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-pke: a public key encryption (PKE) scheme based on the "*original NTRU*" encryption algorithm by Hoffstein, Pipher and Silverman [4] with parameter sets derived from a recent revision [3], that achieves CCA-2 security via NAEP transformation [5];
- ntru-kem: a key encapsulation scheme (KEM) uses the above public key encryption algorithm;
- ss-ntru-pke: a public key encryption scheme based on the provable secure NTRU encryption algorithm [6] that achieves CCA-2 security via NAEP transformation [5];
- ss-ntru-kem: a key encapsulation scheme (KEM) uses the above public key encryption algorithm.

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 $\mathbf{f} := \langle f_0, f_1, \ldots, f_{n-1}\rangle$. We sometimes abuse the notation of vector and polynomial when there is no ambiguity. For a polynomial/vector $\mathbf{f}$, the norms are $\|\mathbf{f}\| := \sqrt{\sum_{i=0}^{n-1} f_i^2}$ and $\|\mathbf{f}\|_\infty := \max(|f_i|)$.

We often use the polynomial rings $\mathcal{R}_q := \mathbb{Z}[x]/F(x)$ 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 $a \bmod q$; $\lfloor a \rfloor_p := (a - [a]_p)/p$ 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. Also note that Gaussian samplers are only used by ss-ntru-pke and ss-ntru-kem.

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. This implementation require a function that samples uniformly from $\mathcal{T}$ and $\mathcal{T}(d, e)$. This function can be build deterministically via a *seed* and a hash function.

Discrete Gaussian sampler (DGS). Input a dimension N and a standard deviation σ it outputs a discrete Gaussian distributed vector. In this implementation we use Box-Muller approach. We remark that there are better samplers in terms of efficiency or security. We leave the investigation of those samplers to future work.

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 schemes

We sketch the algorithms related to our proposed schemes. For simplicity and clearness of the presentation, we omit minor details in this high level description. Those includes, for example, checking the length of the message and encoding/packing ring elements into binary strings. For those detail we refer the readers to the implantation specification document submitted also with this document.

The ntru-pke scheme . The ntru-pke schemes use Algorithms 1, 2 and 3.

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 1. Note that we use the classical NTRU flat form (non-product form, cf. [2]) keys with a pre-fixed number of $+1$s and -1s.

Algorithm 1 ntru-pke.KeyGen

Input: Parameters N, p, q, d and a *seed*.

1: Instantiate Sampler with $\mathcal{T}(d+1, d)$ and *seed*;
2: $\mathbf{f} \leftarrow$ Sampler
3: **if** $\mathbf{f}$ is not invertible mod q **then** go to step 2 **end if**
4: $\mathbf{g} \leftarrow$ Sampler
5: $\mathbf{h} = \mathbf{g}/(p\mathbf{f} + 1) \bmod q$

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

Algorithm 2 ntru-pke.Encrypt

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

1: $\mathbf{m} = \text{Pad}(msg, seed)$
2: $rseed = \text{Hash}(\mathbf{m}|\mathbf{h})$
3: Instantiate Sampler with $\mathcal{T}$ and $rseed$;
4: $\mathbf{r} \leftarrow$ Sampler
5: $\mathbf{t} = \mathbf{r} * \mathbf{h}$
6: $tseed = \text{Hash}(\mathbf{t})$
7: Instantiate Sampler with $\mathcal{T}$ and $tseed$;
8: $\mathbf{m}_{mask} =$ Sampler
9: $\mathbf{m}' = \mathbf{m} - \mathbf{m}_{mask} \pmod p$
10: $\mathbf{c} = \mathbf{t} + \mathbf{m}'$

Output: Ciphertext $\mathbf{c}$

The encryption algorithm in Algorithm 2 uses a padding method to deal with potential insufficient entropy in a message. The padding algorithm works as follows:

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

The Extract() operation in Algorithm 3 is the inverse of Pad() so we omit the details. It outputs a message m and its length $mlen$.

The ntru-kem algorithms We recommend that ntru-kem to be used for ephemeral key establishments via the following algorithms. ntru-kem uses a same key generation algorithm as ntru-pke, namely, Algorithm 2. Here we present the encapsulation and encapsulation algorithms in Algorithms 4 and 5.

In a nutshell, the ntru-kem scheme uses an ntru-pke scheme to transport an encapsulated secret, and uses both this secret and the public key to derive a shared secret via a secure Key Deviation Function (KDF).

Algorithm 3 ntru-pke.DECRYPT

Input: Secret key $\mathbf{f}$, public key $\mathbf{h}$, ciphertext $\mathbf{c}$, and a parameter.

$\mathbf{m}' = \mathbf{f} * \mathbf{c} \pmod{p}$
$\mathbf{t} = \mathbf{c} - \mathbf{m}$
$tseed = \text{HASH}(\mathbf{t})$
Instantiate Sampler with $\mathcal{T}$ and $tseed$;
$\mathbf{m}_{mask} =$ Sampler
$\mathbf{m} = \mathbf{m}' + \mathbf{m}_{mask} \pmod{p}$
$rseed = \text{HASH}(\mathbf{m}|\mathbf{h})$
Instantiate Sampler with $\mathcal{T}$ and $rseed$;
$\mathbf{r} \leftarrow$ Sampler
$msg, mlen = \text{Extract}(\mathbf{m})$
if $p \cdot \mathbf{r} * \mathbf{h} = \mathbf{t}$ **then**
    $result = msg, mlen$
else
    $result = \perp$
end if

Output: $result$

Algorithm 4 ntru-kem.ENCAP

Input: Public key $\mathbf{h}$, a parameter set PARAM, and a *seed*

encaped_secret $\leftarrow \{0,1\}^{8\times \text{CRYPTO\_BYTES}}$
$\mathbf{c} =$ntru-pke.ENCRYPT($\mathbf{h}$, encaped_secret, CRYPTO_BYTES, PARAM, $seed$)
$ss =$ KDF(encaped_secret, $\mathbf{h}$).

Output: A ciphertext $\mathbf{c}$ and the shared secret ss.

Algorithm 5 ntru-kem.DECAP

Input: Secret key f and a parameter set PARAM

encaped_secret =ntru-pke.DECRYPT($\mathbf{f}, \mathbf{h}, \mathbf{c}$, PARAM);
$ss =$ KDF(encaped_secret, $\mathbf{h}$).

Output: The shared secret ss.

The ss-ntru-pke algorithms The ss-ntru-pke schemes use Algorithms 6, 7 and 8.

Algorithm 6 ss-ntru-pke.KeyGen

Input: Parameters N, p, q, σ and a *seed*

1: Instantiate Sampler with χ_σ^N and *seed*;
2: $\mathbf{f} \leftarrow$ Sampler, $\mathbf{g} \leftarrow$ Sampler;
3: $\mathbf{h} = \mathbf{g}/(p\mathbf{f} + 1) \bmod q$

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

ss-ntru-pke uses a similar key generation algorithm as ntru-pke. The major difference is that $\mathbf{f}$ and $\mathbf{g}$ are sampled from Gaussian with deviation σ, rather than from $\mathcal{T}(d, d+1)$. In addition, since ss-ntru-pke works over the polynomial ring $\mathbb{Z}_q[x]/(x^N + 1)$ where every element has an inverse, we are not required to check if $\mathbf{f}$ and $\mathbf{g}$ has an inverse.

Algorithm 7 ss-ntru-pke.Encrypt

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

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

Output: Ciphertext $\mathbf{c}$

2.5 The ss-ntru-kemalgorithms

3 Design Rationale

3.1 Hardness assumption

3.2 Best known attacks

3.3 Advantages and limitations

Most scrutinized lattice-based cryptosystem

Algorithm 8 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: $result$

Algorithm 9 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 $\mathbf{c}$

Algorithm 10 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

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 [1]; this accelerates polynomial multiplication for 2.3 times.

3.5 Known Answer Test Values

4 IPR Statement

References

1. Wei Dai, William Whyte, and Zhenfei Zhang. Optimizing polynomial convolution for ntruencrypt. TBA.
2. Jeffrey Hoffstein, Jill Pipher, John M. Schanck, Joseph H. Silverman, William Whyte, and Zhenfei Zhang. Choosing parameters for ntruencrypt. *IACR Cryptology ePrint Archive*, 2015:708, 2015.
3. Jeffrey Hoffstein, Jill Pipher, John M. Schanck, Joseph H. Silverman, William Whyte, and Zhenfei Zhang. Choosing parameters for ntruencrypt. In *Topics in Cryptology - CT-RSA 2017 - The Cryptographers' Track at the RSA Conference 2017, San Francisco, CA, USA, February 14-17, 2017, Proceedings*, pages 3–18, 2017.
4. Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. NTRU: A ring-based public key cryptosystem. In *Algorithmic Number Theory, Third International Symposium, ANTS-III, Portland, Oregon, USA, June 21-25, 1998, Proceedings*, pages 267–288, 1998.
5. Nick Howgrave-Graham, Joseph H. Silverman, Ari Singer, and William Whyte. NAEP: provable security in the presence of decryption failures. *IACR Cryptology ePrint Archive*, 2003:172, 2003.
6. Damien Stehlé and Ron Steinfeld. Making NTRU as secure as worst-case problems over ideal lattices. In *Advances in Cryptology - EUROCRYPT 2011 - 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, Estonia, May 15-19, 2011. Proceedings*, pages 27–47, 2011.