Lattice-based cryptography, day 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.

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.

ased cryptography,
fficiency

rnstein

ty of Illinois at Chicago;
iversity Bochum

oogle runs “CECPQ1”
ent, encrypting with
urves and NewHope.

oogle+Cloudflare
CPQ2” experiment,
ng with elliptic curves
RU HRSS.

2019: OpenSSH adds support for Streamlined NTRU Prime.

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 th
were intr
Hoffstei
NTRU☢

Announc
at Crypt
Patent

otography,

is at Chicago;
ochum

s "CECPQ1"
oting with
NewHope.

oudflare
xperiment,
liptic curves

2019: OpenSSH adds support for Streamlined NTRU Prime.

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 c
were introduced in
Hoffstein–Pipher–S
NTRU☢ cryptosy

Announced 20 Au
at Crypto 1996 ru
Patent expired i

ago;

1"

e.

es

2019: OpenSSH adds support for Streamlined NTRU Prime.

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 idea
were introduced in the origin
Hoffstein–Pipher–Silverman
NTRU☢ cryptosystem.

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

2019: OpenSSH adds support for Streamlined NTRU Prime.

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.**

2019: OpenSSH adds support for Streamlined NTRU Prime.

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

2019: OpenSSH adds support for Streamlined NTRU Prime.

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.

penSSH adds support for
ned NTRU Prime.

ttice cryptosystems
KB **keys, ciphertexts**;
00000 **cycles enc, dec**;
resist quantum attacks.

s much shorter keys and
xts and similar speeds, but
resist quantum attacks.

based crypto has
keys and ciphertexts, and
esists quantum attacks,
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 pa
attack p
problem
applied
to attac

dds support for
J Prime.

tosystems
ciphertexts;
cles enc, dec;
ntum attacks.

orter keys and
milar speeds, but
ntum attacks.

oto has
iphertexts, and
ntum attacks,
ore 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 conver
attack problem int
problem (suboptim
applied LLL (not s
to attack the latti

rt for

xts;
dec;
acks.

and
ds, but
cks.

, and
cks,

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
applied LLL (not state of th
to attack the lattice problem

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.

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.)

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.

e critical design ideas
roduced in the original
n–Pipher–Silverman
cryptosystem.

ced 20 August 1996
o 1996 rump session.
expired in 2017.

rsion of NTRU paper,
out at Crypto 1996,
ut online in 2016:
//ntru.org/f/hps96.pdf

d 104-byte public keys
ecurity.

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 s

Paramet

$\mathbf{Z}[x]$ is t
with inte

$R = \mathbf{Z}[x$
the ring
integer c

design ideas
n the original
Silverman
stem.

gust 1996
mp session.
n 2017.

TRU paper,
ypto 1996,
n 2016:
rg/f/hps96.pdf

e public keys

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: positiv

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

$R = \mathbf{Z}[x]/(x^N - 1$
the ring of polyno
integer coeffs mod

as
nal

n.

r,

96.pdf

eys

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

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

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

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 $x^N - 1$.

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.)

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.

<u>NTRU secrets</u>

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., $\{-2, -1, 0, 1, 2\}$.)

per converted NTRU
roblem into a lattice
(suboptimally), and then
LLL (not state of the art)
k the lattice problem.

ppersmith–Shamir:
onversion (rescaling) +
ttacks than LLL.
quantification.
ncorrectly credited
NTRU lattice attacks.)

paper, ANTS 1998:
d 147-byte or 503-byte
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: Z
sage: #
sage: #
sage: #
sage:
```

rted NTRU
to a lattice
nally), and then
state of the art)
ce problem.

–Shamir:
(rescaling) $+$
n LLL.
ation.
credited
tice attacks.)

TS 1998:
or 503-byte
^{70} 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> = Z
sage: # now Zx i
sage: # Zx objec
sage: # in x wit
sage:
```

U
e
l then
e art)
n.

) +

ks.)

yte

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 po
sage: # in x with int coe
sage:
```

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:
```

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:
```

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:
```

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:
```

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:
```

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:
```

ecrets

er: positive integer N.

he ring of polynomials
eger coeffs.

$]/(x^N - 1)$ is
of polynomials with
coeffs modulo $x^N - 1$.

s use other moduli:
$- x - 1$ in NTRU Prime.)

ecrets are elements of
each coeff in $\{-1, 0, 1\}$.
s: 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
4*x^3 +
sage:
```

ve integer N.

polynomials

s.

.) is

mials with

lulo $x^N - 1$.

r moduli:

NTRU Prime.)

elements of

in $\{-1, 0, 1\}$.

$-2, -1, 0, 1, 2\}$.)


```
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    # bu
4*x^3 + x^2 + 3*
sage:
```

N.

als

1.

rime.)

of

$, 1\}$.

$1, 2\}$.)


```
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 mu
4*x^3 + x^2 + 3*x
sage:
```

```
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:
```

```
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:
```

```
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:
```

```
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:
```

```
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:
```

```
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:
```

```
x.<x> = ZZ[]
 now Zx is a class
 Zx objects are polys
 in x with int coeffs
 = Zx([3,1,4])

 x + 3
 = Zx([2,7,1])

*x + 2
+g    # built-in add
 8*x + 5
```



```
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: #
sage: #
sage: de
....:
....:
sage:
```

```
Z[]
s a class
ts are polys
h int coeffs
1,4])


7,1])


uilt-in add
```



```
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
sage: # x^(N+1)
sage: def convol
....:   return (
....:
sage:
```

```
s
lys
effs
dd
```

```
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
sage: # x^(N+1) with x, e
sage: def convolution(f,g
....:   return (f*g) % (x
....:
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:
```

```
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:
```

```
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:
```

```
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:
```

```
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:
```

```
*x   # built-in mul
 x^2 + 3*x
*x^2
 x^3 + 3*x^2
*2
 2*x + 6
*(7*x)
+ 7*x^2 + 21*x
*g
 29*x^3 + 18*x^2 + 23*x

*g == f*2+f*(7*x)+f*x^2
```



```
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: d
....:
....:
....:
....:
sage:
```

```
ilt-in mul
x

x^2



 21*x

 18*x^2 + 23*x

+f*(7*x)+f*x^2
```



```
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 random
....:   f = list
....:     for j
....:   return Z
....:
sage:
```

```
l

- 23*x

f*x^2
```



```
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(randrang
....:     for j in range(
....:   return Zx(f)
....:
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:
```

```
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:
```

```
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:
```

```
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:
```

```
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:
```

```
 replace x^N with 1,
 x^(N+1) with x, etc.
ef convolution(f,g):
 return (f*g) % (x^N-1)

 = 3  # global variable
onvolution(f,x)
*x + 4
onvolution(f,x^2)
 4*x + 1
onvolution(f,g)
+ 27*x + 35
```



```
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

1998 NT

Some ch

in NIST

e.g. $N =$

e.g. $N =$

e.g. $N =$

x^N with 1,
with x, etc.
ution(f,g):
f*g) % (x^N-1)

lobal variable
n(f,x)

n(f,x^2)

n(f,g)
35


```
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

1998 NTRU paper

Some choices of N
in NISTPQC subm

e.g. $N = 701$ for N
e.g. $N = 743$ for N
e.g. $N = 761$ for N

1,

tc.

):

^N-1)

iable

```
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 securit

1998 NTRU paper took $N =$

Some choices of N
in NISTPQC submissions:

e.g. $N = 701$ for NTRU HR

e.g. $N = 743$ for NTRUEncr

e.g. $N = 761$ for NTRU Prir

```
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.

```
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.

```
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.

```
ef randomsecret():
 f = list(randrange(3)-1
   for j in range(N))
 return Zx(f)

 = 7
andomsecret()
x^2 - x - 1
andomsecret()
^5 + x^3 - x
andomsecret()
x^5 + x^4 - x^3 - x^2 +
```


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 p

Paramet
e.g., 409

$R_Q = ($Z

is the ri
with inte
and mod

Public k

(Variant
NTRU P
($\mathbf{Z}/4591$

```
secret():
(randrange(3)-1
in range(N))
x(f)


et()
 1
et()
- x
et()
 - x^3 - x^2 +
```


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, pow
e.g., 4096 for NTF

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

Public key is an el

(Variants: e.g., pr
NTRU Prime has
$(\mathbf{Z}/4591)[x]/(x^{761}$

```
ge(3)-1
N))
```

```
x^2 +
```


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

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

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., $(\mathbf{Z}/4591)[x]/(x^{761} - x - 1)$.)

bigger N for security.

TRU paper took $N = 503$.

oices of N
PQC submissions:

$= 701$ for NTRU HRSS.
$= 743$ for NTRUEncrypt.
$= 761$ for NTRU Prime.

against attack algorithms
oday, even for future
with quantum computer.

here are faster attacks!
"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 e

Ciphert
where G
and b, d

for security.

took $N = 503$.

N

missions:

NTRU HRSS.

NTRUEncrypt.

NTRU Prime.

ttack algorithms

for future

ntum computer.

aster attacks!

ees" 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: $bG +$

where $G \in R_Q$ is p

and $b, d \in R$ are s

ty.

= 503.

SS.

rypt.

me.

rithms

e

puter.

cks!

ke.

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.

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.

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.

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.

oublic keys

ter Q, power of 2:
96 for NTRU HRSS.

$\mathbf{Z}/Q)[x]/(x^N - 1)$
ng of polynomials
eger coeffs modulo Q
dulo $x^N - 1$.

ey is an element of R_Q.

s: e.g., prime Q.
Prime has field R_Q: e.g.,
$)[x]/(x^{761} - x - 1)$.)

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:
"weight
$N - W$
in consta

W is anc
e.g., 467

er of 2:
RU HRSS.

$x^N - 1)$
nomials
s modulo Q
1.

ement of R_Q.

ime Q.
field R_Q: e.g.,
$- x - 1)$.)

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
"weight W": W n
$N - W$ zero coeffs
in constant time v

W is another para
e.g., 467 for NTRU

Q

R_Q.

e.g.,

.)

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 co
$N - W$ zero coeffs. (Genera
in constant time via sorting.

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

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.

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.

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.

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.

12

encryption

ext: $bG + d \in R_Q$

$\in R_Q$ is public key

$\in R$ are secrets.

G is invertible in R_Q.

recover b from bG by,

ear algebra. But noise in

spoils linear algebra.

of finding b given

d (or given $G_1, bG_1 + d_1,$

$+ d_2, \ldots$) was renamed

WE problem" by 2010

evsky–Peikert–Regev,

credit to NTRU.

13

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: d
....:
....:
....:
....:
....:
....:
....:
....:
....:
....:
sage: W
sage: r
-x^6 -
sage:
```

$d \in R_Q$
public key
secrets.

tible in R_Q.
from bG by,
. But noise in
ar algebra.

b given
en $G_1, bG_1 + d_1$,
was renamed
m" by 2010
kert–Regev,
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 random
....:   R = rand
....:   assert W
....:   s = N*[0
....:   for j in
....:     while
....:       r =
....:       if n
....:     s[r] =
....:   return Z
....:
sage: W = 5
sage: randomweig
-x^6 - x^5 + x^4
sage:
```

).
py,
se in
.
$+ d_1$,
med
10
v,

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]:
....:     s[r] = 1-2*R(2)
....:   return Zx(s)
....:
sage: W = 5
sage: randomweightw()
-x^6 - x^5 + x^4 + x^3 -
sage:
```

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:
```

require d to have
W": W nonzero coeffs,
zero coeffs. (Generate
ant time via sorting.)

other parameter:
7 for NTRU HRSS.

aditional variant: require
effs 1 and $W/2$ coeffs -1.

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

variant: deterministically
G to $bG + d$ by rounding
eff 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 k

Secret e

Require

Require

to have
onzero coeffs,
s. (Generate
ia sorting.)

meter:
U HRSS.

ariant: require
$W/2$ coeffs -1.

these slides:
weight W.

deterministically
- d by rounding
iple of 3.


```
sage: def randomweighttw():
....:   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: randomweighttw()
-x^6 - x^5 + x^4 + x^3 - x^2
sage:
```

NTRU key genera

Secret e, weight-V
Require e, a invert
Require a invertibl

effs,
te
)

quire
fs -1.

s:

tically
nding


```
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.

```
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.

```
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.

```
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 $a(G/3) - e = 0$.

```
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 $bG + d$).

```
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$_1$.

```
ef 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)


 = 5
andomweightw()
x^5 + x^4 + x^3 - x^2
```


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: d
....:
....:
....:
....:
sage:
sage:
```

```
weightw():
range
 <= N
]
 range(W):
True:
R(N)
ot s[r]: break
 1-2*R(2)
x(s)


htw()
 + x^3 - x^2
```


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 balanc
....:    g=list((
....:     -Q//2 f
....:    return Z
....:
sage:
sage:
```

:

:

break

x^2

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//
....:    -Q//2 for i in r
....:   return Zx(g)
....:
sage:
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:
```

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:
```

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:
```

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:
```

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:
```

key generation

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

ey: $G = 3e/a$ in R_Q.

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

attacks: Ring-0LWE
es weaker than Ring-LWE$_1$.
ng-LWE$_2$ (using G_1, G_2)
es 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: de
....:
....:
....:
....:
....:
sage:
```

tion

V secret a.
ible in R_Q.
e in R_3.

e/a in R_Q.

m: find a
$G/3) - e = 0$.
e of Ring-LWE$_1$
d $bG + d$).

ing-0LWE
than Ring-LWE$_1$.
(using G_1, G_2)
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 invert
....:   Fp = Int
....:   Fpx = Zx
....:   T = Fpx.
....:   return Z
....:
sage:
```

.

).

).

$= 0.$

LWE_1

).

g-LWE_1.

G_2)

g-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(
....:   Fp = Integers(p)
....:   Fpx = Zx.change_r
....:   T = Fpx.quotient(
....:   return Zx(lift(1/
....:
sage:
```

```
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:
```

```
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:
```

```
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:
```

```
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:
```

```
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:
```

```
ef balancedmod(f,Q):
 g=list(((f[i]+Q//2)%Q)
  -Q//2 for i in range(N))
 return Zx(g)


 = 314-159*x
 % 200
+ 114
u - 400) % 200
- 86
alancedmod(u,200)
86
```



```
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 inv
  asser
  g = i
  M = b
  conv 
  while
    r =
    if 
    g =
```

Exercise

`invertm`

Hint: H

divide fi

```
edmod(f,Q):
(f[i]+Q//2)%Q)
or i in range(N))
x(g)

9*x

% 200

d(u,200)
```



```
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 invertmodpow
  assert Q.is_po
  g = invertmodp
  M = balancedmo
  conv = convolu
  while True:
    r = M(conv(g
    if r == 1: r
    g = M(conv(g
```

Exercise: Figure o
`invertmodpowerc`
Hint: How many p
divide first `r-1`? S

```
)):
/2)%Q)
range(N))
```



```
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 divide first `r-1`? Second `r-`

```
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`?

```
ef invertmodprime(f,p):
 Fp = Integers(p)
 Fpx = Zx.change_ring(Fp)
 T = Fpx.quotient(x^N-1)
 return Zx(lift(1/T(f)))

 = 7
 = randomsecret()
3 = invertmodprime(f,3)
onvolution(f,f3)
 6*x^5 + 3*x^4 + 3*x^3 +
+ 3*x + 4
```



```
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
sage: Q
sage:
```

```
modprime(f,p):
egers(p)
.change_ring(Fp)
quotient(x^N-1)
x(lift(1/T(f)))



secret()
tmodprime(f,3)
n(f,f3)
3*x^4 + 3*x^3 +
```



```
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,p):
ring(Fp)
(x^N-1)
T(f)))
(f,3)
3*x^3 +
```



```
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:
```

```
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:
```

```
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:
```

```
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:
```

```
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:
```

```
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:
```

```
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:
```

```
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:
```

```
ertmodpowerof2(f,Q):
t Q.is_power_of(2)
nvertmodprime(f,2)
alancedmod
= convolution
 True:
 M(conv(g,f),Q)
r == 1: return g
 M(conv(g,2-r),Q)
```

: Figure out how
odpowerof2 works.
ow many powers of 2
rst 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 key
  while
    try
      a
      a3
      a0
      e
      G
      
      G0
      s
      r
    exc
      pa
```

```
erof2(f,Q):
wer_of(2)
rime(f,2)
d
tion

,f),Q)
eturn g
,2-r),Q)
```

ut how
of2 works.
powers of 2
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 = random
      a3 = inver
      aQ = inver
      e = random
      G = balanc
             con
      GQ = inver
      secretkey
      return G,s
    except:
      pass
```

):

2

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
      aQ = invertmodpower
      e = randomsecret()
      G = balancedmod(3 *
              convolution(
      GQ = invertmodpower
      secretkey = a,a3,GQ
      return G,secretkey
    except:
      pass
```

```
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
```

```
= 7
= 256
= randomsecret()

x^4 + x^2 + x - 1
= invertmodpowerof2(f,Q)

+ 126*x^5 - 54*x^4 -
- 36*x^2 - 58*x + 61
onvolution(f,g)
5 - 256*x^4 + 256*x + 257
alancedmod(_,Q)
```



```
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
sage:
```

```
secret()

 + x - 1
modpowerof2(f,Q)

 - 54*x^4 -
 - 58*x + 61
n(f,g)
^4 + 256*x + 257
d(_,Q)
```



```
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,secretke
sage:
```

```
f2(f,Q)
 -
 61
x + 257
```



```
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 = keypa
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:
```

```
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:
```

```
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:
```

```
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:
```

```
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:
```

```
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:
```

```
pair():
 True:
:
 = randomweightw()
3 = invertmodprime(a,3)
Q = invertmodpowerof2(a,Q)
 = randomsecret()
 = balancedmod(3 *
      convolution(e,aQ),Q)
Q = invertmodpowerof2(G,Q)
ecretkey = a,a3,GQ
eturn G,secretkey
ept:
ass
```



```
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: de
....:
....:
....:
....:
....:
sage:
```

```
weightw()
tmodprime(a,3)
tmodpowerof2(a,Q)
secret()
edmod(3 *
volution(e,aQ),Q)
tmodpowerof2(G,Q)
= a,a3,GQ
ecretkey
```



```
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 encryp
....:   b,d = bd
....:   bG = con
....:   C = bala
....:   return C
....:
sage:
```

```
e(a,3)
rof2(a,Q)

:
(e,aQ),Q)
rof2(G,Q)
)
```



```
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(
....:     C = balancedmod(b
....:     return C
....:
sage:
```

```
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:
```

```
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:
```

```
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:
```

```
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:
```

```
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:
```

```
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:
```

```
,secretkey = keypair()

6 - 31*x^5 - 118*x^4 -
 + 73*x^2 - 16*x + 7
,a3,GQ = secretkey

x^5 - x^4 + x^3 - 1
onvolution(a,G)
+ 253*x^5 + 253*x^3 -
2 - 3*x - 3
alancedmod(_,Q)
- 3*x^5 - 3*x^3 + 3*x^2
- 3
```



```
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 d

Given ci

$a(bG +$

```
y = keypair()

5 - 118*x^4 -
 - 16*x + 7
secretkey

 + x^3 - 1
n(a,G)
 + 253*x^3 -
 3
d(_,Q)
 3*x^3 + 3*x^2
```



```
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 b

$a(bG + d) = 3be$

```
ir()
^4 -
 7
-
 1
 3 -
 3*x^2
```



```
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 $bG + d$, co
$a(bG + d) = 3be + ad$ in R

```
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.

```
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.

```
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 $Q/2 - 1$.

```
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 $R = \mathbf{Z}[x]/(x^N - 1)$.

```
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.

```
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.

```
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.

```
ef encrypt(bd,G):
  b,d = bd
  bG = convolution(b,G)
  C = balancedmod(bG+d,Q)
  return C

,secretkey = keypair()
  = randomweightw()
  = randomsecret()
  = encrypt((b,d),G)

  + 7*x^5 - 116*x^4 +
3 + 86*x^2 - 74*x - 95
```


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: de
....:
....:
....:
....:
....:
....:
....:
....:
sage:
```

```
t(bd,G):

volution(b,G)
ncedmod(bG+d,Q)

y = keypair()
weightw()
secret()
t((b,d),G)

- 116*x^4 +
2 - 74*x - 95
```


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 decryp
....:     M = bala
....:     conv = c
....:     a,a3,GQ
....:     u = M(co
....:     d = M(co
....:     b = M(co
....:     return b
....:
sage:
```

```
(b,G)
G+d,Q)
ir()
)
+
- 95
```


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,secre
....:   M = balancedmod
....:   conv = convolutio
....:   a,a3,GQ = secretk
....:   u = M(conv(C,a),Q
....:   d = M(conv(u,a3),
....:   b = M(conv(C-d,GQ
....:   return b,d
....:
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:
```

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:
```

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)
```

lecryption

phertext $bG + d$, compute
$d) = 3be + ad$ in R_Q.
have small coeffs,
$- ad$ is not very big.
that coeffs of $3be + ad$
veen $-Q/2$ and $Q/2 - 1$.

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

by $1/a$ in R_3
er d in R_3.
re between -1 and 1,
er 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
sage:
```

$bG + d$, compute
$+ ad$ in R_Q.
ll coeffs,
t very big.
fs of $3be + ad$
2 and $Q/2 - 1$.

R_Q reveals
$\mathbf{Z}[x]/(x^N - 1)$.
ad in R_3.

R_3
3.
n -1 and 1,


```
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,
sage:
```

mpute

Q.

$+ ad$

$- 1$.

ls

$- 1)$.

.

1,


```
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:
```

```
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:
```

```
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:
```

```
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:
```

```
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:
```

```
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:
```

```
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:
```

```
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:
```

```
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:
```

```
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:
```

```
ef 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

ecrypt(C,secretkey)
x^5 - x^2 - x - 1, x^5 +
x^3 + x^2 - x)
,d
x^5 - x^2 - x - 1, x^5 +
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
sage:
```

```
t(C,secretkey):
ncedmod
onvolution
= secretkey
nv(C,a),Q)
nv(u,a3),3)
nv(C-d,GQ),Q)
,d

secretkey)
 - x - 1, x^5 +
 - x)

 - x - 1, x^5 +
 - 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 = random
sage:
```

```
tkey):

n
ey
)
3)
),Q)

)
 x^5 +

 x^5 +
```



```
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:
```

```
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:
```

```
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:
```

```
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:
```

```
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:
```

```
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:
```

```
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:
```

```
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:
```

```
,Q,W = 7,256,5
,secretkey = keypair()

- 97*x^5 - 62*x^4 -
3 - 10*x^2 + 14*x - 22
,a3,GQ = secretkey

x^5 + x^3 + x - 1
onv = convolution
 = balancedmod
3 = M(conv(a,G),Q)
3
+ 3*x^5 + 3*x^4 - 3*x^3
```



```
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: #
sage: M
-x^6 +
sage:
```

```
256,5
y = keypair()

- 62*x^4 -
2 + 14*x - 22
secretkey

 + x - 1
volution
edmod
v(a,G),Q)

 3*x^4 - 3*x^3
```



```
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
sage: M(u,3)
-x^6 + x^5 - x^4
sage:
```

```
ir()
-
- 22
3*x^3
```



```
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 -
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:
```

```
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:
```

```
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:
```

```
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:
```

```
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:
```

```
= randomweightw()
= randomsecret()
= M(conv(b,G)+d,Q)

6 - x^5 + 6*x^4 - 24*x^3
^2 - 98*x - 71
= M(conv(a,C),Q)

2*x^5 - 7*x^4 + 4*x^3 -
1
onv(b,e3)+conv(a,d)
2*x^5 - 7*x^4 + 4*x^3 -
1
```



```
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 de

All coeff
All coeff
and exac

```
weightw()
secret()
(b,G)+d,Q)

 6*x^4 - 24*x^3
 - 71
(a,C),Q)

7*x^4 + 4*x^3 -

+conv(a,d)
7*x^4 + 4*x^3 -
```



```
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 a

All coeffs of d are

All coeffs of a are

and exactly W are

)

24*x^3

*x^3 -

)

*x^3 -

```
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 $\{-1, 0$

All coeffs of a are in $\{-1, 0,$

and exactly W are nonzero.

```
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.

```
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.

```
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.)

```
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 $4W$.

```
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 $Q = 4096$.

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

^3 - x
```


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 ab

```
+ad in R

 + x^3 - 1
),3)
 + x^3 - 1
),a3)
^3 - x - 3
```


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 $W =$

1

1

3

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 $W = 467$, $Q =$

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

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.
$a = b = c = d =$
$1 + x + x^2 + \cdots + x^{W-1}$:
$3be + ad$ has a coeff $4W > Q/2$.

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.
$a = b = c = d =$
$1 + x + x^2 + \cdots + x^{W-1}$:
$3be + ad$ has a coeff $4W > Q/2$.

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

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.
$a = b = c = d =$
$1 + x + x^2 + \cdots + x^{W-1}$:
$3be + ad$ 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".

cryption always work?

fs of d are in $\{-1, 0, 1\}$.
fs of a are in $\{-1, 0, 1\}$,
ctly W are nonzero.

eff of ad in R
olute value at most W.
rgument would work for
weight, d of weight W.)

comments for e, b.
eff of $3be + ad$ in R
olute value at most $4W$.

$= 467$: at most 1868.
ion works for $Q = 4096$.

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

Same argument doesn't work.
$a = b = c = d =$
$1 + x + x^2 + \cdots + x^{W-1}$:
$3be + ad$ 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 2
Nguyen–
Silverma
"The im
decrypti
security

Decrypti
"all the
for vario
may not

lways work?

in $\{-1, 0, 1\}$.
in $\{-1, 0, 1\}$,
nonzero.

n R
at most W.
vould work for
of weight W.)

for e, b.
$+ ad$ in R
at most $4W$.

most 1868.
for $Q = 4096$.

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

Same argument doesn't work.
$a = b = c = d =$
$1 + x + x^2 + \cdots + x^{W-1}$:
$3be + ad$ 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 Howg
Nguyen–Pointchev
Silverman–Singer–
"The impact of
decryption failures
security of NTRU

Decryption failures
"all the security pr
for various NTRU
may not be valid a

x?

, 1}.

1},

W.

x for

t W.)

?

4W.

3.

096.

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

Same argument doesn't work.
$a = b = c = d =$
$1 + x + x^2 + \cdots + x^{W-1}$:
$3be + ad$ 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-Grah
Nguyen–Pointcheval–Proos–
Silverman–Singer–Whyte
"The impact of
decryption failures on the
security of NTRU encryptio

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

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

Same argument doesn't work.
$a = b = c = d =$
$1 + x + x^2 + \cdots + x^{W-1}$:
$3be + ad$ 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".

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

Same argument doesn't work.
$a = b = c = d =$
$1 + x + x^2 + \cdots + x^{W-1}$:
$3be + ad$ 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!

bout $W = 467$, $Q = 2048$?

gument doesn't work.

$c = d =$
$x^2 + \cdots + x^{W-1}$:
d has a coeff $4W > Q/2$.

ffs are usually $<$1024
d are chosen randomly.

TRU handout mentioned
ption-failure option,
mmended smaller Q
ne chance of failures.
TRU paper: decryption
will occur so rarely that
e 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
$a_0 d_{N-1}$

This coe
$a_0, a_1, \ldots$
high cor
d_{N-1}, d

467, $Q = 2048$?

oesn't work.

– x^{W-1}:

oeff $4W > Q/2$.

ally $<$1024

sen randomly.

out mentioned

ure option,

smaller Q

of failures.

r: decryption

so rarely that

n 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 a

$a_0 d_{N-1} + a_1 d_{N-2}$

This coeff is large

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

high correlation w

$d_{N-1}, d_{N-2}, \ldots, d$

2048?

k.

$Q/2$.

4

mly.

oned

)

s.

ion

that

”.

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 $a_0 d_{N-1} + a_1 d_{N-2} + \cdots + a$

This coeff is large $\Leftrightarrow$ $a_0, a_1, \ldots, a_{N-1}$ has high correlation with $d_{N-1}, d_{N-2}, \ldots, d_0$.

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

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

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

2003 Howgrave-Graham–
-Pointcheval–Proos–
n–Singer–Whyte
pact of
on failures on the
of NTRU encryption":

ion failures imply that
security proofs known . . .
us NTRU paddings
be valid after all".

rse: Attacker who sees
ndom decryption failures
re 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
$\operatorname{rev}(d) = d_0 + d_1 x^{N-1} + \cdots + d_{N-1} x$.

Reasona
random
a correla

grave-Graham–
val–Proos–
-Whyte

. on the
encryption”:

s imply that
roofs known . . .
paddings
after all”.

cker who sees
ryption failures
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 guesse
random decryptior
a correlated with s

nam–

–

n”:

at

vn . . .

ees

lures

y!

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 re

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 $x^i \operatorname{rev}(d)$.

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 $x^i \operatorname{rev}(d)$.
$\operatorname{rev}(a)$ correlated with $x^{-i} d$.

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 $x^i \operatorname{rev}(d)$.
$\operatorname{rev}(a)$ correlated with $x^{-i} d$.
$a \operatorname{rev}(a)$ correlated with $d \operatorname{rev}(d)$.

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 $x^i \operatorname{rev}(d)$.
$\operatorname{rev}(a)$ correlated with $x^{-i} d$.
$a \operatorname{rev}(a)$ correlated with $d \operatorname{rev}(d)$.

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

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 $x^i \operatorname{rev}(d)$.
$\operatorname{rev}(a)$ correlated with $x^{-i} d$.
$a \operatorname{rev}(a)$ correlated with $d \operatorname{rev}(d)$.

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

Eurocrypt 2002 Gentry–Szydlo
algorithm then finds a.

x^{N-1} in ad is

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

eff is large $\Leftrightarrow$
. ., a_{N-1} has
relation with
$_{N-2}, \ldots, d_0$.

oeff is large $\Leftrightarrow$
. ., a_{N-1} has high
on with some rotation
, $d_{N-2}, \ldots, d_0$.

correlated with
) for some i, where
$= d_0 + d_1 x^{N-1} + \cdots + d_{N-1} x$.

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

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

Eurocrypt 2002 Gentry–Szydlo
algorithm then finds a.

1999 Ha
2000 Ja
Hoffstei
Fluhrer,
using in

ad is
$+ \cdots + a_{N-1} d_0$.

$\Leftrightarrow$
as
ith
d_0.

e $\Leftrightarrow$
as high
ome rotation
, d_0.

with
i, where
$^{N-1} + \cdots + d_{N-1} x$.

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

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

Eurocrypt 2002 Gentry–Szydlo algorithm then finds a.

1999 Hall–Goldber
2000 Jaulmes–Jou
Hoffstein–Silverma
Fluhrer, etc.: Ever
using invalid mess

$_{N-1}d_0$.

on

$-d_{N-1}x$.

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–Schneie
2000 Jaulmes–Joux, 2000
Hoffstein–Silverman, 2016
Fluhrer, etc.: Even easier at
using invalid messages.

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

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

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.

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

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

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

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

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

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

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

ble guesses given a
decryption failure:
ited with some $x^i \operatorname{rev}(d)$.
orrelated with $x^{-i} d$.
correlated with $d \operatorname{rev}(d)$.

entally confirmed:
of $d \operatorname{rev}(d)$
ne decryption failures
to $a \operatorname{rev}(a)$.
o integers: $a \operatorname{rev}(a)$.

ot 2002 Gentry–Szydlo
n 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
$d \pm 1$, $d \pm x$, . . . , $d \pm x^{N-1}$;
$d \pm 2$, $d \pm 2x$, . . . , $d \pm 2x^{N-1}$;
$d \pm 3$, etc.

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

e.g. $3be$
all other
and $a =$

es given a
n failure:
some $x^i \operatorname{rev}(d)$.
with $x^{-i}d$.
with $d \operatorname{rev}(d)$.

nfirmed:
d)
ion failures
).
: $a \operatorname{rev}(a)$.

entry–Szydlo
ds a.

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

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

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

e.g. $3be+ad = \cdots$
all other coeffs in
and $a = \cdots + x^{47}$

v(d).

v(d).

s

dlo

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

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

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

e.g. $3be+ad = \cdots+390x^{47}$
all other coeffs in $[-389, 38$
and $a = \cdots + x^{478} + \cdots$.

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

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

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

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

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

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

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

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

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

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

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 $a = \cdots + x^{477} + \cdots$.

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

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

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

ll–Goldberg–Schneier,
ulmes–Joux, 2000
n–Silverman, 2016
etc.: Even easier attacks
valid messages.

r changes d to
$d \pm x, \ldots, d \pm x^{N-1}$;
$d \pm 2x, \ldots, d \pm 2x^{N-1}$;
tc.

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

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.

How to

Approac
constant

For each
generate
Use sign
that nob

rg–Schneier,
x, 2000
an, 2016
n easier attacks
ages.

d to
$d \pm x^{N-1}$;
, $d \pm 2x^{N-1}$;

$+\, ad$: adds
^{N-1}a;
$\pm 2x^{N-1}a$;

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.

How to handle inv

Approach 1: Tell u
constantly switch

For each new send
generate new publ
Use signatures to
that nobody else u

er,

tacks

1;

;

/−1;

ds

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.

How to handle invalid messa

Approach 1: Tell user to
constantly switch keys.

For each new sender,
generate new public key.
Use signatures to ensure
that nobody else uses key.

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.

How to handle invalid messages

Approach 1: Tell user to
constantly switch keys.

For each new sender,
generate new public key.
Use signatures to ensure
that nobody else uses key.

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.

How to handle invalid messages

Approach 1: Tell user to
constantly switch keys.

For each new sender,
generate new public key.
Use signatures to ensure
that nobody else uses key.

If user reuses a key:
Blame user for the attacks.

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.

How to handle invalid messages

Approach 1: Tell user to
constantly switch keys.

For each new sender,
generate new public key.
Use signatures to ensure
that nobody else uses key.

If user reuses a key:
Blame user for the attacks.

Approach 2: FO. Modify
encryption and decryption
to eliminate invalid messages.
Most submissions do this.

$e+ad = \cdots + 390x^{478} + \cdots$,

coeffs in $[-389, 389]$;

$\cdots + x^{478} + \cdots$.

$e + ad + ka =$

$90 + k)x^{478} + \cdots$.

ion fails for big k.

or smallest k that fails.

$e + ad + kxa$ also fail?

$a = \cdots + x^{478} + \cdots$,

$= \cdots + x^{477} + \cdots$.

, kx^3, etc.

ern of a coeffs.

How to handle invalid messages

Approach 1: Tell user to constantly switch keys.

For each new sender, generate new public key. Use signatures to ensure that nobody else uses key.

If user reuses a key: Blame user for the attacks.

Approach 2: FO. Modify encryption and decryption to eliminate invalid messages. Most submissions do this.

How to

Eliminat
not enou
using de
random

$\cdot + 390x^{478} + \cdots,$

$[-389, 389];$

$^{3} + \cdots.$

$ka =$

$^{78} + \cdots.$

r big k.

t k that fails.

kxa also fail?

$x^{478} + \cdots,$

$^{477} + \cdots.$

oeffs.

How to handle invalid messages

Approach 1: Tell user to constantly switch keys.

For each new sender, generate new public key. Use signatures to ensure that nobody else uses key.

If user reuses a key: Blame user for the attacks.

Approach 2: FO. Modify encryption and decryption to eliminate invalid messages. Most submissions do this.

How to handle dec

Eliminating invalic not enough: reme using decryption fa random valid mess

$^{8}+\cdots,$

9];

ails.

ail?

,

How to handle invalid messages

Approach 1: Tell user to constantly switch keys.

For each new sender, generate new public key. Use signatures to ensure that nobody else uses key.

If user reuses a key: Blame user for the attacks.

Approach 2: FO. Modify encryption and decryption to eliminate invalid messages. Most submissions do this.

How to handle decryption fa

Eliminating invalid messages not enough: remember atta using decryption failures for random valid messages.

How to handle invalid messages

Approach 1: Tell user to constantly switch keys.

For each new sender, generate new public key. Use signatures to ensure that nobody else uses key.

If user reuses a key: Blame user for the attacks.

Approach 2: FO. Modify encryption and decryption to eliminate invalid messages. Most submissions do this.

How to handle decryption failures

Eliminating invalid messages is not enough: remember attack using decryption failures for random valid messages.

How to handle invalid messages

Approach 1: Tell user to constantly switch keys.

For each new sender, generate new public key. Use signatures to ensure that nobody else uses key.

If user reuses a key: Blame user for the attacks.

Approach 2: FO. Modify encryption and decryption to eliminate invalid messages. Most submissions do this.

How to handle decryption failures

Eliminating invalid messages is not enough: remember attack using decryption failures for random valid messages.

NISTPQC encryption submissions vary in failure rates.

How to handle invalid messages

Approach 1: Tell user to constantly switch keys.

For each new sender, generate new public key. Use signatures to ensure that nobody else uses key.

If user reuses a key: Blame user for the attacks.

Approach 2: FO. Modify encryption and decryption to eliminate invalid messages. Most submissions do this.

How to handle decryption failures

Eliminating invalid messages is not enough: remember attack using decryption failures for random valid messages.

NISTPQC encryption submissions vary in failure rates.

LAC, NewHope, Round5, SABER: *conjectured* failure rate is small enough that generic *non-quantum* attacks provably maintain *some* security. (Security loss? Wrong conjecture? Quantum attacks?)

handle invalid messages

h 1: Tell user to
tly switch keys.

new sender,
e new public key.
atures to ensure
body else uses key.

euses a key:
ser for the attacks.

h 2: FO. Modify
on and decryption
nate invalid messages.
bmissions do this.

How to handle decryption failures

Eliminating invalid messages is not enough: remember attack using decryption failures for random valid messages.

NISTPQC encryption submissions vary in failure rates.

LAC, NewHope, Round5, SABER: *conjectured* failure rate is small enough that generic *non-quantum* attacks provably maintain *some* security. (Security loss? Wrong conjecture? Quantum attacks?)

ThreeBe
failure ra
generic
provably

alid messages

user to
keys.

der,
ic key.
ensure
uses key.

y:
e attacks.

Modify
cryption
d messages.
do this.

How to handle decryption failures

Eliminating invalid messages is not enough: remember attack using decryption failures for random valid messages.

NISTPQC encryption submissions vary in failure rates.

LAC, NewHope, Round5, SABER: *conjectured* failure rate is small enough that generic *non-quantum* attacks provably maintain *some* security. (Security loss? Wrong conjecture? Quantum attacks?)

ThreeBears: *conje*
failure rate is smal
generic *non-quant*
provably maintain

ages

s.

How to handle decryption failures

Eliminating invalid messages is not enough: remember attack using decryption failures for random valid messages.

NISTPQC encryption submissions vary in failure rates.

LAC, NewHope, Round5, SABER: *conjectured* failure rate is small enough that generic *non-quantum* attacks provably maintain *some* security. (Security loss? Wrong conjecture? Quantum attacks?)

ThreeBears: *conjectured*
failure rate is small enough t
generic *non-quantum* attack
provably maintain full securi

How to handle decryption failures

Eliminating invalid messages is not enough: remember attack using decryption failures for random valid messages.

NISTPQC encryption submissions vary in failure rates.

LAC, NewHope, Round5, SABER: *conjectured* failure rate is small enough that generic *non-quantum* attacks provably maintain *some* security. (Security loss? Wrong conjecture? Quantum attacks?)

ThreeBears: *conjectured* failure rate is small enough that generic *non-quantum* attacks provably maintain full security.

How to handle decryption failures

Eliminating invalid messages is not enough: remember attack using decryption failures for random valid messages.

NISTPQC encryption submissions vary in failure rates.

LAC, NewHope, Round5, SABER: *conjectured* failure rate is small enough that generic *non-quantum* attacks provably maintain *some* security. (Security loss? Wrong conjecture? Quantum attacks?)

ThreeBears: *conjectured* failure rate is small enough that generic *non-quantum* attacks provably maintain full security.

Frodo, Kyber: *proven* failure rate is small enough that generic *non-quantum* attacks provably maintain *some* security.

How to handle decryption failures

Eliminating invalid messages is not enough: remember attack using decryption failures for random valid messages.

NISTPQC encryption submissions vary in failure rates.

LAC, NewHope, Round5, SABER: *conjectured* failure rate is small enough that generic *non-quantum* attacks provably maintain *some* security. (Security loss? Wrong conjecture? Quantum attacks?)

ThreeBears: *conjectured* failure rate is small enough that generic *non-quantum* attacks provably maintain full security.

Frodo, Kyber: *proven* failure rate is small enough that generic *non-quantum* attacks provably maintain *some* security.

NTRU, NTRU Prime: proof of no decryption failures. Small impact on efficiency. Much simpler security review.

How to handle decryption failures

Eliminating invalid messages is not enough: remember attack using decryption failures for random valid messages.

NISTPQC encryption submissions vary in failure rates.

LAC, NewHope, Round5, SABER: *conjectured* failure rate is small enough that generic *non-quantum* attacks provably maintain *some* security. (Security loss? Wrong conjecture? Quantum attacks?)

ThreeBears: *conjectured* failure rate is small enough that generic *non-quantum* attacks provably maintain full security.

Frodo, Kyber: *proven* failure rate is small enough that generic *non-quantum* attacks provably maintain *some* security.

NTRU, NTRU Prime: proof of no decryption failures. Small impact on efficiency. Much simpler security review. Bad for publishing attack papers.

handle decryption failures

ing invalid messages is
ugh: remember attack
cryption failures for
valid messages.

QC encryption submissions
failure rates.

ewHope, Round5, SABER:
ured failure rate is small
that generic *non-quantum*
provably maintain *some*
(Security loss? Wrong
re? Quantum attacks?)

ThreeBears: *conjectured* failure rate is small enough that generic *non-quantum* attacks provably maintain full security.

Frodo, Kyber: *proven* failure rate is small enough that generic *non-quantum* attacks provably maintain *some* security.

NTRU, NTRU Prime:
proof of no decryption failures.
Small impact on efficiency.
Much simpler security review.
Bad for publishing attack papers.

Brute-fo

Attacker
$G = 3e/$
Can atta

cryption failures

l messages is
mber attack
ailures for
sages.

tion submissions
es.

Round5, SABER:
e rate is small
ric *non-quantum*
naintain *some*
loss? Wrong
tum attacks?)

ThreeBears: *conjectured*
failure rate is small enough that
generic *non-quantum* attacks
provably maintain full security.

Frodo, Kyber: *proven*
failure rate is small enough that
generic *non-quantum* attacks
provably maintain *some* security.

NTRU, NTRU Prime:
proof of no decryption failures.
Small impact on efficiency.
Much simpler security review.
Bad for publishing attack papers.

Brute-force search

Attacker is given p
$G = 3e/a$, ciphert
Can attacker find

ilures

s is
ck

ssions

ABER:
mall
antum
ome
ong
ks?)

ThreeBears: *conjectured* failure rate is small enough that generic *non-quantum* attacks provably maintain full security.

Frodo, Kyber: *proven* failure rate is small enough that generic *non-quantum* attacks provably maintain *some* security.

NTRU, NTRU Prime: proof of no decryption failures. Small impact on efficiency. Much simpler security review. Bad for publishing attack papers.

Brute-force search

Attacker is given public key $G = 3e/a$, ciphertext $C = b$

Can attacker find b?

ThreeBears: *conjectured* failure rate is small enough that generic *non-quantum* attacks provably maintain full security.

Frodo, Kyber: *proven* failure rate is small enough that generic *non-quantum* attacks provably maintain *some* security.

NTRU, NTRU Prime:
proof of no decryption failures.
Small impact on efficiency.
Much simpler security review.
Bad for publishing attack papers.

Brute-force search

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

ThreeBears: *conjectured* failure rate is small enough that generic *non-quantum* attacks provably maintain full security.

Frodo, Kyber: *proven* failure rate is small enough that generic *non-quantum* attacks provably maintain *some* security.

NTRU, NTRU Prime: proof of no decryption failures. Small impact on efficiency. Much simpler security review. Bad for publishing attack papers.

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!

ThreeBears: *conjectured* failure rate is small enough that generic *non-quantum* attacks provably maintain full security.

Frodo, Kyber: *proven* failure rate is small enough that generic *non-quantum* attacks provably maintain *some* security.

NTRU, NTRU Prime: proof of no decryption failures. Small impact on efficiency. Much simpler security review. Bad for publishing attack papers.

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.)

ThreeBears: *conjectured* failure rate is small enough that generic *non-quantum* attacks provably maintain full security.

Frodo, Kyber: *proven* failure rate is small enough that generic *non-quantum* attacks provably maintain *some* security.

NTRU, NTRU Prime: proof of no decryption failures. Small impact on efficiency. Much simpler security review. Bad for publishing attack papers.

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.

ears: *conjectured*
ate is small enough that
non-quantum attacks
maintain full security.

Kyber: *proven*
ate is small enough that
non-quantum attacks
maintain *some* security.

NTRU Prime:
no decryption failures.
npact on efficiency.
mpler security review.
publishing attack papers.

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.

Equivale

Secret k
secret ke
secret ke

ectured

ll enough that

um attacks

full security.

oven

ll enough that

um attacks

some security.

me:

tion failures.

fficiency.

rity review.

attack papers.

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

secret key $(xa, xe$

secret key $(x^2 a, x^2$

that
s
ty.

that
s
urity.

es.

v.
apers.

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 equivalen
secret key (xa, xe),
secret key (x^2a, x^2e), etc.

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.

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.

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.

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

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

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.

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

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

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

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

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!

rce search

r is given public key
/a, ciphertext $C = bG + d$.
acker find b?

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

s find two different
d? Unlikely. This would
o legitimate decryption.)

ch through choices of a.
$G/3$ is small, use (a, e)
pt. Advantage: can reuse
or 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

Write a
$a_1 =$ bot
$a_2 =$ ren

ublic key
ext $C = bG + d$.
b?

oices of b.
mall: done!

different
y. This would
e decryption.)

choices of a.
all, use (a, e)
tage: can reuse
phertexts.

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 $a_1 + a_2$
$a_1 =$ bottom $\lceil N/2$
$a_2 =$ remaining te

$G + d$.

e!

ould

on.)

f a.

$e)$

reuse

Equivalent keys

Secret key (a, e) is equivalent to
secret key (xa, xe),
secret key $(x^2 a, x^2 e)$, 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 o
$a_2 =$ remaining terms of a.

Equivalent keys

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

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 $e - (G/3)a_2 = (G/3)a_1$.

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
$H(f) = ([f_0 < 0], \ldots, [f_{k-1} < 0])$.

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 $N = 701$.

ent keys

ey (a, e) is equivalent to
ey (xa, xe),
ey (x^2a, x^2e), etc.

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

., $W = 467$:
$\binom{N}{W}2^W \approx 2^{1106.09}$;
$\binom{N}{W}2^W/N \approx 2^{1096.64}$.

., $W = 200$:
$\binom{N}{W}2^W \approx 2^{799.76}$;
$\binom{N}{W}2^W/N \approx 2^{790.31}$.

: 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 v

Given pu
Comput

s equivalent to

),

e), etc.

$2^W/N$ choices.

7:

$)2^W \approx 2^{1106.09}$;

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

0:

$)2^W \approx 2^{799.76}$;

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

re 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 NT

Given public key G

Compute $H = G/3$

nt to

oices.

1106.09;

1096.64.

$2^{799.76}$;

$2^{790.31}$.

ences!

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

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.

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.

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.

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.

attacks

as $a_1 + a_2$ where
ttom $\lceil N/2 \rceil$ terms of a,
naining terms of a.

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

ate all $H(-(G/3)a_2)$.
ate all $H((G/3)a_1)$.
or collisions.
out $3^{N/2}$ operations:
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.

$(e, a) \in$
$(Q, 0)$,
$(Qx, 0)$,
$\vdots$
$(Qx^{N-1}$
$(H, 1)$,
(xH, x),
$\vdots$
$(x^{N-1}H$
by a few

$_2$ where
$2\rceil$ terms of a,
rms of a.

$/3)a_1 + (G/3)a_2$
$(G/3)a_1$.
st certainly
$H((G/3)a_1)$ for
$\ldots, [f_{k-1} < 0])$.

$-(G/3)a_2)$.
$(G/3)a_1)$.
ns.
operations:
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 obta
$(Q, 0)$,
$(Qx, 0)$,
$\vdots$
$(Qx^{N-1}, 0)$,
$(H, 1)$,
(xH, x),
$\vdots$
$(x^{N-1}H, x^{N-1})$
by a few additions

f a,

$G/3)a_2$

y

) for

$< 0]$).

).

:

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, subtracti

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.

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

view of NTRU

ublic key $G = 3e/a$.

e $H = G/3 = e/a$ in R_Q.

obtained from

, x^{N-1}

additions, subtractions.

$_Q$ is obtained from

. . . , $x^{N-1}H$

additions, subtractions.

obtained from

$Qx^2, \ldots, Qx^{N-1}$,

. . . , $x^{N-1}H$

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

$H_0 + H_1x + \cdots + H_{N-1}x^{N-1}$.

$(e_0, e_1, .$

is obtain

$(Q, 0, \ldots$

$(0, Q, \ldots$

$\vdots$

$(0, 0, \ldots$

$(H_0, H_1,$

$(H_{N-1},$

$\vdots$

$(H_1, H_2,$

by a few

TRU

$G = 3e/a$.

$3 = e/a$ in R_Q.

from

, subtractions.

ned from

, subtractions.

from

x^{N-1},

, 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

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

$(e_0, e_1, \ldots, e_{N-1},$

is obtained from

$(Q, 0, \ldots, 0, 0, 0, \ldots$

$(0, Q, \ldots, 0, 0, 0, \ldots$

$\vdots$

$(0, 0, \ldots, Q, 0, 0, \ldots$

$(H_0, H_1, \ldots, H_{N-1}$

$(H_{N-1}, H_0, \ldots, H$

$\vdots$

$(H_1, H_2, \ldots, H_0, 0$

by a few additions

R_Q.

ions.

ions.

ions.

$(e, a) \in R^2$ 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
$H_0 + H_1x + \cdots + H_{N-1}x^{N-1}$.

$(e_0, e_1, \ldots, e_{N-1}, a_0, a_1, \ldots,$
is obtained from
$(Q, 0, \ldots, 0, 0, 0, \ldots, 0)$,
$(0, Q, \ldots, 0, 0, 0, \ldots, 0)$,
$\vdots$
$(0, 0, \ldots, Q, 0, 0, \ldots, 0)$,
$(H_0, H_1, \ldots, H_{N-1}, 1, 0, \ldots,$
$(H_{N-1}, H_0, \ldots, H_{N-2}, 0, 1, \ldots$
$\vdots$
$(H_1, H_2, \ldots, H_0, 0, 0, \ldots, 1)$
by a few additions, subtracti

$(e, a) \in R^2$ 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.

R^2 is obtained from

$, 0),$

$, x^{N-1})$

additions, subtractions.

as

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

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

$(e_0, e_1, \ldots$

is a surp

in lattice

$(Q, 0, \ldots$

ained from

, subtractions.

$H_{N-1}x^{N-1}$.

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

$(e_0, e_1, \ldots, e_{N-1},$
is a surprisingly sh
in lattice generate
$(Q, 0, \ldots, 0, 0, 0, .$

ions.

-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.

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

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

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

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

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

$\ldots, e_{N-1}, a_0, a_1, \ldots, a_{N-1})$
ned from
$., 0, 0, 0, \ldots, 0)$,
$., 0, 0, 0, \ldots, 0)$,

$, Q, 0, 0, \ldots, 0)$,
$\ldots, H_{N-1}, 1, 0, \ldots, 0)$,
$H_0, \ldots, H_{N-2}, 0, 1, \ldots, 0)$,

$\ldots, H_0, 0, 0, \ldots, 1)$
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.

Quotient

"Quotie
is the st

Alice ge
for smal
i.e., $aG/$

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

$\ldots, 0)$,

$\ldots, 0)$,

$\ldots, 0)$,

$_1, 1, 0, \ldots, 0)$,

$_{N-2}, 0, 1, \ldots, 0)$,

$, 0, \ldots, 1)$

, 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.

Quotient NTRU v

"Quotient NTRU"
is the structure we

Alice generates G
for small random
i.e., $aG/3 - e = 0$

$a_{N-1})$

$, 0),$

$\ldots, 0),$

ions.

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

Quotient NTRU vs. Product

"Quotient NTRU" (new nar
is the structure we've seen:

Alice generates $G = 3e/a$ in
for small random e, a:
i.e., $aG/3 - e = 0$ in R_Q.

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

Quotient NTRU vs. Product NTRU

"Quotient NTRU" (new name)
is the structure we've seen:

Alice generates $G = 3e/a$ in R_Q
for small random e, a:
i.e., $aG/3 - e = 0$ in R_Q.

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

Quotient NTRU vs. Product NTRU

“Quotient NTRU” (new name)
is the structure we’ve seen:

Alice generates $G = 3e/a$ in R_Q
for small random e, a:
i.e., $aG/3 - e = 0$ in R_Q.

Bob sends $C = bG + d$ in R_Q.
Alice computes aC in R_Q,
i.e., $3be + ad$ in R_Q.

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

Quotient NTRU vs. Product NTRU

"Quotient NTRU" (new name)
is the structure we've seen:

Alice generates $G = 3e/a$ in R_Q
for small random e, a:
i.e., $aG/3 - e = 0$ in R_Q.

Bob sends $C = bG + d$ in R_Q.
Alice computes aC in R_Q,
i.e., $3be + ad$ in R_Q.

Alice reconstructs $3be + ad$ in R,
using smallness of a, b, d, e.
Alice computes ad in R_3,
deduces d, deduces b.

$\ldots, e_{N-1}, a_0, a_1, \ldots, a_{N-1})$
risingly short vector
e generated by
$\ldots, 0, 0, 0, \ldots, 0)$ etc.

r searches for short vector
attice using (e.g.) BKZ.

peedups. e.g. rescaling:
attice to contain $(e, 10a)$
hosen $10\times$ larger than a.

: Describe search for
s a problem of finding
ce vector near a point;
t vector in a lattice.

Quotient NTRU vs. Product NTRU

"Quotient NTRU" (new name)
is the structure we've seen:

Alice generates $G = 3e/a$ in R_Q
for small random e, a:
i.e., $aG/3 - e = 0$ in R_Q.

Bob sends $C = bG + d$ in R_Q.
Alice computes aC in R_Q,
i.e., $3be + ad$ in R_Q.

Alice reconstructs $3be + ad$ in R,
using smallness of a, b, d, e.
Alice computes ad in R_3,
deduces d, deduces b.

"Produc
2010 Ly

Everyone
Alice ge
for smal

$a_0, a_1, \ldots, a_{N-1})$
ort vector
d by
$\ldots, 0)$ etc.

for short vector
g (e.g.) BKZ.

.g. rescaling:
ontain $(e, 10a)$
larger than a.

e search for
m of finding
near a point;
a lattice.

Quotient NTRU vs. Product NTRU

"Quotient NTRU" (new name)
is the structure we've seen:

Alice generates $G = 3e/a$ in R_Q
for small random e, a:
i.e., $aG/3 - e = 0$ in R_Q.

Bob sends $C = bG + d$ in R_Q.
Alice computes aC in R_Q,
i.e., $3be + ad$ in R_Q.

Alice reconstructs $3be + ad$ in R,
using smallness of a, b, d, e.
Alice computes ad in R_3,
deduces d, deduces b.

"Product NTRU"
2010 Lyubashevsk

Everyone knows ra
Alice generates A
for small random a

$a_{N-1})$

vector
KZ.

ng:
$10a)$
an a.

r
ng
nt;

Quotient NTRU vs. Product NTRU

"Quotient NTRU" (new name)
is the structure we've seen:

Alice generates $G = 3e/a$ in R_Q
for small random e, a:
i.e., $aG/3 - e = 0$ in R_Q.

Bob sends $C = bG + d$ in R_Q.
Alice computes aC in R_Q,
i.e., $3be + ad$ in R_Q.

Alice reconstructs $3be + ad$ in R,
using smallness of a, b, d, e.
Alice computes ad in R_3,
deduces d, deduces b.

"Product NTRU" (new nam
2010 Lyubashevsky–Peikert–

Everyone knows random $G \in$
Alice generates $A = aG + e$
for small random a, e.

Quotient NTRU vs. Product NTRU

“Quotient NTRU” (new name) is the structure we’ve seen:

Alice generates $G = 3e/a$ in R_Q for small random e, a: i.e., $aG/3 - e = 0$ in R_Q.

Bob sends $C = bG + d$ in R_Q. Alice computes aC in R_Q, i.e., $3be + ad$ in R_Q.

Alice reconstructs $3be + ad$ in R, using smallness of a, b, d, e. Alice computes ad in R_3, deduces d, deduces b.

“Product NTRU” (new name), 2010 Lyubashevsky–Peikert–Regev:

Everyone knows random $G \in R_Q$. Alice generates $A = aG + e$ in R_Q for small random a, e.

Quotient NTRU vs. Product NTRU

"Quotient NTRU" (new name) is the structure we've seen:

Alice generates $G = 3e/a$ in R_Q for small random e, a: i.e., $aG/3 - e = 0$ in R_Q.

Bob sends $C = bG + d$ in R_Q. Alice computes aC in R_Q, i.e., $3be + ad$ in R_Q.

Alice reconstructs $3be + ad$ in R, using smallness of a, b, d, e. Alice computes ad in R_3, deduces d, deduces b.

"Product NTRU" (new name), 2010 Lyubashevsky–Peikert–Regev:

Everyone knows random $G \in R_Q$. Alice generates $A = aG + e$ in R_Q for small random a, e.

Bob sends $B = bG + d$ in R_Q and $C = m + bA + c$ in R_Q where b, c, d are small and each coeff of m is 0 or $Q/2$.

Quotient NTRU vs. Product NTRU

"Quotient NTRU" (new name) is the structure we've seen:

Alice generates $G = 3e/a$ in R_Q for small random e, a: i.e., $aG/3 - e = 0$ in R_Q.

Bob sends $C = bG + d$ in R_Q. Alice computes aC in R_Q, i.e., $3be + ad$ in R_Q.

Alice reconstructs $3be + ad$ in R, using smallness of a, b, d, e. Alice computes ad in R_3, deduces d, deduces b.

"Product NTRU" (new name), 2010 Lyubashevsky–Peikert–Regev:

Everyone knows random $G \in R_Q$. Alice generates $A = aG + e$ in R_Q for small random a, e.

Bob sends $B = bG + d$ in R_Q and $C = m + bA + c$ in R_Q where b, c, d are small and each coeff of m is 0 or $Q/2$.

Alice computes $C - aB$ in R_Q, i.e., $m + be + c - ad$ in R_Q. Alice reconstructs m, using smallness of a, b, c, d, e.

t NTRU vs. Product NTRU

nt NTRU" (new name)
ructure we've seen:

nerates $G = 3e/a$ in R_Q
l random e, a:
/3 $- e = 0$ in R_Q.

ds $C = bG + d$ in R_Q.
mputes aC in R_Q,
$+ ad$ in R_Q.

constructs $3be + ad$ in R,
allness of a, b, d, e.
mputes ad in R_3,
d, deduces b.

"Product NTRU" (new name),
2010 Lyubashevsky–Peikert–Regev:

Everyone knows random $G \in R_Q$.
Alice generates $A = aG + e$ in R_Q
for small random a, e.

Bob sends $B = bG + d$ in R_Q
and $C = m + bA + c$ in R_Q
where b, c, d are small and
each coeff of m is 0 or $Q/2$.

Alice computes $C - aB$ in R_Q,
i.e., $m + be + c - ad$ in R_Q.
Alice reconstructs m,
using smallness of a, b, c, d, e.

Quotient
Ring-0L
Ring-LW

Product
Ring-LW
Ring-LW

s. Product NTRU

(new name)
e've seen:

$= 3e/a$ in R_Q
e, a:
in R_Q.

$G + d$ in R_Q.
C in R_Q,
R_Q.

$3be + ad$ in R,
a, b, d, e.
in R_3,
es b.

"Product NTRU" (new name),
2010 Lyubashevsky–Peikert–Regev:

Everyone knows random $G \in R_Q$.
Alice generates $A = aG + e$ in R_Q
for small random a, e.

Bob sends $B = bG + d$ in R_Q
and $C = m + bA + c$ in R_Q
where b, c, d are small and
each coeff of m is 0 or $Q/2$.

Alice computes $C - aB$ in R_Q,
i.e., $m + be + c - ad$ in R_Q.
Alice reconstructs m,
using smallness of a, b, c, d, e.

Quotient NTRU a
Ring-0LWE (attac
Ring-LWE_1 (attac

Product NTRU at
Ring-LWE_1 (attac
Ring-LWE_2 (attac

t NTRU

me)

R_Q

R_Q.

in R,

"Product NTRU" (new name),
2010 Lyubashevsky–Peikert–Regev:

Everyone knows random $G \in R_Q$.
Alice generates $A = aG + e$ in R_Q
for small random a, e.

Bob sends $B = bG + d$ in R_Q
and $C = m + bA + c$ in R_Q
where b, c, d are small and
each coeff of m is 0 or $Q/2$.

Alice computes $C - aB$ in R_Q,
i.e., $m + be + c - ad$ in R_Q.
Alice reconstructs m,
using smallness of a, b, c, d, e.

Quotient NTRU attack prob
Ring-0LWE (attack key) anc
Ring-LWE$_1$ (attack ciphertex

Product NTRU attack probl
Ring-LWE$_1$ (attack key) anc
Ring-LWE$_2$ (attack ciphertex

"Product NTRU" (new name), 2010 Lyubashevsky–Peikert–Regev:

Everyone knows random $G \in R_Q$. Alice generates $A = aG + e$ in R_Q for small random a, e.

Bob sends $B = bG + d$ in R_Q and $C = m + bA + c$ in R_Q where b, c, d are small and each coeff of m is 0 or $Q/2$.

Alice computes $C - aB$ in R_Q, i.e., $m + be + c - ad$ in R_Q. Alice reconstructs m, using smallness of a, b, c, d, e.

Quotient NTRU attack problems: Ring-0LWE (attack key) and Ring-LWE_1 (attack ciphertext).

Product NTRU attack problems: Ring-LWE_1 (attack key) and Ring-LWE_2 (attack ciphertext).

"Product NTRU" (new name), 2010 Lyubashevsky–Peikert–Regev:

Everyone knows random $G \in R_Q$. Alice generates $A = aG + e$ in R_Q for small random a, e.

Bob sends $B = bG + d$ in R_Q and $C = m + bA + c$ in R_Q where b, c, d are small and each coeff of m is 0 or $Q/2$.

Alice computes $C - aB$ in R_Q, i.e., $m + be + c - ad$ in R_Q. Alice reconstructs m, using smallness of a, b, c, d, e.

Quotient NTRU attack problems: Ring-0LWE (attack key) and Ring-LWE$_1$ (attack ciphertext).

Product NTRU attack problems: Ring-LWE$_1$ (attack key) and Ring-LWE$_2$ (attack ciphertext).

Disadantage of Quotient NTRU: maybe Ring-0LWE is a weakness.

"Product NTRU" (new name), 2010 Lyubashevsky–Peikert–Regev:

Everyone knows random $G \in R_Q$. Alice generates $A = aG + e$ in R_Q for small random a, e.

Bob sends $B = bG + d$ in R_Q and $C = m + bA + c$ in R_Q where b, c, d are small and each coeff of m is 0 or $Q/2$.

Alice computes $C - aB$ in R_Q, i.e., $m + be + c - ad$ in R_Q. Alice reconstructs m, using smallness of a, b, c, d, e.

Quotient NTRU attack problems: Ring-0LWE (attack key) and Ring-LWE$_1$ (attack ciphertext).

Product NTRU attack problems: Ring-LWE$_1$ (attack key) and Ring-LWE$_2$ (attack ciphertext).

Disadantage of Quotient NTRU: maybe Ring-0LWE is a weakness.

Disadantage of Product NTRU: maybe Ring-LWE$_2$ is a weakness.

"Product NTRU" (new name),
2010 Lyubashevsky–Peikert–Regev:

Everyone knows random $G \in R_Q$.
Alice generates $A = aG + e$ in R_Q
for small random a, e.

Bob sends $B = bG + d$ in R_Q
and $C = m + bA + c$ in R_Q
where b, c, d are small and
each coeff of m is 0 or $Q/2$.

Alice computes $C - aB$ in R_Q,
i.e., $m + be + c - ad$ in R_Q.
Alice reconstructs m,
using smallness of a, b, c, d, e.

Quotient NTRU attack problems:
Ring-0LWE (attack key) and
Ring-LWE$_1$ (attack ciphertext).

Product NTRU attack problems:
Ring-LWE$_1$ (attack key) and
Ring-LWE$_2$ (attack ciphertext).

Disadantage of Quotient NTRU:
maybe Ring-0LWE is a weakness.

Disadantage of Product NTRU:
maybe Ring-LWE$_2$ is a weakness.

Disadantage of Product NTRU:
extra m in $m + be + c - ad$
needs smaller (weaker) noise.

t NTRU" (new name),
ubashevsky–Peikert–Regev:

e knows random $G \in R_Q$.
nerates $A = aG + e$ in R_Q
l random a, e.

ds $B = bG + d$ in R_Q
$= m + bA + c$ in R_Q
$, c, d$ are small and
eff of m is 0 or $Q/2$.

mputes $C - aB$ in R_Q,
$- be + c - ad$ in R_Q.
constructs m,
nallness of a, b, c, d, e.

Quotient NTRU attack problems:
Ring-0LWE (attack key) and
Ring-LWE_1 (attack ciphertext).

Product NTRU attack problems:
Ring-LWE_1 (attack key) and
Ring-LWE_2 (attack ciphertext).

Disadantage of Quotient NTRU:
maybe Ring-0LWE is a weakness.

Disadantage of Product NTRU:
maybe Ring-LWE_2 is a weakness.

Disadantage of Product NTRU:
extra m in $m + be + c - ad$
needs smaller (weaker) noise.

2016 Pe
is at leas

(new name),
y–Peikert–Regev:

andom $G \in R_Q$.
$= aG + e$ in R_Q
a, e.

$G + d$ in R_Q
$+ c$ in R_Q
small and
0 or $Q/2$.

$- aB$ in R_Q,
ad in R_Q.
m,
a, b, c, d, e.

Quotient NTRU attack problems:
Ring-0LWE (attack key) and
Ring-LWE$_1$ (attack ciphertext).

Product NTRU attack problems:
Ring-LWE$_1$ (attack key) and
Ring-LWE$_2$ (attack ciphertext).

Disadantage of Quotient NTRU:
maybe Ring-0LWE is a weakness.

Disadantage of Product NTRU:
maybe Ring-LWE$_2$ is a weakness.

Disadantage of Product NTRU:
extra m in $m + be + c - ad$
needs smaller (weaker) noise.

2016 Peikert: "Ri
is at least as hard

ne),
-Regev:

$\in R_Q$.
in R_Q

R_Q

R_Q,
.

e.

Quotient NTRU attack problems:
Ring-0LWE (attack key) and
Ring-LWE$_1$ (attack ciphertext).

Product NTRU attack problems:
Ring-LWE$_1$ (attack key) and
Ring-LWE$_2$ (attack ciphertext).

Disadantage of Quotient NTRU:
maybe Ring-0LWE is a weakness.

Disadantage of Product NTRU:
maybe Ring-LWE$_2$ is a weakness.

Disadantage of Product NTRU:
extra m in $m + be + c - ad$
needs smaller (weaker) noise.

2016 Peikert: “Ring-LWE
is at least as hard as NTRU

Quotient NTRU attack problems:
Ring-0LWE (attack key) and
Ring-LWE$_1$ (attack ciphertext).

Product NTRU attack problems:
Ring-LWE$_1$ (attack key) and
Ring-LWE$_2$ (attack ciphertext).

Disadantage of Quotient NTRU:
maybe Ring-0LWE is a weakness.

Disadantage of Product NTRU:
maybe Ring-LWE$_2$ is a weakness.

Disadantage of Product NTRU:
extra m in $m + be + c - ad$
needs smaller (weaker) noise.

2016 Peikert: "Ring-LWE
is at least as hard as NTRU."

Quotient NTRU attack problems: Ring-0LWE (attack key) and Ring-LWE$_1$ (attack ciphertext).

Product NTRU attack problems: Ring-LWE$_1$ (attack key) and Ring-LWE$_2$ (attack ciphertext).

Disadantage of Quotient NTRU: maybe Ring-0LWE is a weakness.

Disadantage of Product NTRU: maybe Ring-LWE$_2$ is a weakness.

Disadantage of Product NTRU: extra m in $m + be + c - ad$ needs smaller (weaker) noise.

2016 Peikert: "Ring-LWE is at least as hard as NTRU."

What this theorem actually says is: you can solve (decisional) Ring-0LWE if you can solve (search) Ring-LWE$_1$ with considerably more noise.

Ring-LWE$_1$ with the same amount of noise (or slightly less!) could be weaker than Ring-0LWE. Also, Ring-LWE$_2$ could be weaker.

So Product NTRU could be less secure than Quotient NTRU.

t NTRU attack problems:
VE (attack key) and
VE$_1$ (attack ciphertext).

NTRU attack problems:
VE$_1$ (attack key) and
VE$_2$ (attack ciphertext).

tage of Quotient NTRU:
Ring-0LWE is a weakness.

tage of Product NTRU:
Ring-LWE$_2$ is a weakness.

tage of Product NTRU:
in $m + be + c - ad$
naller (weaker) noise.

2016 Peikert: "Ring-LWE is at least as hard as NTRU."

What this theorem actually says is: you can solve (decisional) Ring-0LWE if you can solve (search) Ring-LWE$_1$ with considerably more noise.

Ring-LWE$_1$ with the same amount of noise (or slightly less!) could be weaker than Ring-0LWE. Also, Ring-LWE$_2$ could be weaker.

So Product NTRU could be less secure than Quotient NTRU.

Disadva
need FO
not just

Quotient

ttack problems:
k key) and
k ciphertext).

tack problems:
k key) and
k ciphertext).

uotient NTRU:
E is a weakness.

oduct NTRU:
is a weakness.

oduct NTRU:
$e + c - ad$
aker) noise.

2016 Peikert: "Ring-LWE is at least as hard as NTRU."

What this theorem actually says is: you can solve (decisional) Ring-0LWE if you can solve (search) Ring-LWE_1 with considerably more noise.

Ring-LWE_1 with the same amount of noise (or slightly less!) could be weaker than Ring-0LWE. Also, Ring-LWE_2 could be weaker.

So Product NTRU could be less secure than Quotient NTRU.

Disadvantage of P
need FO derandon
not just FO reencr

Quotient NTRU is

lems:
d
xt).

ems:
d
xt).

TRU:
kness.

RU:
kness.

RU:
y
e.

2016 Peikert: "Ring-LWE is at least as hard as NTRU."

What this theorem actually says is: you can solve (decisional) Ring-0LWE if you can solve (search) Ring-LWE_1 with considerably more noise.

Ring-LWE_1 with the same amount of noise (or slightly less!) could be weaker than Ring-0LWE. Also, Ring-LWE_2 could be weaker.

So Product NTRU could be less secure than Quotient NTRU.

Disadvantage of Product NT
need FO derandomization,
not just FO reencryption.

Quotient NTRU is determin

2016 Peikert: "Ring-LWE is at least as hard as NTRU."

What this theorem actually says is: you can solve (decisional) Ring-0LWE if you can solve (search) Ring-LWE_1 with considerably more noise.

Ring-LWE_1 with the same amount of noise (or slightly less!) could be weaker than Ring-0LWE. Also, Ring-LWE_2 could be weaker.

So Product NTRU could be less secure than Quotient NTRU.

Disadvantage of Product NTRU: need FO derandomization, not just FO reencryption.

Quotient NTRU is deterministic.

2016 Peikert: "Ring-LWE is at least as hard as NTRU."

What this theorem actually says is: you can solve (decisional) Ring-0LWE if you can solve (search) Ring-LWE$_1$ with considerably more noise.

Ring-LWE$_1$ with the same amount of noise (or slightly less!) could be weaker than Ring-0LWE. Also, Ring-LWE$_2$ could be weaker.

So Product NTRU could be less secure than Quotient NTRU.

Disadvantage of Product NTRU: need FO derandomization, not just FO reencryption.

Quotient NTRU is deterministic.

Why this (maybe) matters: 2019 Bindel–Hamburg–Hövelmanns–Hülsing–Persichetti proves tight QROM IND-CCA2 security for one-way deterministic systems.

With FO derandomization, all known proofs lose tightness or make stronger assumptions than one-wayness.

ikert: "Ring-LWE
st as hard as NTRU."

is theorem actually says
can solve (decisional)
WE if you can solve
Ring-LWE$_1$ with
ably more noise.

WE$_1$ with the same amount
(or slightly less!) could
er than Ring-0LWE. Also,
WE$_2$ could be weaker.

uct NTRU could be less
han Quotient NTRU.

Disadvantage of Product NTRU: need FO derandomization, not just FO reencryption.

Quotient NTRU is deterministic.

Why this (maybe) matters: 2019 Bindel–Hamburg–Hövelmanns–Hülsing–Persichetti proves tight QROM IND-CCA2 security for one-way deterministic systems.

With FO derandomization, all known proofs lose tightness or make stronger assumptions than one-wayness.

Disadva
NTRU:
encapsul

ng-LWE
as NTRU."

n actually says
(decisional)
can solve
E_1 with
noise.

he same amount
y less!) could
ng-0LWE. Also,
be weaker.

J could be less
ent NTRU.

Disadvantage of Product NTRU: need FO derandomization, not just FO reencryption.

Quotient NTRU is deterministic.

Why this (maybe) matters: 2019 Bindel–Hamburg–Hövelmanns–Hülsing–Persichetti proves tight QROM IND-CCA2 security for one-way deterministic systems.

With FO derandomization, all known proofs lose tightness or make stronger assumptions than one-wayness.

Disadvantage of P
NTRU: more mult
encapsulation and

.”
says
)
mount
uld
Also,
.
less
.

Disadvantage of Product NTRU: need FO derandomization, not just FO reencryption.

Quotient NTRU is deterministic.

Why this (maybe) matters: 2019 Bindel–Hamburg–Hövelmanns–Hülsing–Persichetti proves tight QROM IND-CCA2 security for one-way deterministic systems.

With FO derandomization, all known proofs lose tightness or make stronger assumptions than one-wayness.

Disadvantage of Product NTRU: more multiplications encapsulation and decapsula

Disadvantage of Product NTRU: need FO derandomization, not just FO reencryption.

Quotient NTRU is deterministic.

Why this (maybe) matters: 2019 Bindel–Hamburg–Hövelmanns–Hülsing–Persichetti proves tight QROM IND-CCA2 security for one-way deterministic systems.

With FO derandomization, all known proofs lose tightness or make stronger assumptions than one-wayness.

Disadvantage of Product NTRU: more multiplications in encapsulation and decapsulation.

Disadvantage of Product NTRU: need FO derandomization, not just FO reencryption.

Quotient NTRU is deterministic.

Why this (maybe) matters: 2019 Bindel–Hamburg–Hövelmanns–Hülsing–Persichetti proves tight QROM IND-CCA2 security for one-way deterministic systems.

With FO derandomization, all known proofs lose tightness or make stronger assumptions than one-wayness.

Disadvantage of Product NTRU: more multiplications in encapsulation and decapsulation.

Disadvantage of Quotient NTRU: divisions in key generation are much more expensive than mults.

Disadvantage of Product NTRU: need FO derandomization, not just FO reencryption.

Quotient NTRU is deterministic.

Why this (maybe) matters: 2019 Bindel–Hamburg–Hövelmanns–Hülsing–Persichetti proves tight QROM IND-CCA2 security for one-way deterministic systems.

With FO derandomization, all known proofs lose tightness or make stronger assumptions than one-wayness.

Disadvantage of Product NTRU: more multiplications in encapsulation and decapsulation.

Disadvantage of Quotient NTRU: divisions in key generation are much more expensive than mults.

Fix: if you need to generate many keys, use Montgomery's trick to replace D divisions with 1 division $+ 4(D-1)$ mults.

Disadvantage of Product NTRU: need FO derandomization, not just FO reencryption.

Quotient NTRU is deterministic.

Why this (maybe) matters: 2019 Bindel–Hamburg–Hövelmanns–Hülsing–Persichetti proves tight QROM IND-CCA2 security for one-way deterministic systems.

With FO derandomization, all known proofs lose tightness or make stronger assumptions than one-wayness.

Disadvantage of Product NTRU: more multiplications in encapsulation and decapsulation.

Disadvantage of Quotient NTRU: divisions in key generation are much more expensive than mults.

Fix: if you need to generate many keys, use Montgomery's trick to replace D divisions with 1 division $+ 4(D-1)$ mults.

2020 Bernstein–Brumley–Chen–Tuveri showed how to integrate this into OpenSSL and TLS 1.3.

ntage of Product NTRU:
) derandomization,
FO reencryption.

t NTRU is deterministic.

s (maybe) matters: 2019
Hamburg–Hövelmanns–
-Persichetti proves tight
IND-CCA2 security for
deterministic systems.

) derandomization,
n proofs lose tightness
stronger assumptions
e-wayness.

Disadvantage of Product NTRU: more multiplications in encapsulation and decapsulation.

Disadvantage of Quotient NTRU: divisions in key generation are much more expensive than mults.

Fix: if you need to generate many keys, use Montgomery's trick to replace D divisions with 1 division $+ 4(D-1)$ mults.

2020 Bernstein–Brumley–Chen–Tuveri showed how to integrate this into OpenSSL and TLS 1.3.

Disadvar
double-s

Product NTRU:
mization,
ryption.

s deterministic.

matters: 2019
Hövelmanns–
ti proves tight
2 security for
stic systems.

mization,
ose tightness
assumptions

Disadvantage of Product NTRU: more multiplications in encapsulation and decapsulation.

Disadvantage of Quotient NTRU: divisions in key generation are much more expensive than mults.

Fix: if you need to generate many keys, use Montgomery's trick to replace D divisions with 1 division $+ 4(D-1)$ mults.

2020 Bernstein–Brumley–Chen–Tuveri showed how to integrate this into OpenSSL and TLS 1.3.

Disadvantage of P
double-size ciphert

TRU:

istic.

2019

ns–

ight

for

ms.

ess

ns

Disadvantage of Product NTRU: more multiplications in encapsulation and decapsulation.

Disadvantage of Quotient NTRU: divisions in key generation are much more expensive than mults.

Fix: if you need to generate many keys, use Montgomery's trick to replace D divisions with 1 division $+ 4(D-1)$ mults.

2020 Bernstein–Brumley–Chen–Tuveri showed how to integrate this into OpenSSL and TLS 1.3.

Disadvantage of Product NT

double-size ciphertexts.

Disadvantage of Product NTRU: more multiplications in encapsulation and decapsulation.

Disadvantage of Quotient NTRU: divisions in key generation are much more expensive than mults.

Fix: if you need to generate many keys, use Montgomery's trick to replace D divisions with 1 division $+ 4(D - 1)$ mults.

2020 Bernstein–Brumley–Chen–Tuveri showed how to integrate this into OpenSSL and TLS 1.3.

Disadvantage of Product NTRU: double-size ciphertexts.

Disadvantage of Product NTRU: more multiplications in encapsulation and decapsulation.

Disadvantage of Quotient NTRU: divisions in key generation are much more expensive than mults.

Fix: if you need to generate many keys, use Montgomery's trick to replace D divisions with 1 division $+ 4(D-1)$ mults.

2020 Bernstein–Brumley–Chen–Tuveri showed how to integrate this into OpenSSL and TLS 1.3.

Disadvantage of Product NTRU: double-size ciphertexts.

Fix: 2012 Ding compressed ciphertexts to $\approx 1/2$ size.

Disadvantage of Product NTRU: more multiplications in encapsulation and decapsulation.

Disadvantage of Quotient NTRU: divisions in key generation are much more expensive than mults.

Fix: if you need to generate many keys, use Montgomery's trick to replace D divisions with 1 division $+ 4(D-1)$ mults.

2020 Bernstein–Brumley–Chen–Tuveri showed how to integrate this into OpenSSL and TLS 1.3.

Disadvantage of Product NTRU: double-size ciphertexts.

Fix: 2012 Ding compressed ciphertexts to $\approx 1/2$ size.

Bad news: Ding patented☢ this. I'm skeptical of the idea that tweaks will avoid the patent.

Disadvantage of Product NTRU: more multiplications in encapsulation and decapsulation.

Disadvantage of Quotient NTRU: divisions in key generation are much more expensive than mults.

Fix: if you need to generate many keys, use Montgomery's trick to replace D divisions with 1 division $+ 4(D - 1)$ mults.

2020 Bernstein–Brumley–Chen–Tuveri showed how to integrate this into OpenSSL and TLS 1.3.

Disadvantage of Product NTRU: double-size ciphertexts.

Fix: 2012 Ding compressed ciphertexts to $\approx 1/2$ size.

Bad news: Ding patented☢ this. I'm skeptical of the idea that tweaks will avoid the patent.

2014 Peikert: "As compared with the previous most efficient ring-LWE cryptosystems and KEMs, the new reconciliation mechanism reduces the ciphertext length by nearly a factor of two". No. Minor Ding tweak, same length.

ntage of Product
more multiplications in
lation and decapsulation.

ntage of Quotient NTRU:
; in key generation are
ore expensive than mults.

ou need to generate many
e Montgomery's trick
ce D divisions with
n $+ 4(D - 1)$ mults.

rnstein–Brumley–Chen–
howed how to integrate
OpenSSL and TLS 1.3.

Disadvantage of Product NTRU:
double-size ciphertexts.

Fix: 2012 Ding compressed
ciphertexts to $\approx 1/2$ size.

Bad news: Ding patented☢ this.
I'm skeptical of the idea that
tweaks will avoid the patent.

2014 Peikert: "As compared with
the previous most efficient ring-
LWE cryptosystems and KEMs,
the new reconciliation mechanism
reduces the ciphertext length
by nearly a factor of two". No.
Minor Ding tweak, same length.

Disadvar
2010.02
patent☢
covers P

roduct
iplications in
decapsulation.

Quotient NTRU:
neration are
sive than mults.

o generate many
mery's trick
ons with
− 1) mults.

rumley–Chen–
v to integrate
and TLS 1.3.

Disadvantage of Product NTRU: double-size ciphertexts.

Fix: 2012 Ding compressed ciphertexts to ≈1/2 size.

Bad news: Ding patented☢ this. I'm skeptical of the idea that tweaks will avoid the patent.

2014 Peikert: "As compared with the previous most efficient ring-LWE cryptosystems and KEMs, the new reconciliation mechanism reduces the ciphertext length by nearly a factor of two". No. Minor Ding tweak, same length.

Disadvantage of P

2010.02 Gaborit–A
patent☢, before L
covers Product N

in
tion.
TRU:
re
mults.
many
k
.
en–
rate
1.3.

Disadvantage of Product NTRU:
double-size ciphertexts.

Fix: 2012 Ding compressed
ciphertexts to $\approx 1/2$ size.

Bad news: Ding patented☢ this.
I'm skeptical of the idea that
tweaks will avoid the patent.

2014 Peikert: "As compared with
the previous most efficient ring-
LWE cryptosystems and KEMs,
the new reconciliation mechanism
reduces the ciphertext length
by nearly a factor of two". No.
Minor Ding tweak, same length.

Disadvantage of Product NT

2010.02 Gaborit–Aguilar Me
patent☢, before LPR public
covers Product NTRU.

Disadvantage of Product NTRU: double-size ciphertexts.

Fix: 2012 Ding compressed ciphertexts to $\approx 1/2$ size.

Bad news: Ding patented this. I'm skeptical of the idea that tweaks will avoid the patent.

2014 Peikert: "As compared with the previous most efficient ring-LWE cryptosystems and KEMs, the new reconciliation mechanism reduces the ciphertext length by nearly a factor of two". No. Minor Ding tweak, same length.

Disadvantage of Product NTRU:

2010.02 Gaborit–Aguilar Melchor patent, before LPR publication, covers Product NTRU.

Disadvantage of Product NTRU: double-size ciphertexts.

Fix: 2012 Ding compressed ciphertexts to ≈1/2 size.

Bad news: Ding patented this. I'm skeptical of the idea that tweaks will avoid the patent.

2014 Peikert: "As compared with the previous most efficient ring-LWE cryptosystems and KEMs, the new reconciliation mechanism reduces the ciphertext length by nearly a factor of two". No. Minor Ding tweak, same length.

Disadvantage of Product NTRU:

2010.02 Gaborit–Aguilar Melchor patent, before LPR publication, covers Product NTRU.

Rumors of patent-buyout offers have not shown results (yet?).

Disadvantage of Product NTRU: double-size ciphertexts.

Fix: 2012 Ding compressed ciphertexts to $\approx 1/2$ size.

Bad news: Ding patented☢ this. I'm skeptical of the idea that tweaks will avoid the patent.

2014 Peikert: "As compared with the previous most efficient ring-LWE cryptosystems and KEMs, the new reconciliation mechanism reduces the ciphertext length by nearly a factor of two". No. Minor Ding tweak, same length.

Disadvantage of Product NTRU:

2010.02 Gaborit–Aguilar Melchor patent☢, before LPR publication, covers Product NTRU.

Rumors of patent-buyout offers have not shown results (yet?).

A British law firm named Keltie, not saying who it is representing, has tried to kill the patent, and so far has failed.

To watch Keltie's ongoing appeal: `https://tinyurl.com/y4e66y6b` Some interesting documents.

ntage of Product NTRU:
ize ciphertexts.

.2 Ding compressed
xts to ≈1/2 size.

vs: Ding patented☢ this.
tical of the idea that
vill avoid the patent.

ikert: "As compared with
ious most efficient ring-
yptosystems and KEMs,
reconciliation mechanism
the ciphertext length
y a factor of two". No.
ing tweak, same length.

Disadvantage of Product NTRU:

2010.02 Gaborit–Aguilar Melchor patent☢, before LPR publication, covers Product NTRU.

Rumors of patent-buyout offers have not shown results (yet?).

A British law firm named Keltie, not saying who it is representing, has tried to kill the patent, and so far has failed.

To watch Keltie's ongoing appeal:
https://tinyurl.com/y4e66y6b
Some interesting documents.

Disadvar
NTRU:

Product
years of
("strong
successf

roduct NTRU:
texts.

mpressed
/2 size.

atented☢ this.
e idea that
the patent.

compared with
efficient ring-
s and KEMs,
tion mechanism
text length
of two". No.
, same length.

Disadvantage of Product NTRU:

2010.02 Gaborit–Aguilar Melchor patent☢, before LPR publication, covers Product NTRU.

Rumors of patent-buyout offers have not shown results (yet?).

A British law firm named Keltie, not saying who it is representing, has tried to kill the patent, and so far has failed.

To watch Keltie's ongoing appeal:
https://tinyurl.com/y4e66y6b
Some interesting documents.

Disadvantage (?)
NTRU: much less

Product NTRU is
years of security e
("strong security g
successfully attrac

TRU:

this.
t
.
l with
ing-
Ms,
anism
h
No.
gth.

Disadvantage of Product NTRU:

2010.02 Gaborit–Aguilar Melchor patent☢, before LPR publication, covers Product NTRU.

Rumors of patent-buyout offers have not shown results (yet?).

A British law firm named Keltie, not saying who it is representing, has tried to kill the patent, and so far has failed.

To watch Keltie's ongoing appeal:
https://tinyurl.com/y4e66y6b
Some interesting documents.

Disadvantage (?) of Quotien
NTRU: much less marketing

Product NTRU is backed by
years of security exaggeratio
("strong security guarantees
successfully attracting intere

Disadvantage of Product NTRU:

2010.02 Gaborit–Aguilar Melchor patent☢, before LPR publication, covers Product NTRU.

Rumors of patent-buyout offers have not shown results (yet?).

A British law firm named Keltie, not saying who it is representing, has tried to kill the patent, and so far has failed.

To watch Keltie's ongoing appeal:
https://tinyurl.com/y4e66y6b
Some interesting documents.

Disadvantage (?) of Quotient NTRU: much less marketing.

Product NTRU is backed by 10 years of security exaggeration ("strong security guarantees"), successfully attracting interest.

Disadvantage of Product NTRU:

2010.02 Gaborit–Aguilar Melchor patent☢, before LPR publication, covers Product NTRU.

Rumors of patent-buyout offers have not shown results (yet?).

A British law firm named Keltie, not saying who it is representing, has tried to kill the patent, and so far has failed.

To watch Keltie's ongoing appeal:
`https://tinyurl.com/y4e66y6b`
Some interesting documents.

Disadvantage (?) of Quotient NTRU: much less marketing.

Product NTRU is backed by 10 years of security exaggeration ("strong security guarantees"), successfully attracting interest.

Product NTRU submissions: Frodo, Kyber, LAC, NewHope, NTRU LPRime, Round5, SABER, ThreeBears. (All compressed.)

Quotient NTRU submissions: NTRU, Streamlined NTRU Prime.