NTRU Enhancements 2

This part of the tutorial describes advanced enhancements to the implementation of the NTRU algorithm. Before reading this tutorial, you should have read the introductory tutorial page. The advanced tutorial page discusses other ways of speeding up NTRU operations, independent of the techniques described on this page.

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.

The most time-consuming operations in the NTRU cryptosystem are the convolution multiplications. This tutorial describes ways to speed up those multiplications.

2. LOW HAMMING WEIGHT POLYNOMIALS

Many of the polynomials used in the calculations are "small" polynomials, which is to say that they're small relative to a polynomial whose coefficients are chosen randomly mod q. The cryptosystem gives us a lot of flexibility in how we generate these small polynomials. In the advanced tutorial we discuss one way in which the form of **f**, the private key, can be changed without affecting the central operations of the system. In this tutorial, we'll describe another way of choosing small polynomials that offers considerable performance advantages.

In the advanced tutorial, when we generate keys, we choose **F** to have d_F +1s and set the other coefficients to zero. (**F** is just an example: in general, whenever we choose a small polynomial, we do it by specifying the number of 1s and setting all the other coefficients to zero). Let's have a look at how multiplication of an arbitrary polynomial by a small polynomial works. 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].

Convolution Multiplication Example

First, let's look at a small example of convolution multiplication for polynomials of degree 3. The convolution product

[4, 5, 7] * [5, 3, 2]

is

```
  4 . [5, 3, 2]
 +5 . [2, 5, 3]
 +7 . [3, 2, 5]

=  [20, 12, 8]
 +[10, 25, 15]
 +[21, 14, 35]

=  [20+10+21, 12+25+14, 8+15+35]

=  [51, 51, 58].
```

© Copyright 2014. Security Innovation, Inc. All rights reserved.

See the Algebra tutorial for more details. For the purposes of these examples, we aren't carrying out any modular reductions.

So this general multiplication of two cubic polynomials requires nine multiplies and six adds. Now let's look at what happens if one of the polynomials has only 1s and 0s:

```
   [1, 0, 0, 1, 0] * [5, 3, 2, 9, 10]

=  1 * [5, 3, 2, 9, 10]
  +0 * [10, 5, 3, 2, 9]
  +0 * [9, 10, 5, 3, 2]
  +1 * [2, 9, 10, 5, 3]
  +0 * [3, 2, 9, 10, 5]

=  1 * [5, 3, 2, 9, 10]
  +1 * [2, 9, 10, 5, 3]

=  [5+2, 3+9, 2+10, 9+5, 10+3]

=  [7, 12, 12, 14, 13]
```

Now the convolution requires no multiplications at all, and each coefficient in the result is simply the sum of two numbers. In general, we can say that if **i** is a small polynomial with d_i coefficients equal to 1 and the rest equal to zero, and if **a** is an arbitrary polynomial, then each coefficient of the product **i*a** is obtained by summing d_i of the coefficients of **a**.

What this discussion has told us so far is that if we know that a polynomial has only 1 and 0 coefficients, we can multiply it by any other polynomial in much less time than an ordinary convolution multiplication. But is there a way to speed up multiplications even more? It turns out there is, and the rest of this tutorial explains how.

3. PRODUCTS OF SMALL HAMMING WEIGHT POLYNOMIALS

First, we need to discuss the nature of the "small" polynomials that are used in NTRU (this is discussed in more detail in the Preliminary Security Analysis page). As the outline description of NTRU in the introductory tutorial makes clear, successful decryption depends on the message, the private key, and the blinding value all being small. However, there are security risks with having the polynomials be too small.

Consider an attacker who knows the public key **h = fq * g**. and wants to find the private key **f**.

- If **f** has too few non-zero entries, it's possible for an attacker to simply try all possible **f**s in turn. If he finds an **f** such that **f * h** (mod *q*) is small, he can decrypt. So the search space for the small polynomials must be big enough to prevent these "brute force" attacks.
- Because **h** is constructed from two small objects, it's possible to mount what's known as a "short lattice vector" attack on **h**, in order to recover **f** and **g**. This short lattice vector attack is the best known way of attacking NTRU, in the same way as factorizing the modulus is the best known way of attacking the RSA algorithm. Lattice reduction algorithms, which are the core of this attack, are very well studied. Their running times depend on (among other things):

© Copyright 2014. Security Innovation, Inc. All rights reserved.

1. The dimension of the lattice -- the bigger the dimension, the longer the attack takes.
2. How short the shortest vector is -- the shorter the vector, the easier it is to find.

If **f** had a very small number of 1s in it, it would be very easy for the lattice reduction algorithm to find. So **f** (and all the other small polynomials in the system) have to be long enough to prevent this attack from working.

This means that NTRU private keys have to be small polynomials, but not *too* small. So we can't speed up multiplying simply by taking the number of 1s in each small polynomial to be as small as possible.

Fortunately, there's a way we can get the performance benefit of having a small number of 1s without introducing any security vulnerabilities. The idea is very simple:

Instead of taking **f** to be a single small polynomial, we form it by combining several even smaller polynomials.

Here's an example. In this example, we'll take *N* = 13, and we'll assume that we want our small polynomials to have 9 non-zero entries. (In full-size versions of the cryptosystem, with *N* = 251, we usually take our small polynomials so that about one third of the coefficients are non-zero). Consider the polynomial

$$
\begin{aligned}
i &= i_1 * i_2. \\
&= [1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0] * [1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0] \\
&= [1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0]
\end{aligned}
$$

i has 9 1s, so to calculate **i*a** would take 9 additions for every coefficient in the result. But if we instead use the two components $\mathbf{i}_1$ and $\mathbf{i}_2$, we don't have to calculate **i*a** directly. Instead we can calculate first $\mathbf{i}_2 * \mathbf{a}$, which takes three additions per coefficient, and then $\mathbf{i}_1 * (\mathbf{i}_2 * \mathbf{a})$, which takes another three. So we have calculated **i*a** for a total cost of 6 adds per coefficient, instead of the previous 9 adds, making multiplications take 2/3 as long.

4. SMALL POLYNOMIALS IN NTRU

In NTRU, many quantities are small polynomials, and there are many occasions when we multiply an arbitrary polynomial by a small polynomial. To review, we have the following small polynomials:

F	A small polynomial, part of the private key.
f	The private key: f = 1 + p*F
g	A small polynomial, used in generating the public key.
m	The message, a small polynomial.
r	The random blinding value, used when encrypting. A small polynomial.

And we perform the following convolutions:

- Public key $\mathbf{h} = \mathbf{p} * \mathbf{f}_q * \mathbf{g}$.
- Encrypted message **e = r*h + m** (modulo *q*).
- Partially decrypted message **a = f * e** mod *q*.

For commercial applications of NTRU, *N*=251. To avoid the short lattice vector attacks described above, we want our short vectors to have about 72 non-zero coefficients. We take advantage of the speedups as follows:

Instead of picking

www.securityinnovation.com
© Copyright 2014. Security Innovation, Inc. All rights reserved.

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

with $d_F = 72$, we pick

$\mathbf{f} = 1 + \mathbf{p}*((\mathbf{f}_1 * \mathbf{f}_2) + \mathbf{f}_3)$

with $d_{f1} = 8$, $d_{f2} = 8$, $d_{f3} = 8$. Multiplication by **F** now takes 24 additions per coefficient, not 72.

Instead of picking

r = single small polynomial

with $d_r = 72$, we pick

$\mathbf{r} = (\mathbf{r}_1 * \mathbf{r}_2) + \mathbf{r}_3$

with $d_{r1} = 8$, $d_{r2} = 8$, $d_{r3} = 8$. Multiplication by **r** now takes 24 additions per coefficient, not 72.

We don't alter the form of **g**, as it is only used during key generation.

So these small changes speed up NTRU's convolutions by a factor of 3.

5. SECURITY CONSIDERATIONS

This section briefly considers the security implications of using low Hamming weight polynomials.

In the previous section we mentioned that, to prevent short lattice vector attacks, all small polynomials should have about 72 non-zero coefficients for N=251. The polynomials that we constructed are this size. So this method of forming the small polynomials does not introduce extra vulnerability to lattice attacks.

However, we need to make sure that the brute force search is still impractical. If $N = 251$, and each $\mathbf{f}_i$ has 8 1s, then the number of possible $\mathbf{f}_s$ is

(251 choose 8)^3.

This is an upper bound on how difficult the brute force search is. In fact, an attacker can use what's known as a *meet-in-the-middle* technique to increase the speed of the attack at the expense of using more memory. This attack, related to the one described in Technical Center Note 4 and described in more detail in the paper on small Hamming weight polynomials, is the best direct search-type attack known against polynomials of this form. For N = 251, with polynomials of the form given, it requires the attacker to carry out approximately $2^{88.65}$ operations.

6. 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

© Copyright 2014. Security Innovation, Inc. All rights reserved.

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 NTRU 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.

www.securityinnovation.com
© Copyright 2014. Security Innovation, Inc. All rights reserved.