# 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] INRIA and ENS 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, The Netherlands
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

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. The second author acknowledges the support of Bpifrance in the context of the national project RISQ (P141580). Calculations were carried out on the Saber cluster of the Cryptographic Implementations group at Technische Universiteit Eindhoven. Permanent ID of this document: 99a9debfc18b7d6937a13bac4f943a2b2cd46022. Date: 2017.10.04. See full version [10].


C. Adams and J. Camenisch (Eds.): SAC 2017, LNCS 10719, pp. 235–260, 2018.
https://doi.org/10.1007/978-3-319-72565-9_12

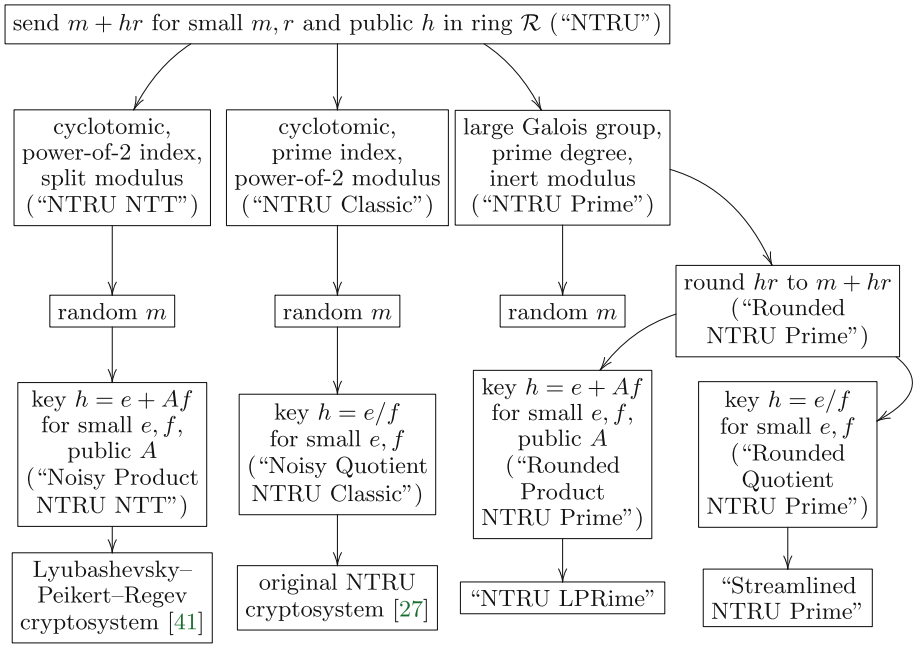


**Fig. 1.1.** Terminology in this paper for selected branches of the NTRU family tree. This paper introduces the NTRU Prime branch. Streamlined NTRU Prime is specified and analyzed in detail as a case study. See Sect. 3 for more options.

## 1 Introduction

This paper presents an efficient implementation of high-security **prime-degree large-Galois-group inert-modulus** ideal-lattice-based cryptography. "Prime degree" etc. are three features that we recommend because they take various mathematical tools away from the attacker; see Appendix A in the full version of this paper. The reader can, if desired, skip the appendix in favor of the following short summary short summary (see also Fig. 1.1):

- "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 [27], 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 [2], 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.[1]

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 [18,22] 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.[2]

**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" [2] by Alkim, Ducas, Pöppelmann, and Schwabe, using about 40000 Haswell cycles for NTRU NTT multiplication. Most of the implementations before [2] 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 [38] announced 33000 cycles for NTRU NTT multiplication the same month. An update of [2] in August 2016 announced 31000 cycles for NTRU NTT multiplication.[3] We now announce 28682 cycles for NTRU Prime multiplication. See Table 1.1 for details.

[1] 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.

[2] If an operation takes 100000 cycles then one can imagine a typical quad-core 3 GHz 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 100 Mbps network.

[3] Each forward NTT in the updated version of [2] takes 8448 cycles (compared to 10968 cycles in the first version, and 9100 cycles in [38, Table 1]). A reverse NTT takes 9464 cycles (compared to 12128 and 9300). The time for pointwise multiplication is not stated in [2] or [38] but can be extrapolated from [23] to take about 5000 cycles.

**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 [33].

| Rec | Constant | Cycles | Ring | Technique | Source |
|---|---|---|---|---|---|
| no | yes | 11722 | $(\mathbb{Z}/8192)[x]/(x^{701}-1)$ | Karatsuba etc. | [33] |
| 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 [2], [38] |
| no | no | <91056 | $(\mathbb{Z}/2048)[x]/(x^{743}-1)$ | Sparse input | `ntruees743ep1` [35] |

Like our paper, [2] and [38] target the Haswell CPU, require constant-time implementations, and aim for more than $2^{128}$ post-quantum security. Unlike our paper, [2] follows the tradition from NTRU and Ring-LWE [41] of using cyclotomic rings. More precisely, [2] is an example of Product NTRU NTT, using the ring $(\mathbb{Z}/q)[x]/(x^p+1)$ with $p = 1024$ and $q = 12289 = 12 \cdot 1024 + 1$.

A disadvantage of requiring the lattice dimension $p$ to be a power of 2, as in [2], is that security levels are quite widely separated. In [2] 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 [2] is unable to offer any options truly comparable to the better-tuned $p = 743$ in "`ntruees743ep1`" (see [35]) 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 [2] 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 [2,38], and the sizes are smaller.

We are not saying that the NTRU Prime rings have *zero* cost. Last month Hülsing, Rijneveld, Schanck, and Schwabe [33] 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[4] 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 [28] 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 [28].

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 [39] 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 [47] 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 [14] announced 119652 cycles for encapsulation and 125736 cycles for decapsulation using a new public-key cryptosystem "Kyber". (Preliminary

[4] The security level in [33] seems somewhat lower than the security level of Streamlined NTRU Prime $4591^{761}$. Taking a larger $p$ in [33] would require jumping to $q = 16384$, and the resulting ciphertext expansion seems likely to outweigh any small speed gap.

speeds announced in January 2017 [6] were slower.) This system uses Module-LWE [37] 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 [45] argued briefly that "one might wish to use Ring-LWE over non-Galois number fields". The argument is essentially one of the arguments from this paper, without credit. The main result of [45] is a worst-case-to-average-case reduction; see Appendix C in the full version of this paper.

**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 [2]; see Appendix E in the full version of this paper for a precise comparison.

We are submitting our complete implementation to eBACS [11] 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 in the full version of this paper show complete algorithms for key generation, encapsulation, and decapsulation in Streamlined NTRU Prime $4591^{761}$, using the Sage [48] 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 Fig. 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 Fig. 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 Sect. 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 Fig. 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 Fig. 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 Fig. 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 Fig. 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 Sect. 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 Sect. 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 Sect. 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 Sect. 3.5.

In the original NTRU specification [27], $m$ 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 [26] 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., Sect. 2.3. Second, the fact that $m$ is determined by $r$ simplifies protection against chosen-ciphertext attacks; see Sect. 3.5.

[43, Sect. 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 [7] in the context of a symmetric-key construction, was used in [4] to construct another public-key encryption system, and was further studied in [3,13]. 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 Sect. 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 [2]) 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 [41] (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 [46], 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 [34] 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 [54], 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.[5] We see no way to simultaneously 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 [51]. 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., [52]) and in practice (see, e.g., [42]), 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 [51] (also called "Simple RSA"), is much easier:

– Choose a uniform random integer $m$ modulo $n$. This step does not even look at the plaintext.

[5] Peikert claimed in [44], modulo terminology, that Product NTRU is "at least as hard" to break as Quotient NTRU (and "likely strictly harder"). This claim ignores the possibility of attacks against the reuse of $r$ in Product NTRU. There are no theorems justifying Peikert's claim, and we are not aware of an argument that eliminating this reuse is less important than eliminating the g/f structure. For comparison, switching from NTRU NTT and NTRU Classic to NTRU Prime eliminates structure used in some state-of-the-art attacks without providing new structure used in other attacks.

– To obtain a shared secret key, simply apply a cryptographic hash function to $m$.
– Encrypt and authenticate the sender's plaintext using this shared key.

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

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

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

It is easy to guess that KEMs simplify NTRU, the same way that KEMs simplify RSA; we are certainly not the first to suggest this. However, all the NTRU-based KEMs we have found in the literature (e.g., [49,53]) 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 Sect. 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 [19, Sect. 6] by Dent, backed by a tight security reduction [19, 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 [30] and, for an analogous attack on code-based cryptography using QC-MDPC codes, [24]. 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 Sect. 3.3, the coefficients of $m$ are chosen from the limited range $\{-1, 0, 1\}$. The NTRU literature [25–27,32] 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 Sect. 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 Sect. 3.5), saving space and time. Beware, however, that if $t$ is *too* small then there are attacks; see our security analysis in Sect. 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 Sect. 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 [21], and the even wider range in [2]. In [54], 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 Sect. 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 [15].

## 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 [12, 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 in the full version of this paper. 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 [29, 31] 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 $x^p$ mod $x^p - x - 1 = x + 1$, 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 [55,56]). 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 Sect. 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 [29]. 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 [50]. Therefore the strategy of the reduction phase is to apply lattice-basis reduction, for example BKZ 2.0 [16], 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 [29].

To estimate the security against this attack we adapt the analysis of [26] 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 $\delta$. 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 [16] to determine the best BKZ 2.0 parameters, specifically the "block size" $\beta$ and the number of "rounds" $n$, needed to reach a root Hermite factor $\delta$. To get a concrete security estimate of the work required to perform BKZ-2.0 with parameters $\beta$ and $n$ we use the conservative formula determined by [26] from the experiments of [17]:

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

This estimate and the underlying experiments rely on "enumeration"; see Appendix M in the full version of this paper 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 [57]. 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 $\{-d, -(d-1), \ldots, d-1, d\}$ by replacing base 2 in the exponentiations in Eqs. 1, 2 and 3 with $2d$. In this case however the range of $t$, by a generalization of Theorem 2.1, decreases to $q \geq 16(d^3 + d^2)t$.

**4.4. Algebraic Attacks.** The attack strategy of Ding [20], Arora–Ge [5], and Albrecht–Cid–Faugère–Fitzpatrick–Perret [1] 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 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 [1, Theorem 7] and [40, Case Study 1].

## 5 Parameters

Algorithm 1 searches for $(p, q, t, \lambda)$, where $\lambda$ is Sect. 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., [36] 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 Eq. (1) to determine the bitsecurity level of the parameters against a straightforward meet-in-the-middle attack. To find $w, w', \beta, n$ 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

**Algorithm 1.** Determine parameter sets for security level above $\ell$.

**Input**: Upper bound $q_b$ for $q$, range $[p_1, p_2]$ for $p$, lower bound $\ell$ for security level
**Result**: Viable parameters $p$, $q$ and $t$ with security level $\lambda$.
$p \leftarrow p_1 - 1$ (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 $w$, $w'$, $\beta$, $n$ 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** $p, q, t, \min\{\lambda_1, \lambda_2\}$

the hybrid attack are of equal cost. For each $w'$ we determine the Hermite factor required with Eq. (4), use the BKZ-2.0 simulator to determine the optimal $\beta$ and $n$ to reach the required Hermite factor and use Eqs. (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 in the full version of this paper.

## 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 in the full version of this paper.) 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 round$(7x/2^{15})$ where $x$ is the first integer. This is not always exactly 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 [2], 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 [9, Sect. 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 $24 \times 24$ 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 $768 \times 768$ product to 11 separate $128 \times 128$ products; then use 5 layers of refined Karatsuba.

We also experimented with variants of the Schönhage–Strassen multiplication method, starting from the framework of [8, Sect. 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. Albrecht, M.R., Cid, C., Faugère, J.-C., Fitzpatrick, R., Perret, L.: Algebraic algorithms for LWE problems (2014). https://eprint.iacr.org/2014/1018
2. Alkim, E., Ducas, L., Pöppelmann, T., Schwabe, P.: Post-quantum key exchange - A new hope. In: USENIX Security Symposium, pp. 327–343. USENIX (2016)
3. Alperin-Sheriff, J., Apon, D.: Dimension-preserving reductions from LWE to LWR. IACR Cryptology ePrint Archive 2016:589 (2016)
4. Alwen, J., Krenn, S., Pietrzak, K., Wichs, D.: Learning with rounding, revisited - new reduction. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013. LNCS, vol. 8042, pp. 57–74. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40041-4_4
5. Arora, S., Ge, R.: New algorithms for learning in presence of errors. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) ICALP 2011. LNCS, vol. 6755, pp. 403–415. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22006-7_34
6. Bai, S., Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J.M., Schwabe, P., Stehlé, D.: Crystals: cryptographic suite for algebraic lattices (2017). http://tinyurl.com/znsjrv5
7. Banerjee, A., Peikert, C., Rosen, A.: Pseudorandom functions and lattices. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 719–737. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29011-4_42
8. Bernstein, D.J.: Multidigit multiplication for mathematicians (2001). https://cr.yp.to/papers.html#m3
9. Bernstein, D.J.: Batch binary Edwards. In: Halevi, S. (ed.) CRYPTO 2009. LNCS, vol. 5677, pp. 317–336. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03356-8_19
10. Bernstein, D.J., Chuengsatiansup, C., Lange, T., van Vredendaal, C.: NTRU Prime: reducing attack surface at low cost (2017). https://eprint.iacr.org/2016/461. Full version of this paper

11. Bernstein, D.J., Lange, T.: eBACS: ECRYPT benchmarking of cryptographic systems. https://bench.cr.yp.to. Accessed 9 Feb 2017
12. Bernstein, D.J., Lange, T.: Non-uniform cracks in the concrete: the power of free precomputation. In: Sako, K., Sarkar, P. (eds.) ASIACRYPT 2013. LNCS, vol. 8270, pp. 321–340. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-42045-0_17
13. Bogdanov, A., Guo, S., Masny, D., Richelson, S., Rosen, A.: On the hardness of learning with rounding over small modulus. In: Kushilevitz, E., Malkin, T. (eds.) TCC 2016. LNCS, vol. 9562, pp. 209–224. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49096-9_9
14. Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J.M., Schwabe, P., Stehlé, D.: CRYSTALS - Kyber: a CCA-secure module-lattice-based KEM (2017). https://eprint.iacr.org/2017/634
15. Groot Bruinderink, L., Hülsing, A., Lange, T., Yarom, Y.: Flush, Gauss, and reload – a cache attack on the BLISS lattice-based signature scheme. In: Gierlichs, B., Poschmann, A.Y. (eds.) CHES 2016. LNCS, vol. 9813, pp. 323–345. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53140-2_16
16. Chen, Y., Nguyen, P.Q.: BKZ 2.0: better lattice security estimates. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 1–20. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25385-0_1
17. Chen, Y., Nguyen, P.Q.: BKZ 2.0: better lattice security estimates (full version) (2011). http://www.di.ens.fr/~ychen/research/Full_BKZ.pdf
18. Chou, T.: Sandy2x: New Curve25519 speed records. In: Dunkelman, O., Keliher, L. (eds.) SAC 2015. LNCS, vol. 9566, pp. 145–160. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-31301-6_8
19. Dent, A.W.: A Designer's guide to KEMs. In: Paterson, K.G. (ed.) Cryptography and Coding 2003. LNCS, vol. 2898, pp. 133–151. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-40974-8_12
20. Ding, J.: Solving LWE problem with bounded errors in polynomial time (2010). https://eprint.iacr.org/2010/558
21. Ducas, L., Durmus, A., Lepoint, T., Lyubashevsky, V.: Lattice signatures and bimodal Gaussians. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013. LNCS, vol. 8042, pp. 40–56. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40041-4_3
22. Faz-Hernández, A., López, J.: Fast implementation of Curve25519 using AVX2. In: Lauter, K., Rodríguez-Henríquez, F. (eds.) LATINCRYPT 2015. LNCS, vol. 9230, pp. 329–345. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-22174-8_18
23. Güneysu, T., Oder, T., Pöppelmann, T., Schwabe, P.: Software speed records for lattice-based signatures. In: Gaborit, P. (ed.) PQCrypto 2013. LNCS, vol. 7932, pp. 67–82. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38616-9_5
24. Guo, Q., Johansson, T., Stankovski, P.: A key recovery attack on MDPC with CCA security using decoding errors. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016. LNCS, vol. 10031, pp. 789–815. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53887-6_29
25. Hirschhorn, P.S., Hoffstein, J., Howgrave-Graham, N., Whyte, W.: Choosing NTRUEncrypt parameters in light of combined lattice reduction and MITM approaches. In: Abdalla, M., Pointcheval, D., Fouque, P.-A., Vergnaud, D. (eds.) ACNS 2009. LNCS, vol. 5536, pp. 437–455. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-01957-9_27

26. Hoffstein, J., Pipher, J., Schanck, J.M., Silverman, J.H., Whyte, W., Zhang, W.: Choosing parameters for NTRUEncrypt (2015). https://eprint.iacr.org/2015/708
27. Hoffstein, J., Pipher, J., Silverman, J.H.: NTRU: a ring-based public key cryptosystem. In: Buhler, J.P. (ed.) ANTS 1998. LNCS, vol. 1423, pp. 267–288. Springer, Heidelberg (1998). https://doi.org/10.1007/BFb0054868
28. Hoffstein, J., Pipher, J., Silverman, J.H.: NTRU: a new high speed public key cryptosystem (2016). Circulated privately in 1996; put online in 2016 at https://web.securityinnovation.com/hubfs/files/ntru-orig.pdf
29. Howgrave-Graham, N.: A hybrid lattice-reduction and meet-in-the-middle attack against NTRU. In: Menezes, A. (ed.) CRYPTO 2007. LNCS, vol. 4622, pp. 150–169. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74143-5_9
30. Howgrave-Graham, N., Nguyen, P.Q., Pointcheval, D., Proos, J., Silverman, J.H., Singer, A., Whyte, W.: The impact of decryption failures on the security of NTRU encryption. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 226–246. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45146-4_14
31. Howgrave-Graham, N., Silverman, J.H., Whyte, W.: A meet-in-the-middle attack on an NTRU private key. Technical report, NTRU Cryptosystems (2003). https://www.securityinnovation.com/uploads/Crypto/NTRUTech004v2.pdf
32. Howgrave-Graham, N., Silverman, J.H., Whyte, W.: Choosing parameter sets for NTRUEncrypt with NAEP and SVES-3 (2005). https://eprint.iacr.org/2005/045
33. Hülsing, A., Rijneveld, J., Schanck, J., Schwabe, P.: High-speed key encapsulation from NTRU. In: Fischer, W., Homma, N. (eds.) CHES 2017. LNCS, vol. 10529, pp. 232–252. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66787-4_12
34. Kirchner, P., Fouque, P.-A.: Comparison between subfield and straightforward attacks on NTRU (2016). https://eprint.iacr.org/2016/717
35. Kumar, V.: ntruees743ep1 software (2014). Included in [11]
36. Laarhoven, T., Mosca, M., van de Pol, J.: Finding shortest lattice vectors faster using quantum search. Des. Codes Cryptography **77**(2–3), 375–400 (2015)
37. Langlois, A., Stehlé, D.: Worst-case to average-case reductions for module lattices. Des. Codes Cryptography **75**(3), 565–599 (2015)
38. Longa, P., Naehrig, M.: Speeding up the number theoretic transform for faster ideal lattice-based cryptography. In: Foresti, S., Persiano, G. (eds.) CANS 2016. LNCS, vol. 10052, pp. 124–139. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-48965-0_8
39. Lyubashevsky, V.: Digital signatures based on the hardness of ideal lattice problems in all rings. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016. LNCS, vol. 10032, pp. 196–214. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53890-6_7
40. Lyubashevsky, V.: 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
41. Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. J. ACM **60**(6), 43 (2013)
42. Meyer, C., Somorovsky, J., Weiss, E., Schwenk, J., Schinzel, S., Tews, E.: Revisiting SSL/TLS implementations: new Bleichenbacher side channels and attacks. In: USENIX Security Symposium, pp. 733–748. USENIX (2014)
43. Peikert, C.: Public-key cryptosystems from the worst-case shortest vector problem: extended abstract. In: STOC, pp. 333–342. ACM (2009)
44. Peikert, C.: "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

45. Peikert, C., Regev, O., Stephens-Davidowitz, N.: Pseudorandomness of Ring-LWE for any ring and modulus. In: STOC, pp. 461–473. ACM (2017)
46. Pöppelmann, T., Güneysu, T.: Towards practical lattice-based public-key encryption on reconfigurable hardware. In: Lange, T., Lauter, K., Lisoněk, P. (eds.) SAC 2013. LNCS, vol. 8282, pp. 68–85. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-43414-7_4
47. Roşca, M., Sakzad, A., Stehlé, D., Steinfeld, R.: Middle-product learning with errors. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017. LNCS, vol. 10403, pp. 283–297. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63697-9_10
48. The Sage Developers. SageMath, the Sage Mathematics Software System (Version 6.5) (2015). http://www.sagemath.org
49. Sakshaug, H.: Security analysis of the NTRUEncrypt public key encryption scheme (2007). https://brage.bibsys.no/xmlui/bitstream/handle/11250/258846/426901_FULLTEXT01.pdf
50. Schnorr, C.P.: Lattice reduction by random sampling and birthday methods. In: Alt, H., Habib, M. (eds.) STACS 2003. LNCS, vol. 2607, pp. 145–156. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-36494-3_14
51. Shoup, V.: A proposal for an ISO standard for public key encryption (2001). https://eprint.iacr.org/2001/112
52. Shoup, V.: OAEP reconsidered. J. Cryptology **15**(4), 223–249 (2002)
53. Stam, M.: A key encapsulation mechanism for NTRU. In: Smart, N.P. (ed.) Cryptography and Coding 2005. LNCS, vol. 3796, pp. 410–427. Springer, Heidelberg (2005). https://doi.org/10.1007/11586821_27
54. Stehlé, D., Steinfeld, R.: Making NTRU as secure as worst-case problems over ideal lattices. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 27–47. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-20465-4_4
55. van Oorschot, P.C., Wiener, M.J.: Parallel collision search with cryptanalytic applications. J. Cryptology **12**(1), 1–28 (1999)
56. van Vredendaal, C.: Reduced memory meet-in-the-middle attack against the NTRU private key. LMS J. Comp. Math. **19**, 43–57 (2016)
57. Wunderer, T.: Revisiting the hybrid attack: improved analysis and refined security estimates (2016). https://eprint.iacr.org/2016/733