NTRU Prime: round 2

20190330

Principal submitter

This submission is from the following team, listed in alphabetical order:

- Daniel J. Bernstein, University of Illinois at Chicago
- Chitchanok Chuengsatiansup, INRIA and ENS de Lyon
- Tanja Lange, Technische Universiteit Eindhoven
- Christine van Vredendaal, Technische Universiteit Eindhoven

E-mail address (preferred): `authorcontact-ntruprime@box.cr.yp.to`

Telephone (if absolutely necessary): +1-312-996-3422

Postal address (if absolutely necessary): Daniel J. Bernstein, Department of Computer Science, University of Illinois at Chicago, 851 S. Morgan (M/C 152), Room 1120 SEO, Chicago, IL 60607–7053.

Auxiliary submitters: There are no auxiliary submitters. The principal submitter is the team listed above.

Inventors/developers: The inventors/developers of this submission are the same as the principal submitter. Relevant prior work is credited below where appropriate.

Owner: Same as submitter.

Signature: $\times$. See also printed version of "Statement by Each Submitter".

Document generated with the help of `pqskeleton` version 20190309.

Contents

1 Introduction

A 2015 algorithm breaks dimension-N SVP (under plausible assumptions) in time $2^{(c+o(1))N}$ as $N \to \infty$ with $c \approx 0.292$. See [16]. For comparison, the best algorithm known just five years earlier had a much worse $c \approx 0.415$, and the best algorithm known just ten years before that took time $2^{\Theta(N \log N)}$.

Gentry's original FHE system at STOC 2009, with standard "cyclotomic" choices of rings, is now known (again under plausible assumptions) to be broken in polynomial time by a quantum algorithm. See [27]. Peikert claimed in 2015 that the weakness in Gentry's system was specific to Gentry's short generators and inapplicable to Ideal-SVP:

> Although cyclotomics have a lot of structure, nobody has yet found a way to exploit it in attacking Ideal-SVP/BDD ... For commonly used rings, principal ideals are an extremely small fraction of all ideals. ... The weakness here is not so much due to the structure of cyclotomics, but rather to the extra structure of principal ideals that have short generators.

However, the attack was then combined with further features of cyclotomics to break Ideal-SVP (again under plausible assumptions) with approximation factor $2^{N^{1/2+o(1)}}$, a terrifying advance compared to the previous $2^{N^{1+o(1)}}$. See [39].

As these attack examples illustrate, the security of lattice-based cryptography is not well understood. There are serious risks of further advances in

- SVP algorithms,
- algorithms that exploit the "approximation factors" used in cryptography,
- algorithms that exploit the structure of cryptographic problems such as LWE,
- algorithms that exploit the multiplicative structure of *efficient* cryptographic problems such as Ring-LWE,
- algorithms that exploit the structure of these problems for the *specific* rings chosen by users, and
- algorithms to break cryptosystems without breaking these problems.

The point of this submission is that the attack surface in lattice-based cryptography can be significantly reduced with only a minor loss of efficiency. In fact, despite the extra security criteria imposed below, the two cryptosystems in this submission are two of the smallest and fastest lattice-based cryptosystems.

2 General algorithm specification (part of 2.B.1)

This submission provides two KEMs: "Streamlined NTRU Prime" and "NTRU LPRime". **Description modularized for round 2, and CCA transforms modified.**

2.1 Shared notation

In each system defined below, three of the parameters are a prime number p, a prime number q, and a positive integer w. The parameter restrictions below imply that $q \geq 17$, that $w \leq p$, and that $x^p - x - 1$ is irreducible in the polynomial ring $(\mathbb{Z}/q)[x]$.

We abbreviate the ring $\mathbb{Z}[x]/(x^p - x - 1)$, the ring $(\mathbb{Z}/3)[x]/(x^p - x - 1)$, and the field $(\mathbb{Z}/q)[x]/(x^p - x - 1)$ as $\mathcal{R}$, $\mathcal{R}/3$, and $\mathcal{R}/q$ respectively. We refer to an element of $\mathcal{R}$ as **small** if all of its coefficients are in $\{-1, 0, 1\}$, and **weight w** if exactly w of its coefficients are nonzero. Define Short as the set of small weight-w elements of $\mathcal{R}$.

Define $\mathsf{Rounded}$ as the set of polynomials $r_0 + r_1x + \cdots + r_{p-1}x^{p-1} \in \mathcal{R}$ where each coefficient r_i is in $\{-(q-1)/2, \ldots, -6, -3, 0, 3, 6, \ldots, (q-1)/2\}$ for $q \in 1 + 3\mathbb{Z}$, or in $\{-(q+1)/2, \ldots, -6, -3, 0, 3, 6, \ldots, (q+1)/2\}$ for $q \in 2 + 3\mathbb{Z}$.

Define $\mathsf{Round} : \mathcal{R}/q \to \mathsf{Rounded}$ as follows: if $a_i \in \{-(q-1)/2, \ldots, (q-1)/2\}$, and r_i is the element of $3\mathbb{Z}$ closest to a_i, then $\mathsf{Round}(a_0 + a_1x + \cdots + a_{p-1}x^{p-1}) = r_0 + r_1x + \cdots + r_{p-1}x^{p-1}$. Note that $\mathsf{Round}(r) = r + e$ for some small $e \in \mathcal{R}$.

Further notation varies between Streamlined NTRU Prime and NTRU LPRime, as summarized in the following tables. The underlining distinguishes mathematical objects such as $\mathsf{SecretKeys} = \mathsf{Short} \times \mathcal{R}/3$ or $\mathsf{SecretKeys} = \mathsf{Short}$ from sets of strings such as $\underline{\mathsf{SecretKeys}}$.

	Streamlined NTRU Prime Core	Streamlined NTRU Prime
public-key space	$\mathsf{PublicKeys} = \mathcal{R}/q$	$\mathsf{PublicKeys}' = \underline{\mathsf{PublicKeys}}$
secret-key space	$\mathsf{SecretKeys} = \mathsf{Short} \times \mathcal{R}/3$	$\mathsf{SecretKeys}' = \underline{\mathsf{SecretKeys}} \times \underline{\mathsf{PublicKeys}} \times \underline{\mathsf{Inputs}}$
input space	$\mathsf{Inputs} = \mathsf{Short}$	not applicable (KEM)
ciphertext space	$\mathsf{Ciphertexts} = \mathsf{Rounded}$	$\mathsf{Ciphertexts}' = \underline{\mathsf{Ciphertexts}} \times \mathsf{Confirm}$
key generation	KeyGen	KeyGen'
public-key operation	$\mathsf{Encrypt}$	Encap
secret-key operation	$\mathsf{Decrypt}$	Decap

	NTRU LPRime Core	NTRU LPRime Expand	NTRU LPRime
public-key space	$\mathsf{PublicKeys} = \mathcal{R}/q \times \mathsf{Rounded}$	$\mathsf{PublicKeys}' = \mathsf{Seeds} \times \mathsf{Rounded}$	$\mathsf{PublicKeys}'' = \underline{\mathsf{PublicKeys}'}$
secret-key space	$\mathsf{SecretKeys} = \mathsf{Short}$	$\mathsf{SecretKeys} = \mathsf{Short}$	$\mathsf{SecretKeys}' = \underline{\mathsf{SecretKeys}} \times \underline{\mathsf{PublicKeys}'} \times \underline{\mathsf{Inputs}}$
input space	$\mathsf{Inputs} = \{0,1\}^I$	$\mathsf{Inputs} = \{0,1\}^I$	not applicable (KEM)
ciphertext space	$\mathsf{Ciphertexts} = \mathsf{Rounded} \times (\mathbb{Z}/\tau)^I$	$\mathsf{Ciphertexts} = \mathsf{Rounded} \times (\mathbb{Z}/\tau)^I$	$\mathsf{Ciphertexts}' = \underline{\mathsf{Ciphertexts}} \times \mathsf{Confirm}$
key generation	KeyGen	KeyGen'	KeyGen''
public-key operation	$\mathsf{Encrypt}$	$\mathsf{Encrypt}'$	Encap
secret-key operation	$\mathsf{Decrypt}$	$\mathsf{Decrypt}$	Decap

2.2 Shared theorems

Theorem 1. *Fix integers $p \ge 3$ and $w \ge 1$. Let $g \in \mathbb{Z}[x]$ be a polynomial of degree at most $p-1$ with each coefficient in $\{-1,0,1\}$. Let i be an integer with $0 \le i < p$. Then $x^i g \bmod x^p - x - 1$ has each coefficient in $\{-2,-1,0,1,2\}$.*

Proof. Write g as $\sum_{j=0}^{p-1} g_j x^j$. By assumption $g_0, g_1, \ldots, g_{p-1} \in \{-1,0,1\}$. Observe that $xg \bmod x^p - x - 1 = g_{p-1} + (g_0 + g_{p-1})x + g_1x^2 + \cdots + g_{p-2}x^{p-1}$, which has $g_0 + g_{p-1} \in \{-2,-1,0,1,2\}$ and all other coefficients in $\{-1,0,1\}$. More generally, for $0 \le i < p$, we get $x^i g \bmod x^p - x - 1 = g_{p-i} + (g_{p-i} + g_{p-i-1})x + \cdots + (g_{p-2} + g_{p-1})x^{i-1} + (g_{p-1} + g_0)x^i + g_1x^{i+1} + \cdots + g_{p-i-1}x^{p-1}$ with all coefficients in $\{-2,-1,0,1,2\}$. □

Theorem 2. *Fix integers $p \ge 3$ and $w \ge 1$. Let $r, g \in \mathbb{Z}[x]$ be polynomials of degree at most $p-1$ with all coefficients in $\{-1,0,1\}$. Assume that r has at most w nonzero coefficients. Then $gr \bmod x^p - x - 1$ has each coefficient in the interval $[-2w, 2w]$.*

Proof. Write r as $\sum_{i=0}^{p-1} r_i x^i$. By assumption $r_i \in \{-1,0,1\}$. Define $S = \{i : r_i \ne 0\}$; then S has at most w elements, and $r = \sum_{i\in S} r_i x^i$. Each coefficient of $x^i g \bmod x^p - x - 1$ is in the range $[-2,2]$ by Theorem 2. Each coefficient of $rg \bmod x^p - x - 1 = \sum_{i\in S} r_i x^i g \bmod x^p - x - 1$ is therefore in the range $[-2w, 2w]$. □

Theorem 3. *Fix integers $p \ge 3$ and $w \ge 1$. Let $m, r, f, g \in \mathbb{Z}[x]$ be polynomials of degree at most $p-1$ with all coefficients in $\{-1,0,1\}$. Assume that f and r each have at most w nonzero coefficients. Then $3fm + gr \bmod x^p - x - 1$ has each coefficient in the interval $[-8w, 8w]$.*

Proof. By Theorem 2, $gr \bmod x^p - x - 1$ has each coefficient in the interval $[-2w, 2w]$, and $fm \bmod x^p - x - 1$ has each coefficient in the interval $[-2w, 2w]$. □

2.3 Streamlined NTRU Prime

Streamlined NTRU Prime has two layers. The inner layer is Streamlined NTRU Prime Core, a perfectly correct deterministic PKE. The outer layer is Streamlined NTRU Prime, a perfectly correct KEM.

Applications are expected to use solely the outer layer. We present the two layers separately to assist reviewers and implementors.

2.3.1 Streamlined NTRU Prime Core parameter space

Streamlined NTRU Prime Core has parameters (p, q, w) subject to the following restrictions: p is a prime number; q is a prime number; w is a positive integer; $2p \geq 3w$; $q \geq 16w + 1$; $x^p - x - 1$ is irreducible in the polynomial ring $(\mathbb{Z}/q)[x]$.

2.3.2 Streamlined NTRU Prime Core key generation

Define $\mathsf{PublicKeys} = \mathcal{R}/q$ and $\mathsf{SecretKeys} = \mathsf{Short} \times \mathcal{R}/3$. The following randomized algorithm KeyGen outputs an element of $\mathsf{PublicKeys} \times \mathsf{SecretKeys}$:

- Generate a uniform random small element $g \in \mathcal{R}$. Repeat this step until g is invertible in $\mathcal{R}/3$. (There are various standard ways to test invertibility: for example, one can check divisibility of g by the irreducible factors of $x^p - x - 1$ modulo 3, or one can deduce invertibility as a side effect of various algorithms to compute $1/g$ in $\mathcal{R}/3$.)
- Compute $1/g$ in $\mathcal{R}/3$.
- Generate a uniform random $f \in \mathsf{Short}$. (Note that f is nonzero and hence invertible in $\mathcal{R}/q$, since $w \geq 1$.)
- Compute $h = g/(3f)$ in $\mathcal{R}/q$. (By assumption q is a prime larger than 3, so 3 is invertible in $\mathcal{R}/q$, so $3f$ is invertible in $\mathcal{R}/q$.)
- Output $(h, (f, 1/g)) \in \mathsf{PublicKeys} \times \mathsf{SecretKeys}$.

2.3.3 Streamlined NTRU Prime Core encryption

Define $\mathsf{Inputs} = \mathsf{Short}$ and $\mathsf{Ciphertexts} = \mathsf{Rounded}$. The following deterministic algorithm $\mathsf{Encrypt}$ maps $\mathsf{Inputs} \times \mathsf{PublicKeys}$ to $\mathsf{Ciphertexts}$:

- Input $r \in \mathsf{Inputs}$ and $h \in \mathsf{PublicKeys}$.
- Compute $hr \in \mathcal{R}/q$.
- Output $\mathsf{Round}(hr)$.

2.3.4 Streamlined NTRU Prime Core decryption

The following deterministic algorithm $\mathsf{Decrypt}$ maps $\mathsf{Ciphertexts} \times \mathsf{SecretKeys}$ to Inputs:

- Input $c \in \mathsf{Ciphertexts}$ and $(f, v) \in \mathsf{Short} \times \mathcal{R}/3$.
- Compute $3fc \in \mathcal{R}/q$. (One can multiply c by 3 and then by f, or multiply c by a precomputed $3f$, or multiply c by f and then by 3.)
- View each coefficient of $3fc$ in $\mathcal{R}/q$ as an integer between $-(q-1)/2$ and $(q-1)/2$, and then reduce modulo 3, obtaining a polynomial $e \in \mathcal{R}/3$.
- Multiply by v in $\mathcal{R}/3$.
- Lift ev in $\mathcal{R}/3$ to a small polynomial $r' \in \mathcal{R}$.
- Output r' if r' has weight w. Otherwise output $(1, 1, \ldots, 1, 0, 0, \ldots, 0)$. (Implementors must be careful to avoid leaking secret information through side channels, and in particular must avoid implementing the weight test here as a branch.)

Theorem 4. *Assume that* $\mathsf{KeyGen}()$ *outputs* $(h, k) \in \mathsf{PublicKeys} \times \mathsf{SecretKeys}$. *Then* $\mathsf{Decrypt}(\mathsf{Encrypt}(r, h), k) = r$ *for each* $r \in \mathsf{Inputs}$.

Proof. By definition of $\mathsf{SecretKeys}$, k has the form (f, v) with $f \in \mathsf{Short}$ and $v \in \mathcal{R}/3$. By definition of KeyGen, $v = 1/g$ for some small $g \in \mathcal{R}$, and $h = g/(3f)$ in $\mathcal{R}/q$.

Define $c = \mathsf{Encrypt}(r, h)$. Then $c = \mathsf{Round}(hr) \in \mathsf{Rounded}$ by definition of $\mathsf{Encrypt}$; i.e., c is obtained by rounding the coefficients of hr, viewed as integers between $-(q-1)/2$ and $(q-1)/2$, to the nearest multiples of 3. Hence $c = m + hr$ in $\mathcal{R}/q$, where m is small; so $3fc = 3fm + 3fhr = 3fm + gr$ in $\mathcal{R}/q$.

To finish we trace through $\mathsf{Decrypt}(c, k)$. All coefficients of the polynomial $3fm + gr$ in $\mathcal{R}$ are in $[-8w, 8w]$ by Theorem 3, and thus in $[-(q-1)/2, (q-1)/2]$ since $q \geq 16w + 1$. Viewing each coefficient of $3fc$ in $\mathcal{R}/q$ as an integer in $[-(q-1)/2, (q-1)/2]$ thus produces exactly $3fm + gr \in \mathcal{R}$, and reducing modulo 3 produces $gr \in \mathcal{R}/3$; i.e., $e = gr$ in $\mathcal{R}/3$, so $ev = e/g = r$ in $\mathcal{R}/3$. Lifting now produces exactly r since r is small; i.e., $r' = r$. Also r has weight w, so r' has weight w, and $\mathsf{Decrypt}(c, k)$ outputs r. □

2.3.5 Streamlined NTRU Prime parameter space

Streamlined NTRU Prime has the Streamlined NTRU Prime Core parameters, plus the following parameters:

- sets $\mathsf{SessionKeys}, \mathsf{Confirm}, \underline{\mathsf{PublicKeys}}, \underline{\mathsf{SecretKeys}}, \underline{\mathsf{Inputs}}, \underline{\mathsf{Ciphertexts}}$ of strings, each set being the set of all strings of a specified length;
- deterministic encoding algorithms $\mathsf{PublicKeys} \to \underline{\mathsf{PublicKeys}}$, $\mathsf{SecretKeys} \to \underline{\mathsf{SecretKeys}}$, $\mathsf{Inputs} \to \underline{\mathsf{Inputs}}$, and $\mathsf{Ciphertexts} \to \underline{\mathsf{Ciphertexts}}$;
- deterministic decoding algorithms $\underline{\mathsf{PublicKeys}} \to \mathsf{PublicKeys}$, $\underline{\mathsf{SecretKeys}} \to \mathsf{SecretKeys}$, $\underline{\mathsf{Inputs}} \to \mathsf{Inputs}$, and $\underline{\mathsf{Ciphertexts}} \to \mathsf{Ciphertexts}$ that always invert encoding;

- a deterministic algorithm $\mathsf{HashConfirm} : \underline{\mathsf{Inputs}} \times \underline{\mathsf{PublicKeys}} \to \mathsf{Confirm}$; and
- a deterministic algorithm $\mathsf{HashSession} : \{0,1\} \times \underline{\mathsf{Inputs}} \times \underline{\mathsf{Ciphertexts}} \times \mathsf{Confirm} \to \mathsf{SessionKeys}$.

2.3.6 Streamlined NTRU Prime key generation

The following randomized algorithm KeyGen' outputs an element of $\underline{\mathsf{PublicKeys}} \times \mathsf{SecretKeys}'$, where $\mathsf{SecretKeys}' = \underline{\mathsf{SecretKeys}} \times \underline{\mathsf{PublicKeys}} \times \underline{\mathsf{Inputs}}$:

- Compute $(K, k) \leftarrow \mathsf{KeyGen}()$.
- Encode K as a string $\underline{K} \in \underline{\mathsf{PublicKeys}}$.
- Encode k as a string $\underline{k} \in \underline{\mathsf{SecretKeys}}$.
- Generate a uniform random $\rho \in \underline{\mathsf{Inputs}}$.
- Output $(\underline{K}, (\underline{k}, \underline{K}, \rho))$.

Modification after round 1: The secret key now contains a ρ component used for implicit rejection.

2.3.7 Streamlined NTRU Prime encapsulation

The following randomized algorithm Encap, given an element of $\underline{\mathsf{PublicKeys}}$, outputs an element of $\mathsf{Ciphertexts}' \times \mathsf{SessionKeys}$, where $\mathsf{Ciphertexts}' = \underline{\mathsf{Ciphertexts}} \times \mathsf{Confirm}$:

- Input $\underline{K} \in \underline{\mathsf{PublicKeys}}$. Decode $\underline{K}$, obtaining $K \in \mathsf{PublicKeys}$.
- Generate a uniform random $r \in \mathsf{Inputs}$. Encode r as a string $\underline{r} \in \underline{\mathsf{Inputs}}$.
- Compute $c = \mathsf{Encrypt}(r, K) \in \mathsf{Ciphertexts}$. Encode c as a string $\underline{c} \in \underline{\mathsf{Ciphertexts}}$.
- Compute $C = (\underline{c}, \mathsf{HashConfirm}(\underline{r}, \underline{K})) \in \underline{\mathsf{Ciphertexts}} \times \mathsf{Confirm}$.
- Output $(C, \mathsf{HashSession}(1, \underline{r}, C))$.

Modification after round 1: $\mathsf{HashConfirm}$ is now given access to the public key $\underline{K}$ as an extra input, and $\mathsf{HashSession}$ is now given access to the ciphertext C as an extra input.

2.3.8 Streamlined NTRU Prime decapsulation

The following deterministic algorithm Decap, given an element of $\mathsf{Ciphertexts}' \times \mathsf{SecretKeys}'$, outputs an element of $\mathsf{SessionKeys}$:

- Input $C = (\underline{c}, \gamma) \in \underline{\mathsf{Ciphertexts}} \times \mathsf{Confirm}$ and $(\underline{k}, \underline{K}, \rho) \in \underline{\mathsf{SecretKeys}} \times \underline{\mathsf{PublicKeys}} \times \underline{\mathsf{Inputs}}$.
- Decode $\underline{c}$, obtaining $c \in \mathsf{Ciphertexts}$.
- Decode $\underline{k}$, obtaining $k \in \mathsf{SecretKeys}$.

- Compute $r' = \mathsf{Decrypt}(c, k) \in \mathsf{Inputs}$.
- Compute $\underline{r'}, c', \underline{c'}, C'$ as in Encap.
- If $C' = C$ then output $\mathsf{HashSession}(1, \underline{r}, C)$. Otherwise output $\mathsf{HashSession}(0, \rho, C)$. (The choice between these two outputs is secret information.)

Modification after round 1: Rejection is now implicit instead of explicit.

2.4 NTRU LPRime

NTRU LPRime has three layers. The inner layer is NTRU LPRime Core, a perfectly correct randomized PKE. The middle layer is NTRU LPRime Expand, a perfectly correct deterministic PKE. The outer layer is NTRU LPRime, a perfectly correct KEM.

Applications are expected to use solely the outer layer. We present the three layers separately to assist reviewers and implementors.

2.4.1 NTRU LPRime Core parameter space

NTRU LPRime Core has parameters (p, q, w, δ, I) subject to the following restrictions: p is a prime number; q is a prime number; w, δ, I are positive integers; $2p \geq 3w$; I is a multiple of 8; $p \geq I$; $q \geq 16w + 2\delta + 3$; $x^p - x - 1$ is irreducible in the polynomial ring $\mathcal{R}/q$.

NTRU LPRime Core also has the following parameters: a positive integer τ; a deterministic algorithm $\mathsf{Top} : \mathbb{Z}/q \to \mathbb{Z}/\tau$; and a deterministic algorithm $\mathsf{Right} : \mathbb{Z}/\tau \to \mathbb{Z}/q$ such that the difference $\mathsf{Right}(\mathsf{Top}(C)) - C \in \mathbb{Z}/q$ is in $\{0, 1, \ldots, \delta\}$ for each $C \in \mathbb{Z}/q$.

2.4.2 NTRU LPRime Core key generation

Define $\mathsf{PublicKeys} = \mathcal{R}/q \times \mathsf{Rounded}$ and $\mathsf{SecretKeys} = \mathsf{Short}$. The following randomized algorithm KeyGen outputs an element of $\mathsf{PublicKeys} \times \mathsf{SecretKeys}$:

- Generate a uniform random $G \in \mathcal{R}/q$. (NTRU LPRime Expand will replace the randomness used in this step with output of a specific RNG.)
- Generate a uniform random $a \in \mathsf{Short}$.
- Compute $aG \in \mathcal{R}/q$.
- Output $((G, \mathsf{Round}(aG)), a) \in \mathsf{PublicKeys} \times \mathsf{SecretKeys}$.

2.4.3 NTRU LPRime Core encryption

Define $\mathsf{Inputs} = \{0,1\}^I$ and $\mathsf{Ciphertexts} = \mathsf{Rounded} \times (\mathbb{Z}/\tau)^I$. The following randomized algorithm $\mathsf{Encrypt}$ maps $\mathsf{Inputs} \times \mathsf{PublicKeys}$ to $\mathsf{Ciphertexts}$:

- Input $r = (r_0, r_1, \dots, r_{I-1}) \in \mathsf{Inputs}$ and $(G, A) \in \mathsf{PublicKeys}$.
- Generate a uniform random $b \in \mathsf{Short}$. (NTRU LPRime Expand will replace the randomness used in this step with output of a specific RNG.)
- Compute bG in $\mathcal{R}/q$.
- Compute bA in $\mathcal{R}/q$. (Only the bottom I coefficients of bA, the coefficients $(bA)_0, (bA)_1, \dots, (bA)_{I-1}$ of $x^0, x^1, \dots, x^{I-1}$ respectively, will be used; other coefficients do not need to be computed.)
- Compute $T = (T_0, T_1, \dots, T_{I-1}) \in (\mathbb{Z}/\tau)^I$ as follows: $T_j = \mathsf{Top}((bA)_j + r_j(q-1)/2)$.
- Output $(\mathsf{Round}(bG), T) \in \mathsf{Ciphertexts}$.

2.4.4 NTRU LPRime Core decryption

The following deterministic algorithm $\mathsf{Decrypt}$ maps $\mathsf{Ciphertexts} \times \mathsf{SecretKeys}$ to Inputs:

- Input $(B, T) \in \mathsf{Rounded} \times (\mathbb{Z}/\tau)^I$ and $a \in \mathsf{SecretKeys}$.
- Compute aB in $\mathcal{R}/q$. (Only the bottom I coefficients of aB will be used.)
- Compute $r'_0, r'_1, \dots, r'_{I-1} \in \{0, 1\}$ as follows. View $\mathsf{Right}(T_j) - (aB)_j + 4w + 1 \in \mathbb{Z}/q$ as an integer between $-(q-1)/2$ and $(q-1)/2$. Then r'_j is the sign bit of this integer: 1 if the integer is negative, otherwise 0.
- Output $(r'_0, r'_1, \dots, r'_{I-1}) \in \mathsf{Inputs}$.

Theorem 5. *Assume that* $\mathsf{KeyGen}()$ *outputs* $(P, a) \in \mathsf{PublicKeys} \times \mathsf{SecretKeys}$. *Fix* $r \in \mathsf{Inputs}$. *Assume that* $\mathsf{Encrypt}(r, P)$ *outputs* C. *Then* $\mathsf{Decrypt}(C, a) = r$.

Proof. By definition of KeyGen, P has the form $(G, \mathsf{Round}(aG))$ for some $G \in \mathcal{R}/q$. Write $A = \mathsf{Round}(aG)$; then $A = aG + d$ in $\mathcal{R}/q$ for some small $d \in \mathcal{R}$.

Write r as $(r_0, r_1, \dots, r_{I-1})$. By definition of $\mathsf{Encrypt}$, C has the form $(\mathsf{Round}(bG), T)$ for some $b \in \mathsf{Short}$, where $T_j = \mathsf{Top}((bA)_j + r_j(q-1)/2)$. Write $B = \mathsf{Round}(bG)$; then $B = bG + e$ in $\mathcal{R}/q$ for some small $e \in \mathcal{R}$.

By definition of Right, the difference $\mathsf{Right}(\mathsf{Top}(C)) - C$ is in $\{0, 1, \dots, \delta\}$ for each $C \in \mathbb{Z}/q$. In particular, $\mathsf{Right}(\mathsf{Top}(C_j)) - C_j \in \{0, 1, \dots, \delta\}$ where $C_j = (bA)_j + r_j(q-1)/2$. Hence

$$\begin{aligned}
&\mathsf{Right}(T_j) - (aB)_j + 4w + 1 \\
&\quad = \mathsf{Right}(\mathsf{Top}(C_j)) - (a(bG + e))_j + 4w + 1 \\
&\quad = \mathsf{Right}(\mathsf{Top}(C_j)) - C_j + C_j - ((abG)_j + (ae)_j) + 4w + 1 \\
&\quad = \mathsf{Right}(\mathsf{Top}(C_j)) - C_j + (bA)_j + r_j(q-1)/2 - ((abG)_j + (ae)_j) + 4w + 1 \\
&\quad = \mathsf{Right}(\mathsf{Top}(C_j)) - C_j + (baG)_j + (bd)_j + r_j(q-1)/2 - ((abG)_j + (ae)_j) + 4w + 1 \\
&\quad = \mathsf{Right}(\mathsf{Top}(C_j)) - C_j + (bd)_j - (ae)_j + 4w + 1 + r_j(q-1)/2 \in \mathbb{Z}/q.
\end{aligned}$$

All coefficients of the polynomials bd and ae are in $[-2w, 2w]$ by Theorem 2, since d, e are small and $a, b \in \mathsf{Short}$; so $(bd)_j - (ae)_j + 4w + 1 \in \{1, \dots, 8w + 1\}$. Also $\mathsf{Right}(\mathsf{Top}(C_j)) - C_j \in$

$\{0, 1, \ldots, \delta\}$. Hence $\mathsf{Right}(T_j) - (aB)_j + 4w + 1 \in r_j(q-1)/2 + \{1, \ldots, 8w + \delta + 1\}$. Viewing each coefficient of $\mathsf{Right}(T_j) - (aB)_j + 4w + 1$ as an integer in $[-(q-1)/2, (q-1)/2]$ thus produces

- an integer in $[1, 8w + \delta + 1]$ if and only if $r_j = 0$ and
- an integer in $[-(q-1)/2, -(q-1)/2 + 8w + \delta]$ if and only if $r_j = 1$

because $8w + \delta + 1 \leq (q-1)/2$ by construction. Hence $r'_j = r_j$ inside $\mathsf{Decrypt}(C, a)$, so $\mathsf{Decrypt}(C, a) = r$. □

2.4.5 NTRU LPRime Expand parameter space

NTRU LPRime Expand has the NTRU LPRime Core parameters, plus the following parameters: a finite nonempty set Seeds; a deterministic algorithm $\mathsf{Generator} : \mathsf{Seeds} \to \mathcal{R}/q$; and a deterministic algorithm $\mathsf{HashShort} : \mathsf{Inputs} \to \mathsf{Short}$.

2.4.6 NTRU LPRime Expand key generation

Define $\mathsf{PublicKeys}' = \mathsf{Seeds} \times \mathsf{Rounded}$. The following randomized algorithm KeyGen' outputs an element of $\mathsf{PublicKeys}' \times \mathsf{SecretKeys}$:

- Generate a uniform random $S \in \mathsf{Seeds}$.
- Replace the uniform random $G \in \mathcal{R}/q$ with $G = \mathsf{Generator}(S)$ inside $\mathsf{KeyGen}()$, obtaining $((G, \mathsf{Round}(aG)), a) \in \mathsf{PublicKeys} \times \mathsf{SecretKeys}$.
- Output $((S, \mathsf{Round}(aG)), a) \in \mathsf{PublicKeys}' \times \mathsf{SecretKeys}$.

2.4.7 NTRU LPRime Expand encryption

The following deterministic algorithm $\mathsf{Encrypt}'$ maps $\mathsf{Inputs} \times \mathsf{PublicKeys}'$ to $\mathsf{Ciphertexts}$:

- Input $r \in \mathsf{Inputs}$ and $(S, A) \in \mathsf{Seeds} \times \mathsf{Rounded}$.
- Compute $G = \mathsf{Generator}(S)$.
- Replace the uniform random $b \in \mathsf{Short}$ with $\mathsf{HashShort}(r)$ inside $\mathsf{Encrypt}(r, (G, A))$, obtaining $c \in \mathsf{Ciphertexts}$.
- Output c.

2.4.8 NTRU LPRime Expand decryption

NTRU LPRime Expand decryption is identical to NTRU LPRime Core decryption.

2.4.9 NTRU LPRime parameter space

NTRU LPRime has the NTRU LPRime Expand parameters, plus the following parameters:

- sets $\mathsf{SessionKeys}, \mathsf{Confirm}, \underline{\mathsf{PublicKeys'}}, \underline{\mathsf{SecretKeys}}, \underline{\mathsf{Inputs}}, \underline{\mathsf{Ciphertexts}}$ of strings, each set being the set of all strings of a specified length;
- deterministic encoding algorithms $\mathsf{PublicKeys'} \to \underline{\mathsf{PublicKeys'}}$, $\mathsf{SecretKeys} \to \underline{\mathsf{SecretKeys}}$, $\mathsf{Inputs} \to \underline{\mathsf{Inputs}}$, and $\mathsf{Ciphertexts} \to \underline{\mathsf{Ciphertexts}}$;
- deterministic decoding algorithms $\underline{\mathsf{PublicKeys'}} \to \mathsf{PublicKeys'}$, $\underline{\mathsf{SecretKeys}} \to \mathsf{SecretKeys}$, $\underline{\mathsf{Inputs}} \to \mathsf{Inputs}$, and $\underline{\mathsf{Ciphertexts}} \to \mathsf{Ciphertexts}$ that always invert encoding;
- a deterministic algorithm $\mathsf{HashConfirm} : \underline{\mathsf{Inputs}} \times \underline{\mathsf{PublicKeys'}} \to \mathsf{Confirm}$; and
- a deterministic algorithm $\mathsf{HashSession} : \{0,1\} \times \underline{\mathsf{Inputs}} \times \underline{\mathsf{Ciphertexts}} \times \mathsf{Confirm} \to \mathsf{SessionKeys}$.

2.4.10 NTRU LPRime key generation

The following randomized algorithm $\mathsf{KeyGen''}$ outputs an element of $\underline{\mathsf{PublicKeys'}} \times \mathsf{SecretKeys'}$, where $\mathsf{SecretKeys'} = \underline{\mathsf{SecretKeys}} \times \underline{\mathsf{PublicKeys'}} \times \underline{\mathsf{Inputs}}$:

- Compute $(K, k) \leftarrow \mathsf{KeyGen'}()$.
- Encode K as a string $\underline{K} \in \underline{\mathsf{PublicKeys'}}$.
- Encode k as a string $\underline{k} \in \underline{\mathsf{SecretKeys}}$.
- Generate a uniform random $\rho \in \underline{\mathsf{Inputs}}$.
- Output $(\underline{K}, (\underline{k}, \underline{K}, \rho))$.

Modification after round 1: The secret key now contains the ρ component, which is used for implicit rejection.

2.4.11 NTRU LPRime encapsulation

The following randomized algorithm Encap, given an element of $\underline{\mathsf{PublicKeys'}}$, outputs an element of $\mathsf{Ciphertexts'} \times \mathsf{SessionKeys}$, where $\mathsf{Ciphertexts'} = \underline{\mathsf{Ciphertexts}} \times \mathsf{Confirm}$:

- Input $\underline{K} \in \underline{\mathsf{PublicKeys'}}$.
- Decode $\underline{K}$, obtaining $K \in \mathsf{PublicKeys'}$.
- Generate a uniform random $r \in \mathsf{Inputs}$.
- Encode r as a string $\underline{r} \in \underline{\mathsf{Inputs}}$.
- Compute $c = \mathsf{Encrypt'}(r, K) \in \mathsf{Ciphertexts}$.
- Encode c as a string $\underline{c} \in \underline{\mathsf{Ciphertexts}}$.

- Compute $C = (\underline{c}, \mathsf{HashConfirm}(\underline{r}, \underline{K})) \in \underline{\mathsf{Ciphertexts}} \times \mathsf{Confirm}$.
- Output $(C, \mathsf{HashSession}(1, \underline{r}, C))$.

Modification after round 1: HashConfirm is now given access to the public key $\underline{K}$ as an extra input, and HashSession is now given access to the ciphertext C as an extra input.

2.4.12 NTRU LPRime decapsulation

The following deterministic algorithm Decap, given an element of $\mathsf{Ciphertexts}' \times \mathsf{SecretKeys}'$, outputs an element of SessionKeys:

- Input $C = (\underline{c}, \gamma) \in \underline{\mathsf{Ciphertexts}} \times \mathsf{Confirm}$ and $(\underline{k}, \underline{K}, \rho) \in \underline{\mathsf{SecretKeys}} \times \underline{\mathsf{PublicKeys}'} \times \underline{\mathsf{Inputs}}$.
- Decode $\underline{c}$, obtaining $c \in \mathsf{Ciphertexts}$.
- Decode $\underline{k}$, obtaining $k \in \mathsf{SecretKeys}$.
- Compute $r' = \mathsf{Decrypt}(c, k) \in \mathsf{Inputs}$.
- Compute $\underline{r'}, c', \underline{c'}, C'$ as in Encap.
- If $C' = C$ then output $\mathsf{HashSession}(1, \underline{r}, C)$. Otherwise output $\mathsf{HashSession}(0, \rho, C)$. (The choice between these two outputs is secret information.)

Modification after round 1: Rejection is now implicit instead of explicit.

3 List of parameter sets (part of 2.B.1)

Options expanded for round 2: There are now three sizes of parameter sets. Specifically:

- $(p, q) = (761, 4591)$, with $w = 286$ for Streamlined NTRU Prime Core and $w = 250$ for NTRU LPRime Core (with the functions Top and Right defined below). These are exactly the trapdoor functions from the round-1 submission.
- New smaller size: $(p, q) = (653, 4621)$, with $w = 288$ for Streamlined NTRU Prime Core and $w = 252$ for NTRU LPRime Core.
- New larger size: $(p, q) = (857, 5167)$, with $w = 322$ for Streamlined NTRU Prime Core and $w = 281$ for NTRU LPRime Core.

Each size has one Streamlined NTRU Prime parameter set and one NTRU LPRime parameter set, for a total of six parameter sets.

Encodings and hash details modified for round 2: The input to the confirmation hash now includes (a hash of) the public key. The input to the session-key hash now includes the ciphertext. Other hash details are modified. The encodings as strings are now more space-efficient, and are defined in a unified way across sizes.

3.1 Shared choices of parameters

The following definitions and parameter choices are shared by all six parameter sets.

Set of session keys. $\mathsf{SessionKeys}$ is the set of 32-byte strings.

Set of confirmation strings. $\mathsf{Confirm}$ is the set of 32-byte strings.

Hashing. Define $\mathsf{Hash}(z)$ as the first 32 bytes of SHA-512(z). Define Hash_b for $b \in \{0, 1, \ldots, 255\}$ as Hash with the input prefixed by byte b: i.e., $\mathsf{Hash}_b(z) = \mathsf{Hash}(b, z)$.

General-purpose encoding of sequences of integers. We define deterministic algorithms Encode and Decode with the following properties.

Let $M = (m_0, \ldots, m_{n-1})$ and $R = (r_0, \ldots, r_{n-1})$ be sequences of integers. Assume that $0 \le r_i < m_i < 2^{14}$ for each i. Then $S = \mathsf{Encode}(R, M)$ is a sequence of bytes, and $\mathsf{Decode}(\mathsf{Encode}(R, M), M) = R$. The length of S depends only on M, not on R. If S' is any sequence of bytes of this length, then $\mathsf{Decode}(S', M)$ is a sequence of n integers, although not necessarily in the same range as R.

The definitions of Encode and Decode are formally expressed in the Python language as Figures 1 and 2 respectively. Encoding works as follows:

- If $n = 0$ then S is the empty sequence ().
- If $n = 1$ then S is r_0 in little-endian form, with 2 bytes if $m_0 > 2^8$, or 1 byte if $2^8 \ge m_0 > 1$, or 0 bytes if $m_0 = 1$.
- If $n \ge 2$ then S is a prefix followed by $\mathsf{Encode}(R', M')$, where R', M' have length $\lceil n/2 \rceil$. Specifically, each pair r_i, r_{i+1} modulo m_i, m_{i+1} for even i is merged into $r = r_i + m_i r_{i+1}$ modulo $m = m_i m_{i+1}$, and then r, m are reduced to r', m' with $0 < r' < m' < 2^{14}$, producing an entry for R' and an entry for M' respectively. Reduction means that if $m \ge 2^{14}$ then $r \bmod 2^8$ is appended to the prefix while r, m are replaced by $\lfloor r/2^8 \rfloor, \lceil m/2^8 \rceil$ respectively; this is repeated 0, 1, or 2 times to reduce m to the correct range. If n is odd then the final r_{n-1}, m_{n-1} are also included in R', M'.

We use these algorithms only for sequences M that have at most two values: more precisely, $m_0 = \cdots = m_{n-2}$. Each parameter set involves, in total, only a logarithmic number of different values. (The computations can be optimized accordingly. The recursion can be eliminated in favor of iteration.)

Encoding of field elements. View each element of $\mathcal{R}/q$ as a polynomial $r_0 + r_1 x + \cdots + r_{p-1} x^{p-1}$ with each $r_i \in \{-(q-1)/2, \ldots, -1, 0, 1, \ldots, (q-1)/2\}$. Add $(q-1)/2$ to each coefficient to obtain a sequence of p elements of $\{0, 1, \ldots, q-1\}$. Apply Encode with $M = (q, \ldots, q)$ to obtain a string.

Encoding of rounded field elements. All of our parameters have $q \equiv 1 \pmod 6$, so $\mathsf{Rounded}$ is the set of polynomials $r_0 + r_1 x + \cdots + r_{p-1} x^{p-1}$ with each $r_i \in \{-(q-1)/2, \ldots, -3, 0, 3, \ldots, (q-1)/2\}$. Add $(q-1)/2$ to each coefficient and divide by 3 to obtain a sequence of p elements of $\{0, 1, \ldots, (q-1)/3\}$. Apply Encode with

```
limit = 16384

def Encode(R,M):
  if len(M) == 0: return []
  S = []
  if len(M) == 1:
    r,m = R[0],M[0]
    while m > 1:
      S += [r%256]
      r,m = r//256,(m+255)//256
    return S
  R2,M2 = [],[]
  for i in range(0,len(M)-1,2):
    m,r = M[i]*M[i+1],R[i]+M[i]*R[i+1]
    while m >= limit:
      S += [r%256]
      r,m = r//256,(m+255)//256
    R2 += [r]
    M2 += [m]
  if len(M)&1:
    R2 += [R[-1]]
    M2 += [M[-1]]
  return S+Encode(R2,M2)
```

Figure 1: General-purpose encoding of sequences of integers. The lists R and M have the same length, and $0 \leq R[i] < M[i] < 2^{14}$ for each i.

$M = ((q-1)/3+1, \dots, (q-1)/3+1)$ to obtain a string.

Encoding of small ring elements. A small element $r = r_0 + r_1 x + \cdots + r_{p-1} x^{p-1}$ of $\mathcal{R}$ is encoded as a $\lceil p/4 \rceil$-byte string $\underline{r}$ as follows. View the polynomial in little-endian form as a sequence of coefficients of $x^0, x^1, \dots, x^{p-1}$. Add 1 to each coefficient, obtaining a sequence of p elements of $\{0,1,2\}$. Write each batch of 4 elements in little-endian form in radix 4, obtaining a byte.

3.2 Shared choices of Streamlined NTRU Prime parameters

The following definitions and parameter choices are shared by the three Streamlined NTRU Prime parameter sets.

Encoding of public keys. The encoding of public keys is the encoding of field elements.

```
limit = 16384

def Decode(S,M):
  if len(M) == 0: return []
  if len(M) == 1: return [sum(S[i]*256**i for i in range(len(S)))%M[0]]
  k = 0
  bottom,M2 = [],[]
  for i in range(0,len(M)-1,2):
    m,r,t = M[i]*M[i+1],0,1
    while m >= limit:
      r,t,k,m = r+S[k]*t,t*256,k+1,(m+255)//256
    bottom += [(r,t)]
    M2 += [m]
  if len(M)&1:
    M2 += [M[-1]]
  R2 = Decode(S[k:],M2)
  R = []
  for i in range(0,len(M)-1,2):
    r,t = bottom[i//2]
    r += t*R2[i//2]
    R += [r%M[i]]
    R += [(r//M[i])%M[i+1]]
  if len(M)&1:
    R += [R2[-1]]
  return R
```

Figure 2: General-purpose decoding of sequences of integers. If the lists R and M have the same length, and $0 \le R[i] < M[i] < 2^{14}$ for each i, then $\mathsf{Decode}(\mathsf{Encode}(R, M), M) = R$.

Encoding of secret keys. The f component of a secret key is encoded as a small ring element. The $v = 1/g$ component of a secret key is encoded as a small ring element.

Encoding of inputs. $\underline{\mathsf{Inputs}}$ is the set of $\lceil p/4 \rceil$-byte strings. The encoding is the encoding of small field elements. (One can, optionally, compress ρ by replacing it with an RNG seed, at the expense of running the RNG to regenerate ρ. Another possibility that removes this expense is to use a short ρ to overwrite $\mathsf{Hash}_3(\underline{r})$ in the definition of $\mathsf{HashSession}$ below.)

Encoding of ciphertexts. The encoding of ciphertexts is the encoding of rounded field elements.

Confirmation hash. $\mathsf{HashConfirm}(\underline{r}, \underline{K})$ is defined as $\mathsf{Hash}_2(\mathsf{Hash}_3(\underline{r}), \mathsf{Hash}_4(\underline{K}))$ for each $\underline{r} \in \underline{\mathsf{Inputs}}$ and each $\underline{K} \in \underline{\mathsf{PublicKeys}}$.

Key-hash caching. Caching $\mathsf{Hash}_4(\underline{K})$ saves time when K is reused in encapsulation or decapsulation. We cache $\mathsf{Hash}_4(\underline{K})$ at the end of secret keys.

Session-key hash. $\mathsf{HashSession}(b, y, z)$ is defined as $\mathsf{Hash}_b(\mathsf{Hash}_3(y), z)$ for each $b \in \{0, 1\}$, each $y \in \underline{\mathsf{Inputs}}$, and each $z \in \underline{\mathsf{Ciphertexts}} \times \mathsf{Confirm}$.

3.3 Shared choices of NTRU LPRime parameters

The following definitions and parameter choices are shared by the three NTRU LPRime parameter sets.

Top bits. The parameter τ is defined as 16.

There are two further parameters τ_1, τ_0 that define the Top function. For each $C \in \{-(q-1)/2, \ldots, (q-1)/2\}$, $\mathsf{Top}(C)$ is defined as $\lfloor(\tau_1(C + \tau_0) + 2^{14})/2^{15}\rfloor$. The parameters τ_1, τ_0 are chosen so that $\mathsf{Top}(C) \in \{0, 1, \ldots, 15\}$. This function from $\{-(q-1)/2, \ldots, (q-1)/2\}$ to $\{0, 1, \ldots, 15\}$ induces a function $\mathsf{Top} : \mathbb{Z}/q \to \mathbb{Z}/16$.

There are also two parameters τ_3, τ_2 that define the Right function. For each $T \in \{0, 1, \ldots, 15\}$, $\mathsf{Right}(T)$ is defined as $\tau_3 T - \tau_2 \in \mathbb{Z}/q$. The parameters τ_3, τ_2 are chosen so that each $C \in \{-(q-1)/2, \ldots, (q-1)/2\}$ satisfies $\tau_3 T - \tau_2 - C \in \{0, 1, \ldots, \delta\}$ where $T = \lfloor(\tau_1(C + \tau_0) + 2^{14})/2^{15}\rfloor$; one can efficiently verify this property by checking each C. This function from $\{0, 1, \ldots, 15\}$ to $\mathbb{Z}/q$ induces a function $\mathsf{Right} : \mathbb{Z}/16 \to \mathbb{Z}/q$.

Expansion. Define $\mathsf{Expand}(K)$, where K is a 32-byte string, as the first $4p$ bytes of AES-256-CTR output using key K, starting with counter 0.

Set of seeds. Seeds is the set of 32-byte strings.

Mapping seeds to ring elements. For each $K \in \mathsf{Seeds}$, $\mathsf{Generator}(K) \in \mathcal{R}/q$ is defined as follows:

- Compute the $4p$-byte string $\mathsf{Expand}(K)$.
- View each 4 bytes of the string in little-endian form, obtaining p elements of $\{0, 1, \ldots, 2^{32} - 1\}$.
- Reduce each of these elements modulo q, obtaining p elements of $\{0, 1, \ldots, q - 1\}$.
- Obtain p elements of $\{-(q-1)/2, \ldots, (q-1)/2\}$ by subtracting $(q-1)/2$ from each integer.
- View these elements as a polynomial in little-endian form, namely $\mathsf{Generator}(K)$.

Number of input bits. I is 256, so $\mathsf{Inputs} = \{0, 1\}^{256}$.

Encoding of inputs. $\underline{\mathsf{Inputs}}$ is the set of 32-byte strings. An element $r = (r_0, r_1, \ldots, r_{255}) \in \mathsf{Inputs}$ is encoded as a string $\underline{r} \in \underline{\mathsf{Inputs}}$ in little-endian form: i.e., the first byte of $\underline{r}$ is $r_0 + 2r_1 + \cdots + 128r_7$, the next byte is $r_8 + 2r_9 + \cdots + 128r_{15}$, etc.

Hashing inputs to short polynomials. For each $r \in \mathsf{Inputs}$, $\mathsf{HashShort}(r) \in \mathsf{Short}$ is defined as follows:

- Compute $K = \mathsf{Hash}_5(\underline{r})$. Here $\underline{r} \in \underline{\mathsf{Inputs}}$ is the encoding of r defined above.

- Compute the $4p$-byte string $\mathsf{Expand}(K)$.
- View each 4 bytes of the string in little-endian form, obtaining p elements of $\{0, 1, \ldots, 2^{32} - 1\}$.
- Clear the bottom bit of each of the first w integers; now each of those integers is 0 modulo 2.
- Set the bottom bit, and clear the next bit, of each of the remaining $p - w$ integers; now each of those integers is 1 modulo 4.
- Sort the integers.
- Reduce each integer modulo 4, and subtract 1, obtaining p elements of $\{-1, 0, 1\}$, of which exactly w are nonzero.
- View these elements as a polynomial in little-endian form, namely $\mathsf{HashShort}(K)$.

Encoding of public keys. An element of $\mathsf{PublicKeys}'$ has two components, $S \in \mathsf{Seeds}$ and $A \in \mathsf{Rounded}$. The S component is a 32-byte string by definition. The A component is encoded as a rounded field element.

Encoding of secret keys. The encoding of secret keys is the encoding of small ring elements.

Encoding of ciphertexts. An element of $\mathsf{Ciphertexts}$ has two components, $B \in \mathsf{Rounded}$ and $T \in (\mathbb{Z}/\tau)^{256}$. The B component is encoded as a rounded field element. The T component is encoded in radix 16 in little-endian form as 128 bytes, with each element of $\mathbb{Z}/\tau$ viewed as an element of $\{0, 1, \ldots, 15\}$: i.e., $(T_0, T_1, \ldots, T_{255}) \in \{0, 1, \ldots, 15\}^{256}$ is encoded as the bytes $(T_0 + 16T_1, T_2 + 16T_3, \ldots, T_{254} + 16T_{255})$.

Confirmation hash. $\mathsf{HashConfirm}(\underline{r}, \underline{K})$ is defined as $\mathsf{Hash}_2(\underline{r}, \mathsf{Hash}_4(\underline{K}))$ for each $\underline{r} \in \underline{\mathsf{Inputs}}$ and each $\underline{K} \in \underline{\mathsf{PublicKeys}'}$.

Key-hash caching. Caching $\mathsf{Hash}_4(\underline{K})$ saves time when K is reused in encapsulation or decapsulation. We cache $\mathsf{Hash}_4(\underline{K})$ at the end of secret keys.

Session-key hash. $\mathsf{HashSession}(b, z)$ is defined as $\mathsf{Hash}_b(z)$ for each $b \in \{0, 1\}$ and each $z \in \underline{\mathsf{Inputs}} \times \underline{\mathsf{Ciphertexts}} \times \mathsf{Confirm}$.

3.4 Parameter set `kem/sntrup653`

Streamlined NTRU Prime with $p = 653$, $q = 4621$, and $w = 288$.

3.5 Parameter set `kem/sntrup761`

Streamlined NTRU Prime with $p = 761$, $q = 4591$, and $w = 286$.

3.6 Parameter set `kem/sntrup857`

Streamlined NTRU Prime with $p = 857$, $q = 5167$, and $w = 322$.

3.7 Parameter set `kem/ntrulpr653`

NTRU LPRime with $p = 653$, $q = 4621$, $w = 252$, $\delta = 289$, $\tau_0 = 2175$, $\tau_1 = 113$, $\tau_2 = 2031$, $\tau_3 = 290$.

3.8 Parameter set `kem/ntrulpr761`

NTRU LPRime with $p = 761$, $q = 4591$, $w = 250$, $\delta = 292$, $\tau_0 = 2156$, $\tau_1 = 114$, $\tau_2 = 2007$, $\tau_3 = 287$.

3.9 Parameter set `kem/ntrulpr857`

NTRU LPRime with $p = 857$, $q = 5167$, $w = 281$, $\delta = 329$, $\tau_0 = 2433$, $\tau_1 = 101$, $\tau_2 = 2265$, $\tau_3 = 324$.

4 Design rationale (part of 2.B.1)

There are many different ideal-lattice-based public-key encryption schemes in the literature, including many versions of NTRU; many Ring-LWE-based cryptosystems; and now Streamlined NTRU Prime and NTRU LPRime. These are actually many different points in a high-dimensional space of possible cryptosystems. We give a unified description of the advantages and disadvantages of what we see as the most important options in each dimension, in particular explaining the choices that we made in Streamlined NTRU Prime and NTRU LPRime. Beware that there are many interactions between options. For example, using Gaussian errors is incompatible with eliminating decryption failures, because there is always a small probability of large samples combining with large values. Using *truncated* Gaussian errors is compatible with eliminating decryption failures, but requires a much larger modulus q. Neither of these options is compatible with the simple tight KEM that we use.

4.1 The ring

The choice of cryptosystem includes a choice of a monic degree-p polynomial $P \in \mathbb{Z}[x]$ and a choice of a positive integer q. As in Section 2, we abbreviate the ring $\mathbb{Z}[x]/P$ as $\mathcal{R}$, and the ring $(\mathbb{Z}/q)[x]/P$ as $\mathcal{R}/q$.

Common choices of $\mathcal{R}/q$ are as follows:

- "NTRU Classic": Rings of the form $(\mathbb{Z}/q)[x]/(x^p - 1)$, where p is a prime and q is a power of 2, are used in the original NTRU cryptosystem [54].
- "NTRU NTT": Rings of the form $(\mathbb{Z}/q)[x]/(x^p + 1)$, where p is a power of 2 and $q \in 1 + 2p\mathbb{Z}$ is a prime, are used in typical "Ring-LWE-based" cryptosystems such as [7].
- "NTRU Prime": Fields of the form $(\mathbb{Z}/q)[x]/(x^p - x - 1)$, where p is prime, are used in this submission.

NTRU Prime uses a prime-degree number field with a large Galois group and an inert modulus, minimizing the number of ring homomorphisms available to the attacker. As an analogy, conservative prime-field discrete-logarithm systems also minimize the number of ring homomorphisms available to the attacker.

We expect the future situation, like the current situation, to be a mix of the following three scenarios:

- Some lattice-based systems are broken whether or not they have unnecessary homomorphisms. As an analogy, some discrete-logarithm systems are broken whether or not they have unnecessary homomorphisms.
- Some lattice-based systems are unbroken whether or not they have unnecessary homomorphisms. As an analogy, some discrete-logarithm systems are unbroken whether or not they have unnecessary homomorphisms.
- Some lattice-based systems are broken *only if* they have unnecessary homomorphisms. As an analogy, some discrete-logarithm systems are broken only if they have unnecessary homomorphisms. Eliminating unnecessary homomorphisms rescues these systems, and removes the need to worry about what attackers can do with these homomorphisms.

The current situation is that homomorphisms eliminated by NTRU Prime are used in the following attack papers: [32], [46], [38], [34], [39], and [15]. See our "NTRU Prime" paper for further details.

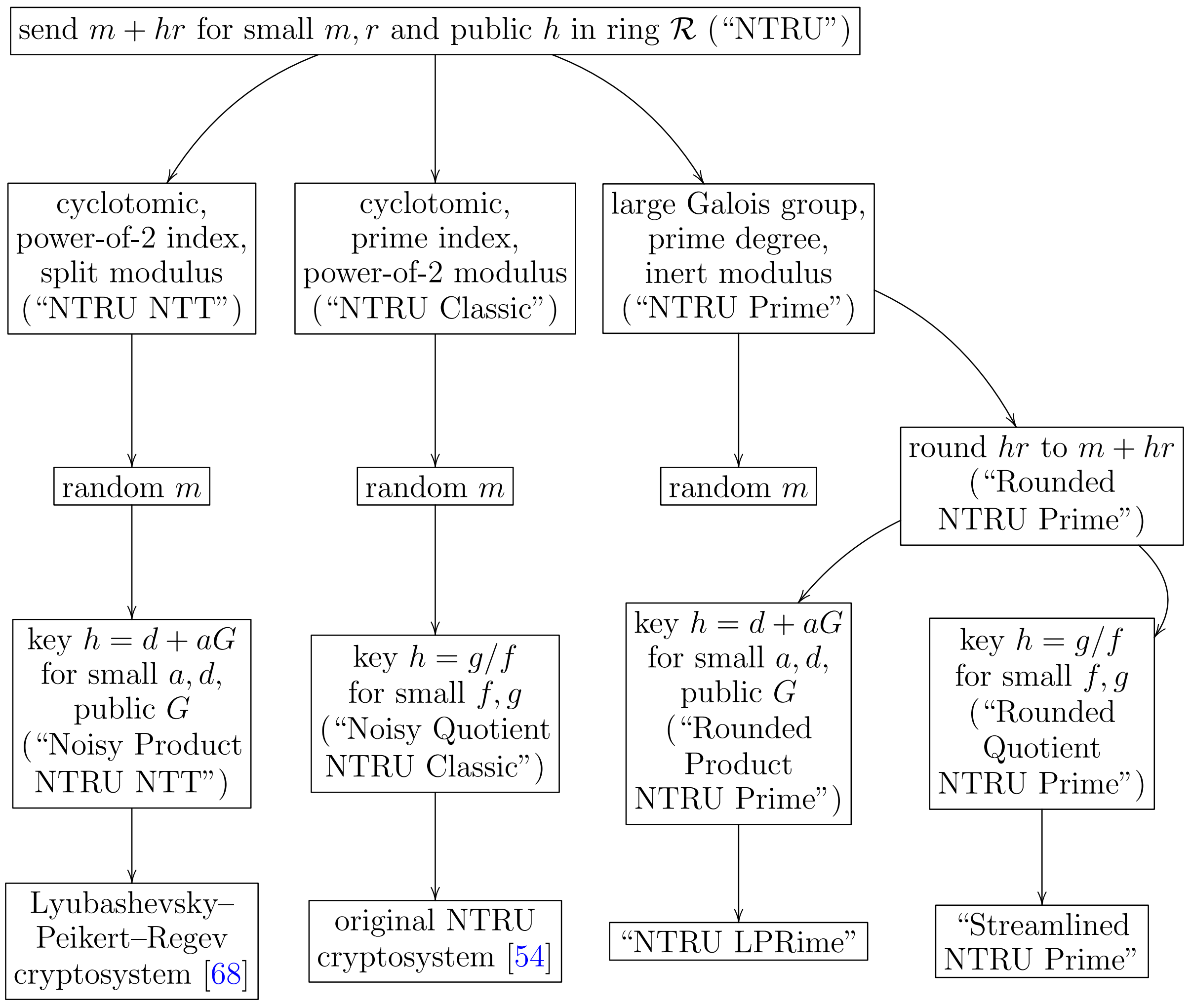
4.2 The public key

The receiver's public key, which we call h, is an element of $\mathcal{R}/q$.

4.3 Inputs and ciphertexts

In the original NTRU system, ciphertexts are elements of the form $m + hr \in \mathcal{R}/q$. Here $h \in \mathcal{R}/q$ is the public key as above, and m, r are small elements of $\mathcal{R}$ chosen by the sender.

Subsequent systems labeled as "NTRU" have generally extended ciphertexts to include ad-



send $m + hr$ for small m, r and public h in ring $\mathcal{R}$ (“NTRU”)
cyclotomic, power-of-2 index, split modulus (“NTRU NTT”)
cyclotomic, prime index, power-of-2 modulus (“NTRU Classic”)
large Galois group, prime degree, inert modulus (“NTRU Prime”)
round hr to $m + hr$ (“Rounded NTRU Prime”)
random m
random m
random m
key $h = d + aG$ for small a, d, public G (“Noisy Product NTRU NTT”)
key $h = g/f$ for small f, g (“Noisy Quotient NTRU Classic”)
key $h = d + aG$ for small a, d, public G (“Rounded Product NTRU Prime”)
key $h = g/f$ for small f, g (“Rounded Quotient NTRU Prime”)
Lyubashevsky–Peikert–Regev cryptosystem [68]
original NTRU cryptosystem [54]
“NTRU LPRime”
“Streamlined NTRU Prime”

ditional information, for various reasons explained below; but these cryptosystems all share the same core design element, sending $m + hr \in \mathcal{R}/q$ where m, r are small secrets and h is public. We suggest systematically using the name “NTRU” to refer to this design element, and more specific names (e.g., “NTRU Classic” vs. “NTRU Prime”) to refer to other design elements.

We refer to (m, r) as “input” rather than “plaintext” because in any modern public-key cryptosystem the input is randomized and is separated from the sender’s plaintext by symmetric primitives such as hash functions. See Section 4.5.

In the original NTRU specification [54], m was allowed to be any element of $\mathcal{R}$ having all coefficients in a standard range. The range was $\{-1, 0, 1\}$ for all of the suggested parameters, with q not a multiple of 3, and we focus on this case for simplicity (although we note that some other lattice-based cryptosystems have taken the smaller range $\{0, 1\}$, or sometimes larger ranges).

Current NTRU Classic specifications such as [53] prohibit m that have an unusually small number of 0's or 1's or -1's. For random m, this prohibition applies with probability $<2^{-10}$, and in case of failure the sender can try encoding the plaintext as a new m, but this is problematic for applications with hard real-time requirements. The reason for this prohibition is that NTRU Classic gives the attacker an "evaluate at 1" homomorphism from $\mathcal{R}/q$ to $\mathbb{Z}/q$, leaking $m(1)$. The attacker scans many ciphertexts to find an occasional ciphertext where the value $m(1)$ is particularly far from 0; this value constrains the search space for the corresponding m by enough bits to raise security concerns. In NTRU Prime, $\mathcal{R}/q$ is a field, so this type of leak cannot occur.

Streamlined NTRU Prime actually uses a different type of ciphertext, which we call a "rounded ciphertext". The sender chooses a small r as input and computes $hr \in \mathcal{R}/q$. The sender obtains the ciphertext by rounding each coefficient of hr, viewed as an integer between $-(q-1)/2$ and $(q-1)/2$, to the nearest multiple of 3. This ciphertext can be viewed as an example of the original ciphertext $m + hr$, but with m chosen so that each coefficient of $m + hr$ is in a restricted subset of $\mathbb{Z}/q$.

With the original ciphertexts, each coefficient of $m + hr$ leaves 3 possibilities for the corresponding coefficients of hr and m. With rounded ciphertexts, each coefficient of $m + hr$ also leaves 3 possibilities for the corresponding coefficients of hr and m, except that the boundary cases $-(q-1)/2$ and $(q-1)/2$ (assuming $q \in 1 + 3\mathbb{Z}$) leave only 2 possibilities. In a pool of 2^{64} rounded ciphertexts, the attacker might find one ciphertext that has 15 of these boundary cases out of 761 coefficients; these occasional exceptions have very little impact on known attacks. It would be possible to randomize the choice of multiples of 3 near the boundaries, but we prefer the simplicity of having the ciphertext determined entirely by r. It would also be possible to prohibit ciphertexts at the boundaries, but as above we prefer to avoid restarting the encryption process.

More generally, we say "Rounded NTRU" for any NTRU system in which m is chosen deterministically by rounding hr to a standard subset of $\mathbb{Z}/q$, and "Noisy NTRU" for the original version in which m is chosen randomly. Rounded NTRU has two advantages over Noisy NTRU. First, it reduces the space required to transmit $m + hr$. Second, the fact that m is determined by r simplifies protection against chosen-ciphertext attacks; see Section 4.5.

[74, Section 4] used an intermediate non-deterministic possibility to provide some space reduction for a public-key cryptosystem: first choose m randomly, and then round $m + hr$, obtaining $m' + hr$. The idea of rounded hr as a *deterministic* substitute for noisy $m + hr$ was introduced in [14] in the context of a symmetric-key construction, was used in [9] to construct another public-key encryption system, and was further studied in [28] and [8]. All of the public-key cryptosystems in these papers have ciphertexts longer than Noisy NTRU, but applying the same idea to Noisy NTRU produces Rounded NTRU, which has shorter ciphertexts.

4.4 Key generation and decryption

In the original NTRU cryptosystem, the public key h has the form $3g/f$ in $\mathcal{R}/q$, where f and g are secret. Decryption computes $fc = fm + 3gr$, reduces modulo 3 to obtain fm, and multiplies by $1/f$ to obtain m.

Streamlined NTRU Prime changes the position of the 3, taking h as $g/(3f)$ rather than $3g/f$. Decryption computes $3fc = 3fm + gr$, reduces modulo 3 to obtain gr, and multiplies by $1/g$ to obtain r. This change lets us compute (m, r) by first computing r and then multiplying by h, whereas otherwise we would first compute m and then multiply by $1/h$. One advantage is that we skip computing $1/h$; another advantage is that we need less space for storing a key pair. This $1/h$ issue does not arise for NTRU variants that compute r as a hash of m, but those variants are incompatible with rounded ciphertexts, as discussed in Section 4.5.

More generally, we say "Quotient NTRU" for NTRU with h computed as a ratio of two secret small polynomials. An alternative is what we call "Product NTRU", namely NTRU with h of the form $d + aG$, where a and d are secret small polynomials. Here $G \in \mathcal{R}/q$ is public, like h, but unlike h it does not need a hidden multiplicative structure: it can be, for example, a standard chosen randomly by a trusted authority, or output of a long hash function applied to a standard randomly chosen seed, or (as proposed in [7]) output of a long hash function applied to a per-receiver seed supplied along with h as part of the public key.

Product NTRU does not allow the same decryption procedure as Quotient NTRU. The first Product NTRU system, introduced by Lyubashevsky, Peikert, and Regev in [68] (originally in talk slides in 2010), sends $e + rG$ as additional ciphertext along with $m + hr + M$, where, as before, m and r are small polynomials, e is another small polynomial, and M is a polynomial consisting of solely 0 or $\lfloor q/2 \rfloor$ in each position. The receiver computes $(m + hr + M) - a(e + rG) = M + m + dr - ae$, and rounds to 0 or $\lfloor q/2 \rfloor$ in each position, obtaining M. Note that $m + dr - ae$ is small, since all of m, d, r, a, e are small.

The ciphertext size here, two elements of $\mathcal{R}/q$, can be improved in various ways. One can replace hr with fewer coefficients, for example by summing batches of two or three coefficients [80], before adding M and m. Rounded Product NTRU rounds $hr+M$ to obtain $m+hr+M$, rounds rG to obtain $e + rG$, and (to similarly reduce key size) rounds aG to obtain $d + aG$. Decryption continues to work even if $m + hr + M$ is compressed to two bits per coefficient.

A disadvantage of Product NTRU is that r is used twice, exposing approximations to both rG and hr. This complicates security analysis compared to simply exposing an approximation to hr. State-of-the-art attacks against Ring-LWE, which reveals approximations to any number of random public multiples of r, are significantly faster for many multiples than for one multiple. Perhaps this indicates a broader weakness, in which each extra multiple hurts security.

Quotient NTRU has an analogous disadvantage: if one moves far enough in the parameter space [60] then state-of-the-art attacks distinguish g/f from random more efficiently than they distinguish $m + hr$ from random. Perhaps this indicates a broader weakness. On the other hand, if one moves far enough in another direction in the parameter space [92], then

g/f has a security proof.

We find both of these issues worrisome: it is not at all clear which of Product NTRU and Quotient NTRU is a safer option.[1] We see no way to simultaneously avoid both types of complications. We have opted to present details of Streamlined NTRU Prime, an example of Quotient NTRU Prime; and of NTRU LPRime, an example of Product NTRU Prime.

If exposing approximations to two multiples of r damages the security of Product NTRU, perhaps exposing fewer bits does less damage. The compression techniques mentioned above, such as replacing $m + hr + M$ with fewer coefficients and releasing only a few top bits of each coefficient, naturally expose fewer bits than uncompressed ciphertexts. NTRU LPRime releases a few top bits of each of the bottom coefficients of $m + hr + M$, enough coefficients to communicate a hard-to-guess input M.

The Quotient NTRU literature, except for the earliest papers, takes f of the form $1 + 3F$, where F is small. This eliminates the multiplication by the inverse of f modulo 3. In Streamlined NTRU Prime we have chosen to skip this speedup for two reasons. First, in the long run we expect cryptography to be implemented in hardware, where a multiplication in $\mathcal{R}/3$ is far less expensive than a multiplication in $\mathcal{R}/q$. Second, this speedup requires noticeably larger keys and ciphertexts for the same security level, and this is important for many applications, while very few applications will notice the CPU time for Streamlined NTRU Prime.

4.5 Padding, KEMs, and the choice of q

In Streamlined NTRU Prime and NTRU LPRime we use the modern "KEM+DEM" approach introduced by Shoup; see [89]. This approach is much nicer for implementors than previous approaches to public-key encryption. For readers unfamiliar with this approach, we briefly review the analogous options for RSA encryption.

RSA maps an input m to a ciphertext $m^e \bmod n$, where (n, e) is the receiver's public key. When RSA was first introduced, its input m was described as the sender's plaintext. This was broken in reasonable attack models, leading to the development of various schemes to build m as some combination of fixed padding, random padding, and a short plaintext; typically this short plaintext is used as a shared secret key. This turned out to be quite difficult to get right, both in theory (see, e.g., [90]) and in practice (see, e.g., [71]), although it does seem possible to protect against arbitrary chosen-ciphertext attacks by building m in a sufficiently convoluted way.

The "KEM+DEM" approach, specifically Shoup's "RSA-KEM" in [89] (also called "Simple

[1]Peikert claimed in [75], modulo terminology, that Product NTRU is "at least as hard" to break as Quotient NTRU (and "likely strictly harder"). This claim ignores the possibility of attacks against the reuse of r in Product NTRU. There are no theorems justifying Peikert's claim, and we are not aware of an argument that eliminating this reuse is less important than eliminating the g/f structure. For comparison, switching from NTRU NTT and NTRU Classic to NTRU Prime eliminates structure used in some state-of-the-art attacks without providing new structure used in other attacks.

RSA"), is much easier:

- Choose a uniform random integer m modulo n. This step does not even look at the plaintext.
- To obtain a shared secret key, simply apply a cryptographic hash function to m.
- Encrypt and authenticate the sender's plaintext using this shared key.

Any attempt to modify m, or the plaintext, will be caught by the authenticator.

"KEM" means "key encapsulation mechanism": $m^e \bmod n$ is an "encapsulation" of the shared secret key $H(m)$. "DEM" means "data encapsulation mechanism", referring to the encryption and authentication using this shared secret key. Authenticated ciphers are normally designed to be secure for many messages, so $H(m)$ can be reused to protect further messages from the sender to the receiver, or from the receiver back to the sender. It is also easy to combine KEMs, for example combining a pre-quantum KEM with a post-quantum KEM, by simply hashing the shared secrets together.

When NTRU was introduced, its input (m, r) was described as a sender plaintext m combined with a random r. This is obviously not secure against chosen-ciphertext attacks. Subsequent NTRU papers introduced various mechanisms to build (m, r) as increasingly convoluted combinations of fixed padding, random padding, and a short plaintext.

It is easy to guess that KEMs simplify NTRU, the same way that KEMs simplify RSA; we are certainly not the first to suggest this. However, all the NTRU-based KEMs we have found in the literature (e.g., [91] and [84]) construct the NTRU input (m, r) by hashing a shorter input and verifying this hash during decapsulation; typically r is produced as a hash of m. These KEMs implicitly assume that m and r can be chosen independently, whereas rounded ciphertexts (see Section 4.3) have r as the sole input. It is also not clear that generic-hash chosen-ciphertext attacks against these KEMs are as difficult as inverting the NTRU map from input to ciphertext: the security theorems are quite loose.

We instead follow a simple generic KEM construction introduced in the earlier paper [42, Section 6] by Dent, backed by a tight security reduction [42, Theorem 8] saying that generic-hash chosen-ciphertext attacks are as difficult as inverting the underlying function:

- Like RSA-KEM, this construction hashes the input, in our case r, to obtain the session key.
- Decapsulation verifies that the ciphertext is the correct ciphertext for this input, preventing per-input ciphertext malleability.
- The KEM uses additional hash output for key confirmation, making clear that a ciphertext cannot be generated except by someone who knows the corresponding input.

Key confirmation might be overkill from a security perspective, since a random session key will also produce an authentication failure; but key confirmation allows the KEM to be audited without regard to the authentication mechanism, and adds only 3% to our ciphertext size.

Dent's security analysis assumes that decryption works for all inputs. We achieve this in Streamlined NTRU Prime by requiring $q \geq 16w + 1$. Recall that decryption sees $3fm + gr$ in $\mathcal{R}/q$ and tries to deduce $3fm + gr$ in $\mathcal{R}$; the condition $q \geq 16w + 1$ guarantees that this works, since each coefficient of $3fm + gr$ in $\mathcal{R}$ is between $-(q-1)/2$ and $(q-1)/2$ by Theorem 3. Taking different shapes of m, r, f, g, or changing the polynomial $P = x^p - x - 1$, would change the bound $16w + 1$; for example, replacing g by $1 + 3G$ would change $16w + 1$ into $24w + 3$.

Similarly, NTRU LPRime takes $q \geq 16w + 2\delta + 3$ to avoid decryption failures. Sending along merely top bits of $m + hr + M$ means that there is an additional error, producing a slightly worse bound than in the Streamlined NTRU Prime case. Another difference in details is that decryption reconstructs only M, not m; NTRU LPRime chooses r deterministically[2] as a hash of M.

In lattice-based cryptography it is standard to take somewhat smaller values of q. The idea is that coefficients in $3fm + gr$ are produced as sums of many $+1$ and -1 terms, and these terms *usually* cancel, rather than conspiring to produce the maximum conceivable coefficient. However, this idea led to attacks that exploited occasional decryption failures; see [56] and, for an analogous attack on code-based cryptography using QC-MDPC codes, [50]. It is common today to choose q so that decryption failures will occur with, e.g., probability 2^{-80}; but this does not meet Dent's assumption that decryption always works. This nonzero failure rate appears to account for most of the complications in the literature on NTRU-based KEMs. We prefer to guarantee that decryption works, making the security analysis simpler and more robust.

4.6 The shape of small polynomials

As noted in Section 4.3, the coefficients of m are chosen from the limited range $\{-1, 0, 1\}$. The NTRU literature [54, 58, 52, 53] generally puts the same limit on the coefficients of r, g, and f, except that if f is chosen with the shape $1 + 3F$ (see Section 4.4) then the literature puts this limit on the coefficients of F. Sometimes these "ternary polynomials" are further restricted to "binary polynomials", excluding coefficient -1.

The NTRU literature further restricts the Hamming weight of r, g, and f. Specifically, a cryptosystem parameter is introduced to specify the number of 1's and -1's. For example, there is a parameter t (typically called "d" in NTRU papers) so that r has exactly t coefficients equal to 1, exactly t coefficients equal to -1, and the remaining $p - 2t$ coefficients equal to 0. These restrictions allow decryption for smaller values of q (see Section 4.5), saving space and time. Beware, however, that if t is *too* small then there are attacks; see our security analysis in Section 6.

[2]This requires another layer of security analysis beyond Dent's security analysis. The core question is whether it is hard to recover a random M from ciphertext and public key, when r is chosen randomly. The next question, the extra layer, is whether it is hard to recover a random M from ciphertext and public key, when r is chosen as a hash of M. The third question, addressed by Dent's security analysis, is whether the KEM is hard to break.

In Streamlined NTRU Prime we keep the requirement that r have Hamming weight $w = 2t$, and keep the requirement that these w nonzero coefficients are all in $\{-1, 1\}$, but we drop the requirement of an equal split between -1 and 1. This allows somewhat more choices of r. The same comments apply to f. Similarly, we require g to have all coefficients in $\{-1, 0, 1\}$ but the distribution is otherwise unconstrained. We also require that f and g be invertible in $\mathcal{R}/q$, which simply means nonzero given that $P(x)$ is irreducible for NTRU Prime, and that g be invertible in $\mathcal{R}/3$.

These changes would affect the conventional NTRU decryption procedure: they expand the *typical* size of coefficients of fm and gr, forcing larger choices of q to avoid *noticeable* decryption failures. But we instead choose q to avoid *all* decryption failures (see Section 4.5), and these changes do not expand our *bound* on the size of the coefficients of fm and gr.

In NTRU LPRime we similarly choose small weight-w polynomials with coefficients in $\{-1, 0, 1\}$ without restricting the distribution of -1 and 1 beyond the weight.

Elsewhere in the literature on lattice-based cryptography one can find larger coefficients: consider, e.g., the quinary polynomials in [44], and the even wider range in [7]. In [92], the coefficients of f and g are sampled from a very wide discrete Gaussian distribution, allowing a proof regarding the distribution of g/f. However, this appears to produce *worse* security for any given key size. Specifically, there are no known attack strategies blocked by a Gaussian distribution, while the very wide distribution forces q to be very large to enable decryption (see Section 4.5), producing a much larger key size (and ciphertext size) for the same security level. Furthermore, wide Gaussian distributions are practically always implemented with variable-time algorithms, creating security problems, as illustrated by the successful cache-timing attacks in [31] and [77].

4.7 Modifications for round 2

The design rationale stated above is identical (modulo reference numbers) to the design rationale stated in our round-1 submission. We are using the same trapdoor functions in round 2, for the same reasons. We have modified the surrounding CCA conversions to add extra layers of defense, as explained below.

Analyzing arguments for other trapdoor functions. As a baseline requirement, we consider only small lattice-based systems. We therefore disregard arguments that systems outside this scope, such as Classic McEliece, have lower risk. (This does not mean that we dispute these arguments.)

Within small lattice-based systems, we prioritize minimizing the difficulty of security review. Quantitative improvements in the exact performance-vs.-security tradeoffs—aiming for *even smaller* lattice-based systems—are a lower priority.

There is, for example, a reasonable argument that allowing occasional decryption failures improves the quantitative tradeoffs between speed and security against known attacks.[3]

[3]It is not clear how much improvement is possible. Some systems try to maximize the improvement

However, it is clear that allowing decryption failures creates a large additional problem for the security reviewer.

Similarly, it has always been clear that NTRU Classic and NTRU NTT allow some speedups that are not available for NTRU Prime. In [21] we wrote "We are not saying that the NTRU Prime rings have *zero* cost." However, NTRU Classic and NTRU NTT force the security reviewer to analyze the impact of unnecessary homomorphisms provided to the attacker.

When we encounter claims that changing our choices could reduce security risks, we analyze the technical merits of the claims and what the change would mean for security reviewers. For example, there is an argument that unnecessary homomorphisms have a longer history in lattice-based cryptography than NTRU Prime does, and are therefore less risky. A closer look shows, however, a recent history of unnecessary homomorphisms being used in increasingly sophisticated attacks that have broken various lattice-based systems. The security reviewer is forced to understand these attacks, and to ask how far the attacks can be pushed.

As another example, there is a claim that the deterministic choice of noise in Rounded NTRU could be exploited in attacks against one-wayness. One can, however, just as easily claim that the pseudorandom choice of noise in Noisy NTRU could be exploited in attacks against one-wayness. Looking beyond one-wayness, and considering a complete KEM designed for IND-CCA2 security, shows that starting from a deterministic trapdoor function eliminates some difficult questions for security reviewers.[4] Making Noisy NTRU deterministic is more complicated than simply using Rounded Quotient NTRU, which is naturally deterministic.

The most difficult question we have faced is the choice between Quotient NTRU and Product NTRU. The underlying problems are similar (see Section 6.2), but the differences could mean that Quotient NTRU is much easier to break, *or* that Product NTRU is much easier to break. Known attacks against extreme problems provide weak evidence in both directions, and security reviewers need to consider how far these attacks can be pushed. A sufficiently severe weakness in Quotient NTRU would outweigh the advantages of being deterministic.

There is a specific argument that Quotient NTRU provides an extra attack tool that is useful for very large q—a lattice with many independent short vectors—while Product NTRU does not. If this argument were correct then it would mean that Quotient NTRU raises an unnecessary question for security reviewers: namely, can the same attack tool be pushed to smaller values of q relevant to small lattice-based systems?

However, the argument is not correct. Product NTRU provides the same attack tool; see Section 6.3. A security reviewer has to ask how far this attack tool can be pushed against Quotient NTRU, but *also* has to ask how far this attack tool can be pushed against Product NTRU. The incorrect statement that this attack tool does not exist against Product NTRU has led to a lack of analysis, which is even more worrisome for the security reviewer than

by using (1) error correction and (2) choices of failure probabilities that are too large for proofs to be meaningfully applicable, but this also maximizes the concerns for the security reviewer.

[4]Derandomizing a randomized PKE is conceptually straightforward but does not have a tight security proof starting from one-wayness. Can the attacker exploit this gap? There are tighter proofs starting from indistinguishability assumptions, but how closely have cryptanalysts studied those asumptions?

the analysis that has been performed against Quotient NTRU.

In the long run, it would be useful to select one of these two options, so that security reviewers can stop worrying about potential weaknesses in the other option. However, the current literature does not seem to provide an adequate basis for making this selection. We provide both options.

CCA conversions. We now hash additional inputs into the session key, for reasons explained in Section 7. Concretely, the confirmation hash now includes the public key, and the session-key hash now includes the ciphertext (which in turn includes the confirmation hash).

We now use "implicit rejection" as an extra layer of defense against chosen-ciphertext attacks. Implicit rejection means that decapsulation of an invalid ciphertext returns a pseudorandom function of the ciphertext rather than returning failure. This makes it difficult for attackers to see which ciphertexts are valid.

If a perfectly correct deterministic PKE is one-way then implicit rejection (with or without confirmation) tightly produces ROM IND-CCA2 security; for a detailed proof see [25]. Beyond this, [83] claims QROM IND-CCA2 security, but the proof starts from a ciphertext-unrecognizability assumption that, compared to one-wayness, has received much less cryptanalytic attention. A tight ROM proof of this assumption appears in [83] given one-wayness and a confirmation hash; but QROM attacks could be faster than ROM attacks, and other attacks could be faster than QROM attacks.

One can argue that, given implicit rejection, current proofs do not show a clear advantage of *also* using confirmation, so one should eliminate confirmation. However, removing defenses purely on the basis of security proofs in limited models is a dangerous practice, as illustrated by [70]. Even if, hypothetically, QROM IND-CCA2 is tightly proven from a well-studied one-wayness assumption, security reviewers will be forced to consider IND-CCA2 attacks beyond QROM IND-CCA2 attacks. Confirmation is helpful in this analysis, since it prevents attackers from generating valid modifications to ciphertexts except by already knowing the corresponding inputs. Also, compared to implicit rejection, confirmation stops attacks at an earlier phase of the decapsulation process, as noted in [25]; there is a plausible argument that this reduces the cost of side-channel protection.

Underlying hash function. We have decided to use only 256 bits of SHA-512 output. To generate 512 bits of output—in particular, a confirmation and a session key—we hash twice with different inputs rather than using a single hash. This is convenient in the context of implicit rejection; simplifies the inclusion of extra inputs such as the public key; and makes it very easy to switch to another 256-bit hash function if desired.

Encodings of sequences of integers. There are three standard strategies to encode an integer r_0 modulo m_0, an integer r_1 modulo m_1, etc. as a sequence of bytes.

The first strategy is to simply output a byte. Specifically, if m_i is divisible by 256, encode r_i modulo m_i as the byte $r_i \bmod 256$ along with an encoding of $\lfloor r_i/256 \rfloor$ modulo $m_i/256$.

The second strategy is to increase m_i to, e.g., $m'_i = 256\lceil m_i/256 \rceil$: encode r_i modulo m_i by encoding r_i modulo m'_i. This extends the applicability of the first strategy to arbitrary mod-

secret key	public key	ciphertext	total	
1518	994	897	1891	`sntrup653`
1125	897	1025	1922	`ntrulpr653`
1763	1158	1039	2197	`sntrup761`
1294	1039	1167	2206	`ntrulpr761`
1238	1047	1175	2222	`ntrulpr4591761` (round 1)
1600	1218	1047	2265	`sntrup4591761` (round 1)
1463	1184	1312	2496	`ntrulpr857`
1999	1322	1184	2506	`sntrup857`

Table 1: Bytes for various objects. Sorted by "total", the total of public-key bytes and ciphertext bytes. The trapdoor functions for `sntrup761` and `ntrulpr761` are identical to the trapdoor functions for round-1 `sntrup4591761` and `ntrulpr4591761` but the encodings are now more efficient.

uli, but it loses some space efficiency for the unused encodings of $\{m_i, m_i + 1, \ldots, m'_i - 1\}$. As an extreme case, a bit is encoded as a byte. The inefficiency disappears as m_i grows.

An alternative to the second strategy, in contexts where there should not be any unused strings (see [23]), is to decrease m_i to $256\lfloor m_i/256\rfloor$, rejecting any input where $r_i \geq 256\lfloor m_i/256\rfloor$. The rejection probability is 1 if $m_i < 256$ but becomes acceptably small if m_i is large enough.

The third strategy is to combine two integers into one: e.g., encode r_0 modulo m_0 and r_1 modulo m_1 by encoding $r_0 + m_0 r_1$ modulo $m_0 m_1$. This reduces the inefficiency of the second strategy, at the expense of working with larger integers.

There are several ways to evaluate the merit of a combination of these strategies. Compared to the combination used in our round-1 submission, our new Encode function has similar simplicity, much more generality, better space-efficiency, typically less arithmetic (systematically fitting products comfortably below 32 bits and other computations comfortably below 16 bits), similar parallelizability, and better vectorizability. For comparison, a conventional "range encoder" has the same generality and space-efficiency but works from left to right and is not parallelizable.

5 Detailed performance analysis (2.B.2)

Expanded and updated for round 2.

5.1 Space

See Table 1.

5.2 Time

Description of platform. The following measurements were collected on a computer named `titan0`. The CPU on `titan0` is an Intel Xeon E3-1275 v3 (Haswell) running at 3.5 GHz. Benchmarks used one core of the CPU. Turbo Boost is disabled. `titan0` has 32GB of RAM and runs Ubuntu 16.04; it ran Ubuntu 14.04 when we collected benchmarks for round 1.

NIST says that the "NIST PQC Reference Platform" is "an Intel x64 running Windows or Linux and supporting the GCC compiler." `titan0` is an Intel x64 running Linux and supporting the GCC compiler. Beware, however, that different Intel CPUs have different cycle counts.

Measurements. The official `supercop-20190110` measurements report the following quartile/median/quartile cycle counts on `titan0`:

- 934376/940852/969872 for `sntrup4591761` key generation.
- 44672/44788/44956 for `sntrup4591761` encapsulation.
- 93452/93676/93856 for `sntrup4591761` decapsulation.
- 44796/44948/45140 for `ntrulpr4591761` key generation.
- 80904/81144/81548 for `ntrulpr4591761` encapsulation.
- 113448/113708/114012 for `ntrulpr4591761` decapsulation.

For comparison, the speeds reported in the round-1 submission were, e.g., 59456 cycles for `sntrup4591761` encapsulation and more than 6 million cycles for `sntrup4591761` key generation. The speedups in `sntrup4591761` key generation came primarily from speedups in constant-time inversion; see [26]. The other speedups came primarily from speedups in constant-time sorting; see [18].

Estimates for round 2. Formally, the above round-1 timings are our current estimates of round-2 timings on the NIST PQC Reference Platform. We expect a closer look to show the following effects: moving from 761 to 857 loses time; moving from 761 to 653 saves time; extra hashing loses time; new multiplication techniques save time.

5.3 Does key-generation time matter?

If a key is used to decapsulate N ciphertexts then the total receiver time is 1 key generation plus N decapsulations. The percentage of time spent on key generation disappears as N increases. We review three arguments that key-generation time is nevertheless important.

The first argument is that RSA key generation is so slow that typical sizes of N often do not compensate; users often limit RSA key sizes specifically to limit key-generation costs. This argument has no obvious applicability to systems with much faster key generation, such as Streamlined NTRU Prime and NTRU LPRime.

The second argument is that many lattice-based cryptosystems do not resist chosen-ciphertext attacks, so they have to generate a new key for every ciphertext, so their key-generation time is important. In other words, these systems require N to be 1. This argument applies to, e.g., BCNS [30], the original version of New Hope [7], the original version of Frodo [29], and HILA5 [82]. However, Streamlined NTRU Prime and NTRU LPRime are designed for IND-CCA2 security. With an IND-CCA2 system, it is safe to generate a key once and use the key any number of times.

The third argument is that N must be 1 for forward secrecy. This argument is incorrect. Forward secrecy requires keys to be erased quickly, but "quickly" is a time limit. Generating a new key for every ciphertext is insufficient (see [66]) and unnecessary. A typical quad-core 3GHz server generating a new short-term `sntrup4591761` key every 10 seconds is using under 1/100000 of its CPU time on key generation with our current software; this is a negligible cost, no matter what N is.

A user who (for some reason) wants to generate many Streamlined NTRU Prime keys more quickly than this can use Montgomery's trick to batch the inversions. Montgomery's trick replaces (e.g.) 1000 inversions with 2997 multiplications and just 1 inversion.

5.4 Do encapsulation time and decapsulation time matter?

Our decision to use fixed-weight vectors instead of variable-weight vectors improves the quantitative tradeoff between space and security against known attacks, but the same decision costs extra time for generating those vectors. Our decision to use a small f in Streamlined NTRU Prime instead of $f = 1 + 3F$ improves the same tradeoff, but costs extra time for a division by f modulo 3. These decisions are based on a broader assessment that, for typical lattice-based KEMs, space is a much more important cost factor than time.

One way to put space and time on the same scale, as in [21, full version, footnote 5] and [19], is to consider a quad-core 3GHz CPU handling a 100Mbps Internet connection. In 1 millisecond, each core runs 3 million cycles, so the 4 cores run a total of 12 million cycles—enough time to decapsulate about 120 `sntrup4591761` ciphertexts. In the same 1 millisecond, the Internet connection transmits only 12500 bytes—only about 12 `sntrup4591761` ciphertexts.

This scenario suggests that each cycle used in decapsulation is 1000× less expensive than each byte used in a ciphertext: for example, saving 20 bytes is worth 20000 cycles. The exact ratio changes if one varies the CPU speed and the network speed. It would be useful for the community to agree upon a spectrum of data points regarding the costs of communication and the costs of computation in various environments.

5.5 How parameters affect performance

Streamlined NTRU Prime public keys: An element of the ring $\mathcal{R}/q$ is represented as slightly more than $p \log_{256} q$ bytes.

Streamlined NTRU Prime ciphertexts, NTRU LPRime public keys, NTRU LPRime ciphertexts: The main space in each object is a *rounded* element of $\mathcal{R}/q$, which is represented as slightly more than $p \log_{256}((q + 2)/3)$ bytes. Rounding thus saves approximately $0.2p$ bytes.

NTRU LPRime ciphertexts also use 128 bytes for Top output. These bytes encode $I = 256$ quantities, each quantity having 4 bits. Reducing the I parameter to 192 would save 32 bytes. It is also possible to use (e.g.) only 3 bits in each quantity, but then δ needs to be larger (about $q/8$), so w needs to be smaller (at most about $3q/64$).

There are 32 extra bytes in each ciphertext for confirmation; the length is defined by the Confirm parameter. There are 32 extra bytes in each NTRU LPRime public key for a seed; the length is defined by the Seeds parameter.

Regarding time: The asymptotic number of bit operations for multiplication in $\mathcal{R}/q$ is essentially linear in $p \log_2 q$, the number of bits in a ring element: one can multiply in $\mathcal{R}/q$ by multiplying integers with $(p \log_2 q)^{1+o(1)}$ bits, and multiplying n-bit integers takes $n^{1+o(1)}$ bit operations by standard FFT-based techniques. Accounting for the real cost of memory increases the asymptotic cost exponent from 1 to 1.5; see Section 6.6. The asymptotic number of bit operations for inversion is also essentially linear in $p \log_2 q$, but inversion is more difficult to parallelize than multiplication. These asymptotics are not a substitute for concrete cost analyses.

6 Analysis of known attacks (2.B.5)

Expanded and updated for round 2.

This section summarizes the *known* cryptanalytic attacks on small lattice-based encryption systems, including Streamlined NTRU Prime and NTRU LPRime. This section also provides estimates of the complexity of these attacks. See Section 7 for applications to parameter selection.

6.1 Warnings

To assist security reviewers, this section points out various ways that analyses from the literature are, or could be, **overestimating** the cost of known attacks. Sometimes the claimed costs are above the actual attack costs. Sometimes the attacks that are analyzed are not the fastest known attacks.

This section also points out various ways that analyses are, or could be, **underestimating** the cost of attacks. Sometimes the claimed costs are below the actual attack costs. Sometimes the attacks that are analyzed are speculative improvements, incorrectly conflating two separate topics: (1) analyses of known attacks; (2) analyses of risks of advances in attacks.

Overestimates create an obvious risk: a system that was assessed to have an acceptable security level, on the basis of overestimates, might turn out to have an unacceptable security level, perhaps even a breakable security level. The risk is higher when the overestimates are quantitatively larger. The risk is also higher when systems aim for low security levels.

Underestimates create risks that are less obvious. The following quote from [24, full version, Appendix B.1.2] explains the danger of underestimates:

> **The more-conservative-is-better fallacy.** One might also argue that ... underestimating the cost of an attack is perfectly safe, since it simply leads users to choose larger parameters.
>
> In fact, "conservative" underestimates can cause users to *lose* security. There are three important effects ignored in the "more conservative is better" argument: first, users are subject to cost constraints, and cannot simply choose larger parameters; second, users *can* choose different systems, and in fact take advantage of this flexibility with the goal of maximizing security subject to the cost constraints; third, underestimates in general vary from one system to another, and in particular the gaps considered in this paper vary from one system to another.

As one of many examples in lattice-based cryptography, consider Schanck's recent analysis [85] of NTRU parameters. Schanck uses two different security-estimation mechanisms, producing roughly a 50% difference in the size required for each security level but also producing different recommendations regarding the best error fraction to choose inside system designs. Specifically, Schanck states that "a small weight parameter can be detrimental to the size vs. security trade-off" but that this is not clearly shown by the "Core-SVP" estimate. Below we review various reasons to think that "Core-SVP" is an underestimate, and that this underestimate has the effect of understating the influence of weight on security.

6.2 Attack problems

Consider the following three different problems. Each problem is to find a small $s \in \mathcal{R}$ of weight w, but the problems vary in the information provided about s:

- Problem 1: We are given $A \in \mathcal{R}/q$ such that $As + e = 0$ for some small $e \in \mathcal{R}$.
- Problem 2: We are given $A \in \mathcal{R}/q$ and given $As + e$ for some small $e \in \mathcal{R}$.
- Problem 3: We are given $A_1, A_2 \in \mathcal{R}/q$ and given $A_1 s + e_1, A_2 s + e_2$ for some small $e_1, e_2 \in \mathcal{R}$.

A solution to Problem 2 for all (s, e, A) implies a solution to Problem 3 for all (s, e_1, A_1, e_2, A_2): simply forget e_2, A_2. It also implies a solution to Problem 1 for all (s, e) where e is a multiple of s, and in particular a solution to Problem 1 whenever s is invertible in $\mathcal{R}/q$. A solution to Problem 2 for *uniform random* (s, e, A) implies a solution to Problem 3 for uniform random (s, e_1, A_1, e_2, A_2). On the other hand, there is no proof known that a solution to Problem 2 for uniform random (s, e, A) implies a solution to Problem 1 for uniform random (s, e).

One can see these problems, for uniform random inputs, as models of attacks on Quotient NTRU (e.g., Streamlined NTRU Prime) and Product NTRU (e.g., NTRU LPRime):

- Problem 1 models the problem of finding a secret key from a public key in Quotient NTRU. The decisional version of Problem 1 models the problem of distinguishing a public key from uniform.
- Problem 2 models the problem of finding a plaintext in Quotient NTRU from a public key and a ciphertext. One can reduce to the uniform-A case by assuming that public keys are indistinguishable from uniform.
- Problem 2 also models the problem of finding a secret key from a public key in Product NTRU. The decisional version of Problem 2 models the problem of distinguishing a public key from uniform.
- Problem 3 models the problem of finding a plaintext in Product NTRU from a public key and a ciphertext (although normally e_2 has a much larger range than e_1 in this setting, perhaps making Problem 3 more difficult than it would be for smaller e_2). Again one can reduce to the uniform case by assuming that public keys are indistinguishable from uniform.

There are various reasons that these models could be underestimating or overestimating security. For example:

- Considering the decisional versions of Problems 1 and 2 (so as to decompose attack analyses into key attacks and ciphertext attacks) could be underestimating security. On the other hand, considering only the search problems could be overestimating security: an attacker could find a secret key short enough to decrypt ciphertexts without finding something as short as the original secret key.
- Some inputs in NTRU LPRime are actually generated as hash outputs. All known proofs from one-wayness (1) require modeling the hash as a random oracle and (2) have a large looseness factor. Similar comments apply to many other Product NTRU cryptosystems in the literature.
- The A in NTRU LPRime is pseudorandom output from a public seed. Here the proofs are even less satisfactory.
- Rounded NTRU (as in Streamlined NTRU Prime and NTRU LPRime) chooses the error e deterministically through rounding. As noted in Section 4.3, the deterministic choice leaks information at the edges of the q interval. An analysis that ignores this effect is overestimating security, although our calculations indicate that this is a small effect.
- Noisy NTRU chooses e pseudorandomly. Compared to rounding, this might make s harder to find, but it might also make s easier to find. Known attacks do not support conclusions either way. Known reductions between Ring-LWE and Ring-LWR are too weak to support conclusions either way.

As noted earlier, we are concerned about the possibility that Problem 1 could be weaker than Problem 2, but we are also concerned about the possibility that Problem 3 could be weaker than Problem 2. We analyze known attacks against all three problems.

Historically, both Problem 1 and Problem 2 were already studied in the earliest NTRU papers, but attacks have improved dramatically since then and are still a moving target. We plan to periodically issue updated attack analyses to reflect the progress of research.

6.3 Lattice perspectives on the attack problems

One can view each of the search problems stated above as the problem of finding a short nonzero solution to a homogeneous system of equations over $\mathcal{R}/q$, if “short” is given an appropriate definition in each line:

- Problem 1: Given $A \in \mathcal{R}/q$, find a short nonzero solution $(s, e) \in \mathcal{R}^2$ to $As + e = 0$.
- Problem 2: Given $A, b \in \mathcal{R}/q$, find a short nonzero solution $(s, t, e) \in \mathcal{R}^3$ to $As+e = bt$. (By hypothesis there is a solution of the form $(s, 1, e)$.)
- Problem 3: Given $A_1, b_1, A_2, b_2 \in \mathcal{R}/q$, find a short nonzero solution $(s, t_1, t_2, e_1, e_2) \in \mathcal{R}^5$ to the system of equations $A_1s + e_1 = b_1t_1$, $A_2s + e_2 = b_2t_2$.

Each solution space is a full-rank lattice. To see this, first rewrite the equations over $\mathcal{R}/q$ as equations over $\mathcal{R}$, lifting the inputs to $\mathcal{R}$ and using an explicit multiple of q in each equation:

- Problem 1: Given $A \in \mathcal{R}$, find a short nonzero solution $(s, e, r) \in \mathcal{R}^3$ to $As + e = qr$.
- Problem 2: Given $A, b \in \mathcal{R}$, find a short nonzero solution $(s, t, e, r) \in \mathcal{R}^4$ to $As + e = bt + qr$.
- Problem 3: Given $A_1, b_1, A_2, b_2 \in \mathcal{R}$, find a short nonzero solution $(s, t_1, t_2, e_1, e_2, r_1, r_2) \in \mathcal{R}^7$ to the system of equations $A_1s + e_1 = b_1t_1 + qr_1$, $A_2s + e_2 = b_2t_2 + qr_2$.

Then eliminate the variable having coefficient 1 in each equation:

- Problem 1: Find a short nonzero vector (s, e) in the image of the map $(s, r) \mapsto (s, qr - As)$ from $\mathcal{R}^2$ to $\mathcal{R}^2$.
- Problem 2: Find a short nonzero vector (s, t, e) in the image of the map $(s, t, r) \mapsto (s, t, bt + qr - As)$ from $\mathcal{R}^3$ to $\mathcal{R}^3$.
- Problem 3: Find a short nonzero vector (s, t_1, t_2, e_1, e_2) in the image of the map $(s, t_1, t_2, r_1, r_2) \mapsto (s, t_1, t_2, b_1t_1 + qr_1 - A_1s, b_2t_2 + qr_2 - A_2s)$ from $\mathcal{R}^5$ to $\mathcal{R}^5$.

These are short-vector problems in lattices of ranks $2p, 3p, 5p$ respectively.

One should not think that larger ranks are necessarily more difficult. On the contrary, recall that Problem 3 (rank $5p$) cannot be more difficult than Problem 2 (rank $3p$), since one can simply disregard A_2, b_2, e_2, t_2. More generally, for each problem, one can separate each equation over $\mathcal{R}$ into p equations over $\mathbb{Z}$, and choose how many of the equations over $\mathbb{Z}$ to discard. Later we will see that this flexibility saves some time in known attacks.

Each of the lattices shown above has (generically) many independent short vectors. The point here is that the solution spaces are $\mathcal{R}$-modules: i.e., one can multiply any solution by an element of $\mathcal{R}$ to obtain another solution. In particular, multiplying a short solution by x produces another short solution. (For NTRU NTT the size is identical. For NTRU Prime the size usually increases slightly; multiplying p times by x is the same as multiplying by $x + 1$, which almost always increases the weight considerably. See Section 6.5.)

May and Silverman [69] in 2001 pointed out that forcing some of the coefficients of s to be 0 saves time in various attacks against Problem 1, because it reduces the lattice rank, even though (1) this reduces the success probability—perhaps the secret s is nonzero in those coefficients—and (2) the resulting lattice is no longer an $\mathcal{R}$-module. This speedup also applies to Problem 2, and here the attacker can force many more coefficients of s and t to be 0: the standard choice is to force almost all of the coefficients of t to be 0, constraining t to $\mathbb{Z}$ (which still allows $t = 1$), and also optionally force some of the coefficients of s to be 0. Similar comments apply to Problem 3.

The lattice obtained in Problem 2 by taking $t \in \mathbb{Z}$, the image of the map $(s, t, r) \mapsto (s, t, bt + qr - As)$ from $\mathcal{R} \times \mathbb{Z} \times \mathcal{R}$ to $\mathcal{R} \times \mathbb{Z} \times \mathcal{R}$, is typically called the "Bai–Galbraith embedding" of Problem 2. Similarly, the image of the map $(s, t_1, t_2, r_1, r_2) \mapsto (s, t_1, t_2, b_1t_1 + qr_1 - A_1s, b_2t_2 + qr_2 - A_2s)$ from $\mathcal{R} \times \mathbb{Z}^2 \times \mathcal{R}^2$ to $\mathcal{R} \times \mathbb{Z}^2 \times \mathcal{R}^2$ is the "Bai–Galbraith embedding" of Problem 3. Projecting away the first component of this lattice gives the image of the map $(s, t_1, t_2, r_1, r_2) \mapsto (t_1, t_2, b_1t_1 + qr_1 - A_1s, b_2t_2 + qr_2 - A_2s)$ from $\mathcal{R} \times \mathbb{Z}^2 \times \mathcal{R}^2$ to $\mathbb{Z}^2 \times \mathcal{R}^2$; this image is typically called the "Kannan embedding" of Problem 3.

The natural $\mathcal{R}$-modules shown above for Problems 2 and 3, without the added constraint $t \in \mathbb{Z}$, do not seem to be very widely known. On the contrary, there seems to be a common belief that Problem 1 can be attacked via an $\mathcal{R}$-module while Problems 2 and 3 cannot. Typically this is phrased as a claim that "NTRU" (meaning Quotient NTRU) is easier to attack than "Ring-LWE" (meaning Product NTRU) since "the NTRU lattice" (the $\mathcal{R}$-module shown above for Problem 1) has many independent short vectors while "the Ring-LWE lattice" (the Bai-Galbraith embedding or the Kannan embedding) has unique short vectors modulo negation. However:

- There are many choices of lattices for attacking Ring-LWE. These choices include the natural $\mathcal{R}$-modules shown above, which have many independent short vectors.
- The fastest attacks known on all of these problems select a sublattice that is not an $\mathcal{R}$-module. These attacks obtain only a limited benefit from the original number of short vectors.

See Figure 3. Mathematically, it is defensible to consistently define "the" NTRU lattice and "the" Ring-LWE lattice as the $\mathcal{R}$-modules shown above for Problems 1 and 2 respectively, but then it is incorrect to claim that "the" Ring-LWE lattice has unique short vectors modulo negation.

We are not saying that the $\mathcal{R}$-module structure is outside the attack surface of lattice-based cryptography. On the contrary: it is important to understand how the $\mathcal{R}$-module structure can be exploited in attacks. For example, the analysis of [60] says that if q is very large ("overstretched") in Problem 1 then lattice attacks against the $\mathcal{R}$-module benefit from the

	Problem 1	Problem 2
natural lattice; $\mathcal{R}$-module	image of the map $(s,r) \mapsto (s, qr - As)$ from $\mathcal{R}^2$ to $\mathcal{R}^2$	image of the map $(s,t,r) \mapsto (s,t,bt + qr - As)$ from $\mathcal{R}^3$ to $\mathcal{R}^3$
	$\rotatebox{90}{$\subseteq$}$	$\rotatebox{90}{$\subseteq$}$
sublattice obtained from speedup of [69]; not an $\mathcal{R}$-module	subset where some coefficients of s are forced to be 0	subset where some coefficients of s,t are forced to be 0

Figure 3: Choices of lattices for attacking Problem 1 ("NTRU") and Problem 2 ("Ring-LWE with 1 sample"). In both cases, the natural lattice for attacking the problem is an $\mathcal{R}$-module and thus has many independent short vectors. In both cases, the speedup from [69] selects a sublattice that is not an $\mathcal{R}$-module.

---

presence of many independent short vectors (so [69] is a slowdown for such large q rather than a speedup). The analogous analysis for Problems 2 and 3 has not been done; [60, Section 7] appears to claim, incorrectly, that no $\mathcal{R}$-module is available to attack Problems 2 and 3. As another example, algebraic structure features prominently in quantum polynomial-time breaks of some cyclotomic lattice problems (see [27]) and non-quantum quasi-polynomial-time breaks of some multiquadratic lattice problems (see [15]). However, Section 6 of this document focuses on *known* attacks against *small* lattice-based encryption systems; this focus excludes the attacks of [60], [27], and [15].

6.4 Estimate all the $\{\mathbf{LWE}, \mathbf{NTRU}\}$ schemes!

In 2018, Albrecht, Curtis, Deo, Davidson, Player, Postlethwaite, Virdia, and Wunderer [4] published a table showing many different security estimates for many different post-quantum proposals. In the latest version of this table, the "primal" attack against NTRU Prime has

- non-quantum estimates ranging from 2^{155} through 2^{410} for Streamlined NTRU Prime 4591^{761} (for comparison, our round-1 submission estimated 2^{248});
- non-quantum estimates ranging from 2^{156} through 2^{398} for NTRU LPRime 4591^{761};
- quantum estimates ranging from 2^{140} through 2^{200} for Streamlined NTRU Prime 4591^{761}; and
- quantum estimates ranging from 2^{141} through 2^{202} for NTRU LPRime 4591^{761}.

The table also lists somewhat higher estimates for the cost of a "dual" attack against NTRU LPRime 4591^{761}.

We now review the analysis that produced these estimates. We focus on the "primal" attack here; note that this could be an overestimate of security if the "dual" attack actually produces better results. We use the smallest reported estimates, 2^{140} and 2^{141}, as case studies.

Streamlined NTRU Prime key recovery. The objective of the following attack is to find a small weight-w element $f \in \mathcal{R}$ and a small element $g \in \mathcal{R}$ given the ratio $h = g/(3f) \in \mathcal{R}/q$. The distribution of g is not exactly uniform, since g is required to be invertible modulo 3, but this has very little effect on the analysis. We obtain Problem 1 by relabeling $3h, f, g$ as $A, s, -e$ respectively.

Consider, as explained in Section 6.3, the rank-$2p$ lattice of pairs $(s, qr - As) \in \mathcal{R}^2$ for $(s, r) \in \mathcal{R}^2$. This lattice has a short vector $(f, -g)$.

In the first case study, the 2^{140} estimate for Streamlined NTRU Prime 4591^{761} key recovery, the 2-norm of f is $\sqrt{286} = 16.91\ldots$, and the 2-norm of g is usually close to $\sqrt{2p/3} = 22.52\ldots$. The analysis does not take into account the variations in the 2-norm of g; this simplification could overestimate or underestimate security.

The attack picks an integer k with $0 \le k \le p$, picks k of the first p positions (the positions holding s) in the lattice vectors, and considers the sublattice where these positions are all 0. This sublattice has rank $2p - k$. The chance that $(f, -g)$ is in this sublattice is $\binom{p-k}{w}/\binom{p}{w}$: there are $\binom{p}{w}$ ways to choose positions for the nonzero entries of f, and only $\binom{p-k}{w}$ ways where these positions avoid the specified k positions. In the first case study, [4] chooses $k = 11$, and then the chance is $0.00536\ldots$.

There are also rotations such as $(xf, -xg)$ in the lattice, and possibly in the sublattice. At this point [4] assumes that these rotations provide p chances for short vectors in the sublattice, increasing the overall chance to $0.983\ldots$ in this case study. There is an implicit assumption here that the chances are independent; as noted in [69], this is not true in general, and the attacker can search for choices of positions of entries where the non-independence gives a better overall chance. The calculation in [4] could thus overestimate security. In the opposite direction, there is a reason that the calculation underestimates security: most rotations for NTRU Prime are larger than (f, g) and thus more difficult for lattice attacks to find. See Section 6.5 below.

The attack also picks an integer m with $0 \le m \le p$, picks m of the last p positions (the positions holding $qr - As$) in the lattice vectors, and projects the (sub)lattice away from the other $p - m$ positions. Formally, consider the map that

- inputs $(s, r) \in \mathcal{R}^2$ where the specified k positions in s are 0, and
- outputs $(s, \mathrm{Extract}(qr - As))$ where Extract outputs m out of the p input positions.

The image of this map is a lattice of rank $p - k + m$ and determinant q^m. In the first case study, [4] chooses $m = 576$, so the rank is 1326.

Because g is (almost always) larger than f, the attack "rescales" the lattice, assigning higher weight to the $p - k$ positions in s than to the m remaining positions. If the scale factor is λ then the scaled 2-norm of f is $\lambda\sqrt{w}$, while the 2-norm of the remaining components of g is usually close to $\sqrt{2m/3}$, for a total of $\sqrt{\lambda^2 w + 2m/3}$. Meanwhile the scaling increases the lattice determinant to $\lambda^{p-k} q^m$. If g is actually smaller than f then similar comments apply but with $\lambda < 1$.

The attack now applies a basis-reduction algorithm, specifically BKZ, hoping to discover the short vector. BKZ has one important parameter for the analysis, a "block size" β. The analysis predicts the success of BKZ according to the following heuristics:

- BKZ-β finds a nonzero vector of length approximately $\delta^d (\det L)^{1/d}$ in a "random" rank-d lattice L, where $\delta = (\beta(\pi\beta)^{1/\beta}/(2\pi e))^{1/(2(\beta-1))}$.

  This is clearly wrong for $\beta < 13$ (since then $\delta < 1$), and not reasonable for $\beta < 36$ (since δ increases with β in this range). For larger β, limited experimental evidence suggests that the formula overstates the ability of BKZ-β to find short vectors, underestimating security. This formula is from an asymptotic analysis from [35] as $\beta \to \infty$; even if the asymptotic analysis is correct, it does not correctly predict the behavior of BKZ-β for concrete values of β.

- The "geometric-series assumption" from [86] holds.

  Considerable experimental evidence shows that this assumption is close to correct. There are, however, some systematic deviations from the assumption; see generally [13].

- BKZ-β finds a unique (modulo negation) shortest nonzero vector v if and only if the 2-norm of v is at most $\delta^{2\beta-d}(\det L)^{1/d}\sqrt{d/\beta}$.

  This heuristic was introduced in [7]. Limited experimental evidence suggests that this heuristic overestimates security, and that smaller block sizes have a considerable probability of success. See generally [6]; see also the more precise (and heuristically argued to be more accurate) probability formulas in [97, Chapter 5] for a similar problem.

In the first case study, [4] chooses $\beta = 528$, so $\delta = 1.003274\ldots$, and chooses $\lambda = 1.32$. The short vector usually has 2-norm around $\sqrt{\lambda^2 w + 2m/3} = 29.7\ldots$. Meanwhile $\delta^{2\beta-d}(\det L)^{1/d}\sqrt{d/\beta} = 29.8\ldots$, so the heuristics say that BKZ will find the short vector.

The parameter search was not comprehensive in [4], so it could have overestimated security for any particular parameter set. We observe that $(k, \beta) = (13, 525)$ would have produced slightly better results for the same case study.

Finally, the analysis assumes that BKZ-β costs $2^{0.265\beta}$ quantum operations. For this case study with the parameters from [4], there are $1/0.983\ldots$ retries of BKZ-528, for a total of $2^{139.94\ldots}$ quantum operations.

To illustrate the accuracy questions regarding δ, we tried the BKZ simulator from [36] for the choices made in [4] for the first case study. This simulator computes the vector lengths produced by a particular model of BKZ; see also [13] for a newer model of BKZ. The simulator reports that $\beta = 528$ reaches only $\delta \approx 1.00332$; that limiting the number of BKZ "tours" (see Section 6.9) to 8, as commonly recommended, reaches only $\delta \approx 1.00334$; and that reaching $\delta \approx 1.00327$ then requires increasing β to 544. On the other hand, such a large jump in

β is overkill: each step in β also affects $\sqrt{d/\beta}$, which has a larger effect than $\delta^{2\beta-d}$ in this parameter range.

NTRU LPRime key recovery. The objective of the following attack is to find a small weight-w element $a \in \mathcal{R}$ and a small element $d \in \mathcal{R}$ given $G \in \mathcal{R}/q$ and $b = Ga + d \in \mathcal{R}/q$. We obtain Problem 2 by relabeling G, a, d as A, s, e respectively.

Consider, as in Section 6.3, the rank-$3p$ lattice of tuples $(s, t, bt + qr - As) \in \mathcal{R}^3$ for $(s, t, r) \in \mathcal{R}^3$. This lattice has a short vector $(a, 1, d)$. As before, the analysis does not take into account the variations in the 2-norm of d.

The attack has a parameter k as before. The attack chooses a sublattice where k of the positions for s are forced to be 0 (as before), and where $p - 1$ of the positions for t are forced to be 0, forcing $t \in \mathbb{Z}$. These choices reduce the lattice rank from $3p$ to $2p - k + 1$. The probability analysis is as before, but rotations are used entirely for t and not for s, so increasing k is less useful. For the second case study, [4] takes $k = 0$.

The attack has a parameter m as before, and projects away $p - m$ positions as before: consider the map that inputs $(s, t, r) \in \mathcal{R} \times \mathbb{Z} \times \mathcal{R}$ where the specified k positions in s are 0, and outputs $(s, t, \mathrm{Extract}(bt + qr - As))$. The image of this map is a lattice of rank $p - k + 1 + m$ and determinant q^m. For the second case study, [4] takes $m = 590$, so the rank is 1352.

The attack rescales the lattice as before (in the s component; no scaling is applied to the t component), and uses BKZ as before, raising all the same questions as before. For the second case study, [4] takes $\beta = 532$, and, using the same 0.265β formula discussed above, assigns 140.98 bits of security to NTRU LPRime 4591^{761}. We point out that the same analysis, for the same case study, allows $\beta = 531$, reducing 140.98 to 140.715.

Compared to the attack against Streamlined NTRU Prime, this attack starts with a more complicated lattice. After forcing some entries to be 0 (destroying the $\mathcal{R}$-module structure as before), the attack ends up with a noticeably larger rank: 1 extra for t, and more for the change in k. On the other hand, the attack is targeting a noticeably smaller vector.

Ciphertext attacks. The key-recovery attack against NTRU LPRime can also be viewed as a (random-key) ciphertext attack against Streamlined NTRU Prime. However, the error weight for Streamlined NTRU Prime is larger (e.g., 286 instead of 250 for 4591^{761}), producing somewhat larger cost estimates from the same analysis.

One can similarly build a lattice for a ciphertext attack against NTRU LPRime. See the analysis of Problem 3 in Section 6.3. This is the distinction between "LWE n samples" and "LWE $2n$ samples" in [4]. There are some schemes where this produces different estimates, but for NTRU LPRime 4591^{761} the results in [4] are the same. This is not surprising given that [4] does not use all of the available samples for the NTRU LPRime 4591^{761} key-recovery attack—it projects down to a somewhat smaller rank. Having many more samples allows other attacks (see Section 6.10), but we have not seen attacks exploiting the number of samples in NTRU LPRime.

Larger issues. It is not clear that the issues listed above make many bits of difference in security estimates. The elephant in the room, however, is the large range of estimates appearing in [4] for the quantum cost of *the same attack*. There is also a non-quantum elephant in the room, namely the large range of estimates appearing in [4] for the non-quantum cost of the same attack.

The underlying problem is that [4] uses many conflicting formulas (from a range of sources) for the cost of BKZ-β in the final stage of the analysis. It acknowledges that "most of these estimates must be either too optimistic or pessimistic for the attacker". A closer look at the original sources of the formulas shows that each of the formulas has questionable accuracy, and possibly terrible accuracy, raising many problematic questions for security reviewers. See Section 6.9.

Another potentially very large problem is that known "hybrid attacks" are not included in [4], even though the NTRU literature indicates that the most effective attacks are hybrid attacks. See Sections 6.7 and 6.8.

6.5 Rotations

Let $\mathcal{F}$ be the space of possible keys f, i.e., the set of small $f \in \mathcal{R}$ of weight w. There is a natural equivalence relation on $\mathcal{F}$ in the context of Section 6.3: two elements $f, f' \in \mathcal{F}$ are equivalent if $f = uf'$ for some unit $u \in \mathcal{R}$. For example, f is equivalent to $-f$.

The analysis from [4] described in Section 6.4 implicitly assumes that the equivalence class of f has size exactly $2p$, i.e., size p modulo negation. The argument for this is that, in NTRU Classic, each element $f \in \mathcal{F}$ is equivalent to the rotations $x^i f \in \mathcal{F}$ for $i \in \{0, 1, \ldots, p-1\}$, and to $-x^i f$; presumably these rotations and negations are all distinct.

We point out a (presumably) small problem and a larger problem with this analysis. The small problem is that there could be unit multiples $uf \in \mathcal{F}$ where u does not have the form $\pm x^i$. There is an intuitive argument that this is rare: multiples such as $(1+x)f$ usually have much higher weight than f (and are thus more difficult for the lattice attack in Section 6.4 to find). However, we have not found literature quantifying this.

The larger problem is that [4] used the same analysis for NTRU Prime. The reason this is a problem is that, for NTRU Prime, the statement $x^i f \in \mathcal{F}$ is unjustified. The following numerical evidence suggests that the same assumption is overestimating the sizes of equivalence classes for NTRU Prime, and thus underestimating Streamlined NTRU Prime security.

Take $p = 761$ and $w = 286$. We generated 10000000 random $f \in \mathcal{F}$, and counted the number of small weight-w rotations $x^i f$ with $-5000 \le i \le 5000$ (never finding any with $|i| > 1000$). In 86.7% of the cases, there were 10 or fewer such rotations. In 99.2% of the cases, there were 20 or fewer such rotations. In 99.96% of the cases, there were 30 or fewer such rotations. In 99.999% of the cases, there were 40 or fewer such rotations.

To understand why rotations are less effective for NTRU Prime than for NTRU Classic, write the degree of a random f as $p - c$. Typically c is around p/w, since there are w terms in f. Multiplying f by $x, x^2, \ldots, x^{c-1}$ produces elements of $\mathcal{F}$, but multiplying f by x^c replaces

x^{p-c} with $x^p \bmod x^p - x - 1 = x + 1$, changing its weight and thus leaving $\mathcal{F}$. It is possible but rare for subsequent multiplications by x to reenter $\mathcal{F}$. Similarly, one expects only about p/w divisions by x to stay within $\mathcal{F}$, for a total of only about $2p/w$ equivalent keys, or p/w when negations are taken into account. We have constructed classes of pathological examples of f that allow considerably more rotations (including non-consecutive rotations), but none of these classes will be encountered by chance.

We have also considered various expansions of $\mathcal{F}$. We counted the number of rotations having 1-norm w, without the requirement of smallness; this changed the percentages to 84.1%, 98.9%, 99.93%, 99.997% respectively. We counted the number of rotations having 2-norm at most $1.1\sqrt{w}$; then there were 62.9% with 200 or fewer such rotations, 99.8% with 300 or fewer such rotations, and 99.995% with 350 or fewer such rotations. This $1.1\sqrt{w}$ corresponds to a weight increase of 21%; to put this into perspective, the increase in weight from NTRU LPRime 4591^{761} to Streamlined NTRU Prime 4591^{761} is only 14.4%.

In our round-1 submission we assumed an equivalence-class size not far below $2p$, with the caveat that this appeared to underestimate security. To simplify comparability to the analysis of [4], our calculations below assume an equivalence-class size of exactly $2p$, again with this caveat.

We comment that switching from Quotient NTRU to Product NTRU does not remove the need for this type of quantitative analysis. On the contrary, there are many options for taking high-probability sublattices of the natural $\mathcal{R}$-module for Problem 2 in Section 6.3. Perhaps sublattices of the Bai–Galbraith embedding are optimal, but this has not been proven.

6.6 Interlude: memory, parallelization, and the cost of sorting

Large-scale attacks are limited by (1) the amount of hardware that the attacker can afford and (2) the amount of time that the attacker spends running this hardware. Chips of total area A, running for time T, can perform a total of AT bit operations. This also costs energy proportional to AT, which does not pose a scaling problem: the chips are laid out in two dimensions, receiving energy (and dissipating heat) through the third dimension.

Now consider the cost of sorting M small items, where M is very large. Parallel "mesh" sorting algorithms such as the Schnorr–Shamir algorithm [88] sort M items in real time $M^{1/2+o(1)}$ using hardware area $M^{1+o(1)}$. This is optimal in realistic models of computation.

(Analogous sorting algorithms reduce the exponent 1/2 to 1/3 in abstract "three-dimensional mesh" models of computation. However, in more realistic models that account for the cost of energy transmission, the best exponent for known algorithms is 1/2.)

To summarize, sorting M items is as expensive as performing $M^{3/2+o(1)}$ bit operations. This is important for some attack subroutines described below, and for many other cryptanalytic computations.

Textbook presentations of sorting instead say that sorting M items costs just $M^{1+o(1)}$ "operations". Memory is treated as a minor secondary issue ("we assume that enough RAM is available"). This view deviates from reality in two critical ways.

First, a single "operation" that accesses memory has real cost growing with the amount of memory. Specifically, the time and energy grow linearly with the communication distance, which in turn grows as the square root of the amount of memory. Concretely, Intel reported in [45, page 9] that its energy cost of moving data is 11.20 pJ "per 5 mm" to move 8 bytes at 22nm. The same report indicates that this communication cost is "More difficult to scale down", whereas *computation* cost will "scale well with process and voltage". Other communication technologies similarly have cost scaling with distance.

Second, an attacker who can afford a huge amount of memory can instead use the same amount of hardware for a parallelizable low-memory computation, obtaining an almost linear parallelization speedup. Concretely, for the same cost as G gigabytes of memory, one can buy graphics-processing units (GPUs) with a total of approximately $32G$ parallel "cores", each of which is capable of carrying out a useful arithmetic operation every clock cycle. For large-scale attackers it is even more cost-effective to build special-purpose hardware, using an even larger number of smaller cores focused on the task at hand.

The bottom line is that simply counting "operations" does not correctly assign cost exponents to algorithms. Compare, for example, the following two types of algorithms:

- Algorithm 1 uses $A^{1/4}$ size-A sorting steps, a total of $A^{5/4+o(1)}$ "operations". It finishes in real time $A^{3/4+o(1)}$ on hardware area A, since each sorting step takes time $A^{1/2+o(1)}$.
- Algorithm 2 uses $A^{3/2}$ parallelizable low-memory operations. It finishes in real time $A^{1/2+o(1)}$ on hardware area A.

Algorithm 1 uses fewer "operations" than Algorithm 2, by a factor $A^{1/4+o(1)}$. However, Algorithm 2 takes less time than Algorithm 1, by a factor $A^{1/4+o(1)}$, using the same amount of hardware. Algorithm 2 is thus preferable for the attacker.

6.7 Meet-in-the-middle attack

As a warmup for hybrid attacks, we review Odlyzko's meet-in-the-middle attack [57, 55] on NTRU. We adapt the attack to the context of Streamlined NTRU Prime: for example, we account for the impact of changing $x^p - 1$ to $x^p - x - 1$, and using small f rather than $f = 1 + 3F$ with small F. For direct comparisons to the original NTRU cryptosystem, we assume that the weight w in Streamlined NTRU Prime is taken as $2t$, and that the original NTRU cryptosystem takes exactly t entries 1 and exactly t entries -1.

Odlyzko's attack works by splitting the space of possible keys $\mathcal{F}$ into two parts such that $\mathcal{F} = \mathcal{F}_1 \oplus \mathcal{F}_2$. Then in each loop of the algorithm partial keys are drawn from $\mathcal{F}_1$ and $\mathcal{F}_2$ until a collision function (defined in terms of the public key h) indicates that $f_1 \in \mathcal{F}_1$ and $f_2 \in \mathcal{F}_2$ have been found such that $f = f_1 + f_2$ is the private key.

The number of choices for f is $\binom{p}{t}\binom{p-t}{t}$ in original NTRU and $\binom{p}{2t}2^{2t}$ in Streamlined NTRU Prime. A first estimate is that the number of loops L in the algorithm is the square root of the number of choices of f. However, this estimate does not account for equivalent keys. The algorithm succeeds if it finds any key in the equivalence class of f defined in Section 6.5. If most equivalence classes have size approximately E then a better estimate is that the number of loops is divided by $\sqrt{E}$: i.e.,

$$L = \sqrt{\binom{p}{2t} 2^{2t}} \Big/ \sqrt{E} \tag{1}$$

for Streamlined NTRU Prime.

Analyses of NTRU normally take $E = 2p$, although this might overestimate security for NTRU Classic, and seems to underestimate security for NTRU Prime. See Section 6.5. One could modify the attack to use a larger set $\mathcal{F}$, but this seems to lose more than it gains.

Almost-collision probabilities. Odlyzko defines a collision function as follows. Given f_1, multiply by the given g/f, extract some coordinates of the product, and assign a bit to each coordinate by partitioning $\mathbb{Z}/q$ into two halves. Given f_2, multiply by $-g/f$ and similarly extract bits. The number of coordinates is chosen large enough that collisions on all these bits are unlikely to occur by chance.

There is, however, no guarantee that a collision will occur for the target f. If $f = f_1 + f_2$ then $(g/f)(f_1 + f_2) = g$ so each coordinate c_1 of $(g/f)f_1$ differs from the corresponding coordinate c_2 of $(-g/f)f_2$ by something small, namely the corresponding coordinate of g. Usually c_1 and c_2 will be in the same half of $\mathbb{Z}/q$, but it is possible for c_1 to be at one edge of one half while c_2 is in the other half, and considering more coordinates makes this type of failure more likely. Analyses vary in whether they calculate this failure probability, ignore this failure probability, or do more work in the algorithm to eliminate the failures.

Memory consumption. In each loop of the algorithm, t vectors of size p are added and their coefficients are reduced modulo q. A simple estimate of the attack cost is Lpt, where L is the number of loops.

There is, however, a problem here: finding the target collision requires L accesses to an array of size L, a tremendous amount of memory. This means that the L factor is a severe underestimate of the cost: the real cost of the computation is $L^{3/2+o(1)}$. See Section 6.6.

The pt factor could be an overestimate if the work inside the loop can be reduced. On the other hand, straightforward approaches such as caching sums of vectors exacerbate the memory problem.

Reducing memory consumption. There are well-known techniques to perform collision search using very little memory. However, for meet-in-the-middle attacks one needs "golden-collision search", which is more difficult than collision search.

General techniques for golden-collision search from [73] were adapted in [95] to Odlyzko's approximate-collision context. The bottom line—we disregard $o(1)$ here for simplicity—is that one can reduce the storage capacity by a factor s at the expense of increasing the number of loops by a factor $\sqrt{s}$.

For example, instead of L loops using L hardware (real cost $L^{3/2}$), one can perform $L^{9/8}$ loops using only $L^{3/4}$ hardware (again real cost $L^{3/2}$). This gives the user flexibility in adapting the meet-in-the-middle attack to the amount of hardware available.

Quantum search. Göpfert, van Vredendaal, and Wunderer [49], generalizing a suggestion by Schanck, replace Odlyzko's meet-in-the-middle attack with a quantum search through $\mathcal{F}$. (Even more generally, [49] replaces the hybrid attack—see Section 6.8—with a quantum hybrid attack.) This again takes L loops but has the advantage of using very little memory.

On the other hand, the L loops here are required to be serial. Limiting the real time to T requires $(L/T)^2$ parallel quantum searches, increasing the real cost to L^2/T. Estimating the cost as just L is underestimating security, as NIST has recognized in the context of quantum attacks against AES.

6.8 Hybrid attacks

The latest estimates for the cost of a hybrid attack are from Wunderer's thesis [97] in 2018, including various examples where the security estimates are far below the security estimates from [4]. We now review the analysis of hybrid attacks. Note that [96] and [97] point out various errors, underestimates, and overestimates in previous work.

Hybrid attacks were originated by Howgrave-Graham [55] in the context of Problem 1, and are featured in typical security analyses of Problem 1, but are ignored in most security analyses of Problems 2 and 3. This appears to be a historical accident, arising from exaggerations of the difference between Problem 1 and Problem 2.

Notation. We follow the standard convention of having matrices act as linear transformations on column vectors: M maps v to Mv. For conciseness, we abbreviate the column vector $\begin{pmatrix}1\\2\\3\end{pmatrix}$ as $(1, 2, 3)$, not to be confused with the row vector $\begin{pmatrix}1 & 2 & 3\end{pmatrix}$. Similarly, if v and w are column vectors, we write (v, w) for the column vector obtained by concatenating v with w.

Setup. The hybrid attack begins with the general problem of finding a short vector of the form $(qr + Cu, u)$, where r and u are integer vectors, given a matrix C. This includes all of the lattice problems from Section 6.3:

- Problem 1: Take the matrix C with $Cs = -As$.
- Problem 2: Take the matrix C with $C(s, t) = bt - As$.
- Problem 3: Take the matrix C with $C(s, t_1, t_2) = (b_1t_1 - A_1s, b_2t_2 - A_2s)$.

Optionally choose a sublattice where some of the coordinates of (r, u) are 0, as in Section 6.4; this produces a problem of the same form, with a shorter (r, u) and with C shrunk accordingly. Also, optionally project away some of the output coordinates, as in Section 6.4; this again produces a problem of the same form.

Basis reduction. The hybrid attack has two main computational phases. The first phase is lattice-basis reduction, specifically BKZ-β for some block size β.

In the lattice attack in Section 6.4, β is chosen large enough that the shortest basis vector is (hopefully) the target vector modulo negation. The hybrid attack tries to do better by allowing β to be smaller (making BKZ-β faster), and using the second phase below to compensate for the longer basis. The optimal balance between the two phases depends on the cost of BKZ-β; underestimating (or overestimating) the cost can make the hybrid attack seem less (or more) useful than it actually is.

Specifically, the hybrid attack has a parameter σ, an integer between 0 and the length of u. The attack views u as having two parts (t, s), where s has length σ; the problem is now to find a short vector of the form $(qr + C_1t + C_2s, t, s)$. Say C_1 is an $a \times b$ matrix, so $b + \sigma$ is the length of u.

The attack applies basis reduction to the matrix

$$\begin{pmatrix} qI_a & C_1 \\ 0 & I_b \end{pmatrix}.$$

The output of basis reduction is a relatively short matrix R generating the same lattice: i.e., the lattice of $(qr + C_1t, t)$ for all (r, t) is exactly the lattice of Rz for all z. The original lattice of $(qr + C_1t + C_2s, t, s)$ is the lattice of $(Rz + Ss, s)$ where S has the form $(C_2, 0)$; i.e., the matrices

$$\begin{pmatrix} qI_a & C_1 & C_2 \\ 0 & I_b & 0 \\ 0 & 0 & I_\sigma \end{pmatrix}, \qquad \begin{pmatrix} R & S \\ 0 & I_\sigma \end{pmatrix}$$

generate the same lattice.

As an extreme case, if $\sigma = 0$, then this is the same basis reduction as in Section 6.4, and the second phase is skipped. At the opposite extreme, if σ is the full length of u, then the first matrix above is simply qI_a, reduction outputs $R = qI_a$, and the second phase of the hybrid attack is the same as the meet-in-the-middle attack described in Section 6.7.

For intermediate values of σ, the analysis quantifies the effect of basis reduction as follows. The lattice has rank $d = a + b$ and determinant q^a. Define $\delta = (\beta(\pi\beta)^{1/\beta}/(2\pi e))^{1/(2(\beta-1))}$ as in Section 6.4, and define $k = \min\{d, \lfloor (b/\log_q \delta)^{1/2} \rfloor\}$. The analysis then states that the d output Gram–Schmidt vectors have approximately the following lengths: $d - k$ copies of q, followed by

$$q^{1-b/k}\delta^{k-1}, q^{1-b/k}\delta^{k-3}, \ldots, q^{1-b/k}\delta^{-(k-1)}.$$

The product of these d lengths is $q^{d-k}(q^{1-b/k})^k = q^{d-b} = q^a$ as required. The general shape of these lengths, with $d - k$ orthogonal vectors (not affected by reduction) followed by a geometric drop, is based on many experiments; on the other hand, (1) the experiments do not support the formula for δ (see Section 6.4), and (2) the experiments show some systematic deviations from the geometric-series assumption. Also, these formulas do not account for the possibility of rescaling smaller input positions; taking only scale 1 can overestimate security.

A standard speedup here for $k < d$, given that the analysis predicts that the first $d - k$ vectors will be unchanged, is to not bother reducing those $d - k$ vectors. This allows basis reduction to work with a lattice of rank k instead of d. This speedup is not visible in BKZ-β cost models that use only β and that ignore the lattice rank; some of the cost models in Section 6.9 are of this type.

Search. The second phase of the hybrid attack searches for a short vector of the form $(Rz + Ss, s)$. Recall that s has length σ. The matrix R is produced by the basis reduction in the first phase.

The attack chooses a set of possibilities for s to search. Weight restrictions make the search faster. We return below to the optimal choice of this set.

We will see in a moment how the attack tries to expand a candidate s into a short vector $(Rz + Ss, s)$. Having the target s in the set searched is not enough to guarantee the success of the attack: even if $(Rz + Ss, s)$ is short, the attack might instead find $(Rz' + Ss, s)$ for $z' \neq z$ and thus discard s. To evaluate the chance that the attack succeeds, the analysis compares (1) the size of the short vector $(Rz + Ss, s)$ to (2) how far R has been reduced. Having the attack succeed relies on the first phase to have reduced the basis sufficiently, which in turn needs β to be large enough.

To check a candidate s, the hybrid attack checks whether $Ss - \mathrm{Near}_R(Ss)$ is small. Here $\mathrm{Near}_R(Ss)$ is defined as the output of Babai's nearest-plane algorithm starting from the matrix R and the point Ss. This output is in the lattice generated by R; in other words, $Ss - \mathrm{Near}_R(Ss)$ is a reduction of Ss modulo R. The algorithm runs in polynomial time.

If the target $Rz+Ss$ is sufficiently small, compared to how well reduced R is, then $\mathrm{Near}(Ss) = -Rz$, and the attack finds $Rz+Ss$ as desired. The probability that this occurs is heuristically estimated in [97, Section 5.3.2, formula (5.6)] as

$$\prod_{1\le i\le d}\left(1 - \frac{2}{B(\frac{d-1}{2},\frac{1}{2})}\int_{\min\{r_i,1\}}^{1}(1-t^2)^{(d-3)/2}\,dt\right)$$

where r_i is the ith Gram–Schmidt length divided by $2|Rz + Ss|_2$, and

$$B\left(\frac{d-1}{2},\frac{1}{2}\right) = \int_0^1 t^{(d-3)/2}(1-t)^{-1/2}\,dt = 2\int_0^1 (1-t^2)^{(d-3)/2}\,dt.$$

More experimental evidence is needed to test the accuracy of this estimate.

(Wunderer mentions that the integrals here can be computed by, e.g., the Sage computer-algebra system. We comment that Sage has a built-in function `beta` for B, and, more to the point, a built-in function for each factor P_i in the product $\prod_{1\le i\le d} P_i$ above, given r_i: after defining `T = RealDistribution('beta',((d-1)/2,1/2))` one can compute P_i as `1-T.cum_distribution_function(1-ri^2)`.)

The previous literature, instead of asking whether the algorithm finds $Rz + Ss$, usually asks whether the target is small enough to apply a particular theorem *guaranteeing* that the algorithm finds $Rz+Ss$. This can overestimate or underestimate security. Overestimate: the algorithm can work even when the theorem does not guarantee that it works. Underestimate: there is an error in the usual applications of the theorem, as pointed out in [97, pages 70–71].

Choice of the set to search. If the target s is extracting a limited number of positions from a weight-w vector then s probably has weight below w. The set of s searched in, e.g., [96] is the set of s having the single most likely weight, and, within this, the single most likely split between the number of 1's and the number of -1's. (The set is further restricted in the meet-in-the-middle context.)

However, for typical parameters, there are many choices of weights and splits that are almost as likely. Searching all of those together gains performance, since the work in the first phase is shared. This was pointed out in [97, Section 7.3] for the quantum hybrid attack, but it also applies to the non-quantum case.

This does not mean that one should search *all* possibilities. Searching very unlikely weights is less efficient than restarting the algorithm with new random choices of positions. To optimize the choice of the set of s, one needs to take into account how the choice affects the cost of the search (which is one component of the total cost of the algorithm) and the probability that the algorithm succeeds.

The cost details depend on whether one uses a meet-in-the-middle attack or a quantum search; see below. Regarding probabilities, the main question analyzed in [97, Section 7.3] is the chance that s is in the set.

Choose an interval $I \subseteq \{0, 1, \dots, \sigma\}$, and take the set $\bigcup_{i \in I} S_i$, where S_i is the set of small weight-i length-σ vectors s. More generally, [97, Section 7.3] considers removing part of S_ℓ from $\bigcup_{i \in I} S_i$, where ℓ is the least likely $i \in I$.

Assume again that s is obtained by extracting σ positions from a uniform random small weight-w length-p vector f. There are $\binom{\sigma}{i}2^i$ elements of S_i, and each element is compatible with $\binom{p-\sigma}{w-i}2^{w-i}$ choices of f, out of the $\binom{p}{w}2^w$ total choices of f. Hence $s \in \bigcup_{i \in I} S_i$ with probability $\sum_{i \in I} \binom{\sigma}{i}\binom{p-\sigma}{w-i}/\binom{p}{w}$.

We point out that this analysis ignores the following correlation: if s has lower/higher weight then the rest of the vector has higher/lower weight and is thus less/more likely to be found by the Near computation. It is not clear whether ignoring this correlation overestimates or underestimates security.

Meet-in-the-middle. To save time, the second phase of the hybrid attack actually identifies s through a collision search.

Choose a partition of the σ positions in s into two halves. Write s as $s_1 + s_2$ where s_1 is supported on the first half and s_2 is supported on the second half. Choose a set of possibilities for s_1 to search, and a set of possibilities for s_2 to search, so that the sums $s_1 + s_2$ cover many of the most likely choices for s.

Above we considered the chance that $\text{Near}(Ss) = -Rz$ for the short target $Rz + Ss$. If this occurs then it is not unreasonable to also hope that $\text{Near}(Ss_1) + \text{Near}(Ss_2) = \text{Near}(Ss)$, i.e., that $Ss_1 - \text{Near}(Ss_1)$ and $Ss_2 - \text{Near}(Ss_2)$ have sum $Ss - \text{Near}(Ss) = Rz + Ss$, so the coordinates of $Ss_1 - \mathrm{Near}(Ss_1)$ and $\mathrm{Near}(Ss_2) - Ss_2$ are close, producing a collision of extracted bits as in Section 6.7.

The meet-in-the-middle approach is generally advertised as reducing the number of Near computations to its square root. However, the memory-consumption issues analyzed in Section 6.7 are applicable here, so the real cost improvement is only a fourth root. Furthermore, the decomposition of s into s_1 and s_2 reduces the success probability of the algorithm, for two reasons:

- Collisions are not guaranteed to occur. On the contrary, the chance that $\mathrm{Near}(Ss_1) + \mathrm{Near}(Ss_2) = \mathrm{Near}(Ss)$ can be much smaller than 1. The latest heuristic analysis is in [97], but again more experimental evidence is needed. Most of the literature on hybrid attacks treats the collision chance as 1, underestimating security.
- Searching only the most likely weights for s_1 and the most likely weights for s_2 is more restrictive than searching only the most likely weights for s.

A reasonable (but perhaps not optimal) choice of sets to search is the following. Say s_1 has length σ_1 and s_2 has length σ_2, with $\sigma = \sigma_1 + \sigma_2$. Choose an interval $I_1 \subseteq \{0, 1, \ldots, \sigma_1\}$, and search all $s_1 \in \bigcup_{i \in I_1} S_{i,1}$, where $S_{i,1}$ is the set of small weight-i length-σ_1 vectors s_1. Similarly choose an interval $I_2 \subseteq \{0, 1, \ldots, \sigma_2\}$, and search all $s_2 \in \bigcup_{i \in I_2} S_{i,2}$, where $S_{i,2}$ is the set of small weight-i length-σ_2 vectors s_2.

The total size of the s_1 search space and the s_2 search space is $\sum_{i \in I_1} 2^{i_1} \binom{\sigma_1}{i_1} + \sum_{i \in I_2} 2^{i_2} \binom{\sigma_2}{i_2}$. The chance that (s_1, s_2) is in the product of search spaces, when s_1 and s_2 are obtained by extracting disjoint positions from a uniform random small weight-w length-p vector f, is $\sum_{i_1 \in I_1} \sum_{i_2 \in I_2} \binom{\sigma_1}{i_1} \binom{\sigma_2}{i_2} \binom{p-\sigma_1-\sigma_2}{w-i_1-i_2} / \binom{p}{w}$.

As an experiment, we took $p = 761$, $w = 286$, $\sigma_1 = 199$, and $\sigma_2 = 200$, and searched (not comprehensively) for the choices of (I_1, I_2) that Pareto-optimize (1) the total size of the s_1 and s_2 search spaces and (2) the chance that (s_1, s_2) is in the product of the search spaces. We found, e.g., $(2^{293.575}, 2^{-0.000320566})$ for $I_1 = \{0, \ldots, 97\}$ and $I_2 = \{0, \ldots, 96\}$; $(2^{268.079}, 2^{-1.02933})$ for $I_1 = I_2 = \{0, \ldots, 98\}$; $(2^{262.99}, 2^{-2.11644})$ for $I_1 = I_2 = \{0, \ldots, 75\}$; $(2^{256.791}, 2^{-4.26975})$ for $I_1 = \{0, \ldots, 72\}$ and $I_2 = \{0, \ldots, 71\}$; $(2^{249.187}, 2^{-8.14923})$ for $I_1 = \{0, \ldots, 68\}$ and $I_2 = \{0, \ldots, 67\}$; and many intermediate examples. We also explored a slightly more general choice of sets, taking only a fraction of S_i for the maximum $i \in I_2$; this did not noticeably change the shape of the Pareto-optimal curve, although it did fill in even more intermediate points. We did not find any Pareto-optimal examples with the maxima of I_1 and I_2 separated by more than 1.

The literature usually treats each invocation of the Near algorithm as having cost 1, underestimating security. Wunderer quotes [52] as claiming just $d^2/2^{1.06}$ bit operations for Near in rank d, but all known algorithms are slower than this.

Quantum search. An alternative, from [49], is a quantum search through the set of s. If p_s is the probability of s then inside the quantum search one should choose each s with probability proportional to $p_s^{2/3}$, and run the search for $(\sum_s p_s^{2/3})^{3/2}$ iterations; see [49] for an explanation of the exponents here.

Like the quantum search in Section 6.7, this fits in low memory, and it eliminates issues arising from the decomposition of s into $s_1 + s_2$. However, as in Section 6.7, the cost increases if the total attack time is limited.

6.9 The cost of BKZ

The first version of BKZ, introduced by Schnorr and Euchner [87], works as follows. BKZ solves SVP on the lattice generated by the first β basis vectors $b_1, \ldots, b_\beta$, and updates the basis to include the short vector found by SVP. BKZ then does the same for $b_2, \ldots, b_{\beta+1}$, and then $b_3, \ldots, b_{\beta+2}$, and so on through all of the basis vectors (with block sizes decreasing below β for the final vectors); this completes one BKZ "tour". BKZ repeats these tours until the basis stops changing.

BKZ solves SVP on the lattice generated by $b_1, \ldots, b_\beta$ using "enumeration", as in [79, 47, 59]. Enumeration is a pruned combinatorial search for integer vectors $(c_1, \ldots, c_\beta)$ such that $c_1 b_1 + \cdots + c_\beta b_\beta$ is short (specifically, shorter than the shortest vector found so far). The pruning uses a formula for the maximum size of c_1 given that $c_1 b_1 + \cdots + c_\beta b_\beta$ is short; then, for each c_1, a formula for the maximum size of c_2 given that $c_1 b_1 + \cdots + c_\beta b_\beta$ is short; etc. These formulas produce smaller outputs and thus faster enumeration when $b_1, \ldots, b_\beta$ are closer to orthogonal, so BKZ calls LLL as a preprocessing step before each SVP computation.

Today "BKZ" refers to a more complicated algorithm with many adjustable parameters:

- For large β, each use of LLL is replaced by more expensive preprocessing, typically recursive calls to BKZ-β' for some $\beta' < \beta$. This takes time but speeds up enumeration.
- Instead of searching every coefficient c_i that could *possibly* lead to a shorter vector, the attacker searches a more limited range of coefficients c_i that are *most likely* to lead to a sufficiently short vector. This reduces the success probability of the SVP step, but compensates with a larger savings in time.
- Bases are extended to larger generating sets. The attacker accumulates a database of short combinations of the original basis vectors, and tries to reduce each new vector by subtracting nearby elements in the database, rather than merely by subtracting multiples of the original basis vectors. Various specific strategies for doing this are called "sieving" algorithms.
- The number of tours is limited, typically to 8. This limit is based on graphs showing that, in small-scale experiments, most of the improvements in output quality are from the earliest tours; see, e.g., [51, Figure 1]. However, the usual calculations of β are sensitive to the exact output quality, so a small loss of output quality can produce a large slowdown from a larger value of β. We have not found quantitative analyses in the literature claiming that 8 is optimal; we have not even found analyses claiming that the optimal number of tours is constant as β increases.

See, e.g., [36] for a 2011 version of BKZ with some of these features ("BKZ 2.0"); [16, 63] for more recent sieving methods; [10] for more recent pruning methods; and [5] for further variants. See also [65] for quantum versions of some sieving algorithms, and [11] for a quantum enumeration algorithm.

BKZ cost asymptotics. There is a consensus that the version of BKZ in [87] costs $2^{O(\beta^2)}$ if the number of tours is limited and if β is not too small compared to the lattice dimension; that recursive preprocessing, first used in [59], reduces $O(\beta^2)$ to $O(\beta \log \beta)$; and that sieving reduces $O(\beta \log \beta)$ to $O(\beta)$.

There are some proofs of more precise asymptotic upper bounds for the number of "operations" used for some SVP algorithms. For example, an algorithm from [81] solves dimension-β SVP using $2^{2.465\ldots\beta+o(\beta)}$ "operations", and an algorithm from [2] solves dimension-β SVP using $2^{\beta+o(\beta)}$ "operations". However, these upper bounds are generally believed to be vastly larger than the number of "operations" in the best algorithms known.

There are also heuristic analyses that try to produce more accurate asymptotics for various algorithms. (There are also many other aspects of lattice attacks relying on heuristic analyses; see, e.g., Section 6.4.) For example, the analysis of [72] states that a particular sieving algorithm uses $2^{0.415\ldots\beta+o(\beta)}$ "operations". A new wave of sieving algorithms starting in 2013 culminated in the analysis of [16], concluding that the algorithm of [16] takes just $2^{0.292\ldots\beta+o(\beta)}$ "operations". The constants $0.415\ldots$ and $0.292\ldots$ here are $\log_2(4/3)$ and $\log_2\sqrt{3/2}$ respectively.

Memory consumption. Enumeration fits into very little memory even for large β. Kuo, Schneider, Dagdelen, Reichelt, Buchmann, Cheng, and Yang [62] showed that enumeration parallelizes effectively within and across GPUs.

Sieving algorithms in the literature typically use a massive database, size $2^{0.2075\ldots\beta+o(\beta)}$ (and often even larger). This multiplies the real cost of the algorithms by $2^{0.1037\ldots\beta+o(\beta)}$; see Section 6.6.

It is sometimes claimed that sieving algorithms *must* use $2^{0.2075\ldots\beta+o(\beta)}$ space, so sieving algorithms necessarily cost this much, so SVP algorithms necessarily cost this much. The first step in this argument was disproven by "tuple lattice sieving", which achieved a smaller space exponent. Tuple lattice sieving has a larger time exponent and is ignored in [4].

How concrete cost formulas for BKZ are produced. The asymptotic formula $0.292\ldots\beta + o(\beta)$ means β times something that *converges* to $0.292\ldots$ as $\beta \to \infty$. The o symbol, by definition, says nothing about any particular value of β. For example, $0.292\ldots\beta + o(\beta)$ could be 1000 for $\beta = 500$, even though $0.292\ldots \cdot 500$ is under 147.

Readers often think—or hope—that $o(\beta)$ can be ignored, i.e., replaced with 0. However, ignoring $o()$ in asymptotics can lead to misstating security levels by an unlimited number of bits: e.g., claiming 2^{100} "operations" or 2^{300} "operations" for an attack that actually uses 2^{200} "operations". Furthermore, small-scale experiments show deviations, often large deviations, from the resulting formulas. Consider, e.g., [16, Figure 3], which reports a best fit of $2^{0.387\beta-15}$ for its fastest sieving experiments.

There are two common responses in the literature on lattice-based cryptography to the problem of a cost formula that does not match experiments. One standard response—when the experiments are consistently slower, as often happens—is that the cost formula ignored "polynomial factors" that are visible in the experiments. This leaves the security reviewer with many unanswered questions: How large could these factors become when attacks are scaled up? Why should we think that the underestimate is only polynomial? Could the cost formula *overestimate* security for larger sizes, for example by a subexponential factor that is not visible in small-scale experiments?

Another standard response is to complicate the formula in a way that

- is consistent with the known asymptotics and
- leaves more free variables to improve the fit of the formula to the experiments.

For example, the formula $0.187\beta\log_2\beta - 1.019\beta + 16.1$, one of the cost formulas used in [4], was obtained as follows. A heuristic asymptotic analysis states $\Theta(\beta\log\beta)$ for some enumeration algorithms. Experiments are a poor fit to $a\beta\log\beta$, so instead hypothesize a formula of the form $a\beta\log_2\beta + b\beta + c$. Unsurprisingly, the extra variables allow a better fit to the experiments. This again leaves the security reviewer with many unanswered questions: Why should we think that the formula accurately predicts the cost of much larger computations? What happens if the actual cost follows a formula with a different shape?

Four of the conflicting formulas used in [4] are 0.292β ("Core-SVP"), $0.292\beta + 16.4$, $0.292\beta + \log_2\beta$, and $0.292\beta + 16.4 + \log_2(8d)$. The first formula simply takes the $0.292\ldots\beta + o(\beta)$ asymptotic and imagines that $o(\beta)$ is 0 (and rounds to 0.292). The second and third formulas imagine that there is a constant-factor overhead, or a β-factor overhead. The fourth formula is like the second formula but includes 8 tours, each with d SVP calls. Another four formulas replace 0.292 with 0.265 for the quantum case.

Another two formulas are 0.368β for the non-quantum case and 0.2975β for the quantum case. These are the "operation" exponents for two "minimum-space" algorithms that have space exponents 0.208.

The other four formulas use enumeration rather than sieving. Two of these formulas, $0.187\beta\log_2\beta - 1.019\beta + 16.1$ as mentioned above and $0.000784\beta^2 + 0.366\beta - 0.9 + \log_2(8d)$ (from [53]), were obtained by fitting two different shapes to the same data. These formulas are (as pointed out by Schanck) within 10 bits of each other for all costs below 2^{256}, and it is not clear which formula has larger errors in this range (even assuming accuracy of the original data). The third formula, $0.125\beta\log_2\beta - 0.755\beta + 2.25$, was introduced as a "lower bound" on a class of enumeration algorithms. The last formula, $(0.187\beta\log_2\beta - 1.019\beta + 16.1)/2$ for the quantum case, starts from the first formula and hypothesizes an exact 1/2 factor from quantum search; [11] justifies a $1/2 + o(1)$ factor, but does not justify exactly 1/2. Note that quantum searches normally have considerable overhead for their inner loops.

The cutoff between sieving and enumeration. Imagine, in the quantum case, that the $(0.187\beta\log_2\beta - 1.019\beta + 16.1)/2$ formula above accurately predicts enumeration cost, and that the $0.265\beta + 16.4$ formula above accurately predicts sieving cost. Then sieving and enumeration both cost about 2^{114} for $\beta \approx 368$; enumeration is faster for smaller β, and sieving is faster for larger β.

We emphasize that this 368 cutoff is extremely sensitive to the exact numbers in the formulas, since $\log_2 \beta$ grows very slowly with β. For example, imagine that another algorithm improves the secondary enumeration term from $-1.019\beta/2$ to $-1.119\beta/2$. This makes the cutoff leap from cost 2^{114} at $\beta \approx 368$ to cost 2^{151} at $\beta \approx 509$.

For comparison, [48] estimated that its pruning improvements saved a factor $2^{0.25\beta}$ compared to earlier pruning methods. The $(0.187\beta \log_2 \beta - 1.019\beta + 16.1)/2$ formula already takes account (indirectly) of this improvement, but it does not take account of more recent improvements such as [10].

As a more extreme example, take the original enumeration formula, but add 0.104β to the sieving formula to account for the real cost of memory. This produces an even more dramatic change in the cutoff, up to 2^{225} operations at $\beta \approx 610$.

Improvements in sieving could swing the cutoff the other way. The recent paper [5] claims that this has happened in the non-quantum case—that its latest sieving algorithm is "400 times faster" than enumeration in a small-scale experiment, which surely means that sieving has an even larger advantage for cryptographic sizes. However, this experiment used 246 gigabytes of RAM. For the same hardware cost, an attacker can purchase many thousands of GPU "cores", massively parallelizing enumeration as explained in [62]. The comparison in [5] ignores this competitor (and also ignores more recent enumeration algorithms such as [10]). Larger examples will give the attacker larger benefits from the massive parallelizability of enumeration, and will make the costs hidden in [5] more obvious.

Perhaps the approach of [5] can outperform enumeration algorithms for an attack that fits within a reasonable security target, or perhaps not; more analysis is required. To quantify the cutoff between the known algorithms, a security reviewer needs to investigate both classes of algorithms in detail, taking care to avoid oversimplified asymptotics, unjustified extrapolations, and unrealistic cost models. Future improvements in sieving or in enumeration will require a reevaluation of the cutoff.

6.10 Algebraic attacks

The attack strategy of Ding [43], Arora–Ge [12], and Albrecht–Cid–Faugère–Fitzpatrick–Perret [3] takes subexponential time to break dimension-n LWE with noise width $o(\sqrt{n})$, and polynomial time to break LWE with constant noise width. However, these attacks require many LWE samples, whereas typical cryptosystems in the NTRU family provide far less data to the attacker. When these attacks are adapted to cryptosystems that provide only (say) $2n$ samples, they end up taking more than $2^{0.5n}$ time, even when the noise is limited to $\{0, 1\}$. See generally [3, Theorem 7] and [67, Case Study 1].

6.11 A selection of estimates

There are many proposals of lattice-based cryptosystems, and of parameter sets for those cryptosystems. What are the pre-quantum and post-quantum security levels of each of these parameter sets?

The literature contains many different strategies to compute answers to this question. Given the issues described in Sections 6.4 through 6.9, it seems reasonably clear that all of these strategies are wrong, but that some strategies are more wrong than others. This creates serious problems in using the results: for example, deciding what parameter sizes are needed, or evaluating the merits of more specific design decisions such as the choice of weight.

It is important to put a warning note next to estimates that allow free access to arbitrarily large amounts of memory; see Section 6.6. It is also important to put a warning note next to estimates that disregard hybrid attacks. These issues interact: it is harder to see the value of hybrid attacks if sieving is assumed to handle BKZ-β cheaply, and it is harder to see the true cost of sieving if memory is incorrectly assumed to be free.

With all of the above caveats in mind, we consider four options for cost estimates: operation exponent 0.292β or $0.18728\beta \log_2 \beta - 1.0192\beta + 16.1$; real cost of memory or free memory; hybrid attacks allowed or not; quantum speedups allowed or not. Three combinations turn out to be redundant (for known algorithms), leaving the following 13 cost estimates that we use in Section 7:

- Non-quantum 0.292β, non-hybrid, free memory. Many post-quantum proposals have reported results from this estimate, and NIST has asked for a comparison using the same estimate.
- Non-quantum 0.292β, hybrid, free memory. This is the same cost model but corrects the omission of hybrid attacks.
- Non-quantum 0.396β, non-hybrid, real cost of memory. This corrects the cost model. The 0.396β assumes exponent 0.292β for the number of memory accesses and exponent 0.208β for the amount of memory.
- Non-quantum 0.396β, hybrid, real cost of memory.
- Non-quantum $0.18728\beta \log_2 \beta - 1.0192\beta + 16.1$, non-hybrid, real cost of memory. This has the advantage of naturally being a low-memory algorithm.
- Non-quantum $0.18728\beta \log_2 \beta - 1.0192\beta + 16.1$, hybrid, free memory.
- Non-quantum $0.18728\beta \log_2 \beta - 1.0192\beta + 16.1$, hybrid, real cost of memory.
- Quantum 0.265β, non-hybrid, free memory. The improvement from 0.292 to 0.265 relies on quantum random access to memory, and there is no known speedup in a realistic model.
- Quantum 0.265β, hybrid, free memory.
- Quantum 0.396β, hybrid, real cost of memory.
- Quantum $(0.18728\beta \log_2 \beta - 1.0192\beta + 16.1)/2$, non-hybrid, real cost of memory. This

is another low-memory algorithm.

- Quantum $(0.18728\beta\log_2\beta - 1.0192\beta + 16.1)/2$, hybrid, free memory.
- Quantum $(0.18728\beta\log_2\beta - 1.0192\beta + 16.1)/2$, hybrid, real cost of memory.

Hybrid attack estimates are more difficult to compute than non-hybrid attack estimates: there are more parameters (especially in the non-quantum case), and the formulas are more complicated. We use the heuristic beta-distribution formula from [97] to compute block sizes, but for the moment we assume collision probability 1 (underestimating security) rather than using the collision-probability formula from [97].

To simplify near-future comparisons, we use only "Core" estimates (one SVP call), even though these underestimate security. We also exclude $0.292\beta + 16.4$, $0.292\beta + \log_2\beta$, etc.: these minor variants distract attention from potentially much larger problems with the main term.

We also exclude the formula $0.000784\beta^2 + \cdots$ from [53]. As noted above, this formula is close to the $0.18728\beta\log_2\beta + \cdots$ formula throughout the entire range of cryptographic interest (up through 256 bits of security), and it is not clear which formula has larger errors in this range; including both formulas does not seem to add any value. The quadratic formula was pushed far beyond the cryptographic range in [4] (primarily because [4] ignored hybrid attacks, which use smaller block sizes), turning the growth of the formula into a distraction for the reader, so we opt for the $0.18728\beta\log_2\beta + \cdots$ formula.

All of the quantum speedups are Grover searches, or generalizations such as amplitude amplification. The speedup factor is thus limited by the real time available to the attacker, compared to the time required for each iteration of the search. More precise cost estimates require evaluating the number of qubits, number of qubit operations, and latency for each iteration, along with the general costs of quantum computation.

7 Expected strength (2.B.4) in general

Expanded and updated for round 2.

7.1 Security definitions

Streamlined NTRU Prime and NTRU LPRime are both designed for IND-CCA2 security.

It is possible to save time, especially in decapsulation, by abandoning protection against chosen-ciphertext attacks. This submission intentionally avoids providing any such options. It is not clear that the speedup is relevant to users (see Section 5.4), whereas there is a clear risk that providing options vulnerable to chosen-ciphertext attacks will lead to deployment of those options in scenarios that turn out to allow such attacks.

Internals. Streamlined NTRU Prime Core, NTRU LPRime Core, and NTRU LPRime

Expand are designed for OW-Passive security,[5] i.e., one-wayness against passive attacks. OW-Passive security means that it is difficult to find an input (generated uniformly at random) given a public key (generated by the key-generation procedure) and the corresponding ciphertext (generated by encrypting the input).

We emphasize that OW-Passive security is not adequate for applications. Streamlined NTRU Prime Core, NTRU LPRime Core, and NTRU LPRime Expand are internal modules that are not meant to be exposed to users. However, these modules have the advantage of being relatively simple targets for cryptanalysis. These are also useful targets to consider, since OW-Passive security of Streamlined NTRU Prime Core and NTRU LPRime Expand tightly implies security of the complete KEMs against ROM IND-CCA2 attacks.

Malleability. Lattice-based encryption has various symmetries, analogous to well-known ECC symmetries. For example, if $c = \mathsf{Round}(hr)$ is a Rounded Quotient NTRU ciphertext for r under public key h, then $-c = \mathsf{Round}(-hr)$ is a Rounded Quotient NTRU ciphertext for r under public key $-h$. Furthermore, as in ECC, it is natural for software to decode multiple strings to the same ciphertext and to decode multiple strings to the same public key: e.g., allowing q as a synonym for 0.

Shoup [89] refers to equivalent encodings of ciphertexts as "benign malleability". Benign malleability of ciphertexts violates IND-CCA2 if ciphertexts are defined as strings, but does not violate IND-CCA2 if ciphertexts are defined as the mathematical objects produced by decoding. Benign malleability of public keys does not violate IND-CCA2 either way, and symmetries such as $(c, h) \leftrightarrow (-c, -h)$ do not violate IND-CCA2 either way.

The standard argument for allowing equivalent encodings and symmetries is that (1) it is easy to build protocols where these are not problems, (2) ignoring these issues simplifies the primitives, and (3) the literature has not settled upon a definition of what primitives should be trying to accomplish, beyond IND-CCA2. The standard argument for prohibiting equivalent encodings and symmetries—even without a clear definition of the goal—is that (1) people sometimes build protocols where these *are* problems and (2) primitives can protect at least some of these protocols without much extra cost.

To ensure IND-CCA2 security with ciphertexts defined as strings, we reencode the new ciphertext produced during decapsulation and compare to the original ciphertext. However, this reencoding step does nothing to prevent public-key malleability. We also note that there is a risk of implementations skipping the reencoding step and merely comparing unencoded ciphertexts; test suites that include invalid ciphertexts will not necessarily include enough forms of malleability.[6]

After considering the costs (in performance and in complexity), we have decided to hash the ciphertext string and the public-key string into our session key. This hashing enforces

[5]OW-Passive security is typically called "OW-CPA" security, but there are no attacker-chosen plaintexts in the definition.

[6]A single test of a modified encoding is enough *if* the encoding is decoded to the original ciphertext: skipping the reencoding step will produce the original session key, failing the test. However, if the modification changes the decoder output (e.g., if q is not reduced to 0), then skipping the reencoding step will pass the test, and further tests are required.

unique encodings: any modification to either string will produce an unrelated session key. This hashing should also rule out other symmetries. We caution reviewers, however, that the security consequences of this hashing need to be formalized and proven, and that it is safer for protocols to rely on simpler promises from primitives.

Multi-target attacks. We recommend handling multi-target attacks by aiming for a very high single-target security level, and then relying on the fact that T-target attacks gain at most a factor T. This approach is simple and effective, and is not much more expensive than merely stopping single-target attacks.

A different approach is to try to eliminate the gain from T-target attacks—or, if this is not possible, to at least reduce the gain below a factor T. This approach complicates the entire attack surface. For every single-target attack, the reviewer is forced to ask how much more effective multi-target attacks can be. Occasionally the literature convincingly answers this question, but usually it does not.

It is *conceivable* that our hashing of the public key (see above) makes multi-target attacks more difficult than they otherwise would be. For example, consider an attacker who guesses many inputs, computes the corresponding 256-bit confirmations, and scans many ciphertexts, searching for these confirmations. The public key is hashed into the confirmation, so this attack is limited to ciphertexts for a particular key, limiting the total number of targets.[7] On the other hand, this attack was already so expensive as to be (1) uninteresting and (2) unlikely to be the most effective multi-target attack against the complete system.

It is easy to argue that various 256-bit quantities are overkill, and that we could safely truncate to smaller sizes. However, the maximum possible savings are only a small fraction of our key sizes and ciphertext sizes. Consistently using 256 bits simplifies security review.

7.2 Quantitative security and rationale

Our general strategy for evaluating and selecting parameter sets is as follows:

- Take a broad initial collection of parameter sets.
- Compute the estimated performance and estimated security for each parameter set.
- Restrict attention to parameter sets offering nearly optimal tradeoffs between estimated performance and estimated security.
- Choose a performance requirement, and restrict attention to parameter sets meeting this requirement.
- Take the parameter set with the highest estimated security.

As the performance requirement varies, the parameter sets that can be produced by this

[7] The attack still benefits from the number of ciphertexts encrypted to one key. For comparison, encrypting each guessed input and comparing to all the ciphertexts has the same benefit, but the encryption is somewhat more expensive than hashing. The communication costs in either version of the attack can be reduced; see [73].

procedure are *not* all possible Pareto-optimal parameter sets. The third step imposes a restriction stronger than Pareto-optimality. By making "nearly" sufficiently stringent, one can limit attention to, e.g., only 10 parameter sets with estimated security between 2^{64} and 2^{256}. This type of limitation was already used in, e.g., [17], and provides a principled way to reduce concerns about an attacker potentially choosing parameters from a large set to target a secret weakness. See generally [20].

Initial collection of parameter sets. Our latest scans are through all primes p, q with $256 < p < 1024$; $p < q < 16p$; $512 \leq p \log_{256} q \leq 2048$; $q \bmod 6 = 1$; and $x^p - x - 1$ irreducible in $(\mathbb{Z}/q)[x]$.

For each (p, q), we take the maximum possible w for Streamlined NTRU Prime so that there are no decryption failures according to Theorem 3. Later we return to the question of whether this choice of w is optimal.

In the end we reuse the same (p, q) for NTRU LPRime, with smaller w as required for NTRU LPRime. This reuse has obvious advantages, and we have not found significant differences between optimizing (p, q) for Streamlined NTRU Prime and optimizing (p, q) for NTRU LPRime.

Estimated performance. For a performance metric we use $p \log_2 q$, a simple lower bound on the number of bits in a Streamlined NTRU Prime public key. The actual number of bits is slightly more than this, depending on the details of how elements of $\mathcal{R}/q$ are encoded. Another possibility is to use ciphertext size instead of public-key size, or the sum of the two sizes; this makes smaller ratios q/p look marginally better.

Yet another possibility is to take a combination of bytes and cycles (estimated or measured). See Section 5.4.

Tradeoffs. It is easiest to see tradeoffs graphically, so we have drawn many graphs with estimated performance on the x-axis and estimated security on the y-axis, as in [19, attachment] and [85]. The main reason that there are many graphs is that there are many different ways to estimate security against known attacks; see Section 6.

As one example of a previous graph for a particular security estimate, [85, Figure 12] compares performance (the total of public-key size and ciphertext size) to pre-quantum security against known hybrid attacks, assuming BKZ-β cost exponent $0.18728\beta \log_2 \beta - 1.0192\beta + 16.1$ and assuming free memory access. The graph clearly shows the impact of four different weight fractions w/p:

- Weight fractions 0.375 and 0.5 give the best tradeoffs, with 0.5 marginally better for extremely high security levels and 0.375 marginally better for lower security levels.
- Weight fraction 0.6 is noticeably worse, and weight fraction $0.666\ldots$ is even worse.

For comparison, [85, Figure 11] uses BKZ-β cost exponent 0.292β and ignores hybrid attacks. In this graph, weight 0.375 is clearly better than weight 0.5.

The underlying reason for this discrepancy is as follows. Large weights force a large q and thus larger sizes, so there is an obvious benefit of using smaller weights. On the other hand, reducing the weight creates more and more benefit for hybrid attacks, and eventually this turns out to be more important. A moderate weight fraction turns out to be optimum. If hybrid attacks are ignored then low weights produce less benefit for the attacker, and the optimum weight fraction is smaller.

We see no justification for ignoring hybrid attacks. On the other hand, there are at least three arguments that the optimal weight fraction against known attacks is smaller than the optimal weight fraction in [85, Figure 12]:

- Accounting for the real cost of memory makes meet-in-the-middle attacks more expensive, reducing the benefit of hybrid attacks. Compare our Figure 4 to Figure 5.
- Switching from $0.18728\beta\log_2\beta - 1.0192\beta + 16.1$ to 0.396β improves the non-hybrid baseline, further limiting the benefit of hybrid attacks. See Figure 7.
- In the quantum setting, quantum hybrid attacks work in low memory, but quantum enumeration again improves the non-hybrid baseline, again limiting the benefit of hybrid attacks. See Figure 9.

Our graphs make clear that taking weight fraction 0.25 or smaller is noticeably suboptimal for the third setting, and that taking weight fraction 0.5 or larger is noticeably suboptimal for the second setting. We restrict attention to parameters with a weight fraction between 0.25 and 0.5.

Schanck in [85] speculates that “improved combinatorial attacks” will lead to an optimal weight fraction between 0.5 and $0.666\ldots$, meaning that smaller choices will turn out to damage security at any particular level of performance. One can alternatively speculate that improvements in BKZ-β will have the opposite effect, meaning that larger choices will turn out to damage security at any particular level of performance. Past improvements have been in both of these directions, and there is no way to simultaneously avoid both risks.

Density of the parameter space. As our graphs illustrate, the NTRU Prime parameter space provides many closely spaced parameter options. We could easily provide an even closer spacing, for example by allowing more polynomials with small coefficients and maximum-size Galois group. Our general parameter-selection methodology would then impose narrower limits on the weight fraction, bringing the fraction even closer to its optimal value, to focus on a few particularly attractive parameter sets.

There is, however, a difficulty here: different security estimates do not agree on precisely what the optimal value is. Slightly increasing security to its optimum for one estimate means slightly decreasing security for another estimate. The resulting parameter sets would not be robust against variations in the current security estimates (never mind the likelihood of future advances in attacks against lattice-based cryptography). Our current parameter space gives us a much more robust parameter selection; see below.

The choice of w. We return to the question mentioned earlier of whether w should be chosen as large as possible for each (p, q).

Analyses generally indicate that larger weights are harder to find, with the following exception: taking w/p larger than $2/3$ reduces the number of possible keys. This makes simple meet-in-the-middle attacks faster for $w/p > 2/3$. The same effect speeds up various estimates for hybrid attacks if w/p is close enough to 1. For example, for $(p, q) = (619, 9679)$, some of our estimates say that $w = 604$ has slightly less security than smaller values of w.

This phenomenon has no effect on our parameter selection. The cases where w is reduced for better estimated security still have $w \geq 2/3$, and have worse size-security tradeoffs than parameter sets where the maximum w is below $2/3$. For simplicity our parameter definitions prohibit $w/p > 2/3$, although we have included $w/p > 2/3$ in our graphs.

Choice of performance requirement. Consider the problem of fitting a client's public key into a single Internet packet. This allows a server to immediately set up a cryptographic session, encapsulating a session key in its response packet, without having to buffer any data; such buffers are a traditional target for denial-of-service attacks.

IPv6 guarantees that a 1280-byte packet will be transmitted successfully (while measurements such as [94] and [93] indicate that slightly larger packets often encounter failures), so it is natural to set 1280 bytes as a packet-size limit. This does not mean that the key-size limit should be as large as 1280 bytes. Part of the space in a packet is consumed by overhead, and the amount of overhead depends on other protocol details. For example, a minimal IPv6 header consumes 40 bytes (as opposed to 20 bytes for IPv4); a UDP header consumes 8 bytes; protocol designers often include 32 bytes for an ECC key; protocol designers often include a connection identifier; etc. Leaving some room below 1280 bytes provides flexibility for the protocol designer.

We rank (p, q, w) by estimated security, subject to requiring $0.25 \leq w/p \leq 0.5$ and $p \log_{256} q \leq 1280$. The top three results are the following:

- $(823, 4513, 282)$, with $w/p \approx 0.34$ and $p \log_{256} q \approx 1248.89$; 31 bytes of flexibility (assuming space-optimal encoding into bytes).
- $(787, 4243, 265)$, with $w/p \approx 0.34$ and $p \log_{256} q \approx 1185.50$; 94 bytes of flexibility.
- $(761, 4591, 286)$, with $w/p \approx 0.38$ and $p \log_{256} q \approx 1157.16$; 122 bytes of flexibility.

The ordering of these three results is consistent across all 13 security estimates in our computations, except that the second and third results are exchanged in one ranking. Note that there is a more important impact of the allowed range of weight fractions: the top two results would have disappeared if we had required $w \geq \lfloor 3p/8 \rfloor$, as in [85]. We have selected the $(761, 4591, 286)$ parameter set, with 122 bytes of flexibility. In our round-1 submission we used some of this flexibility to accommodate a somewhat less space-efficient encoding of public keys; we are now using a more space-efficient encoding.

Ciphertexts for Streamlined NTRU Prime are smaller than public keys. If the public key from the client fits into a packet then the ciphertext from the server will also fit into a packet (even if it is accompanied by, e.g., a 32-byte cookie with a 16-byte authenticator). The same choices of p and q also give the application the option of deploying NTRU LPRime, where the sizes of public key and ciphertext are approximately reversed.

Beyond this parameter set, NIST has requested specification of additional parameter sets for NTRU Prime. It is straightforward to follow the same methodology to generate further parameter sets. We have generated a smaller parameter set from a tighter performance requirement, and a larger parameter set from a looser performance requirement, as explained below.

A tighter performance requirement. Consider the following problem. An application deploys a code-based encryption system, Classic McEliece, with 1MB public keys. The application cannot afford to frequently transmit such large public keys, so the application also deploys NTRU Prime for forward secrecy. For efficiency, the application uses a single 1280-byte IPv6 packet in each direction to initiate session keys for both systems, rather than imposing the delay of an extra round trip to first establish a session key for one system and then establish a session key for the other. The client begins with the server's long-term Classic McEliece public key, and uses this public key to encrypt an ephemeral NTRU Prime public key to the server. The server's response establishes an NTRU Prime session key.

In more detail, the client sends a packet containing a 226-byte Classic McEliece ciphertext. This ciphertext establishes a session key. The client uses this session key to encrypt an ephemeral NTRU Prime public key; the ciphertext is also included in this packet. The encrypted key also needs to be authenticated; the conventional modular option at this point is to use 16 bytes for a high-security authenticator under the same session key. A space-saving alternative is to observe that plaintext confirmation can be hashed together with additional data (replace, e.g., a 32-byte confirmation C with a 32-byte $H(C, D)$) to implicitly authenticate the additional data; the Classic McEliece ciphertext includes plaintext confirmation.

To summarize, the NTRU Prime public key here must fit into $1280 - 226 = 1054$ bytes, and below this we want to leave extra space for flexibility—for example, 48 bytes to accommodate IPv6+UDP header information.

Ranking parameters with $0.25 \leq w/p \leq 0.5$ and $p \log_{256} q \leq 1054$ consistently produces the following top two results:

- $(653, 4621, 288)$, with $w/p \approx 0.44$ and $p \log_{256} q \approx 993.702$; 60 bytes of flexibility.
- $(647, 3559, 222)$, with $w/p \approx 0.34$ and $p \log_{256} q \approx 954.103$; 99 bytes of flexibility.

We select $(653, 4621, 288)$.

A looser performance requirement. Internet packets are normally transmitted through links that actually allow 1500-byte packets. The reason for the 1280-byte packet-size limit in IPv6 (and the reason that slightly larger packets often encounter failures) is that transmitting IPv6 packets through non-IPv6 networks often ends up wrapping each packet in a non-cryptographic tunnel, consuming extra space in each packet. This also occurs to a smaller extent with IPv4.

Imagine that, in the long term, tunnel overhead is systematically reduced below 64 bytes, allowing 1436-byte packets to be safely transmitted through the Internet. Ranking parameter sets with $p \log_{256} q \leq 1436$ gives the following top two candidates:

	pre-quantum				post-quantum				
BKZ model	enum		sieving		enum		sieving		
memory cost	free	real	free	real	free	real	free	real	
ignoring hybrid	292	292	129	174	146	146	117	174	`kem/sntrup653`
including hybrid	197	239	129	174	139	139	117	160	
ignoring hybrid	296	296	130	176	148	148	118	176	`kem/ntrulpr653`
including hybrid	195	236	130	176	142	142	118	160	
ignoring hybrid	368	368	153	208	185	185	139	208	`kem/sntrup761`
including hybrid	230	277	153	208	169	169	139	180	
ignoring hybrid	364	364	155	210	187	187	140	210	`kem/ntrulpr761`
including hybrid	222	275	155	210	168	168	140	185	
ignoring hybrid	437	437	175	237	220	220	159	237	`kem/sntrup857`
including hybrid	262	325	175	237	196	196	159	208	
ignoring hybrid	429	429	176	239	222	222	160	239	`kem/ntrulpr857`
including hybrid	256	318	176	239	196	196	160	209	

Table 2: For each selected parameter set, various estimates of security levels against known attacks. Warnings: Memory is not free. Attackers will not ignore hybrid attacks if those are the most cost-effective attacks. Both "enum" and "sieving" use questionable formulas. See Section 6 for further caveats regarding these numbers.

- $(857, 5167, 322)$, with $w/p \approx 0.38$ and $p \log_{256} q \approx 1321.40$; 114 bytes of flexibility.
- $(863, 4111, 256)$, with $w/p \approx 0.30$ and $p \log_{256} q \approx 1295.07$; 140 bytes of flexibility.

These candidates do not have a consistent ranking across security estimates. Estimates that ignore hybrid attacks usually rank the 863 parameter set higher, while estimates that take hybrid attacks into account usually rank the 857 parameter set higher. Estimates that account for hybrid attacks *and* for the real cost of memory consistently rank the 857 parameter set higher. We thus select the $(857, 5167, 322)$ parameter set. The same parameter set was also noted by Schanck in [85].

Table of estimates for selected parameter sets. See Table 2. To help the reader verify this table, and to provide a foundation for further improvements in the estimates, we are posting a script [22] that computes all 16 estimates. The script takes a total of $3 + 5 + 7$ minutes on one laptop core for our three Streamlined NTRU Prime parameter sets. It takes several times longer for our three NTRU LPRime parameter sets; recall that Product NTRU gives the attacker more flexibility than Quotient NTRU in choosing lattice ranks. The table rounds each estimate down to the nearest integer.

As noted in Section 6.11, there are 13 non-redundant estimates: "pre-quantum enum free/real ignoring hybrid" are the same, "post-quantum enum free/real ignoring hybrid" are the same, and "pre/post-quantum sieving real ignoring hybrid" are the same. More of the estimates match for our selected parameter sets: hybrid attacks reduce the estimates for "enum" and for "post-quantum sieving real" but usually not for other "sieving" columns.

Some numbers are included in the table primarily for comparability to [4]. The largest numbers are for "pre-quantum enum ignoring hybrid", but those numbers are superseded by hybrid attacks. The smallest numbers are for "post-quantum sieving free", but those numbers ignore the cost of memory and, even more extreme, the cost of QRAM.

We again emphasize that there are many ways that all of the estimates are, or could be, overestimating the cost of known attacks, and many ways that the estimates are, or could be, underestimating the cost of known attacks. For example, the Streamlined NTRU Prime estimates seem to overcount rotations; see Section 6.5. This particular effect cannot make more than about 10 bits of difference in our security estimates, but other effects could be larger, and there are many effects that need further study. See generally Section 6.

Looking beyond known attacks. We also emphasize that advances in attacks could reduce costs below the actual costs of known attacks (which in turn could be above or below current *estimates* of the costs of known attacks). One can optimistically hope that advances will merely eliminate existing overheads—for example, a variant of sieving with lower communication costs would reduce the gap between "real" and "free"—but what is much more worrisome is the possibility of larger breakthroughs.

NIST has written that "submitters of algorithms where the complexity of the best known attack has recently decreased significantly, or is otherwise poorly understood, should be especially conservative" in assigning cryptosystems to categories. Clearly the precondition applies to lattice-based cryptography in general, but it is not at all clear how to quantify "especially conservative". We originally assigned our $(761, 4591)$ parameter sets to NIST's Category 5, while various other proposals assigned similar parameter sets to NIST's Category 3; we have decided to reassign this parameter set to Category 3 to avoid spending time on disputes about what "especially conservative" means. To prevent misunderstanding of this change in category, we emphasize that the only speedups in attacks against NTRU Prime since its original publication have been speedups applicable to a much wider range of lattice-based cryptosystems, whereas the opposite is not true: some other lattice-based systems have suffered from serious attacks that have not affected NTRU Prime. Quantitatively, the smallest number listed in Table 2 for $(761, 4591)$ is 2^{139} quantum "operations", while a Grover attack against AES-256 would use just 2^{128} quantum "operations".

8 Expected strength (2.B.4) for each parameter set

8.1 Parameter set `kem/sntrup653`

Category 2.

8.2 Parameter set `kem/sntrup761`

Category 3.

8.3 Parameter set `kem/sntrup857`

Category 4.

8.4 Parameter set `kem/ntrulpr653`

Category 2.

8.5 Parameter set `kem/ntrulpr761`

Category 3.

8.6 Parameter set `kem/ntrulpr857`

Category 4.

9 Advantages and limitations (2.B.6)

There are several proposals of lattice-based cryptosystems that appear to provide high security with keys and ciphertexts fitting into just a few kilobytes. This proposal is designed to have the smallest attack surface, minimizing the number of avenues available to cryptanalysts. Some recent attacks against lattice-based cryptosystems rely on homomorphisms eliminated by this proposal.

At the same time this proposal provides unusually small sizes and excellent speed. One of the reasons for this performance is that this proposal provides the flexibility to target any desired lattice dimension rather precisely, without the "jumps" that appear in most proposals. Future advances in understanding the exact security level of lattice-based cryptography will allow this proposal to be tuned accordingly.

Beware, however, that there are other recent attacks against lattice-based cryptography, including impressive advances against SVP. As noted before, the security of lattice-based cryptography is not well understood. This is a general limitation of lattice-based cryptography. The same limitation is shared by many—but not all—post-quantum proposals.

Additional comments for round 2. Are users going to end up deploying a post-quantum system that turns out to be devastatingly insecure, allowing feasible attacks? Quite possibly, yes. Experience suggests that the most likely way for this to happen is as follows:

- as a direct result of design decisions, security review of the post-quantum system will be much more complicated than necessary;
- as a result, the limited human resources available for security review will run out of time to seriously review everything;
- as a result, serious vulnerabilities will be missed simply because nobody ever looked at the relevant weakness in the attack surface.

As an analogy, OCB2 was broken in 2018, 14 years after a security "proof" at Asiacrypt and 9 years after ISO standardization. The attacks are simple and fast, but this does not mean that they were easy to find: OCB2 is only one of many cryptographic standards, and it has many interacting components that needed attack analysis. Compared to serious security review for OCB2, serious security review for post-quantum systems is another big step up in difficulty, further increasing the risk of security disasters.

The breadth of the attack surface in lattice-based cryptography is illustrated by new attacks published after the beginning of round 1, such as the following:

- Laarhoven and Mariano [64] saved "between a factor 20 to 40 in the time complexity for SVP".
- Bai, Stehlé, and Wen [13] introduced a new BKZ variant that "produces bases of better quality" for the "same cost assigned to the underlying SVP-solver".
- Aono, Nguyen, and Shen [11] adapted "recent quantum tree algorithms by Montanaro and Ambainis–Kokainis" to the context of enumeration.
- Anvers, Vercauteren, and Verbauwhede [41] showed that "an attacker can significantly reduce the security of (Ring/Module)-LWE/LWR based schemes that have a relatively high failure rate".
- Another paper by Anvers, Vercauteren, and Verbauwhede [40] showed that, for LAC-128, "the failure rate is 2^{48} times bigger than estimated under the assumption of independence".
- Hamburg estimated that an initial design of Round5, at its highest claimed security level, actually had failure rate 2^{-55}, presumably allowing attacks with a number of ciphertexts only moderately above 2^{55}.
- Pellet-Mary, Hanrot, and Stehlé [76] broke through the previously claimed half-exponential approximation-factor barrier for number-theoretic attacks against Ideal-SVP. (The attack of [76] relies on an exponential-time precomputation involving the S-unit lattice; but the attack from Campbell–Groves–Shepherd [32] has a much faster precomputation for cyclotomic fields in the case $S = \{\}$, so perhaps there is a similarly fast precomputation for cyclotomic fields for more general S.)

A security reviewer needs to absorb a tremendous amount of recent material simply to evaluate the security of lattice-based cryptography against *known* attacks, and then needs to evaluate the risks of further attacks. Given the number of different attack avenues and the volume of recent progress, it is not reasonable to think that all of the attack avenues have been adequately explored.

For comparison, it is much easier to understand the state-of-the-art attacks against, e.g., the Classic McEliece system. However, Classic McEliece does not work for applications where a public key needs to fit into (e.g.) a single 1280-byte IPv6 packet.

Every small post-quantum proposal is risky. Every lattice-based proposal forces security reviewers to consider the complicated, unstable picture of general lattice attacks. But most lattice-based proposals *also* force security reviewers to worry about, e.g., decryption failures. (Were failure probabilities calculated correctly? How do the calculations take account of error correction? Can attackers find ciphertexts more likely to fail? How does this interact with quantum computation? Do security proofs correctly account for failures?) We are aware of nine round-1 submissions without decryption failures:

- BIG QUAKE (code-based, not in round 2)
- Classic McEliece (code-based)
- DAGS (code-based, not in round 2)
- NTRU-HRSS-KEM (lattice-based)
- NTRU Prime (lattice-based)
- NTS-KEM (code-based)
- Odd Manhattan (lattice-based, not in round 2)
- pqRSA (factoring-based, not in round 2)
- RQC (code-based)

Only two of these are lattice-based submissions that progressed to round 2. More submissions could adjust parameters to avoid decryption failures, but the broader point is that this submission was systematically designed from the outset to simplify the task of security reviewers—subject to the requirement of being a small lattice-based system.

Most lattice-based proposals rely on number fields with small Galois groups, giving tools to the attacker that are *not* necessary for small lattice-based systems. Five years ago these tools broke the usual cyclotomic case of Gentry's original FHE system at STOC 2009, as noted earlier. Followup work has used the same tools for state-of-the-art attacks against a wider range of lattice problems. Security reviewers are forced to ask whether the same tools will damage the security of, e.g., the small-Galois-group case of Ring-LWE.

There were eight round-1 lattice-based encryption proposals that do *not* rely on number fields with small Galois groups. (Three Bears is sometimes counted as a lattice submission, and the number field it uses is not a cyclotomic field, but the number field still has a small Galois group, so Three Bears is not in this list.) Here is one illustrative example of a parameter set from each proposal, showing public-key size + ciphertext size:

• Streamlined NTRU Prime 4591^{761}:	1218 bytes + 1047 bytes.
• LOTUS 128 (not in round 2):	658944 bytes + 1144 bytes.
• Titanium CCA lite (not in round 2):	14720 bytes + 3008 bytes.
• Round2 n1 l1:	3455 bytes + 4837 bytes.
• Frodo 640:	9616 bytes + 9736 bytes.
• EMBLEM II.c (not in round 2):	10016 bytes + 14792 bytes.
• Lizard N663 (not in round 2):	1390592 bytes + 10896 bytes.
• Odd Manhattan 128 (not in round 2):	1626240 bytes + 180224 bytes.

NIST has advocated diversity to protect against the possibility that "a single type of attack will eliminate the bulk of the candidates remaining in the standardization process". If small Galois groups are broken then only three lattice-based encryption proposals will survive, and NTRU Prime will be the only survivor that can fit a public key into an Internet packet.

If *all* number fields are broken without regard to the Galois structure, then every small lattice-based proposal is broken, and the only remaining lattice-based proposals are much larger "unstructured" proposals such as Frodo. NTRU Prime starts from the requirement of being a small lattice-based system, and then reduces the attack surface subject to this requirement.

References

[1] Carlisle Adams and Jan Camenisch, editors. *Selected Areas in Cryptography - SAC 2017 - 24th International Conference, Ottawa, ON, Canada, August 16-18, 2017, Revised Selected Papers*, volume 10719 of *Lecture Notes in Computer Science*. Springer, 2018.

[2] Divesh Aggarwal, Daniel Dadush, Oded Regev, and Noah Stephens-Davidowitz. Solving the shortest vector problem in 2^n time using discrete Gaussian sampling: Extended abstract. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 733–742. ACM, 2015. http://arxiv.org/abs/1412.7994.

[3] Martin R. Albrecht, Carlos Cid, Jean-Charles Faugère, Robert Fitzpatrick, and Ludovic Perret. Algebraic algorithms for LWE problems. *ACM Comm. Computer Algebra*, 49(2):62, 2015. https://eprint.iacr.org/2014/1018.

[4] Martin R. Albrecht, Benjamin R. Curtis, Amit Deo, Alex Davidson, Rachel Player, Eamonn W. Postlethwaite, Fernando Virdia, and Thomas Wunderer. Estimate all the {LWE, NTRU} schemes! In Dario Catalano and Roberto De Prisco, editors, *Security and Cryptography for Networks - 11th International Conference, SCN 2018, Amalfi, Italy, September 5-7, 2018, Proceedings*, volume 11035 of *Lecture Notes in Computer Science*, pages 351–367. Springer, 2018. https://eprint.iacr.org/2018/331.

[5] 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, 2019. https://eprint.iacr.org/2019/089.

[6] Martin R. Albrecht, Florian Göpfert, Fernando Virdia, and Thomas Wunderer. Revisiting the expected cost of solving uSVP and applications to LWE. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I*, volume 10624 of *Lecture Notes in Computer Science*, pages 297–322. Springer, 2017. https://eprint.iacr.org/2017/815.

[7] Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. Post-quantum key exchange - A new hope. In Thorsten Holz and Stefan Savage, editors, *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016*, pages 327–343. USENIX Association, 2016. https://eprint.iacr.org/2015/1092.

[8] Jacob Alperin-Sheriff and Daniel Apon. Dimension-preserving reductions from LWE to LWR. *IACR Cryptology ePrint Archive*, 2016:589, 2016. https://eprint.iacr.org/2016/589.

[9] Joël Alwen, Stephan Krenn, Krzysztof Pietrzak, and Daniel Wichs. Learning with rounding, revisited – new reduction, properties and applications. In Canetti and Garay [33], pages 57–74. https://eprint.iacr.org/2013/098.

[10] Yoshinori Aono and Phong Q. Nguyen. Random sampling revisited: Lattice enumeration with discrete pruning. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part II*, volume 10211 of *Lecture Notes in Computer Science*, pages 65–102, 2017. https://eprint.iacr.org/2017/155.

[11] Yoshinori Aono, Phong Q. Nguyen, and Yixin Shen. Quantum lattice enumeration and tweaking discrete pruning. In Peyrin and Galbraith [78], pages 405–434. https://eprint.iacr.org/2018/546.

[12] Sanjeev Arora and Rong Ge. New algorithms for learning in presence of errors. In Luca Aceto, Monika Henzinger, and Jirí Sgall, editors, *Automata, Languages and Programming - 38th International Colloquium, ICALP 2011, Zurich, Switzerland, July 4-8, 2011, Proceedings, Part I*, volume 6755 of *Lecture Notes in Computer Science*, pages 403–415. Springer, 2011. https://users.cs.duke.edu/~rongge/LPSN.pdf.

[13] Shi Bai, Damien Stehlé, and Weiqiang Wen. Measuring, simulating and exploiting the head concavity phenomenon in BKZ. In Peyrin and Galbraith [78], pages 369–404. https://eprint.iacr.org/2018/856.

[14] Abhishek Banerjee, Chris Peikert, and Alon Rosen. Pseudorandom functions and lattices. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology -*

EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings, volume 7237 of *Lecture Notes in Computer Science*, pages 719–737. Springer, 2012. https://eprint.iacr.org/2011/401.

[15] Jens Bauch, Daniel J. Bernstein, Henry de Valence, Tanja Lange, and Christine van Vredendaal. Short generators without quantum computers: The case of multiquadratics. In Coron and Nielsen [37], pages 27–59. https://multiquad.cr.yp.to.

[16] Anja Becker, Léo Ducas, Nicolas Gama, and Thijs Laarhoven. New directions in nearest neighbor searching with applications to lattice sieving. In Krauthgamer [61], pages 10–24. https://eprint.iacr.org/2015/1128.

[17] Daniel J. Bernstein. Curve25519: New Diffie-Hellman speed records. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *Public Key Cryptography - PKC 2006, 9th International Conference on Theory and Practice of Public-Key Cryptography, New York, NY, USA, April 24-26, 2006, Proceedings*, volume 3958 of *Lecture Notes in Computer Science*, pages 207–228. Springer, 2006. https://cr.yp.to/papers.html#curve25519.

[18] Daniel J. Bernstein. djbsort, 2018. https://sorting.cr.yp.to.

[19] Daniel J. Bernstein. Re: OFFICIAL COMMENT: NTRU Prime, 2018. https://groups.google.com/a/list.nist.gov/d/msg/pqc-forum/l5IaJTe_pUI/QKaLZ4uMAAAJ.

[20] Daniel J. Bernstein, Tung Chou, Chitchanok Chuengsatiansup, Andreas Hülsing, Eran Lambooij, Tanja Lange, Ruben Niederhagen, and Christine van Vredendaal. How to manipulate curve standards: A white paper for the black hat. In Liqun Chen and Shin'ichiro Matsuo, editors, *Security Standardisation Research - Second International Conference, SSR 2015, Tokyo, Japan, December 15-16, 2015, Proceedings*, volume 9497 of *Lecture Notes in Computer Science*, pages 109–139. Springer, 2015. https://eprint.iacr.org/2014/571.

[21] Daniel J. Bernstein, Chitchanok Chuengsatiansup, Tanja Lange, and Christine van Vredendaal. NTRU Prime: Reducing attack surface at low cost. In Adams and Camenisch [1], pages 235–260. https://eprint.iacr.org/2016/461.

[22] Daniel J. Bernstein, Chitchanok Chuengsatiansup, Tanja Lange, and Christine van Vredendaal. Script to compute various security estimates, 2019. https://ntruprime.cr.yp.to/security.html.

[23] Daniel J. Bernstein, Mike Hamburg, Anna Krasnova, and Tanja Lange. Elligator: elliptic-curve points indistinguishable from uniform random strings. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*, pages 967–980. ACM, 2013. https://eprint.iacr.org/2013/325.

[24] Daniel J. Bernstein and Tanja Lange. Non-uniform cracks in the concrete: The power of free precomputation. In Kazue Sako and Palash Sarkar, editors, *Advances in Cryptology - ASIACRYPT 2013 - 19th International Conference on the Theory and Application of Cryptology and Information Security, Bengaluru, India, December 1-5, 2013, Proceedings, Part II*, volume 8270 of *Lecture Notes in Computer Science*, pages 321–340. Springer, 2013. https://cr.yp.to/papers.html#nonuniform.

[25] Daniel J. Bernstein and Edoardo Persichetti. Towards KEM unification, 2018. https://eprint.iacr.org/2018/526.

[26] Daniel J. Bernstein and Bo-Yin Yang. Fast constant-time gcd computation and modular inversion, 2019. https://eprint.iacr.org/2019/266.

[27] Jean-François Biasse and Fang Song. Efficient quantum algorithms for computing class groups and solving the principal ideal problem in arbitrary degree number fields. In Krauthgamer [61], pages 893–902. http://fangsong.info/files/pubs/BS_SODA16.pdf.

[28] Andrej Bogdanov, Siyao Guo, Daniel Masny, Silas Richelson, and Alon Rosen. On the hardness of learning with rounding over small modulus. In Eyal Kushilevitz and Tal Malkin, editors, *Theory of Cryptography - 13th International Conference, TCC 2016-A, Tel Aviv, Israel, January 10-13, 2016, Proceedings, Part I*, volume 9562 of *Lecture Notes in Computer Science*, pages 209–224. Springer, 2016. https://eprint.iacr.org/2015/769.

[29] Joppe W. Bos, Craig Costello, Léo Ducas, Ilya Mironov, Michael Naehrig, Valeria Nikolaenko, Ananth Raghunathan, and Douglas Stebila. Frodo: Take off the ring! Practical, quantum-secure key exchange from LWE. In *ACM Conference on Computer and Communications Security*, pages 1006–1018. ACM, 2016. https://eprint.iacr.org/2016/659.

[30] Joppe W. Bos, Craig Costello, Michael Naehrig, and Douglas Stebila. Post-quantum key exchange for the TLS protocol from the ring learning with errors problem. In *IEEE Symposium on Security and Privacy*, pages 553–570. IEEE Computer Society, 2015. https://eprint.iacr.org/2014/599.

[31] Leon Groot Bruinderink, Andreas Hülsing, Tanja Lange, and Yuval Yarom. Flush, Gauss, and reload - A cache attack on the BLISS lattice-based signature scheme. In Benedikt Gierlichs and Axel Y. Poschmann, editors, *Cryptographic Hardware and Embedded Systems - CHES 2016 - 18th International Conference, Santa Barbara, CA, USA, August 17-19, 2016, Proceedings*, volume 9813 of *Lecture Notes in Computer Science*, pages 323–345. Springer, 2016. https://eprint.iacr.org/2016/300.

[32] Peter Campbell, Michael Groves, and Dan Shepherd. Soliloquy: a cautionary tale, 2014. http://docbox.etsi.org/Workshop/2014/201410_CRYPTO/S07_Systems_and_Attacks/S07_Groves_Annex.pdf.

[33] Ran Canetti and Juan A. Garay, editors. *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, volume 8042 of *Lecture Notes in Computer Science*. Springer, 2013.

[34] Hao Chen, Kristin Lauter, and Katherine E. Stange. Vulnerable Galois RLWE families and improved attacks. *IACR Cryptology ePrint Archive*, 2016. https://eprint.iacr.org/2016/193.

[35] Yuanmi Chen. *Réduction de réseau et sécurité concrete du chiffrement completement homomorphe*. PhD thesis, Paris 7, 2013.

[36] 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 - 17th International Conference on the Theory and Application of Cryptology and Information Security, Seoul, South Korea, December 4-8, 2011. Proceedings*, volume 7073 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2011. https://www.iacr.org/archive/asiacrypt2011/70730001/70730001.pdf.

[37] Jean-Sébastien Coron and Jesper Buus Nielsen, editors. *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part I*, volume 10210 of *Lecture Notes in Computer Science*, 2017.

[38] Ronald Cramer, Léo Ducas, Chris Peikert, and Oded Regev. Recovering short generators of principal ideals in cyclotomic rings. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 559–585. Springer, 2016. https://eprint.iacr.org/2015/313.

[39] Ronald Cramer, Léo Ducas, and Benjamin Wesolowski. Short Stickelberger class relations and application to Ideal-SVP. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part I*, volume 10210 of *Lecture Notes in Computer Science*, pages 324–348, 2017. https://eprint.iacr.org/2016/885.

[40] Jan-Pieter D'Anvers, Frederik Vercauteren, and Ingrid Verbauwhede. The impact of error dependencies on Ring/Mod-LWE/LWR based schemes. *IACR Cryptology ePrint Archive*, 2018:1172, 2018. https://eprint.iacr.org/2018/1172.

[41] Jan-Pieter D'Anvers, Frederik Vercauteren, and Ingrid Verbauwhede. On the impact of decryption failures on the security of LWE/LWR based schemes. *IACR Cryptology ePrint Archive*, 2018:1089, 2018. https://eprint.iacr.org/2018/1089.

[42] Alexander W. Dent. A designer's guide to KEMs. In Kenneth G. Paterson, editor, *Cryptography and Coding, 9th IMA International Conference, Cirencester, UK, December 16-18, 2003, Proceedings*, volume 2898 of *Lecture Notes in Computer Science*, pages 133–151. Springer, 2003. https://eprint.iacr.org/2002/174.

[43] Jintai Ding. Solving LWE problem with bounded errors in polynomial time. *IACR Cryptology ePrint Archive*, 2010:558, 2010. https://eprint.iacr.org/2010/558.

[44] Léo Ducas, Alain Durmus, Tancrède Lepoint, and Vadim Lyubashevsky. Lattice signatures and bimodal Gaussians. In Canetti and Garay [33], pages 40–56. https://eprint.iacr.org/2013/383.

[45] Dave Dunning. Fabrics—why we love them and why we hate them, 2015. https://www.openfabrics.org/images/eventpresos/workshops2015/DevWorkshop/Tuesday/tuesday_10.pdf.

[46] Kirsten Eisenträger, Sean Hallgren, and Kristin E. Lauter. Weak instances of PLWE. In Antoine Joux and Amr M. Youssef, editors, *Selected Areas in Cryptography - SAC 2014 - 21st International Conference, Montreal, QC, Canada, August 14-15, 2014, Revised Selected Papers*, volume 8781 of *Lecture Notes in Computer Science*, pages 183–194. Springer, 2014. https://eprint.iacr.org/2014/784.

[47] Ulrich Fincke and Michael Pohst. Improved methods for calculating vectors of short length in a lattice, including a complexity analysis. *Mathematics of Computation*, 44(170):463–471, 1985. http://www.ams.org/journals/mcom/1985-44-170/S0025-5718-1985-0777278-8/S0025-5718-1985-0777278-8.pdf.

[48] Nicolas Gama, Phong Q. Nguyen, and Oded Regev. Lattice enumeration using extreme pruning. In Henri Gilbert, editor, *Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Monaco / French Riviera, May 30 - June 3, 2010. Proceedings*, volume 6110 of *Lecture Notes in Computer Science*, pages 257–278. Springer, 2010. https://www.iacr.org/archive/eurocrypt2010/66320257/66320257.pdf.

[49] Florian Göpfert, Christine van Vredendaal, and Thomas Wunderer. A hybrid lattice basis reduction and quantum search attack on LWE. In Tanja Lange and Tsuyoshi Takagi, editors, *Post-Quantum Cryptography - 8th International Workshop, PQCrypto 2017, Utrecht, The Netherlands, June 26-28, 2017, Proceedings*, volume 10346 of *Lecture Notes in Computer Science*, pages 184–202. Springer, 2017. https://eprint.iacr.org/2017/221.

[50] Qian Guo, Thomas Johansson, and Paul Stankovski. A key recovery attack on MDPC with CCA security using decoding errors. Cryptology ePrint Archive, Report 2016/858, 2016. https://eprint.iacr.org/2016/858.

[51] Guillaume Hanrot, Xavier Pujol, and Damien Stehlé. Terminating BKZ. *IACR Cryptology ePrint Archive*, 2011:198, 2011. https://eprint.iacr.org/2011/198.

[52] 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, 7th International Conference, ACNS 2009, Paris-Rocquencourt, France, June 2-5, 2009. Proceedings*, volume

5536 of *Lecture Notes in Computer Science*, pages 437–455, 2009. https://assets.securityinnovation.com/static/downloads/NTRU/resources/params.pdf.

[53] Jeffrey Hoffstein, Jill Pipher, John M. Schanck, Joseph H. Silverman, William Whyte, and Zhenfei Zhang. Choosing parameters for NTRUEncrypt. *IACR Cryptology ePrint Archive*, 2015. https://eprint.iacr.org/2015/708.

[54] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. NTRU: A ring-based public key cryptosystem. In Joe Buhler, editor, *Algorithmic Number Theory, Third International Symposium, ANTS-III, Portland, Oregon, USA, June 21-25, 1998, Proceedings*, volume 1423 of *Lecture Notes in Computer Science*, pages 267–288. Springer, 1998.

[55] Nick Howgrave-Graham. A hybrid lattice-reduction and meet-in-the-middle attack against NTRU. In Alfred Menezes, editor, *Advances in Cryptology - CRYPTO 2007, 27th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2007, Proceedings*, volume 4622 of *Lecture Notes in Computer Science*, pages 150–169. Springer, 2007. https://www.iacr.org/archive/crypto2007/46220150/46220150.pdf.

[56] Nick Howgrave-Graham, Phong Q. Nguyen, David Pointcheval, John Proos, Joseph H. Silverman, Ari Singer, and William Whyte. The impact of decryption failures on the security of NTRU encryption. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, volume 2729 of *Lecture Notes in Computer Science*, pages 226–246. Springer, 2003. http://www.di.ens.fr/~pointche/Documents/Papers/2003_crypto.pdf.

[57] Nick Howgrave-Graham, Joseph H Silverman, and William Whyte. A meet-in-the-middle attack on an NTRU private key. Technical report, NTRU Cryptosystems, June 2003. Report, 2003. https://www.securityinnovation.com/uploads/Crypto/NTRUTech004v2.pdf.

[58] Nick Howgrave-Graham, Joseph H. Silverman, and William Whyte. Choosing parameter sets for NTRUEncrypt with NAEP and SVES-3, 2005. https://eprint.iacr.org/2005/045.

[59] Ravi Kannan. Improved algorithms for integer programming and related lattice problems. In *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*, STOC '83, pages 193–206, New York, NY, USA, 1983. ACM.

[60] Paul Kirchner and Pierre-Alain Fouque. Revisiting lattice attacks on overstretched NTRU parameters. In Coron and Nielsen [37], pages 3–26. https://www.di.ens.fr/~fouque/euro17a.pdf.

[61] Robert Krauthgamer, editor. *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*. SIAM, 2016.

[62] Po-Chun Kuo, Michael Schneider, Özgür Dagdelen, Jan Reichelt, Johannes A. Buchmann, Chen-Mou Cheng, and Bo-Yin Yang. Extreme enumeration on GPU and in clouds: How many dollars you need to break SVP challenges. In Bart Preneel and Tsuyoshi Takagi, editors, *Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings*, volume 6917 of *Lecture Notes in Computer Science*, pages 176–191. Springer, 2011. http://www.iis.sinica.edu.tw/papers/byyang/12158-F.pdf.

[63] Thijs Laarhoven. Sieving for shortest vectors in lattices using angular locality-sensitive hashing. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I*, volume 9215 of *Lecture Notes in Computer Science*, pages 3–22. Springer, 2015. https://eprint.iacr.org/2014/744.pdf.

[64] Thijs Laarhoven and Artur Mariano. Progressive lattice sieving. In Tanja Lange and Rainer Steinwandt, editors, *Post-Quantum Cryptography - 9th International Conference, PQCrypto 2018, Fort Lauderdale, FL, USA, April 9-11, 2018, Proceedings*, volume 10786 of *Lecture Notes in Computer Science*, pages 292–311. Springer, 2018. https://eprint.iacr.org/2018/079.

[65] Thijs Laarhoven, Michele Mosca, and Joop van de Pol. Finding shortest lattice vectors faster using quantum search. *Des. Codes Cryptography*, 77(2-3):375–400, 2015. https://eprint.iacr.org/2014/907.

[66] Adam Langley. How to botch TLS forward secrecy, 2013. https://www.imperialviolet.org/2013/06/27/botchingpfs.html.

[67] Vadim Lyubashevsky. Future directions in lattice cryptography (talk slides), 2016. http://troll.iis.sinica.edu.tw/pkc16/slides/Invited_Talk_II--Directions_in_Practical_Lattice_Cryptography.pptx.

[68] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. *J. ACM*, 60(6):43, 2013. https://eprint.iacr.org/2012/230.

[69] 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, Providence, RI, USA, March 29-30, 2001, Revised Papers*, volume 2146 of *Lecture Notes in Computer Science*, pages 110–125. Springer, 2001.

[70] Alfred Menezes. Another look at HMQV. *J. Mathematical Cryptology*, 1(1):47–64, 2007. https://eprint.iacr.org/2005/205.

[71] Christopher Meyer, Juraj Somorovsky, Eugen Weiss, Jörg Schwenk, Sebastian Schinzel, and Erik Tews. Revisiting SSL/TLS implementations: New Bleichenbacher side channels and attacks. In Kevin Fu and Jaeyeon Jung, editors, *Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, August 20-22, 2014.*,

pages 733–748. USENIX Association, 2014. https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/meyer.

[72] Phong Q. Nguyen and Thomas Vidick. Sieve algorithms for the shortest vector problem are practical. *J. Mathematical Cryptology*, 2(2):181–207, 2008. https://people.csail.mit.edu/vidick/JoMC08.pdf.

[73] Paul C. van Oorschot and Michael J. Wiener. Parallel collision search with cryptanalytic applications. *J. Cryptology*, 12(1):1–28, 1999. http://people.scs.carleton.ca/~paulv/papers/JoC97.pdf.

[74] Chris Peikert. Public-key cryptosystems from the worst-case shortest vector problem: extended abstract. In Michael Mitzenmacher, editor, *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 333–342. ACM, 2009. https://eprint.iacr.org/2008/481.

[75] Chris Peikert. "A useful fact about Ring-LWE that should be known better: it is *at least as hard* to break as NTRU, and likely strictly harder. 1/" (tweet), 2017. http://archive.is/B9KEW.

[76] Alice Pellet-Mary, Guillaume Hanrot, and Damien Stehlé. Approx-SVP in ideal lattices with pre-processing, 2019. https://eprint.iacr.org/2019/215.

[77] Peter Pessl, Leon Groot Bruinderink, and Yuval Yarom. To BLISS-B or not to be: Attacking strongSwan's implementation of post-quantum signatures. In *CCS*, pages 1843–1855. ACM, 2017. https://eprint.iacr.org/2017/490.

[78] Thomas Peyrin and Steven D. Galbraith, editors. *Advances in Cryptology - ASIACRYPT 2018 - 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2-6, 2018, Proceedings, Part I*, volume 11272 of *Lecture Notes in Computer Science*. Springer, 2018.

[79] Michael Pohst. On the computation of lattice vectors of minimal length, successive minima and reduced bases with applications. *SIGSAM Bull.*, 15(1):37–44, February 1981.

[80] Thomas Pöppelmann and Tim Güneysu. Towards practical lattice-based public-key encryption on reconfigurable hardware. In Tanja Lange, Kristin E. Lauter, and Petr Lisonek, editors, *Selected Areas in Cryptography - SAC 2013 - 20th International Conference, Burnaby, BC, Canada, August 14-16, 2013, Revised Selected Papers*, volume 8282 of *Lecture Notes in Computer Science*, pages 68–85. Springer, 2013. https://www.ei.rub.de/media/sh/veroeffentlichungen/2013/08/14/lwe_encrypt.pdf.

[81] Xavier Pujol and Damien Stehlé. Solving the shortest lattice vector problem in time $2^{2.465n}$. *IACR Cryptology ePrint Archive*, 2009. https://eprint.iacr.org/2009/605.

[82] Markku-Juhani O. Saarinen. HILA5: on reliability, reconciliation, and error correction for Ring-LWE encryption. In Adams and Camenisch [1], pages 192–212. https://eprint.iacr.org/2017/424.

[83] Tsunekazu Saito, Keita Xagawa, and Takashi Yamakawa. Tightly-secure key-encapsulation mechanism in the quantum random oracle model. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part III*, volume 10822 of *Lecture Notes in Computer Science*, pages 520–551. Springer, 2018. https://eprint.iacr.org/2017/1005.

[84] Halvor Sakshaug. Security analysis of the NTRUEncrypt public key encryption scheme, 2007. brage.bibsys.no/xmlui/bitstream/handle/11250/258846/426901_FULLTEXT01.pdf.

[85] John M. Schanck. A comparison of NTRU variants. *IACR Cryptology ePrint Archive*, 2018:1174, 2018. https://eprint.iacr.org/2018/1174.

[86] Claus-Peter Schnorr. Lattice reduction by random sampling and birthday methods. In Helmut Alt and Michel Habib, editors, *STACS*, volume 2607 of *Lecture Notes in Computer Science*, pages 145–156. Springer, 2003. http://www.math.uni-frankfurt.de/~dmst/research/papers/schnorr.random_sampling.2003.ps.

[87] Claus-Peter Schnorr and M. Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. *Math. Program.*, 66:181–199, 1994.

[88] Claus-Peter Schnorr and Adi Shamir. An optimal sorting algorithm for mesh connected computers. In Juris Hartmanis, editor, *Proceedings of the 18th Annual ACM Symposium on Theory of Computing, May 28-30, 1986, Berkeley, California, USA*, pages 255–263. ACM, 1986.

[89] Victor Shoup. A proposal for an ISO standard for public key encryption. *IACR Cryptology ePrint Archive*, 2001. https://eprint.iacr.org/2001/112.

[90] Victor Shoup. OAEP reconsidered. *J. Cryptology*, 15(4):223–249, 2002. https://eprint.iacr.org/2000/060.

[91] Martijn Stam. A key encapsulation mechanism for NTRU. In Nigel P. Smart, editor, *Cryptography and Coding, 10th IMA International Conference, Cirencester, UK, December 19-21, 2005, Proceedings*, volume 3796 of *Lecture Notes in Computer Science*, pages 410–427. Springer, 2005.

[92] Damien Stehlé and Ron Steinfeld. Making NTRU as secure as worst-case problems over ideal lattices. In Kenneth G. Paterson, editor, *Advances in Cryptology - EUROCRYPT 2011 - 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, Estonia, May 15-19, 2011. Proceedings*, volume 6632 of *Lecture Notes in Computer Science*, pages 27–47. Springer, 2011. https://www.iacr.org/archive/eurocrypt2011/66320027/66320027.pdf.

[93] Iljitsch van Beijnum. Internet packet sizes part 2: IPv4 path MTU discovery is dead, 2014. http://www.bgpexpert.com/article.php?article=152.

[94] Iljitsch van Beijnum. Maximum packet sizes on the internet, 2014. http://www.bgpexpert.com/article.php?article=151.

[95] Christine van Vredendaal. Reduced memory meet-in-the-middle attack against the NTRU private key. *LMS Journal of Computation and Mathematics*, 19(A):43–57, 001 2016. https://eprint.iacr.org/2016/177.

[96] Thomas Wunderer. Revisiting the hybrid attack: Improved analysis and refined security estimates, 2016. https://eprint.iacr.org/2016/733.

[97] Thomas Wunderer. *On the Security of Lattice-Based Cryptography Against Lattice Reduction and Hybrid Attacks*. PhD thesis, Darmstadt University of Technology, Germany, 2018. https://tuprints.ulb.tu-darmstadt.de/8082/.

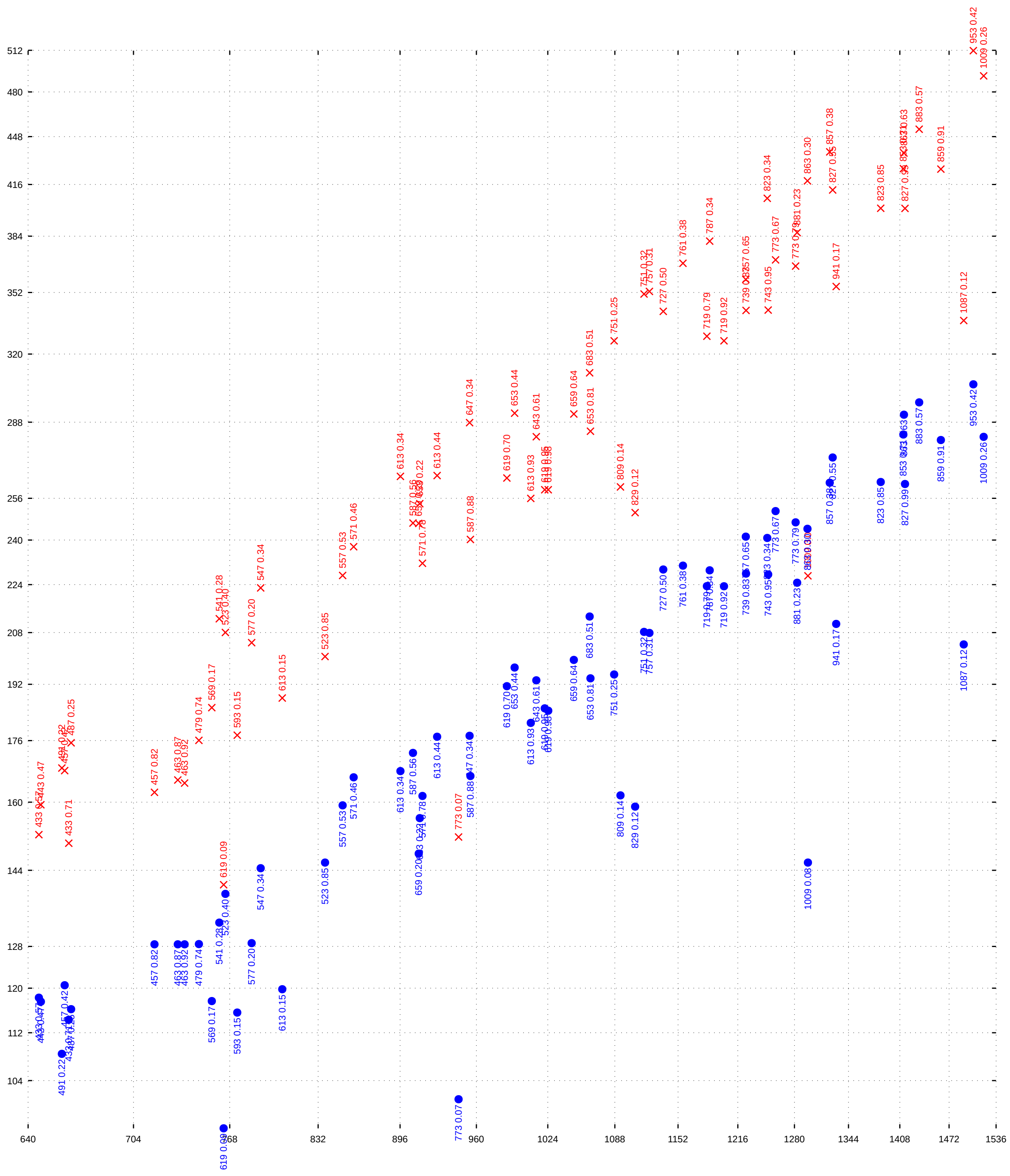


Figure 4: Estimated pre-quantum security level against known attacks, assuming enumeration-based model of BKZ-β cost, assuming free access to memory. Horizontal axis: $p \log_{256} q$. Label: $p, w/p$. Log scale. Red: Ignoring hybrid attacks. Blue: Accounting for all attacks. See text for caveats regarding estimates.

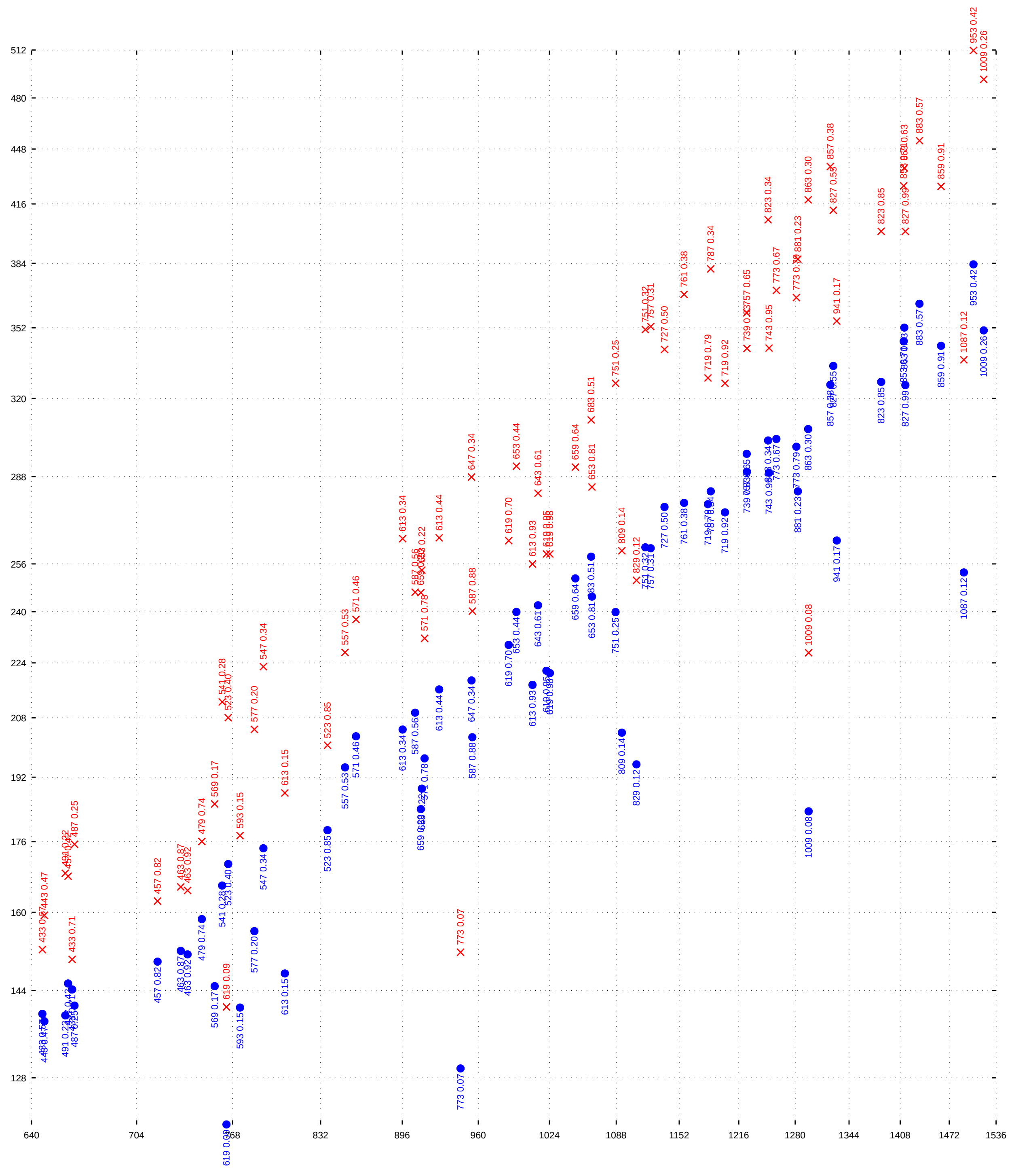


Figure 5: Estimated pre-quantum security level against known attacks, assuming enumeration-based model of BKZ-β cost, accounting for real cost of memory. Horizontal axis: $p \log_{256} q$. Label: $p, w/p$. Log scale. Red: Ignoring hybrid attacks. Blue: Accounting for all attacks. See text for caveats regarding estimates.

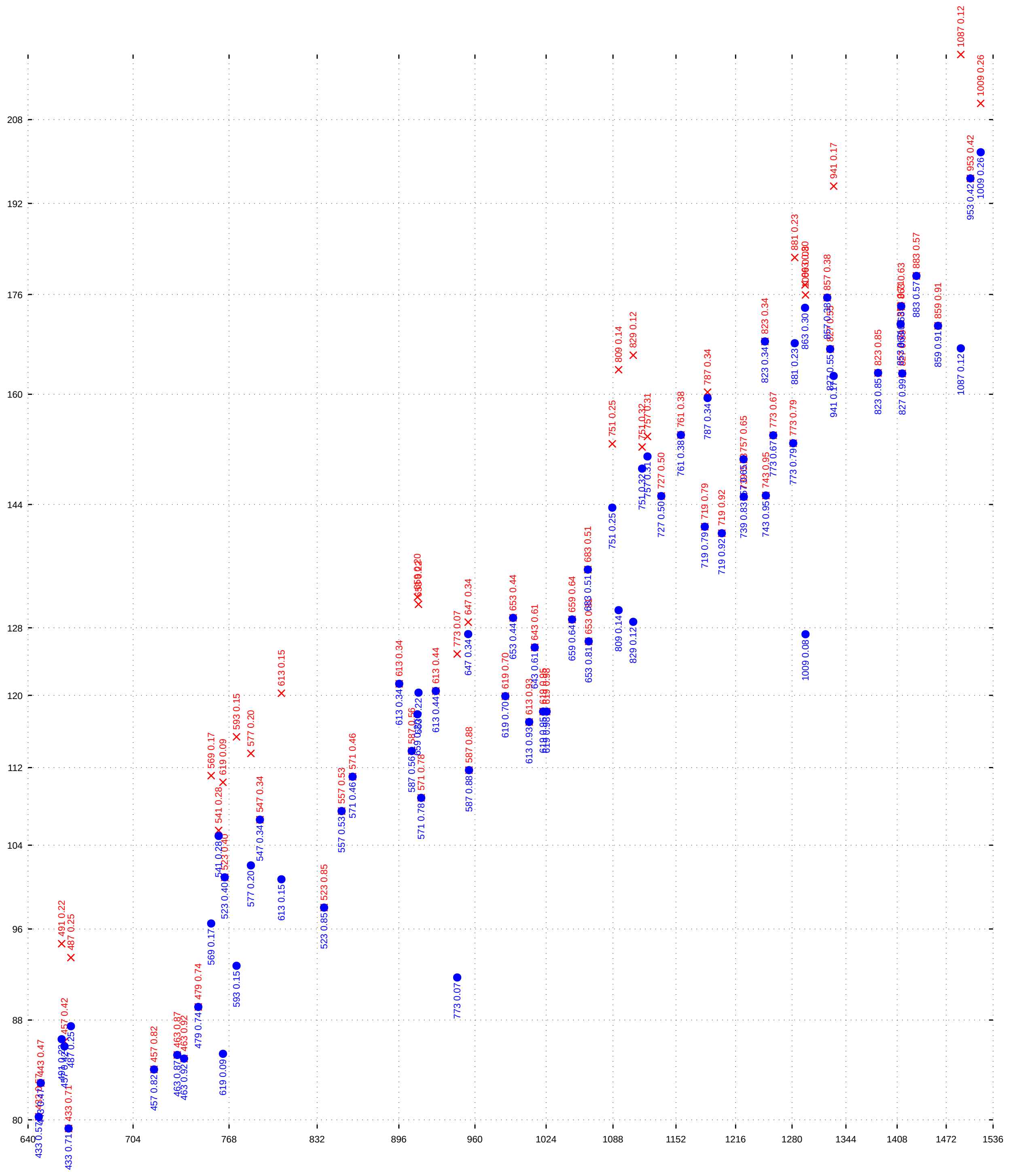


Figure 6: Estimated pre-quantum security level against known attacks, assuming sieving-based model of BKZ-β cost, assuming free access to memory. Horizontal axis: $p \log_{256} q$. Label: $p, w/p$. Log scale. Red: Ignoring hybrid attacks. Blue: Accounting for all attacks. See text for caveats regarding estimates.

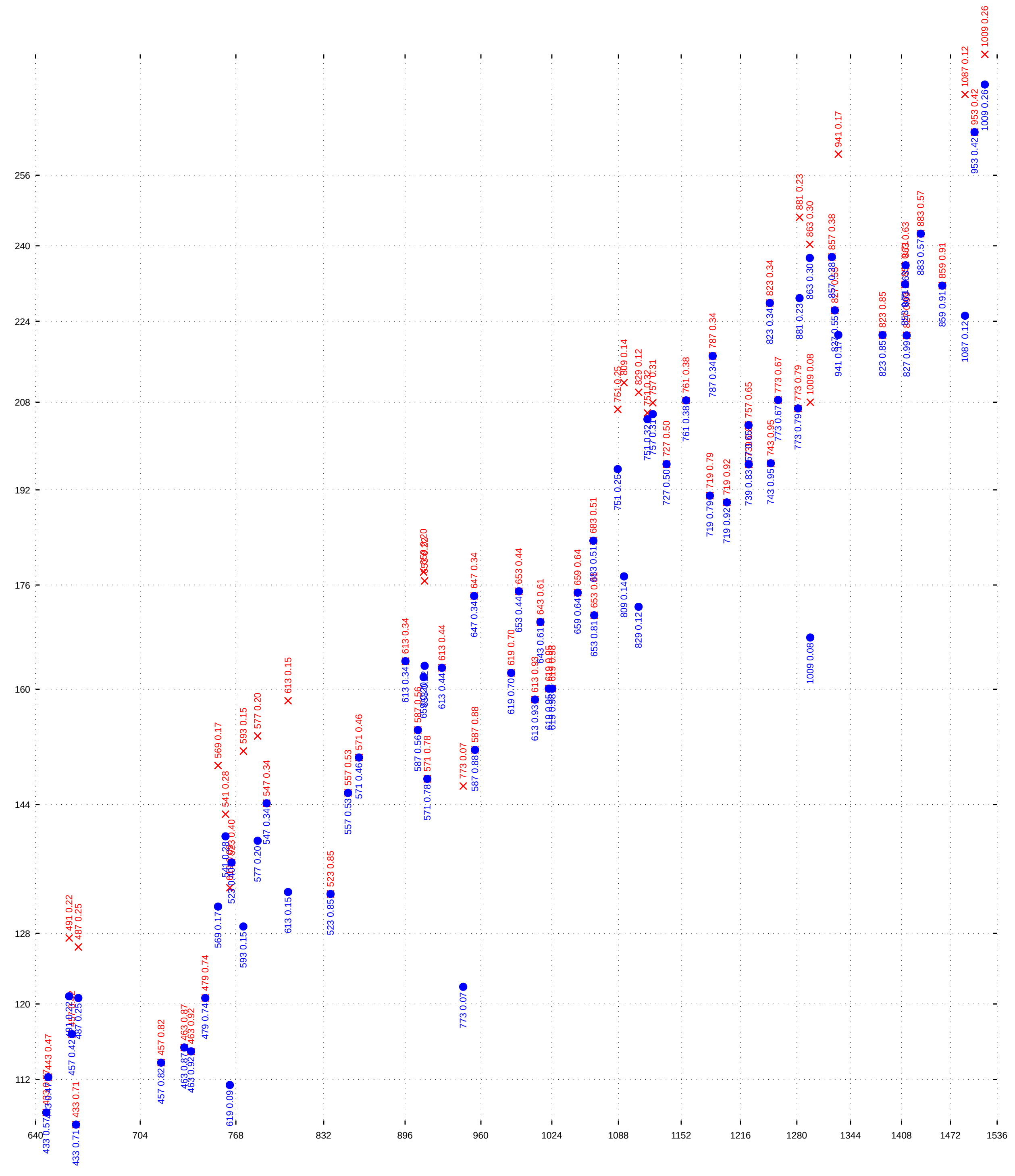


Figure 7: Estimated pre-quantum security level against known attacks, assuming sieving-based model of BKZ-β cost, accounting for real cost of memory. Horizontal axis: $p \log_{256} q$. Label: $p, w/p$. Log scale. Red: Ignoring hybrid attacks. Blue: Accounting for all attacks. See text for caveats regarding estimates.

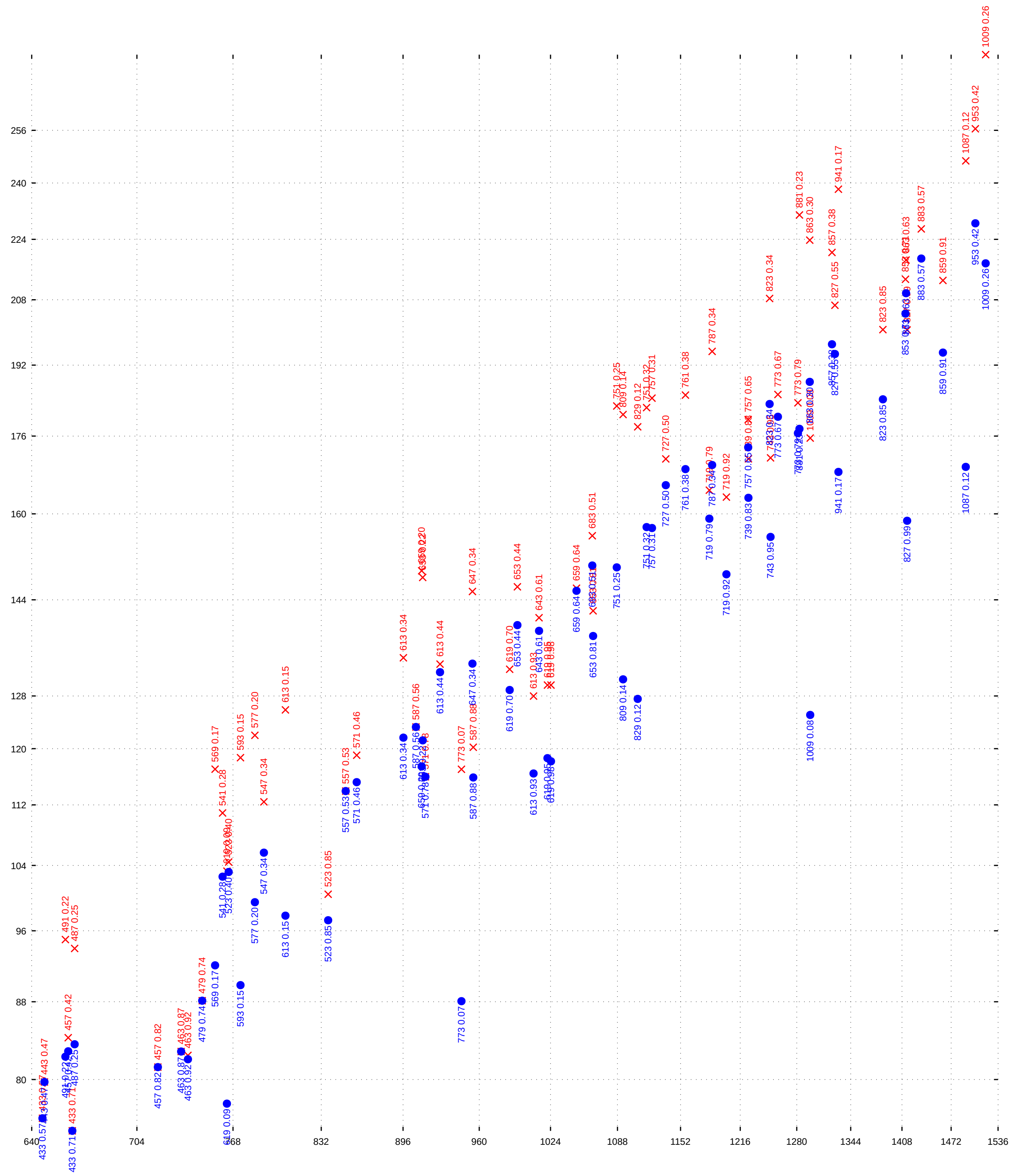


Figure 8: Estimated post-quantum security level against known attacks, assuming enumeration-based model of BKZ-β cost, assuming free access to memory. Horizontal axis: $p \log_{256} q$. Label: $p, w/p$. Log scale. Red: Ignoring hybrid attacks. Blue: Accounting for all attacks. See text for caveats regarding estimates.

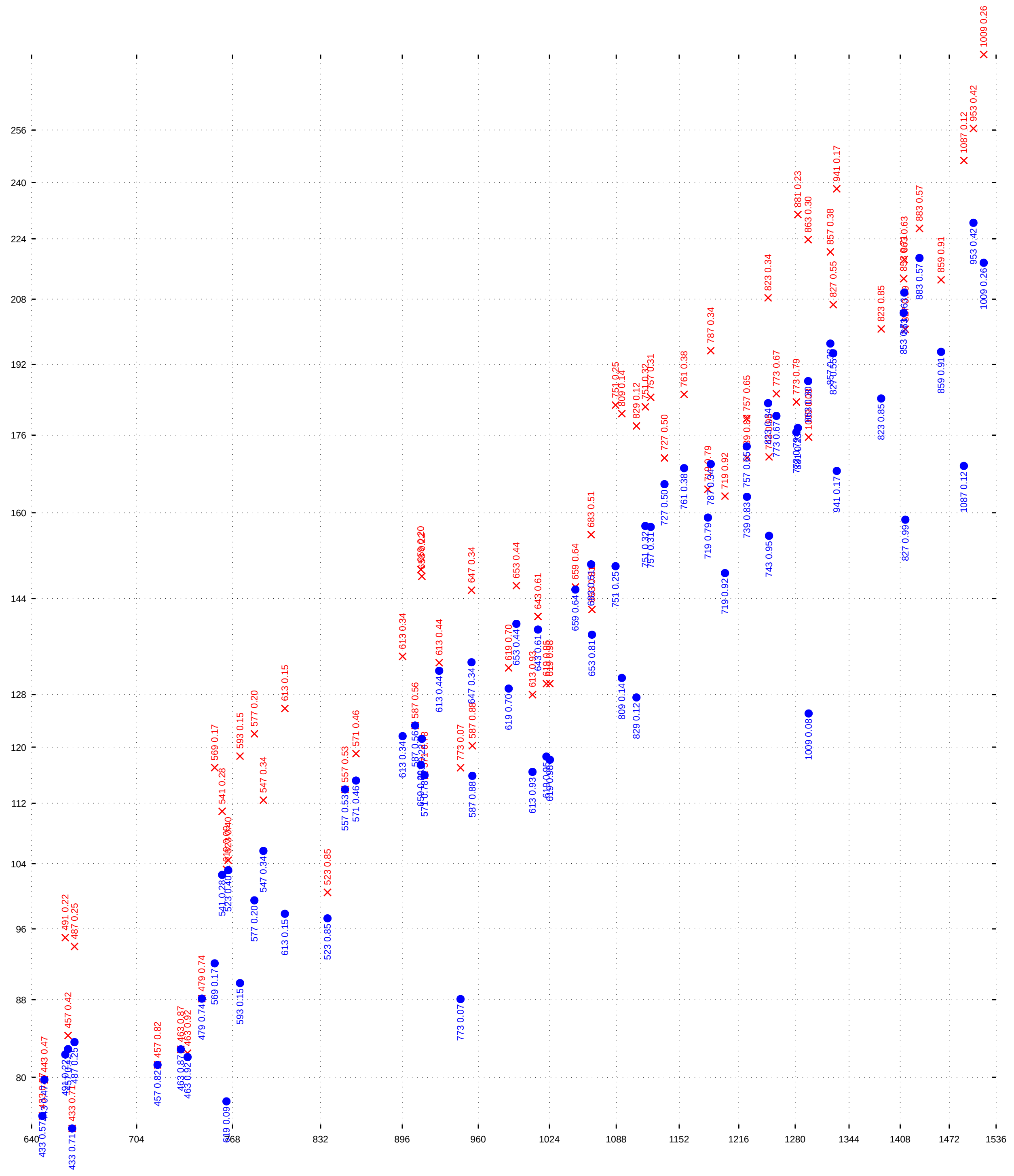


Figure 9: Estimated post-quantum security level against known attacks, assuming enumeration-based model of BKZ-β cost, accounting for real cost of memory. Horizontal axis: $p \log_{256} q$. Label: $p, w/p$. Log scale. Red: Ignoring hybrid attacks. Blue: Accounting for all attacks. See text for caveats regarding estimates.

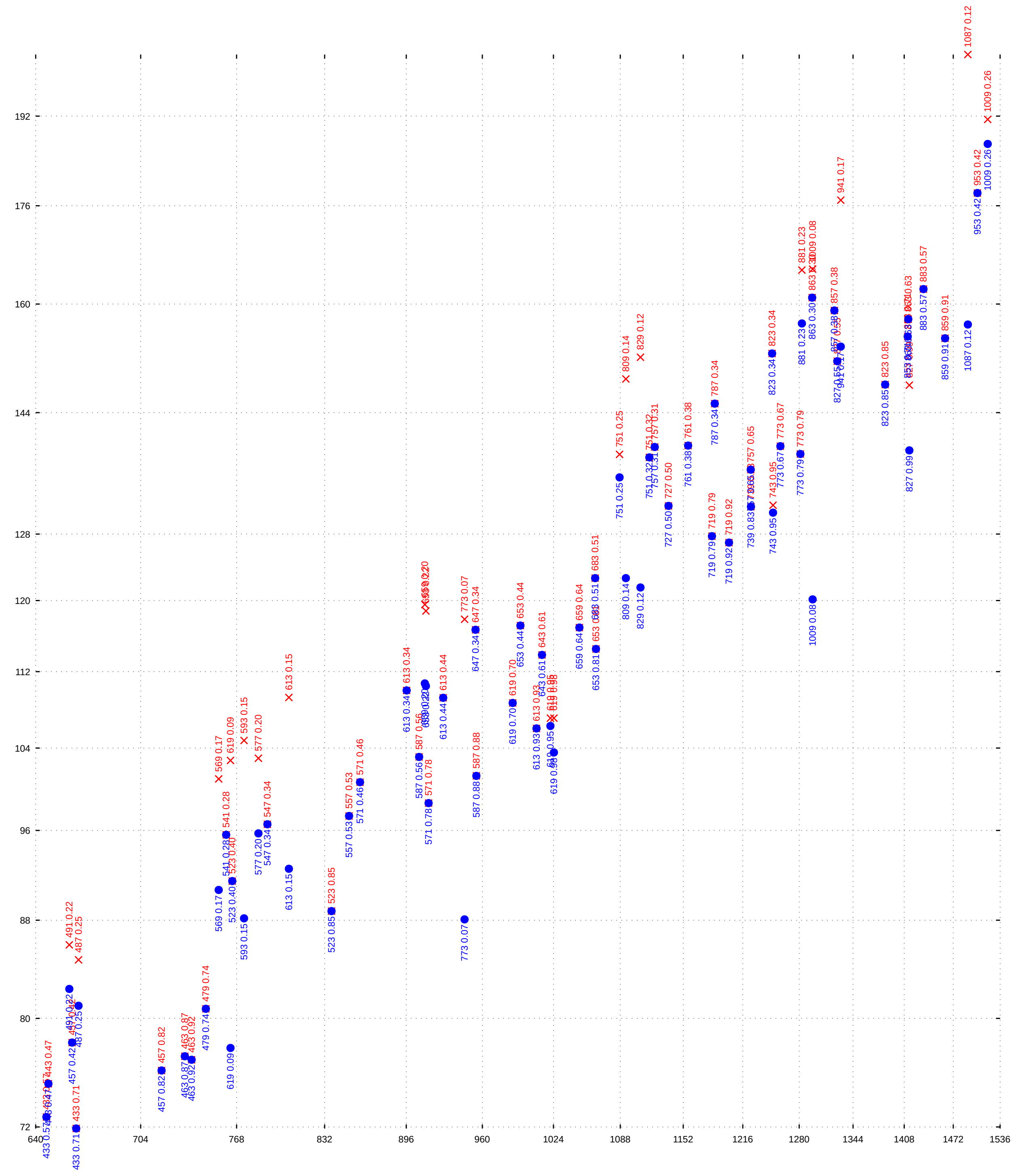


Figure 10: Estimated post-quantum security level against known attacks, assuming sieving-based model of BKZ-β cost, assuming free access to memory. Horizontal axis: $p \log_{256} q$. Label: $p, w/p$. Log scale. Red: Ignoring hybrid attacks. Blue: Accounting for all attacks. See text for caveats regarding estimates.

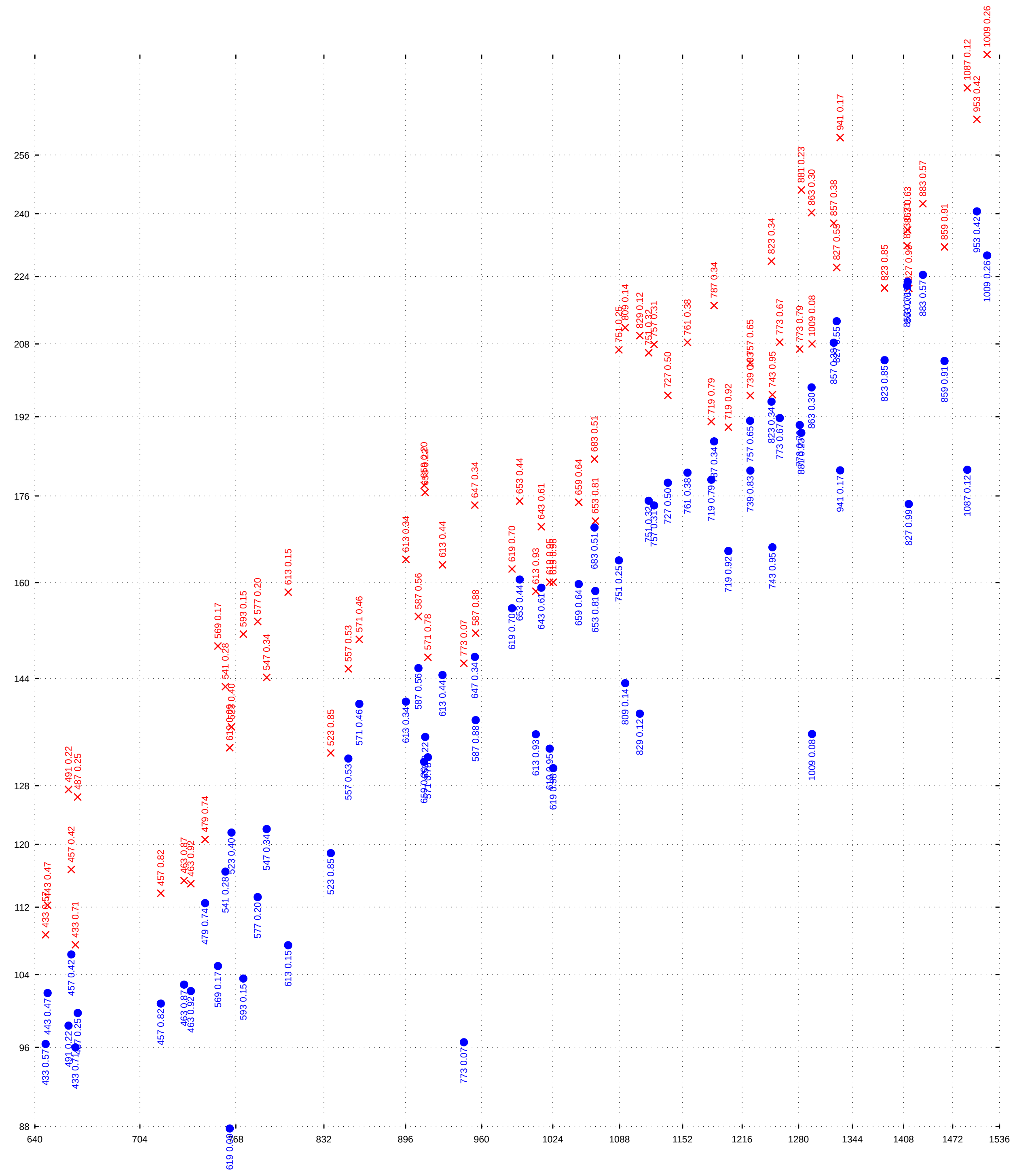


Figure 11: Estimated post-quantum security level against known attacks, assuming sieving-based model of BKZ-β cost, accounting for real cost of memory. Horizontal axis: $p \log_{256} q$. Label: $p, w/p$. Log scale. Red: Ignoring hybrid attacks. Blue: Accounting for all attacks. See text for caveats regarding estimates.