NTRU

Algorithm Specifications And Supporting Documentation

Cong Chen, Oussama Danba, Jeffrey Hoffstein, Andreas Hülsing,
Joost Rijneveld, John M. Schanck, Tsunekazu Saito, Peter Schwabe,
William Whyte, Keita Xagawa, Takashi Yamakawa, Zhenfei Zhang

September 30, 2020

Contents

1 Written specification **4**

1.1 Overview 4
1.2 Definitions 4
1.3 Parameter sets 5
1.3.1 NTRU 5
1.3.2 NTRU-HPS 5
1.3.3 NTRU-HRSS 6
1.4 Additional parameters 6
1.4.1 Hash 6
1.4.2 Sample_fg 6
1.4.3 Sample_rm 6
1.4.4 key_seed_bits 7
1.5 Derived constants 7
1.5.1 logq 7
1.5.2 sample_iid_bits 7
1.5.3 sample_fixed_type_bits 7
1.5.4 sample_key_bits 7
1.5.5 sample_plaintext_bits 7
1.5.6 packed_s3_bytes 7
1.5.7 packed_sq_bytes 7
1.5.8 packed_rq0_bytes 7
1.5.9 dpke_public_key_bytes 7
1.5.10 dpke_private_key_bytes 8
1.5.11 dpke_plaintext_bytes 8
1.5.12 dpke_ciphertext_bytes 8
1.5.13 kem_public_key_bytes 8
1.5.14 kem_private_key_bytes 8
1.5.15 kem_ciphertext_bytes 8
1.5.16 kem_shared_key_bits 8
1.5.17 prf_key_bits 8
1.6 Summary of recommended parameters and derived constants 9
1.7 Externally defined algorithms 9
1.7.1 SHAKE256 9
1.7.2 SHA3_256 9
1.8 Encodings 10
1.8.1 Bit strings and byte arrays 10
1.8.2 Polynomials 10
1.8.3 pack_Rq0 10
1.8.4 unpack_Rq0 10
1.8.5 pack_Sq 11
1.8.6 unpack_Sq 11
1.8.7 pack_S3 12
1.8.8 unpack_S3 12
1.9 Arithmetic 12
1.9.1 S2_inverse and S3_inverse 12
1.9.2 Sq_inverse 13
1.9.3 Lift 13
1.10 Sampling 13
1.10.1 Sample_fg 13
1.10.2 Sample_rm 14
1.10.3 Ternary 15
1.10.4 Ternary_Plus 15

1.10.5 Fixed_Type . . . 16
1.11 Passively secure DPKE . . . 16
1.11.1 DPKE_Key_Pair . . . 16
1.11.2 DPKE_Public_Key . . . 17
1.11.3 DPKE_Encrypt . . . 17
1.11.4 DPKE_Decrypt . . . 18
1.12 Strongly secure KEM . . . 19
1.12.1 Key_Pair . . . 19
1.12.2 Encapsulate . . . 19
1.12.3 Decapsulate . . . 20

2 Design rationale **20**
2.1 Summary of merger . . . 20
2.2 Detailed description of previous NTRU variants . . . 21
2.2.1 The ANTS'98 NTRU PPKE . . . 21
2.2.2 The ANTS'98 NTRU DPKE . . . 21
2.2.3 The first round NTRUEncrypt submission . . . 21
2.2.4 The first round NTRU-HRSS-KEM submission . . . 23
2.2.5 The Saito–Xagawa–Yamakawa variant of NTRU-HRSS-KEM . . . 24
2.3 The NTRU submission . . . 25
2.4 Variants of the NTRU submission . . . 26
2.4.1 Faster key generation for single-use keys . . . 26
2.4.2 Prime q . . . 26
2.4.3 NTRU-HPS-like parameter sets with faster key generation . . . 26
2.4.4 Arbitrary weight $\mathbf{m}$ and fixed-weight $\mathbf{f}$ . . . 27
2.4.5 An IND-CCA2 PKE using Q-OAEP . . . 27
2.5 Available size vs. security trade-offs . . . 27
2.6 Parameter selection . . . 27

3 Performance analysis **29**
3.1 Description of platform . . . 29
3.2 Performance of reference and AVX2 implementations . . . 29
3.3 Memory usage . . . 29

4 Known Answer Test values **30**

5 Expected security **30**
5.1 Security definition for key-establishment . . . 30
5.2 Security definition for ephemeral-only key-establishment . . . 30
5.3 Security categories . . . 31

6 Cost of known attacks **31**
6.1 Attacks based on lattices . . . 32
6.2 Quality of lattice reduction . . . 32
6.3 Cost of SVP algorithms . . . 33
6.3.1 Effect of quantum search . . . 33
6.4 The cost of lattice attacks . . . 34
6.4.1 Short vectors in NTRU lattices . . . 34
6.4.2 Costing the primal attack . . . 34
6.4.3 Costing the hybrid attack . . . 34
6.5 Rationale for security categories . . . 35

7 Advantages and limitations **35**

1 Written specification

1.1 Overview

This document specifies a key encapsulation mechanism (KEM) based on Hoffstein, Pipher, and Silverman's NTRU encryption scheme [19, 20]. The KEM is constructed using a generic transformation from a correct deterministic public key encryption scheme (correct DPKE). NTRU was originally described as a partially correct probabilistic public key encryption scheme (partially correct PPKE), and most instantiations in the literature are based on this PPKE (e.g. [9, 26, 24, 17, 18, 25]). However, a preprint of the NTRU paper circulated at CRYPTO'96 [19] describes how NTRU can be made both deterministic [19, Section 4.2] and perfectly correct [19, Section 4.3]. Modulo a few small changes introduced by Hülsing, Rijnveld, Schanck, and Schwabe in [25], the correct DPKE that we describe here is obtained by applying the preprint's transformations for determinism and correctness to the PPKE from ANTS'98 [20].

The DPKE is parameterized by coprime positive integers (n, p, q), sample spaces $(\mathcal{L}_f, \mathcal{L}_g, \mathcal{L}_r, \mathcal{L}_m)$, and an injection $\mathsf{Lift} : \mathcal{L}_m \to \mathbb{Z}[\mathbf{x}]$. We recommend two narrowly defined families of parameter sets that we refer to as NTRU-HPS and NTRU-HRSS. The NTRU-HPS parameter sets follow Hoffstein, Pipher, and Silverman's use of *fixed-weight* sample spaces [19, 20] and allow several choices of q for each n. The NTRU-HRSS parameter sets follow Hülsing, Rijnveld, Schanck, and Schwabe's use of *arbitrary weight* sample spaces [25] and fix q as a function of n.

This submission is a merger of the NTRUEncrypt and NTRU-HRSS-KEM submissions. We have unified all aspects of the designs except for the use of fixed-weight sampling. In that regard, our NTRU-HPS parameter sets follow the NTRUEncrypt submission, and our NTRU-HRSS parameter sets follow the NTRU-HRSS-KEM submission. We continue to recommend ntruhrss701 (NTRU-HRSS with $n = 701$), which was the only parameter set recommended in the NTRU-HRSS-KEM submission. The move toward perfect correctness forces us to deprecate the parameter sets recommended in the NTRU-Encrypt submission. We have selected ntruhps2048509 (NTRU-HPS with $n = 509$ and $q = 2048$) and ntruhps4096821 (NTRU-HPS with $n = 821$ and $q = 4096$) to replace the NTRUEncrypt submission's ntru-pke-443 and ntru-pke-743 parameter sets. We have also selected ntruhps2048677 (NTRU-HPS with $n = 677$ and $q = 2048$) as an alternative to ntruhrss701.

The KEM that we construct has a tight proof of IND-CCA2 security in the random oracle model (ROM) under the assumption that our DPKE is OW-CPA secure. It also has a tight proof of IND-CCA2 security in the *quantum accessible* random oracle model (QROM) under a non-standard assumption stated by Saito, Xagawa, and Yamakawa [35]. Our KEM is interoperable with the KEM constructed by Saito, Xagawa, and Yamakawa in [35, Section 5.1], but it can also be viewed as an application of the $U^{\not\perp}_m$ transformation of Hofheinz, Hövelmanns, and Kiltz [21], or of the SimpleKEM transformation of Bernstein and Persichetti [5]. This is because our DPKE is slightly different from the NTRU DPKE proposed by Saito, Xagawa, and Yamakawa ([35, Figure 10]). Our DPKE achieves Bernstein and Persichetti's notion of *rigidity* [5, Section 6] without applying "re-encryption." This change affects the internal behavior of the KEM, but the result remains interoperable with the Saito–Xagawa–Yamakawa NTRU KEM.

1.2 Definitions

The following definitions are with respect to a fixed odd prime n.

1. $(\mathbb{Z}/n)^\times$ is the multiplicative group of integers modulo n.
2. $\mathbf{\Phi}_1$ is the polynomial $(\mathbf{x} - 1)$.
3. $\mathbf{\Phi}_n$ is the polynomial $(\mathbf{x}^n - 1)/(\mathbf{x} - 1) = \mathbf{x}^{n-1} + \mathbf{x}^{n-2} + \cdots + 1$.
4. R is the quotient ring $\mathbb{Z}[\mathbf{x}]/(\mathbf{\Phi}_1\mathbf{\Phi}_n)$.
5. S is the quotient ring $\mathbb{Z}[\mathbf{x}]/(\mathbf{\Phi}_n)$.
6. $R/3$ is the quotient ring $\mathbb{Z}[\mathbf{x}]/(3, \mathbf{\Phi}_1\mathbf{\Phi}_n)$.

7. R/q is the quotient ring $\mathbb{Z}[\mathbf{x}]/(q, \mathbf{\Phi}_1\mathbf{\Phi}_n)$. The *canonical R/q-representative* of $\mathbf{a} \in \mathbb{Z}[\mathbf{x}]$ is the unique polynomial $\mathbf{b} \in \mathbb{Z}[\mathbf{x}]$ of degree at most $n-1$ with coefficients in $\{-q/2, -q/2+1, \ldots, q/2-1\}$ such that $\mathbf{a} \equiv \mathbf{b} \pmod{(q, \mathbf{\Phi}_1\mathbf{\Phi}_n)}$. We write $\underline{\mathsf{Rq}}(\mathbf{a})$ for the canonical R/q-representative of $\mathbf{a}$. We write $\mathsf{Rq}(\mathbf{a})$ when the choice of representative is not normative.

8. $S/2$ is the quotient ring $\mathbb{Z}[\mathbf{x}]/(2, \mathbf{\Phi}_n)$. The *canonical $S/2$-representative* of $\mathbf{a} \in \mathbb{Z}[\mathbf{x}]$ is the unique polynomial $\mathbf{b} \in \mathbb{Z}[\mathbf{x}]$ of degree at most $n-2$ with coefficients in $\{0, 1\}$ such that $\mathbf{a} \equiv \mathbf{b} \pmod{(2, \mathbf{\Phi}_n)}$. We write $\underline{\mathsf{S2}}(\mathbf{a})$ for the canonical $S/2$-representative of $\mathbf{a}$. We write $\mathsf{S2}(\mathbf{a})$ when the choice of representative is not normative.

9. $S/3$ is the quotient ring $\mathbb{Z}[\mathbf{x}]/(3, \mathbf{\Phi}_n)$. The *canonical $S/3$-representative* of $\mathbf{a} \in \mathbb{Z}[\mathbf{x}]$ is the unique polynomial $\mathbf{b} \in \mathbb{Z}[\mathbf{x}]$ of degree at most $n-2$ with coefficients in $\{-1, 0, 1\}$ such that $\mathbf{a} \equiv \mathbf{b} \pmod{(3, \mathbf{\Phi}_n)}$. We write $\underline{\mathsf{S3}}(\mathbf{a})$ for the canonical $S/3$-representative of $\mathbf{a}$. We write $\mathsf{S3}(\mathbf{a})$ when the choice of representative is not normative.

10. S/q is the quotient ring $\mathbb{Z}[\mathbf{x}]/(q, \mathbf{\Phi}_n)$. The *canonical S/q-representative* of $\mathbf{a} \in \mathbb{Z}[\mathbf{x}]$ is the unique polynomial $\mathbf{b} \in \mathbb{Z}[\mathbf{x}]$ of degree at most $n-2$ with coefficients in $\{-q/2, -q/2+1, \ldots, q/2-1\}$ such that $\mathbf{a} \equiv \mathbf{b} \pmod{(q, \mathbf{\Phi}_n)}$. We write $\underline{\mathsf{Sq}}(\mathbf{a})$ for the canonical S/q-representative of $\mathbf{a}$. We write $\mathsf{Sq}(\mathbf{a})$ when the choice of representative is not normative.

11. A polynomial is *ternary* if its coefficients are in $\{-1, 0, 1\}$.

12. A ternary polynomial $\mathbf{v} = \sum_i v_i \mathbf{x}^i$ has the *non-negative correlation* property if $\sum_i v_i v_{i+1} \geq 0$.

13. $\mathcal{T}$ is the set of non-zero ternary polynomials of degree at most $n-2$. Equivalently, $\mathcal{T}$ is the set of canonical $S/3$-representatives.

14. $\mathcal{T}_+$ is the subset of $\mathcal{T}$ consisting of polynomials with the non-negative correlation property.

15. $\mathcal{T}(d)$, for an even positive integer d, is the subset of $\mathcal{T}$ consisting of polynomials that have exactly $d/2$ coefficients equal to $+1$ and $d/2$ coefficients equal to -1.

1.3 Parameter sets

1.3.1 NTRU

An NTRU parameter set is $(n, p, q, \mathcal{L}_f, \mathcal{L}_g, \mathcal{L}_r, \mathcal{L}_m, \mathsf{Lift})$ where n, p, and q are coprime positive integers; $\mathcal{L}_f$, $\mathcal{L}_g$, $\mathcal{L}_r$, and $\mathcal{L}_m$ are sets of integer polynomials; and Lift is an injection $\mathcal{L}_m \to \mathbb{Z}[\mathbf{x}]$ for which $\underline{\mathsf{S3}}(\mathsf{Lift}(\mathbf{m})) = \mathbf{m}$ for all $\mathbf{m} \in \mathcal{L}_m$. An NTRU parameter set is *correct* if

$$(p \cdot \mathbf{r} \cdot \mathbf{g} + \mathbf{f} \cdot \mathsf{Lift}(\mathbf{m})) \bmod (\mathbf{\Phi}_1\mathbf{\Phi}_n) \tag{1}$$

has coefficients in $\{-q/2, \ldots, q/2-1\}$ for all $(\mathbf{f}, \mathbf{g}, \mathbf{r}, \mathbf{m}) \in (\mathcal{L}_f, \mathcal{L}_g, \mathcal{L}_r, \mathcal{L}_m)$.

1.3.2 NTRU-HPS

An NTRU-HPS parameter set is an NTRU parameter set for which

- n is a prime and both 2 and 3 are of order $n-1$ in $(\mathbb{Z}/n)^\times$,
- $p = 3$,
- q is a power of two,
- $\mathcal{L}_f = \mathcal{T}$,
- $\mathcal{L}_g = \mathcal{T}(q/8-2)$,
- $\mathcal{L}_r = \mathcal{T}$,
- $\mathcal{L}_m = \mathcal{T}(q/8-2)$, and
- Lift is the identity $\mathbf{m} \mapsto \mathbf{m}$.

We only recommend parameter sets with $q/8 - 2 \leq 2n/3$. Parameter sets with larger q may replace the $q/8 - 2$ in the definition of $\mathcal{L}_g$ and $\mathcal{L}_m$ with $2\lfloor n/3 \rfloor$. In either case, the parameters are correct; each coefficient in Eq. 1 is a sum of at most $q/8 - 2$ terms in $\{0, \pm 1, \pm 2, \pm 3, \pm 4\}$. Specific NTRU-HPS parameter sets are denoted ntruhps[q][n], e.g. ntruhps2048509 is NTRU-HPS with $n = 509$ and $q = 2048$. The recommended NTRU-HPS parameter sets are ntruhps2048509, ntruhps2048677, ntruhps4096821.

1.3.3 NTRU-HRSS

An NTRU-HRSS parameter set is an NTRU parameter set for which

- n is a prime and both 2 and 3 are of order $n - 1$ in $(\mathbb{Z}/n)^\times$,
- $p = 3$,
- $q = 2^{\lceil 7/2 + \log_2(n) \rceil}$,
- $\mathcal{L}_f = \mathcal{T}_+$,
- $\mathcal{L}_g = \{\mathbf{\Phi}_1 \cdot \mathbf{v} \ : \ \mathbf{v} \in \mathcal{T}_+\}$,
- $\mathcal{L}_r = \mathcal{T}$,
- $\mathcal{L}_m = \mathcal{T}$, and
- Lift is $\mathbf{m} \mapsto \mathbf{\Phi}_1 \cdot \underline{\mathsf{S3}}(\mathbf{m}/\mathbf{\Phi}_1)$.

These parameters are correct for any choice of $q > 8\sqrt{2}(n-1)$ [25]. The q recommended here is the smallest power of two that provides correctness. Specific NTRU-HRSS parameter sets are denoted ntruhrss[n], e.g. ntruhrss701 is NTRU-HRSS with $n = 701$. The recommended NTRU-HRSS parameter set is ntruhrss701.

1.4 Additional parameters

1.4.1 Hash

A hash function.

Recommended value: SHA3_256

1.4.2 Sample_fg

A routine for sampling from $\mathcal{L}_f \times \mathcal{L}_g$.

Recommended value: The routine of Section 1.10.1.

Note: Sample_fg is listed as a parameter because the use of different routines will lead to different known answer test results. The choice of Sample_fg does not affect interoperability of otherwise identical parameter sets. The choice of Sample_fg may affect the derived constants sample_iid_bits, sample_fixed_type_bits, and sample_key_bits. The choice may also affect security, see Section 6.4.1.

1.4.3 Sample_rm

A routine for sampling from $\mathcal{L}_r \times \mathcal{L}_m$.

Recommended value: The routine of Section 1.10.2.

Note: Sample_rm is listed as a parameter because the use of different routines will lead to different known answer test results. The choice of Sample_rm does not affect interoperability of otherwise identical parameter sets. The choice of Sample_fg may affect the derived constants sample_iid_bits, sample_fixed_type_bits, and sample_plaintext_bits. The choice may also affect security, see Section 6.4.1.

1.4.4 key_seed_bits

The number of random bits consumed by KeyGen. [1.12.1]

Recommended value: $\mathsf{sample\_key\_bits} + \mathsf{prf\_key\_bits}$

1.5 Derived constants

1.5.1 logq

Formula: $\log_2(q)$

1.5.2 sample_iid_bits

The number of random bits consumed by the Ternary routine. [1.10.3]

Formula: $8 \cdot (n-1)$

1.5.3 sample_fixed_type_bits

The number of random bits consumed by the Fixed_Type routine. [1.10.5]

Formula: $30 \cdot (n-1)$

1.5.4 sample_key_bits

The number of random bits consumed by the Sample_fg routine. [1.10.1]

Formula: $\begin{cases} \mathsf{sample\_iid\_bits} + \mathsf{sample\_iid\_bits} & (\text{NTRU-HRSS}) \\ \mathsf{sample\_iid\_bits} + \mathsf{sample\_fixed\_type\_bits} & (\text{NTRU-HPS}) \end{cases}.$

1.5.5 sample_plaintext_bits

The number of random bits consumed by the Sample_rm routine. [1.10.2]

Formula: $\begin{cases} \mathsf{sample\_iid\_bits} + \mathsf{sample\_iid\_bits} & (\text{NTRU-HRSS}) \\ \mathsf{sample\_iid\_bits} + \mathsf{sample\_fixed\_type\_bits} & (\text{NTRU-HPS}) \end{cases}.$

1.5.6 packed_s3_bytes

The number of bytes output by pack_S3. [1.8.7]

Formula: $\lceil (n-1)/5 \rceil$

1.5.7 packed_sq_bytes

The number of bytes output by pack_Sq. [1.8.5]

Formula: $\lceil (n-1) \cdot \mathsf{logq}/8 \rceil$

1.5.8 packed_rq0_bytes

The number of bytes output by pack_Rq0. [1.8.3]

Formula: $\lceil (n-1) \cdot \mathsf{logq}/8 \rceil$

1.5.9 dpke_public_key_bytes

The number of bytes in a public key for the DPKE.

Formula: packed_rq0_bytes

1.5.10 dpke_private_key_bytes

The number of bytes in a private key for the DPKE.

Formula: $2 \cdot \mathsf{packed\_s3\_bytes} + \mathsf{packed\_sq\_bytes}$

1.5.11 dpke_plaintext_bytes

The number of bytes in a plaintext for the DPKE.

Formula: $2 \cdot \mathsf{packed\_s3\_bytes}$

1.5.12 dpke_ciphertext_bytes

The number of bytes in a ciphertext for the DPKE.

Formula: packed_rq0_bytes

1.5.13 kem_public_key_bytes

The number of bytes in a public key for the KEM.

Formula: dpke_public_key_bytes

1.5.14 kem_private_key_bytes

The number of bytes in a private key for the KEM.

Formula: $\mathsf{dpke\_private\_key\_bytes} + \lceil \mathsf{prf\_key\_bits}/8 \rceil$

1.5.15 kem_ciphertext_bytes

The number of bytes in a ciphertext for the KEM.

Formula: dpke_ciphertext_bytes

1.5.16 kem_shared_key_bits

The number of bits output by Hash.

Recommended value: 256

1.5.17 prf_key_bits

The number of bits used to key the implicit rejection mechanism.

Formula: 256

1.6 Summary of recommended parameters and derived constants

	ntruhps2048509	ntruhps2048677	ntruhps4096821	ntruhrss701
n	509	677	821	701
q	2048	2048	4096	8192
Hash	SHA3_256	SHA3_256	SHA3_256	SHA3_256
Sample_fg	[1.10.1]	[1.10.1]	[1.10.1]	[1.10.1]
Sample_rm	[1.10.2]	[1.10.2]	[1.10.2]	[1.10.2]
sample_fixed_type_bits	15240	20280	24600	—
sample_iid_bits	4064	5408	6560	5600
sample_key_bits	19304	25688	31160	11200
sample_plaintext_bits	19304	25688	31160	11200
packed_s3_bytes	102	136	164	140
packed_rq0_bytes	699	930	1230	1138
packed_sq_bytes	699	930	1230	1138
dpke_public_key_bytes	699	930	1230	1138
dpke_private_key_bytes	903	1202	1558	1418
dpke_plaintext_bytes	204	272	328	280
dpke_ciphertext_bytes	699	930	1230	1138
kem_public_key_bytes	699	930	1230	1138
kem_private_key_bytes	935	1234	1590	1450
kem_ciphertext_bytes	699	930	1230	1138
kem_shared_key_bits	256	256	256	256

1.7 Externally defined algorithms

1.7.1 SHAKE256

Input:

- A bit string M of arbitrary length.
- A positive integer d.

Output:

- A bit string of length d.

Operation:

1. Output KECCAK$[512](M||1111, d)$, as defined in [33].

1.7.2 SHA3_256

Input:

- A bit string M of arbitrary length.

Output:

- A bit string of length 256.

Operation:

1. Output KECCAK$[512](M||01, 256)$, as defined in [33].

1.8 Encodings

1.8.1 Bit strings and byte arrays

A bit string is an element of $\{0,1\}^*$. A byte array is an element of $(\{0,1\}^8)^*$. The public API is defined in terms of byte arrays. However, the externally defined functions SHA3_256 and SHAKE256operate on bit strings, as do some internal functions. We define $\mathsf{bits\_to\_bytes}(b)$ and $\mathsf{bytes\_to\_bits}(B, \ell)$ to handle the conversions. When converting a bit string to a byte array the bit string is right padded with zeros until its length is a multiple of 8. Bytes are then formed by bracketing, and the order of the bits within each byte is reversed. For example, the bit string $(b_1, \ldots, b_7, b_8, b_9, \ldots, b_{13})$ is encoded as $\mathsf{bits\_to\_bytes}((b_1, \ldots, b_{13})) = ((b_8, b_7, \ldots, b_1), (0, 0, 0, b_{13}, \ldots, b_9))$. The inverse procedure takes a length parameter: $\mathsf{bytes\_to\_bits}(((b_8, b_7, \ldots, b_1), (0, 0, 0, b_{13}, \ldots, b_9)), 13) = (b_1, \ldots, b_7, b_8, b_9, \ldots, b_{13})$.

1.8.2 Polynomials

In this document polynomials are treated as zero indexed arrays. We write v_i for the coefficient of $\mathbf{x}^i$ in $\mathbf{v}$. Implementations are free to choose their internal representation of polynomials. Only the encoding of polynomials into byte arrays is normative.

1.8.3 pack_Rq0

Input:

- A polynomial $\mathbf{a}$ that satisfies $\mathbf{a} \equiv 0 \pmod{(q, \mathbf{\Phi}_1)}$.

Output:

- A byte array of length packed_rq0_bytes that encodes the first $n-1$ coefficients of $\underline{\mathsf{Rq}}(\mathbf{a})$.

Operation:

1. Set $\mathbf{v} = \underline{\mathsf{Rq}}(\mathbf{a})$
2. Set $(b_1, b_2, \ldots, b_{(n-1)\mathsf{logq}}) = (0, 0, \ldots, 0)$
3. Set $i = 0$
4. While $i < (n-1)$
5. Set $(b_{i\mathsf{logq}+1}, b_{i\mathsf{logq}+2}, \ldots, b_{i\mathsf{logq}+\mathsf{logq}}) \in \{0,1\}^{\mathsf{logq}}$ such that $\sum_{j=0}^{\mathsf{logq}-1} 2^j b_{i\mathsf{logq}+1+j} \equiv v_i \pmod{q}$
6. Set $i = i + 1$
7. End
8. Output $\mathsf{bits\_to\_bytes}((b_1, b_2, \ldots, b_{(n-1)\mathsf{logq}}))$

Notes:

1. The coefficient v_{n-1} is not encoded. The condition $\mathbf{a} \equiv 0 \pmod{(q, \mathbf{\Phi}_1)}$ implies that $v_{n-1} \equiv -\sum_{i=0}^{n-2} v_i \pmod{q}$, so v_{n-1} can be recovered from the first $n-1$ coefficients.

1.8.4 unpack_Rq0

Input:

- A byte array B of length packed_rq0_bytes.

Output:

- A polynomial $\mathbf{a}$ that satisfies $\mathbf{a} \equiv 0 \pmod{(q, \mathbf{\Phi}_1)}$.

Operation:

1. Set $(b_1, b_2, \ldots, b_{(n-1)\mathsf{logq}}) = \mathsf{bytes\_to\_bits}(B, (n-1)\mathsf{logq})$

2. Set $\mathbf{v} = 0$
3. Set $i = 0$
4. While $i < (n-1)$
5. Set $c = \sum_{j=0}^{\mathsf{logq}-1} 2^j b_{i\mathsf{logq}+1+j}$
6. Set $\mathbf{v} = \mathbf{v} + c \cdot \mathbf{x}^i - c \cdot \mathbf{x}^{n-1}$
7. Set $i = i + 1$
8. End
9. Output $\mathsf{Rq}(\mathbf{v})$

1.8.5 pack_Sq

Input:

– A polynomial $\mathbf{a}$.

Output:

– A byte array of length packed_sq_bytes that encodes $\underline{\mathsf{Sq}}(\mathbf{a})$.

Operation:

1. Set $\mathbf{v} = \underline{\mathsf{Sq}}(\mathbf{a})$
2. Set $(b_1, b_2, \ldots, b_{(n-1)\mathsf{logq}}) = (0, 0, \ldots, 0)$
3. Set $i = 0$
4. While $i < (n-1)$
5. Set $(b_{i\mathsf{logq}+1}, b_{i\mathsf{logq}+2}, \ldots, b_{i\mathsf{logq}+\mathsf{logq}})$ such that $\sum_{j=0}^{\mathsf{logq}-1} 2^j b_{i\mathsf{logq}+1+j} \equiv v_i \pmod q$
6. Set $i = i + 1$
7. End
8. Output bits_to_bytes$((b_1, b_2, \ldots, b_{(n-1)\mathsf{logq}}))$.

1.8.6 unpack_Sq

Input:

– A byte array B of length packed_sq_bytes.

Output:

– A polynomial.

Operation:

1. Set $(b_1, b_2, \ldots, b_{(n-1)\mathsf{logq}}) =$ bytes_to_bits$(B, (n-1)\mathsf{logq})$
2. Set $\mathbf{v} = 0$
3. Set $i = 0$
4. While $i < (n-1)$
5. Set $c = \sum_{j=0}^{\mathsf{logq}-1} 2^j b_{i\mathsf{logq}+1+j}$
6. Set $\mathbf{v} = \mathbf{v} + c \cdot \mathbf{x}^i$
7. Set $i = i + 1$
8. End
9. Output $\mathsf{Sq}(\mathbf{v})$

1.8.7 pack_S3

Input:

– A polynomial $\mathbf{a}$.

Output:

– A byte array of length packed_s3_bytes that encodes $\underline{\mathsf{S3}}(\mathbf{a})$.

Operation:

1. Set $\mathbf{v} = \underline{\mathsf{S3}}(\mathbf{a})$
2. Set $(b_1, b_2, \ldots, b_{8\lceil(n-1)/5\rceil}) = (0, 0, \ldots, 0)$
3. Set $i = 0$
4. While $i < \lceil(n-1)/5\rceil$
5. Set $(c_1, c_2, \ldots, c_5) \in \{0, 1, 2\}^5$ so that $c_j \equiv v_{5i+j} \pmod 3$
6. Set $(b_{8i+1}, b_{8i+2}, \ldots, b_{8i+8})$ so that $\sum_{j=0}^{7} 2^j b_{8i+1+j} = \sum_{j=0}^{4} 3^j c_{1+j}$
7. $i = i + 1$
8. End
9. Output bits_to_bytes$((b_1, b_2, \ldots, b_{8\lceil(n-1)/5\rceil}))$

1.8.8 unpack_S3

Input:

– A byte array B of length packed_s3_bytes.

Output:

– A polynomial.

Operation:

1. Set $(b_1, b_2, \ldots, b_{8\lceil(n-1)/5\rceil}) =$ bytes_to_bits$(B, 8\lceil(n-1)/5\rceil)$
2. Set $\mathbf{v} = 0$
3. Set $i = 0$
4. While $i < \lceil(n-1)/5\rceil$
5. Set $(c_1, c_2, \ldots, c_5) \in \{0, 1, 2\}^5$ so that $\sum_{j=0}^{7} 2^j b_{8i+1+j} = \sum_{j=0}^{4} 3^j c_{1+j}$.
6. Set $(v_{5i+1}, v_{5i+2}, v_{5i+3}, v_{5i+4}, v_{5i+5}) = (c_1, c_2, c_3, c_4, c_5)$
7. $i = i + 1$
8. End
9. Output S3$(\mathbf{v})$

1.9 Arithmetic

Algorithms for integer addition, integer multiplication, polynomial addition, polynomial multiplication, modular reduction (Rq, S2, S3, Sq), and canonical representatives ($\underline{\mathsf{Rq}}$, $\underline{\mathsf{S3}}$, $\underline{\mathsf{Sq}}$) are omitted.

1.9.1 S2_inverse and S3_inverse

The conditions on n in the definition of NTRU-HPS and NTRU-HRSS ensure that $S/2$ and $S/3$ are finite fields. The routines S2_inverse and S3_inverse compute inverses in $S/2$ and $S/3$ respectively. Implementing these routines in constant time is non-trivial. Pseudocode for one method is provided in [25]. A faster S3_inverse is described in [6].

1.9.2 Sq_inverse

Input:

- A polynomial $\mathbf{a}$.

Output:

- A polynomial $\mathbf{b}$ that satisfies $\underline{\mathsf{Sq}}(\mathbf{a} \cdot \mathbf{b}) = 1$.

Operation:

1. Set $\mathbf{v}_0 = \underline{\mathsf{S2}}(\mathsf{S2\_inverse}(\mathbf{a}))$ [1.9.1]
2. Set $t = 1$
3. While $t < \mathsf{logq}$
4. Set $\mathbf{v}_0 = \mathsf{Sq}(\mathbf{v}_0 \cdot (2 - \mathbf{a} \cdot \mathbf{v}_0))$
5. Set $t = 2t$
6. End
7. Output $\mathsf{Sq}(\mathbf{v}_0)$

Notes:

1. Line 4 can be performed in R/q.

1.9.3 Lift

Input:

- A polynomial $\mathbf{m}$.

Output:

- (NTRU-HPS) The polynomial $\underline{\mathsf{S3}}(\mathbf{m})$.
- (NTRU-HRSS) The polynomial $\mathbf{\Phi}_1 \cdot \underline{\mathsf{S3}}(\mathbf{m}/\mathbf{\Phi}_1)$.

Notes:

1. The ternary polynomial $\underline{\mathsf{S3}}(1/\mathbf{\Phi}_1)$ has periodic coefficients. Explicitly,

$$\underline{\mathsf{S3}}(1/\mathbf{\Phi}_1) = \begin{cases} \underline{\mathsf{S3}}\left(\sum_{i=0}^{n-2} i \cdot \mathbf{x}^i\right) & \text{if } n \equiv 1 \bmod 3, \\ \underline{\mathsf{S3}}\left(\sum_{i=0}^{n-2} (1-i) \cdot \mathbf{x}^i\right) & \text{if } n \equiv 2 \bmod 3. \end{cases}$$

This leads to a fast algorithm for computing $\underline{\mathsf{S3}}(\mathbf{m}/\mathbf{\Phi}_1)$; pseudocode is given in [25].

1.10 Sampling

1.10.1 Sample_fg

Input:

- A bit string fg_bits of length $\mathsf{sample\_key\_bits}$.

Output:

- A polynomial $\mathbf{f}$ in $\mathcal{L}_f$.
- A polynomial $\mathbf{g}$ in $\mathcal{L}_g$.

Operation:

- NTRU-HPS
  1. Parse fg_bits as $f\_bits \parallel g\_bits$ with
     - f_bits of length sample_iid_bits
     - g_bits of length sample_fixed_type_bits
  2. Set $\mathbf{f} = \mathsf{Ternary}(f\_bits)$ [1.10.3]
  3. Set $\mathbf{g} = \mathsf{Fixed\_Type}(g\_bits)$ [1.10.5]
- NTRU-HRSS
  1. Parse fg_bits as $f\_bits \parallel g\_bits$ with
     - f_bits of length sample_iid_bits
     - g_bits of length sample_iid_bits
  2. Set $\mathbf{f} = \mathsf{Ternary\_Plus}(f\_bits)$ [1.10.4]
  3. Set $\mathbf{g}_0 = \mathsf{Ternary\_Plus}(g\_bits)$ [1.10.4]
  4. Set $\mathbf{g} = \mathbf{\Phi}_1 \cdot \mathbf{g}_0$

Notes:

1. Our recommended Ternary and Ternary_Plus routines consume sample_iid_bits $= 8n-8$ bits, so g_bits starts at a byte boundary.

1.10.2 Sample_rm

Input:

- A bit string rm_bits of length sample_plaintext_bits.

Output:

- A polynomial $\mathbf{r} \in \mathcal{L}_r$.
- A polynomial $\mathbf{m} \in \mathcal{L}_m$.

Operation:

- NTRU-HPS
  1. Parse rm_bits as $r\_bits \parallel m\_bits$ with
     - r_bits of length sample_iid_bits
     - m_bits of length sample_fixed_type_bits
  2. Set $\mathbf{r} = \mathsf{Ternary}(r\_bits)$ [1.10.3]
  3. Set $\mathbf{m} = \mathsf{Fixed\_Type}(m\_bits)$ [1.10.5]
  4. Output $(\mathbf{r}, \mathbf{m})$
- NTRU-HRSS
  1. Parse rm_bits as $r\_bits \parallel m\_bits$ with
     - r_bits of length sample_iid_bits
     - m_bits of length sample_iid_bits
  2. Set $\mathbf{r} = \mathsf{Ternary}(r\_bits)$ [1.10.3]
  3. Set $\mathbf{m} = \mathsf{Ternary}(m\_bits)$ [1.10.3]
  4. Output $(\mathbf{r}, \mathbf{m})$

1.10.3 Ternary

Input:

– A bit string $(b_1, b_2, \cdots, b_\ell)$ of length sample_iid_bits.

Output:

– A ternary polynomial.

Operation:

1. Set $\mathbf{v} = 0$
2. Set $i = 0$
3. While $i < n - 1$
4. Set $\mathbf{v} = \mathbf{v} + \left(\sum_{j=0}^{7} 2^j b_{8i+j+1}\right) \cdot \mathbf{x}^i$
5. Set $i = i + 1$
6. End
7. Output S3($\mathbf{v}$)

Notes:

1. This implementation assumes sample_iid_bits $= 8 \cdot (n - 1)$.

1.10.4 Ternary_Plus

Input:

– A bit string $(b_1, b_2, \cdots, b_\ell)$ of length sample_iid_bits.

Output:

– A ternary polynomial that satisfies the non-negative correlation property.

Operation:

1. Set $\mathbf{v} = \text{Ternary}((b_1, b_2, \cdots, b_\ell))$ [1.10.3]
2. Set $t = \sum_{i=0}^{n-2} v_i \cdot v_{i+1}$
3. Set $s = -1$ if $t < 0$, otherwise set $s = 1$
4. Set $i = 0$
5. While $i < n - 1$
6. Set $v_i = s \cdot v_i$
7. Set $i = i + 2$
8. End
9. Output S3($\mathbf{v}$)

Notes:

1. The value t in Line 2 satisfies $-n < t < n$.

1.10.5 Fixed_Type

Input:

– A bit string $(b_1, b_2, \cdots, b_\ell)$ of length sample_fixed_type_bits.

Output:

– A ternary polynomial with exactly $q/16-1$ coefficients equal to 1 and $q/16-1$ coefficients equal to -1.

Operation:

1. Set $A = [0, 0, \ldots, 0]$ (the zero array of length $n-1$)
2. Set $\mathbf{v} = 0$ (the zero polynomial)
3. Set $i = 0$
4. While $i < q/16 - 1$
5.   Set $A_i = 1 + \sum_{j=0}^{29} 2^{2+j} b_{30i+1+j}$
6.   Set $i = i + 1$
7. End
8. While $i < q/8 - 2$
9.   Set $A_i = 2 + \sum_{j=0}^{29} 2^{2+j} b_{30i+1+j}$
10.   Set $i = i + 1$
11. End
12. While $i < n - 1$
13.   Set $A_i = 0 + \sum_{j=0}^{29} 2^{2+j} b_{30i+1+j}$
14.   Set $i = i + 1$
15. End
16. Sort A
17. Set $i = 0$
18. While $i < n - 1$
19.   Set $\mathbf{v} = \mathbf{v} + (A_i \bmod 4)\mathbf{x}^i$
20.   Set $i = i + 1$
21. End
22. Output S3($\mathbf{v}$)

Notes:

1. This implementation assumes sample_fixed_type_bits $= 30 \cdot (n-1)$.
2. Sorting must be implemented in constant time.

1.11 Passively secure DPKE

1.11.1 DPKE_Key_Pair

Input:

– A bit string *coins* of length sample_key_bits

Output:

– A byte array *packed_private_key* of length dpke_private_key_bytes

– A byte array *packed_public_key* of length dpke_public_key_bytes

Operation:

1. Set $(\mathbf{f}, \mathbf{g}) = \mathsf{Sample\_fg}(coins)$ [1.10.1]
2. Set $\mathbf{f}_p = \mathsf{S3\_inverse}(\mathbf{f})$ [1.9.1]
3. Set $(\mathbf{h}, \mathbf{h}_q) = \mathsf{DPKE\_Public\_Key}(\mathbf{f}, \mathbf{g})$ [1.11.2]
4. Set $packed\_private\_key = \mathsf{pack\_S3}(\mathbf{f}) \parallel \mathsf{pack\_S3}(\mathbf{f}_p) \parallel \mathsf{pack\_Sq}(\mathbf{h}_q)$ [1.8.7, 1.8.5]
5. Set $packed\_public\_key = \mathsf{pack\_Rq0}(\mathbf{h})$ [1.8.3]
6. Output $(packed\_private\_key, packed\_public\_key)$

1.11.2 DPKE_Public_Key

Input:

– A polynomial $\mathbf{f} \in \mathcal{L}_f$

– A polynomial $\mathbf{g} \in \mathcal{L}_g$

Output:

– A polynomial $\mathbf{h}$ that satisfies $\underline{\mathsf{Rq}}(\mathbf{h} \cdot \mathbf{f}) = 3 \cdot \mathbf{g}$

– An polynomial $\mathbf{h}_q$ that satisfies $\underline{\mathsf{Sq}}(\mathbf{h} \cdot \mathbf{h}_q) = 1$

Operation:

1. Set $\mathbf{G} = 3 \cdot \mathbf{g}$
2. Set $\mathbf{v}_0 = \mathsf{Sq}(\mathbf{G} \cdot \mathbf{f})$
3. Set $\mathbf{v}_1 = \mathsf{Sq\_inverse}(\mathbf{v}_0)$ [1.9.2]
4. Set $\mathbf{h} = \mathsf{Rq}(\mathbf{v}_1 \cdot \mathbf{G} \cdot \mathbf{G})$
5. Set $\mathbf{h}_q = \mathsf{Rq}(\mathbf{v}_1 \cdot \mathbf{f} \cdot \mathbf{f})$
6. Output $(\mathbf{h}, \mathbf{h}_q)$

Notes:

1. The choice of $\mathcal{L}_g$ in NTRU-HPS and NTRU-HRSS ensures that $\mathbf{G} \equiv 0 \pmod{(q, \mathbf{\Phi}_1)}$. As a consequence, the output condition on $\mathbf{h}$ is satisfied even though the inverse is computed in S/q instead of R/q.

1.11.3 DPKE_Encrypt

Input:

– A byte array *packed_public_key* of length dpke_public_key_bytes.

– A byte array *packed_rm* of length dpke_plaintext_bytes.

Output:

– A byte array *packed_ciphertext* of length dpke_ciphertext_bytes.

Operation:

1. Parse *packed_rm* as $packed\_r \parallel packed\_m$ with
   – *packed_r* of length packed_s3_bytes, and
   – *packed_m* of length packed_s3_bytes.
2. Set $\mathbf{r} = \underline{\mathsf{S3}}(\mathsf{unpack\_S3}(packed\_r))$ [1.10.3]

3. Set $\mathbf{m}_0 = \mathsf{unpack\_S3}(packed\_m)$ [1.8.8]
4. Set $\mathbf{m}_1 = \mathsf{Lift}(\mathbf{m}_0)$ [1.9.3]
5. Set $\mathbf{h} = \mathsf{unpack\_Rq0}(packed\_public\_key)$ [1.8.4]
6. Set $\mathbf{c} = \mathsf{Rq}(\mathbf{r} \cdot \mathbf{h} + \mathbf{m}_1)$
7. Set $packed\_ciphertext = \mathsf{pack\_Rq0}(\mathbf{c})$ [1.8.3]
8. Output $packed\_ciphertext$

1.11.4 DPKE_Decrypt

Input:

- A byte array $packed\_private\_key$ of length dpke_private_key_bytes.
- A byte array $packed\_ciphertext$ of length dpke_ciphertext_bytes.

Output:

- A byte array $packed\_rm$ of length dpke_plaintext_bytes.
- A bit $fail$.

Operation:

1. Parse $packed\_private\_key$ as $packed\_f \parallel packed\_fp \parallel packed\_hq$ with
   - $packed\_f$ of length packed_s3_bytes
   - $packed\_fp$ of length packed_s3_bytes
   - $packed\_hq$ of length packed_sq_bytes
2. Set $\mathbf{c} = \mathsf{unpack\_Rq0}(packed\_ciphertext)$ [1.8.4]
3. Set $\mathbf{f} = \underline{\mathsf{S3}}(\mathsf{unpack\_S3}(packed\_f))$ [1.8.8]
4. Set $\mathbf{f}_p = \mathsf{unpack\_S3}(packed\_fp)$ [1.8.8]
5. Set $\mathbf{h}_q = \mathsf{unpack\_Sq}(packed\_hq)$ [1.8.6]
6. Set $\mathbf{v}_1 = \underline{\mathsf{Rq}}(\mathbf{c} \cdot \mathbf{f})$
7. Set $\mathbf{m}_0 = \underline{\mathsf{S3}}(\mathbf{v}_1 \cdot \mathbf{f}_p)$
8. Set $\mathbf{m}_1 = \mathsf{Lift}(\mathbf{m}_0)$ [1.9.3]
9. Set $\mathbf{r} = \underline{\mathsf{Sq}}((\mathbf{c} - \mathbf{m}_1) \cdot \mathbf{h}_q)$
10. Set $packed\_rm = \mathsf{pack\_S3}(\mathbf{r}) \parallel \mathsf{pack\_S3}(\mathbf{m}_0)$. [1.8.7]
11. If $\mathbf{r} \in \mathcal{L}_r$ and $\mathbf{m}_0 \in \mathcal{L}_m$ set $fail = 0$
12. Else set $fail = 1$
13. Output $(packed\_rm, fail)$

Notes:

1. This implementation assumes that only the KEM interface is exposed to users. Implementations that expose the DPKE to users are required to return $(\mathsf{pack\_S3}(0) \parallel \mathsf{pack\_S3}(0), 1)$ on failure.
2. Line 2 will discard bits from the final byte of $packed\_ciphertext$ when $(n-1) \cdot \mathsf{logq}$ is not a multiple of 8. Implementations should set the failure flag if the discarded bits are not zero.

1.12 Strongly secure KEM

1.12.1 Key_Pair

Input:

- A bit string *seed* of length key_seed_bits.

Output:

- A byte array *packed_private_key* of length kem_private_key_bytes.
- A byte array *packed_public_key* of length kem_public_key_bytes.

Operation:

1. Parse *seed* as $fg\_bits \parallel prf\_key$ with
   - fg_bits of length sample_key_bits
   - prf_key of length prf_key_bits
2. Set $(packed\_dpke\_private\_key, packed\_public\_key) = \mathsf{DPKE\_Key\_Pair}(fg\_bits)$
3. Set $packed\_private\_key = packed\_dpke\_private\_key \parallel \mathsf{bits\_to\_bytes}(prf\_key)$
4. Output $(packed\_private\_key, packed\_public\_key)$

Notes:

1. This implementation assumes that key_seed_bits = sample_key_bits + prf_key_bits. Implementations may expand fg_bits and prf_key from a 256 bit seed.

1.12.2 Encapsulate

Input:

- A byte array *packed_public_key* of length kem_public_key_bytes.

Output:

- A bit string *shared_key* of length kem_shared_key_bits.
- A byte array *packed_ciphertext* of length kem_ciphertext_bytes.

Operation:

1. Let *coins* be a string of sample_plaintext_bits uniform random bits
2. Set $(\mathbf{r}, \mathbf{m}) = \mathsf{Sample\_rm}(coins)$ [1.10.2]
3. Set $packed\_rm = \mathsf{pack\_S3}(\mathbf{r}) \parallel \mathsf{pack\_S3}(\mathbf{m})$ [1.8.7]
4. Set $shared\_key = \mathsf{Hash}(\mathsf{bytes\_to\_bits}(packed\_rm, 8 \cdot \mathsf{dpke\_plaintext\_bytes}))$ [1.7.2]
5. Set $packed\_ciphertext = \mathsf{DPKE\_Encrypt}(packed\_public\_key, packed\_rm)$ [1.11.3]

Notes:

1. Implementations may expand *coins* from a 256 bit seed.

1.12.3 Decapsulate

Input:

– A byte array $packed\_private\_key$ of length $\mathsf{kem\_private\_key\_bytes}$.

– A byte array $packed\_ciphertext$ of length $\mathsf{kem\_ciphertext\_bytes}$.

Output:

– A bit string $shared\_key$ of length $\mathsf{kem\_shared\_key\_bits}$.

Operation:

1. Parse $packed\_private\_key$ as $packed\_dpke\_private\_key \parallel prf\_key$ with
   – $packed\_dpke\_private\_key$ of length $\mathsf{dpke\_private\_key\_bytes}$
   – prf_key of length $\lceil \mathsf{prf\_key\_bits}/8 \rceil$
2. Parse $packed\_dpke\_private\_key$ as $packed\_f \parallel packed\_fp \parallel packed\_hq$ with
   – $packed\_f$ of length $\mathsf{packed\_s3\_bytes}$
   – $packed\_fp$ of length $\mathsf{packed\_s3\_bytes}$
   – $packed\_hq$ of length $\mathsf{packed\_sq\_bytes}$
3. Set $(packed\_rm, fail) = \mathsf{DPKE\_Decrypt}(packed\_dpke\_private\_key,\ packed\_ciphertext)$ [1.11.4]
4. Set $shared\_key = \mathsf{Hash}(\mathsf{bytes\_to\_bits}(packed\_rm, 8 \cdot \mathsf{dpke\_plaintext\_bytes}))$ [1.7.2]
5. Set $random\_key = \mathsf{Hash}(\mathsf{bytes\_to\_bits}(prf\_key, \mathsf{prf\_key\_bits}) \parallel \mathsf{bytes\_to\_bits}(packed\_ciphertext, 8 \cdot \mathsf{kem\_ciphertext\_bytes}))$ [1.7.2]
6. if $fail = 0$ output $shared\_key$, else output $random\_key$.

2 Design rationale

2.1 Summary of merger

– The NTRUEncrypt submission proposes an IND-CCA2 PKE that is derived from the ANTS'98 NTRU PPKE using the NAEP padding mechanism. The IND-CCA2 security of the KEM is supported by a non-tight reduction to the OW-CPA security of the PPKE in the ROM; the reduction does not go through in the QROM. The submission does not recommend correct parameter sets.

– The NTRU-HRSS-KEM submission proposes an IND-CCA2 KEM that is derived from the ANTS'98 NTRU PPKE using the Targhi–Unruh transformation. The IND-CCA2 security of the KEM is supported by a non-tight reduction to the OW-CPA security of the PPKE in both the ROM and the QROM. The QROM reduction requires a length-preserving message confirmation hash, which adds 141 bytes to ntruhrss701 ciphertexts. The submission insists on perfectly correct parameters.

– A paper by Saito, Xagawa, and Yamakawa [35] proposes a variant of NTRU-HRSS-KEM that eliminates the length-preserving message confirmation hash. The variant is an IND-CCA2 KEM that is derived from a *deterministic* PKE using *re-encryption* and *implicit rejection*. The IND-CCA2 security of the KEM is supported by a tight reduction to the OW-CPA security of the DPKE in the ROM, and a non-tight reduction in the QROM. The QROM reduction is tight if one assumes *sparse pseudorandomness* [35, Definition 3.2] of the underlying DPKE. The tight reductions require perfect correctness. The NTRU DPKE is slightly more expensive than the PPKE; this variant is otherwise a clear improvement over NTRU-HRSS-KEM.

– The (merged) NTRU submission is based on the Saito–Xagawa–Yamakawa variant of NTRU-HRSS-KEM, but it eliminates an expensive part of the decapsulation routine. This efficiency enhancement maintains interoperability with the Saito–Xagawa–Yamakawa variant, has no impact on security, and cancels some of the added cost of the DPKE. All of the proposed parameter sets are correct, and features from the NTRUEncrypt submission have been incorporated to allow for a broader range of size vs. security vs. efficiency trade-offs. The submission does not recommend a direct construction of an IND-CCA2 PKE, but this could change if the open problem in Section 2.4.5 is resolved.

2.2 Detailed description of previous NTRU variants

In this section we present all of the schemes that have influenced the design of the NTRU submission in a unified format. The ANTS'98 NTRU PPKE and DPKE are presented without comment, but we remark on various properties of the NTRUEncrypt submission, the NTRU-HRSS-KEM submission, and the Saito–Xagawa–Yamakawa variant of NTRU-HRSS-KEM. We take some liberties with the use of "Sample" routines to streamline the presentation.

2.2.1 The ANTS'98 NTRU PPKE

$\mathsf{KeyGen}(seed)$	$\mathsf{Encrypt}(\mathbf{h}, \mathbf{m}, coins)$	$\mathsf{Decrypt}((\mathbf{f}, \mathbf{f}_p), \mathbf{c})$
1. $c \leftarrow 0$	1. $\mathbf{r} \leftarrow \mathsf{Sample\_r}(coins)$	1. $\mathbf{a} \leftarrow (\mathbf{c} \cdot \mathbf{f}) \bmod (q, \mathbf{\Phi}_1\mathbf{\Phi}_n)$
2. do $\{\mathbf{f} \leftarrow \mathsf{Sample\_f}(seed, c);\ c \leftarrow c+1\}$	2. $\mathbf{c} \leftarrow (\mathbf{r} \cdot \mathbf{h} + \mathbf{m}) \bmod (q, \mathbf{\Phi}_1\mathbf{\Phi}_n)$	2. $\mathbf{m} \leftarrow (\mathbf{a} \cdot \mathbf{f}_p) \bmod (3, \mathbf{\Phi}_1\mathbf{\Phi}_n)$
3. until $\mathbf{f}$ is invertible mod $(2, \mathbf{\Phi}_1\mathbf{\Phi}_n)$	3. return $\mathbf{c}$	3. return $\mathbf{m}$
and $\mathbf{f}$ is invertible mod $(3, \mathbf{\Phi}_1\mathbf{\Phi}_n)$		
4. $\mathbf{g} \leftarrow \mathsf{Sample\_g}(seed, c)$		
5. $\mathbf{h} \leftarrow (3 \cdot \mathbf{g}/\mathbf{f}) \bmod (q, \mathbf{\Phi}_1\mathbf{\Phi}_n)$		
6. $\mathbf{f}_p \leftarrow (1/\mathbf{f}) \bmod (3, \mathbf{\Phi}_1\mathbf{\Phi}_n)$		
7. return $((\mathbf{f}, \mathbf{f}_p), \mathbf{h})$		

Figure 1: The PPKE from the ANTS'98 paper.

2.2.2 The ANTS'98 NTRU DPKE

$\mathsf{KeyGen}(seed)$	$\mathsf{Encrypt}(\mathbf{h}, (\mathbf{r}, \mathbf{m}))$	$\mathsf{Decrypt}((\mathbf{f}, \mathbf{f}_p, \mathbf{h}_q), \mathbf{c})$
1. $c \leftarrow 0$	1. $\mathbf{c} \leftarrow (\mathbf{r} \cdot \mathbf{h} + \mathbf{m}) \bmod (q, \mathbf{\Phi}_1\mathbf{\Phi}_n)$	1. $\mathbf{a} \leftarrow (\mathbf{c} \cdot \mathbf{f}) \bmod (q, \mathbf{\Phi}_1\mathbf{\Phi}_n)$
2. do $\{\mathbf{f} \leftarrow \mathsf{Sample\_f}(seed, c);\ c \leftarrow c+1\}$	2. return $\mathbf{c}$	2. $\mathbf{m} \leftarrow (\mathbf{a} \cdot \mathbf{f}_p) \bmod (3, \mathbf{\Phi}_1\mathbf{\Phi}_n)$
3. until $\mathbf{f}$ is invertible mod $(2, \mathbf{\Phi}_1\mathbf{\Phi}_n)$		3. $\mathbf{r} \leftarrow ((\mathbf{c} - \mathbf{m}) \cdot \mathbf{h}_q) \bmod (q, \mathbf{\Phi}_1\mathbf{\Phi}_n)$
and $\mathbf{f}$ is invertible mod $(3, \mathbf{\Phi}_n\mathbf{\Phi}_n)$		4. return $(\mathbf{r}, \mathbf{m})$
4. $\mathbf{g} \leftarrow \mathsf{Sample\_g}(seed, c)$		
5. $\mathbf{h} \leftarrow (3 \cdot \mathbf{g}/\mathbf{f}) \bmod (q, \mathbf{\Phi}_1\mathbf{\Phi}_n)$		
6. $\mathbf{h}_q \leftarrow (1/\mathbf{h}) \bmod (q, \mathbf{\Phi}_1\mathbf{\Phi}_n)$		
7. $\mathbf{f}_p \leftarrow (1/\mathbf{f}) \bmod (3, \mathbf{\Phi}_1\mathbf{\Phi}_n)$		
8. return $((\mathbf{f}, \mathbf{f}_p, \mathbf{h}_q), \mathbf{h})$		

Figure 2: The DPKE that is obtained by applying the reasoning of [19, Section 4.2] to Figure 1.

2.2.3 The first round NTRUEncrypt submission

The first round NTRUEncrypt submission applies Howgrave-Graham, Silverman, Singer, and Whyte's NAEP padding mechanism [23] to the ANTS'98 PPKE. The combination is sometimes referred to as SVES-3. The presentation in Figures 3 and 4 is slightly non-standard. We have factored a PPKE out of SVES-3 to which various generic transformations can be applied. This PPKE features *message masking*, which eliminates some obstructions to the IND-CPA security of the ANTS'98 PPKE. The SVES-3 scheme (a.k.a. ntru-pke) is reconstructed in Figure 4. The NTRUEncrypt submission also includes a KEM, which we describe below.

$\underline{\mathsf{KeyGen}(seed)}$	$\underline{\mathsf{Encrypt}(\mathbf{h}, \mathbf{m}, coins)}$	$\underline{\mathsf{Decrypt}(\mathbf{f}, \mathbf{c})}$
1. $c \leftarrow 0$	1. $\mathbf{r} \leftarrow \mathsf{Sample\_r}(coins)$	1. $\mathbf{a} \leftarrow (\mathbf{c} \cdot \mathbf{f}) \bmod (q, \mathbf{\Phi}_1\mathbf{\Phi}_n)$
2. do $\{\mathbf{f} \leftarrow \mathsf{Sample\_f}(seed, c);\ c \leftarrow c+1\}$	2. $\mathbf{s} \leftarrow (\mathbf{r} \cdot \mathbf{h}) \bmod (q, \mathbf{\Phi}_1\mathbf{\Phi}_n)$	2. $\mathbf{m}' \leftarrow \mathbf{a} \bmod (3, \mathbf{\Phi}_1\mathbf{\Phi}_n)$
3. until $\mathbf{f}$ is invertible mod $(2, \mathbf{\Phi}_n)$	3. $\mathbf{t} \leftarrow \mathsf{Sample\_}\mathcal{T}'(H_1(\mathbf{s}))$	3. $\mathbf{s} \leftarrow \mathbf{c} - \mathbf{m}' \bmod (q, \mathbf{\Phi}_1\mathbf{\Phi}_n)$
4. $\mathbf{g} \leftarrow \mathsf{Sample\_g}(seed, c)$	4. $\mathbf{m}' \leftarrow (\mathbf{m} - \mathbf{t}) \bmod (3, \mathbf{\Phi}_1\mathbf{\Phi}_n)$	4. $\mathbf{t} \leftarrow \mathsf{Sample\_}\mathcal{T}'(H_1(\mathbf{s}))$
5. $\mathbf{h} \leftarrow (3 \cdot \mathbf{g}/\mathbf{f}) \bmod (q, \mathbf{\Phi}_1\mathbf{\Phi}_n)$	5. $\mathbf{c} \leftarrow (\mathbf{s} + \mathbf{m}') \bmod (q, \mathbf{\Phi}_1\mathbf{\Phi}_n)$	5. $\mathbf{m} \leftarrow (\mathbf{m}' + \mathbf{t}) \bmod (3, \mathbf{\Phi}_1\mathbf{\Phi}_n)$
6. return $(\mathbf{f}, \mathbf{h})$	6. return $\mathbf{c}$	6. return $\mathbf{m}$

Figure 3: A PPKE implicit in the NTRUEncrypt submission.

$\underline{\mathsf{CCAEncrypt}\ (\mathbf{h}, msg)}$	$\underline{\mathsf{CCADecrypt}\ ((\mathbf{f}, \mathbf{h}), \mathbf{c})}$
1. $coins \leftarrow_\$ \{0,1\}^{256}$	1. $\mathbf{m} \leftarrow \mathsf{Decrypt}(\mathbf{f}, \mathbf{c})$
2. $\mathbf{m} \leftarrow \mathsf{Pad}(msg,\ coins)$	2. $(msg,\ coins) \leftarrow \mathsf{Pad}^{-1}(\mathbf{m})$
3. $\mathbf{c} \leftarrow \mathsf{Encrypt}(\mathbf{h}, \mathbf{m}, H_2(\mathbf{h}, \mathbf{m}))$	3. if $\mathsf{Encrypt}(\mathbf{h}, \mathbf{m}, H_2(\mathbf{h}, \mathbf{m})) \neq \mathbf{c}$ return $\perp$
4. return $\mathbf{c}$	4. else return msg

Figure 4: The ntru-pke scheme from the NTRUEncrypt submission. The Pad function encodes the bit string msg, the length of msg, and $coins$ as a binary polynomial. A maximum message length is provided as a parameter.

Sample spaces The NTRUEncrypt submission uses sample spaces of ternary polynomials of degree at most $n-1$. We write $\mathcal{T}'$ and $\mathcal{T}'(d)$ to distinguish these from the sets of ternary polynomials of degree at most $n-2$ that are used elsewhere in this document. The submission does not fix d as a function of (n, q) and, instead, has integer parameters d_f and d_g. The recommended sample spaces are

$$\mathcal{L}_f = \{1 + 3 \cdot \mathbf{F}\ :\ \mathbf{F} \in \mathcal{T}'(d_f)\}, \quad \mathcal{L}_g = \mathcal{T}'(d_g), \quad \mathcal{L}_m = \mathcal{T}', \quad \text{and } \mathcal{L}_r = \mathcal{T}'.$$

The encryption routine also samples a polynomial $\mathbf{t} \in \mathcal{T}'$.

The choice of $\mathcal{L}_f$ simplifies key generation by ensuring that $\mathbf{f}$ is equivalent to 1 modulo $(2, \mathbf{\Phi}_1)$ and modulo $(3, \mathbf{\Phi}_1)$. It also simplifies decryption by ensuring that $\mathbf{f} \equiv 1 \pmod{(3, \mathbf{\Phi}_1\mathbf{\Phi}_n)}$. However, the choice of $\mathcal{L}_f$ decreases security (and/or increases communication cost) for any fixed decryption failure probability. With $\mathcal{L}_f = \mathcal{T}'(d)$ perfect correctness requires $q > 8d$, but with $\mathcal{L}_f$ as above perfect correctness requires $q > 12d + 1$. This condition can be satisfied by increasing q (which increases communication cost and decreases security) or by decreasing d (which decreases security).

Inverses The NTRUEncrypt submission allows prime n for which $\mathbf{\Phi}_n$ is reducible modulo 2. An invertibility test (Line 3 of KeyGen) is needed to ensure that $\mathbf{f}_q$ exists. The choice of $\mathcal{L}_f$ ensures that $\mathbf{f} \not\equiv 0 \pmod{(2, \mathbf{\Phi}_1)}$, so invertibility only needs to be tested modulo $(2, \mathbf{\Phi}_n)$. The submission recommends $n = 443$ and $n = 743$. The polynomial $\mathbf{\Phi}_{443}$ is irreducible in $(\mathbb{Z}/2)[\mathbf{x}]$, so the test never fails and can be skipped. The polynomial $\mathbf{\Phi}_{743}$ is a product of two terms of degree 371 in $(\mathbb{Z}/2)[\mathbf{x}]$, so the test is unlikely to fail but cannot be skipped.

The polynomials $\mathbf{\Phi}_{443}$ and $\mathbf{\Phi}_{743}$ are both reducible modulo 3. However, the choice of $\mathcal{L}_f$ eliminates the need to compute inverses modulo $(3, \mathbf{\Phi}_1\mathbf{\Phi}_n)$.

Message masking Lines 3 and 4 of Encrypt mask $\mathbf{m}$ with an element of $\mathcal{T}'$. This serves two purposes. First, lattice reduction can easily recover $\mathbf{m}$ from $\mathbf{c}$ when $\mathbf{m}$ is very short, so it is only safe to use the ANTS'98 PPKE to encrypt random messages. Second, if one takes $\mathcal{L}_g = \mathcal{T}'(d_g)$ and $\mathcal{L}_m = \mathcal{T}'$ in the ANTS'98 PPKE, then $\mathbf{h} \equiv 0 \pmod{(q, \mathbf{\Phi}_1)}$ and ciphertexts satisfy $\mathbf{c} \equiv \mathbf{m} \pmod{(q, \mathbf{\Phi}_1)}$. This precludes IND-CPA security. With message masking $\mathbf{c} \equiv \underline{\mathsf{S3}}(\mathbf{m} - \mathbf{t}) \pmod{(q, \mathbf{\Phi}_1)}$. When $coins$ has sufficiently large min-entropy, one can assume that $\mathbf{t}$ is drawn uniformly from $\mathcal{T}'$ and that $\mathbf{c} \bmod (q, \mathbf{\Phi}_1)$ reveals nothing about $\mathbf{m}$.

The assumption that $\mathbf{t}$ is uniform could fail when the coins for the PPKE are taken to be a hash of $\mathbf{m}$. The coins internal to CCAEncrypt in Figure 4 ensure that $\mathbf{t}$ has large min-entropy even when msg is chosen adversarially.

Security reductions Howgrave-Graham, Silverman, Singer, and Whyte provide a (non-tight) reduction from the ROM IND-CCA2 security of SVES-3 to the OW-CPA security of the ANTS'98 PPKE. The reduction accounts for partial correctness, but has not received much scrutiny. The transformation in Figure 4 is equivalent to one proposed by Fujisaki and Okamoto in [13, Section 3]. The reduction in [13] assumes IND-CPA security of the underlying PKE and does not handle partial correctness, but this perspective may be useful for future analysis. As far as we are aware, the NAEP transformation has not been studied in the QROM. Similar transformations have only been shown to be secure in the QROM after non-trivial modifications — see Section 2.4.5 below.

KEM The NTRUEncrypt submission constructs a KEM by encrypting a random 256 bit string using the PKE in Figure 4. The shared secret is computed as $H_3(msg, \mathbf{h})$. Alternatively, one could skip the calls to Pad, choose $\mathbf{m}$ as Sample_m$(coins)$, and output $H_3(\mathbf{m})$ as the shared secret. The resulting KEM would then be an instance of $\mathsf{KEM}_m^{\perp}$ from [21]. From this perspective there are some small changes to the scheme that would lead to tighter security reductions in the ROM [21, Section 3.3], and larger modifications that would yield a security reduction in the QROM [37][21, Section 4.3].

2.2.4 The first round NTRU-HRSS-KEM submission

The first round NTRU-HRSS-KEM submission makes a few small changes to the ANTS'98 PPKE to eliminate invertibility tests. The KEM is constructed using a variant of the Fujisaki–Okamoto transformation that is due to Targhi and Unruh [37].

KeyGen$(seed)$	Encrypt$(\mathbf{h}, \mathbf{m}, coins)$	Decrypt$((\mathbf{f}, \mathbf{f}_p), \mathbf{c})$
1. $(\mathbf{f}, \mathbf{g}) \leftarrow$ Sample_fg$(seed)$	1. $\mathbf{r} \leftarrow$ Sample_r$(coins)$	1. $\mathbf{a} \leftarrow (\mathbf{c} \cdot \mathbf{f}) \bmod (q, \mathbf{\Phi}_1\mathbf{\Phi}_n)$
2. $\mathbf{f}_q \leftarrow (1/\mathbf{f}) \bmod (q, \mathbf{\Phi}_n)$	2. $\mathbf{m}' \leftarrow \mathsf{Lift}(\mathbf{m})$	2. $\mathbf{m}' \leftarrow (\mathbf{a} \cdot \mathbf{f}_p) \bmod (3, \mathbf{\Phi}_n)$
3. $\mathbf{h} \leftarrow (3 \cdot \mathbf{g} \cdot \mathbf{f}_q) \bmod (q, \mathbf{\Phi}_1\mathbf{\Phi}_n)$	3. $\mathbf{c} \leftarrow (\mathbf{r} \cdot \mathbf{h} + \mathbf{m}') \bmod (q, \mathbf{\Phi}_1\mathbf{\Phi}_n)$	3. return $\mathbf{m}'$
4. $\mathbf{f}_p \leftarrow (1/\mathbf{f}) \bmod (3, \mathbf{\Phi}_n)$	4. return $\mathbf{c}$	
5. return $((\mathbf{f}, \mathbf{f}_p), \mathbf{h})$		

Figure 5: The PPKE from the NTRU-HRSS-KEM submission.

Encapsulate $(\mathbf{h})$	Decapsulate $((\mathbf{f}, \mathbf{f}_p, \mathbf{h}), (\mathbf{c}_1, c_2))$
1. $c_0 \leftarrow_\$ \{0,1\}^{256}$	1. $\mathbf{m} \leftarrow \mathsf{Decrypt}((\mathbf{f}, \mathbf{f}_p), \mathbf{e})$
2. $\mathbf{m} \leftarrow$ Sample_m(c_0)	2. $\mathbf{c}'_1 \leftarrow \mathsf{Encrypt}(\mathbf{h}, \mathbf{m}, H_1(\mathbf{m}))$
3. $\mathbf{c}_1 \leftarrow \mathsf{Encrypt}(\mathbf{h}, \mathbf{m}, H_1(\mathbf{m}))$	3. $k \leftarrow H_2(\mathbf{m})$
4. $k \leftarrow H_2(\mathbf{m})$	4. $c'_2 \leftarrow H_3(\mathbf{m})$
5. $c_2 \leftarrow H_3(\mathbf{m})$	5. if $(\mathbf{c}'_1, c'_2) \neq (c_1, c_2)$ return $\perp$
6. return $((\mathbf{c}_1, c_2), k)$	6. else return k

Figure 6: The KEM from the NTRU-HRSS-KEM submission.

Sample spaces The NTRU-HRSS-KEM submission uses

$$\mathcal{L}_f = \mathcal{T}_+, \quad \mathcal{L}_g = \{\mathbf{\Phi}_1 \cdot \mathbf{v} \;:\; \mathbf{v} \in \mathcal{T}_+\}, \quad \mathcal{L}_r = \mathcal{T}, \quad \mathcal{L}_m = \mathcal{T},$$

and takes $\mathsf{Lift}(\mathbf{m}) = \mathbf{\Phi}_1 \cdot \underline{\mathsf{S3}}(\mathbf{m}/\mathbf{\Phi}_1)$.

Invertibility tests The choice of $\mathcal{L}_g$ ensures that $\mathbf{g} \equiv 0 \pmod{(q, \mathbf{\Phi}_1)}$. A consequence is that $\mathbf{h}$ can be computed as $\mathsf{Rq}(3 \cdot \mathbf{g} \cdot \mathsf{Sq}(1/\mathbf{f}))$ instead of $\mathsf{Rq}(3 \cdot \mathbf{g} \cdot \mathsf{Rq}(1/\mathbf{f}))$. The conditions on n ensure that $S/2$ is a finite field and the choice of $\mathcal{L}_f$ ensures that $2 \nmid \mathbf{f}$, so $\mathbf{f}$ has an inverse in S/q. Second, Line 2 of the decryption procedure recovers the message modulo $(p, \mathbf{\Phi}_n)$ instead of modulo $(p, \mathbf{\Phi}_1\mathbf{\Phi}_n)$. A consequence is that $\mathbf{f}_p$ can be computed as $\mathsf{S3}(1/\mathbf{f})$ instead of $\mathsf{R3}(1/\mathbf{f})$. The conditions on n also ensure that $S/3$ is a finite field. The second change does come with a small cost: the message space is restricted to ternary polynomials of degree at most $n-2$ (i.e. canonical $S/3$-representatives) rather than ternary polynomials of degree at most $n-1$.

Lift The ciphertext is computed as an element of R/q, but the message is recovered as an element of $S/3$. The choice of canonical representatives of $S/3$ defines a canonical embedding of the message space into R/q. However, there are practical benefits to allowing different embeddings, and the Lift parameter allows us to select one. The choice of $\mathsf{Lift}(\mathbf{m}) = \mathbf{\Phi}_1 \cdot \underline{\mathsf{S3}}(\mathbf{m}/\mathbf{\Phi}_1)$, in context with the other parameter choices, ensures that ciphertexts satisfy $\mathbf{c} \equiv 0 \pmod{(q, \mathbf{\Phi}_1)}$. This choice of Lift increases the minimum q that provides perfect correctness, but allows the NTRU-HRSS-KEM submission to avoid fixed-weight sampling and message masking.

Use of $\mathcal{T}_+$ The correctness condition (Eq. 1) involves terms of the form $\mathbf{\Phi}_1 \cdot \mathbf{u} \cdot \mathbf{v}$ with $\mathbf{u} \in \mathcal{T}_+$ and $\mathbf{v} \in \mathcal{T}$. The condition is satisfied if, for all $\mathbf{u} \in \mathcal{T}_+$ and $\mathbf{v} \in \mathcal{T}$, the coefficients of $\mathbf{\Phi}_1 \cdot \mathbf{u} \cdot \mathbf{v}$ are between $-q/8$ and $q/8-1$. Lemma 1 of [25] uses the non-negative correlation property to bound the size of the coefficients of $\mathbf{\Phi}_1 \cdot \mathbf{u} \cdot \mathbf{v}$ by $\sqrt{2}(n-1)$. This implies that the scheme is correct when $q > 8\sqrt{2}(n-1)$, which is a factor of $\sqrt{2}$ better than the naive bound.

The Targhi–Unruh transformation KEM variants of the FO transformation were studied by Dent in [11]. The transformation in the NTRU-HRSS-KEM submission is [11, Table 5] with an additional condition that the plaintext-confirmation hash (c_2 in Figure 6) is *length-preserving*. In the ROM, Dent provides a reduction from the IND-CCA2 security of the KEM to the OW-CPA security of the PPKE with a tightness gap that is proportional to the number of random oracle queries. Targhi and Unruh [37] provide an analogous reduction in the QROM which requires an injective hash function for the plaintext-confirmation hash. Their reduction has a tightness gap proportional to the sixth power of the number of random oracle queries. An NTRU-HRSS-KEM plaintext is a degree $n-1$ polynomial with ternary coefficients. The plaintext-confirmation hash adds 141 bytes to the length of a ntruhrss701 ciphertext.

2.2.5 The Saito–Xagawa–Yamakawa variant of NTRU-HRSS-KEM

Saito, Xagawa, and Yamakawa [35] present a variant of NTRU-HRSS-KEM that has a tight security reduction in the ROM and avoids the plaintext-confirmation hash. They achieve this with two independent changes: 1) their KEM is based on a DPKE, and 2) their KEM responds to malformed ciphertexts with a pseudorandom key rather than an error symbol.

$\underline{\mathsf{KeyGen}'(\mathit{seed})}$	$\underline{\mathsf{Encrypt}(\mathbf{h}, (\mathbf{r}, \mathbf{m}))}$	$\underline{\mathsf{Decrypt}((\mathbf{f}, \mathbf{f}_p, \mathbf{h}_q), \mathbf{c})}$
1. $(\mathbf{f}, \mathbf{g}) \leftarrow \mathsf{Sample\_fg}(\mathit{seed})$	1. $\mathbf{m}' \leftarrow \mathsf{Lift}(\mathbf{m})$	1. $\mathbf{a} \leftarrow (\mathbf{c} \cdot \mathbf{f}) \bmod (q, \mathbf{\Phi}_1\mathbf{\Phi}_n)$
2. $\mathbf{f}_q \leftarrow (1/\mathbf{f}) \bmod (q, \mathbf{\Phi}_n)$	2. $\mathbf{c} \leftarrow (\mathbf{r} \cdot \mathbf{h} + \mathbf{m}') \bmod (q, \mathbf{\Phi}_1\mathbf{\Phi}_n)$	2. $\mathbf{m} \leftarrow (\mathbf{a} \cdot \mathbf{f}_p) \bmod (3, \mathbf{\Phi}_n)$
3. $\mathbf{h} \leftarrow (3 \cdot \mathbf{g} \cdot \mathbf{f}_q) \bmod (q, \mathbf{\Phi}_1\mathbf{\Phi}_n)$	3. return $\mathbf{c}$	3. $\mathbf{m}' \leftarrow \mathsf{Lift}(\mathbf{m})$
4. $\mathbf{h}_q \leftarrow (1/\mathbf{h}) \bmod (q, \mathbf{\Phi}_n)$		4. $\mathbf{r}' \leftarrow ((\mathbf{c} - \mathbf{m}')\mathbf{h}_q) \bmod (q, \mathbf{\Phi}_n)$
5. $\mathbf{f}_p \leftarrow (1/\mathbf{f}) \bmod (3, \mathbf{\Phi}_n)$		5. $\mathbf{r} \leftarrow \mathbf{r}' \bmod (3, \mathbf{\Phi}_n)$
6. return $((\mathbf{f}, \mathbf{f}_p, \mathbf{h}_q), \mathbf{h})$		6. return $(\mathbf{r}, \mathbf{m})$

Figure 7: The DPKE from Saito–Xagawa–Yamakawa.

$\underline{\mathsf{KeyGen}\,(\mathit{seed})}$	$\underline{\mathsf{Encapsulate}\,(\mathbf{h})}$	$\underline{\mathsf{Decapsulate}\,(((\mathbf{f}, \mathbf{f}_p, \mathbf{h}_q, s), \mathbf{h}), \mathbf{c})}$
1. $((\mathbf{f}, \mathbf{f}_p, \mathbf{h}_q), \mathbf{h}) \leftarrow \mathsf{KeyGen}'(\mathit{seed})$	1. $\mathit{coins} \leftarrow_{\$} \{0,1\}^{256}$	1. $(\mathbf{r}, \mathbf{m}) \leftarrow \mathsf{Decrypt}((\mathbf{f}, \mathbf{f}_p, \mathbf{h}_q), \mathbf{e})$
2. $s \leftarrow_{\$} \{0,1\}^{256}$	2. $(\mathbf{r}, \mathbf{m}) \leftarrow \mathsf{Sample\_rm}(\mathit{coins})$	2. $k_1 \leftarrow H_1(\mathbf{m})$
3. return $((\mathbf{f}, \mathbf{f}_p, \mathbf{h}_q, s), \mathbf{h})$	3. $\mathbf{c} \leftarrow \mathsf{Encrypt}(\mathbf{h}, (\mathbf{r}, \mathbf{m}))$	3. $k_2 \leftarrow H_2(s, \mathbf{c})$
	4. $k \leftarrow H_1(\mathbf{r}, \mathbf{m})$	4. if $\mathsf{Encrypt}(\mathbf{h}, (\mathbf{r}, \mathbf{m})) = \mathbf{c}$ return k_1
	5. return $(\mathbf{c}, k)$	5. else return k_2

Figure 8: The KEM from Saito–Xagawa–Yamakawa.

Sample spaces The sample spaces match those of NTRU-HRSS-KEM.

DPKE The DPKE is essentially what one would obtain by applying the reasoning of the CRYPTO'96 NTRU preprint [19, Section 4.2] to the NTRU-HRSS-KEM PPKE (Figure 5). It differs only in that the $\mathbf{r}$ component is reduced modulo $(3, \mathbf{\Phi}_n)$ during decryption (Figure 7, Line 5 of Decrypt).

Implicit rejection The KEM rejects invalid ciphertexts by returning a pseudorandom key instead of an error symbol — a technique called implicit rejection. Implicit rejection was first used by Persichetti in a code-based cryptosystem [34]. It was proposed as part of a generic OW-CPA DPKE to IND-CCA2 KEM transformation in [21], and this transformation is supported by a tight security reduction in the ROM [21, 5]. Saito, Xagawa, and Yamakawa give a tight reduction in the QROM when the DPKE satisfies a non-standard *sparse pseudorandomness* assumption [35]. For NTRU-HRSS-KEM the unproven part of this assumption states that an adversary who is given an honestly generated $\mathbf{h}$ cannot distinguish an honestly generated ciphertext from an element of $\{\mathbf{v} \in R/q \ : \ \mathbf{v} \equiv 0 \pmod{(q, \mathbf{\Phi}_1)}\}$ drawn uniformly at random.

2.3 The NTRU submission

The second round NTRU submission is based on the Saito–Xagawa–Yamakawa variant of NTRU-HRSS-KEM [35]. We make two small changes to the decryption procedure of the DPKE to avoid re-encryption, but note that our Encapsulate and Decapsulate routines are identical to those Figure 8 in terms of their input/output behavior.

$\mathsf{KeyGen}'(seed)$	$\mathsf{Encrypt}(\mathbf{h}, (\mathbf{r}, \mathbf{m}))$	$\mathsf{Decrypt}((\mathbf{f}, \mathbf{f}_p, \mathbf{h}_q), \mathbf{c})$
1. $(\mathbf{f}, \mathbf{g}) \leftarrow \mathsf{Sample\_fg}(seed)$	1. $\mathbf{m}' \leftarrow \mathsf{Lift}(\mathbf{m})$	1. if $\mathbf{c} \not\equiv 0 \pmod{(q, \mathbf{\Phi}_1)}$ return $(0, 0, 1)$
2. $\mathbf{f}_q \leftarrow (1/\mathbf{f}) \bmod (q, \mathbf{\Phi}_n)$	2. $\mathbf{c} \leftarrow (\mathbf{r} \cdot \mathbf{h} + \mathbf{m}') \bmod (q, \mathbf{\Phi}_1\mathbf{\Phi}_n)$	2. $\mathbf{a} \leftarrow (\mathbf{c} \cdot \mathbf{f}) \bmod (q, \mathbf{\Phi}_1\mathbf{\Phi}_n)$
3. $\mathbf{h} \leftarrow (3 \cdot \mathbf{g} \cdot \mathbf{f}_q) \bmod (q, \mathbf{\Phi}_1\mathbf{\Phi}_n)$	3. return $\mathbf{c}$	3. $\mathbf{m} \leftarrow (\mathbf{a} \cdot \mathbf{f}_p) \bmod (3, \mathbf{\Phi}_n)$
4. $\mathbf{h}_q \leftarrow (1/\mathbf{h}) \bmod (q, \mathbf{\Phi}_n)$		4. $\mathbf{m}' \leftarrow \mathsf{Lift}(\mathbf{m})$
5. $\mathbf{f}_p \leftarrow (1/\mathbf{f}) \bmod (3, \mathbf{\Phi}_n)$		5. $\mathbf{r} \leftarrow ((\mathbf{c} - \mathbf{m}') \cdot \mathbf{h}_q) \bmod (q, \mathbf{\Phi}_n)$
6. return $((\mathbf{f}, \mathbf{f}_p, \mathbf{h}_q), \mathbf{h})$		6. if $(\mathbf{r}, \mathbf{m}) \in \mathcal{L}_r \times \mathcal{L}_m$ return $(\mathbf{r}, \mathbf{m}, 0)$
		7. else return $(0, 0, 1)$

Figure 9: The DPKE for the NTRU submission.

$\mathsf{KeyGen}\,(seed)$	$\mathsf{Encapsulate}\,(\mathbf{h})$	$\mathsf{Decapsulate}\,((\mathbf{f}, \mathbf{f}_p, \mathbf{h}_q, s), \mathbf{c})$
1. $((\mathbf{f}, \mathbf{f}_p, \mathbf{h}_q), \mathbf{h}) \leftarrow \mathsf{KeyGen}'(seed)$	1. $coins \leftarrow_{\$} \{0,1\}^{256}$	1. $(\mathbf{r}, \mathbf{m}, fail) \leftarrow \mathsf{Decrypt}((\mathbf{f}, \mathbf{f}_p, \mathbf{h}_q), \mathbf{c})$
2. $s \leftarrow_{\$} \{0,1\}^{256}$	2. $(\mathbf{r}, \mathbf{m}) \leftarrow \mathsf{Sample\_rm}(coins)$	2. $k_1 \leftarrow H_1(\mathbf{r}, \mathbf{m})$
3. return $((\mathbf{f}, \mathbf{f}_p, \mathbf{h}_q, s), \mathbf{h})$	3. $\mathbf{c} \leftarrow \mathsf{Encrypt}(\mathbf{h}, (\mathbf{r}, \mathbf{m}))$	3. $k_2 \leftarrow H_2(s, \mathbf{c})$
	4. $k \leftarrow H_1(\mathbf{r}, \mathbf{m})$	4. if $fail = 0$ return k_1
	5. return $(\mathbf{c}, k)$	5. else return k_2

Figure 10: The KEM for the NTRU submission.

Sample spaces Our NTRU-HPS parameter sets take

$$\mathcal{L}_f = \mathcal{T}, \quad \mathcal{L}_g = \mathcal{T}(q/8-2), \quad \mathcal{L}_r = \mathcal{T}, \quad \mathcal{L}_m = \mathcal{T}(q/8-2)$$

and $\mathsf{Lift}(\mathbf{m}) = \mathbf{m}$. The choice of $\mathcal{L}_g$ ensures that $\mathbf{h} \equiv 0 \pmod{(q, \mathbf{\Phi}_1)}$, and along with the choice of $\mathcal{L}_m$ this ensures that $\mathbf{c} \equiv 0 \pmod{(q, \mathbf{\Phi}_1)}$. The weight parameter $q/8 - 2$ is the largest that is compatible with perfect correctness.

Our NTRU-HRSS parameter sets use

$$\mathcal{L}_f = \mathcal{T}_+, \quad \mathcal{L}_g = \{\mathbf{\Phi}_1 \cdot \mathbf{v} \ : \ \mathbf{v} \in \mathcal{T}_+\}, \quad \mathcal{L}_r = \mathcal{T}, \quad \mathcal{L}_m = \mathcal{T}$$

and $\mathsf{Lift}(\mathbf{m}) = \mathbf{\Phi}_1 \cdot \underline{\mathsf{S3}}(\mathbf{m}/\mathbf{\Phi}_1)$. The choice of $\mathcal{L}_g$ ensures that $\mathbf{h} \equiv 0 \pmod{(q, \mathbf{\Phi}_1)}$, and along with the choice of Lift this ensures that $\mathbf{c} \equiv 0 \pmod{(q, \mathbf{\Phi}_1)}$.

Avoiding re-encryption Bernstein and Persichetti [5] cast implicit rejection as a generic transformation from a *rigid* correct DPKE to an IND-CCA2 KEM. A DPKE is rigid if, for all keys (sk, pk), ciphertexts c, and plaintexts p, $(\mathsf{Encrypt}(pk, p) = c) \Leftrightarrow (\mathsf{Decrypt}(sk, c) = p)$. Correctness implies the forward direction. Re-encryption can be used to ensure that the reverse direction holds (as in Line 4 of Figure 8). However, as observed in [5, Section 6], it is possible to construct rigid correct DPKEs that do not rely on re-encryption.

The DPKE in Figure 9 is rigid. If $\mathsf{Decrypt}((\mathbf{f}, \mathbf{f}_p, \mathbf{h}_q), \mathbf{c})$ outputs $(\mathbf{r}, \mathbf{m}, 0)$, then $(\mathbf{r}, \mathbf{m})$ is in the plaintext space by Line 6 and $\mathbf{c} \equiv 0 \pmod{(q, \mathbf{\Phi}_1)}$ by Line 1. Line 5 further implies that $\mathbf{c}$ satisfies $\mathbf{c} \equiv \mathbf{rh} + \mathsf{Lift}(\mathbf{m}) \pmod{(q, \mathbf{\Phi}_n)}$. Hence, $\mathbf{c} = \mathsf{Encrypt}(\mathbf{h}, (\mathbf{r}, \mathbf{m}))$. Note that it is important that we skipped the reduction modulo $(3, \mathbf{\Phi}_n)$ in Line 5 of Decrypt in Figure 7.

Adam Langley has observed[1] that there is no need to check $\mathbf{c} \equiv 0 \pmod{(q, \mathbf{\Phi}_1)}$ when ciphertexts are unpacked using the unpack_Rq0 (Section 1.8.4) routine.

2.4 Variants of the NTRU submission

2.4.1 Faster key generation for single-use keys

We recommend that the KEM be used as is in an ephemeral setting. However, users may be tempted to take a certain shortcut in key generation when they know that a key will only be used once. The *improper-key variant* of an NTRU-HPS parameter set takes $\mathcal{L}_f = \{1 + 3\mathbf{F} : \mathbf{F} \in \mathcal{T}\}$. The improper-key variant of an NTRU-HRSS parameter set takes $\mathcal{L}_f = \{1 + 3\mathbf{F} : \mathbf{F} \in \mathcal{T}_+\}$. Improper keys have a non-zero decryption failure probability, and they should not be re-used. Implementations that use improper keys may skip the computation of $\mathbf{f}_p$ in KeyGen and the multiplication by $\mathbf{f}_p$ in Decrypt (since $\mathbf{f}_p = 1$). No change is made to the encapsulation routine, so the decision to use an improper key is local to each user. Decryption failure rates for improperly generated keys are given in Table 1.

ntruhps2048509	ntruhps2048677	ntruhps4096821	ntruhrss701
$2^{-214.3}$	$2^{-213.9}$	$2^{-433.2}$	$2^{-796.6}$

Table 1: Decryption failure rates for improperly generated keys. Calculated using the scripts at https://github.com/jschanck/decryption-failures.

2.4.2 Prime q

It is relatively easy to define variants of NTRU-HPS and NTRU-HRSS that use prime q. When all other parameters are equal, these variants will be slightly less efficient. However, there are size vs. security trade-offs that are not available when q is a power of two, and approximating a desired trade-off with a power of two q has a cost in terms of efficiency, security, and compactness. The cost is particularly large for NTRU-HRSS parameter sets for which the fractional part of $\log_2(n)$ is larger than $1/2$. We have plotted the size vs. security trade-offs that are available with prime q in Figure 11.

2.4.3 NTRU-HPS-like parameter sets with faster key generation

Applications that use long-term keys will likely be able to tolerate the cost of key generation in NTRU-HPS. However, there are correct NTRU-HPS-like parameter sets that avoid the need to compute $\mathbf{f}_p$. For example, one can take $\mathcal{L}_f = \{1 + 3\mathbf{F} : \mathbf{F} \in \mathcal{T}\}$, $\mathcal{L}_g = \mathcal{T}(d)$, $\mathcal{L}_r = \mathcal{T}$, and $\mathcal{L}_m = \mathcal{T}(d)$ with d the largest even integer less than $q/12 - 2$. We have plotted the size vs. security trade-offs that are available with these parameter sets in Figure 11. We have also plotted the prime q variants of these parameter sets.

[1] Personal communication, Dec. 14, 2018.

2.4.4 Arbitrary weight m and fixed-weight f

The NTRUEncrypt submission uses fixed-weight $\mathbf{f}$ and $\mathbf{g}$, arbitrary weight $\mathbf{m}$ and $\mathbf{r}$, and trivial Lift. Our NTRU-HPS parameter sets use fixed-weight $\mathbf{m}$ and $\mathbf{g}$, arbitrary weight $\mathbf{f}$ and $\mathbf{r}$, and trivial Lift. This choice ensures that ciphertexts satisfy $\mathbf{c} \equiv 0 \pmod{(q, \mathbf{\Phi}_1)}$, and it minimizes the least q that provides correctness. An alternative is to define Lift as in NTRU-HRSS. Then one can take

$$\mathcal{L}_f = \mathcal{T}_+(d), \quad \mathcal{L}_g = \mathcal{T}(d), \quad \mathcal{L}_r = \mathcal{T}, \quad \text{and } \mathcal{L}_m = \mathcal{T}$$

with d the largest even integer less than $q/(6+2\sqrt{2})$. This leads to a slightly worse size vs. security trade-off than NTRU-HPS, but it limits the use of fixed-weight sampling to key generation. These may be attractive parameters for applications in which NTRU-HPS encapsulation is too expensive, NTRU-HRSS public keys and ciphertexts are too large, and the cost of key generation is not a concern. We are not currently aware of any applications with these constraints, so we have chosen not to recommend parameters of this form.

2.4.5 An IND-CCA2 PKE using Q-OAEP

Targhi and Unruh propose a modified OAEP padding mechanism, Q-OAEP, in [37]. If we choose parameters with $\mathcal{L}_r = \mathcal{T}$ and $\mathcal{L}_m = \mathcal{T}$, then we can apply Q-OAEP to the DPKE in Figure 9 as follows. The encryption routine uses three hash functions H_1, H_2, and H_3; it takes a public key $\mathbf{h}$ and a message $\mathbf{m} \in \mathcal{T}$ as input; and it computes

$$\mathit{coins} \leftarrow \{0,1\}^{256}; \quad \mathbf{r} \leftarrow \mathsf{Sample\_r}(\mathit{coins}); \quad \mathbf{t} \leftarrow (\mathbf{m} - \mathsf{Sample\_}\mathcal{T}(H_1(\mathbf{r}))) \bmod (3, \mathbf{\Phi}_n)$$

$$\mathbf{s} \leftarrow (\mathbf{r} - \mathsf{Sample\_}\mathcal{T}(H_2(\mathbf{t}))) \bmod (3, \mathbf{\Phi}_n); \quad \mathbf{c} \leftarrow \mathsf{Encrypt}(\mathbf{h}, (\mathbf{s}, \mathbf{t})).$$

It then outputs the ciphertext $(\mathbf{c}, H_3(\mathbf{s}, \mathbf{t}))$. Unfortunately the reduction given in [37] requires H_3 to be length-preserving, and this eliminates the benefit of using a PKE instead of a KEM+DEM. Removing the length-preserving hash, and tightening the reduction, are interesting open problems.

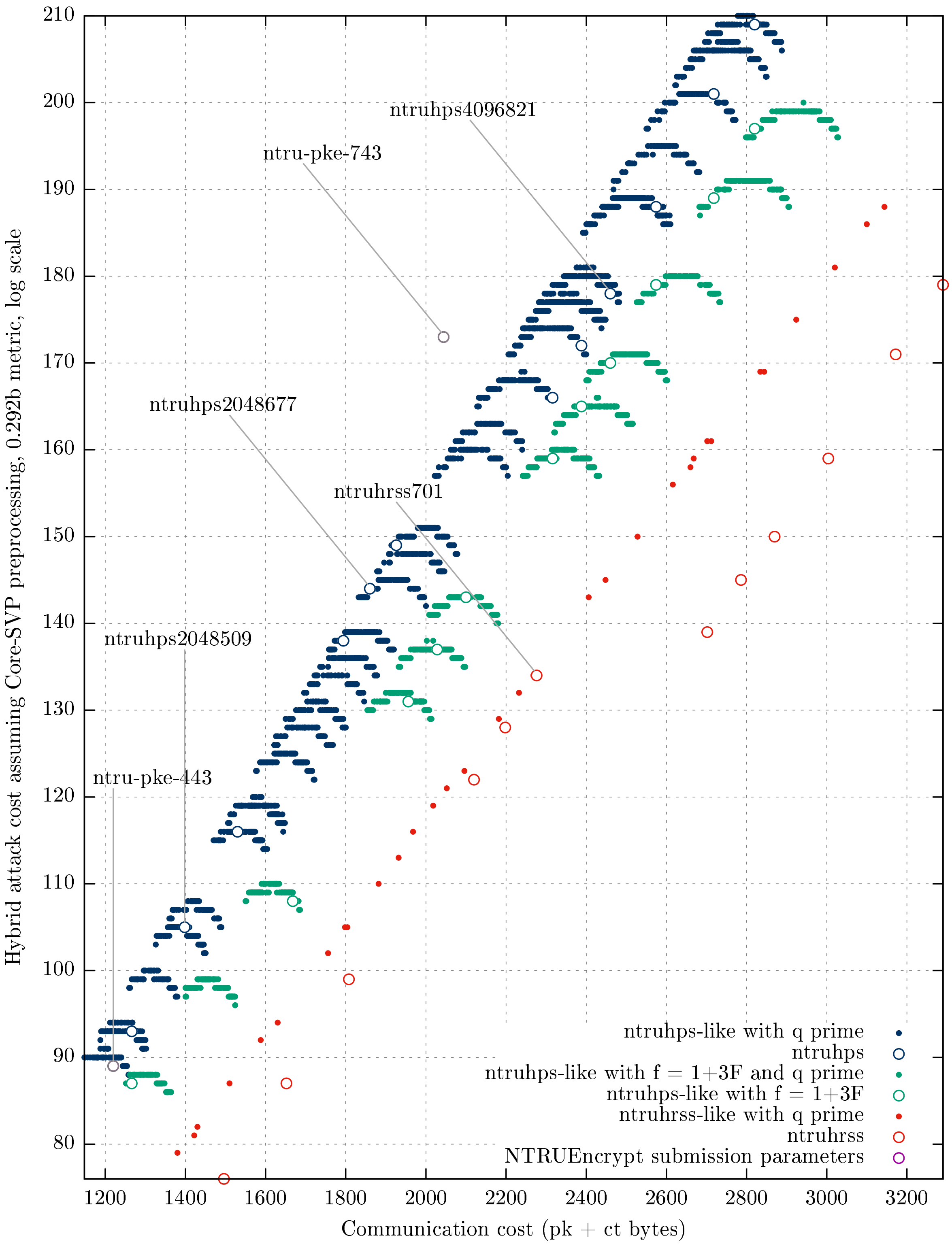
2.5 Available size vs. security trade-offs

Figure 11 shows size vs. security trade-offs for several NTRU variants. Additional plots can be found in the appendix. We have plotted all NTRU-HPS-like parameter sets with $461 \le n \le 941$ and weight parameter d with $n/3 \le d \le 2n/3$. We have plotted all NTRU-HRSS-like parameter sets with $461 \le n \le 941$ and the smallest q that provides perfect correctness. We have also plotted the ntru-pke-443 and ntru-pke-743 parameter sets that were recommended in the first round NTRUEncrypt submission. Note that these NTRUEncrypt parameter sets do not provide perfect correctness. The security axis is explained in Section 6.4.3.

2.6 Parameter selection

The ntruhrss701 parameter set was originally selected because $n = 701$ provides the highest security level among NTRU-HRSS parameter sets with $q := 2^{\lceil 7/2 + \log_2(n) \rceil} \le 8192$. In selecting NTRU-HPS parameter sets we have also attempted to maximize security while minimizing q.

We have decided to only recommended NTRU-HPS parameter sets with $n/3 \le q/8 - 2 \le 2n/3$. Recall that $q/8 - 2$ is the weight of vectors in $\mathcal{L}_g$ and $\mathcal{L}_m$. We view a weight parameters outside of this range as a potential security risk. Figure 12 shows a wider range of NTRU-HPS parameter sets. Each curve represents a choice of n. Exceeding the upper bound on weight clearly leads to a sub-optimal size vs. security trade-off. The lower bound, however, is heuristic and excludes some potentially interesting parameter sets, e.g. ntruhps1024557 and ntruhps2048859, which appear above and to the left of our recommended parameter sets in Figure 12. The apparent benefit of these parameter sets is diminished in Figure 13, which uses a different estimate for the cost of attacks.



ntruhps4096821
ntru-pke-743
ntruhps2048677
ntruhrss701
ntruhps2048509
ntru-pke-443
ntruhps-like with q prime
ntruhps
ntruhps-like with f = 1+3F and q prime
ntruhps-like with f = 1+3F
ntruhrss-like with q prime
ntruhrss
NTRUEncrypt submission parameters
Hybrid attack cost assuming Core-SVP preprocessing, 0.292b metric, log scale
Communication cost (pk + ct bytes)
210
200
190
180
170
160
150
140
130
120
110
100
90
80
1200
1400
1600
1800
2000
2200
2400
2600
2800
3000
3200

Figure 11:

3 Performance analysis

3.1 Description of platform

In order to obtain benchmarks, we evaluate our reference implementation on a machine using the Intel x64-86 instruction set. In particular, we use a single core of a 3.2 GHz Intel Core i3-6100T CPU. We follow the standard practice of disabling TurboBoost and hyper-threading. The system has 32 KiB L1 instruction cache, 32 KiB L1 data cache, 256 KiB L2 cache and 3072 KiB L3 cache. Furthermore, it has 16GiB of RAM, running at 1066 MHz. When performing the benchmarks, the system ran on Linux kernel 5.3.0-3-amd64, Debian 11 (Bullseye). We compiled the code using GCC version `10.2.0-9`, with the compiler optimization flag `-O3`.

We used the same platform described above to evaluate our AVX2 implementation. For the AVX2 implementation, we included the additional compiler flag `'-march=native'`.

3.2 Performance of reference and AVX2 implementations

Table 2: Key and ciphertext sizes and cycle counts for all of the recommended parameter sets. Cycle counts were obtained on one core of an Intel Core i7-4770K (Haswell); "ref" refers to the C reference implementation, "AVX2" to the implementation using AVX2 vector instructions; **sk** stands for secret key, **pk** for public key, and **ct** for ciphertext.

ntruhps2048509					
Sizes (in Bytes)		**Haswell Cycles (ref)**		**Haswell Cycles (AVX2)**	
sk:	935	**gen:**	8 906 821	**gen:**	191 279
pk:	699	**enc:**	643 128	**enc:**	61 331
ct:	699	**dec:**	1 662 377	**dec:**	40 026
ntruhps2048677					
Sizes (in Bytes)		**Haswell Cycles (ref)**		**Haswell Cycles (AVX2)**	
sk:	1235	**gen:**	15 387 578	**gen:**	309 216
pk:	931	**enc:**	1 092 814	**enc:**	83 519
ct:	931	**dec:**	2 897 664	**dec:**	59 729
ntruhps4096821					
Sizes (in Bytes)		**Haswell Cycles (ref)**		**Haswell Cycles (AVX2)**	
sk:	1592	**gen:**	22 511 180	**gen:**	431 667
pk:	1230	**enc:**	1 566 922	**enc:**	98 809
ct:	1230	**dec:**	4 237 744	**dec:**	75 384
ntruhrss701					
Sizes (in Bytes)		**Haswell Cycles (ref)**		**Haswell Cycles (AVX2)**	
sk:	1452	**gen:**	16 487 419	**gen:**	340 823
pk:	1138	**enc:**	1 069 326	**enc:**	50 441
ct:	1138	**dec:**	3 113 303	**dec:**	62 267

3.3 Memory usage

The memory usage benchmarks of our reference implementations range from 11 KiB for ntruhps2048509 to 18 KiB for ntruhps4096821, and our AVX2 implementation of ntruhrss701 requires 47 KiB. Note that memory consumption was not an optimization target, and these numbers should not be considered to be a lower bound.

4 Known Answer Test values

All KAT values are included in subdirectories of the directory `KAT/` of the submission package. The KAT values were generated by the `PQCgenKAT_kem` program provided by NIST. The complete list of KAT files is:

- `KAT/ntruhps2048509/PQCkemKAT_935.req`,
- `KAT/ntruhps2048509/PQCkemKAT_935.rsp`.
- `KAT/ntruhps2048677/PQCkemKAT_1234.req`,
- `KAT/ntruhps2048677/PQCkemKAT_1234.rsp`.
- `KAT/ntruhps4096821/PQCkemKAT_1590.req`,
- `KAT/ntruhps4096821/PQCkemKAT_1590.rsp`.
- `KAT/ntruhrss701/PQCkemKAT_1450.req`,
- `KAT/ntruhrss701/PQCkemKAT_1450.rsp`.

5 Expected security

5.1 Security definition for key-establishment

When used with NTRU-HPS (Section 1.3.2) or NTRU-HRSS (Section 1.3.3) parameter sets, the key encapsulation mechanism specified in Section 1.12 is perfectly correct and achieves the standard notion of security against adaptive chosen ciphertext attacks (IND-CCA2 security) in the random oracle model.

A reduction from the IND-CCA2 security of the KEM to the OW-CPA security of the DPKE is given in [35]. Alternative proofs of security can be obtained by viewing the KEM in Section 1.12 can as an application of the $U_m^{\not\perp}$ transformation of Hofheinz, Hövelmanns, and Kiltz [21], or as an application of the "$H(1, p)$" variant of the SimpleKEM transformation of Bernstein and Persichetti [5]. All of these reductions are tight in the random oracle model and non-tight in the quantum random oracle model. A tight reduction in the QROM can be had if one assumes that the DPKE in Section 1.12 is *sparse* and *pseudorandom* [35, Definition 3.2]. It is clear that our DPKE is sparse, but we are not aware of any significant investigation of its pseudorandomness.

5.2 Security definition for ephemeral-only key-establishment

The ephemeral-only variants of our recommended parameter sets (Section 2.4.1) do not provide perfect correctness, and cannot rely on the same security theorems as our IND-CCA2 KEM. Nevertheless, if one assumes OW-CPA security of the DPKE, and that the private key is only used once, session keys that are produced using these parameter sets are indistinguishable from uniform in the ROM. An adversary who observes a transcript $(\mathbf{h}, \mathbf{c})$ for which $\mathsf{Decapsulate}((\mathbf{f}, \mathbf{f}_p, \mathbf{h}_q, s), \mathbf{c}) = k$ can only distinguish k from $k^* \leftarrow_{\$} \{0,1\}^{256}$ if

- $\mathsf{Decrypt}((\mathbf{f}, \mathbf{f}_p, \mathbf{h}), \mathbf{c}) = (\mathbf{r}, \mathbf{m}, 0)$ and the adversary queries $H_1(\mathbf{r}, \mathbf{m})$, or
- $\mathsf{Decrypt}((\mathbf{f}, \mathbf{f}_p, \mathbf{h}), \mathbf{c}) = (0, 0, 1)$ and the adversary queries $H_2(s, \mathbf{c})$.

Suppose that the adversary makes at most 2^{w_1} queries to H_1 and at most 2^{w_2} queries to H_2. The first case implies either a violation of OW-CPA security or that the adversary has queried $(\mathbf{r}, \mathbf{m})$ by chance, which occurs with probability at most $2^{w_1 - \mathsf{sample\_plaintext\_bits}}$. The second case occurs with probability at most $2^{w_2 - \mathsf{prf\_key\_bits}}$.

5.3 Security categories

NIST security categories 1, 3, and 5 are defined relative to the "computational resources" that are required for a key search on a block cipher with a 128-, 192-, or 256-bit key, respectively. The call for proposals states that "computational resources may be measured using a variety of different metrics" and that the thresholds must be satisfied "with respect to all metrics that NIST deems to be potentially relevant to practical security." NIST has, understandably, not specified an exhaustive set of relevant metrics, so we have chosen to provide two security evaluations: one relative to non-local models of computation, and one relative to local models of computation.

A model of computation is *non-local* if it allows unit-cost communication at arbitrary distance[2]. Random access machines, boolean circuits, and Clifford+T quantum circuits are all non-local models of computation. A model of computation is *local* if signals within it propagate at finite speed (e.g. the speed of light). Single-tape Turing machines, VLSI models, and anyonic quantum computers are all local models of computation. A parallel machine model that allows non-local communication between otherwise local machines is non-local. A quantum machine model that allows non-local classical computation is also non-local. Some non-local models can be considered local when their memory usage is restricted, e.g. circuits of constant width and random access machines with $O(1)$ bits of memory.

Key search attacks on block ciphers perform well in local models of computation, and it seems unlikely that non-locality can be used to significantly improve performance. The same cannot be said for attacks on NTRU. Several of the best attack algorithms, e.g. sieve algorithms for the shortest vector problem, achieve significantly better performance in non-local models than they are known to achieve in local models. That said, attacks in local models have not received the same level of attention, and the situation is unstable. We discuss a conjecture by Ducas below that, if true, would cause us to revise our security categories relative to local models.

Security categories relative to **non-local** models	
Parameter set	Category
ntruhps2048509	–
ntruhrss701	1
ntruhps2048677	1
ntruhps4096821	3

Security categories relative to **local** models	
Parameter set	Category
ntruhps2048509	1
ntruhrss701	3
ntruhps2048677	3
ntruhps4096821	5

6 Cost of known attacks

Note (2020-07-30): *This section no longer reflects the state-of-the-art. A revised analysis will be advertised on the pqc-forum mailing list.*

Some background on lattices is assumed. Throughout this section we view $\mathbb{Z}[\mathbf{x}]/(\mathbf{\Phi}_1\mathbf{\Phi}_n)$ as $(\mathbb{Z}^n, +, \circledast)$ with $\mathbf{u} \circledast \mathbf{v} = \mathbf{uv} \bmod (\mathbf{\Phi}_1\mathbf{\Phi}_n)$. We write $\langle\cdot,\cdot\rangle$ and $|\cdot|$ for the euclidean inner product and norm. We present a basis of a lattice as an ordered set of vectors $\mathbf{B} = (\mathbf{b}_1, \ldots, \mathbf{b}_d)$. We write $\pi_{\mathbf{B},i}(\mathbf{v})$ for the projection of $\mathbf{v}$ orthogonal to the first $i-1$ vectors in $\mathbf{B}$. We suppress the $\mathbf{B}$ from $\pi_{\mathbf{B},i}$ when it is clear from context. We denote the Gram-Schmidt vectors of $\mathbf{B}$ by $(\mathbf{b}_1^*, \mathbf{b}_2^*, \ldots, \mathbf{b}_d^*) = (\pi_1(\mathbf{b}_1), \pi_2(\mathbf{b}_2), \ldots, \pi_d(\mathbf{b}_d))$ We denote the volume of $\mathbb{R}/L$ by $\mathrm{vol}(L)$. Note that $\mathrm{vol}(L)$ can be computed as $\prod_{i=1}^{d}\langle\mathbf{b}_i^*, \mathbf{b}_i^*\rangle$ for any choice of basis. We write $\mathbf{B}_{[\ell,r]}$, with $\ell < r$, for the block $(\mathbf{b}_\ell, \mathbf{b}_{\ell+1}, \ldots \mathbf{b}_r)$. We write $\mathbf{B}_{[\ell,r]}^*$ for the projected block $(\pi_\ell(\mathbf{b}_\ell), \ldots, \pi_\ell(\mathbf{b}_r))$.

[2]In so far as computation is a physical process, it should be obvious how to define "unit-cost communication" and "distance" within a model of computation. That said, there's no harm in viewing, say, the lambda calculus as a non-local model of computation.

6.1 Attacks based on lattices

The best known attacks on NTRU begin from the observation that the set

$$M_{\mathbf{h},\mathbf{s}} := \{(\mathbf{a},\mathbf{b}) \in \mathbb{Z}^{2n} \;:\; \mathbf{a} \circledast \mathbf{h} \equiv \mathbf{b} \circledast \mathbf{s} \pmod{q}\}$$

is a lattice. The choice of sample spaces in NTRU-HPS ensures that, for each public key $\mathbf{h}$, there is some ternary vector $(\mathbf{f},\mathbf{g}) \in M_{\mathbf{h},3}$. Hence, a key-only attack on NTRU-HPS might involve searching for short vectors in $M_{\mathbf{h},3}$. Likewise, a key-only attack on NTRU-HRSS might involve $M_{\mathbf{h},3\mathbf{\Phi}_1}$. A decryption attack on either system might involve a search for vectors close to $(\mathbf{0},\mathbf{c})$ in $M_{\mathbf{h},1}$. We will suppress the choice of $\mathbf{s}$ for the remainder.

Attacks involving $M_{\mathbf{h}}$ have gone through several reformulations. Hoffstein, Pipher, and Silverman considered exact key recovery in [19]. Coppersmith and Shamir later observed that *any* short vector in $M_{\mathbf{h}}$, not just $(\mathbf{f},\mathbf{g})$, would be useful in attacks [10]. Coppersmith and Shamir also observed that the norm of the target vector could be decreased by considering the projection of $M_{\mathbf{h}}$ orthogonal to $(\mathbf{\Phi}_n,\mathbf{\Phi}_n)$. May [30] reformulated the key-recovery problem as an instance of unique-SVP and considered other projected sublattices. For example, he considered "dimension reduced" lattices obtained by projection orthogonal to a set of standard basis vectors. May [30], and May and Silverman [31], further observed that attackers can trade the cost of lattice reduction against the probability of guessing a projection that results in an easier lattice problem. Howgrave-Graham's hybrid attack combines May's dimension reduction with an exhaustive search strategy that further decreases the amount of lattice reduction that needs to be performed [22].

In parallel with these reformulations, there have been tremendous advances in algorithms for lattice problems. We will mention a few of the key results below.

6.2 Quality of lattice reduction

There are various ways to measure the quality of a basis $\mathbf{B} \subset L$. The least we can ask for is that the basis is size-reduced, which simply says that $\mathbf{b}_j$ is the shortest vector in $\mathbf{b}_j + \mathbb{Z}\mathbf{b}_i$ for each $i < j$. The Lenstra–Lenstra–Lovász (LLL) algorithm produces size-reduced bases with an additional guarantee that $\mathbf{b}_i^*$ is a shortest vector in the projected block $\mathbf{B}^*_{[i,i+1]}$ for all $1 \le i < d$. Such bases are called LLL reduced. Similar notions of reduction can be defined relative to larger block sizes, i.e. with the condition on $\mathbf{B}^*_{[i,i+1]}$ replaced by a condition on $\mathbf{B}^*_{[i,i+b-1]}$. In the extreme, with block size equal to rank, a Hermite–Korkine–Zolatarev (HKZ) reduced basis is a size-reduced bases for which $\mathbf{b}_1$ is a shortest vector in L, $\mathbf{b}_2^*$ is a shortest vector in $\pi_{\mathbf{B},2}(L)$, and so on. Between the two extremes there are a variety of algorithms that produce *block reduced* bases by solving polynomially many instances of SVP in projected blocks.

The quality of block reduced bases can be evaluated in terms of the *root hermite factor* $\delta = \left(|\mathbf{b}_1| / \operatorname{vol}(L)^{1/d}\right)^{1/d}$ or the basis *profile* $(|\mathbf{b}_1^*|, |\mathbf{b}_2^*|, \ldots, |\mathbf{b}_d^*|)$. In practice, these quantities are estimated using the Gaussian heuristic and the geometric series assumption.

Gaussian heuristic The Gaussian heuristic states that the shortest vectors of a unit-volume lattice of rank b are of length approximately

$$\operatorname{gh}(b) = (\operatorname{vol}(\text{unit ball in dim. } b))^{-1/b} = \frac{\Gamma(b/2+1)^{1/b}}{\sqrt{\pi}}.$$

Geometric series assumption The geometric series assumption was introduced by Schnorr in the analysis of his Block Korkine–Zolatarev (BKZ) algorithm [36]. The output of BKZ with block size b is a BKZ-b reduced basis, $\mathbf{B} = (\mathbf{b}_1, \ldots, \mathbf{b}_d)$. Let $\delta(b) = \operatorname{gh}(b)^{1/(b-1)}$. The geometric series assumption states that the profile of $\mathbf{B}$ is close to a particular geometric series:

$$(|\mathbf{b}_1^*|, \ldots, |\mathbf{b}_d^*|) \approx \operatorname{vol}(L)^{1/d} \cdot (\delta(b)^d, \ldots, \delta(b)^{-d}).$$

The assumption is thought to be accurate when $50 \le b \ll d$.

BKZ is heuristically expected to make polynomially many calls to SVP oracles in dimensions $\le b$ before outputting a BKZ-b reduced basis. Other algorithms, like slide reduction [14] and DBKZ

[32], provably make polynomially many calls to SVP oracles in dimension b. These algorithms achieve slightly different notions of block reduction, but to simplify our discussion we will only refer to BKZ-b reduced bases. We apply the geometric series assumption to bases output by any algorithm that makes polynomially many calls to an SVP oracle in dimension $\leq b$.

6.3 Cost of SVP algorithms

Table 3 gives asymptotic RAM operation counts and memory usage for state-of-the-art sieve algorithms. HKL18 refers to the algorithm of Herold, Kirshanova, and Laarhoven [16]; BGJ15 refers to the algorithm of Becker, Gama, and Joux [3]; BDGL16 refers to the algorithm of Becker, Ducas, Gama, and Laarhoven [4].

The non-asymptotic performance of these algorithms is poorly understood. For instance, it is not at all clear which sieve performs best in, say, dimension 300. The current records in lattice reduction challenges have been set by simpler variants of these sieves [1], and these variants do not achieve the asymptotic complexities listed in Table 3. Nevertheless, we use the asymptotic operation count of BDGL16 to estimate the non-asymptotic operation count of the best sieve in fixed dimension. Also note that the memory usage for BDGL16 given in Table 3 is for a variant of the algorithm that "performs quite poorly in practice" according to the authors [4].

Cost of sieving in the RAM model		
Sieve	$\log_2(\text{operations})$	$\log_2(\text{memory})$
HKL18	$(0.3588\ldots + o(1)) \cdot b$	$(0.1887\cdots + o(1)) \cdot b$
BGJ15	$(0.3112\ldots + o(1)) \cdot b$	$(0.2075\ldots + o(1)) \cdot b$
BDGL16	$(0.2925\ldots + o(1)) \cdot b$	$(0.2075\ldots + o(1)) \cdot b$

Table 3:

Table 4 gives the asymptotic operation count and memory usage for sieves that do not require random access. Systolic NV08 refers to Kirchner's observation that the Nguyen–Vidick sieve can be implemented on a ring of data processing units [27]. The simplified variant of BGJ15 from [1] is called `bgj1`. Local `bgj1`* refers to Ducas' conjecture [12] that `bgj1` can be implemented locally with complexity matching its RAM complexity.

Cost of sieving in local models		
Sieve	$\log_2(\text{operations})$	$\log_2(\text{memory})$
Systolic NV08	$(0.4150\ldots + o(1)) \cdot b$	$(0.2075\ldots + o(1)) \cdot b$
Local `bgj1`*	$(0.3496\ldots + o(1)) \cdot b$	$(0.2075\ldots + o(1)) \cdot b$

Table 4:

6.3.1 Effect of quantum search

Quantum variants of sieve algorithms have been studied, e.g. [29]. All of the known improvements come from applying Grover search to exponentially large lists of vectors. The improvements thus rely on unit-cost superposition queries to classical memory (QRAM), which is an even stronger non-local resource than standard Clifford+T quantum circuits[3] Even with this very strong resource the best

[3] Clifford+T circuits already require non-locality in the form of controlled-NOT gates that can be applied between arbitrary qubits. An analogous form of non-locality is provided in the boolean circuit model. A general purpose random access memory requires a number of gates that grows linearly with the memory size in either model. We maintain that QRAM is a stronger resource. If a program is "compiled" to a boolean circuit, a bit access with fixed address can be replaced by a single fan-out gate (or similar). On the other hand, if a program is compiled to a Clifford+T circuit, a bit access with a fixed superposition of addresses requires a number of controlled-NOT gates that grows linearly with the number of addresses in the support of the superposition. Quantum variants of sieve algorithms repeatedly query exponentially many addresses in superposition.

claimed operation count is $2^{(0.265\ldots+o(1))\cdot b}$ [28]. We do not believe that this translates into an attack that is relevant to the security of NTRU in practice.

6.4 The cost of lattice attacks

Recent work on sieve algorithms has made it clear that the community's understanding of the asymptotic cost of solving the shortest vector problem is still in flux. Consequently, there has been a push to ignore polynomial factors in the cost of lattice reduction and to focus on the cost of solving SVP in projected blocks of a given size. We follow this approach here, and we assess only the *core-SVP* cost of lattice attacks. Core-SVP cost was defined in [2]; the core-SVP cost of block reduction with block size b is the cost of one call to an SVP solver in dimension b.

6.4.1 Short vectors in NTRU lattices

We write $(\mathbf{f}, \mathbf{g})$ for the target vector[4]. We write s for (a lower bound on) the expected size of a coefficient in $(\mathbf{f}, \mathbf{g})$. For NTRU-HPS we take $s = (q/8-2)/(n-1)$. This gives $1/2$ for ntruhps2048509, $127/338$ for ntruhps2048677, and $51/82$ for ntruhps4096821. The NTRU-HRSS-KEM submission proposed a sampler with $s = 10/16$, and we use this value for NTRU-HRSS even though our recommended sampler (1.10.3) produces vectors with $s = 170/256$. This results in a slight security underestimate.

6.4.2 Costing the primal attack

The *primal attack* on NTRU applies May's dimension reduction and Coppersmith and Shamir's projection orthogonal to $(\mathbf{\Phi}_n, \mathbf{\Phi}_n)$. The dimension reduction is chosen based on a volume parameter m. The attack computes a BKZ-b reduced basis $\mathbf{V} = (\mathbf{v}_1, \ldots, \mathbf{v}_d)$ of a projected sublattice of $M_{\mathbf{h}}$ of rank $d = (n-1)+m$ and volume q^m. The block size b is chosen to be the minimal value for which $|\mathbf{v}^*_{d-b}| \geq s\sqrt{b}$ under the geometric series assumption. In our analysis, we minimize the block size over all choices of m.

	Non-local		Local			
	b	$\lfloor 0.2925 \cdot b \rfloor$	b	$\lfloor 0.3496 \cdot b \rfloor$	b	$\lfloor 0.4150 \cdot b \rfloor$
ntruhps2048509	364	106	364	127	364	151
ntruhrss701	470	136	470	164	470	195
ntruhps2048677	496	145	496	173	496	205
ntruhps4096821	612	179	612	213	612	253

Table 5: Core-SVP cost of the primal attack

6.4.3 Costing the hybrid attack

Howgrave-Graham's hybrid attack [22] partitions $M_{\mathbf{h}}$ into three sublattices L_1, L_2, and L_3. The partition is chosen according to two parameters: a combinatorial search parameter k, and a volume parameter m. The sublattice L_1 is the integer span of $\{(\mathbf{x}^i, \mathbf{x}^i \circledast \mathbf{h}) \ : \ 0 \leq i < k\}$; it is of rank k and unit volume. The sublattice L_2 is the integer span of $\{(\mathbf{x}^i, \mathbf{x}^i \circledast \mathbf{h}) \ : \ k \leq i < n\} \cup \{(\mathbf{0}, q\mathbf{x}^i) \ : \ 0 \leq i < m\}$; it is of rank $d = n - k + m$ and volume q^m. The sublattice L_3 is the integer span of the remaining q vectors; it is of rank $n - m$ and volume q^{n-m}.

The attacker produces a BKZ-b reduced basis $\mathbf{V} = (\mathbf{v}_1, \ldots, \mathbf{v}_d)$ for L_2. The block size b is chosen so that $|\mathbf{v}^*_d| \geq 2s$. Heuristically, we expect that the result of size-reducing $\sum_{i=0}^{k-1} f_i \cdot (\mathbf{x}^i, \mathbf{x}^i \circledast \mathbf{h})$ against $\mathbf{V}$ will be equivalent to $(\mathbf{f}, \mathbf{g})$ modulo q. The attacker can choose k to balance the cost of lattice reduction against the cost of guessing the first k coefficients of $\mathbf{f}$. The enumeration of vectors in L_1 can be pruned based on the secret key distribution. The sublattice L_3 can be used in a meet-in-the-middle search strategy [22] or for a "checking routine" in a quantum search.

[4] The primal attack can be adapted for message recovery using Kannan's embedding technique. The cost is essentially identical to key recovery, and can be computed using the scripts provided with the submission package.

For each k we compute the b for which the cost of a single call to an SVP solver in dimension b matches the cost of guessing k coefficients of $\mathbf{f}$. Among these values of b we find the minimum for which $|\mathbf{v}_d^*| \geq 2s$ under the geometric series assumption.

We assume that guessing k coefficients of $\mathbf{f}$ costs $2^{(1/2+o(1))\nu k}$ operations where νk is Shannon entropy of the first k coefficients of f (with randomness taken over the coins in key generation). For non-local models of computation we assume this search is performed using a classical meet-in-the-middle strategy. For local models of computation we assume this search is performed using (parallel) quantum search with a 2^{96} gate limit on circuit depth.

In our analysis, we take ν to be the Shannon entropy of f_1. This will slightly overestimate νk for NTRU-HPS, however we expect it to be a good approximation when $k \ll n$. A larger source of error is that we ignore the $n-1$ other short vectors of the form $(\mathbf{x}^i \circledast \mathbf{f}, \mathbf{x}^i \circledast \mathbf{g})$ with $1 \leq i \leq n-1$. Both of these sources of error lead to security overestimates. However, we believe they are more than compensated for by security underestimates that come from 1) costing only a single call to the SVP solver, and 2) ignoring the probability that the attack fails. See [38] for a detailed discussion of the failure probability.

	Non-local		Local			
	b	$\lfloor 0.2925 \cdot b \rfloor$	b	$\lfloor 0.3495 \cdot b \rfloor$	b	$\lfloor 0.4150 \cdot b \rfloor$
ntruhps2048509	359	105	351	122	335	139
ntruhrss701	459	134	465	162	448	185
ntruhps2048677	494	144	–	–	483	200
ntruhps4096821	612	178	–	–	–	–

Table 6: Core-SVP cost of the hybrid attack. Dashes indicate that hybrid attack is outperformed by the primal attack. The poor performance of the hybrid attack in these cases is due to the depth restriction on quantum search.

6.5 Rationale for security categories

The security categories given in Section 5.3 are based on the $\lfloor 0.2925 \cdot b \rfloor$ and $\lfloor 0.4150 \cdot b \rfloor$ columns of Tables 5 and 6. We emphasize that these tables represent very conservative security estimates, and the values in these tables do not have units of bit operations. NIST's recommended classical gate count thresholds for security categories 1, 3, and 5 are 2^{143}, 2^{207}, and 2^{272}, respectively. In some cases, e.g. the assignment of category 1 to ntruhps2048677 relative to the RAM model, our estimates clearly exceed these thresholds. In other cases, e.g. the assignment of category 3 to ntruhps4096821 relative to the RAM model, we are assuming that the overhead that is missing from our analysis covers the gap.

The $\lfloor 0.3495 \cdot b \rfloor$ columns in Tables 5 and 6 give security estimates relative to Ducas' conjecture about the complexity of bgj1 in a local model. If this conjecture is true, then our security categories in local models may need to be revised downwards.

7 Advantages and limitations

Our submission has a number of advantages.

- It is **correct**. The IND-CCA2 KEM always establishes a key; it never aborts because of a decryption failure. This simplifies the analysis of the scheme, and makes it an attractive drop-in replacement for KEMs that are in use today.

- It is **well studied**. Among the assumptions underlying post-quantum cryptosystems, the OW-CPA security of NTRU is well studied. NTRU, and similar systems, have frequently been used to benchmark new techniques in lattice reduction [36, 7, 15, 8]. This history of concrete cryptanalysis should inspire some confidence in NTRU. The tight reduction from the IND-CCA2 security of our KEM to the OW-CPA security of the ANTS'98 DPKE means that this history is relevant to the concrete security of our KEM.

- It is **flexible**. The underlying DPKE can be parameterized for a variety of use cases with different size, security, and efficiency requirements. We have discussed this in Section 2.4 and depicted some of the trade-offs in Figures 11, 12, and 13.
- It is **simple**. The DPKE has only two parameters, n and q, and can be described entirely in terms of simple integer polynomial arithmetic. The transformation to an IND-CCA2 secure KEM is conceptually simple.
- It is **fast**. ntruhrss701 was among the fastest submissions in the first round. We expect that this will remain true in the second round.
- It is **compact**. Our ntruhps2048677 parameter set achieves level one security with a wide security margin, level three security under a reasonable assumption, and has public keys and ciphertexts of only 930 bytes.
- It is **patent free**. The relevant patents have expired.

It also has several limitations.

- NTRU is unlikely to be the fastest submission, unlikely to be the most compact submission, and unlikely to be the most secure submission. However, it will be competitive on products of these measures.
- The choice of optimal parameters for NTRU is currently limited by a poor understanding of the non-asymptotic behavior of new algorithms for SVP. This is a limitation that is shared with all lattice based cryptosystems.
- There is structure in NTRU that is not *strictly* necessary, and this may also be seen as a limitation. It is possible to eliminate the structure of a sparse ternary secret at a cost in terms of correctness or compactness. It is also possible to eliminate the cyclotomic structure of the ring; comparisons with NTRU Prime will reveal the cost of doing so.

Addition parameter plots

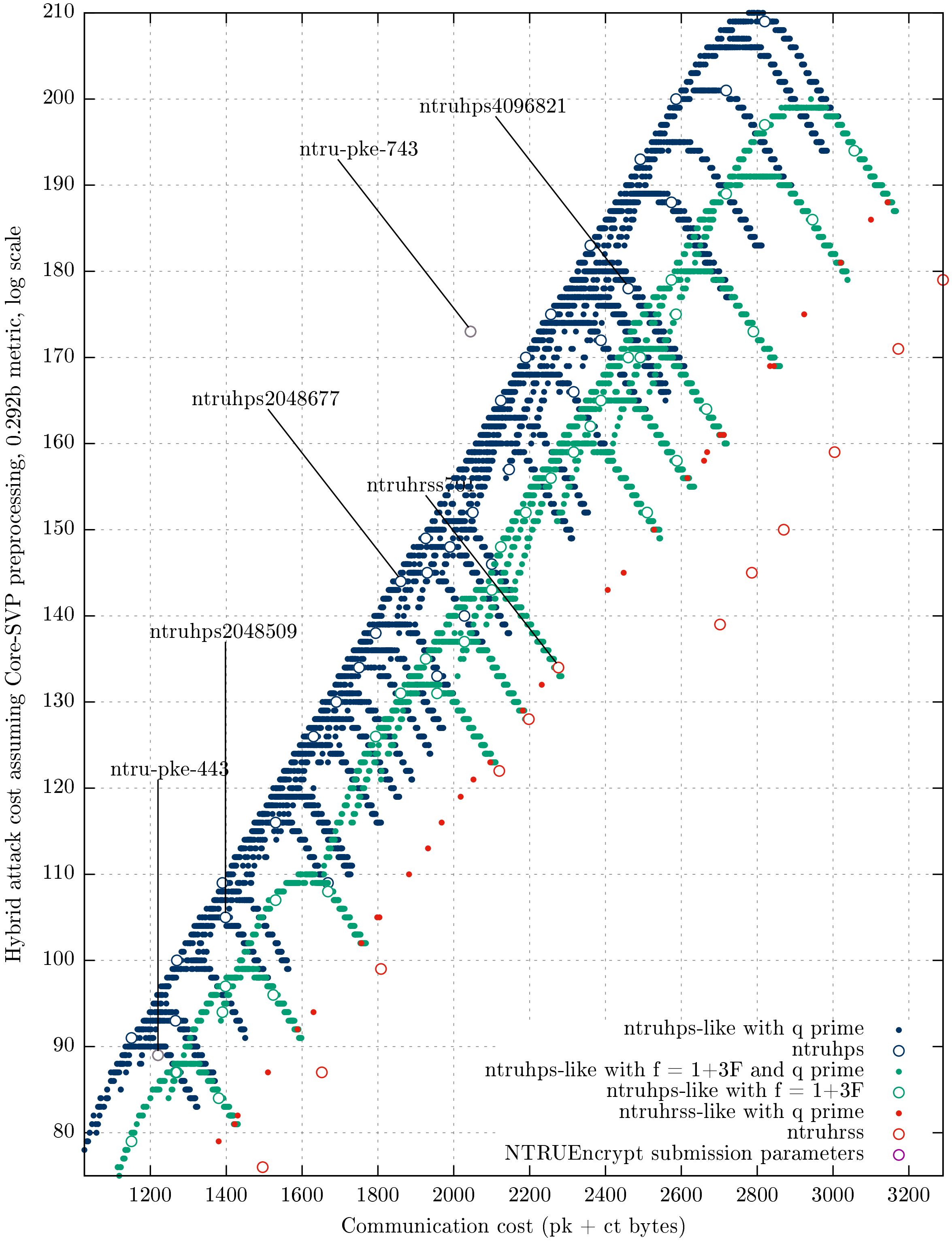


Figure 12: The same analysis as Figure 11 with a wider range of weight parameters.

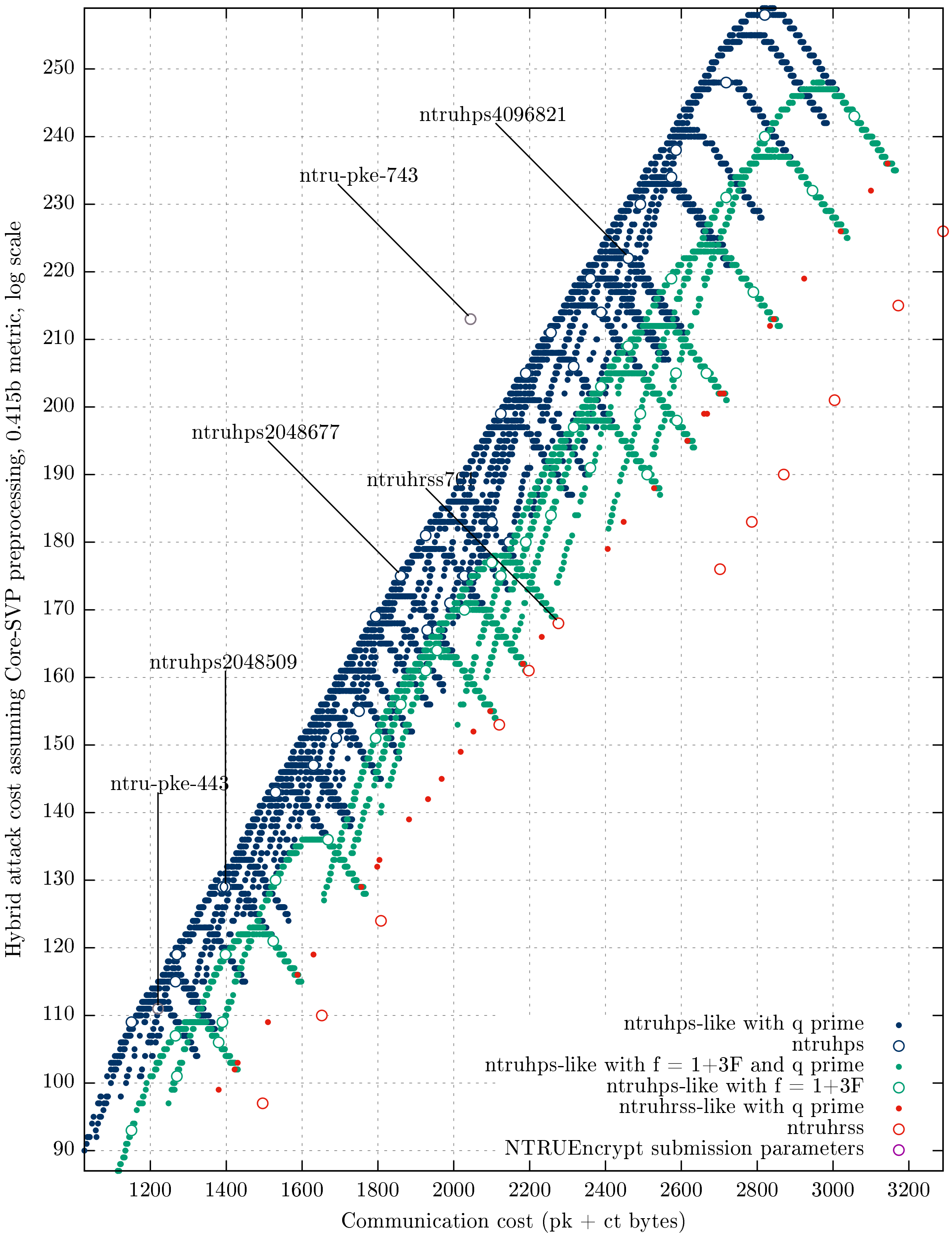


Figure 13: The same parameters as Figure 12, but using a cost of $2^{0.415b}$ for SVP in dimension b. More expensive lattice reduction degrades the apparent benefit of a low weight parameter.

References

[1] Martin R. Albrecht, Léo Ducas, Gottfried Herold, Elena Kirshanova, Eamonn W. Postlethwaite, and Marc Stevens. The general sieve kernel and new records in lattice reduction. Cryptology ePrint Archive, Report 2019/089, 2019. https://eprint.iacr.org/2019/089. 33

[2] Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. Post-quantum key exchange – a new hope. In Thorsten Holz and Stefan Savage, editors, *Proceedings of the 25th USENIX Security Symposium*. USENIX Association, 2016. https://cryptojedi.org/papers/#newhope. 34

[3] Antoine Joux Anja Becker, Nicolas Gama. Speeding-up lattice sieving without increasing the memory, using sub-quadratic nearest neighbor search. Cryptology ePrint Archive, Report 2015/522, 2015. https://eprint.iacr.org/2015/522. 33

[4] Anja Becker, Léo Ducas, Nicolas Gama, and Thijs Laarhoven. New directions in nearest neighbor searching with applications to lattice sieving. In *27th Annual ACM-SIAM Symposium on Discrete Algorithms*. ACM-SIAM, 2016. 33

[5] Daniel J. Bernstein and Edoardo Persichetti. Towards kem unification. Cryptology ePrint Archive, Report 2018/526, 2018. https://eprint.iacr.org/2018/526. 4, 25, 26, 30

[6] Daniel J. Bernstein and Bo-Yin Yang. Fast constant-time gcd computation and modular inversion. Cryptology ePrint Archive, Report 2019/266, 2019. https://eprint.iacr.org/2019/266. 12

[7] Johannes Buchmann and Christoph Ludwig. Practical lattice basis sampling reduction. In Florian Hess, Sebastian Pauli, and Michael Pohst, editors, *Algorithmic Number Theory – ANTS-VII*, LNCS, pages 222–237. Springer, 2006. https://eprint.iacr.org/2005/072. 35

[8] Yuanmi Chen and Phong Q. Nguyen. BKZ 2.0: Better lattice security estimates. In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology – ASIACRYPT 2011*, volume 7073 of *LNCS*, pages 1–20. Springer, 2011. http://www.iacr.org/archive/asiacrypt2011/70730001/70730001.pdf. 35

[9] Consortium for Efficient Embedded Security. EESS #1: Implementation aspects of NTRU-Encrypt and NTRUSign v. 2.0, June 2003. http://grouper.ieee.org/groups/1363/lattPK/submissions/EESS1v2.pdf. 4

[10] Don Coppersmith and Adi Shamir. Lattice attacks on NTRU. In Walter Fumy, editor, *Advances in Cryptology – EUROCRYPT '97*, volume 1233 of *LNCS*, pages 52–61. Springer, 1997. http://dx.doi.org/10.1007/3-540-69053-0_5. 32

[11] Alexander W. Dent. A designer's guide to KEMs. In Kenneth G. Paterson, editor, *Cryptography and Coding*, volume 2898 of *LNCS*, pages 133–151. Springer, 2003. http://www.cogentcryptography.com/papers/designer.pdf. 24

[12] Léo Ducas. Shortest vector from lattice sieving: a few dimensions for free. Conference talk, April 2018. https://eurocrypt.iacr.org/2018/Slides/Monday/TrackB/01-01.pdf. 33

[13] Eiichiro Fujisaki and Tatsuaki Okamoto. How to enhance the security of public-key encryption at minimum cost. In *Public Key Cryptography*, pages 53–68, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg. 23

[14] Nicolas Gama and Phong Q Nguyen. Finding short lattice vectors within mordell's inequality. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 207–216. ACM, 2008. 32

[15] Nicolas Gama and Phong Q. Nguyen. Predicting lattice reduction. In Nigel Smart, editor, *Advances in Cryptology – EUROCRYPT 2008: 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Istanbul, Turkey, April 13-17, 2008. Proceedings*, LNCS, pages 31–51. Springer, 2008. https://www.iacr.org/archive/eurocrypt2008/49650031/49650031.pdf. 35

[16] Gottfried Herold, Elena Kirshanova, and Thijs Laarhoven. Speed-ups and time–memory trade-offs for tuple lattice sieving. In Michel Abdalla and Ricardo Dahab, editors, *Public-Key Cryptography – PKC 2018*, pages 407–436, Cham, 2018. Springer International Publishing. 33

[17] Philip S. Hirschhorn, Jeffrey Hoffstein, Nick Howgrave-Graham, and William Whyte. Choosing NTRUEncrypt parameters in light of combined lattice reduction and MITM approaches. In Michel Abdalla, David Pointcheval, Pierre-Alain Fouque, and Damien Vergnaud, editors, *Applied Cryptography and Network Security – ACNS 2009*, volume 5536 of *LNCS*, pages 437–455. Springer, 2009. https://eprint.iacr.org/2005/045. 4

[18] Jeff Hoffstein, Jill Pipher, John M. Schanck, Joseph H. Silverman, William Whyte, and Zhenfei Zhang. Choosing parameters for NTRUEncrypt. In Helena Handschuh, editor, *Cryptographers' Track at the RSA Conference – CTA-RSA 2017*, volume 10159 of *LNCS*, pages 3–18. Springer, 2017. https://eprint.iacr.org/2015/708. 4

[19] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. NTRU: A new high speed public key cryptosystem, 1996. draft from at CRYPTO '96 rump session. http://web.securityinnovation.com/hubfs/files/ntru-orig.pdf. 4, 21, 25, 32

[20] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. NTRU: A ring-based public key cryptosystem. In Joe P. Buhler, editor, *Algorithmic Number Theory – ANTS-III*, volume 1423 of *LNCS*, pages 267–288. Springer, 1998. http://dx.doi.org/10.1007/BFb0054868. 4

[21] Dennis Hofheinz, Kathrin Hövelmanns, and Eike Kiltz. A modular analysis of the fujisaki-okamoto transformation. In Yael Kalai and Leonid Reyzin, editors, *Theory of Cryptography*, pages 341–371, Cham, 2017. Springer International Publishing. 4, 23, 25, 30

[22] Nick Howgrave-Graham. A hybrid lattice-reduction and meet-in-the-middle attack against NTRU. In Alfred Menezes, editor, *Advances in Cryptology – CRYPTO 2007*, volume 4622 of *LNCS*, pages 150–169. Springer, 2007. http://www.iacr.org/archive/crypto2007/46220150/46220150.pdf. 32, 34

[23] Nick Howgrave-Graham, Joseph H. Silverman, Ari Singer, and William Whyte. NAEP: Provable security in the presence of decryption failures. Cryptology ePrint Archive, Report 2003/172, 2003. https://eprint.iacr.org/2003/172. 21

[24] Nick Howgrave-Graham, Joseph H. Silverman, and William Whyte. Choosing parameter sets for NTRUEncrypt with NAEP and SVES-3. In Alfred Menezes, editor, *Cryptographers' Track at the RSA Conference – CT-RSA 2005*, volume 3376 of *LNCS*, pages 118–135. Springer, 2005. https://eprint.iacr.org/2005/045. 4

[25] Andreas Hülsing, Joost Rijneveld, John Schanck, and Peter Schwabe. High-speed key encapsulation from NTRU. In Wieland Fischer and Naofumi Homma, editors, *Cryptographic Hardware and Embedded Systems – CHES 2017*, LNCS. Springer, 2017. http://cryptojedi.org/papers/#ntrukem. 4, 6, 12, 13, 24

[26] IEEE. IEEE Standard Specification for Public Key Cryptographic Techniques Based on Hard Problems over Lattices. IEEE Std 1363.1-2008, 2009. http://dx.doi.org/10.1109/IEEESTD.2009.4800404. 4

[27] Paul Kirchner. Re: Sieving vs. enumeration. Message to cryptanalytic-algorithms mailing list, May 2016. https://groups.google.com/forum/#!msg/cryptanalytic-algorithms/BoSRL0uHIjM/wAkZQlwRAgAJ. 33

[28] Thijs Laarhoven. *Search problems in cryptography*. PhD thesis, Eindhoven University of Technology, 2015. http://www.thijs.com/docs/phd-final.pdf. 34

[29] Thijs Laarhoven, Michele Mosca, and Joop van de Pol. Finding shortest lattice vectors faster using quantum search. *Designs, Codes and Cryptography*, 77(2):375–400, 2015. https://eprint.iacr.org/2014/907/. 33

[30] Alexander May. Cryptanalysis of NTRU, 1999. https://www.cits.ruhr-uni-bochum.de/imperia/md/content/may/paper/cryptanalysisofntru.ps. 32

[31] Alexander May and Joseph H. Silverman. Dimension reduction methods for convolution modular lattices. In Joseph H. Silverman, editor, *Cryptography and Lattices: International Conference – CaLC 2001*, volume 2146 of *LNCS*, pages 110–125. Springer, 2001. http://dx.doi.org/10.1007/3-540-44670-2_10. 32

[32] Daniele Micciancio and Michael Walter. Practical, predictable lattice basis reduction. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 820–849. Springer, 2016. 33

[33] NIST. FIPS PUB 202 – SHA-3 standard: Permutation-based hash and extendable-output functions, 2015. http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf. 9

[34] Edoardo Persichetti. Improving the efficiency of code-based cryptography. Ph.D. thesis, 2012. http://persichetti.webs.com/Thesis%20Final.pdf. 25

[35] Tsunekazu Saito, Keita Xagawa, and Takashi Yamakawa. Tightly-secure key-encapsulation mechanism in the quantum random oracle model. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 520–551. Springer, 2018. 4, 20, 24, 25, 30

[36] Claus Peter Schnorr. Lattice reduction by random sampling and birthday methods. In Helmut Alt and Michel Habib, editors, *Symposium on Theoretical Aspects of Computer Science – STACS 2003*, volume 2607 of *LNCS*, pages 145–156. Springer, 2003. https://pdfs.semanticscholar.org/a323/ef7dcaaf2f8d4a52b63393986ba23140faa6.pdf. 32, 35

[37] Ehsan Ebrahimi Targhi and Dominique Unruh. Quantum security of the Fujisaki-Okamoto and OAEP transforms. Cryptology ePrint Archive, Report 2015/1210, 2015. https://eprint.iacr.org/2015/1210. 23, 24, 27

[38] Thomas Wunderer. Revisiting the hybrid attack: Improved analysis and refined security estimates. Cryptology ePrint Archive, Report 2016/733, 2016. https://eprint.iacr.org/2016/733. 35

Changelog:

2020-07-30

- Added Tsunekazu Saito, Keita Xagawa, and Takashi Yamakawa as co-authors.
- Section 1.6: Corrected value of sample_fixed_type_bits for ntruhps4096821
- Section 1.11.4: Fixed typo in Line 5 DPKE_Decrypt (unpack_Rq0 → unpack_Sq).
- Section 1.11.4: Added Note 2.
- Section 1.12.3: Fixed description of private key parsing.
- Section 2.4.1: Added decryption failure rates for improperly generated keys.
- Section 3.2: Added performance numbers for AVX2 implementations of ntruhps*.
- Section 6: Added disclaimer about forthcoming revised security analysis.
- Section 6: Removed subsection 'Non-asymptotic memory usage of sieving'
- Section 6.2: Corrected typo in definition of geometric series assumption.