On NTRU-ν-um Modulo $X^N - 1$

Marc Joye

Zama, Paris, France

Abstract

NTRU-ν-um is a fully homomorphic encryption schemes making use of NTRU as a building block. NTRU-ν-um comes originally in two versions: a first instantiation working with polynomials modulo $X^N - 1$ with N a prime [cyclic version] and a second instantiation working with polynomials modulo $X^N + 1$ with N a power of two [negacyclic version]. The cyclic version is now deprecated.

This work shows that the cyclic version of NTRU-ν-um is not secure. Specifically, it does not provide indistinguishability of encryptions. More critically, the scheme leaks the underlying private LWE keys. Source code for mounting the attacks is provided. The attacks were practically validated on the given parameter sets.

Keywords: NTRU-ν-um · Fully homomorphic encryption · Key recovery attack

1 NTRU-ν-um

This section provides a brief description of the cyclic version of NTRU-ν-um (read NTRUnium). We refer to the original ePrint report [5][1] for further details. The scheme relies on the NTRU problem [3].

Let $\mathcal{R}_q = (\mathbb{Z}/q\mathbb{Z})[X]/(X^N - 1)$ with N prime. Following [5], an NTRU sample is a polynomial $c \in \mathcal{R}_q$ of the form

$$c \leftarrow \frac{e_1}{f} + e_2 + \mu$$

[1] Report has been updated after the attacks were reported. The current version of the report and companion paper [6] only contain the negacyclic version of the scheme.

where

$$\begin{cases} f \in \mathcal{R}_q \text{ is the private key} \\ e_1, e_2 \in \mathcal{R}_q \text{ are error polynomials} \\ \mu \in \mathcal{R}_q \text{ is the encoding of a cleartext } m \end{cases}$$

such that

- f is invertible in $\mathcal{R}_q$ and has random coefficients (uniformly) chosen in $\{-1, 0, 1\}$;
- e_1 and e_2 are drawn according to some distributions;
- $\mu = \Delta m$ with $\Delta = q/p$ for some $p \mid q$ and $m \in \mathcal{R}_p$.

The decryption of c proceeds in three steps as:

1. Compute in $\mathcal{R}_q$ polynomial $d \leftarrow c f = e_1 + e_2 f + \mu f$;
2. Rescale and round d to get $\bar{d} \in \mathcal{R}_p$ as $\bar{d} \leftarrow \lceil d/\Delta \rfloor \pmod p$;
3. Multiply in $\mathcal{R}_p$ by f^{-1} and return $m \leftarrow \bar{d} f^{-1}$.

Remark 1. Letting $e = e_1 + e_2 f$, correctness of the decryption requires that $\|e\|_\infty < \Delta/2$.

Example parameters Taken from [5, Sect. 5], suggested parameters are $q = 2^Q$ with $30 \leqslant Q \leqslant 42$, $p = 2^P$ with $4 \leqslant P \leqslant 11$, and $N \in \{2039, 4093, 8191, 16381\}$ (primes).

2 Analysis

The blind rotation that is used in NTRU-ν-um as part of the (programmable) bootstrapping procedure takes as inputs an LWE ciphertext $\mathbf{c} \in (\mathbb{Z}/N\mathbb{Z})^{n+1}$ and a collection of n bootstrapping keys, and outputs an NTRU ciphertext $c' \in \mathcal{R}_q$. Each bootstrapping key is a 'gadget' NTRU-type encryption of a key digit of the LWE secret key. Specifically, given parameters B and ℓ such that $B^\ell \mid q$, the bootstrapping key corresponding to key digit $s_i \in \{0, 1\}$ is given by

$$\mathsf{bsk}[i] \leftarrow \big(\mathrm{NTRU}(s_i B^j)\big)_{0 \leqslant j \leqslant \ell - 1} \in (\mathcal{R}_q)^\ell$$

for $1 \leqslant i \leqslant n$. An inspection of [5, Theorem 1] indicates that the variance of the noise present in the output ciphertext has a term proportional to the

variance of the noise present in the input bootstrapping keys; the expansion factor being given by

$$\rho = \tfrac{1}{12}\, n\, N\, \ell\, (B^2 - 1) \ .$$

A similar analysis can be done for ternary LWE keys. In this case, expansion factor ρ has an extra multiplicative factor of 2 (again see [5, Theorem 1]).

Applied to the parameter sets with binary LWE keys and with ternary LWE keys in [5, Table 1], this leads to:

Table 1: Values of $\sqrt{\rho}$ for different parameter sets.

	q	N	$\sqrt{\rho}$
Binary LWE keys			
NTRU-ν-um-C-11-B	2^{30}	$2^{11} - 9$	$2^{15.64}$
NTRU-ν-um-C-12-B	2^{38}	$2^{12} - 3$	$2^{18.14}$
NTRU-ν-um-C-13-B	2^{41}	$2^{13} - 1$	$2^{19.68}$
NTRU-ν-um-C-14-B	2^{42}	$2^{14} - 3$	$2^{20.23}$
Ternary LWE keys			
NTRU-ν-um-C-11-T	2^{30}	$2^{11} - 9$	$2^{14.40}$
NTRU-ν-um-C-12-T	2^{38}	$2^{12} - 3$	$2^{20.46}$
NTRU-ν-um-C-13-T	2^{42}	$2^{13} - 1$	$2^{20.20}$
NTRU-ν-um-C-14-T	2^{42}	$2^{14} - 3$	$2^{20.70}$

In all cases, if σ_{bsk} denotes the standard deviation of the noise in the bootstrapping keys then the noise in a ciphertext c' resulting from the blind rotation has a standard deviation which is lower-bounded by $\sqrt{\rho} \cdot \sigma_{\mathsf{bsk}}$. That noise should not "touch" the underlying cleartext, as otherwise decryption would not be possible. In particular, this requires $\sqrt{\rho} \cdot \sigma_{\mathsf{bsk}} \ll \Delta/2$ (cf. Remark 1), which limits the size of σ_{bsk} for the parameter sets of Table 1.

3 Attacks

In this section, we exhibit two attacks against NTRU-ν-um modulo $X^N - 1$. The first attack is a general ciphertext-only attack that breaks the semantic security of NTRU-ν-um encryption from mildly noisy ciphertexts. The second attack targets NTRU-ν-um as a fully homomorphic encryption. It builds on the knowledge of the bootstrapping keys. This second attack is a total break as it uncovers the underlying LWE private keys.

3.1 A General Attack from Mildly Noisy Ciphertexts

Quotient polynomial $X^N - 1$ factors as $(X-1)\Phi_N(X)$ where $\Phi_N(X) = \sum_{i=0}^{N-1} X^i$. Importantly, since $X-1$ divides $X^N - 1$, an NTRU sample verifies

$$c\,f \equiv \underbrace{e_1 + e_2 f}_{=e} + \mu f \pmod{(q, X-1)}$$

or, equivalently,

$$c(1)\,f(1) \equiv \underbrace{e_1(1) + e_2(1)\,f(1)}_{=e(1)} + \mu(1)\,f(1) \pmod{q} \ . \tag{1}$$

The right-hand side of Equation (1) contains two terms. First, there is an error term, $e(1) = e_1(1) + e_2(1)\,f(1)$. Letting $e \coloneqq e(X) = \sum_{i=0}^{N-1} e_i X^i$ with $\mathbb{E}[e_i] = 0$ and $\mathrm{Var}(e_i) = \sigma^2$, it turns out that $\mathbb{E}[e(1)] = 0$ and

$$\begin{aligned} \mathrm{Var}(e(1)) &= \mathrm{Var}\left(\textstyle\sum_{i=0}^{N-1} e_i\right) \\ &= N\sigma^2 \ . \end{aligned} \tag{2}$$

The second term in Equation (1), $\mu(1)\,f(1)$, satisfies

$$\begin{aligned} \mu(1)\,f(1) \bmod q &= \Delta m(1)\,f(1) \bmod q \\ &= \Delta \cdot \left(m(1)\,f(1) \bmod p\right) \ . \end{aligned} \tag{3}$$

A useful observation Another way to look at Equation (1) is to view $c(1) \in \mathbb{Z}/q\mathbb{Z}$ as an NTRU encryption with $N = 1$ of cleartext $m(1) \in \mathbb{Z}/p\mathbb{Z}$. This is a valid ciphertext, provided that the corresponding noise $e(1) \in \mathbb{Z}$ is smaller (in absolute value) than $\Delta/2$. Schematically, together with Eq. (3), we have:

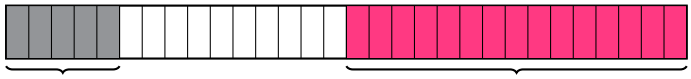
$d(1) =$ 

Figure 1: Noisy value $d(1) = \Delta \cdot (m(1)\,f(1) \bmod p) + e(1) \in \mathbb{Z}/q\mathbb{Z}$ matching valid ciphertext $c(1) = \frac{d(1)}{f(1)} \in \mathbb{Z}/q\mathbb{Z}$, where $e(1) = e_1(1) + e_2(1) f(1)$.

We call *'mildly noisy'* an NTRU sample $c \in \mathcal{R}_q$ whose noise $e \in \mathcal{R}$ satisfies

$$\|e\|_\infty \ll \frac{\Delta}{2\sqrt{N}} \ .$$

Informally, this means that more than $\log_2 \sqrt{N}$ bits following (each coefficient of) cleartext message m are "clean"; that is, are unaffected by the noise. Such a ciphertext is a valid ciphertext modulo $X-1$ and underlies, modulo $X-1$, the noisy value $d(1)$ as illustrated in Fig. 1.

The notion of mildly noisy ciphertexts suggests the following attack. The noise $e(1)$, seen as a signed integer, is such that $|e(1)| \ll \Delta/2$. In the representation of $d(1) \bmod q$, as can be seen in Fig. 1, this implies that the two bits following $m(1)\,f(1) \pmod p$ must be 00 or 11. The configurations 01 or 10 are not possible. Furthermore, since private key $f(X) = \sum_{i=1}^{N-1} f_i X^i$ with $f_i \in \{-1, 0, 1\}$, the norm of its evaluation at 1 is upper-bounded by N. In other words, $f(1)$, seen as a signed integer, lies in $\{-N, \ldots, N\}$. The value of $f(1)$ is unknown but can be searched exhaustively.

For each candidate value for $f(1) \in \{0, \ldots, N\}$, one checks whether $c(1)\,f(1) \bmod q$ $(= d(1) \bmod q)$ has 01 or 10 for its two bits following the "cleartext" position. If so, the tested candidate value for $f(1)$ is disregarded. The operation is repeated with another mildly noisy ciphertext until a single candidate for $f(1)$ remains. If we call F_1 this unique candidate then the correct value for $f(1)$ is $\pm F_1 \pmod q$—the test does not permit to recover the sign. To make the test more selective, extra noise can be increasingly added to mildly noisy ciphertexts c prior to applying the test. In practice, for typical parameters, about 10 ciphertexts suffice to recover the value of $\pm f(1)$. Once this value is known, ciphertexts in NTRU-ν-um can easily be distinguished.

The scheme is therefore not semantically secure in the case *mildly noisy* ciphertexts are known and made available to an attacker.

Remark 2. A GP-Pari implementation of the above attack is given in appendix; see `findkeyat1()`.

Variants There are several possible variants of the presented attack. For example, instead of adding extra noise for more effectiveness, one can test whether the resulting configurations are different from 000 or 111, and so on for increasingly longer sequences of 0 and of 1.

The knowledge of a pair of cleartext/ciphertext allows the resolution of the ambiguity on the sign of $f(1)$.

When several pairs of cleartext/ciphertext are available, another way to discriminate a candidate value for $f(1)$ is to check whether or not the ciphertext correctly decrypts modulo $X-1$. The attacks can also be combined.

3.2 A Key Recovery Attack against NTRU-ν-um

The evaluation keys in a fully homomorphic encryption scheme are public keys for operating over encrypted data. In the case of NTRU-ν-um, they consist of bootstrapping keys and of key-switching keys.

As seen in Section 2, given a binary LWE key $\mathbf{s} = (s_1, \ldots, s_n) \in \{0,1\}^n$, the bootstrapping keys are NTRU-type encryptions of bits s_i (and of $s_i\, B^j$). Ternary LWE keys $\mathbf{s'} \in \{-1,0,1\}^n$ can be expressed as the difference of two binary strings: $\mathbf{s'} = \mathbf{s}^{(1)} - \mathbf{s}^{(2)}$ where $\mathbf{s}^{(1)}, \mathbf{s}^{(2)} \in \{0,1\}^n$ [7]. In this case, the bootstrapping keys are NTRU-type encryptions of bits $s_i^{(l)}$ (and of $s_i^{(l)}\, B^j$) for $l \in \{1,2\}$.

A bit $\beta \in \{0,1\}$—and in particular a key bit of an LWE key—is represented as a degree-0 plaintext $\mu(X) = \Delta\,\beta$, and thus $\mu(1) = \Delta\,\beta$. Decrypting the corresponding NTRU ciphertext modulo $X - 1$ is therefore enough to recover the value of β using key $\ell(1)$. This supposes however that the corresponding ciphertext is mildly noisy.

The analysis in Section 2 concludes that the noise in a bootstrapping key cannot be too large. Specifically, the standard deviation should be such that $\sigma_{\mathsf{bsk}} \ll \frac{\Delta}{2\sqrt{\rho}}$ for some $\rho > N$, and so

$$\sigma_{\mathsf{bsk}} \ll \frac{\Delta}{2\sqrt{N}} \ .$$

As a result, the NTRU-ν-um bootstrapping keys are mildly noisy ciphertexts, which, as we have demonstrated, leaks the value of $\ell(1)$. Moreover, as the bootstrapping keys contain encryptions of LWE key bits, these LWE key bits can be recovered using $\ell(1)$. This in turn leads to the full recovery of the LWE private key.

4 Concluding Remarks

Working modulo $X^N - 1$ in NTRU-ν-um presents the convenient property that $X^N = 1$, which simplifies the mechanism of programmable bootstrapping. This work unfortunately shows that this results in an insecure setting.

NTRU-ν-um also has a variant modulo $X^N + 1$ with N a power of two. The attacks presented in this report do not apply in this case—note that polynomial $X^N + 1$ is irreducible when N is a power of two. This negacyclic version of NTRU-ν-um should nevertheless be used with caution. The parameter selection in NTRU is quite tricky; in particular, NTRU is known to have its security decreasing in the so-called 'overstretched' regime—that is,

when modulus q is much larger than degree N [1, 4]. An improved analysis in [2] concretely settles the NTRU fatigue point at $q \approx 0.004\, N^{2.484}$. It has to be noted that all the parameter sets suggested for NTRU-ν-um largely exceed the fatigue point.

References

[1] Martin R. Albrecht, Shi Bai, and Léo Ducas. A subfield lattice attack on overstretched NTRU assumptions. In M. Robshaw and J. Katz, editors, *Advances in Cryptology – CRYPTO 2016, Part I*, volume 9814 of *Lecture Notes in Computer Science*, pages 153–178. Springer, 2016. doi:10.1007/978-3-662-53018-4_6.

[2] Léo Ducas and Wessel P. J. van Woerden. NTRU fatigue: How stretched is overstretched? In M. Tibouchi and H. Wang, editors, *Advances in Cryptology – ASIACRYPT 2021, Part IV*, volume 13093 of *Lecture Notes in Computer Science*, pages 3–32. Springer, 2021. doi:10.1007/978-3-030-92068-5_1.

[3] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. NTRU: A ring-based public key cryptosystem. In J. P. Buhler, editor, *Algorithmic Number Theory Symposium (ANTS-III)*, volume 1423 of *Lecture Notes in Computer Science*, pages 267–288. Springer, 1198. doi:10.1007/BFb0054868.

[4] Paul Kirchner and Pierre-Alain Fouque. Revisiting lattice attacks on overstretched NTRU parameters. In J.-S. Coron and J. B. Nielsen, editors, *Advances in Cryptology – EUROCRYPT 2017, Part I*, volume 10210 of *Lecture Notes in Computer Science*, pages 3–26. Springer, 2017. doi:10.1007/978-3-319-56620-7_1.

[5] Kamil Kluczniak. NTRU-ν-um: Secure fully homomorphic encryption from NTRU with small modulus. Cryptology ePrint Archive, Report 2022/089, 2022. https://eprint.iacr.org/archive/2022/089/20220125:072357.

[6] Kamil Kluczniak. NTRU-ν-um: Secure fully homomorphic encryption from NTRU with small modulus. In *2022 ACM SIGSAC Conference on Computer and Communications Security (ACM CCS 2022)*, page 17831797. ACM Press, 2022. doi:10.1145/3548606.3560700.

[7] Daniele Micciancio and Yuriy Polyakov. Bootstrapping in FHEW-like cryptosystems. In M. Brenner et al., editors, *9th Workshop on Encrypted Computing & Applied Homomorphic Cryptography (WAHC 2021)*, pages 17–28. ACM Press, 2021. doi:10.1145/3474366.3486924.

[8] The PARI Group, Univ. Bordeaux. *PARI/GP version 2.13.4*, 2022. Available from http://pari.math.u-bordeaux.fr/.

A Code

Below is an implementation of the attacks. They are developed with the computer algebra system PARI/GP [8].

```
PREC = 2^30;     \\ default precision (i.e., modulus q)
MSG = 2^4;       \\ default message precision (i.e., modulus p)
STDEV = 2^-25;   \\ default standard deviation (torus notation)
DIM = 2039;      \\ default N (prime)

/* ----------------------------------------------------------------
                            General functions
   ---------------------------------------------------------------- */
\\ Draws at random an element in N(0,sd^2) using Box-Muller method,
\\ and converts it as an integer in Z/qZ
nrandom(q = PREC, sd = STDEV) = {
  my(X);
  X = sqrt(-2*log(random(1.)))*cos(2*Pi*random(1.));
  return(Mod(round(q*sd*X),q));
}

\\ Computes the inverse of a polynomial f in (Z/qZ)[X]/(X^N - 1)
polinv2(f) = {
  my(i, q, l, g);
  q = polcoef(lift(f),0).mod; l = valuation(q,2);
  g = Mod(1,2)*f; g = liftall(1/g); g = Mod(1,q)*Mod(g,f.mod);
  for(i=2, l,
    g = g*(2 - f*g);
  );
  return(g);
}

/* ----------------------------------------------------------------
                                NTRUnium
   ---------------------------------------------------------------- */
\\ Generates private key f
NTRUnium_keygen(N = DIM, q = PREC) = {
  my(i, f);
  f = Polrev(vector(N, i, random(3) - 1));
  return(Mod(1,q)*Mod(f, x^N - 1));
}

\\ Encodes a message m in (Z/pZ)[X]/(X^N - 1) as an
\\ element mu in (Z/qZ)[X]/(X^N - 1)
NTRUnium_encode(m, q = PREC) = {
  my(p, mu);
  p = polcoef(lift(m),0).mod;
  mu = q/p*liftall(m);
  return(Mod(1,q)*Mod(mu,m.mod));
}

\\ Encrypts encoded message mu with key f
NTRUnium_encrypt(f, mu, sd = STDEV) = {
  my(i, q, N, e1, e2, finv);
  q = polcoef(lift(f),0).mod; N = poldegree(f.mod);
  e1 = sum(i=0, N-1, nrandom(q,sd)*x^i);
  e2 = sum(i=0, N-1, nrandom(q,sd)*x^i);
  finv = polinv2(f);
  return(e1*finv + e2 + mu);
}
```

```
\\ Decrypts ciphertext c with key f
NTRUnium_decrypt(f, c, p = MSG) = {
  my(q, d, finv);
  q = polcoef(lift(c),0).mod;
  d = liftall(c*f); d = round(d*p/q); d = Mod(1,p)*Mod(d,f.mod);
  f = Mod(1,p)*f; finv = polinv2(f);
  return(d*finv);
}

/* ----------------------------------------------------------------
                                Attacks
   ---------------------------------------------------------------- */
findkeyat1(keylist = [], sd = STDEV, N = DIM, q = PREC, p = MSG) = {
  if(keylist == [], keylist = vector(N,i,i));
  cont = 1;
  while(cont,
    print("-> ", #keylist);
    m = Mod(1,p)*Mod(sum(i=0,N-1, random(p)*x^i), x^N - 1);
    mu = NTRUnium_encode(m, q);
    c = NTRUnium_encrypt(f, mu, sd);
    C1 = subst(lift(c),x,1);
    potlist = [];
    for(i=1, #keylist,
      tmp = lift(C1*keylist[i]);
      tau = 2^2;
      tmp = bitand(tmp,(tau-1)*q/(tau*p));
      tmp = tau*p*tmp/q;
      potkey = ((tmp==0) || (tmp==tau-1));
      if(potkey, potlist = concat(potlist,i));
    );
    tt = #keylist;
    keylist = vecextract(keylist, potlist);
    cont = (#keylist < tt);
  );
  return(keylist);
}

keybitreco(F1, sd = STDEV, N = DIM, q = PREC, p = MSG) = {
  beta = Mod(1,p)*Mod(random(2), x^N - 1);
  mu = NTRUnium_encode(beta, q);
  c = NTRUnium_encrypt(f, mu, sd);
  C1 = subst(lift(c),x,1);
  d = liftall(C1*F1); d = Mod(1,p)*round(d*p/q);
  betap = d/lift(F1);
  ok = (betap == lift(beta));
  return(ok);
}

setup(N = DIM, q = PREC, p = MSG) = {
  ok = 0;
  until(ok,
    f = NTRUnium_keygen(N, q);
    tmp = iferr(finv = polinv2(f), E, E);
    ok = (type(tmp) != "t_ERROR"));
  F1 = centerlift(subst(lift(f),x,1));
  return(F1);
}
```