NIST PQ Submission: NTRUEncrypt A lattice based encryption algorithm

Cong Chen[2], Jeffrey Hoffstein[1], William Whyte[2], and Zhenfei Zhang[2]

[1] Brown University, Providence RI, USA, jhoff@math.brown.edu
[2] OnBoard Security, Wilmington MA, USA,
{cchen,wwhyte,zzhang}@onboardsecurity.com

1 Cover Sheet

This is an overview document of NTRU lattice-based cryptosystem for the submission of NIST post-quantum cryptography call for standardization. The submitted cryptosystem consists of :

- ntru-kem: a CPA secure key encapsulation scheme (KEM) based on the original NTRUEncrypt algorithm by Hoffstein, Pipher and Silverman [**?**] with parameter sets derived from a recent revision [**?**];
- ntru-pke: a public key encryption (PKE) scheme based on the above NTRUEncrypt algorithm that achieves CCA-2 security via NAEP transformation [**?**];
- ss-ntru-kem: a key encapsulation scheme based on the original NTRUEncrypt algorithm with parameter sets derived from the Stehle-Sternfield "provable secure" NTRU [**?**];
- ss-ntru-pke: a public key encryption scheme based on the above provable secure NTRUEncrypt algorithm that achieves CCA-2 security via NAEP transformation [**?**].

This documents addresses the following requirements:

- **Specifications**
- **Performance analysis**
- **A statement of the advantages and limitations**
- **Cover sheet**
- **Reference implementation**
- **Security analysis**
- **Statement of IPR**

Submission information:

- **Principal submitter:** Zhenfei Zhang, zzhang@onboardsecurity.com, Onboard Security, 187 Ballardvale St. Suite A202, Wilmington, MA, 01887, U.S.
- **Auxiliary submitters:** Chen Cong, Jeffrey Hoffstein and William Whyte.

– **Inventors of the cryptosystem:** Jeffrey Hoffstein, Jill Pipher, John M. Schanck, Joseph H. Silverman, William Whyte and Zhenfei Zhang.
– **Name of the owner of the cryptosystem:** Onboard Security Inc.
– **Backup point of contact:** William Whyte, wwhyte@onboardsecurity.com, Onboard Security, 187 Ballardvale St. Suite A202, Wilmington, MA, 01887, U.S.

2 Algorithm Specifications

2.1 Notation

We use lower case bold letters for vectors, upper case bold letters for matrices. For a polynomial $f(x) = f_0 + f_1 x + \cdots + f_{n-1}x^{n-1}$, we denote its vector form by $\mathbf{f} \coloneqq \langle f_0, f_1, \ldots, f_{n-1}\rangle$. We sometimes abuse the notation of vector and polynomial when there is no ambiguity. For a polynomial/vector $\mathbf{f}$, the norms are $\|\mathbf{f}\| \coloneqq \sqrt{\sum_{i=0}^{n-1} f_i^2}$ and $\|\mathbf{f}\|_\infty \coloneqq \max(|f_i|)$.

We often use the polynomial rings $\mathcal{R}_q \coloneqq \mathbb{Z}[x]/F(x)$ with $F(x) = x^n \pm 1$. A cyclic rotated matrix of a polynomial $f(x)$ over the ring $\mathcal{R}_q$ is a matrix $\mathbf{M} = (\mathbf{f}_1, \mathbf{f}_2, \ldots, \mathbf{f}_n)^T$ with $\mathbf{f}_i = f(x)x^{i-1} \bmod F(x)$. If $F(x) = x^n - 1$ it is literally cyclic, and close to cyclic, up to signs, if $F(x) = x^n + 1$.

For a real a, we let $\lfloor a \rceil$ denote the closet integer to a. For an integer a, we use $[a]_q$ to denote $a \bmod q$; $\lfloor a \rfloor_p \coloneqq (a - [a]_p)/p$ for the operation of rounding a to the closest multiple of p. Modular operations are center lifted, for example $a \bmod q$ returns an integer within $-q/2$ and $q/2$. These notations are also extended to vectors and matrices.

We set the notation:

$$\mathcal{T}_N = \{\text{trinary polynomials}\}$$
$$\mathcal{T}_N(d, e) = \left\{ \begin{array}{l} \text{trinary polynomials with exactly} \\ d \text{ ones and } e \text{ minus ones} \end{array} \right\}$$

If N is fixed we will write $\mathcal{T}$ and $\mathcal{T}(d, e)$ instead.

2.2 NTRU lattice

Let $f(x)$, $g(x)$ and $h(x)$ be 3 polynomials in $\mathcal{R}_q$, where $f(x)$ and $g(x)$ have very small coefficients; $h(x) = p^{-1}g(x)f^{-1}(x)$. We express by $\mathbf{f}$, $\mathbf{g}$ and $\mathbf{h}$ the vector form of the polynomials. Also let $\mathbf{F}$, $\mathbf{G}$ and $\mathbf{H}$ be the matrix obtained from nega-cyclic rotations. The NTRU lattice with regard to h is defined as

$$\mathcal{L}_h = \{(u, v) \in \mathcal{R}_q^2 : uh = v\}$$

or rather, the vector/matrix form:

$$\mathcal{L}_h = \{(\mathbf{u}, \mathbf{v}) : \mathbf{uH} = \mathbf{v} \bmod q\}$$

where there exists a public basis $\mathbf{P} = \begin{bmatrix} 0 & q\mathbf{I}_N \\ \mathbf{I}_N & \mathbf{H} \end{bmatrix}$ and a secret generator $[p\mathbf{F}|\mathbf{G}]$.

2.3 Auxiliary functions

Let us firstly define some auxiliary functions. Those functions are building blocks for our algorithm. We give a generic description for those functions. Implementer may choose to use better (more secure or more efficient) instantiations when and if they are available.

Hash function. Through out the paper we will use HASH to denote a cryptographic secure hash function that takes arbitrary input length and outputs a binary string with arbitrary length. In our submission, we use SHA3-512 for such instantiation.

Trinary Polynomial Generation function.

Discrete Gaussian sampler (DGS). Input a dimension N and a standard deviation σ it outputs a discrete Gaussian distributed vector.

Deterministic Discrete Gaussian sampler (DDGS). Input a dimension N, a standard deviation σ and a seed s, it deterministically outputs a discrete Gaussian distributed vector.

2.4 The ntru-kem algorithms

In an NTRU cryptosystem, $\mathbf{f}$ (and $\mathbf{g}$, if required) are the private keys, while $\mathbf{h}$ is the public key. Those keys can be generated via algorithm **??**. Note that we use the classical NTRU flat form (non-product form, cf. [**?**]) keys with a pre-fixed number of $+1$s and -1s.

Algorithm 1 ntru-kem.KEYGEN

Input: Parameters N, p, q, d
Output: Public key $\mathbf{h}$ and secret key $(p\mathbf{f}, \mathbf{g})$
1: $\mathbf{f} \leftarrow T(d+1, d)$
2: **if** $\mathbf{f}$ is not invertible mod q **then** go to step 1 **end if**
3: $\mathbf{g} \leftarrow T(d+1, d)$
4: **if** $\mathbf{g}$ is not invertible mod p **then** go to step 3 **end if**
5: $\mathbf{h} = \mathbf{g}/(p\mathbf{f}+1) \bmod q$
6: **return** $\mathbf{h}$, $\mathbf{g}$ and $\mathbf{f}$

We recommend that ntru-kem to be used for ephemeral key establishments via the following algorithms.

Here the padding algorithm works as follows:

– Convert msg into a bit string. This forms the binary coefficients for lower part of polynomial m.

Algorithm 2 ntru-kem.ENCAP

Input: Public key $\mathbf{h}$, message msg of length $mlen$, and a parameter set

1: $\mathbf{m} = \text{Pad}(msg)$
2: $\mathbf{r} \leftarrow T(d+1, d)$
3: $\mathbf{c} = p \cdot \mathbf{r} * \mathbf{h} + \mathbf{m}$

Output: A ciphertext $\mathbf{c}$

- The last 167 coefficients of $m(x)$ are randomly chosen from $\{-1, 0, 1\}$. This gives over 256 bits entropy.
- The length of msg is converted into an 8 bit binary string, and forms the last 173 to 168 coefficients of m.

Algorithm 3 ntru-kem.DECAP

Input: Secret key f and a parameter set

1: $\mathbf{m} = (p \cdot \mathbf{c} * \mathbf{f}) \bmod p$
2: $msg = \text{Extract}(\mathbf{m})$

Output: A message msg

The extract operation is the inverse of Pad so we omit the details. It outputs a message m and its length $mlen$.

We remark that when used in an KEM mode, one should use both m and pk to derive the session key, i.e. $\mathsf{KDF}(m, pk, \dots)$. Also the responder needs to pick an m that has sufficient entropy for the given security level. In our implementation we require 32 bytes for m, regardless of security level.

2.5 The ntru-pke algorithms

ntru-pke uses a same key generation algorithm as ntru-kem. Here we present the encryption and decryption algorithms.

Algorithm 4 ntru-pke.ENCRYPT

Input: Public key $\mathbf{h}$, message msg of length $mlen$, and a parameter set

1: $\mathbf{m} = \text{Pad}(msg)$
2: $\mathbf{r} = \text{hash}(\mathbf{m}|\mathbf{h})$
3: $\mathbf{t} = \mathbf{r} * \mathbf{h}$
4: $\mathbf{m}_{mask} = \text{hash}(\mathbf{t})$
5: $\mathbf{m}' = \mathbf{m} - \mathbf{m}_{mask} \pmod p$
6: $\mathbf{c} = \mathbf{t} + \mathbf{m}'$

Output: Ciphertext $\mathbf{c}$

Algorithm 5 ntru-pke.DECRYPT

Input: Secret key f, public key h, ciphertext c, and a parameter.

$\mathbf{m}' = \mathbf{f} * \mathbf{c} \pmod{p}$
$\mathbf{t} = \mathbf{c} - \mathbf{m}$
$\mathbf{m}_{mask} = \text{hash}(\mathbf{t})$
$\mathbf{m} = \mathbf{m}' + \mathbf{m}_{mask} \pmod{p}$
$\mathbf{r} = hash(\mathbf{m}|\mathbf{h})$
$msg = \text{Extract}(\mathbf{m})$
if $p \cdot \mathbf{r} * \mathbf{h} = \mathbf{t}$ **then**
    $result = msg$
else
    $result = \perp$
end if

Output: $result$

2.6 The ss-ntru-kem algorithms

Algorithm 6 ss-ntru-kem.KEYGEN

Input: Parameters N, p, q, σ

Output: Public key $\mathbf{h}$ and secret key $(p\mathbf{f}, \mathbf{g})$

$\mathbf{f} \leftarrow \chi_\sigma^N$; $\mathbf{g} \leftarrow \chi_\sigma^N$
$\mathbf{h} = \mathbf{g}/(p\mathbf{f} + 1) \bmod q$
return $\mathbf{h}$, $\mathbf{g}$ and $\mathbf{f}$

Algorithm 7 ss-ntru-kem.ENCAP

Input: Public key h, message msg of length $mlen$, and a parameter set

$m = \text{Pad}(msg)$
$\mathbf{r} \leftarrow \chi_\sigma^N$; $\mathbf{e} \leftarrow \chi_\sigma^N$
$\mathbf{c} = p \cdot \mathbf{r} * \mathbf{h} + p \cdot \mathbf{e} + \mathbf{m}$

Output: Ciphertext $\mathbf{c}$

2.7 The ss-ntru-pkealgorithms

ss-ntru-pke uses a same key generation algorithm as ntru-kem. Here we present the encryption and decryption algorithms.

Algorithm 8 ss-ntru-kem.DECAP

Input: Secret key f and a parameter set

$\mathbf{m} = (p \cdot \mathbf{c} * \mathbf{f}) \bmod p$

$msg = \text{Extract}(\mathbf{m})$

Output: Ciphertext c

Algorithm 9 ss-ntru-pke.ENCRYPT

Input: Public key $\mathbf{h}$, message msg of length $mlen$, and a parameter set

$\mathbf{m} = \text{Pad}(msg)$

$\mathbf{r} = \text{DDGS}(\mathbf{m}|\mathbf{h})$

$\mathbf{e} = \text{DGS}$

$\mathbf{t} = p \cdot \mathbf{r} * \mathbf{h}$

$\mathbf{m}_{mask} = \text{hash}((\mathbf{t} \bmod p))$

$\mathbf{m}' = \mathbf{m} - \mathbf{m}_{mask} \pmod p$

$\mathbf{c} = \mathbf{t} + p \cdot \mathbf{e} + \mathbf{m}'$

Output: Ciphertext $\mathbf{c}$

Algorithm 10 ss-ntru-pke.DECRYPT

Input: Secret key f, public key h, ciphertext c, and a parameter.

$\mathbf{m}' = \mathbf{f} * \mathbf{c} \pmod p$

$\mathbf{t} = \mathbf{c} - \mathbf{m}$

$\mathbf{m}_{mask} = \text{hash}((\mathbf{t} \bmod p))$

$\mathbf{m} = \mathbf{m}' + \mathbf{m}_{mask} \pmod p$

$\mathbf{r} = \text{DDGS}(\mathbf{m}|\mathbf{h})$

$\mathbf{e} = p^{-1}(\mathbf{t} - \mathbf{r} * \mathbf{h})$

if $|\mathbf{e}|_\infty \geq \tau\sigma$ **then**

    $result = \perp$

else

    $result = \text{Extract}(\mathbf{m})$

end if

Output: $result$

3 Design Rationale

3.1 Hardness assumption

3.2 Best known attacks

3.3 Advantages and limitations

Most scrutinized lattice-based cryptosystem

NTRU *trapdoor.* In general lattice based signature offers best performance among quantum-safe solutions, in terms of the combination of signature sizes and public key sizes. However, the performance changes greatly with how the trapdoor is the constructed. NTRU trapdoor is in general the most efficient one in the literature; yet it survived 20 years of cryptanalysis, which none other lattice based solution has gone through.

Potential application for signature aggregation . **ZZ: shall we mention it at all?**

3.4 Performance and implementations

Optimizations not in this submission package. There are two optimizations that we are aware of, that are not included in this submission package. Namely

- AVX2 based optimization for polynomial multiplication [**?**]; this accelerates polynomial multiplication for 2.3 times.

3.5 Known Answer Test Values

4 IPR Statement