Lattice-based cryptography, part 2: efficiency

D. J. Bernstein

University of Illinois at Chicago;
Ruhr University Bochum

2016: Google runs "CECPQ1" experiment, encrypting with elliptic curves and NewHope.

2019: Google+Cloudflare run "CECPQ2" experiment, encrypting with elliptic curves and NTRU HRSS.

2019: OpenSSH adds support for Streamlined NTRU Prime. 2022: OpenSSH enables this *by default*.

These lattice cryptosystems have ≈**1KB keys, ciphertexts**; have ≈**100000 cycles enc, dec**; **maybe resist quantum attacks**.

ECC has much shorter keys and ciphertexts and similar speeds, but doesn't resist quantum attacks.

Isogeny-based crypto has shorter keys and ciphertexts, and maybe resists quantum attacks, but uses many more cycles.

All of the critical design ideas were introduced in the original Hoffstein–Pipher–Silverman NTRU cryptosystem.

Announced 20 August 1996 at Crypto 1996 rump session.
Patent expired in 2017.

First version of NTRU paper, handed out at Crypto 1996, finally put online in 2016:
https://ntru.org/f/hps96.pdf

Proposed 104-byte public keys for 2^{80} security.

1996 paper converted NTRU attack problem into a lattice problem (suboptimally), and then applied LLL (not state of the art) to attack the lattice problem.

1997 Coppersmith–Shamir: better conversion (rescaling) + better attacks than LLL.
No clear quantification.
(Often incorrectly credited for first NTRU lattice attacks.)

NTRU paper, ANTS 1998: proposed 147-byte or 503-byte keys for 2^{77} or 2^{170} security.

NTRU secrets

Parameter: positive integer N.

$\mathbf{Z}[x]$ is the ring of polynomials with integer coeffs.

$R = \mathbf{Z}[x]/(x^N - 1)$ is the ring of polynomials with integer coeffs modulo $x^N - 1$.

(Variants use other moduli: e.g. $x^N - x - 1$ in NTRU Prime.)

NTRU secrets are elements of R with each coeff in $\{-1, 0, 1\}$. (Variants: e.g., $\{-2, -1, 0, 1, 2\}$.)

```
sage: Zx.<x> = ZZ[]
sage: # now Zx is a class
sage: # Zx objects are polys
sage: # in x with int coeffs
sage: f = Zx([3,1,4])
sage: f
4*x^2 + x + 3
sage: g = Zx([2,7,1])
sage: g
x^2 + 7*x + 2
sage: f+g     # built-in add
5*x^2 + 8*x + 5
sage:
```

```
sage: f*x   # built-in mul
4*x^3 + x^2 + 3*x
sage: f*x^2
4*x^4 + x^3 + 3*x^2
sage: f*2
8*x^2 + 2*x + 6
sage: f*(7*x)
28*x^3 + 7*x^2 + 21*x
sage: f*g
4*x^4 + 29*x^3 + 18*x^2 + 23*x
 + 6
sage: f*g == f*2+f*(7*x)+f*x^2
True
sage:
```

```
sage: # replace x^N with 1,
sage: # x^(N+1) with x, etc.
sage: def convolution(f,g):
....:   return (f*g) % (x^N-1)
....:
sage: N = 3  # global variable
sage: convolution(f,x)
x^2 + 3*x + 4
sage: convolution(f,x^2)
3*x^2 + 4*x + 1
sage: convolution(f,g)
18*x^2 + 27*x + 35
sage:
```

```
sage: def randomsecret():
....:   f = list(randrange(3)-1
....:     for j in range(N))
....:   return Zx(f)
....:
sage: N = 7
sage: randomsecret()
-x^3 - x^2 - x - 1
sage: randomsecret()
x^6 + x^5 + x^3 - x
sage: randomsecret()
-x^6 + x^5 + x^4 - x^3 - x^2 +
 x + 1
sage:
```

Will use bigger N for security.

1998 NTRU paper took $N = 503$.

Some choices of N
in NISTPQC submissions:

e.g. $N = 701$ for NTRU HRSS.
e.g. $N = 743$ for NTRUEncrypt.
e.g. $N = 761$ for NTRU Prime.

Overkill against attack algorithms known today, even for future attacker with quantum computer.

Maybe there are faster attacks!
Claimed "guarantees" are fake.

NTRU public keys

Parameter Q, power of 2:
e.g., 4096 for NTRU HRSS.

$R_Q = (\mathbf{Z}/Q)[x]/(x^N - 1)$
is the ring of polynomials
with integer coeffs modulo Q
and modulo $x^N - 1$.

Public key is an element of R_Q.

(Variants: e.g., prime Q.
NTRU Prime has field R_Q: e.g.,
$(\mathbf{Z}/4591)[x]/(x^{761} - x - 1)$.)

NTRU encryption

Ciphertext: $bG + d \in R_Q$ where $G \in R_Q$ is public key and $b, d \in R$ are secrets.

Usually G is invertible in R_Q. Easy to recover b from bG by, e.g., linear algebra. But noise in $bG + d$ spoils linear algebra.

Problem of finding b given $G, bG + d$ (or given $G_1, bG_1 + d_1$, $G_2, bG_2 + d_2, \ldots$) was renamed "Ring-LWE problem" by 2010 Lyubashevsky–Peikert–Regev, without credit to NTRU.

Variant: require d to have "weight W": W nonzero coeffs, $N - W$ zero coeffs. (Generate in constant time via sorting.)

W is another parameter: e.g., 467 for NTRU HRSS.

More traditional variant: require $W/2$ coeffs 1 and $W/2$ coeffs -1.

Variant I'll use in these slides: choose b to have weight W.

Another variant: deterministically round bG to $bG + d$ by rounding each coeff to multiple of 3.

```
sage: def randomweightw():
....:    R = randrange
....:    assert W <= N
....:    s = N*[0]
....:    for j in range(W):
....:      while True:
....:        r = R(N)
....:        if not s[r]: break
....:      s[r] = 1-2*R(2)
....:    return Zx(s)
....:
sage: W = 5
sage: randomweightw()
-x^6 - x^5 + x^4 + x^3 - x^2
sage:
```

NTRU key generation

Secret e, weight-W secret a.
Require e, a invertible in R_Q.
Require a invertible in R_3.

Public key: $G = 3e/a$ in R_Q.

Ring-0LWE problem: find a
given $G/3$ and $a(G/3) - e = 0$.
Homogeneous slice of Ring-LWE$_1$
(find b given G and $bG + d$).

Known attacks: Ring-0LWE
sometimes weaker than Ring-LWE$_1$.
Also, Ring-LWE$_2$ (using G_1, G_2)
sometimes weaker than Ring-LWE$_1$.

```
sage: def balancedmod(f,Q):
....:     g=list(((f[i]+Q//2)%Q)
....:      -Q//2 for i in range(N))
....:     return Zx(g)
....:
sage:
sage: u = 314-159*x
sage: u % 200
-159*x + 114
sage: (u - 400) % 200
-159*x - 86
sage: balancedmod(u,200)
41*x - 86
sage:
```

```
sage: def invertmodprime(f,p):
....:   Fp = Integers(p)
....:   Fpx = Zx.change_ring(Fp)
....:   T = Fpx.quotient(x^N-1)
....:   return Zx(lift(1/T(f)))
....:
sage: N = 7
sage: f = randomsecret()
sage: f3 = invertmodprime(f,3)
sage: convolution(f,f3)
6*x^6 + 6*x^5 + 3*x^4 + 3*x^3 +
 3*x^2 + 3*x + 4
sage:
```

```
def invertmodpowerof2(f,Q):
  assert Q.is_power_of(2)
  g = invertmodprime(f,2)
  M = balancedmod
  conv = convolution
  while True:
    r = M(conv(g,f),Q)
    if r == 1: return g
    g = M(conv(g,2-r),Q)
```

Exercise: Figure out how `invertmodpowerof2` works.

Hint: How many powers of 2 divide first `r-1`? Second `r-1`?

```
sage: N = 7
sage: Q = 256
sage: f = randomsecret()
sage: f
-x^6 - x^4 + x^2 + x - 1
sage: g = invertmodpowerof2(f,Q)
sage: g
47*x^6 + 126*x^5 - 54*x^4 -
 87*x^3 - 36*x^2 - 58*x + 61
sage: convolution(f,g)
-256*x^5 - 256*x^4 + 256*x + 257
sage: balancedmod(_,Q)
1
sage:
```

```
def keypair():
  while True:
    try:
      a = randomweightw()
      a3 = invertmodprime(a,3)
      aQ = invertmodpowerof2(a,Q)
      e = randomsecret()
      G = balancedmod(3 *
                 convolution(e,aQ),Q)
      GQ = invertmodpowerof2(G,Q)
      secretkey = a,a3,GQ
      return G,secretkey
    except:
      pass
```

```
sage: G,secretkey = keypair()
sage: G
-126*x^6 - 31*x^5 - 118*x^4 -
 33*x^3 + 73*x^2 - 16*x + 7
sage: a,a3,GQ = secretkey
sage: a
-x^6 + x^5 - x^4 + x^3 - 1
sage: convolution(a,G)
-3*x^6 + 253*x^5 + 253*x^3 -
 253*x^2 - 3*x - 3
sage: balancedmod(_,Q)
-3*x^6 - 3*x^5 - 3*x^3 + 3*x^2
 - 3*x - 3
sage:
```

```
sage: def encrypt(bd,G):
....:     b,d = bd
....:     bG = convolution(b,G)
....:     C = balancedmod(bG+d,Q)
....:     return C
....:
sage: G,secretkey = keypair()
sage: b = randomweightw()
sage: d = randomsecret()
sage: C = encrypt((b,d),G)
sage: C
120*x^6 + 7*x^5 - 116*x^4 +
 102*x^3 + 86*x^2 - 74*x - 95
sage:
```

NTRU decryption

Given ciphertext $bG + d$, compute $a(bG + d) = 3be + ad$ in R_Q.
a, b, d, e have small coeffs, so $3be + ad$ is not very big.
Assume that coeffs of $3be + ad$ are between $-Q/2$ and $Q/2 - 1$.

Then $3be + ad$ in R_Q reveals $3be + ad$ in $R = \mathbf{Z}[x]/(x^N - 1)$.
Reduce modulo 3: ad in R_3.

Multiply by $1/a$ in R_3 to recover d in R_3.
Coeffs are between -1 and 1, so recover d in R.

```
sage: def decrypt(C,secretkey):
....:     M = balancedmod
....:     conv = convolution
....:     a,a3,GQ = secretkey
....:     u = M(conv(C,a),Q)
....:     d = M(conv(u,a3),3)
....:     b = M(conv(C-d,GQ),Q)
....:     return b,d
....:
sage: decrypt(C,secretkey)
(x^6 - x^5 - x^2 - x - 1, x^5 +
 x^4 + x^3 + x^2 - x)
sage: b,d
(x^6 - x^5 - x^2 - x - 1, x^5 +
 x^4 + x^3 + x^2 - x)
```

```
sage: N,Q,W = 7,256,5
sage: G,secretkey = keypair()
sage: G
44*x^6 - 97*x^5 - 62*x^4 -
 126*x^3 - 10*x^2 + 14*x - 22
sage: a,a3,GQ = secretkey
sage: a
-x^6 - x^5 + x^3 + x - 1
sage: conv = convolution
sage: M = balancedmod
sage: e3 = M(conv(a,G),Q)
sage: e3
-3*x^6 + 3*x^5 + 3*x^4 - 3*x^3
 + 3*x
sage:
```

```
sage: b = randomweightw()
sage: d = randomsecret()
sage: C = M(conv(b,G)+d,Q)
sage: C
-120*x^6 - x^5 + 6*x^4 - 24*x^3
 + 56*x^2 - 98*x - 71
sage: u = M(conv(a,C),Q)
sage: u
8*x^6 - 2*x^5 - 7*x^4 + 4*x^3 -
 6*x - 1
sage: conv(b,e3)+conv(a,d)
8*x^6 - 2*x^5 - 7*x^4 + 4*x^3 -
 6*x - 1
sage:
```

```
sage: # u is 3be+ad in R
sage: M(u,3)
-x^6 + x^5 - x^4 + x^3 - 1
sage: M(conv(a,d),3)
-x^6 + x^5 - x^4 + x^3 - 1
sage: conv(M(u,3),a3)
-3*x^5 + x^4 + x^3 - x - 3
sage: M(_,3)
x^4 + x^3 - x
sage: d
x^4 + x^3 - x
sage:
```

Does decryption always work?

All coeffs of d are in $\{-1, 0, 1\}$.
All coeffs of a are in $\{-1, 0, 1\}$,
and exactly W are nonzero.

Each coeff of ad in R
has absolute value at most W.
(Same argument would work for
a of any weight, d of weight W.)

Similar comments for e, b.
Each coeff of $3be + ad$ in R
has absolute value at most $4W$.

e.g. $W = 467$: at most 1868.
Decryption works for $Q = 4096$.

What about $W = 467$, $Q = 2048$?

Same argument doesn't work.
$a = b = d = e =$
$1 + x + x^2 + \cdots + x^{W-1}$:
$3be + ad$ has a coeff $4W > Q/2$.

But coeffs are usually $<$1024
when a, d are chosen randomly.

1996 NTRU handout mentioned
no-decryption-failure option,
but recommended smaller Q
with some chance of failures.
1998 NTRU paper: decryption
failure "will occur so rarely that
it can be ignored in practice".

Crypto 2003 Howgrave-Graham–Nguyen–Pointcheval–Proos–Silverman–Singer–Whyte "The impact of decryption failures on the security of NTRU encryption":

Decryption failures imply that "all the security proofs known . . . for various NTRU paddings may not be valid after all".

Even worse: Attacker who sees some random decryption failures can figure out the secret key!

Coeff of x^{N-1} in ad is

$a_0 d_{N-1} + a_1 d_{N-2} + \cdots + a_{N-1} d_0$.

This coeff is large $\Leftrightarrow$

$a_0, a_1, \ldots, a_{N-1}$ has

high correlation with

$d_{N-1}, d_{N-2}, \ldots, d_0$.

Some coeff is large $\Leftrightarrow$

$a_0, a_1, \ldots, a_{N-1}$ has high

correlation with some rotation

of $d_{N-1}, d_{N-2}, \ldots, d_0$.

i.e. a is correlated with

$x^i \operatorname{rev}(d)$ for some i, where

$\operatorname{rev}(d) = d_0 + d_1 x^{N-1} + \cdots + d_{N-1} x$.

Reasonable guesses given a random decryption failure:
a correlated with some $x^i\,\mathrm{rev}(d)$.
$\mathrm{rev}(a)$ correlated with $x^{-i}d$.
$a\,\mathrm{rev}(a)$ correlated with $d\,\mathrm{rev}(d)$.

Experimentally confirmed:
Average of $d\,\mathrm{rev}(d)$
over some decryption failures
is close to $a\,\mathrm{rev}(a)$.
Round to integers: $a\,\mathrm{rev}(a)$.

Eurocrypt 2002 Gentry–Szydlo algorithm then finds a.

1999 Hall–Goldberg–Schneier, 2000 Jaulmes–Joux, 2000 Hoffstein–Silverman, 2016 Fluhrer, etc.: Even easier attacks using invalid messages.

Attacker changes d to
$d \pm 1$, $d \pm x$, $\ldots$, $d \pm x^{N-1}$;
$d \pm 2$, $d \pm 2x$, $\ldots$, $d \pm 2x^{N-1}$;
$d \pm 3$, etc.

This changes $3be + ad$: adds
$\pm a$, $\pm xa$, $\ldots$, $\pm x^{N-1}a$;
$\pm 2a$, $\pm 2xa$, $\ldots$, $\pm 2x^{N-1}a$;
$\pm 3a$, etc.

e.g. $3be+ad = \cdots + 390x^{478} + \cdots$,
all other coeffs in $[-389, 389]$;
and $a = \cdots + x^{478} + \cdots$.

Then $3be + ad + ka =$
$\cdots + (390 + k)x^{478} + \cdots$.
Decryption fails for big k.

Search for smallest k that fails.

Does $3be + ad + kxa$ also fail?
Yes *if* $xa = \cdots + x^{478} + \cdots$,
i.e., if $a = \cdots + x^{477} + \cdots$.

Try kx^2, kx^3, etc.
See pattern of a coeffs.

Brute-force search

Attacker is given public key $G = 3e/a$, ciphertext $C = bG + d$. Can attacker find b?

Search $\binom{N}{W}2^W$ choices of b. If $d = C - bG$ is small: done!

(Can this find two different secrets d? Unlikely. This would also stop legitimate decryption.)

Or search through choices of a. If $e = aG/3$ is small, use (a, e) to decrypt. Advantage: can reuse attack for many ciphertexts.

Equivalent keys

Secret key (a, e) is equivalent to secret key (xa, xe), secret key (x^2a, x^2e), etc.

Search only $\approx\binom{N}{W}2^W/N$ choices.

$N = 701$, $W = 467$:

$$\binom{N}{W}2^W \approx 2^{1106.09};$$
$$\binom{N}{W}2^W/N \approx 2^{1096.64}.$$

$N = 701$, $W = 200$:

$$\binom{N}{W}2^W \approx 2^{799.76};$$
$$\binom{N}{W}2^W/N \approx 2^{790.31}.$$

Exercise: Find more equivalences!

Collision attacks

Write a as $a_1 + a_2$ where
$a_1 =$ bottom $\lceil N/2 \rceil$ terms of a,
$a_2 =$ remaining terms of a.

$e = (G/3)a = (G/3)a_1 + (G/3)a_2$
so $e - (G/3)a_2 = (G/3)a_1$.
Eliminate e: almost certainly
$H(-(G/3)a_2) = H((G/3)a_1)$ for
$H(f) = ([f_0 < 0], \ldots, [f_{k-1} < 0])$.

Enumerate all $H(-(G/3)a_2)$.
Enumerate all $H((G/3)a_1)$.
Search for collisions.
Only about $3^{N/2}$ operations:
$\approx 2^{555.52}$ for $N = 701$.

Lattice view of NTRU

Given public key $G = 3e/a$.
Compute $H = G/3 = e/a$ in R_Q.

$a \in R$ is obtained from
$1, x, \ldots, x^{N-1}$
by a few additions, subtractions.

$aH \in R_Q$ is obtained from
$H, xH, \ldots, x^{N-1}H$
by a few additions, subtractions.

$e \in R$ is obtained from
$Q, Qx, Qx^2, \ldots, Qx^{N-1}$,
$H, xH, \ldots, x^{N-1}H$
by a few additions, subtractions.

$(e, a) \in R^2$ is obtained from

$(Q, 0)$,

$(Qx, 0)$,

$\vdots$

$(Qx^{N-1}, 0)$,

$(H, 1)$,

(xH, x),

$\vdots$

$(x^{N-1}H, x^{N-1})$

by a few additions, subtractions.

Write H as

$H_0 + H_1 x + \cdots + H_{N-1} x^{N-1}$.

$(e_0, e_1, \ldots, e_{N-1}, a_0, a_1, \ldots, a_{N-1})$
is obtained from

$(Q, 0, \ldots, 0, 0, 0, \ldots, 0),$
$(0, Q, \ldots, 0, 0, 0, \ldots, 0),$
$\vdots$
$(0, 0, \ldots, Q, 0, 0, \ldots, 0),$
$(H_0, H_1, \ldots, H_{N-1}, 1, 0, \ldots, 0),$
$(H_{N-1}, H_0, \ldots, H_{N-2}, 0, 1, \ldots, 0),$
$\vdots$
$(H_1, H_2, \ldots, H_0, 0, 0, \ldots, 1)$

by a few additions, subtractions.

$(e_0, e_1, \ldots, e_{N-1}, a_0, a_1, \ldots, a_{N-1})$
is a surprisingly short vector
in lattice generated by
$(Q, 0, \ldots, 0, 0, 0, \ldots, 0)$ etc.

Attacker searches for short vector
in this lattice using (e.g.) BKZ.

Many speedups. e.g. rescaling:
set up lattice to contain $(e, 10a)$
if e is chosen $10\times$ larger than a.

Exercise: Describe search for
(d, b) as a problem of finding

- a lattice vector near a point;
- a short vector in a lattice.