NTRU Cryptosystems Technical Report

Report # 014, Version 1
Title: Almost Inverses and Fast NTRU Key Creation
Author: Joseph H. Silverman
Release Date: March 15, 1999

Abstract. We explain how to use the "Almost Inverse Algorithm" of Schroeppel, Orman, O'Malley, and Spatscheck [1] to efficiently compute NTRU public/private key pairs.

Let $m(X)$ be a polynomial in $(\mathbf{Z}/2\mathbf{Z})[X]$. The "Almost Inverse Algorithm" of Schroeppel, Orman, O'Malley, and Spatscheck [1] gives an efficient way to compute the inverse of the polynomial $a(X)$ in the ring $(\mathbf{Z}/2\mathbf{Z})[X]/(m(X))$ provided that $\gcd(a(X), m(X)) = 1$ and $m(0) = 1$. Here is how the almost inverse algorithm works for the polynomial $m(X) = X^N - 1$ used by the NTRU Public Key Cryptosystem.

Inversion in $(\mathbf{Z}/2\mathbf{Z})[X]/(X^N - 1)$

```
Input:   a(X)
Output:  b(X) ≡ a(X)^(-1) in (Z/2Z)[X]/(X^N - 1)
Step 1:  Initialization:  k := 0, b(X) := 1, c(X) := 0,
                          f(X) := a(X), g(X) := X^N - 1
Step 2:  Loop:
Step 3:  do while f_0 = 0
Step 4:      f(X) := f(X)/X, c(X) := c(X) * X, k := k + 1
Step 5:  if f(X) = 1 then return X^(N-k) b(X)  (mod X^N - 1)
Step 6:  if deg(f) < deg(g) then
Step 7:      exchange f and g and exchange b and c
Step 8:  f(X) := f(X) + g(X)  (mod 2)
Step 9:  b(X) := b(X) + c(X)  (mod 2)
Step 10: goto Loop
```

Note that the number f_0 in Step 3 is the constant coefficient of f, and that the return value $X^{N-k}b(X) \pmod{X^N - 1}$ in Step 4 is simply $b(X)$ with its coefficients cyclically shifted k places. We also note that the speed of the Inversion Procedure can be significantly enhanced by a number of implementation tricks, such as expanding the operations on b, c, f, g into inline loop-unrolled code. We refer the reader to [1] for a list of practical suggestions.

In order to create NTRU public/private key pairs, one needs to compute the inverse of a polynomial modulo p for primes other than 2. Here is an adaptation of the almost inverse algorithm for the prime $p = 3$, since this is the other value required for the standard NTRU parameter sets. (At the end of this note we will give a version for arbitrary primes.)

Inversion in $(\mathbf{Z}/3\mathbf{Z})[X]/(X^N - 1)$

```
Input:    a(X)
Output:   b(X) ≡ a(X)^{-1} in (Z/3Z)[X]/(X^N − 1)
Step 1:   Initialization:  k := 0, b(X) := 1, c(X) := 0,
                           f(X) := a(X), g(X) := X^N − 1
Step 2:   Loop:
Step 3:   do while f_0 = 0
Step 4:       f(X) := f(X)/X, c(X) := c(X) * X, k := k + 1
Step 5:   if f(X) = ±1 then return ±X^{N−k} b(X)  (mod X^N − 1)
Step 6:   if deg(f) < deg(g) then
Step 7:       exchange f and g and exchange b and c
Step 8:   if f_0 = g_0
Step 9:       f(X) := f(X) − g(X)  (mod 3)
Step 10:      b(X) := b(X) − c(X)  (mod 3)
Step 11:  else
Step 12:      f(X) := f(X) + g(X)  (mod 3)
Step 13:      b(X) := b(X) + c(X)  (mod 3)
Step 14:  goto Loop
```

In this routine, all computations are done modulo 3, so all coefficients are chosen from the set $\{-1, 0, 1\}$. Also, the two ± 1's in Step 5 are chosen to have the same sign.

The creation of NTRU public/private key pairs often requires finding the inverse of a polynomial $f(X)$ modulo not only a prime, but also a prime power, in particular a power of 2. However, once an inverse is determined modulo a prime p, a simple method based on Newton iteration allows one to rapidly compute the inverse modulo powers p^r. The following algorithm converges doubly exponentially, in the sense that it requires only about $\log_2(r)$ steps to find the inverse of $a(X)$ modulo p^r, once one knows an inverse modulo p.

Inversion in $(\mathbf{Z}/p^r\mathbf{Z})[X]/(X^N - 1)$

```
Input:    a(X), p (a prime), r
          b(X) ≡ a(X)^{-1}  (mod p)
Output:   b(X) ≡ a(X)^{-1}  (mod p^r)
Step 1:   q = p
Step 2:   do while q < p^r
Step 3:       q = q^2
Step 4:       b(X) := b(X)(2 − a(X)b(X))  (mod q)
```

Finally, in the interest of completeness, we give a version of the almost inverse algorithm for an arbitrary prime p.

Inversion in $(\mathbf{Z}/p\mathbf{Z})[X]/(X^N - 1)$

```
Input:    a(X), p (a prime)
Output:   b(X) ≡ a(X)^{-1} in (Z/pZ)[X]/(X^N - 1)
Step 1:   Initialization:  k := 0, b(X) := 1, c(X) := 0,
                           f(X) := a(X), g(X) := X^N - 1
Step 2:   Loop:
Step 3:   do while f_0 = 0
Step 4:        f(X) := f(X)/X, c(X) := c(X) * X, k := k + 1
Step 5:   if deg(f) = 0 then
Step 6:        b(X) := f_0^{-1} b(X)   (mod p)
Step 7:        return X^{N-k} b(X)   (mod X^N - 1)
Step 8:   if deg(f) < deg(g) then
Step 9:        exchange f and g and exchange b and c
Step 10:  u := f_0 g_0^{-1}   (mod p)
Step 11:  f(X) := f(X) - u * g(X)   (mod p)
Step 12:  b(X) := b(X) - u * c(X)   (mod p)
Step 13:  goto Loop
```

Why It Works

Since no explanation is given in [1], we briefly explain why the "almost inverse algorithm" works. The idea is that one starts with the vector $(f, g) = (a, m)$. One then multiplies (on the right) by the following matrices:

$$A = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \qquad B = \begin{pmatrix} X^{-1} & 0 \\ 0 & 1 \end{pmatrix}, \qquad C_u = \begin{pmatrix} 1 & 0 \\ -u & 1 \end{pmatrix}.$$

Note that the effect of these transformations is

$$(f, g)A = (g, f), \qquad (f, g)B = (X^{-1}f, g), \qquad (f, g)C_u = (f - ug, g).$$

So Step 4 is the matrix B, Step 9 is the matrix A, and Step 11 is the matrix C_u. Note that in Step 11, the value of u is chosen so that $f - ug$ is divisible by X (i.e., so that its constant term is 0). Then in Step 4 we divide f by X until its constant term is non-zero. Also, in Step 9 we make sure that $\deg(f) \geq \deg(g)$. The net effect is that each time through the loop the total degree $\deg(f) + \deg(g)$ is reduced by at least 1, so eventually f becomes a constant (provided $\gcd(f, g) = 1$). Hence the algorithm terminates in at most $\deg(a) + \deg(m)$ iterations.

Thus the algorithm produces a sequence of transformations $D_1, D_2, \ldots, D_r$, where each D_i is one of A, B, or C_u, so that

$$(a, m)D_1 D_2 D_3 \cdots D_{r-1} D_r = (\alpha, *),$$

where α is a non-zero number modulo p. Unfortunately, the coefficients of the product $D_1D_2\cdots D_r$ are not polynomials, because the matrix B has X^{-1} as an entry. Let k be the number of times that B appears in the product $D_1D_2\cdots D_r$. (It is easily seen that this is the value of k being computed by the algorithm.) Then $X^kD_1D_2\cdots D_r$ has coefficients that are polynomials, say

$$X^kD_1D_2\cdots D_r = \begin{pmatrix} a' & * \\ m' & * \end{pmatrix}.$$

Now multiplying on the left by (a, m) yields

$$\begin{aligned}(aa' + mm', *) &= (a, m)\begin{pmatrix} a' & * \\ m' & * \end{pmatrix} \\ &= (a, m)X^kD_1D_2\cdots D_r \\ &= X^k(\alpha, *),\end{aligned}$$

so we have

$$aa' \equiv \alpha X^k \pmod{m}.$$

The question now is how does the almost inverse algorithm construct this value a'? The answer is that while it is applying the transformations $D_1, D_2, \ldots, D_r$ starting from (a, m), it is applying the same transformations starting from $(b, c) = (1, 0)$, except that in place of $B = \begin{pmatrix} X^{-1} & 0 \\ 0 & 1 \end{pmatrix}$, it instead applies $XB = \begin{pmatrix} 1 & 0 \\ 0 & X \end{pmatrix}$. Since B has been used k times, at the end of the algorithm the value of (b, c) is

$$(b, c) = (1, 0)X^kD_1D_2\ldots D_r = (1, 0)\begin{pmatrix} a' & * \\ m' & * \end{pmatrix} = (a', *).$$

In other words, at the end of the algorithm, b has a value satisfying

$$ab \equiv \alpha X^k \pmod{m}.$$

Since the value of α is simply f_0 (the constant term of f, which actually equals f at this stage of the algorithm), we see that $a^{-1} = f_0^{-1}X^{N-k}b$. (Note X^{-k} is equal to X^{N-k}, since we are working modulo $X^N - 1$.)

References

[1] R. Schroeppel, S. O'Malley, H. Orman, O. Spatscheck, Fast key exchange with elliptic curve systems, *Advances in Cryptology — CRYPTO 95*, Lecture Notes in Computer Science 973, D. Coppersmith, ed., Springer-Verlag, New York, 1995, 43–56.

Comments and questions concerning this technical report should be addressed to

`techsupport@ntru.com`

Additional information concerning NTRU Cryptosystems and the NTRU Public Key Cryptosystem are available at

`www.ntru.com`
