NTRU Prime

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

[1] Department of Mathematics and Computer Science
Technische Universiteit Eindhoven, P.O. Box 513, 5600 MB Eindhoven, NL
c.chuengsatiansup@tue.nl, tanja@hyperelliptic.org, c.v.vredendaal@tue.nl

[2] Department of Computer Science
University of Illinois at Chicago, Chicago, IL 60607–7045, USA
djb@cr.yp.to

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 proposes NTRU Prime, which tweaks NTRU to use rings without these structures; proposes Streamlined NTRU Prime, which optimizes NTRU Prime from an implementation perspective; finds high-security post-quantum parameters for Streamlined NTRU Prime; and optimizes a constant-time implementation of those parameters. The performance results are surprisingly competitive with the best previous speeds for lattice-based cryptography.

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

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 defenses against potential attacks; see Appendix A. The reader can skip the appendix and simply remember the following facts:

This work was supported by the Netherlands Organisation for Scientic 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 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: 2016.05.11.

- 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 classic NTRU cryptosystem, and have none of our recommended defenses.
- 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, and have none of our recommended defenses.
- Fields of the form $(\mathbb{Z}/q)[x]/(x^p - x - 1)$, where p is prime, are used in "NTRU Prime", introduced in this paper, and have all of our recommended defenses.

Specifically, we use only about 50000 cycles on one core of an Intel Haswell CPU for **constant-time** multiplication in the field $(\mathbb{Z}/9829)[x]/(x^{739} - x - 1)$. We define a public-key cryptosystem "Streamlined NTRU Prime 9829^{739}" using this field. Our in-depth security analysis indicates that this cryptosystem provides more than 2^{128} **post-quantum** security, while eliminating the annoying possibility of "decryption failures" that appear in most lattice-based cryptosystems.

Multiplication is the main bottleneck in both encryption and decryption, so we easily outperform, e.g., pre-quantum Curve25519 as a public-key cryptosystem. We take advantage of Haswell's vectorized multiplier, but modern Curve25519 implementations such as [22] and [30] also do this, so we also expect to outperform Curve25519 on other platforms.

Our public keys are field elements, easily squeezed into 1232 bytes. We explain how to further squeeze ciphertexts (transporting 256-bit session keys) into just 1141 bytes. Obviously these sizes are not competitive with 256-bit ECC key sizes, but they are small enough for many applications: 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.

1.1. Comparison to previous multiplication speeds. The previous state of the art in implementations of lattice-based cryptography was last year's paper "Post-quantum key exchange: a new hope" [3] by Alkim, Ducas, Pöppelmann, and Schwabe. Most of the implementations before [3] 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 are also not aware of anything faster than [3] at a high security level. For example, [11] reports 134136 Haswell cycles for `ntruees787ep1` encryption; almost all of this time is for multiplication in $(\mathbb{Z}/2048)[x]/(x^{787} - 1)$ using variable-time sparse-polynomial-multiplication algorithms.

Like our paper, [3] targets the Haswell CPU, requires constant-time implementations, and aims for more than 2^{128} post-quantum security. Unlike our paper, [3] follows the classic NTRU/Ring-LWE tradition of using cyclotomic rings. More precisely, [3] uses the same type of ring $(\mathbb{Z}/q)[x]/(x^p + 1)$ as previous Ring-LWE papers, specifically with $p = 1024$ and $q = 12289 = 12 \cdot 1024 + 1$.

The conventional wisdom is that rings of this type 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 crit-

defenses	constant	cycles	ring	technique	source
yes	yes	50000	$(\mathbb{Z}/9829)[x]/(x^{739}-x-1)$	Toom etc.	this paper
no	yes	40000	$(\mathbb{Z}/12289)[x]/(x^{1024}+1)$	NTT	"new hope" [3]
no	no	>100000	$(\mathbb{Z}/2048)[x]/(x^{787}-1)$	sparse input	`ntruees787ep1` [29]

Fig. 1.1. Comparison of multiplication results. "Defenses" means that the ring has this paper's defenses against potential attacks. "Constant" means that the software runs in constant time. "Cycles" is approximate multiplication time on an Intel Haswell; see text for calculations. All rings are used in public-key cryptosystems aiming for $\geq 2^{128}$ post-quantum security.

ically 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 required by our conservative security recommendations, would make the NTTs several times more expensive. The best NTT-based method known to multiply in, e.g., $(\mathbb{Z}/8819)[x]/(x^{1021}-x-1)$ requires replacing $x^{1021}-x-1$ with $x^{2048}+1$, and also replacing 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, using an optimized 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. We do not claim that our 50000-cycle speed outperforms the multiplications in [3], but it is quite close: one multiplication in [3] takes about 40000 cycles. (Each forward NTT in [3] takes 10948 cycles; each reverse NTT takes 11896 cycles; the time for pointwise multiplication is not stated in [3] but can be extrapolated from [34] to take about 5000 cycles.) To summarize, our security recommendations do *not* create a large speed penalty.

1.2. Public-key encryption vs. unauthenticated key exchange. The cryptographic operations in [3] actually sound significantly less efficient: 126236 cycles for the client (sending 2048 bytes) and $107534+22104=129638$ cycles for the server (also sending 2048 bytes). The main reason for this gap is that we are targeting traditional public-key encryption while [3] is targeting unauthenticated key exchange, a larger operation.

This distinction reflects 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 can also be used for post-quantum server authentication, and for fast key erasure (often called "forward secrecy"), protecting against key theft. However, the details and costs are quite different.

total bytes	client bytes	server bytes	key erasure	source
1141	1141	0	no	this paper
3514	1141+1141	1232	yes	this paper
3514	1141+1232	1141	yes	this paper
4096+signature	2048	2048+signature	yes	[3]+signature

Fig. 1.2. 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.

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 [3]), 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. 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 of KEMs in Section 3.4. 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:

- The client cost for public-key encryption is less expensive than the client side of unauthenticated key exchange.
- The server cost for decryption is less expensive than the server side of unauthenticated key exchange.
- The signature is skipped entirely: the server avoids the cost of generating it, the client avoids the cost of verifying it, and both sides avoid the network traffic.

On the other hand, if key erasure is desired, then the second approach needs another layer. 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 (e.g., a key discarded 60 seconds after its first use and replaced by a new key). The client sends a ciphertext to each of these keys, and the two plaintexts are hashed to produce the shared secret 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 has to perform two public-key encryption operations, and the server has to perform two decryption operations. This is 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, assuming key-generation costs are amortized; one would expect an optimized unauthenticated key-exchange protocol to be somewhat faster than this.

More importantly, the server also has to send its short-term post-quantum public key to the client. This could be more expensive than handling the signature in the first approach, depending on which signature system is in use. Post-quantum signatures are not cheap, but [3] considers the current world of pre-quantum signatures. This 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; otherwise neither integrity nor confidentiality is protected.

This analysis gives us two basic reasons for focusing on the traditional data flow in public-key encryption. First, we see clear value in understanding the lowest possible costs for post-quantum public-key encryption. Users who can afford to deploy post-quantum encryption on top of their current pre-quantum systems should do so, even if they cannot afford the extra costs of post-quantum key erasure. Of course, it might still be possible for an attacker to use key theft to break the post-quantum system *and* a quantum computer to break the pre-quantum systems, so users who *can* afford post-quantum key erasure should deploy it.

Second, the combination of pre-quantum signatures and post-quantum unauthenticated key exchange in [3] might cost slightly less CPU time than using two layers of post-quantum public-key encryption without signatures, but it does not seem to be competitive in bandwidth. See Figure 1.2. Furthermore, users will have to move to *post-quantum* signatures before attackers have quantum computers, and all known post-quantum signature systems add considerable expense in CPU time or bandwidth or both. A pure post-quantum encryption solution, without signatures, avoids the need for this upgrade and appears to minimize long-term costs.

Acknowledgements We wish to thank John Schanck for detailed discussion of the security of NTRU and for suggesting the “transitional security” terminology.

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 caution potential users that many details of Streamlined NTRU Prime are new and 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

```
p = 739; q = 9829; t = 204
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 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 encodeZx(m): # assumes coefficients in range {-1,0,1,2}
  m = [m[i]+1 for i in range(p)] + [0]*(-p % 4)
  return ''.join([int2str(m[i]+m[i+1]*4+m[i+2]*16+m[i+3]*64,1) for i in range(0,len(m),4)])

def decodeZx(mstr):
  m = [str2int(mstr[i:i+1]) for i in range(len(mstr))]
  m = concat([[m[i]%4,(m[i]//4)%4,(m[i]//16)%4,m[i]//64] for i in range(len(m))])
  return Zx([m[i]-1 for i in range(p)])

def encodeRq(h):
  h = [lift(h[i]) for i in range(p)] + [0]*(-p % 3)
  h = ''.join([int2str(h[i]+h[i+1]*10240+h[i+2]*10240^2,5) for i in range(0,len(h),3)])
  return h[0:1232]

def decodeRq(hstr):
  h = [str2int(hstr[i:i+5]) for i in range(0,len(hstr),5)]
  h = concat([[h[i]%10240,(h[i]//10240)%10240,h[i]//10240^2] for i in range(len(h))])
  if max(h) >= q: raise Exception("pk out of range")
  return Rq(h)

def encoderoundedRq(c):
  c = [1638 + nicelift(c[i]/3) for i in range(p)] + [0]*(-p % 2)
  c = ''.join([int2str(c[i]+c[i+1]*4096,3) for i in range(0,len(c),2)])
  return c[0:1109]

def decoderoundedRq(cstr):
  c = [str2int(cstr[i:i+3]) for i in range(0,len(cstr),3)]
  c = concat([[c[i]%4096,c[i]//4096] for i in range(len(c))])
  if max(c) > 3276: raise Exception("c out of range")
  return 3*Rq([c[i]-1638 for i in range(p)])
```

Fig. 2.1. Complete non-constant-time reference implementation of Streamlined NTRU Prime 9829^{739} using the Sage computer-algebra system, part 1: auxiliary functions for encoding and decoding of polynomials as byte strings.

opportunity to rethink and reoptimize all of the details of NTRU from an implementation perspective.

2.1. Parameters. Streamlined NTRU Prime is actually a family of cryptosystems parametrized by positive integers (p, q, t) subject to the following restric-

```
def randomR(): # R element with 2t coeffs +-1
  L = [2*randrange(2^31) for i in range(2*t)]
  L += [4*randrange(2^30)+1 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([randrange(3)-1 for i in range(p)])
    if R3(g).is_unit(): break
  f = randomR()
  h = Rq(g)/(3*Rq(f))
  pk = encodeRq(h)
  return pk,encodeZx(f) + encodeZx(R(lift(1/R3(g)))) + pk

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

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[:185]),decodeZx(sk[185:370]),decodeRq(sk[370:])
  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. 2.2. Complete non-constant-time reference implementation of Streamlined NTRU Prime 9829^{739} using the Sage computer-algebra system, part 2: key generation, encapsulation, decapsulation, tests.

tions: p is a prime number; q is a prime number; $p \geq \max\{2t, 3\}$; $q \geq 48t + 1$; $x^p - x - 1$ 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.

Our case study in this paper is Streamlined NTRU Prime 9829^{739}. This specific cryptosystem has parameters $p = 739$, $q = 9829$, and $t = 204$. The following subsections specify the algorithms for general parameters but the reader may wish to focus on these particular parameters. Figures 2.1 and 2.2 show complete algorithms for key generation, encapsulation, and decapsulation in Streamlined NTRU Prime 9829^{739}, using the Sage [56] 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 2.2.

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, an element of $\mathbb{Z}/q$ for $q = 9829$ is traditionally encoded as 14 bits, but three such elements are easily encoded together as 40 bits, saving 5% of the space; 739 elements of $\mathbb{Z}/q$ would traditionally take 10346 bits, but 246 triples and a final element take just 9856 bits. See Figure 2.1 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 3.4 for comparison to older notions of public-key encryption.

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 $q = 9829$, 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 2.1.

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 2.1. 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 $q = 9829$ and $p = 739$, we use 12 bits for each coefficient of c and thus 8872 bits (padded to a byte boundary) for $\overline{c}$, saving 10% compared to the size of a public key and 15% compared to separately encoding each element of $\mathbb{Z}/q$. See `encoderoundedRq` in Figure 2.1. 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 2.1.

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 $[-24t, 24t]$ by Theorem 2.1 below, and thus in $[-(q-1)/2, (q-1)/2]$ since $q \geq 48t+1$. 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 $p \geq 2$ 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 $[-24t, 24t]$.*

Proof. Write fm as $a_0 + a_1x + \cdots + a_{2p-2}x^{2p-2}$. Each coefficient a_j has the form $\sum_i f_i m_{j-i}$; each term $f_i m_{j-i}$ here is in $[-1, 1]$, and there are at most $2t$ nonzero terms, so $a_j \in [-2t, 2t]$. Now $fm \bmod x^p - x - 1 = (a_0 + a_p) + (a_1 + a_p + a_{p+1})x +$

$(a_2 + a_{p+1} + a_{p+2})x^2 + \cdots + (a_{p-2} + a_{2p-3} + a_{2p-2})x^{p-2} + (a_{p-1} + a_{2p-2})x^{p-1}$, with each coefficient visibly in $[-6t, 6t]$. Similar reasoning for gr implies that each coefficient of $gr \bmod x^p - x - 1$ is in $[-6t, 6t]$. Hence each coefficient of $3fm + gr \bmod x^p - x - 1$ is in $[-24t, 24t]$. □

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.

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 (classic NTRU); $x^p + 1$ where p is a power of 2; and $x^p - x - 1$ for prime p (NTRU Prime). Choices of q include powers of 2 (classic NTRU); split primes q; 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 $\mathcal{R}/q$, secretly computed by dividing two small polynomials. It is invertible in $\mathcal{R}/q$ but has no other publicly visible structure.

An alternative is to transmit the public key h as two elements $d, hd \in \mathcal{R}/q$, where d is chosen as a uniform random invertible element of $\mathcal{R}/q$. 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 the receiver can skip all divisions in the secret computation of the public key h: the receiver simply 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 skip this alternative in Streamlined NTRU Prime. We also skip the idea of supporting both key formats as a run-time option: this would complicate implementations.

3.3. Inputs and ciphertexts. Classic NTRU 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. The multiplication of h by r is the main bottleneck in encryption and the main target of our implementation work; see Sections 6 and 7.

The receiver can quickly recover the input (m, r) from the ciphertext $m+hr$; see Section 2.4 for Streamlined NTRU Prime and Section 3.5 for a broader view. We say "input" rather than "plaintext" because in any modern public-key cryptosystem the input is randomized and is separated from the sender's plaintext by some hashing; see Section 3.4.

In the original NTRU specification [37], 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 [36] 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 classic NTRU 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 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 739 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.

The advantage of choosing m in this way is that $\approx q/3$ possibilities take less space than q possibilities. See Section 2.3. Analogous ciphertext compression

is standard in the context of code-based cryptography, and dates back to a 1986 paper [50] by Niederreiter, but the idea is surprisingly difficult to find in the literature on lattice-based cryptography. We have borrowed the "rounding" name from a few recent papers on "learning with rounding" (see [5], [4], and [15]), but we have not found any mention of the impact on ciphertext sizes.

3.4. Padding and KEMs. In Streamlined NTRU Prime we use the modern "KEM+DEM" approach introduced by Shoup; see [60]. 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., [61]) and in practice (see, e.g., [48]), 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 [60] (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 NTRU-based KEMs we have found in the literature (e.g., [64] and [57]) 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. Furthermore, it is 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 [25, Section 6] by Dent, backed by a tight security reduction [25, Theorem 8] from inversion to generic-hash chosen-ciphertext attacks:

- 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. This assumption is not valid for most lattice-based encryption schemes (see Section 3.7), but it is valid for Streamlined NTRU Prime. This difference appears to account for most of the complications in subsequent papers on NTRU-based KEMs.

As a spinoff of analyzing KEM options, we found a fast chosen-ciphertext attack against the code-based KEM proposed in [53]. The problem is that [53] 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 [12] on the plain McEliece system. McBits [10] avoids this problem, because it uses a separate output bit from the KEM to indicate a decoding failure.

3.5. Key generation and decryption. Classic NTRU computes the public key as $3g/f$ 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 3.4.

It is important for security to compute inverses such as $1/f$ in constant time. For Streamlined 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, and one can easily build constant-time exponentiations from our constant-time multiplications. We actually use a more complicated but much faster approach. We first convert the extended Euclidean algorithm into a one-coefficient-at-a-time Berlekamp–Massey/"almost-inverse" algorithm (see, e.g., [62]). All of the conditional branches amount to simple input selections, which we convert into constant-time arithmetic. We compute the maximum number of iterations for the algorithm, and always perform this number of iterations, again using constant-time arithmetic so that dummy iterations produce the correct result.

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 [37,42,35,36] 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 3.5) 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 3.7), 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 we do not further limit the distribution of coefficients. Our security analysis conservatively underestimates the size of the key space, ignoring these changes.

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

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 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 near 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 739 random 32-bit numbers, and a constant-time size-1024 sorting network (Batcher's "odd-even sorting network" [6]) takes just 24064 constant-time compare-exchange steps, which we note are easily vectorizable.

NTRU papers starting with [38] 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.)

Elsewhere in the literature on lattice-based cryptography one can find larger coefficients: consider, e.g., the quinary polynomials in [26], and the even wider range in [3]. In [65], 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 3.7), producing a much larger key size (and ciphertext size) for the same security level.

3.7. Choosing q. In Streamlined NTRU Prime we require $q \geq 48t + 1$. Recall that decryption sees $3fm + gr$ in $\mathcal{R}/q$ and tries to deduce $3fm + gr$ in $\mathcal{R}$; the condition $q \geq 48t + 1$ 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 $48t + 1$; for example, replacing g by $1 + 3G$ would change $48t + 1$ into $72t + 3$.

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. But this approach raises several questions:

- Will users randomly encounter decryption failures?
- Can attackers trigger decryption failures by generating many more ciphertexts?
- Can attackers tweak these ciphertexts to trigger decryption failures?
- What should implementors do if decryption *does* fail?

The literature on the first two questions is already quite complicated, and it is difficult to find literature on the third and fourth questions. The only safe assumption is that decryption failures compromise security, allowing attackers to learn f from the pattern of decryption failures; see [40], and see also the discussion of NTRU complications in Section 3.4. We prefer to guarantee that decryption works, making the security analysis simpler and more robust.

4 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.

4.1. Meet-in-the-middle attack. Odlyzko's meet-in-the-middle attack [41,39] 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 of f is $\binom{p}{t}\binom{p-t}{t}$, so a first estimate is that the number of loops in the algorithm is $\sqrt{\binom{p}{t}\binom{p-t}{t}}$. However, in classic NTRU, a key (f,g) is equivalent to all of the rotated keys (x^if,x^ig), and the algorithm succeeds if it finds any of these rotated keys. The p rotations are almost always distinct, producing a speedup factor very close to $\sqrt{p}$.

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. 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 classic NTRU; i.e., the rotation speedup is only around $\sqrt{p/t}$ rather than $\sqrt{p}$.

On the other hand, some keys f allow considerably more rotations. We have decided to conservatively assume a speedup factor of $\sqrt{p-t}$, even though we do not know how to achieve this speedup for random keys f. This means that the number of loops before this attack is expected to find f is

$$L = \frac{\sqrt{\binom{p}{t}\binom{p-t}{t}}}{\sqrt{p-t}}. \tag{1}$$

In each loop t vectors of size p are added and their coefficients are reduced modulo q. We thus estimate the attack cost as $L \log_2(pt)$. The storage requirement of the attack is $L \log_2 \binom{p}{t}$.

We can reduce this storage by applying collision search to the meet-in-the-middle attack (see [51,66]). In this case we can reduce the storage capacity by a factor w at the expense of increasing the running time by a factor $\sqrt{w}$.

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 approximately $(2\lfloor p/3 \rfloor + 2t)^{1/2}$. According to the Gaussian heuristic, which states the length of the shortest vector can be approximated by $\det(B)^{1/(2p)}\sqrt{\pi e p} = \sqrt{\pi e p q}$, we expect this vector 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 methods to solve SVP in the NTRU public lattice are the hybrid attack and sieving algorithms, which we will discuss in the next sections.

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 classic NTRU's 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 [39].

The attack works in two phases: the lattice basis reduction phase and the meet-in-the-middle phase.

In the lattice reduction step it is observed that applying lattice reduction techniques will mostly reduce the middle vectors of the basis [58]. Therefore the strategy is to apply lattice-basis reduction, for example BKZ 2.0 [20], 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 [39].

To estimate the security against this attack we follow the approach of [36]. 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

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

work, where the $\log_2(p-t)$ term accounts for the correct forms and

$$v(a,b) = \frac{\binom{p-w'}{t-a}\binom{p-w'-t+a}{t-b}}{\binom{p}{t}\binom{p-t}{t}}.$$

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

$$\log_2(\delta) \le \frac{(p-w)\log_2(q)}{(2p-(w+w'))^2} - \frac{1}{2p-(w'+w)}. \tag{3}$$

We use the BKZ 2.0 simulator of [20] 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 [36] from the experiments of [21]:

$$\text{Estimate}(\beta, p, n) = 0.000784314\beta^2 + 0.366078\beta - 6.125 + \log_2(p \cdot n) + 7. \quad (4)$$

Using these estimates we can determine the optimal w and w' to attack a parameter set and thereby estimate its security.

4.4. Sieving algorithms. For very large dimensions, the performance of enumeration algorithms [54,31,43] is slightly super-exponential and is known to be suboptimal. The provable sieving algorithms of Pujol and Stehlé [55] solve SVP in time $2^{2.465p+o(p)}$ and space $2^{1.233p+o(p)}$, and more recent SVP algorithms [1] take time $2^{p+o(p)}$. More importantly, under heuristic assumptions, sieving is much faster. The most recent work on lattice sieving (see [7,45]) has pushed the heuristic complexity down to $2^{0.292p+o(p)}$. A closer look at polynomial factors indicates that the $o(p)$ here is positive, and the analysis of [49] suggests that these algorithms are not helpful for cryptographic sizes, but we nevertheless use $2^{0.292p}$ to estimate the speed of sieving attacks.

5 Parameters

Algorithm 1 determines all parameter sets (p, q, t, λ), where the *pre-quantum* security level λ is at least ℓ according to the attack analysis of Section 4. For example, we estimate pre-quantum security 2^{215} for our recommended parameters $(p, q, t) = (739, 9829, 204)$. We expect *post-quantum* security levels to be somewhat lower (e.g., [46] saves a factor 1.1 in the best known asymptotic SVP exponents) 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 $GF(q)$. 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 choose $w = \lambda_3$ (initially $\lambda_3 = 0$) and do a binary search for w' such that the two phases of the hybrid attack are of equal cost. For each K we determine the Hermite factor required with Equation (3), use the BKZ-2.0 simulator to determine the optimal β and n to acquire the Hermite factor and use Equations (4) 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. Experiments show that decreasing t to t_1 linearly decreases the security level by approximately $t - t_1$.

The results of the algorithm for $q_b = 10000$, $[p_1, p_2] = [672, 1024]$ and $\ell = 112$ can be found in Appendix B.

6 Polynomial multiplication

The main bottleneck operation in both encryption and decryption is polynomial multiplication. It is well known that schoolbook multiplication is asymptotically

Algorithm 1: Algorithm to determine parameter sets for security level above ℓ.

Data: Upper bound q_b for q, range $[p_1, p_2]$ for p and lower bound ℓ for the security level
Result: Viable parameters p, q and t with security level λ.
begin
  $p \leftarrow p_1 - 1$ (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**
      $t \leftarrow \min\left(\lfloor (q-1)/48 \rfloor, \lfloor p/3 \rfloor\right)$
      $\lambda_1 \leftarrow \text{mitmcosts}(p, t)$
      $\lambda_2 \leftarrow 0.292p$
      if $\min(\lambda_1, \lambda_2) \geq \ell$ **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.
        $\lambda_3 \leftarrow \max(\text{hybridbkzcost}, \text{hybridmitmcost})$
        return $p, q, t, \min(\lambda_1, \lambda_2, \lambda_3)$

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

We 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. We considered sizes up to $1024n \times 1024n$ (where n reflects number of bits, limbs or terms). We analyzed the resulting ranges of double-precision floating-point numbers (53-bit mantissa) for various input sizes, making sure to avoid overflow.

After comparing the results of this analysis to the parameter possibilities in Appendix B, we decided to focus on $768n \times 768n$. This section explains how we decompose $768n$ into $128n$ using Toom6, then decompose $128n$ into $4n$ using five levels of refined Karatsuba, then use schoolbook multiplication for $4n \times 4n$. All 6 pieces of the Toom6 are of the same size, each half of refined Karatsuba at each level is of the same size, and everything is 4-way vectorizable; we exploit this in Section 7.

6.1. Top level. At the top level we use Toom6 to decompose $768n$ into 6 pieces of $128n$. For instance, let one of the $768n$ polynomials be $a(x) = a_0 + a_1x + a_2x^2 + \cdots + a_{767}x^{767}$. It is then decomposed into

$$a(x, y) = A_0(x) + A_1(x)y + A_2(x)y^2 + A_3(x)y^3 + A_4(x)y^4 + A_5(x)y^5,$$

where $y = x^{128}$ and

$$\begin{aligned}
A_0(x) &= a_0 \quad + a_1 \quad x + a_2 \quad x^2 + \cdots + a_{127}x^{127};\\
A_1(x) &= a_{128} + a_{129}x + a_{130}x^2 + \cdots + a_{255}x^{127};\\
A_2(x) &= a_{256} + a_{257}x + a_{258}x^2 + \cdots + a_{383}x^{127};\\
A_3(x) &= a_{384} + a_{257}x + a_{258}x^2 + \cdots + a_{511}x^{127};\\
A_4(x) &= a_{512} + a_{257}x + a_{258}x^2 + \cdots + a_{639}x^{127};\\
A_5(x) &= a_{640} + a_{257}x + a_{258}x^2 + \cdots + a_{767}x^{127}.
\end{aligned}$$

Let another polynomial be $b(x) = b_0 + b_1x + b_2x^2 + \cdots + b_{767}x^{767}$. We can decompose this in a similar way as a, such that the multiplication of a and b becomes

$$\begin{aligned}
ab = C_0 + C_1x^{128} + C_2x^{256} + C_3x^{384} + C_4x^{512} + C_5x^{640}\\
+ C_6x^{768} + C_7x^{896} + C_8x^{1024} + C_9x^{1152} + C_{10}x^{1280},
\end{aligned}$$

where

$$\begin{aligned}
C_0 &= A_0B_0;\\
C_1 &= A_0B_1 + A_1B_0;\\
C_2 &= A_0B_2 + A_1B_1 + A_2B_0;\\
C_3 &= A_0B_3 + A_1B_2 + A_2B_1 + A_3B_0;\\
C_4 &= A_0B_4 + A_1B_3 + A_2B_2 + A_3B_1 + A_4B_0;\\
C_5 &= A_0B_5 + A_1B_4 + A_2B_3 + A_3B_2 + A_4B_1 + A_5B_0;\\
C_6 &= A_1B_5 + A_2B_4 + A_3B_3 + A_4B_2 + A_5B_1;\\
C_7 &= A_2B_5 + A_3B_4 + A_4B_3 + A_5B_2;\\
C_8 &= A_3B_5 + A_4B_4 + A_5B_3;\\
C_9 &= A_4B_5 + A_5B_4;\\
C_{10} &= A_5B_5.
\end{aligned}$$

Note that we leave out x but A_i and B_i are polynomials in x.

There are 11 values C_i, therefore Toom6 requires 11 multiplications (see below) in order to interpolate those C_i. We chose to evaluate y at 0, ±1, ±2, ±3, ±4, 5 and ∞. One of the advantages of using $+$ and $-$ is that some intermediate results can be reused to save some computations. For example, to compute $s_1 = a(x,1) \cdot b(x,1)$ and $s_{-1} = a(x,-1) \cdot b(x,-1)$, we first compute $t_0 = A_0(x) + A_2(x) + A_4(x)$ and $t_1 = A_1(x) + A_3(x) + A_5(x)$, then compute $t_0 + t_1$ and $t_0 - t_1$ to obtain s_1 and s_{-1} respectively. The 11 multiplications of $128n \times 128n$

that we need to compute are

$$\begin{aligned}
0 &: A_0 \cdot B_0;\\
1 &: (A_0+A_1+A_2+A_3+A_4+A_5)\cdot(B_0+B_1+B_2+B_3+B_4+B_5);\\
-1 &: (A_0-A_1+A_2-A_3+A_4-A_5)\cdot(B_0-B_1+B_2-B_3+B_4-B_5);\\
2 &: (A_0+2A_1+2^2A_2+2^2A_3+2^4A_4+2^5A_5)\cdot(B_0+2B_1+2^2B_2+2^2B_3+2^4B_4+2^5B_5);\\
-2 &: (A_0-2A_1+2^2A_2-2^2A_3+2^4A_4-2^5A_5)\cdot(B_0-2B_1+2^2B_2-2^2B_3+2^4B_4-2^5B_5);\\
3 &: (A_0+3A_1+3^3A_2+3^3A_3+3^4A_4+3^5A_5)\cdot(B_0+3B_1+3^3B_2+3^3B_3+3^4B_4+3^5B_5);\\
-3 &: (A_0-3A_1+3^3A_2-3^3A_3+3^4A_4-3^5A_5)\cdot(B_0-3B_1+3^3B_2-3^3B_3+3^4B_4-3^5B_5);\\
4 &: (A_0+4A_1+4^4A_2+4^4A_3+4^4A_4+4^5A_5)\cdot(B_0+4B_1+4^4B_2+4^4B_3+4^4B_4+4^5B_5);\\
-4 &: (A_0-4A_1+4^4A_2-4^4A_3+4^4A_4-4^5A_5)\cdot(B_0-4B_1+4^4B_2-4^4B_3+4^4B_4-4^5B_5);\\
5 &: (A_0+5A_1+5^5A_2+5^5A_3+5^5A_4+5^5A_5)\cdot(B_0+5B_1+5^5B_2+5^5B_3+5^5B_4+5^5B_5);\\
\infty &: A_5 \cdot B_5 .
\end{aligned}$$

6.2. Middle level. The middle-level $128n \times 128n$ of 11 multiplications inside Toom6 are computed using 5-level refined Karatsuba. Recall the "refined Karatsuba identity" from [8, Section 2]:

$$(F_0 + t^n F_1)(G_0 + t^n G_1) = (1 - t^n)(F_0G_0 - t^n F_1G_1) + t^n(F_0 + F_1)(G_0 + G_1).$$

Level 1. For the first level of Karatsuba, we split one $128n$ of f (and one $128n$ of g) into two $64n$'s, namely, F_0 and F_1, with $F = F_0 + x^{64}F_1$ (and into two $64n$'s, namely, G_0 and G_1, with $G = G_0 + x^{64}G_1$) as

$$\begin{aligned}
F_0 &= f_0+f_1x+f_2x^2+\cdots+f_{63}x^{63}; \quad F_1 = f_{64}+f_{65}x+f_{66}x^2+\cdots+f_{127}x^{63};\\
G_0 &= g_0+g_1x+g_2x^2+\cdots+g_{63}x^{63}; \quad G_1 = g_{64}+g_{65}x+g_{66}x^2+\cdots+g_{127}x^{63}.
\end{aligned}$$

Then, we have

$$fg = (1 - x^{64})(F_0G_0 - x^{64}F_1G_1) + x^{64}(F_0 + F_1)(G_0 + G_1).$$

Level 2. For the second level, we further split one $64n$ of F_0 (and those of F_1) into two $32n$'s, namely, F_{00} and F_{01}, with $F_0 = F_{00} + x^{32}F_{01}$ (and into F_{10} and F_{11} with $F_1 = F_{10} + x^{32}F_{11}$) as

$$\begin{aligned}
F_{00} &= f_0+f_1x+f_2x^2+\cdots+f_{31}x^{31}; \quad F_{01} = f_{32}+f_{33}x+f_{34}x^2+\cdots+f_{63}x^{31};\\
F_{10} &= f_{64}+f_{65}x+f_{66}x^2+\cdots+f_{95}x^{31}; \quad F_{11} = f_{96}+f_{97}x+f_{98}x^2+\cdots+f_{127}x^{31}.
\end{aligned}$$

Let $F_2 = F_0+F_1$, then F_2 is further split into F_{20} and F_{21} with $F_2 = F_{20}+x^{32}f_{21}$ as

$$\begin{aligned}
F_{20} &= (f_0+f_{64})+(f_1+f_{65})x+(f_2+f_{66})x^2+\cdots+(f_{31}+f_{95})x^{31};\\
F_{21} &= (f_{32}+f_{96})+(f_{33}+f_{97})x+(f_{34}+f_{98})x^2+\cdots+(f_{63}+f_{127})x^{31}.
\end{aligned}$$

Similarly, we split G_0, G_1 and $G_2 = G_0 + G_1$ to obtain G_{00}, G_{01}, G_{10}, G_{11}, G_{20} and G_{21}. Then, we have

$$\begin{aligned}
F_0G_0 &= (1-x^{32})(F_{00}G_{00} - x^{32}F_{01}G_{01}) + x^{32}(F_{00}+F_{01})(G_{00}+G_{01});\\
F_1G_1 &= (1-x^{32})(F_{10}G_{10} - x^{32}F_{11}G_{11}) + x^{32}(F_{10}+F_{11})(G_{10}+G_{11});\\
F_2G_2 &= (1-x^{32})(F_{20}G_{20} - x^{32}F_{21}G_{21}) + x^{32}(F_{20}+F_{21})(G_{20}+G_{21}).
\end{aligned}$$

Level 3. For the third level, we further split one $32n$ into two $16n$'s, for instance, F_{00} is split into F_{000} and F_{001} with $F_{00} = F_{000} + x^{16}F_{001}$ as

$$F_{000}{=}f_0{+}f_1x{+}f_2x^2{+}\cdots{+}f_{15}x^{15};\quad F_{001}{=}f_{16}{+}f_{17}x{+}f_{18}x^2{+}\cdots{+}f_{31}x^{15}.$$

Similar splits also apply to F_{01}, F_{10}, F_{11}, F_{20} and F_{21}.

Let $F_{02} = F_{00} + F_{01}$, then F_{02} is further split into F_{020} and F_{021} with $F_{02} = F_{020} + x^{16}F_{021}$ as

$$\begin{aligned}
F_{020} &= (f_0 + f_{32}) + (f_1 + f_{33})x + (f_2 + f_{34})x^2 + \cdots + (f_{15} + f_{47})x^{15};\\
F_{021} &= (f_{16} + f_{48}) + (f_{17} + f_{49})x + (f_{18} + f_{50})x^2 + \cdots + (f_{31} + f_{63})x^{15}.
\end{aligned}$$

Similar splits also apply to $F_{12} = F_{10} + F_{11}$ and $F_{22} = F_{20} + F_{21}$.

We do the same for G_{ij} where $0 \le i, j \le 2$. Then, we have

$$F_{ij}G_{ij} = (1-x^{16})(F_{ij0}G_{ij0} - x^{16}F_{ij1}G_{ij1}) + x^{16}(F_{ij0}+F_{ij1})(G_{ij0}+G_{ij1});$$

where $0 \le i, j \le 2$.

Level 4. For the fourth level, we further split one $16n$ into two $8n$'s, for instance, F_{000} is split into F_{0000} and F_{0001} with $F_{000} = F_{0000} + x^8F_{0001}$ as

$$F_{0000}{=}f_0{+}f_1x{+}f_2x^2{+}\cdots{+}f_7x^7;\quad F_{0001}{=}f_8{+}f_9x{+}f_{10}x^2{+}\cdots{+}f_{15}x^7.$$

Similar splits also apply to F_{ijk} and G_{ijk} where $0 \le i, j, k \le 2$. Then we have

$$F_{ijk}G_{ijk}{=}(1{-}x^8)(F_{ijk0}G_{ijk0}{-}x^8F_{ijk1}G_{ijk1}){+}x^8(F_{ijk0}{+}F_{ijk1})(G_{ijk0}{+}G_{ijk1});$$

where $0 \le i, j, k \le 2$.

Level 5. Finally, for the fifth level, we further split one $8n$ into two $4n$'s, for instance, F_{0000} is split into F_{00000} and F_{00001} with $F_{0000} = F_{00000} + x^4F_{00001}$ as

$$F_{00000} = f_0 + f_1x + f_2x^2 + f_3x^3;\quad F_{00001} = f_4 + f_5x + f_6x^2 + f_7x^3.$$

Similar splits also apply to F_{ijkl} and G_{ijkl} where $0 \le i, j, k, l \le 2$. Then we have

$$\begin{aligned}
F_{ijkl}G_{ijkl} = (1-x^8)&(F_{ijkl0}G_{ijkl0} - x^8F_{ijkl1}G_{ijkl1})\\
&+ x^8(F_{ijkl0} + F_{ijkl1})(G_{ijkl0} + G_{ijkl1});
\end{aligned}$$

where $0 \le i, j, k, l \le 2$.

6.3. Lowest level. The lowest-level multiplication of $4n \times 4n$ is computed using schoolbook multiplication. For instance, $F_{00000}G_{00000}$ is computed as follows

$$\begin{aligned}
h_0 &= f_0g_0; & h_4 &= f_1g_3 + f_2g_2 + f_3g_1;\\
h_1 &= f_0g_1 + f_1g_0; & h_5 &= f_2g_3 + f_3g_2;\\
h_2 &= f_0g_2 + f_1g_1 + f_2g_0; & h_6 &= f_3g_3.\\
h_3 &= f_0g_3 + f_1g_2 + f_2g_1 + f_3g_0;
\end{aligned}$$

Since we use 5-level Karatsuba, we need to perform 3^5 $4n \times 4n$ multiplications to do one 128×128 computation.

7 Vectorization

Each Haswell core has two 256-bit floating-point vector multiplication units. These are 4-way vectorized multiplications which are also integrated with additions. This means that in one cycle, Haswell can compute 8 independent multiply-accumulate instructions $ab + c$ for 64-bit double-precision inputs a, b, c.

Most of the computations in Section 6 are obviously suitable for vectorization. We vectorize the computations of Toom6 for decomposing a and b in the top level (before the 11 multiplications) by taking four consecutive polynomial coefficients for each vector. For example, to evaluate at $y = 1$, we have to compute $A_0 + A_1$ as part of $A_0 + A_1 + A_2 + A_3 + A_4 + A_5$. We take a_0, a_1, a_2, a_3 as one vector and $a_{128}, a_{129}, a_{130}, a_{131}$ as another vector, then add them together. We then move to the next four coefficients of A_0 and A_1.

We also vectorize computations of refined Karatsuba for splitting inputs in the middle level (before the schoolbook multiplication), similarly to the Toom6 decomposition. For example, to compute $F_{20} = F_0 + F_1 = (f_0 + f_{64}) + (f_1 + f_{65})x + (f_2 + f_{66})x^2 + \cdots + (f_{31} + f_{95})x^{31}$, we take f_0, f_1, f_2, f_3 as inputs for one vector and $f_{64}, f_{65}, f_{66}, f_{67}$ for another vector, then add them together.

The $4n \times 4n$ schoolbook multiplications in the lowest level violate this vector structure. Each schoolbook multiplication uses 16 multiplications, including 9 multiply-accumulate instructions, with extensive communication across input lanes. Instead of trying to vectorize *inside* a schoolbook multiplication, we transpose inputs and vectorize *across* independent schoolbook multiplications. Inside a $128n \times 128n$ multiplication there are many (specifically, $3^5 = 243$) of these independent multiplications. For example, we vectorize 16 multiplications of $F_{00000}G_{00000}$, $F_{00001}G_{00001}$, $F_{00010}G_{00010}$ and $F_{00011}G_{00011}$ together.

We transpose the results of schoolbook multiplication back to the original format. We vectorize the merging of results in refined Karatsuba and the interpolation of C_i in Toom6 in the same way as splitting refined Karatsuba and decomposing Toom6.

We benchmarked our software on an Intel Haswell CPU, being careful to disable Turbo Boost. Each multiplication in $(\mathbb{Z}/9829)[x]/(x^{739} - x - 1)$ takes just 51488 cycles.

References

1. Divesh Aggarwal, Daniel Dadush, Oded Regev, and Noah Stephens-Davidowitz. Solving the shortest vector problem in 2^n time via discrete Gaussian sampling. *CoRR*, abs/1412.7994, 2014.
2. Martin Albrecht, Shi Bai, and Léo Ducas. A subfield lattice attack on overstretched NTRU assumptions: Cryptanalysis of some FHE and graded encoding schemes. *IACR Cryptology ePrint Archive*, 2016. https://eprint.iacr.org/2016/127.
3. Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. Post-quantum key exchange – a new hope. *IACR Cryptology ePrint Archive*, 2015. https://eprint.iacr.org/2015/1092.
4. Joël Alwen, Stephan Krenn, Krzysztof Pietrzak, and Daniel Wichs. Learning with rounding, revisited – new reduction, properties and applications. In Canetti and Garay [17], pages 57–74.
5. 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.
6. 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.
7. Anja Becker, Léo Ducas, Nicolas Gama, and Thijs Laarhoven. New directions in nearest neighbor searching with applications to lattice sieving. In Krauthgamer [44], pages 10–24.
8. 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.
9. Daniel J. Bernstein. A subfield-logarithm attack against ideal lattices, 2014. https://blog.cr.yp.to/20140213-ideal.html.
10. 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.
11. Daniel J. Bernstein and Tanja Lange. eBACS: ECRYPT benchmarking of cryptographic systems. https://bench.cr.yp.to (accessed 4 March 2016).
12. 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.
13. 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.

14. 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 [44], pages 893–902.
15. 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.
16. 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.
17. 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.
18. 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.
19. 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.
20. 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.
21. 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.
22. 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.
23. Henri Cohen. *A course in computational algebraic number theory*, volume 138 of *Graduate Texts in Mathematics*. Springer-Verlag, Berlin, 1993.
24. Ronald Cramer, Léo Ducas, Chris Peikert, and Oded Regev. Recovering short generators of principal ideals in cyclotomic rings. *IACR Cryptology ePrint Archive*, 2015. https://eprint.iacr.org/2015/313.
25. 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.
26. Léo Ducas, Alain Durmus, Tancrède Lepoint, and Vadim Lyubashevsky. Lattice signatures and bimodal Gaussians. In Canetti and Garay [17], pages 40–56.
27. 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.
28. Yara Elias, Kristin E. Lauter, Ekin Ozman, and Katherine E. Stange. Provably weak instances of Ring-LWE. In Gennaro and Robshaw [33], pages 63–92.

29. Mark Etzel. `ntruees787ep1` software, 2010. Included in [11].
30. 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.
31. 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.
32. 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.
33. 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.
34. Tim Güneysu, Tobias Oder, Thomas Pöppelmann, and Peter Schwabe. Software speed records for lattice-based signatures. In Gaborit [32], pages 67–82.
35. 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.
36. 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.
37. 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.
38. Jeffrey Hoffstein and Joseph H. Silverman. Random small Hamming weight products with applications to cryptography. *Discrete Applied Mathematics*, 130(1):37–49, 2003.
39. 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.
40. 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.
41. Nick Howgrave-Graham, Joseph H Silverman, and William Whyte. A meet-in-the-middle attack on an NTRU private key. Technical report, Technical report, NTRU

Cryptosystems, June 2003. Report, 2003. https://www.securityinnovation.com/uploads/Crypto/NTRUTech004v2.pdf.
42. Nick Howgrave-Graham, Joseph H. Silverman, and William Whyte. Choosing parameter sets for NTRUEncrypt with NAEP and SVES-3. *IACR Cryptology ePrint Archive*, 2005. https://eprint.iacr.org/2005/045.
43. 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.
44. 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.
45. Thijs Laarhoven. Sieving for shortest vectors in lattices using angular locality-sensitive hashing. In Gennaro and Robshaw [33], pages 3–22.
46. 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.
47. Adeline Langlois and Damien Stehlé. Hardness of decision (R)LWE for any modulus. Cryptology ePrint Archive, Report 2012/091, https://eprint.iacr.org/2012/091.
48. 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.
49. 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.
50. Harald Niederreiter. Knapsack-type cryptosystems and algebraic coding theory. *Problems of Control and Information Theory*, 15:159–166, 1986.
51. Paul C. van Oorschot and Michael J. Wiener. Parallel collision search with cryptanalytic applications. *J. Cryptology*, 12(1):1–28, 1999.
52. Hiroyuki Osada. The Galois groups of the polynomials $X^n + aX^l + b$. *J. Number Theory*, 25(2):230–238, 1987.
53. Edoardo Persichetti. Secure and anonymous hybrid encryption from coding theory. In Gaborit [32], pages 174–187.
54. 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.
55. 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.
56. The Sage Developers. *SageMath, the Sage Mathematics Software System (Version 6.5)*, 2015. http://www.sagemath.org.
57. Halvor Sakshaug. Security analysis of the NTRUEncrypt public key encryption scheme, 2007. brage.bibsys.no/xmlui/bitstream/handle/11250/258846/426901_FULLTEXT01.pdf.
58. 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.

59. Ernst S. Selmer. On the irreducibility of certain trinomials. *Math. Scand.*, 4:287–302, 1956.
60. Victor Shoup. A proposal for an ISO standard for public key encryption. *IACR Cryptology ePrint Archive*, 2001. https://eprint.iacr.org/2001/112.
61. Victor Shoup. OAEP reconsidered. *J. Cryptology*, 15(4):223–249, 2002.
62. Joseph H. Silverman. Almost inverses and fast NTRU key creation, 1999. https://www.securityinnovation.com/uploads/Crypto/NTRUTech014.pdf.
63. 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.
64. 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.
65. 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.
66. Christine van Vredendaal. Reduced memory meet-in-the-middle attack against the NTRU private key, 2016. Cryptology ePrint Archive, Report 2016/177, https://eprint.iacr.org/2016/177.

A Avoiding rings with worrisome structure

The October 2014 Campbell–Groves–Shepherd preprint "SOLILOQUY: a cautionary tale" [16] sent shock waves through lattice-based cryptography. What the preprint claimed, in a nutshell, was a polynomial-time quantum attack against a lattice-based cryptosystem. This claim led to some disputes, but after various followup papers it is now generally agreed that there is a polynomial-time quantum attack (and a subexponential-time pre-quantum attack) against, e.g., the Smart–Vercauteren system [63] from PKC 2010.

This appendix explains how the attack strategy exploits special features of the usual choices of rings inside ideal-lattice-based cryptography, and how modifying the choices of rings blocks core elements of the attack strategy. The first author had already publicly recommended these modifications in February 2014, to defend against a different attack strategy whose performance is the topic of ongoing research. See [9].

Switching from the Smart–Vercauteren system to NTRU (or to a typical Ring-LWE-based system) also stops the attack of [16]. One might speculate that this is an adequate defense against the attack. However, a recent paper by Albrecht, Bai, and Ducas [2] introduces another attack strategy that breaks some "overstretched NTRU assumptions" and that is *also* blocked by our modifications.

We emphasize that normal NTRU parameters are not affected by any of the attacks discussed in this appendix. However, we are skeptical of the notion that the most recent papers are the end of the attack story. We believe that it is prudent to apply our modifications to NTRU, and to Ring-LWE-based cryptosystems. We suggest the name "NTRU Prime" for the resulting cryptosystems; the "NTRU" part of the name acknowledges that we are tweaking the classic NTRU cryptosystem, and "Prime" reflects the fact that our modifications eliminate several different types of factorizations.

A.1. The Campbell–Groves–Shepherd attack. The preprint [16] actually describes a cryptosystem named "SOLILOQUY" that the authors say they privately developed in 2007, and then attacks the system. However, as mentioned briefly in [16], the key-recovery problem for this system is identical to the key-recovery problem for the Smart–Vercauteren system.

In these systems, everyone shares a standard monic irreducible polynomial $P \in \mathbb{Z}[x]$ with small coefficients. [63, Section 7] takes a power-of-2 cyclotomic polynomial, such as the polynomial $x^{1024}+1$; [16] 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 do not seem to have been defended by the authors of [16]) but subsequently presented a different polynomial-time quantum algorithm for this stage; see [13] and, for yet another approach, [14]. 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 closest-vector problem in what is called the "log-unit lattice". One normally expects CVPs to take exponential time, but *for cyclotomic polynomials* P one can efficiently write down a very short basis for the log-unit lattice (or at worst a small-index sublattice). This basis consists of logarithms of various "cyclotomic units", as explained very briefly in [16] and in much more detail in the followup paper [24] 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.

A.2. Recommendations. 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 [59] and [52]. Furthermore, in the context of 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" q; see Appendix B 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 recommendations 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 recommendations 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.3. How the recommendations stop attacks. The recent attack of [2] relies on having a large proper subfield of the field $\mathbb{Q}[x]/P$: a subfield of degree much larger than 1 but smaller than the degree of P. The degree of the field is a multiple of the degree of every subfield, so by taking a prime degree we obviously rule out this attack: the only subfields of $\mathbb{Q}[x]/P$ are $\mathbb{Q}$ and the entire field $\mathbb{Q}[x]/P$.

Our recommendations might seem to be overkill from this perspective. [2] recommends eliminating subfields (with credit to us), but it does not join our recommendation to require very large Galois groups.

To understand why we also require very large Galois groups, consider the suggestion from [2] 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 the attack of [2]. It does not, however, stop the attack of [16]: 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 [16] 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" [23, 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. To summarize: despite intensive research, all known CVP attacks are very difficult.

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 [27], [28], and [19] start by applying such homomorphisms; the attacks are restricted in other ways, but we see no reason to provide homomorphisms to the attacker in the first place. It is sometimes claimed that "modulus switching" makes the choice of q irrelevant (for example, [47] says "we prove that the arithmetic form of the modulus q is irrelevant to the computational hardness of LWE and RLWE"), but an attacker switching from q to another modulus will noticeably increase noise, interfering with typical attack algorithms.

A.4. Worst-case-to-average-case reduction. We briefly consider an argument against our recommendations. The argument says that using cyclotomic fields with split modulus (i.e., with P splitting into linear factors in $(\mathbb{Z}/q)[x]$) is desirable for security because it allows any attack algorithm against a particular type of cryptosystem to be converted into an algorithm to solve a "hard" cyclotomic-ideal-lattice problem in the worst case. The conversion has several steps, first producing an algorithm to solve Decision-Ring-LWE, then producing

an algorithm to solve Search-Ring-LWE, then producing an algorithm to solve the "hard" cyclotomic-ideal-lattice problem.

We have several counterarguments. First, there is actually very little evidence of serious study of the allegedly "hard" cyclotomic-ideal-lattice problem. It is entirely possible that the problem is breakable while our recommendations are secure. The problem is considerably more complicated, and less attractive to cryptanalysts, than truly well-known problems such as SVP.

Second, 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. See Section 4.4. This does not mean that there is any loss of security in, e.g., NTRU, but it calls into question the notion that lattice problems have been thoroughly studied.

Third, the conversion seems to be very far from tight. Even if one assumes that there are no better attacks against the "hard" cyclotomic-ideal-lattice problem, 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 very recently posted [18, Section 6] by Chatterjee, Koblitz, Menezes, and Sarkar. The independent work actually focuses on the simpler but similar setting of LWE, rather than Ring-LWE; the conclusions are similar to ours.

B Parameters

Parameters				
p	q	t	λ	key size
439	6833	142	112	5592
457	6037	125	118	5740
461	7607	153	118	5944
	8779	153	116	6040
467	3911	81	116	5573
463	6481	135	121	5863
	6841	142	120	5899
	9371	154	116	6109
479	5689	118	126	5976
	6089	126	126	6022
491	6287	130	131	6196
	8627	163	128	6420
	9277	163	127	6472
499	8243	166	132	6492
	9029	166	131	6558
503	2879	59	117	5781
	8663	167	133	6580
523	3331	69	127	6121
	7151	148	142	6697
	7159	149	142	6698
541	2297	47	116	6041
	2437	50	118	6087
547	3001	62	129	6319
557	4759	99	147	6805
	9323	185	153	7345
569	3929	81	144	6794
571	4201	87	147	6873
	7177	149	159	7315
577	1861	38	115	6268
587	5233	109	159	7252
	8263	172	166	7639
599	7001	145	169	7652
	9551	198	170	7920
607	6317	131	170	7664
613	3319	69	149	7170
	4363	90	160	7412
	9157	190	176	8068
617	1511	31	112	6517
619	2297	47	132	6912
	6907	143	176	7895
	9397	195	178	8170

Parameters				
p	q	t	λ	key size
619	9679	201	178	8196
631	2081	43	130	6956
	2693	56	141	7191
643	6247	130	182	8108
647	3559	74	160	7633
653	2311	48	140	7297
	4621	96	182	7950
	8419	175	190	8515
659	2137	44	136	7290
	6781	141	189	8388
	7481	155	192	8481
673	1493	31	120	7097
	9413	196	196	8884
677	3251	67	162	7899
683	5623	117	190	8509
691	1499	31	122	7290
	5471	113	191	8581
	6449	134	198	8745
719	2087	43	145	7929
	2351	48	151	8053
	5153	107	197	8867
	9133	190	209	9460
727	5827	121	205	9094
739	9829	204	215	9802
743	7541	157	216	9571
751	3067	63	170	8699
	3823	79	187	8938
757	1193	24	115	7737
	3727	77	188	8981
	6869	143	221	9649
	7879	164	221	9799
761	1619	33	135	8113
	4091	85	194	9131
	4591	95	202	9258
	7883	164	222	9851
769	1433	29	128	8063
	6599	137	224	9758
773	877	18	100	7558
	2099	43	153	8531
	8317	173	225	10066
	9811	204	225	10251

Parameters				
p	q	t	λ	key size
787	4243	88	203	9485
797	1259	26	124	8208
809	1801	37	148	8749
	6113	127	232	10176
811	8543	177	236	10593
823	4513	94	216	9992
	8069	168	240	10682
827	7219	150	241	10601
	9767	203	241	10961
829	1657	34	145	8866
853	9721	202	249	11300
857	5167	107	234	10572
863	1523	31	143	9125
	4111	85	218	10361
	8779	182	251	11306
881	3217	67	204	10265
	7673	159	257	11370
883	8089	168	257	11463
907	7727	160	264	11715
	8807	183	264	11886
937	1823	37	165	10150
941	2521	52	194	10634
947	3917	81	233	11303
953	6343	132	277	12038
	8237	171	278	12397
967	8243	171	282	12580
971	1913	39	173	10586
	4871	101	258	11895
	9551	198	283	12839
977	5783	120	275	12211
	7817	162	285	12635
991	9349	194	289	13072
997	5393	112	274	12360
1009	1237	25	141	10366
	4219	87	251	12152
	4259	88	254	12165
1013	3923	81	245	12093
	7177	149	295	12976
1019	6691	139	297	12950
1021	5393	112	281	12658
	8819	183	298	13382

Table B.1. Streamlined NTRU Prime parameter sets with $400 < p < 1024$ and $q < 10000$. The estimated pre-quantum security level is 2^{λ}. Parameter sets with $\lambda < 112$ are omitted. The listed key size is $\lceil p \log_2 q \rceil$, not $p\lceil \log_2 q \rceil$(see Subsection 3.2).