectors in a slightly different way, permitting even greater speed improvements.

FURTHER READING

A complete description of the NTRU Public Key Cryptosystem with full technical details is given in the paper

NTRU: A Ring Based Public Key Cryptosystem, Jeffrey Hoffstein, Jill Pipher, Joseph H. Silverman, in *Algorithmic Number Theory* (ANTS III), Portland, OR, June 1998, J.P. Buhler (ed.), Lecture Notes in Computer Science 1423, Springer-Verlag, Berlin, 1998, 267-288.

Further enhancements to NTRU, including the use of **f**=1+**p*****F** and **p**=X+2, are described in *Optimizations for NTRU*,

J. Hoffstein, J. Silverman, Public-Key Cryptography and Computational Number Theory (Warsaw, September 11-15, 2000), DeGruyter, to appear.

A description of how to speed up NTRU and other cryptosystems by the use of quantities with small Hamming weight can be found in *Random Small Hamming Weight Products with Applications to Cryptography*, J. Hoffstein, J. Silverman, Com2MaC Workshop on Cryptography (Pohang, Korea, June 2000), Discrete Mathematics, to appear.

These papers and short notes giving further information may be downloaded in a variety of formats from the Technical Center.

The following are some additional sources to learn about algebra, number theory, algorithms, and cryptography.

- *A Course in Computational Algebraic Number Theory*, H. Cohen, GTM 138, Springer-Verlag, Berlin, 1993.
- *A Friendly Introduction to Number Theory*, J.H. Silverman, Prentice-Hall, New Jersey, 1997.
- *Cryptography: Theory and Practice*, D. Stinson, CRC Press, Boca Raton, 1995.
- *Handbook of Cryptography*, S. Vanstone, P. Van Oorschot, A. Menezes, CRC Press, Boca Raton, 1996.