NTRU Prime: reducing attack surface at low cost

Daniel J. Bernstein[1], Chitchanok Chuengsatiansup[2], Tanja Lange[3], and Christine van Vredendaal[3]

[1] Department of Computer Science
University of Illinois at Chicago, Chicago, IL 60607–7045, USA
djb@cr.yp.to
[2] Laboratoire LIP, École Normale Supérieure de Lyon
46 Allée d'Italie 69364 Lyon Cedex 07, France
chitchanok.chuengsatiansup@ens-lyon.fr
[3] Department of Mathematics and Computer Science
Technische Universiteit Eindhoven, P.O. Box 513, 5600 MB Eindhoven, NL
tanja@hyperelliptic.org, c.v.vredendaal@tue.nl

Abstract. Several ideal-lattice-based cryptosystems have been broken by recent attacks that exploit special structures of the rings used in those cryptosystems. The same structures are also used in the leading proposals for post-quantum lattice-based cryptography, including the classic NTRU cryptosystem and typical Ring-LWE-based cryptosystems.

This paper (1) proposes NTRU Prime, which tweaks NTRU to use rings without these structures; (2) proposes Streamlined NTRU Prime, a public-key cryptosystem optimized from an implementation perspective, subject to the standard design goal of IND-CCA2 security; (3) finds high-security post-quantum parameters for Streamlined NTRU Prime; and (4) optimizes a constant-time implementation of those parameters. The resulting sizes and speeds show that reducing the attack surface has very low cost.

Keywords: post-quantum cryptography, public-key encryption, lattice-based cryptography, ideal lattices, NTRU, Ring-LWE, security, Soliloquy, Karatsuba, software implementation, vectorization, fast sorting

1 Introduction

This paper presents an efficient implementation of high-security **prime-degree large-Galois-group inert-modulus** ideal-lattice-based cryptography. "Prime

Author list in alphabetical order; see https://www.ams.org/profession/leaders/culture/CultureStatement04.pdf. This work was supported by the Netherlands Organisation for Scientific Research (NWO) under grant 639.073.005; by the Commission of the European Communities through the Horizon 2020 program under project number 645622 (PQCRYPTO) and project number 645421 (ECRYPT-CSA); and by the National Science Foundation under grant 1314919. Calculations were carried out on the Saber cluster of the Cryptographic Implementations group at Technische Universiteit Eindhoven. Permanent ID of this document: 99a9debfc18b7d6937a13bac4f943a2b2cd46022. Date: 2017.08.16.

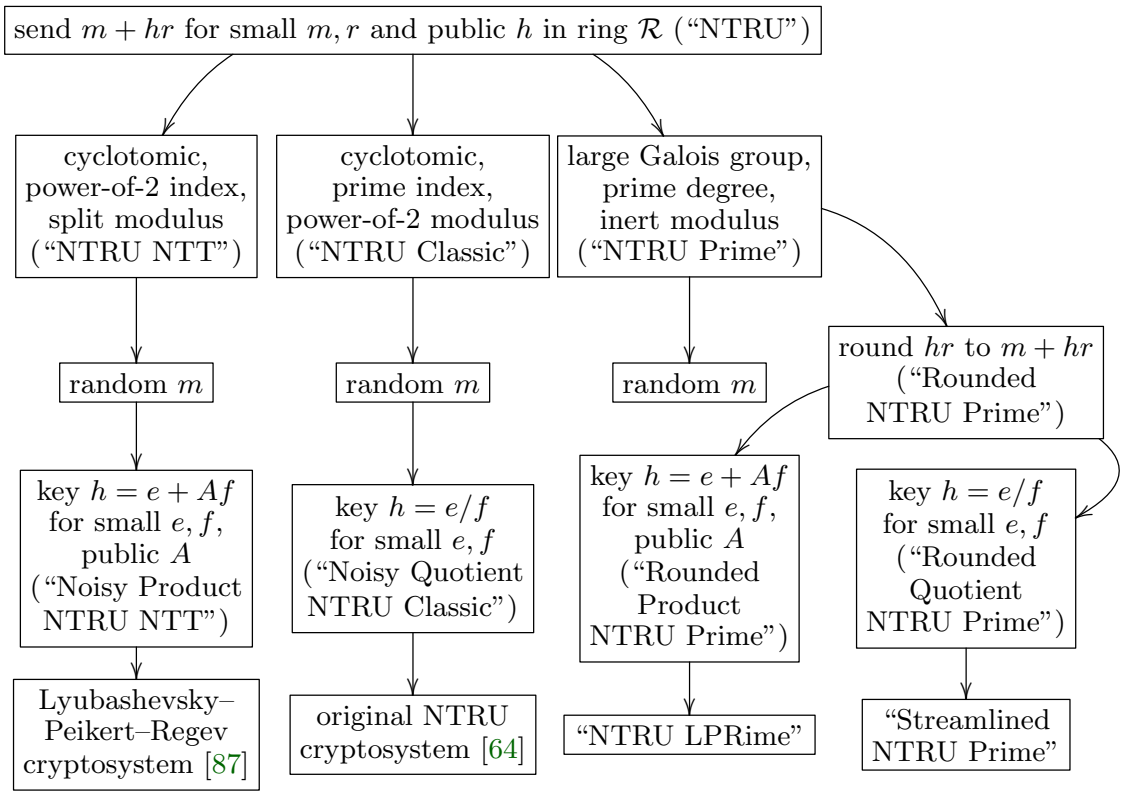

Fig. 1.1. Terminology in this paper for selected branches of the NTRU family tree. This paper introduces the NTRU Prime branch. Streamlined NTRU Prime is specified and analyzed in detail as a case study. See Section 3 for more options.

degree" etc. are three features that we recommend because they take various mathematical tools away from the attacker; see Appendix A. The reader can, if desired, skip the appendix in favor of the following short summary:

- "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 [64], and are excluded by our recommendation.
- "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 [5], and are excluded by our recommendation.
- "NTRU Prime": Fields of the form $(\mathbb{Z}/q)[x]/(x^p - x - 1)$, where p is prime, are used in this paper, and follow our recommendation.

Specifically, we use only 28682 cycles on one core of an Intel Haswell CPU for **constant-time** multiplication in the field $(\mathbb{Z}/4591)[x]/(x^{761} - x - 1)$.

We define a public-key cryptosystem "Streamlined NTRU Prime 4591^{761}" using this field, aiming for the standard design goal of IND-CCA2 security at the standard 2^{128} **post-quantum** security level. Streamlined NTRU Prime 4591^{761} uses just 59600 cycles for encryption (more precisely, "encapsulating" a 256-bit session key), and just 97452 cycles for decryption ("decapsulation").

Our public keys are field elements, easily squeezed into 1218 bytes. We explain how to further squeeze ciphertexts into just 1047 bytes. Obviously these sizes are not competitive with 256-bit ECC key sizes, but they are small enough for many applications.[4]

Streamlined NTRU Prime provides several implementation advantages and security-auditing advantages beyond the NTRU Prime choice of ring: for example, it eliminates the annoying possibility of "decryption failures" that appear in most lattice-based cryptosystems. Our security analysis indicates that Streamlined NTRU Prime 4591^{761} actually provides a large security margin beyond our target security level, compensating for potential progress in estimating the actual cost of lattice attacks.

To put our speed in perspective: Modern implementations [40,53] of the popular Curve25519 elliptic curve use more than 150000 Haswell cycles for scalar multiplication. However, one should not conclude that post-quantum lattice-based cryptography is faster than pre-quantum ECC. The total time for cryptography includes time to communicate keys and ciphertexts; lattice-based cryptography has much larger keys and ciphertexts than ECC.[5]

1.1. Comparison to previous multiplication speeds aiming for high security. Before our work, the state of the art in implementations of lattice-based cryptography was the November 2015 paper "Post-quantum key exchange: a new hope" [5] by Alkim, Ducas, Pöppelmann, and Schwabe, using about 40000 Haswell cycles for NTRU NTT multiplication. Most of the implementations before [5] are, in our view, obviously unsuitable for deployment because they access the CPU cache at secret addresses, taking variable time and allowing side-channel attacks.

We announced 51488 cycles for NTRU Prime multiplication in May 2016, in a preliminary version of this paper. Longa and Naehrig [83] announced 33000 cycles for NTRU NTT multiplication the same month. An update of [5] in August 2016 announced 31000 cycles for NTRU NTT multiplication.[6] We now announce 28682 cycles for NTRU Prime multiplication.

Like our paper, [5] and [83] target the Haswell CPU, require constant-time implementations, and aim for more than 2^{128} post-quantum security. Unlike our paper, [5] follows the tradition from NTRU and Ring-LWE [87] of using cyclotomic rings. More precisely, [5] is an example of Product NTRU NTT, using the ring $(\mathbb{Z}/q)[x]/(x^p+1)$ with $p = 1024$ and $q = 12289 = 12 \cdot 1024 + 1$.

[4] For example, our ciphertexts fit into the 1500-byte Ethernet MTU for plaintexts up to a few hundred bytes, avoiding the implementation hassle of packet fragmentation.

[5] If an operation takes 100000 cycles then one can imagine a typical quad-core 3GHz CPU completing 1 million operations in just 8 seconds. However, if each operation involves 1000 bytes of network data, then the data for 1 million operations will take 80 seconds to be transmitted through a typical 100Mbps network.

[6] Each forward NTT in the updated version of [5] takes 8448 cycles (compared to 10968 cycles in the first version, and 9100 cycles in [83, Table 1]). A reverse NTT takes 9464 cycles (compared to 12128 and 9300). The time for pointwise multiplication is not stated in [5] or [83] but can be extrapolated from [60] to take about 5000 cycles.

rec	constant	cycles	ring	technique	source
no	yes	11722	$(\mathbb{Z}/8192)[x]/(x^{701}-1)$	Karatsuba etc.	[71]
yes	yes	28682	$(\mathbb{Z}/4591)[x]/(x^{761}-x-1)$	Karatsuba etc.	this paper
no	yes	31000	$(\mathbb{Z}/12289)[x]/(x^{1024}+1)$	NTT	New Hope [5], [83]
no	no	<91056	$(\mathbb{Z}/2048)[x]/(x^{743}-1)$	sparse input	`ntruees743ep1` [77]

Table 1.1. Comparison of multiplication results. "Rec" means that the ring follows this paper's recommendation to reduce attack surface. "Constant" means that the software runs in constant time. "Cycles" is approximate multiplication time on an Intel Haswell. All rings are used in public-key cryptosystems designed for at least 2^{128} post-quantum security. The estimated pre-quantum security levels are 2^{248} for Streamlined NTRU Prime 4591^{761}; 2^{256} for `ntruees743ep1`; 2^{281} for New Hope; not stated in [71].

A disadvantage of requiring the lattice dimension p to be a power of 2, as in [5], is that security levels are quite widely separated. In [5] there is a claim of "94 bits of post quantum security" for one dimension-512 system; we are not aware of any dimension-512 system that is claimed today to reach the standard 2^{128} post-quantum security target. Jumping to the next power of 2, namely $p = 1024$, means at least doubling key sizes, ciphertext sizes, encryption time, etc. This severe discontinuity in the security-performance graph means that [5] is unable to offer any options truly comparable to the better-tuned $p = 743$ in "`ntruees743ep1`" (see [77]) or $p = 761$ in this paper. Of course one can view $p = 1024$ as an additional buffer against the possibility of improved attacks; but dimension is only one contributing factor to security, and size does matter.

The conventional wisdom is that, despite the large p, rings of the type used in [5] are particularly efficient. These rings allow multiplication at the cost of three "number-theoretic transforms" (NTTs), i.e., fast Fourier transforms over finite fields, with only a small overhead for "pointwise multiplication". This multiplication strategy relies critically on choosing an NTT-friendly polynomial such as $x^{1024}+1$ and choosing an NTT-friendly prime such as 12289.

Tweaking the polynomial and prime, as we recommend, would make the NTTs several times more expensive. A typical NTT-based method to multiply in, e.g., $(\mathbb{Z}/8819)[x]/(x^{1021}-x-1)$ would replace $x^{1021}-x-1$ with $x^{2048}-1$, and would also replace 8819 with two or three NTT-friendly primes. The conventional wisdom therefore implies that we pay a very large penalty for requiring a large Galois group (NTT-friendly polynomials always have small Galois groups) and an inert modulus (NTT-friendly primes are never inert).

We do much better by scrapping the NTTs and multiplying in a completely different way. The May 2016 version of this paper presented details of a combination of several layers of Karatsuba's method and Toom's method. This approach does not need NTT-friendly polynomials, and it does not need NTT-friendly primes. (The approach is like NTTs in that a significant part of the work is for separately transforming each input, allowing transforms to be skipped in many settings.) We now do even better by tweaking various details, as explained later in this paper; in particular, our current software uses purely Karatsuba's

method. The resulting multiplication speed is slightly faster than in [5] and [83], and the sizes are smaller.

We are not saying that the NTRU Prime rings have *zero* cost. Last month Hülsing, Rijneveld, Schanck, and Schwabe [71] announced 11722 cycles for NTRU Classic multiplication, specifically multiplication in the ring $(\mathbb{Z}/q)[x]/(x^p - 1)$ with $p = 701$ and $q = 8192$, again using a combination of several layers of Karatsuba's method and Toom's method. The power-of-2 moduli in NTRU Classic avoid the cost of reducing modulo medium-size primes. These moduli force a moderate discontinuity in the security-performance graph[7] but it seems likely that taking $(\mathbb{Z}/q)[x]/(x^p - 1)$ with *prime* q would be slightly faster than NTRU Prime at every security level.

1.2. Priority dates and additional followup work. Our recommendation to switch lattice-based cryptography to prime-degree large-Galois-group inert-modulus lattice-based cryptography was announced in February 2014.

In 2016, the NTRU authors posted a draft [65] that they had circulated at Crypto 1996. Page 21 of the draft says "One could also consider variants of standard NTRU by using rings such as $A = \mathbb{Z}[X]/(X^N - X - 1)$. This would slow computations somewhat, while providing greater mixing of the coefficients." Our announcement was published earlier; pinpoints stronger mathematical reasons to use these rings (not merely "providing greater mixing" but also taking subfields and automorphisms away from the attacker); adds the further requirement to use quotient fields; and is a recommendation, not merely a "could".

We posted a preliminary version of this paper in May 2016, as mentioned above. That version included, among other things, an improved cryptosystem, a detailed security analysis, and new performance results showing that the NTRU Prime ring recommendation is compatible with high speed. All of this was written independently of the above quote from [65].

Lyubashevsky, in response to the possibility that "some rings could give rise to more difficult instances of Ring-SIS and Ring-LWE than other rings", introduced a signature system [85] in August 2016 for which a polynomial-time attack would imply a polynomial-time attack against similar problems for all rings. Rosca, Sakzad, Steinfeld, and Stehlé introduced an encryption system [103] in June 2017 with similar properties. The concrete performance of these systems is unclear.

In June 2017, Bos–Ducas–Kiltz–Lepoint–Lyubashevsky–Schanck–Schwabe–Stehlé [28] announced 119652 cycles for encapsulation and 125736 cycles for decapsulation using a new public-key cryptosystem "Kyber". (Preliminary speeds announced in January 2017 [9] were slower.) This system uses Module-LWE [82] with three elements of $(\mathbb{Z}/7681)/(x^{256} + 1)$, for a total of 768 coefficients. Ciphertexts occupy 1184 bytes.

In March 2017, Peikert, Regev, and Stephens–Davidowitz [97] argued briefly that "one might wish to use Ring-LWE over non-Galois number fields". The

[7] The security level in [71] seems somewhat lower than the security level of Streamlined NTRU Prime 4591^{761}. Taking a larger p in [71] would require jumping to $q = 16384$, and the resulting ciphertext expansion seems likely to outweigh any small speed gap.

argument is essentially one of the arguments from this paper, without credit. The main result of [97] is a worst-case-to-average-case reduction; see Appendix C.

Acknowledgements We wish to thank John Schanck for detailed discussion of the security of NTRU and for suggesting the "transitional security" terminology; Dan Shepherd and Manuel Pancorbo Castro for pointing out a stronger bound for Theorem 2.1; and Sean Parkinson for helpful comments.

2 Streamlined NTRU Prime: an optimized cryptosystem

This section specifies "Streamlined NTRU Prime", a public-key cryptosystem. The next section compares Streamlined NTRU Prime to alternatives.

We emphasize that Streamlined NTRU Prime is designed for the standard goal of IND-CCA2 security, i.e., security against adaptive chosen-ciphertext attacks. A server can reuse a public key any number of times, amortizing the costs of key generation and key distribution. The cost of setting up a new session key, *including* post-quantum server authentication, is then just one encryption for the client and one decryption for the server. This gives Streamlined NTRU Prime important performance advantages over unauthenticated key-exchange mechanisms such as [5]; see Appendix E for a precise comparison.

We are submitting our complete implementation to eBACS [22] for benchmarking. However, we caution potential users that many details of Streamlined NTRU Prime were first published in May 2016 and still require careful security review. We have not limited ourselves to the minimum changes that would be required to switch to NTRU Prime from an existing version of the NTRU public-key cryptosystem; we have taken the opportunity to rethink and reoptimize all of the details of NTRU from an implementation and security perspective. We recommend NTRU Prime, but it is too early to recommend Streamlined NTRU Prime.

2.1. Parameters. Streamlined NTRU Prime is actually a family of cryptosystems parametrized by positive integers (p, q, t) subject to the following restrictions: p is a prime number; q is a prime number; $t \geq 1$; $p \geq 3t$; $q \geq 32t + 1$; $x^p - x - 1$ 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 $\{-1, 0, 1\}$. We refer to a small element as **t-small** if exactly $2t$ of its coefficients are nonzero, i.e., its Hamming weight is $2t$.

Our case study in this paper is Streamlined NTRU Prime 4591^{761}. This specific cryptosystem has parameters $p = 761$, $q = 4591$, and $t = 143$. The following subsections specify the algorithms for general parameters but the reader may wish to focus on these particular parameters. Figures Z.1 and Z.2 show complete algorithms for key generation, encapsulation, and decapsulation in Streamlined NTRU Prime 4591^{761}, using the Sage [104] computer-algebra system.

2.2. Key generation. The receiver generates a public key as follows:

- Generate a uniform random small element $g \in \mathcal{R}$. Repeat this step until g is invertible in $\mathcal{R}/3$.
- Generate a uniform random t-small element $f \in \mathcal{R}$. (Note that f is nonzero and hence invertible in $\mathcal{R}/q$, since $t \geq 1$.)
- 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 $\mathcal{R}/q$.)
- Encode h as a string $\underline{h}$. The public key is $\underline{h}$.
- Save the following secrets: f in $\mathcal{R}$; and $1/g$ in $\mathcal{R}/3$.

See `keygen` in Figure Z.2.

The encoding of public keys as strings is another parameter for Streamlined NTRU Prime. Each element of $\mathbb{Z}/q$ is traditionally encoded as $\lceil \log_2 q \rceil$ bits, so the public key is traditionally encoded as $p\lceil \log_2 q \rceil$ bits. If q is noticeably smaller than a power of 2 then one can easily compress a public key by merging adjacent elements of $\mathbb{Z}/q$, with a lower limit of $p \log_2 q$ bits. For example, 5 elements of $\mathbb{Z}/q$ for $q = 4591$ are easily encoded together as 8 bytes, saving 1.5% compared to separately encoding each element as 13 bits, and 20% compared to separately encoding each element as 2 bytes. See Figure Z.1 for further encoding details.

2.3. Encapsulation. Streamlined NTRU Prime is actually a "key encapsulation mechanism" (KEM). This means that the sender takes a public key as input and produces a ciphertext and session key as output. See Section 3.5 for comparison to older notions of public-key encryption, and for an explanation of how to use a KEM to encrypt a user-provided message.

Specifically, the sender generates a ciphertext as follows:

- Decode the public key $\underline{h}$, obtaining $h \in \mathcal{R}/q$.
- Generate a uniform random t-small element $r \in \mathcal{R}$.
- Compute $hr \in \mathcal{R}/q$.
- Round each coefficient of hr, viewed as an integer between $-(q-1)/2$ and $(q-1)/2$, to the nearest multiple of 3, producing $c \in \mathcal{R}$. (If $q \in 1 + 3\mathbb{Z}$, as in our case study $q = 4591$, then each coefficient of c is in $\{-(q-1)/2, \ldots, -6, -3, 0, 3, 6, \ldots, (q-1)/2\}$. If $q \in 2 + 3\mathbb{Z}$ then each coefficient of c is in $\{-(q+1)/2, \ldots, -6, -3, 0, 3, 6, \ldots, (q+1)/2\}$.)
- Encode c as a string $\overline{c}$.
- Hash r, obtaining a left half C ("key confirmation") and a right half K.
- The ciphertext is the concatenation $C\,\overline{c}$. The session key is K.

See `encapsulate` in Figure Z.2.

The hash function for r is another parameter for Streamlined NTRU Prime. We encode r as a byte string by adding 1 to each coefficient, obtaining an element of $\{0, 1, 2\}$ encoded as 2 bits in the usual way, and then packing 4 adjacent coefficients into a byte, consistently using little-endian form. See `encodeZx` in Figure Z.1. We hash the resulting byte string with SHA-512, obtaining a 256-bit key confirmation C and a 256-bit session key K.

The encoding of ciphertexts c as strings $\overline{c}$ is another parameter for Streamlined NTRU Prime. This encoding can be more compact than the encoding of public keys because each coefficient of c is in a limited subset of $\mathbb{Z}/q$. Concretely, for $q = 4591$ and $p = 761$, we use 32 bits for each 3 coefficients of c and a total

of 8120 bits (padded to a byte boundary) for $\overline{c}$, saving 16% compared to the size of a public key, 18% compared to separately encoding each element of $\mathbb{Z}/q$ as 13 bits, and 33% compared to separately encoding each element of $\mathbb{Z}/q$ as 2 bytes. See `encoderoundedRq` in Figure Z.1. Key confirmation adds 256 bits to ciphertexts.

2.4. Decapsulation. The receiver decapsulates a ciphertext $C\,\overline{c}$ as follows:

- Decode $\overline{c}$, obtaining $c \in \mathcal{R}$.
- Multiply by $3f$ in $\mathcal{R}/q$.
- 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 $1/g$ in $\mathcal{R}/3$.
- Lift e/g in $\mathcal{R}/3$ to a small polynomial $r' \in \mathcal{R}$.
- Compute c', C', K' from r' as in encapsulation.
- If r' is t-small, $c' = c$, and $C' = C$, then output K'. Otherwise output False.

See `decapsulate` in Figure Z.2.

If $C\,\overline{c}$ is a legitimate ciphertext then c is obtained by rounding the coefficients of hr to the nearest multiples of 3; i.e., $c = m + hr$ in $\mathcal{R}/q$, where m is small. All coefficients of the polynomial $3fm + gr$ in $\mathcal{R}$ are in $[-16t, 16t]$ by Theorem 2.1 below, and thus in $[-(q-1)/2, (q-1)/2]$ since $q \geq 32t+1$. Viewing each coefficient of $3fc = 3fm + gr$ 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 $e/g = r$ in $\mathcal{R}/3$. Lifting now produces exactly r since r is small; i.e., $r' = r$. Hence $(c', C', K') = (c, C, K)$. Finally, $r' = r$ is t-small, $c' = c$, and $C' = C$, so decapsulation outputs $K' = K$, the same session key produced by encapsulation.

Theorem 2.1 *Fix integers $p \geq 3$ and $t \geq 1$. 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 $2t$ nonzero coefficients. Then $3fm + gr \bmod x^p - x - 1$ has each coefficient in the interval $[-16t, 16t]$.*

3 The design space of lattice-based encryption

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. 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.

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.

3.1. 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 $\mathcal{R}/q$.

The choices of P mentioned in Section 1 include $x^p - 1$ for prime p (NTRU Classic); $x^p + 1$ where p is a power of 2 (NTRU NTT); and $x^p - x - 1$ for prime p (NTRU Prime). Choices of q include powers of 2 (NTRU Classic); split primes q (NTRU NTT); and inert primes q (NTRU Prime).

Of course, Streamlined NTRU Prime makes the NTRU Prime choices here. Most of the optimizations in Streamlined NTRU Prime can also be applied to other choices of P and q, with a few exceptions noted below.

3.2. The public key. The receiver's public key, which we call h, is an element of $\mathcal{R}/q$. It is invertible in $\mathcal{R}/q$ but has no other obvious public structure.

3.3. 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 additional 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.

The multiplication of h by r is the main bottleneck in encryption in all of these systems and the main target of our implementation work; see Section 6. 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 3.5.

In the original NTRU specification [64], 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 specifications such as [63] 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 the original NTRU system 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 $m + hr$; see, e.g., Section 2.3. Second, the fact that m is determined by r simplifies protection against chosen-ciphertext attacks; see Section 3.5.

[94, 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 [10] in the context of a symmetric-key construction, was used in [7] to construct another public-key encryption system, and was further studied in [27] and [6]. 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.

3.4. 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.

The 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.

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 3.5.

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 $e + Af$, where e and f are secret small polynomials. Here $A \in \mathcal{R}/q$ 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 [5]) 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 [87] (originally in talk slides in 2010), sends $d + Ar$ as additional ciphertext along with $m + hr + M$, where d is another small polynomial, and M is a polynomial consisting of solely 0 or $\lfloor q/2 \rfloor$ in each position. The receiver computes $(m + hr + M) - (d + Ar)f = M + m + er - df$, and rounds to 0 or $\lfloor q/2 \rfloor$ in each position, obtaining M. Note that $m + er - df$ is small, since all of m, e, r, d, f 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 simply summing batches of three coefficients [100], before adding M and m. Rounded Product NTRU rounds $hr + M$ to obtain $m + hr + M$, rounds Ar to obtain $d + Ar$, and (to similarly reduce key size) rounds Af to obtain $e + Af$. Decryption continues to work even if $m + hr + M$ is compressed to two bits per coefficient. "NTRU LPRime" is an example of Rounded Product NTRU Prime in which r is chosen deterministically as a hash of M.

A disadvantage of Product NTRU is that r is used twice, exposing approximations to both Ar 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 [74] 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 [114], 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.[8] We see no way to simultaneously

[8] Peikert claimed in [96], 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

avoid both types of complications. Since Quotient NTRU has a much longer history, we have opted to present details of Streamlined NTRU Prime, an example of Quotient NTRU Prime.

3.5. Padding, KEMs, and the choice of q. In Streamlined NTRU Prime we use the modern "KEM+DEM" approach introduced by Shoup; see [109]. 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., [110]) and in practice (see, e.g., [90]), 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 [109] (also called "Simple 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

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.

NTRU-based KEMs we have found in the literature (e.g., [113] and [105]) 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 3.3) 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 [46, Section 6] by Dent, backed by a tight security reduction [46, 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 $q \geq 32t + 1$. Recall that decryption sees $3fm+gr$ in $\mathcal{R}/q$ and tries to deduce $3fm+gr$ in $\mathcal{R}$; the condition $q \geq 32t + 1$ guarantees that this works, since each coefficient of $3fm + gr$ in $\mathcal{R}$ is between $-(q-1)/2$ and $(q-1)/2$ by Theorem 2.1. Taking different shapes of m, r, f, g, or changing the polynomial $P = x^p - x - 1$, would change the bound $32t + 1$; for example, replacing g by $1 + 3G$ would change $32t + 1$ into $48t + 3$.

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 [68] and, for an analogous attack on code-based cryptography using QC-MDPC codes, [61]. 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.

3.6. The shape of small polynomials. As noted in Section 3.3, the coefficients of m are chosen from the limited range $\{-1, 0, 1\}$. The NTRU literature [64,70,62,63] 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 3.4) 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 3.5), saving space and time. Beware, however, that if t is *too* small then there are attacks; see our security analysis in Section 4.

We keep the requirement that r have Hamming weight $2t$, and keep the requirement that these $2t$ 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.

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 3.5), and these changes do not expand our *bound* on the size of the coefficients of fm and gr.

Elsewhere in the literature on lattice-based cryptography one can find larger coefficients: consider, e.g., the quinary polynomials in [50], and the even wider range in [5]. In [114], 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 3.5), 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 attack in [30].

4 Pre-quantum security of Streamlined NTRU Prime

In this section we adapt existing *pre-quantum* NTRU attack strategies to the context of Streamlined NTRU Prime and quantify their effectiveness. In particular, 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.

Underestimating attack cost can *damage* security, for reasons explained in [23, full version, Appendix B.1.2], so we prefer to use accurate cost estimates. However, accurately evaluating the cost of lattice attacks is generally quite difficult. The literature very often explicitly resorts to underestimates. Comprehensively fixing this problem is beyond the scope of this paper, but we have started work in this direction, as illustrated by Appendix M. At the same time it is clear that the best attack algorithms known today are much better than the best

attack algorithms known a few years ago, so it is unreasonable to expect that the algorithms have stabilized. We plan to periodically issue updated security estimates to reflect ongoing work.

4.1. Meet-in-the-middle attack. Odlyzko's meet-in-the-middle attack [69,67] on NTRU 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 NTRU Classic and $\binom{p}{2t}2^{2t}$ in Streamlined NTRU Prime. A first estimate is that the number of loops in the algorithm is the square root of the number of choices of f. However, this estimate does not account for equivalent keys. In NTRU Classic, a key (f, g) is equivalent to all of the rotated keys $(x^i f, x^i g)$ and to the negations $(-x^i f, -x^i g)$, and the algorithm succeeds if it finds any of these rotated keys. The $2p$ rotations and negations are almost always distinct, producing a speedup factor very close to $\sqrt{2p}$.

The structure of the NTRU Prime ring is less friendly to this attack. Say f has degree $p - c$; typically c is around $p/2t$, since there are $2t$ 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 $p/2t$ divisions by x to stay within $\mathcal{F}$, for a total of only about p/t equivalent keys, or $2p/t$ when negations are taken into account. We have confirmed these estimates with experiments.

One could modify the attack to use a larger set $\mathcal{F}$, but this seems to lose more than it gains. Furthermore, similar wraparounds for g compromise the effectiveness of the collision function. To summarize, the extra term in $x^p - x - 1$ seems to increase the attack cost by a factor around $\sqrt{t}$, compared to NTRU Classic; i.e., the rotation speedup is only around $\sqrt{2p/t}$ rather than $\sqrt{2p}$.

On the other hand, some keys f allow considerably more rotations. We have decided to assume a speedup factor of $\sqrt{2(p-t)}$, since we designed some pathological polynomials f with that many (not consecutive) rotations in the set. For random r the speedup is much smaller. This means that the number of loops before this attack is expected to find f is bounded by

$$L = \sqrt{\binom{p}{2t}2^{2t}} \Big/ \sqrt{2(p-t)}. \tag{1}$$

In each loop, t vectors of size p are added and their coefficients are reduced modulo q. We thus estimate the attack cost as Lpt. The storage requirement of the attack is approximately $L \log_2 L$. We can reduce this storage by applying collision search to the meet-in-the-middle attack (see [92,116]). In this case we can reduce the storage capacity by a factor s at the expense of increasing the running time by a factor $\sqrt{s}$.

4.2. Streamlined NTRU Prime lattice. As with NTRU we can embed the problem of recovering the private keys f, g into a lattice problem. Saying $3h = g/f$ in $\mathcal{R}/q$ is the same as saying $3hf + qk = g$ in $\mathcal{R}$ for some polynomial k; in other words, there is a vector (k, f) of length $2p$ such that

$$\begin{pmatrix} k & f \end{pmatrix} \begin{pmatrix} qI & 0 \\ H & I \end{pmatrix} = \begin{pmatrix} k & f \end{pmatrix} B = \begin{pmatrix} g & f \end{pmatrix},$$

where H is a matrix with the i'th vector corresponding to $x^i \cdot 3h \bmod x^p - x - 1$ and I is the $p \times p$ identity matrix. We will call B the *Streamlined NTRU Prime public lattice basis*. This lattice has determinant q^p. The vector (g, f) has norm at most $\sqrt{2p}$. The Gaussian heuristic states that the length of the shortest vector in a random lattice is approximately $\det(B)^{1/(2p)}\sqrt{\pi ep} = \sqrt{\pi epq}$, which is much larger than $\sqrt{2p}$, so we expect (g, f) to be the shortest nonzero vector in the lattice.

Finding the secret keys is thus equivalent to solving the Shortest Vector Problem (SVP) for the Streamlined NTRU Prime public lattice basis. The fastest currently known method to solve SVP in the NTRU public lattice is the hybrid attack, which we discuss below.

A similar lattice can be constructed to instead try to find the input pair (m, r). However, there is no reason to expect the attack against (m, r) to be easier than the attack against (g, f): r has the same range as f, and m has essentially the same range as g. Recall that Streamlined NTRU Prime does not have the original NTRU problem of leaking $m(1)$. There are occasional boundary constraints on m (see Section 3.3), and there is also an $\mathcal{R}/3$ invertibility constraint on g, but these effects are minor.

4.3. Hybrid security. The best known attack against the NTRU lattice is the hybrid lattice-basis-reduction-and-meet-in-the-middle attack described in [67]. The attack works in two phases: the reduction phase and the meet-in-the-middle phase.

Applying lattice-basis-reduction techniques will mostly reduce the middle vectors of the basis [106]. Therefore the strategy of the reduction phase is to apply lattice-basis reduction, for example BKZ 2.0 [38], to a submatrix B' of the public basis B. We then get a reduced basis $T = UBY$:

$$\left(\begin{array}{c|c|c} I_w & 0 & 0 \\ \hline 0 & U' & 0 \\ \hline 0 & 0 & I_{w'} \end{array}\right) \cdot \left(\begin{array}{c|c|c} qI_w & 0 & 0 \\ \hline * & B' & 0 \\ \hline * & * & I_{w'} \end{array}\right) \cdot \left(\begin{array}{c|c|c} I_w & 0 & 0 \\ \hline 0 & Y' & 0 \\ \hline 0 & 0 & I_{w'} \end{array}\right) = \left(\begin{array}{c|c|c} qI_w & 0 & 0 \\ \hline * & T' & 0 \\ \hline * & * & I_{w'} \end{array}\right)$$

Here Y is orthonormal and T' is again in lower triangular form.

In the meet-in-the-middle phase we can use a meet-in-the-middle algorithm to guess options for the last w' coordinates of the key by guessing halves of the key and looking for collisions. If the lattice basis was reduced sufficiently in the first phase, a collision resulting in the private key will be found by applying a rounding algorithm to the half-key guesses. More details on how to do this can be found in [67].

To estimate the security against this attack we adapt the analysis of [63] to the set of keys that we use in Streamlined NTRU Prime. Let w be the dimension of I_w and w' be the dimension of $I_{w'}$. For a sufficiently reduced basis the meet-in-the-middle phase will require on average

$$-\frac{1}{2}\left(\log_2(2(p-t)) + \sum_{0\le a\le \min\{(\}2t,w')} 2^a \binom{w'}{a} v(a)\log_2(v(a))\right) \tag{2}$$

work, where the $\log_2(2(p-t))$ term accounts for equivalent keys and

$$v(a) = \frac{2^{2t-a}\binom{p-w'}{2t-a}}{2^{2t}\binom{p}{2t}} = \frac{2^{-a}\binom{p-w'}{2t-a}}{\binom{p}{2t}}. \tag{3}$$

The quality of a basis after lattice reduction can be measured by the Hermite factor $\delta = ||\mathbf{b_1}||/\det(B)^{1/p}$. Here $||\mathbf{b_1}||$ is the length of the shortest vector among the rows of B. To be able to recover the key in the meet-in-the-middle phase, the $(2p-w-w')\times(2p-w-w')$ matrix T' has to be sufficiently reduced. For given w and w' this is the case if the lattice reduction reaches the required value of δ. This Hermite factor has to satisfy

$$\log_2(\delta) \le \frac{(p-w)\log_2(q)}{(2p-(w+w'))^2} - \frac{1}{2p-(w'+w)}. \tag{4}$$

We use the BKZ 2.0 simulator of [38] to determine the best BKZ 2.0 parameters, specifically the "block size" β and the number of "rounds" n, needed to reach a root Hermite factor δ. To get a concrete security estimate of the work required to perform BKZ-2.0 with parameters β and n we use the conservative formula determined by [63] from the experiments of [39]:

$$\text{Estimate}(\beta,p,n) = 0.000784314\beta^2 + 0.366078\beta - 6.125 + \log_2(p\cdot n) + 7. \tag{5}$$

This estimate and the underlying experiments rely on "enumeration"; see Appendix M for a comparison to "sieving". This analysis also assumes that the probabily of two halves of the key colliding is 1. We will also conservatively assume this, but a more realistic estimate can be found in [118]. Using these estimates we can determine the optimal w and w' to attack a parameter set and thereby estimate its security.

Lastly we note that this analysis is easily adaptable to generalizing the coefficients to be in the set $\{-d,-(d-1),\dots,d-1,d\}$ by replacing base 2 in the exponentiations in Equations 1, 2 and 3 with $2d$. In this case however the range of t, by a generalization of Theorem 2.1, decreases to $q \ge 16(d^3+d^2)t$.

4.4. Algebraic attacks. The attack strategy of Ding [47], Arora–Ge [8], and Albrecht–Cid–Faugère–Fitzpatrick–Perret [4] 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 such as NTRU and NTRU Prime provide far less data to the attacker. When these attacks are adapted to cryptosystems that

Algorithm 1: Determine parameter sets for security level above ℓ.

Input: Upper bound q_b for q, range $[p_1, p_2]$ for p, lower bound ℓ for security level

Result: Viable parameters p, q and t with security level λ.

$p \leftarrow p_1 - 1$ (the prime we are currently investigating)

while $p \leq p_2$ **do**

  $p \leftarrow \text{nextprime}(p)$

  $Q \leftarrow \text{viableqs}(p, q_b)$

  for $q \in Q$ **do**

    $t \leftarrow \min\{\lfloor (q-1)/32 \rfloor, \lfloor p/3 \rfloor\}$

    $\lambda_1 \leftarrow \text{mitmcosts}(p, t)$

    if $\lambda_1 \geq \ell$ **then**

      Find w, w', β, n such that BKZ-2.0 costs are approximately equal to meet-in-the-middle costs in the hybrid attack.

      $\lambda_2 \leftarrow \max\{\text{hybridbkzcost}, \text{hybridmitmcost}\}$

      return $p, q, t, \min\{\lambda_1, \lambda_2\}$

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 [4, Theorem 7] and [86, Case Study 1].

5 Parameters

Algorithm 1 searches for (p, q, t, λ), where λ is Section 4's estimate of the *pre-quantum* security level for parameters (p, q, t). For example, we used Algorithm 1 to find our recommended parameters $(p, q, t) = (761, 4591, 143)$ with estimated pre-quantum security 2^{248}. We expect *post-quantum* security levels to be somewhat lower (e.g., [80] saves a factor 1.1 in the best known asymptotic SVP exponents), and lattice security remains a tricky research topic, but there is a comfortable security margin above our target 2^{128}.

In the parameter generation algorithm the subroutine nextprime(i) returns the first prime number $>i$. The subroutine viableqs(p, q_b) returns all primes q larger than p and smaller than q_b for which it holds that $x^p - x - 1$ is irreducible in $(\mathbb{Z}/q)[x]$. The subroutine mitmcosts uses the estimates from Equation (1) to determine the bitsecurity level of the parameters against a straightforward meet-in-the-middle attack. To find w, w', β, n we set w to the hybridbkzcost of the previous iteration (initially 0) and do a binary search for w' such that the two phases of the hybrid attack are of equal cost. For each w' we determine the Hermite factor required with Equation (4), use the BKZ-2.0 simulator to determine the optimal β and n to reach the required Hermite factor and use Equations (5) and (2) to determine the hybridbkzcost and hybridmitmcost.

Note that this algorithm outputs the largest value of t such that there are no decryption failures according to Theorem 2.1 and that no more than 2/3 of

the coefficients of f are set. Experiments show that decreasing t to t_1 linearly decreases the security level by approximately $t - t_1$.

The results of the algorithm for $q_b = 20000$, $[p_1, p_2] = [500, 950]$, and $\ell = 128$ can be found in Appendix P.

6 Vectorized polynomial multiplication

Our optimized implementation of Streamlined NTRU Prime 4591^{761} takes a total of 157052 Haswell cycles for encapsulation and decapsulation. Almost 75% of this time is spent on four multiplications of polynomials modulo $x^p - x - 1$. (Another 15% is spent on generating a t-small element; see Appendices S and T.) This section explains how we perform each multiplication in under 30000 cycles.

6.1. Sizes of inputs and intermediate results. Three of the multiplications are in $\mathcal{R}/q = (\mathbb{Z}/q)[x]/(x^p - x - 1)$. Specifically, encapsulation multiplies the public key h by r; decapsulation multiplies the ciphertext c by $3f$, and later multiplies h by r'.

Each element of $\mathbb{Z}/q$ is conventionally represented as an element of $\mathbb{Z}$ between 0 and $q-1$. Each element of $\mathcal{R}/q$ is then represented as an element of $\mathbb{Z}[x]$ with p coefficients between 0 and $q-1$. The product of two such elements in $\mathbb{Z}[x]$ has coefficients between 0 and $p(q-1)^2$. The product in $\mathcal{R} = \mathbb{Z}[x]/(x^p - x - 1)$ has coefficients between 0 and $2p(q-1)^2$; see the proof of Theorem 2.1. Reducing these coefficients modulo q produces the desired product in $\mathcal{R}/q$.

A standard improvement, "signed digits" or "signed coefficients", is to instead represent each element of $\mathbb{Z}/q$ as an element of $\mathbb{Z}$ between $-(q-1)/2$ and $(q-1)/2$. This is an improvement because the product in $\mathbb{Z}[x]$ then has coefficients between $-p(q-1)^2/4$ and $p(q-1)^2/4$, an interval just half as long as before. This fits each coefficient into fewer bits, and allows the coefficient arithmetic to use less precision.

We use signed digits but go much further by observing that, in NTRU and its variants, each multiplication has an input that is guaranteed to be small. For example, r in Streamlined NTRU Prime has coefficients in $\{-1, 0, 1\}$, so the product in $\mathbb{Z}[x]$ has coefficients between $-p(q-1)/2$ and $p(q-1)/2$, a much smaller interval than before. Even better, r has Hamming weight $2t$, so the product in $\mathbb{Z}[x]$ has coefficients between $-t(q-1)$ and $t(q-1)$, and the product in $\mathcal{R}$ has coefficients between $-2t(q-1)$ and $2t(q-1)$, as in Theorem 2.1. Note that $2t(q-1) = 1312740 < 2^{20.4}$ for Streamlined NTRU Prime 4591^{761}.

The same bounds apply to the multiplication by r', since r' is constructed to have coefficients in $\{-1, 0, 1\}$ and is (eventually) checked to have Hamming weight $2t$. Similar comments apply to $3f$, except for a factor 3 in the bounds. We actually multiply by f, so identical bounds apply, and then multiply each output coefficient by 3.

The fourth multiplication is in $\mathcal{R}/3 = (\mathbb{Z}/3)[x]/(x^p - x - 1)$: decapsulation multiplies e by a precomputed $1/g$. For simplicity we currently reuse the same $\mathcal{R}/q$ code for this multiplication in $\mathcal{R}/3$. The output coefficients here are bounded

by $2p$ in absolute value; $2p$ is below $q/2$ for Streamlined NTRU Prime 4591^{761}. We could save time by performing arithmetic on more tightly packed $\mathcal{R}/3$ elements.

6.2. Choosing Haswell multiplication instructions. The Haswell instruction set includes "AVX" and "AVX2" instructions operating on 256-bit vectors. We now compare various multiplication instructions to the requirements of the polynomial multiplications in Streamlined NTRU Prime 4591^{761}. For this subsection we assume schoolbook multiplication of polynomials; later we consider the impact of polynomial-multiplication techniques that use fewer arithmetic operations.

The `vpmullw` instruction performs 16 separate multiplications of integers modulo 2^{16}. A new `vpmullw` instruction can start every cycle. Using `vpmullw` to perform p^2 separate multiplications modulo 2^{16} thus takes $p^2/16 \approx 36195$ cycles.

Polynomial multiplication involves a similar number of additions, which one might think take extra time. However, the same Haswell core can start a new `vpaddw` instruction, which performs 16 separate additions mod 2^{16}, twice every cycle, *in parallel* with the `vpmullw` instructions. The multiplication instructions occupy "port 0" on the core, while the addition instructions are handled by "port 1" and "port 5"; the "ports" in a core operate in parallel.

A more serious problem is that 2^{16} is not large enough for the output coefficients in $\mathbb{Z}[x]$, which as noted above can range from $-t(q-1) = -656370$ to $t(q-1) = 656370$. One can safely add as many as 14 integers between $-(q-1)/2$ and $(q-1)/2$ while staying within an interval of length $14(q-1) < 2^{16}$, but to safely add more integers one must first "squeeze" the sums. This means reducing the sums modulo q into a smaller range, although not necessarily "freezing" them into the minimum range, -2295 through 2295.

The best squeezing method we found uses `vpmulhrsw`, which performs 16 separate copies of the following operation: multiply two integers between -2^{15} and 2^{15}, divide by 2^{15}, and round to an integer. We take the second integer as 7; then the output is $\text{round}(7x/2^{15})$ where x is the first integer. This is not always exactly $\text{round}(x/4591)$ but it is close. We then multiply by 4591 and subtract from x, obtaining something that cannot be much larger than 2295 in absolute value. The exact bound depends on exactly how big x is allowed to be; for example, if x is between -32000 and 32000, then the output is between -2881 and 2881. (At the end of the computation we use several more instructions to freeze x.)

An alternative is to switch to `vpmulld`, which performs 8 separate multiplications of integers modulo 2^{32}, and `vpaddd`, which performs 8 separate additions of integers modulo 2^{32}. This has the advantage of not requiring any reductions until the end of the computation, but it has two much larger disadvantages: first, each instruction handles only 8 operations instead of 16; second, `vpmulld` occupies port 0 for 2 cycles instead of 1.

A better alternative is to switch to `vfmadd231ps`, which performs 8 separate operations of the form $ab+c$ on single-precision floating-point inputs a, b, c. Port 0 and port 1 can each handle a new `vfmadd231ps` instruction every cycle, for a total of 16 $ab + c$ operations every cycle. The advantage over `vpmullw` is that a

single-precision floating-point number can exactly represent any integer between -2^{24} and 2^{24}. Again no reductions are required until the end of the computation.

There are some slowdowns not discussed above, but quite concise schoolbook-polynomial-multiplication code using `vfmadd231ps` performs a multiplication in $\mathcal{R}/q$ in just 50000 cycles. The number of coefficient multiplications here is an order of magnitude larger than the number of coefficient multiplications inside NTT-based multiplication in $(\mathbb{Z}/12289)[x]/(x^{1024}+1)$, but this cycle count is only 1.6× more than the New Hope software [5], which relies on double-precision floating-point arithmetic. This illustrates the importance of keeping intermediate results small, so that one can efficiently use small multipliers without spending much time on reductions.

6.3. Karatsuba's method. Karatsuba's method uses a linear amount of extra work to reduce a $2n$-coefficient multiplication to three n-coefficient multiplications We use specifically the "refined Karatsuba identity" from [18, Section 2]:

$$(F_0 + x^n F_1)(G_0 + x^n G_1) = (1 - x^n)(F_0 G_0 - x^n F_1 G_1) + x^n (F_0 + F_1)(G_0 + G_1).$$

The initial computations of $F_0 + F_1$ and $G_0 + G_1$ each take n additions. The final computations take $5n - 3$ additions. For simplicity we actually use $5n$ additions, zero-padding each intermediate product from $2n-1$ coefficients to $2n$ coefficients.

For schoolbook multiplication our main concern was the Haswell multiplication instructions: 16 single-precision floating-point multiplications per cycle sounded better than 16 16-bit integer multiplications per cycle, since floating-point operations have more precision. Karatsuba's method adds emphasis to the addition instructions, and here the integer story might sound clearly better:

- The Haswell can start two `vpaddw` instructions per cycle: as noted above, one on port 1 and one on port 5. This is a total of 32 separate additions modulo 2^{16} per cycle.
- The Haswell floating-point addition instruction `vaddps` is limited to port 1, for a total of 8 single-precision floating-point additions per cycle. One can do better by using `vfmadd231ps` for additions (artificially multiplying by 1), for a total of 16 single-precision floating-point additions per cycle, but this is still just half as many additions per cycle as the integer case.
- Furthermore, floating-point numbers occupy more space than 16-bit integers, and floating-point additions have higher latency. These are not problems for schoolbook multiplication, which (at the size we use) easily fits into level-1 cache and is highly parallel, but Karatsuba's method uses more space and is less parallel.

On the other hand, floating-point numbers still have the advantage of more precision. Two Karatsuba layers applied to integers between -2295 and 2295 produce results between -9180 and 9180, still fitting into 16-bit integers; meanwhile the same layers applied to integers in $\{-1, 0, 1\}$ produce results between -4 and 4; but then the products can overflow 16-bit integers. There is a `vpmulhd` instruction that produces the high 16 bits of each product, but reduction then costs many more instructions.

Our current software starts with 768-coefficient polynomials (zero-padded from the 761-coefficient inputs) stored as vectors of 16-bit integers. We use multiple layers of Karatsuba's method: specifically, 5 layers, down to 24×24 schoolbook multiplications. To avoid reductions, we use floating-point arithmetic for the schoolbook multiplications, and we squeeze inputs partway through the Karatsuba layers: specifically, we squeeze 96-coefficient polynomials. We also convert from integers to floating-point numbers partway through the Karatsuba layers, trying to minimize the total cost of conversions and Karatsuba additions. We use floating-point operations to squeeze 192-coefficient products, convert those products back to integers, and then squeeze intermediate results in the final Karatsuba additions so as to avoid overflowing 16-bit integers.

6.4. Other multiplication methods. Karatsuba's method is asymptotically superseded by Toom's method and various FFT-based methods. For large input sizes, it is clear that FFT-based methods are the best. However, for small to medium input sizes, it is unclear which methods or combinations of methods are best.

We have analyzed many different combinations of schoolbook multiplication, refined Karatsuba, the arbitrary-degree variant of Karatsuba for degrees 3, 4, 5, or 6, and Toom's method for splitting into 3, 4, 5, or 6 pieces. Many methods involve multiplications by large constants, spoiling the smallness of our second polynomial, but this is not a problem in double-precision floating-point arithmetic. Our best double-precision result so far is 46784 cycles, achieved as follows: use Toom's method with evaluation points $0, 1, -1, 2, -2, 3, -3, 4, -4, 5, \infty$ to reduce a 768×768 product to 11 separate 128×128 products; then use 5 layers of refined Karatsuba.

We also experimented with variants of the Schönhage–Strassen multiplication method, starting from the framework of [16, Section 9]. The Schönhage–Strassen multiplication method is like Karatsuba's method in that it does not involve multiplications by large constants, but as $n \to \infty$ it uses only $n^{1+o(1)}$ arithmetic operations. The conventional wisdom is that the Schönhage–Strassen method is of purely asymptotic interest, but we found a tuned variant to be surprisingly competitive, around 32000 cycles, again mixing 16-bit integer arithmetic with floating-point arithmetic.

References

1. ECRYPT yearly report on algorithms and keysizes, 2005. https://www.cosic.esat.kuleuven.be/ecrypt/ecrypt1/documents/D.SPA.16-1.0.pdf.
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, Shi Bai, and Léo Ducas. A subfield lattice attack on overstretched NTRU assumptions - cryptanalysis of some FHE and graded encoding schemes. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part I*, volume 9814 of *Lecture Notes in Computer Science*, pages 153–178. Springer, 2016. https://eprint.iacr.org/2016/127.
4. 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.
5. 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.
6. Jacob Alperin-Sheriff and Daniel Apon. Dimension-preserving reductions from LWE to LWR. *IACR Cryptology ePrint Archive*, 2016:589, 2016. http://eprint.iacr.org/2016/589.
7. 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.
8. 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.
9. Shi Bai, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, and Damien Stehlé. Crystals: cryptographic suite for algebraic lattices, 2017. https://www.cs.bris.ac.uk/Research/CryptographySecurity/RWC/2017/tancrede.lepoint.pdf.
10. 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.
11. William D. Banks and Igor E. Shparlinski. A variant of NTRU with non-invertible polynomials. In Alfred Menezes and Palash Sarkar, editors, *Progress in Cryptology - INDOCRYPT 2002, Third International Conference on Cryptology in India, Hyderabad, India, December 16-18, 2002*, volume 2551 of *Lecture Notes in Computer Science*, pages 62–70. Springer, 2002. https://faculty.missouri.edu/~bankswd/papers/2003_generalized_ntru.pdf.
12. Paulo S. L. M. Barreto, Shay Gueron, Tim Gueneysu, Rafael Misoczki, Edoardo Persichetti, Nicolas Sendrier, and Jean-Pierre Tillich. CAKE: Code-based algorithm for key encapsulation. 2017. https://eprint.iacr.org/2017/757.
13. Kenneth E. Batcher. Sorting networks and their applications. In *American Federation of Information Processing Societies: AFIPS Conference Proceedings: 1968 Spring Joint Computer Conference, Atlantic City, NJ, USA, 30 April - 2 May 1968*, volume 32 of *AFIPS Conference Proceedings*, pages 307–314. Thomson Book

Company, Washington D.C., 1968. http://www.cs.kent.edu/~batcher/sort.pdf.

14. Jens Bauch, Daniel J. Bernstein, Henry de Valence, Tanja Lange, and Christine van Vredendaal. Short generators without quantum computers: The case of multiquadratics. 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 27–59, 2017. https://multiquad.cr.yp.to.
15. Anja Becker, Léo Ducas, Nicolas Gama, and Thijs Laarhoven. New directions in nearest neighbor searching with applications to lattice sieving. In Krauthgamer [76], pages 10–24. https://eprint.iacr.org/2015/1128.
16. Daniel J. Bernstein. Multidigit multiplication for mathematicians. 2001. https://cr.yp.to/papers.html#m3.
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. Batch binary Edwards. In Shai Halevi, editor, *Advances in Cryptology - CRYPTO 2009, 29th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2009. Proceedings*, volume 5677 of *Lecture Notes in Computer Science*, pages 317–336. Springer, 2009. https://cr.yp.to/papers.html#bbe.
19. Daniel J. Bernstein. Complexity news: discrete logarithms in multiplicative groups of small-characteristic finite fields—the algorithm of Barbulescu, Gaudry, Joux, Thomé, 2013. https://cr.yp.to/talks/2013.07.18/slides-djb-20130718-a4.pdf.
20. Daniel J. Bernstein. A subfield-logarithm attack against ideal lattices, 2014. https://blog.cr.yp.to/20140213-ideal.html.
21. Daniel J. Bernstein, Tung Chou, and Peter Schwabe. McBits: Fast constant-time code-based cryptography. In Guido Bertoni and Jean-Sébastien Coron, editors, *Cryptographic Hardware and Embedded Systems - CHES 2013 - 15th International Workshop, Santa Barbara, CA, USA, August 20-23, 2013. Proceedings*, volume 8086 of *Lecture Notes in Computer Science*, pages 250–272. Springer, 2013. https://cr.yp.to/papers.html#mcbits.
22. Daniel J. Bernstein and Tanja Lange. eBACS: ECRYPT benchmarking of cryptographic systems. https://bench.cr.yp.to (accessed 9 February 2017).
23. 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.
24. Thomas A. Berson. Failure of the McEliece public-key cryptosystem under message-resend and related-message attack. In Burton S. Kaliski Jr., editor, *Advances in Cryptology - CRYPTO '97, 17th Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 1997, Proceedings*,

volume 1294 of *Lecture Notes in Computer Science*, pages 213–220. Springer, 1997. http://www.anagram.com/berson/mceliece.pdf.

25. Jean-François Biasse and Fang Song. On the quantum attacks against schemes relying on the hardness of finding a short generator of an ideal in $\mathbb{Q}(\zeta_{p^n})$, 2015. http://cacr.uwaterloo.ca/techreports/2015/cacr2015-12.pdf.
26. 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 [76], pages 893–902.
27. 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.
28. Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, and Damien Stehlé. CRYSTALS – Kyber: a CCA-secure module-lattice-based KEM, 2017. https://eprint.iacr.org/2017/634.
29. Richard P. Brent and H. T. Kung. The area-time complexity of binary multiplication. *J. ACM*, 28(3):521–534, 1981. http://maths-people.anu.edu.au/~brent/pd/rpb055.pdf.
30. 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.
31. Daniel Cabarcas, Patrick Weiden, and Johannes A. Buchmann. On the efficiency of provably secure NTRU. In Michele Mosca, editor, *Post-Quantum Cryptography - 6th International Workshop, PQCrypto 2014, Waterloo, ON, Canada, October 1-3, 2014. Proceedings*, volume 8772 of *Lecture Notes in Computer Science*, pages 22–39. Springer, 2014.
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. Wouter Castryck, Ilia Iliashenko, and Frederik Vercauteren. On the tightness of the error bound in Ring-LWE. Cryptology ePrint Archive, Report 2016/240, 2016. https://eprint.iacr.org/2016/240.
35. Wouter Castryck, Ilia Iliashenko, and Frederik Vercauteren. Provably weak instances of Ring-LWE revisited. 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 I*, volume 9665 of *Lecture Notes in Computer Science*, pages 147–167. Springer, 2016. https://eprint.iacr.org/2016/239.

36. Sanjit Chatterjee, Neal Koblitz, Alfred Menezes, and Palash Sarkar. Another look at tightness II: Practical issues in cryptography, 2016. https://eprint.iacr.org/2016/360.
37. 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.
38. 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.
39. Yuanmi Chen and Phong Q. Nguyen. BKZ 2.0: Better lattice security estimates (full version), 2011. http://www.di.ens.fr/~ychen/research/Full_BKZ.pdf.
40. Tung Chou. Sandy2x: New Curve25519 speed records. In Orr Dunkelman and Liam Keliher, editors, *Selected Areas in Cryptography - SAC 2015 - 22nd International Conference, Sackville, NB, Canada, August 12-14, 2015, Revised Selected Papers*, volume 9566 of *Lecture Notes in Computer Science*, pages 145–160. Springer, 2015. https://eprint.iacr.org/2015/943.
41. Henri Cohen. *A course in computational algebraic number theory*, volume 138 of *Graduate Texts in Mathematics*. Springer-Verlag, Berlin, 1993.
42. Henri Cohen. *Advanced topics in computational number theory*, volume 193 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, 2000.
43. 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.
44. 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. http://eprint.iacr.org/2016/885.
45. Rafaël del Pino, Vadim Lyubashevsky, and David Pointcheval. The whole is less than the sum of its parts: Constructing more efficient lattice-based AKEs. In Zikas and Prisco [119], pages 273–291. https://eprint.iacr.org/2016/435.
46. 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.
47. Jintai Ding. Solving LWE problem with bounded errors in polynomial time. *IACR Cryptology ePrint Archive*, 2010:558, 2010. https://eprint.iacr.org/2010/558.
48. Hans Dobbertin, Antoon Bosselaers, and Bart Preneel. RIPEMD-160: A strengthened version of RIPEMD. In Dieter Gollmann, editor, *Fast Software Encryption, Third International Workshop, Cambridge, UK, February 21-23, 1996, Proceedings*, volume 1039 of *Lecture Notes in Computer Science*, pages 71–82. Springer,

1996. http://homes.esat.kuleuven.be/~bosselae/ripemd160/pdf/AB-9601/AB-9601.pdf.
49. Jean Louis Dornstetter. On the equivalence between Berlekamp's and Euclid's algorithms. *IEEE Trans. Information Theory*, 33(3):428–431, 1987.
50. 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.
51. 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.
52. Yara Elias, Kristin E. Lauter, Ekin Ozman, and Katherine E. Stange. Provably weak instances of Ring-LWE. In Gennaro and Robshaw [57], pages 63–92. https://eprint.iacr.org/2015/106.
53. Armando Faz-Hernández and Julio López. Fast implementation of Curve25519 using AVX2. In Kristin E. Lauter and Francisco Rodríguez-Henríquez, editors, *Progress in Cryptology - LATINCRYPT 2015 - 4th International Conference on Cryptology and Information Security in Latin America, Guadalajara, Mexico, August 23-26, 2015, Proceedings*, volume 9230 of *Lecture Notes in Computer Science*, pages 329–345. Springer, 2015.
54. 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.
55. Philippe Gaborit, editor. *Post-Quantum Cryptography - 5th International Workshop, PQCrypto 2013, Limoges, France, June 4-7, 2013. Proceedings*, volume 7932 of *Lecture Notes in Computer Science*. Springer, 2013.
56. Steven D. Galbraith and Pierrick Gaudry. Recent progress on the elliptic curve discrete logarithm problem. Cryptology ePrint Archive, Report 2015/1022, 2015. https://eprint.iacr.org/2015/1022.
57. 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*. Springer, 2015.
58. Shay Gueron and Vlad Krasnov. Fast quicksort implementation using AVX instructions. *Comput. J.*, 59(1):83–90, 2016.
59. Shay Gueron and Fabian Schlieker. Speeding up R-LWE post-quantum key exchange. In *NordSec 2016*, volume 10014 of *LNCS*, pages 187–198, 2016. https://eprint.iacr.org/2016/467.
60. Tim Güneysu, Tobias Oder, Thomas Pöppelmann, and Peter Schwabe. Software speed records for lattice-based signatures. In Gaborit [55], pages 67–82.
61. 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.
62. 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.

63. 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.
64. 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.
65. Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. NTRU: a new high speed public key cryptosystem, 2016. Circulated at Crypto 1996; put online in 2016 at https://web.securityinnovation.com/hubfs/files/ntru-orig.pdf.
66. Jeffrey Hoffstein and Joseph H. Silverman. Random small Hamming weight products with applications to cryptography. *Discrete Applied Mathematics*, 130(1):37–49, 2003. https://assets.securityinnovation.com/static/downloads/NTRU/resources/TECH_ARTICLE_RAND.pdf.
67. 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.
68. 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.
69. 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.
70. 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.
71. Andreas Hülsing, Joost Rijneveld, John M. Schanck, and Peter Schwabe. High-speed key encapsulation from NTRU. 2017. https://eprint.iacr.org/2017/667; CHES 2017, to appear.
72. Antoine Joux. A new index calculus algorithm with complexity $L(1/4 + o(1))$ in small characteristic. 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 355–379. Springer, 2013.
73. 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.
74. Paul Kirchner and Pierre-Alain Fouque. Comparison between subfield and straightforward attacks on NTRU. Cryptology ePrint Archive, Report 2016/717, 2016. https://eprint.iacr.org/2016/717.
75. Donald E. Knuth. *The Art of Computer Programming, Volume III: Sorting and Searching*. Addison-Wesley, 1973.

76. 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.
77. Virendra Kumar. `ntruees743ep1` software, 2014. Included in [22].
78. 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.
79. Thijs Laarhoven. Sieving for shortest vectors in lattices using angular locality-sensitive hashing. In Gennaro and Robshaw [57], pages 3–22. https://eprint.iacr.org/2014/744.pdf.
80. 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.
81. Adam Langley. How to botch TLS forward secrecy, 2013. https://www.imperialviolet.org/2013/06/27/botchingpfs.html.
82. Adeline Langlois and Damien Stehlé. Worst-case to average-case reductions for module lattices. *Des. Codes Cryptography*, 75(3):565–599, 2015. https://eprint.iacr.org/2012/090.
83. Patrick Longa and Michael Naehrig. Speeding up the number theoretic transform for faster ideal lattice-based cryptography, 2016. Cryptology ePrint Archive, Report 2016/504, https://eprint.iacr.org/2016/504, https://www.microsoft.com/en-us/research/project/lattice-cryptography-library/.
84. Rüdiger Loos. Generalized polynomial remainder sequences. In Bruno Buchberger, George Edwin Collins, and Rüdiger Loos, editors, *Computer Algebra: Symbolic and Algebraic Computation*, pages 115–137, Vienna, 1982. Springer Vienna.
85. Vadim Lyubashevsky. Digital signatures based on the hardness of ideal lattice problems in all rings. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part II*, volume 10032 of *Lecture Notes in Computer Science*, pages 196–214, 2016. https://eprint.iacr.org/2016/796.
86. 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.
87. 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.
88. Artur Mariano, Christian H. Bischof, and Thijs Laarhoven. Parallel (probable) lock-free hash sieve: A practical sieving algorithm for the SVP. In *44th International Conference on Parallel Processing, ICPP 2015, Beijing, China, September 1-4, 2015*, pages 590–599. IEEE Computer Society, 2015. https://eprint.iacr.org/2015/041.
89. James Massey. Shift-register synthesis and BCH decoding. *IEEE Transactions on Information Theory*, 15(1):122–127, January 1969.

90. 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.
91. Daniele Micciancio and Michael Walter. Fast lattice point enumeration with minimal overhead. In Piotr Indyk, editor, *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 276–294. SIAM, 2015. https://eprint.iacr.org/2014/569.
92. Paul C. van Oorschot and Michael J. Wiener. Parallel collision search with cryptanalytic applications. *J. Cryptology*, 12(1):1–28, 1999.
93. Hiroyuki Osada. The Galois groups of the polynomials $X^n + aX^l + b$. *J. Number Theory*, 25(2):230–238, 1987.
94. 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.
95. Chris Peikert. How (not) to instantiate Ring-LWE. In Zikas and Prisco [119], pages 411–430. https://eprint.iacr.org/2016/351.
96. 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.
97. Chris Peikert, Oded Regev, and Noah Stephens-Davidowitz. Pseudorandomness of Ring-LWE for any ring and modulus. In Hamed Hatami, Pierre McKenzie, and Valerie King, editors, *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 461–473. ACM, 2017. https://eprint.iacr.org/2017/258.
98. Edoardo Persichetti. Secure and anonymous hybrid encryption from coding theory. In Gaborit [55], pages 174–187.
99. 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.
100. 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.
101. 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.
102. Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, May 22-24, 2005*, pages 84–93. ACM, 2005.
103. Miruna Rosca, Amin Sakzad, Ron Steinfeld, and Damien Stehle. Middle-product learning with errors. 2017. https://eprint.iacr.org/2017/628; Crypto 2017, to appear.

104. The Sage Developers. *SageMath, the Sage Mathematics Software System (Version 6.5)*, 2015. http://www.sagemath.org.
105. Halvor Sakshaug. Security analysis of the NTRUEncrypt public key encryption scheme, 2007. brage.bibsys.no/xmlui/bitstream/handle/11250/258846/426901_FULLTEXT01.pdf.
106. 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.
107. 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.
108. Ernst S. Selmer. On the irreducibility of certain trinomials. *Math. Scand.*, 4:287–302, 1956.
109. Victor Shoup. A proposal for an ISO standard for public key encryption. *IACR Cryptology ePrint Archive*, 2001. https://eprint.iacr.org/2001/112.
110. Victor Shoup. OAEP reconsidered. *J. Cryptology*, 15(4):223–249, 2002. https://eprint.iacr.org/2000/060.
111. Joseph H. Silverman. Almost inverses and fast NTRU key creation, 1999. https://assets.securityinnovation.com/static/downloads/NTRU/resources/NTRUTech014.pdf.
112. Nigel P. Smart and Frederik Vercauteren. Fully homomorphic encryption with relatively small key and ciphertext sizes. In Phong Q. Nguyen and David Pointcheval, editors, *Public Key Cryptography - PKC 2010, 13th International Conference on Practice and Theory in Public Key Cryptography, Paris, France, May 26-28, 2010. Proceedings*, volume 6056 of *Lecture Notes in Computer Science*, pages 420–443. Springer, 2010. https://eprint.iacr.org/2009/571.
113. 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.
114. 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.
115. Josef Stein. Computational problems associated with Racah algebra. *Journal of Computational Physics*, 1(3):397–405, 1967.
116. 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.
117. Xiaoyun Wang and Hongbo Yu. How to break MD5 and other hash functions. In *EUROCRYPT*, volume 3494 of *LNCS*, pages 19–35, 2005.
118. Thomas Wunderer. Revisiting the hybrid attack: Improved analysis and refined security estimates. *IACR Cryptology ePrint Archive*, 2016:733, 2016.
119. Vassilis Zikas and Roberto De Prisco, editors. *Security and Cryptography for Networks - 10th International Conference, SCN 2016, Amalfi, Italy, August 31 - September 2, 2016, Proceedings*, volume 9841 of *Lecture Notes in Computer Science*. Springer, 2016.

A Avoiding rings with worrisome structure

Practically all proposals for ideal-lattice-based cryptography use cyclotomic rings such as $\mathbb{Z}[x]/(x^{1024}+1)$. These rings have many automorphisms, such as $x \mapsto x^3$. Typical encryption proposals work specifically in cyclotomic quotient rings such as $(\mathbb{Z}/12289)[x]/(x^{1024}+1)$, allowing further nontrivial ring homomorphisms such as $x \mapsto 7$.

In February 2014, as noted in Section 1, we publicly recommended[9] changing the choice of rings in ideal-lattice-based cryptography. Part of the recommendation is to switch from cyclotomic rings to new rings very far from having any non-identity automorphisms: specifically, rings of the form $\mathbb{Z}[x]/(x^p-x-1)$. Another part of the recommendation is to use quotient *fields* such as $(\mathbb{Z}/4591)[x]/(x^{761}-x-1)$, eliminating further nonzero homomorphisms. The complete recommendation has a precise mathematical definition explained below.

Attacks published *after* this recommendation have already built an excellent track record for the recommendation. For example, it is now generally agreed that the Smart–Vercauteren system [112] *using the old rings* is broken by a polynomial-time quantum attack and by a subexponential-time pre-quantum attack. Nobody has extended these attacks to our new rings. The attacks rely upon various interesting operations that are available in the old rings and not in the new rings; see below.

A.1. Cryptographic risk management. An important part of the cryptographer's job is to extrapolate beyond known attacks (just like other scientists formulating theories beyond current knowledge), with the goal of making the safest decisions about an unclear future. For example:

- Dobbertin, Bosselaers, and Preneel wrote in 1996 [48] that "it is anticipated that these techniques can be used to produce collisions for MD5" and recommended switching to other hash functions. This was long before the MD5 attack by Wang and Yu [117].
- Many discrete-logarithm experts recommended prime fields (see, e.g., [1, page 25]) long before recent attacks such as [72] against small-characteristic finite-field discrete logarithms.
- Many discrete-logarithm experts specifically recommend prime-field ECDL over small-characteristic ECDL (see, e.g., [1, page 62]), even though none of the efforts to break small-characteristic ECDL have been successful (see the recent survey [56]).
- Experts frequently recommend one unbroken system over another unbroken system of similar speed, saying that possible attack strategies against the first system are better understood.

[9] See the blog post [20] by the first author; see also the 2013 note [19, page 2]. It is proper to credit this "NTRU Prime" paper as the paper introducing this recommendation; [19] and [20] were preliminary announcements. We mention these announcements because the timeline illustrates the points made in Section A.1.

– Our general recommendation included, as a special case, switching the Smart–Vercauteren system from the old rings to the new rings. This came before the polynomial-time quantum attacks against the system using the old rings.

History shows that this approach, despite its uncertainties, produces much better security than merely asking what has already been broken.

Our recommendation to change the choice of rings was, and is, a broad recommendation for ideal-lattice-based cryptography: it applies to Smart–Vercauteren, to NTRU, to Ring-LWE-based cryptosystems, etc. We suggest the name "NTRU Prime" for cryptosystems obtained in this way from NTRU or from Ring-LWE. Here "NTRU" refers to a central idea used in all of these cryptosystems (as noted in Section 3.3), namely sending $m + hr$ where m, r are small secrets and h is public; and "Prime" reflects the fact that our modifications eliminate several different types of factorizations.

We emphasize that normal NTRU parameters are not affected by any of the known attacks discussed in this appendix. The same holds for cyclotomic Ring-LWE. However, we are skeptical of the notion that the most recent papers are the end of the attack story.

There is widespread agreement on the general goal of removing unnecessary algebraic structure from cryptography. As an example, the common recommendation of prime fields for DL takes various operations (in particular, automorphisms) away from attackers, and all available evidence is that this rescues *some* fraction of DL systems without enabling any new attacks. This provides ample justification for the recommendation, not merely in the cases where there are already known attacks (such as FFDL) but also in other cases (such as ECDL). Similarly, our recommendation of the new rings takes various operations (again, in particular, automorphisms) away from attackers, and all available evidence is that this rescues *some* fraction of lattice-based systems without enabling any new attacks. This provides ample justification for the recommendation, not merely in the cases where there are already known attacks (such as Smart–Vercauteren) but also in other cases (such as NTRU).

Of course, one could speculate that our recommendation does not help security. A recent paper by Albrecht, Bai, and Ducas [3] broke some "overstretched NTRU assumptions" using the old rings, but Kirchner and Fouque [74] extended this attack to all rings, and one could speculate that *all* attacks against the old rings can be somehow adapted to the new rings. One could even speculate that our recommendation somehow *hurts* security.

However, the state of the art in cryptanalysis points in the opposite direction to these speculations. Some systems are unbroken with rings that comply with the recommendation and broken with rings that do not comply with the recommendation; there is nothing the other way around. We successfully and publicly *predicted* this situation in advance as a consequence of worrisome structure, specifically unnecessary ring homomorphisms.

Ideas for improving security usually hurt performance; see, e.g., Appendix C. If following our recommendation makes lattice-based systems much more expensive, then one has to ask whether using the increased costs in other ways, such as

larger lattice dimensions, would produce a larger security benefit. But our performance results show that Streamlined NTRU Prime is competitive with, and in some ways even better than, previous lattice-based public-key cryptosystems.

We again emphasize that we are *not* saying that standard NTRU parameters are known to be broken in a way rescued by NTRU Prime. We are saying that taking unnecessary ring homomorphisms away from the attacker is reducing the attack surface. NTRU Prime provides a less structured alternative to NTRU Classic and NTRU NTT, the same way that prime-field FFDL/ECDL have always provided less structured alternatives to small-characteristic FFDL/ECDL.

A.2. Case study: the Campbell–Groves–Shepherd attack. The October 2014 Campbell–Groves–Shepherd preprint "Soliloquy: a cautionary tale" [32] sent shock waves through lattice-based cryptography. The preprint describes a lattice-based cryptosystem named "Soliloquy" that the authors say they privately developed in 2007, and then claims a polynomial-time quantum key-recovery attack against this system. As mentioned briefly in [32], the key-recovery problem for this system is identical to the key-recovery problem for the Smart–Vercauteren system. The same problem has also appeared in various other homomorphic encryption systems and multilinear-map systems.

In these systems, everyone shares a standard monic irreducible polynomial $P \in \mathbb{Z}[x]$ with small coefficients. Smart and Vercauteren [112, Section 7] take power-of-2 cyclotomic polynomials, such as the polynomial $x^{1024}+1$, but [112, Section 3] allows more general polynomials, and [32] allows any cyclotomic polynomial. The receiver's public key consists of an integer α and a prime number q dividing $P(\alpha)$. Note that $q\mathcal{R}+(x-\alpha)\mathcal{R}$ is a prime ideal of the ring $\mathcal{R} = \mathbb{Z}[x]/P$; the receiver's secret key is a small generator $g \in \mathcal{R}$ of this ideal. The encryption and decryption procedures are not difficult but are not relevant here.

The first stage of the attack finds *some* generator of the ideal, expressed as a product of powers of small ring elements. Biasse and Song questioned the claimed performance of the algorithm for this stage (and these claims were not defended by the authors of [32]) but subsequently presented a different polynomial-time quantum algorithm for this stage; see [25] and [26]. Even without quantum computers, well-known techniques complete this stage in subexponential time.

The second stage of the attack reduces the generator to a small generator (either g or something else that is just as good for decryption, such as $-g$). This is a decoding problem in what is called the "log-unit lattice". One normally expects decoding problems to take exponential time, but *for cyclotomic polynomials* P (under certain technical assumptions) one can efficiently write down a very short basis for the log-unit lattice. This basis consists of logarithms of various "cyclotomic units", as explained very briefly in [32] and analyzed in detail in the followup paper [43] by Cramer, Ducas, Peikert, and Regev. For example, for $P = x^{1024}+1$, the ring $\mathcal{R}$ contains $(1-x^3)/(1-x) = 1+x+x^2$, and also contains the reciprocal $(1-x)/(1-x^3) = (1-x^{2049})/(1-x^3) = 1+x^3+\cdots+x^{2046}$; so $(1-x^3)/(1-x)$ is a unit in R, a typical example of a cyclotomic unit.

This attack was originally stated as an attack specifically against principal ideals with short generators. However, Cramer, Ducas, and Wesolowski [44] re-

cently used this attack as a subroutine in a polynomial-time quantum attack against a broader problem. For cyclotomic fields (again under certain technical assumptions), given *any nonzero ideal*, the Cramer–Ducas–Wesolowski attack finds a vector whose length is within a factor $2^{\tilde{O}(\sqrt{\deg P})}$ of the length of the shortest nonzero vector. This is a tremendous improvement compared to the $2^{\tilde{O}(\deg P)}$ from previous attacks against the same problem. This attack makes further use of cyclotomic structure to find a nearby principal multiple of the input ideal, and then applies the original attack to find the shortest generator of this principal ideal.

A.3. Mathematical specification of our recommendation. We recommend taking a standard monic irreducible polynomial P whose degree is a prime p, and whose "Galois group" is as large as possible, isomorphic to the permutation group S_p of size $p!$. Most polynomials of degree p have Galois group S_p, and we specifically suggest the small polynomial $P = x^p - x - 1$, which is irreducible and has Galois group S_p; see [108] and [93]. Furthermore, in contexts using moduli (such as NTRU and Ring-LWE), we recommend taking a prime modulus q that is "inert" in $\mathcal{R}$, i.e., where P is irreducible in $(\mathbb{Z}/q)[x]$, i.e., where $(\mathbb{Z}/q)[x]/P$ is a field. This happens with probability approximately $1/p$ for a "random" prime q; see Appendix P for many examples of acceptable pairs (p, q).

One way to define the Galois group is as the group of automorphisms of the smallest field that contains all the complex roots of P. Consider, for example, the field $\mathbb{Q}(\zeta)$ where $\zeta = \exp(2\pi i/2048)$. The notation $\mathbb{Q}(\zeta)$ means the smallest field containing both $\mathbb{Q}$ and ζ; explicitly, $\mathbb{Q}(\zeta)$ is the set of complex numbers $q_0 + q_1\zeta + \cdots + q_{1023}\zeta^{1023}$ with $q_0, q_1, \ldots, q_{1023} \in \mathbb{Q}$. The complex roots of $P = x^{1024} + 1$ are $\zeta, \zeta^3, \zeta^5, \ldots, \zeta^{2047}$, all of which are in $\mathbb{Q}(\zeta)$, so $\mathbb{Q}(\zeta)$ is the smallest field that contains all the complex roots of P. There are exactly 1024 automorphisms of this field (invertible maps from the field to itself preserving $0, 1, +, -, \cdot$). These automorphisms are naturally labeled $1, 3, 5, \ldots, 2047$; the automorphism with label i maps ζ to ζ^i, so it maps ζ^j to ζ^{ij}. In other words, automorphism i permutes the complex roots of P the same way that ith powering does; the Galois group is thus isomorphic to the multiplicative group $(\mathbb{Z}/2048)^*$.

NTRU traditionally takes $P = x^p - 1$ with p prime and q a power of 2, typically 2048. These choices violate our recommendation in several ways. First of all, $x^p - 1$ is not irreducible. One can tweak NTRU to work modulo the cyclotomic polynomial $\Phi_p = (x^p - 1)/(x - 1)$, but this polynomial does not have prime degree. Furthermore, the Galois group of Φ_p has size only $p - 1$, vastly smaller than $(p - 1)!$. Also, the modulus q is not prime.

Ring-LWE-based systems typically take $P = x^p + 1$ where p is a power of 2 and q is a prime in $1 + 2p\mathbb{Z}$. These choices also violate our recommendation in several ways. The polynomial P is irreducible, but it does not have prime degree. Furthermore, its Galois group has size only p, vastly smaller than $p!$. The modulus q is prime, but P is very far from irreducible modulo q: in fact, it splits into linear factors modulo q.

A.4. How the recommendation reduces attack surface. A recent pre-quantum quasipolynomial-time attack by Bauch, Bernstein, de Valence, Lange,

and van Vredendaal [14] finds short generators of ideals in a large class of multiquadratic rings. This attack exploits the fact that a multiquadratic field has many subfields. The degree of the field is a multiple of the degree of every subfield, so by taking a prime degree we obviously rule out any such attack: the only subfields of $\mathbb{Q}[x]/P$ are $\mathbb{Q}$ and the entire field $\mathbb{Q}[x]/P$.

To understand why we also require very large Galois groups, consider the suggestion from [3] to use the field $\mathbb{Q}(\zeta + \zeta^{-1})$ with $\zeta = \exp(2\pi i/2p)$, where both p and $(p-1)/2$ are prime. This field has prime degree $(p-1)/2$ and thus stops subfield attacks. It does not, however, stop the attack of [32]: one can easily write down a very short basis consisting of logs of cyclotomic units in this field, such as $(\zeta^3 - \zeta^{-3})/(\zeta - \zeta^{-1})$.

More generally, if a number field of prime degree p has a Galois group of size p then the field is a subfield of a cyclotomic field. Even more generally, the Kronecker–Weber theorem states that any "abelian" number field is a subfield of a cyclotomic field. This might not enable an attack along the lines of [32] if the cyclotomic field has degree much larger than p, but we do not think that it is wise to rely on this.

Of course, prohibiting minimum size p is not the same as requiring maximum size $p!$; there is a large gap between p and $p!$. But having a Galois group of size, say, $2p$ means that one can write down a degree-$2p$ extension field with $2p$ automorphisms, and one can then try to apply these automorphisms to build many units, generalizing the construction of cyclotomic units. From the perspective of algebraic number theory, the fact that, e.g., $(\zeta^3 - \zeta^{-3})/(\zeta - \zeta^{-1})$ is a unit is not a numerical accident: it is the same as saying that the ideal I generated by $\zeta - \zeta^{-1}$ is also generated by $\zeta^3 - \zeta^{-3}$, i.e., that I is preserved by the $\zeta \mapsto \zeta^3$ automorphism. This in turn can be seen from the factorization of I into prime ideals, together with the structure of Galois groups acting on prime ideals—a rigid structural feature that is not specific to the cyclotomic case.

Having a much larger Galois group means that P will have at most a small number of roots in any field of reasonable degree. This eliminates all known methods of efficiently performing computations with more than a small number of automorphisms.

It is of course still possible to compute a minimum-length basis for the log-unit lattice, but all known methods are very slow. Cohen's classic book "A course in computational algebraic number theory" [41, page 217] describes the task of computing "a system of fundamental units" (i.e., a basis for the log-unit lattice) as one of the five "main computational tasks of algebraic number theory". One can compute *some* basis in subexponential time by techniques similar to the number-field sieve for integer factorization, but for almost all P the resulting basis elements will not be very short and will not be close to orthogonal, and finding a very short basis takes exponential time by all known methods.

The theory of units generalizes to what are called "S-units"; see, e.g., [42, Chapter 7]. For any polynomial P it is trivial to write down *a few* independent S-units for various S; we have heard speculation that this would somehow allow attacks. We point out that this is a reinvention of a special case of "free relations",

a small speedup to NFS. Writing down a few independent S-units is much less than writing down an entire basis, and does not seem helpful for decoding the log-S-unit lattice. More broadly, this well-known lattice remains strong against all known decoding attacks, *except* attacks that exploit extremely special algebraic features of number fields with small Galois groups.

Finally, we choose q as an inert prime so that there are no ring homomorphisms from $(\mathbb{Z}/q)[x]/P$ to any smaller nonzero ring. The attack strategies of [51], [52], and [37] start by applying such homomorphisms;[10] the attacks are restricted in other ways, but we see no reason to provide homomorphisms to the attacker in the first place. The examples from [52] were shown in [35] to be breakable in a simpler way without homomorphisms (some noise components turn out to always be 0); but, as pointed out in [37, Section 2.2] (see also [34, Section 5]), many other examples have been broken by homomorphisms without being broken by the algorithm from [35] or any other known attacks. It is sometimes claimed that "modulus switching" makes the choice of q irrelevant, but an attacker switching from q to another modulus will noticeably increase noise, interfering with typical attack algorithms.

C Worst-case-to-average-case reductions

We now consider an argument against our NTRU Prime recommendation: specifically, an argument that using cyclotomic fields with split modulus (i.e., with P splitting into linear factors in $(\mathbb{Z}/q)[x]$) is *desirable* for security, whereas we recommend *against* this choice.

The argument begins with statements from Lyubashevsky, Peikert, and Regev [87] that the Ring-LWE problem—with cyclotomic P, split q, and a wide error—has "very strong hardness guarantees" and in turn provides a "truly practical lattice-based public-key cryptosystem with an efficient security reduction". These statements allude to a conversion

- from any attack algorithm against the cryptosystem
- into an algorithm to solve the worst case of a "hard" SVP-like ideal-lattice problem.

This conversion is, internally, the composition of three theorems, first producing an algorithm to attack Decision-Ring-LWE, then producing an algorithm to attack Search-Ring-LWE, and finally producing an algorithm to attack the "hard" problem.

These statements and theorems have created a common belief that some "truly practical" lattice-based cryptosystems are guaranteed to be secure. The particular cryptosystems that are subjected to this belief have cyclotomic P

[10] Exactly the same homomorphisms are used in [95, Section 3, "Attack Framework"], where they are relabeled as "reduction modulo an ideal divisor". Choosing an inert prime eliminates all nontrivial ideal divisors.

(occasionally generalized to Galois P) and split q. The belief is not directly contradicted by, e.g., attacks against the Smart–Vercauteren system: no theorems of this type have been proven for that system.

A more extreme argument against NTRU Prime—*and* against NTRU *and* Ring-LWE-based systems—is the argument that one should actually use the original LWE problem. This argument begins with similar theorems, but this time the conversion (introduced by Regev [102]) applies to different "LWE-based" cryptosystems, and the conversion ends with the worst case of a "hard" SVP-like lattice problem. If all relevant parameters are equal then this problem is clearly at least as difficult to break as the worst case of an SVP-like ideal-lattice problem, since ideal lattices are a special case of lattices.

We have five counterarguments to both of these arguments. First, asymptotic attacks against SVP have improved dramatically in the last few years, reducing the asymptotic security level of d-dimensional lattices from approximately $0.41d$ bits to approximately $0.29d$ bits. This does not mean that there is any loss of security in, e.g., NTRU (see Appendix M below), but it calls into question the notion that SVP has been thoroughly studied.

Second, even in the extreme context of LWE, the allegedly "hard" SVP-like lattice problems are *not* the classic SVP problem. The same issue is even more obvious in the context of Ring-LWE: the "hard" SVP-like ideal-lattice problems are considerably more complicated, and less attractive to cryptanalysts, than truly well-known problems such as SVP. It is not easy to justify the notion that these allegedly "hard" problems have been studied more thoroughly than, e.g., the problem of breaking the NTRU cryptosystem. Simply labeling problems as "hard" does not make them so.

Third, it is not true that the cryptosystems have been proven to be as difficult to break as the "hard" problems. The underlying issue is that the conversion is very far from tight. Even if one assumes that there are no better attacks against the "hard" problems than the attacks known today, the conversion does not guarantee a reasonable cryptographic security level for any reasonably efficient cryptosystem. We have not found any paper proposing a specific lattice-based cryptosystem for which the conversion is meaningful.

Our work analyzing the tightness of this conversion is less detailed than an independent analysis recently posted [36, Section 6] by Chatterjee, Koblitz, Menezes, and Sarkar. The analysis in [36] features an astonishing 2^{504} tightness gap for reasonable LWE parameters. The analysis concludes that there is a "flimsy scientific foundation" for "the claim that, because of worst-case/average-case reductions, the more recent lattice-based encryption schemes have better security than classical NTRU".

Fourth, even if the conversion were tight, switching to these "provable" cryptosystems would impose considerable costs. One example of these costs is quantified in the analysis of [31]. Another example of these costs is implicit in the security analysis of [5]: the Ring-LWE parameters in [5] are not chosen to ensure the hardness of Ring-LWE, but merely to ensure hardness against attacks using

the very small number of output samples actually provided by the cryptosystem. Secure parameters for Ring-LWE would be more expensive.

These costs raise an important question mentioned in Appendix A: would it be better for security to use the same resources in other ways? In particular, taking larger parameters in the original cryptosystem would straightforwardly increase security against all known attacks. Compared to this option, switching to "provable" cryptosystems at the same cost means *reducing* security against known attacks. The argument to impose this security loss is, fundamentally, speculation that attacks against the original cryptosystem will improve much more than attacks against an SVP-like ideal-lattice problem that is claimed to be "hard" but that has little evidence of serious study.

Replacing Ring-LWE with the more extreme case of LWE somewhat simplifies the statement of the "hard" problem but imposes even more serious costs: "size be damned", in the words of a commentator who does not wish to be identified. Switching to LWE is likely to make network traffic a bottleneck. If this is affordable then much larger parameters in the original cryptosystem should also be affordable, producing an even larger increase in security against all known attacks. The argument for LWE thus rests on an even more extreme speculation regarding the progress of attacks.

For comparison, our recommendation to switch from NTRU Classic and NTRU NTT to NTRU Prime has low costs (as illustrated by the efficiency of Streamlined NTRU Prime), so it does not produce a noticeable security loss against known attacks. This recommendation is thus quite different from recommendations to switch to cryptosystems based on Ring-LWE, or more extreme cryptosystems based on LWE.

Notice that none of these four counterarguments are questioning the *correctness* of the proofs of the aforementioned theorems. The core problem is that, even if the theorems are correct exactly as stated, there are severe restrictions in what the theorems actually say.

Fifth, the very recent paper [97] says that it has generalized the result of [87] to arbitrary P and arbitrary q. The central argument for using cyclotomic fields thus appears to have been nothing more than a temporary deficiency in proof techniques. We caution the reader that this generalization has the same weaknesses mentioned above: there is no guarantee of the hardness of SVP, there is no guarantee that the "hard" problems are as hard as SVP, there is a severe lack of tightness, and there are considerable costs.

E Public-key encryption vs. unauthenticated key exchange

Our speed comparison in Section 1 focused on the cost of multiplication: for example, 31000 Haswell cycles for the NTRU NTT ring $(\mathbb{Z}/12289)[x]/(x^{1024}+1)$ used in New Hope [5]. However, the complete cryptographic operations reported in [5] cost 110986 cycles for the client and $88920+19422 = 108342$ cycles for the server, a total cost similar to the cost of seven multiplications.

There are 4 multiplications in [5], plus considerable overhead. The cost in [5] for 4 NTT multiplications is 4 pointwise multiplications and just 6 NTTs, rather than the 4 pointwise multiplications and 12 NTTs that one might expect; the protocol in [5] skips some NTTs.

We report 28682 Haswell cycles for our case study $(\mathbb{Z}/4591)[x]/(x^{761}-x-1)$ of an NTRU Prime ring; this is not much faster than New Hope multiplication. We also use 4 multiplications. Our current software uses multiplication as a black box. But our costs are just 59600 cycles for the client and 97452 cycles for the server.

Overall New Hope spends 40% more time than Streamlined NTRU Prime 4591^{761} for the client and server to share a fresh 256-bit session key.[11] More importantly, New Hope sends much more network traffic. In the latest version of [5], the server sends 1824 bytes and the client sends 2048 bytes. For us, the server sends a 1218-byte public key and the client sends a 1047-byte ciphertext, saving 42% of the New Hope network traffic. If the client already knows the public key then it simply sends a ciphertext, saving 73% of the New Hope network traffic.

New Hope's larger dimension can be viewed as providing an additional buffer against the possibility of improved attacks, as mentioned in Section 1; but this difference does not explain the gap in network traffic. The point of this appendix is to analyze this gap.

E.1. How servers are identified. Fundamentally, [5] and this paper are actually taking two completely different approaches to securing communication. Both approaches support the most urgent goal of post-quantum cryptography, namely encrypting today's data in a way that will not be decrypted by future quantum computers. Both approaches also support server authentication, so that the client will not be fooled into encrypting data to a "man in the middle" rather than the server. However, the details and costs of these two approaches are quite different.

In the first approach, the server's long-term identifier is a public key for a *signature system*. To start a secure session, the client and server perform an unauthenticated post-quantum key exchange (as in [5]), obtaining a shared secret key used to authenticate and encrypt subsequent messages by standard symmetric techniques. The server signs a hash of the key exchange, so the client knows that it is talking to the server; this is what stops the "man in the middle". At the end of the session, the client and server erase the shared secret key.

In the second approach, the server's long-term identifier is a public key for an *encryption system*. To start a secure session, the client sends a ciphertext to the server. Decryption provides both the client and the server with a shared secret key used to authenticate and encrypt subsequent messages; see our discussion

[11] The exact speed gap depends on further optimizations. For example, Gueron and Schlieker [59] modified the New Hope protocol to reduce the overhead for generating a uniform random polynomial; an exact speed comparison requires more benchmarking work, since [59] switched from Haswell to Skylake. We can certainly save some time in Streamlined NTRU Prime by caching transforms of repeated multiplication inputs such as f, and by speeding up the multiplication modulo 3.

of KEMs in Section 3.5. The client knows that it is talking to the server since nobody else has the shared secret key.

The second approach is less expensive for several reasons:

- In the first approach, the client cost is signature verification *plus* the client side of unauthenticated key exchange. In the second approach, the client cost is merely one public-key encryption.
- In the first approach, the server cost is signature generation *plus* the server side of unauthenticated key exchange. In the second approach, the server cost is merely one decryption.
- In the first approach, the network traffic is a signature *plus* unauthenticated key exchange. In the second approach, the network traffic is merely one ciphertext.

As a concrete example of total costs for the first approach, the recent paper [45, Table 2] combines [5] with a lattice-based signature system, reporting a total of more than 4700 bytes for post-quantum authenticated-server key exchange after significant compression effort. CPU time is not reported in [45] but presumably the signatures add significantly to the cost of [5].

As another example, [5] suggests combining post-quantum unauthenticated key exchange with the current world of pre-quantum signatures. The total traffic is then 3936 bytes, assuming a typical 64-byte ECC signature. The total CPU time is the time reported in [5] *plus* the cost of generating and verifying an ECC signature.

This type of combination provides *transitional* security: if the signature is verified before the attacker has a quantum computer then both integrity and confidentiality are protected, even against future quantum computers. However, it does not provide post-quantum security: if the signature is verified *after* the attacker has a quantum computer then neither integrity nor confidentiality is protected. Users will thus need *another* upgrade, switching all deployments to post-quantum signatures before attackers have quantum computers.

For comparison, we use just 1047 bytes of network traffic to set up a session key with true post-quantum server authentication. There is no need for a subsequent upgrade.

E.2. Key erasure ("forward secrecy"). An attacker who steals physical server hardware has a copy of the server's long-term secret key. The attacker can pose as the server for as long as this key is valid: often 90 days, often much longer. Furthermore, if the key is an encryption key, then the attacker can decrypt any previously recorded ciphertexts for this key. The low-cost encryption approach described above does not provide fast key erasure: there are many ciphertexts encrypted to the server's long-term key, and the plaintexts expose useful information.

For comparison, in a TLS "ECDHE" session, the client and server exchange short-term ECC keys and use the corresponding Diffie–Hellman shared secret as a session key. The client and server subsequently switch to new short-term ECC keys and erase their old secret keys, so they have no way to recompute

total bytes	client bytes	server bytes	key erasure	source
1047	1047	0	no	this paper
3312	1047+1047	1218	yes	this paper
3312	1047+1218	1047	yes	this paper
3872+signature	2048	1824+signature	yes	[5]+signature

Fig. E.1. Bandwidth used by different techniques of authenticated-server key exchange, assuming client already knows server's long-term key. In first line, long-term key is encryption key; client sends ciphertext; session key is hash of plaintext. In second line, server also sends short-term encryption key; client sends another ciphertext to that key; session key is hash of two plaintexts. In third line, short-term encryption key is generated by client rather than server. In fourth line, long-term key is signature key, and server signs hash of unauthenticated key exchange.

the shared secret. Servers nevertheless retain keys for "many months" in some cases, as explained in [81], but this period is no longer tied to the lifetime of the signature key that identifies the server; it is possible for a good implementation to erase keys much more quickly. Generating a new short-term key every minute has negligible cost.

The least expensive way to add post-quantum security to ECDHE is to add a layer of post-quantum public-key encryption using a long-term server key. Attackers stealing keys are stopped by ECDHE if they do not have quantum computers, and attackers with quantum computers are stopped by the post-quantum encryption if they are not stealing keys. However, it might still be possible for an attacker to use key theft to break the post-quantum system *and* a quantum computer to break ECDHE, so there is a comprehensible security benefit to deploying post-quantum key erasure.

Post-quantum key erasure is compatible with the second approach described above, although it does increase costs. The simplest protocol allowing key erasure is as follows. The server maintains a long-term post-quantum public key and also a short-term post-quantum public key. The client sends a ciphertext to each of these keys, and the two plaintexts are hashed to produce the shared secret key. A server-generated MAC under this secret key implicitly confirms server ownership of the short-term public key, at which point the client can safely send secret data under the same key. An attacker who later steals the server's secrets does not know the short-term secret key (the key has been erased), cannot decrypt the ciphertext sent to that key, and cannot compute the shared secret.

With this protocol, the client performs two public-key encryption operations, and the server performs two decryption operations, assuming key-generation costs are amortized. This is an *authenticated-server* key-exchange protocol using the same amount of computation as a naive *unauthenticated* key-exchange protocol in which the client sends a ciphertext to a short-term server key and the server sends a ciphertext to a short-term client key. One would expect an optimized unauthenticated key-exchange protocol to be somewhat faster than this, but it is not at all clear that this speedup can outweigh the costs of generating and verifying signatures.

As for bandwidth, this protocol requires the server to send its short-term post-quantum public key to the client. This could be smaller or larger than the signature in the first approach. See Figure E.1 for a comparison of the bandwidth of various techniques for authenticated-server key exchange.

One can alternatively have the client send a short-term public key and a ciphertext to the server's long-term public key, and the server send a ciphertext to the client's short-term public key, with MACs at each moment under secret keys obtained by hashing all plaintexts received so far. Similar comments apply to authenticated-client authenticated-server key exchange, but for simplicity we skip the details.

The general idea of all of these protocols is that, instead of checking a signature from X's long-term signature public key, one can check a MAC from X under a secret key derived from a plaintext sent to X's long-term encryption public key. Of course this plaintext needs to be generated with enough randomness: for example, it can simply be the output of KEM decapsulation.

To summarize, a post-quantum public-key cryptosystem is a simple, highly flexible tool that provides all desired long-term security features. Combining signatures with unauthenticated key exchange is more complicated, does not add any security features, and does not seem to provide significant performance benefits.

We emphasize that, as in the bulk of the cryptographic literature, we assume that each party can generate random numbers. Occasionally the literature considers the theory of deterministic parties (or, equivalently, parties with completely broken random-number generators); authentication to a deterministic party is equivalent to signing, and encryption to or from a deterministic party cannot meet the standard definitions of security.

E.3. Questioning the value of unauthenticated key exchange. With the above analysis in mind, we review the arguments given in [5, Section 2.3] for studying unauthenticated key exchange.

The first argument is that the "protection of stored transcripts against future decryption using quantum computers" is "much more urgent" than post-quantum authentication. This is indisputably the most urgent issue for many users. However, unauthenticated key exchange is not the only way to achieve this protection: a pure post-quantum public-key cryptosystem provides the same protection, and *also* straightforwardly provides authentication.

The second argument is that the unauthenticated key-exchange protocol in [5] is simpler than earlier authenticated key-exchange protocols. However, this ignores the possibility of building authenticated key exchange in an even simpler way from post-quantum public-key encryption.

The third argument is that combining unauthenticated key exchange with signatures allows the two components to be designed and optimized separately. However, it is not clear how this is better than building the same security features from a single well-optimized component, namely a public-key cryptosystem.

F Frequently asked questions

Why is rounding better than random errors? Rounding produces shorter ciphertexts and is deterministic. See the comparison of "Rounded NTRU" and "Noisy NTRU" in Section 3.3, and see the review of Dent's simple KEM (which relies on determinism) in Section 3.5.

Why not add random errors and *then* round? This is an older idea to shorten ciphertexts. See Section 3.3 for references. Pure rounding has the advantage of being deterministic.

Isn't there a proof that some Ring-LWE-based cryptosystem is at least as secure as lattice problems? No, not for the range of key sizes considered in this paper. See Appendix C.

Isn't there a proof that some Ring-LWE-based cryptosystem is more secure than NTRU? Or at least as secure? No. Either system could be more secure. Each system has security concerns (illustrated by attacks on extreme parameters) that are inapplicable to the other system. See the comparison of "Product NTRU" and "Quotient NTRU" in Section 3.4.

K An attack on a KEM

As a spinoff of analyzing KEM options, we found a fast chosen-ciphertext attack against the code-based KEM proposed in [98].

The problem is that [98] switches to predictable KEM output if decoding fails. The attacker can easily modify the ciphertext to flip a small number of bits (e.g., one or two bits) in the unknown error vector and to generate an authenticator from the predictable KEM output; decryption will succeed if the flipped error vector is decodable, and will almost certainly fail otherwise. Repeating these modifications quickly reveals the unknown error vector from the pattern of decryption failures, similarly to Berson's attack [24] on the plain McEliece system.

McBits [21] avoids this problem, because it uses a separate output bit from the KEM to indicate a decoding failure.

L Proof of Theorem 2.1

Proof. We first show that fm has coefficients in $[-4t, 4t]$. Let $f = \sum_{i=0}^{p-1} f_i x^i$, a ternary polynomial of exactly $2t$ terms. Let $m = \sum_{i=0}^{p-1} m_i x^i$ a ternary polynomial. Now we rewrite fm as $\sum_{i=0}^{p-1} f_i(x^i m)$. Since this sum adds at most $2t$ terms of each degree, it just remains to be shown that the coefficients of $x^i m$ are all in the range $[-2, 2]$. To show this we observe that $xm \equiv m_{p-1} + (m_0 + m_{p-1})x + m_1 x^2 + \cdots + m_{p-2}x^{p-1} \bmod x^p - x - 1$, which has coefficients in the range $[-1, 1]$, except $|m_0 + m_{p-1}|$ may be 2. Generalizing this, for $2 \leq i < p$,

we get $x^i m \equiv m_{p-i} + (m_{p-i} + m_{p-i-1})x + \cdots + (m_{p-2} + m_{p-1})x^{i-1} + (m_{p-1} + m_0)x^i + m_1 x^{i+1} + \cdots + m_{p-i-1}x^{p-1} \bmod x^p - x - 1$ with coefficients in $[-2, 2]$. Similar reasoning for gr implies that each coefficient of $gr \bmod x^p - x - 1$ is in $[-4t, 4t]$. Hence each coefficient of $3fm + gr \bmod x^p - x - 1$ is in $[-16t, 16t]$. □

M Memory, parallelization, and sieving algorithms

The security estimates in Section 4 rely on enumeration algorithms [99,54,73,63]. For very large dimensions, the performance of enumeration algorithms is slightly super-exponential and is known to be suboptimal. The provable sieving algorithms of Pujol and Stehlé [101] solve dimension-β SVP in time $2^{2.465\ldots\beta+o(\beta)}$ and space $2^{1.233\ldots\beta+o(\beta)}$, and more recent SVP algorithms [2] take time $2^{\beta+o(\beta)}$. More importantly, under heuristic assumptions, sieving is much faster. The most recent work on lattice sieving (see [15,79]) has pushed the heuristic complexity down to $2^{0.292\ldots\beta+o(\beta)}$.

Simply comparing 0.292β to enumeration exponents suggests that sieving could be faster than enumeration for sizes of β of relevance to cryptography. However, this comparison ignores two critical caveats regarding the performance of sieving. First, a closer look at polynomial factors indicates that the $o(\beta)$ here is positive. Consider, e.g., [15, Figure 3], which reports a best fit of $2^{0.387\beta-15}$ for its fastest sieving experiments. The comparison in [91] takes this caveat into account and concludes that the sieving cutoff is "far out of reach".

Second, sieving needs much more storage as β grows: at least $2^{0.208\ldots\beta+o(\beta)}$ bits of storage, again with positive $o(\beta)$. Furthermore, sieving is bottlenecked by random access to storage, and this random access also becomes slower as the amount of storage increases. The slowdown is approximately the square root of the storage in realistic cost models; see, e.g., [29].

Enumeration fits into very little memory even for large β. Kuo, Schneider, Dagdelen, Reichelt, Buchmann, Cheng, and Yang [78] showed that enumeration parallelizes effectively within and across GPUs. An attacker who can afford enough hardware for sieving for large β can instead use the same amount of hardware for enumeration, obtaining an almost linear parallelization speedup.

We do not mean to suggest that the operation-count ratio should be multiplied by the sieving storage (accounting for this enumeration speedup) *and* further by the square root of the storage (accounting for the cost of random access inside sieving): this would ignore the possibility of a speedup from parallelizing sieving. "Mesh" sorting algorithms such as the Schnorr–Shamir algorithm [107] sort n small items in time just $O(\sqrt{n})$, which is optimal in realistic models of parallel computation; these algorithms can be used as subroutines inside sieving, reducing the asymptotic cost penalty to just $2^{0.104\ldots\beta+o(\beta)}$. However, this is still much less effective parallelization than [78].

This cost penalty for sieving is ignored in measurements such as [88] and [15, Figure 3], and in the resulting comparisons such as [91]. These measurements are limited to sieving sizes that fit into DRAM on a single computer, and do not account for the important increase in memory cost as β increases. Another way

to see the same issue would be to scale sieving *down* to a small enough size to fit into GPU multiprocessors; this would demonstrate a sieving speedup for smaller β, for fundamentally the same reason that there will be a sieving slowdown for larger β.

In the absence of any realistic analyses of sieving cost for large β, we have decided to omit sieving from our security estimates. There is very little reason to believe that sieving can beat enumeration inside any attack that fits within our 2^{128} security target.

P Parameters

The following table shows Streamlined NTRU Prime parameter sets with $465 < p < 970$ and $q < 20000$. The estimated pre-quantum security level is 2^λ. Parameter sets with $\lambda < 128$ are omitted. Key size is computed as $\lceil p \log_2(q) \rceil$ (see Section 2.2).

Parameters				
p	q	t	λ	key size
467	3911	122	130	5573
479	5689	159	132	5976
	6089	159	130	6022
491	6287	163	135	6196
	8627	163	130	6420
	9277	163	129	6472
499	8243	166	134	4692
	9029	166	132	6558
503	2879	89	137	5781
	8663	167	134	6580
509	10939	169	133	6830
	11087	169	133	6840
521	15271	173	132	7242
	15359	173	132	7246
	16001	173	132	7277
523	3331	104	147	6121
	7151	174	146	6697
	7159	174	146	6698
	11003	174	138	7022
	19853	174	129	7467
541	2297	71	140	6041
	2437	76	150	6087
	17789	180	137	7639
557	4759	148	164	6805
	9323	185	155	7345
	13339	185	148	7633
	13963	185	148	7669
	14827	185	147	7718
	18719	185	143	7906
563	10627	187	155	7531
569	1579	49	129	6046
	3929	122	166	6794
	11489	189	156	7675
	12781	189	154	7763
	16193	189	150	7957
571	4201	131	168	6873
	7177	190	166	7315
577	1861	58	141	6268
	19081	192	150	8205
587	5233	163	176	7252
	8263	195	170	7639
593	1381	43	131	6186
	14423	197	161	8193
	19697	197	156	8460
599	7001	199	178	7652
	9551	199	171	7920
601	17257	200	161	8460
	18701	200	159	8529
607	6317	197	184	7664
	17747	202	163	8568
	18749	202	162	8617
613	1459	45	137	6444
	3319	103	174	7170
	4363	136	183	7412
	9157	204	178	8068
	10529	204	175	8191
	11867	204	173	8297
	12109	204	173	8315
	13799	204	170	8431
617	19469	204	164	8735
	1511	47	140	6517
	12829	205	173	8421
	16007	205	169	8618
	16073	205	169	8621
619	2297	71	160	6912
	6907	206	187	7895
	9397	206	180	8170
	9679	206	180	8196
	12203	206	175	8403
	13933	206	172	8522
	14173	206	172	8537

Parameters				
p	q	t	λ	key size
631	2081	65	158	6956
	2693	84	171	7191
	11287	210	182	8495
	16481	210	174	8840
641	13691	213	182	8808
643	6247	195	199	8108
	14737	214	181	8904
	15797	214	180	8969
	17189	214	178	9047
631	3559	111	188	7633
	16573	215	180	9069
	16883	215	180	9086
	18461	215	178	9170
653	2311	72	169	7297
	4621	144	199	7950
	8419	217	197	8515
	17477	217	182	9203
	17627	217	182	9211
	19163	217	180	9290
	19507	217	179	9307
659	2137	66	166	7290
	6781	211	205	8388
	7481	219	203	8481
	19571	219	182	9395
661	13327	220	191	9058
	15907	220	187	9226
673	1493	46	149	7097
	9413	224	204	8884
	17123	224	190	9465
677	3251	101	192	7899
683	5623	175	214	8509
	13313	227	200	9358
691	1499	46	147	7290
	5471	170	217	8581
	6449	201	219	8745
	12281	230	205	9387
701	17921	233	201	9905
709	10337	236	217	9455
	11923	236	214	9601
719	2087	65	179	7929
	2351	73	185	8053
	5153	161	226	8867
	9133	239	225	9460
	10531	239	221	9608
	10739	239	221	9628
	14831	239	213	9963
	19079	239	208	10224
727	5827	182	232	9094
	12241	242	221	9873
	17317	242	213	10236
739	9829	246	232	9802
	10859	246	229	9908
	12713	246	225	10076
	17183	246	218	10397
	19429	246	216	10528
743	7541	235	240	9571
	11251	247	231	10000
	16451	247	221	10407
	17959	247	219	10501
751	3067	95	211	8699
	3823	119	223	8938
757	1193	37	138	7737
	3727	116	224	8981
	6869	214	247	9649
	7879	246	246	9799
	10979	252	237	10161
	12973	252	233	10344
	13789	252	231	10410
	14737	252	230	10483
761	1619	50	170	8113
	4091	127	229	9131

Parameters				
p	q	t	λ	key size
761	4591	143	248	9258
	7883	246	248	9851
	13829	253	233	10468
	14107	253	232	10490
	19001	253	225	10817
769	1433	44	162	8063
	6599	206	251	9758
	17729	256	231	10854
773	877	27	131	7558
	2099	65	190	8531
	8317	257	252	10066
	9811	257	247	10251
	13757	257	239	10628
787	4243	132	240	9485
797	1259	39	158	8208
809	1801	56	187	8749
	6113	191	265	10176
	14107	269	254	11152
811	8543	266	269	10593
	10457	270	263	10829
	11831	270	260	10974
	14083	270	255	11177
823	4513	141	255	9992
	8069	252	275	10681
	11197	274	267	11070
827	7219	225	276	10601
	9767	275	273	10961
	13159	275	264	11317
	19081	275	254	11760
829	1657	51	183	8866
	12227	276	267	11256
	13037	276	265	11333
	14107	276	263	11427
853	9721	284	285	11300
	12377	284	278	11597
	15511	284	272	11875
857	5167	161	274	10572
	13367	285	278	11747
	14797	285	275	11872
859	12487	286	280	11690
	17203	286	272	12087
863	1523	47	182	9125
	4111	128	263	10361
	8779	274	293	11306
881	3217	100	248	10265
	7673	239	300	11370
	15733	293	284	12283
883	8089	252	302	11463
	14639	294	287	12219
887	13007	295	292	12123
907	7727	241	311	11715
	8807	275	313	11886
	12109	302	304	12303
	15467	302	297	12622
	17389	302	293	12776
919	11827	306	311	12434
	13933	306	306	12652
	14771	306	304	12729
929	12953	309	313	12692
	14011	309	310	12797
	18229	309	302	13149
937	1823	56	206	10150
	12401	312	318	12742
941	2521	78	239	10634
	10781	313	324	12606
947	3917	122	282	11303
953	6343	198	322	12038
	8237	257	333	12397
	16693	317	316	13367
967	8243	257	338	12580

R Randomized projective coordinates

Rather than directly transmitting an NTRU public key h as an element of $\mathcal{R}/q$, one can transmit it as two elements $d, hd \in \mathcal{R}/q$, where d is chosen as a uniform random invertible element of $\mathcal{R}/q$. This was proposed by Banks and Shparlinski in [11] (along with the idea of not requiring h to be invertible), and was reused in [12] for an analogous code-based cryptosystem "CAKE". This is what would be called "randomized projective coordinates" in the ECC context, whereas simply sending h would be called "affine coordinates".

The advantage of representing h as a fraction $(hd)/d$ is that, for Quotient NTRU, the receiver can skip the division in the secret computation of the public key h. The receiver instead computes h as a fraction, and then multiplies the numerator and denominator by a uniform random invertible element of $\mathcal{R}/q$ to hide all information beyond what h would have revealed.

The obvious disadvantage of sending d, hd is that public keys become twice as large; a further disadvantage is that arithmetic on h turns into arithmetic on both d and hd. Key size is important, and we expect key generation to be amortized across many uses of h, so we have skipped this alternative in Streamlined NTRU Prime. We have also skipped the idea of supporting both key formats as a run-time option: this would complicate implementations.

S Fast sorting, and fast constant-time sorting

We wrote a simple general-purpose library to sort an array of n signed 32-bit integers in constant time. Here "constant time" means that the time depends only on n, not on the values of the integers; all branches and all memory addresses are independent of the values of the integers.

Our library takes approximately $17.8n$ Haswell cycles for $n = 16$; $20.6n$ Haswell cycles for $n = 256$; $25.3n$ Haswell cycles for the case $n = 761$ used in Appendix T; $25.7n$ Haswell cycles for $n = 1024$; $31.6n$ Haswell cycles for $n = 4096$; $67.5n$ Haswell cycles for $n = 65536$ (no longer fitting into level-1 cache); and $131.5n$ Haswell cycles for $n = 1048576$ (no longer fitting into level-2 cache). The library sorts the array in place, using a negligible amount of temporary storage.

For comparison, [58] reported that Intel's "Integrated Performance Primitives" library took about $130n$ Haswell cycles for $n = 1000$ and about $160n$ Haswell cycles for $n = 1000000$. These results use radix sort, which is not constant time (memory addresses depend on the values of the integers being sorted, presumably leaking information through cache timing) and not in-place.

We are not aware of any faster Haswell results than IPP in the literature. Our library is about $5\times$ faster for $n = 1000$ and about $1.2\times$ faster for $n = 1000000$, setting new speed records for in-memory sorting at a wide range of sizes.

We now explain how we constructed our sorting library.

Most sorting methods (see generally [75]) fail our constant-time requirement when they are implemented in a straightforward way. There are easy generic

```
void int32_sort(crypto_int32 *x,int n)
{
  int top,p,q,i;

  if (n < 2) return;
  top = 1;
  while (top < n - top) top += top;

  for (p = top;p > 0;p >>= 1) {
    for (i = 0;i < n - p;++i)
      if (!(i & p))
        minmax(x + i,x + i + p);
    for (q = top;q > p;q >>= 1)
      for (i = 0;i < n - q;++i)
        if (!(i & p))
          minmax(x + i + p,x + i + q);
  }
}
```

Fig. S.1. Reference implementation of Batcher's merge exchange.

transformations into constant-time algorithms, but the resulting algorithms typically take time quadratic in n. The main exceptions are *sorting networks*.

An n-input sorting network is a sequence of instructions that correctly sorts every possible n-entry array $(K_0, K_1, \ldots, K_{n-1})$. Each instruction is a pair (i, j), specifying that array entries K_i and K_j are replaced by, respectively, $\min\{K_i, K_j\}$ and $\max\{K_i, K_j\}$. As Knuth puts it in [75, Section 5.3.4], a sorting network is an "*oblivious* sequence of comparisons, in the sense that whenever we compare K_i versus K_j the subsequent comparisons for the case $K_i < K_j$ are exactly the same as for the case $K_i > K_j$, but with i and j interchanged".

This obliviousness—the fact that the comparison is forgotten after it is used to sort (K_i, K_j), and that the indices (i, j) are then independent of the input—allows a particularly cheap transformation into a constant-time sorting algorithm: one simply has to perform each min-max computation in constant time. This approach to constant-time sorting is certainly not new: it was used in, e.g., [21].

However, the same obliviousness also has a well-known cost. The sorting network we use (see below) involves $(1/4)(e^2-e+4)n-1$ comparisons in the simplest case $n = 2^e$, while a non-oblivious comparison-based sorting algorithm such as heapsort involves only about en comparisons, saving a factor around $e/4$. Sorting networks are thus generally believed to be non-competitive in performance, except when n is tiny.

We point out that well-known trends in CPU microarchitecture are making sorting networks competitive for a much wider range of n. Concretely, the vector units on a single Haswell core can, every cycle, perform 8 "min" operations on 32-bit integers and also 8 "max" operations on 32-bit integers. There are also separate load/store units that can load 16 32-bit integers every cycle and store 8 32-bit integers every cycle. The Haswell is far less efficient at performing the basic operations used in other sorting algorithms, such as data-dependent branches or (in radix sort) updates of counters at variable memory locations.

The data flow in vector units is constrained. For example, the "min" operation computes a 256-bit vector $(\min\{a_0, b_0\}, \ldots, \min\{a_7, b_7\})$ given vectors $(a_0, \ldots, a_7)$ and $(b_0, \ldots, b_7)$. If one wants to instead compare, e.g., a_0 with b_3 then a vector permutation is required, taking some extra time. The cost of a sorting network thus depends not only on the number of min-max operations, but also on how those operations are laid out.

We use the classic "odd-even merging network" introduced by Batcher [13] in 1968. Figure S.1 is a C translation of Knuth's "merge exchange" in [75, Algorithm 5.2.2M], a simplified presentation of Batcher's odd-even merging network. Beware that many other descriptions of Batcher's method require n to be a power of 2.

To understand the vectorizability of this sorting network, consider the first `i` loop in Figure S.1 for $n = 761$. The `top` variable is set to 512, and the `p` variable starts at 512. The `i` loop compares `x[0]` with `x[512]`, compares `x[1]` with `x[513]`, etc., and finally compares `x[248]` with `x[760]`. To vectorize this we simply pick up `x[0]` through `x[7]` as a vector, pick up `x[512]` through `x[519]` as a vector, perform vector min and max operations, etc.

Later in the computation, when `p` is small, the vectorization becomes somewhat more intricate, requiring some permutations of vector entries.

The most obvious bottleneck in our computation is 2 cycles consumed by 2 vector stores for each 8 min-max operations. We could reduce the number of stores by merging levels of min-max operations, although this would complicate the control flow. More importantly, this bottleneck explains only about $5.9n$ Haswell cycles for, e.g., $n = 1024$; we have not yet analyzed the gap but suspect that there is still room for improvement.

T Further notes on constant-time computations

This appendix reviews ways to perform various computations inside NTRU in constant time. The sorting subroutine used here is presented separately in Appendix S because sorting is of interest far beyond the NTRU context. Any bounded-time computation can be mechanically converted into a constant-time computation, but the resulting time often expands quadratically; obtaining a *fast* constant-time computation often requires a new algorithm, as in Appendix S.

T.1. Generating polynomials of specified weight. The obvious way to generate t-small polynomials is choose a random position for a nonzero coefficient and repeat until the weight has reached $2t$. This takes variable time, raising security questions. Here is a constant-time alternative:

- Generate a target list $(\pm 1, \ldots, \pm 1, 0, \ldots, 0)$ starting with $2t$ nonzero entries, each chosen randomly as either 1 or -1.
- Use a constant-time algorithm to sort a list of p random numbers, and at the same time apply the same permutation to the target list. (This is a standard technique to randomly shuffle a target list—see, e.g., [75, Section 5, first answer to Exercise 11, or Exercise 13 in the next edition]—plus the extra constraint of selecting a constant-time sorting algorithm.)

This is theoretically perfect when the numbers do not collide.

We generate 32-bit random numbers, replace the bottom 2 bits with the target list, sort the numbers, and extract the bottom 2 bits. This produces 30-bit collisions once every few thousand ciphertexts, making smaller coefficients marginally more likely to appear closer to the beginning of the list. We could check for these collisions and restart if they occur, but the information leak is negligible.

Modern stream ciphers take only a few thousand cycles to generate 761 random 32-bit numbers. See Appendix S for the cost of sorting those numbers.

T.2. Inversion inside key generation. Key generation for Quotient NTRU involves computing a reciprocal (unless randomized projective coordinates are used; see Appendix R). In particular, key generation for Streamlined NTRU Prime involves computing $1/f$ in $\mathcal{R}/q = (\mathbb{Z}/q)[x]/(x^p - x - 1)$ and computing $1/g$ in $\mathcal{R}/3 = (\mathbb{Z}/3)[x]/(x^p - x - 1)$.

In NTRU Prime, $\mathbb{Z}/q$ and $\mathcal{R}/q$ are fields of size q and q^p respectively, so inversion is the same as computing $(q-2)$nd powers and (q^p-2)nd powers respectively by Fermat's little theorem. Standard exponentiation algorithms take constant time when the underlying multiplication algorithms take constant time. Similar comments apply to $\mathcal{R}/3$: the ring $\mathcal{R}/3$ is not necessarily a field, but one can still find an appropriate exponent, or invert separately modulo each factor of $x^p - x - 1$.

We instead use Stevin's 16th-century gcd algorithm (see, e.g., [84]), the polynomial version of Euclid's gcd algorithm. "Extended" gcd computations are the textbook method to compute modular reciprocals. A variant of Stevin's gcd algorithm often used for NTRU is the "almost-inverse" gcd algorithm (see, e.g., [111]), the polynomial version of Stein's binary gcd algorithm [115], which focuses on eliminating low bits rather than high bits.

Each of these gcd algorithms is naturally expressed as a series of divisions. In particular, Stevin's algorithm is a series of divisions. Each division, implemented in the textbook way, is a series of single-coefficient division steps. The number of division steps inside the division is a variable, the degree of the quotient.

As in [49], we replace the two-level series of series of division steps with a single-level series of division steps. The point of [49] is that the resulting algorithm—applied to the special case where one input is a power of x, and stopping with a "half-gcd"—is, aside from labeling, exactly the Berlekamp–Massey algorithm [89] for detecting linear recurrences.

Each conditional branch in the Berlekamp–Massey algorithm is a simple input selection, which can be efficiently simulated by constant-time arithmetic. There are also multiplications by variable powers of x, which at low cost can be absorbed into constant-time multiplications by x in earlier steps. A constant-time version of the Berlekamp–Massey algorithm was used in [21].

A generalized constant-time version of Stevin's algorithm is surprisingly simple when the details are chosen carefully. The algorithm state consists of two integers d, e; a polynomial f of constant degree p; and a polynomial g of de-

gree at most p. These polynomials are, respectively, x^{p-d} and x^{p-e} times two remainders appearing in Stevin's algorithm. The main loop has three steps:

- Subtract a multiple of f from g to clear the coefficient of x^p in g.
- Multiply g by x, while subtracting 1 from e.
- If $e < d$ and the coefficient of x^p in g is nonzero, swap (d, f) with (e, g).

For the extended version of the algorithm one also keeps track of f and g as multiples of the original g modulo the original f.

Our current key-generation software uses about 6 million cycles. With more effort one can eliminate most of these cycles, but our current key-generation cost is already negligible. Specifically:

- The standard design goal of IND-CCA2 security means that it is safe to generate a key once and use the key any number of times. The New Hope situation [5] is completely different: New Hope is not designed to resist, and does not resist, chosen-ciphertext attacks, so it generates a new key for every ciphertext, so its key-generation time is important.
- Forward secrecy (see generally Appendix E) does *not* require constant generation of new keys. A typical quad-core 3GHz server generating a new short-term key every minute is using under 1/100000 of its CPU time on key generation with our current software.
- A user who (for some reason) wants to generate many 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. This reduces the cost of generating each key below 300000 cycles.
- If, despite all of the above, a user still notices the cost of our inversion software: "Fast gcd" techniques incorporate subquadratic-time multiplication methods such as Karatsuba's method, and are compatible with constant-time computations.

Our software is analogous to the original Curve25519 software [17], which emphasized encryption/decryption speed and did not bother speeding up occasional key-generation computations.

T.3. Product-form polynomials. NTRU papers starting with [66] have used "product-form polynomials", i.e., polynomials of the form $AB + C$. The weight of $AB + C$ is generally higher than the total weight of A, B, C (since the terms of A and B cross-multiply), and a rather small total weight of A, B, C maintains security against all known attacks. To multiply by $AB + C$ one can multiply by A, then multiply by B, then multiply the original input by C. This saves time for non-constant-time sparse-polynomial-multiplication algorithms, but it loses time for constant-time algorithms, so we ignore this idea. Even *with* this idea, the best speeds for NTRU using sparse polynomial multiplication are not competitive with our speeds.

Z Sage script (reference implementation)

Our reference implementation of Streamlined NTRU Prime 4591^{761} is the Sage script in Figures Z.1 and Z.2 (subsequent pages).

We also have a portable C reference implementation. Unlike the Sage script, the C reference implementation allows "benign malleability" of ciphertexts, as defined in [109]. Specifically, each 32-bit ciphertext word encodes three integers between 0 and 1530; if larger integers appear then they are silently reduced modulo 1531. Similar comments apply to public keys.

Except for this malleability, the reference implementations are intended to produce identical results. The Sage script is more concise, so any discrepancy should be presumed to be a bug in the C implementation. We have checked that the two implementations produce identical outputs in a series of tests where the underlying `random32` functions are synchronized. Further auditing is of course required before deployment of the software.

```
p = 761; q61 = 765; q = 6*q61+1; t = 143
Zx.<x> = ZZ[]; R.<xp> = Zx.quotient(x^p-x-1)
Fq = GF(q); Fqx.<xq> = Fq[]; Rq.<xqp> = Fqx.quotient(x^p-x-1)
F3 = GF(3); F3x.<x3> = F3[]; R3.<x3p> = F3x.quotient(x^p-x-1)

import hashlib
def hash(s): h = hashlib.sha512(); h.update(s); return h.digest()

def random32(): return randrange(-2^31,2^31)
def random32even(): return random32() & (-2)
def random321mod4(): return (random32() & (-3)) | 1
def randomrange3(): return ((random32() & 0x3fffffff) * 3) >> 30

import itertools
def concat(lists): return list(itertools.chain.from_iterable(lists))

def nicelift(u):
  return lift(u + q//2) - q//2

def nicemod3(u): # r in {0,1,-1} with u-r in {...,-3,0,3,...}
  return u - 3*round(u/3)

def int2str(u,bytes):
  return ''.join(chr((u//256^i)%256) for i in range(bytes))

def str2int(s):
  return sum(ord(s[i])*256^i for i in range(len(s)))

def seq2str(u,radix,batch,bytes): # radix^batch <= 256^bytes
  return ''.join(int2str(sum(u[i+t]*radix^t for t in range(batch)),bytes)
                 for i in range(0,len(u),batch))

def str2seq(s,radix,batch,bytes):
  u = [str2int(s[i:i+bytes]) for i in range(0,len(s),bytes)]
  return concat([(u[i]//radix^j)%radix for j in range(batch)] for i in range(len(u)))

def encodeZx(m): # assumes coefficients in range {-1,0,1}
  m = [m[i]+1 for i in range(p)] + [0]*(-p % 4)
  return seq2str(m,4,4,1)

def decodeZx(mstr):
  m = str2seq(mstr,4,4,1)
  return Zx([m[i]-1 for i in range(p)])

def encodeRq(h):
  h = [q//2 + nicelift(h[i]) for i in range(p)] + [0]*(-p % 5)
  return seq2str(h,6144,5,8)[:1218]

def decodeRq(hstr):
  h = str2seq(hstr,6144,5,8)
  if max(h) >= q: raise Exception("pk out of range")
  return Rq([h[i]-q//2 for i in range(p)])

def encoderoundedRq(c):
  c = [q61 + nicelift(c[i]/3) for i in range(p)] + [0]*(-p % 6)
  return seq2str(c,1536,3,4)[:1015]

def decoderoundedRq(cstr):
  c = str2seq(cstr,1536,3,4)
  if max(c) > q61*2: raise Exception("c out of range")
  return 3*Rq([c[i]-q61 for i in range(p)])
```

Fig. Z.1. Complete non-constant-time reference implementation of Streamlined NTRU Prime 4591^{761} using the Sage computer-algebra system, part 1: auxiliary functions for encoding and decoding of polynomials as byte strings.

```
def randomR(): # R element with 2t coeffs +-1
  L = [random32even() for i in range(2*t)]
  L += [random321mod4() for i in range(p-2*t)]
  L.sort()
  L = [(L[i]%4)-1 for i in range(p)]
  return Zx(L)

def keygen():
  while True:
    g = Zx([randomrange3()-1 for i in range(p)])
    if R3(g).is_unit(): break
  grecip = [nicemod3(lift(gri)) for gri in list(1/R3(g))]
  f = randomR()
  h = Rq(g)/(3*Rq(f))
  pk = encodeRq(h)
  return pk,encodeZx(f) + encodeZx(grecip) + pk

def encapsulate(pk):
  h = decodeRq(pk)
  r = randomR()
  hr = h * Rq(r)
  m = Zx([-nicemod3(nicelift(hr[i])) for i in range(p)])
  c = Rq(m) + hr
  fullkey = hash(encodeZx(r))
  return fullkey[:32] + encoderoundedRq(c),fullkey[32:]

def decapsulate(cstr,sk):
  f,ginv,h = decodeZx(sk[:191]),decodeZx(sk[191:382]),decodeRq(sk[382:])
  confirm,c = cstr[:32],decoderoundedRq(cstr[32:])
  f3mgr = Rq(3*f) * c
  f3mgr = [nicelift(f3mgr[i]) for i in range(p)]
  r = R3(ginv) * R3(f3mgr)
  r = Zx([nicemod3(lift(r[i])) for i in range(p)])
  hr = h * Rq(r)
  m = Zx([-nicemod3(nicelift(hr[i])) for i in range(p)])
  checkc = Rq(m) + hr
  fullkey = hash(encodeZx(r))
  if sum(r[i]==0 for i in range(p)) != p-2*t: return False
  if checkc != c: return False
  if fullkey[:32] != confirm: return False
  return fullkey[32:]

for keys in range(5):
  pk,sk = keygen()
  for ciphertexts in range(5):
    c,k = encapsulate(pk)
    assert decapsulate(c,sk) == k

print len(pk),'bytes in public key'
print len(sk),'bytes in secret key'
print len(c),'bytes in ciphertext'
print len(k),'bytes in shared secret'
```

Fig. Z.2. Complete non-constant-time reference implementation of Streamlined NTRU Prime 4591^{761} using the Sage computer-algebra system, part 2: key generation, encapsulation, decapsulation, tests.