Lattice-based public-key cryptosystems

D. J. Bernstein

NIST post-quantum competition:
82 submissions in first round,
from hundreds of people.
− 13 submissions that NIST
declared incomplete or improper.
− 5 withdrawn submissions.
− 3 merged submissions.

22 signature-system submissions.
5 lattice-based: Dilithium;
DRS (broken); FALCON☢;
pqNTRUSign☢; qTESLA.

47 encryption-system submissions.
20 lattice-based:
Compact LWE☢ (broken);
Ding☢; EMBLEM; Frodo;
HILA5 (CCA broken); KCL☢;
KINDI; Kyber; LAC; LIMA;
Lizard☢; LOTUS; NewHope;
NTRUEncrypt; NTRU HRSS;
NTRU Prime; Odd Manhattan;
Round2☢; SABER; Titanium.

Lattice-based public-key cryptosystems

D. J. Bernstein

NIST post-quantum competition:
82 submissions in first round,
from hundreds of people.
– 13 submissions that NIST declared incomplete or improper.
– 5 withdrawn submissions.
– 3 merged submissions.

22 signature-system submissions.
5 lattice-based: Dilithium;
DRS (broken); FALCON☢;
pqNTRUSign☢; qTESLA.

47 encryption-system submissions.
20 lattice-based:
Compact LWE☢ (broken);
Ding☢; EMBLEM; Frodo;
HILA5 (CCA broken); KCL☢;
KINDI; Kyber; LAC; LIMA;
Lizard☢; LOTUS; NewHope;
NTRUEncrypt; NTRU HRSS;
NTRU Prime; Odd Manhattan;
Round2☢; SABER; Titanium.

☢: submitter claims patent on this submission. Warning: even without ☢, submission could be covered by other patents!

pased
ey cryptosystems

ernstein

ost-quantum competition:
nissions in first round,
ndreds of people.
bmissions that NIST
 incomplete or improper.
ndrawn submissions.
rged submissions.

ture-system submissions.
-based: Dilithium;
roken); FALCON☢;
USign☢; qTESLA.

47 encryption-system submissions.
20 lattice-based:
Compact LWE☢ (broken);
Ding☢; EMBLEM; Frodo;
HILA5 (CCA broken); KCL☢;
KINDI; Kyber; LAC; LIMA;
Lizard☢; LOTUS; NewHope;
NTRUEncrypt; NTRU HRSS;
NTRU Prime; Odd Manhattan;
Round2☢; SABER; Titanium.

☢: submitter claims patent on this submission. Warning: even without ☢, submission could be covered by other patents!

First ser
encrypti
Hoffstei

Announ
at Crypt
Patentec

ystems

---

m competition:
first round,
people.
that NIST
te or improper.
bmissions.
issions.

m submissions.
ilithium;
LCON☢;
TESLA.

47 encryption-system submissions.
20 lattice-based:
Compact LWE☢ (broken);
Ding☢; EMBLEM; Frodo;
HILA5 (CCA broken); KCL☢;
KINDI; Kyber; LAC; LIMA;
Lizard☢; LOTUS; NewHope;
NTRUEncrypt; NTRU HRSS;
NTRU Prime; Odd Manhattan;
Round2☢; SABER; Titanium.

☢: submitter claims patent on this submission. Warning: even without ☢, submission could be covered by other patents!

First serious lattic
encryption system
Hoffstein–Pipher–S

Announced 20 Au
at Crypto 1996 ru
Patented until 201

ition:
d,
-
oper.
ions.

47 encryption-system submissions.
20 lattice-based:
Compact LWE☢ (broken);
Ding☢; EMBLEM; Frodo;
HILA5 (CCA broken); KCL☢;
KINDI; Kyber; LAC; LIMA;
Lizard☢; LOTUS; NewHope;
NTRUEncrypt; NTRU HRSS;
NTRU Prime; Odd Manhattan;
Round2☢; SABER; Titanium.

☢: submitter claims patent on this submission. Warning: even without ☢, submission could be covered by other patents!

First serious lattice-based encryption system: NTRU f
Hoffstein–Pipher–Silverman.

Announced 20 August 1996
at Crypto 1996 rump sessior
Patented until 2017.

47 encryption-system submissions.
20 lattice-based:
Compact LWE☢ (broken);
Ding☢; EMBLEM; Frodo;
HILA5 (CCA broken); KCL☢;
KINDI; Kyber; LAC; LIMA;
Lizard☢; LOTUS; NewHope;
NTRUEncrypt; NTRU HRSS;
NTRU Prime; Odd Manhattan;
Round2☢; SABER; Titanium.

☢: submitter claims patent on this submission. Warning: even without ☢, submission could be covered by other patents!

First serious lattice-based encryption system: NTRU from Hoffstein–Pipher–Silverman.

Announced 20 August 1996 at Crypto 1996 rump session. Patented until 2017.

47 encryption-system submissions.
20 lattice-based:
Compact LWE☢ (broken);
Ding☢; EMBLEM; Frodo;
HILA5 (CCA broken); KCL☢;
KINDI; Kyber; LAC; LIMA;
Lizard☢; LOTUS; NewHope;
NTRUEncrypt; NTRU HRSS;
NTRU Prime; Odd Manhattan;
Round2☢; SABER; Titanium.

☢: submitter claims patent on this submission. Warning: even without ☢, submission could be covered by other patents!

First serious lattice-based encryption system: NTRU from Hoffstein–Pipher–Silverman.

Announced 20 August 1996 at Crypto 1996 rump session. Patented until 2017.

First version of NTRU paper, handed out at Crypto 1996, finally put online in 2016:
web.securityinnovation.com
/hubfs/files/ntru-orig.pdf

47 encryption-system submissions.
20 lattice-based:
Compact LWE☢ (broken);
Ding☢; EMBLEM; Frodo;
HILA5 (CCA broken); KCL☢;
KINDI; Kyber; LAC; LIMA;
Lizard☢; LOTUS; NewHope;
NTRUEncrypt; NTRU HRSS;
NTRU Prime; Odd Manhattan;
Round2☢; SABER; Titanium.

☢: submitter claims patent on this submission. Warning: even without ☢, submission could be covered by other patents!

First serious lattice-based encryption system: NTRU from Hoffstein–Pipher–Silverman.

Announced 20 August 1996 at Crypto 1996 rump session. Patented until 2017.

First version of NTRU paper, handed out at Crypto 1996, finally put online in 2016:
`web.securityinnovation.com/hubfs/files/ntru-orig.pdf`

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

ption-system submissions.
e-based:
t LWE☢ (broken);
EMBLEM; Frodo;
CCA broken); KCL☢;
Kyber; LAC; LIMA;
; LOTUS; NewHope;
ncrypt; NTRU HRSS;
Prime; Odd Manhattan;
☢; SABER; Titanium.

mitter claims patent on
mission. Warning: even
☢, submission could be
by other patents!

First serious lattice-based encryption system: NTRU from Hoffstein–Pipher–Silverman.

Announced 20 August 1996 at Crypto 1996 rump session. Patented until 2017.

First version of NTRU paper, handed out at Crypto 1996, finally put online in 2016:
web.securityinnovation.com
/hubfs/files/ntru-orig.pdf

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

1996 pa
attack p
problem
applied l
to attac

em submissions.

(broken);
l; Frodo;
en); KCL☢;
C; LIMA;
NewHope;
TRU HRSS;
d Manhattan;
R; Titanium.

ms patent on
Varning: even
ssion could be
patents!

First serious lattice-based encryption system: NTRU from Hoffstein–Pipher–Silverman.

Announced 20 August 1996 at Crypto 1996 rump session. Patented until 2017.

First version of NTRU paper, handed out at Crypto 1996, finally put online in 2016: `web.securityinnovation.com/hubfs/files/ntru-orig.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 lattic

ssions.

;

e;

S;

an;

m.

on

ven

d be

First serious lattice-based encryption system: NTRU from Hoffstein–Pipher–Silverman.

Announced 20 August 1996 at Crypto 1996 rump session. Patented until 2017.

First version of NTRU paper, handed out at Crypto 1996, finally put online in 2016: `web.securityinnovation.com/hubfs/files/ntru-orig.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

First serious lattice-based encryption system: NTRU from Hoffstein–Pipher–Silverman.

Announced 20 August 1996 at Crypto 1996 rump session. Patented until 2017.

First version of NTRU paper, handed out at Crypto 1996, finally put online in 2016:
web.securityinnovation.com/hubfs/files/ntru-orig.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.

First serious lattice-based encryption system: NTRU from Hoffstein–Pipher–Silverman.

Announced 20 August 1996 at Crypto 1996 rump session. Patented until 2017.

First version of NTRU paper, handed out at Crypto 1996, finally put online in 2016:
web.securityinnovation.com
/hubfs/files/ntru-orig.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.

Coppersmith–Shamir, Eurocrypt 1997: better conversion + better attacks than LLL. Quantitative impact? Unclear.

First serious lattice-based encryption system: NTRU from Hoffstein–Pipher–Silverman.

Announced 20 August 1996 at Crypto 1996 rump session. Patented until 2017.

First version of NTRU paper, handed out at Crypto 1996, finally put online in 2016: web.securityinnovation.com/hubfs/files/ntru-orig.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.

Coppersmith–Shamir, Eurocrypt 1997: better conversion + better attacks than LLL. Quantitative impact? Unclear.

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

ious lattice-based
on system: NTRU from
n–Pipher–Silverman.

ced 20 August 1996
to 1996 rump session.
d until 2017.

rsion of NTRU paper,
out at Crypto 1996,
ut online in 2016:
curityinnovation.com
/files/ntru-orig.pdf

d 104-byte public keys
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.

Coppersmith–Shamir, Eurocrypt 1997: better conversion + better attacks than LLL. Quantitative impact? Unclear.

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

Let's try

Debian:
Fedora:
Source:
Web: sa

Sage is
+ many
+ a few

```
sage: 1
1000000
sage: fa
3172135
sage:
```

e-based
: NTRU from
Silverman.

gust 1996
mp session.
.7.

TRU paper,
ypto 1996,
n 2016:
novation.com
tru-orig.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.

Coppersmith–Shamir, Eurocrypt 1997: better conversion + better attacks than LLL. Quantitative impact? Unclear.

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

Let's try NTRU or

Debian: apt inst
Fedora: yum inst
Source: www.sage
Web: sagecell.s

Sage is Python 2
+ many math libra
+ a few syntax dif

```
sage: 10^6 # pow
1000000
sage: factor(314
317213509 * 9903
sage:
```

rom

n.

r,

.com

.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.

Coppersmith–Shamir, Eurocrypt 1997: better conversion + better attacks than LLL. Quantitative impact? Unclear.

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

Let's try NTRU on the comp

Debian: `apt install sage`

Fedora: `yum install sager`

Source: www.sagemath.org

Web: sagecell.sagemath

Sage is Python 2
+ many math libraries
+ a few syntax differences:

```
sage: 10^6 # power, not x
1000000
sage: factor(314159265358
317213509 * 990371647
sage:
```

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.

Coppersmith–Shamir, Eurocrypt 1997: better conversion + better attacks than LLL. Quantitative impact? Unclear.

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

Let's try NTRU on the computer.

Debian: `apt install sagemath`
Fedora: `yum install sagemath`
Source: `www.sagemath.org`
Web: `sagecell.sagemath.org`

Sage is Python 2
+ many math libraries
+ a few syntax differences:

```
sage: 10^6 # power, not xor
1000000
sage: factor(314159265358979323)
317213509 * 990371647
sage:
```

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

mith–Shamir, Eurocrypt
etter conversion +
ttacks than LLL.
ative impact? Unclear.

paper, ANTS 1998:
d 147-byte or 503-byte
2^{77} or 2^{170} security.

Let's try NTRU on the computer.

Debian: `apt install sagemath`
Fedora: `yum install sagemath`
Source: www.sagemath.org
Web: sagecell.sagemath.org

Sage is Python 2
+ many math libraries
+ a few syntax differences:

```
sage: 10^6 # power, not xor
1000000
sage: factor(314159265358979323)
317213509 * 990371647
sage:
```

```
sage: Z
sage: #
sage: #
sage: #
sage:
```

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

mir, Eurocrypt
ersion +
n LLL.
ct? Unclear.

TS 1998:
e or 503-byte
70 security.

Let's try NTRU on the computer.

Debian: `apt install sagemath`
Fedora: `yum install sagemath`
Source: `www.sagemath.org`
Web: `sagecell.sagemath.org`

Sage is Python 2
\+ many math libraries
\+ a few syntax differences:

```
sage: 10^6 # power, not xor
1000000
sage: factor(31415926535897932
317213509 * 990371647
sage:
```



```
sage: Zx.<x> = Z
sage: # now Zx i
sage: # Zx objec
sage: # in x wit
sage:
```

U
e
l then
e art)
n.

rypt

ar.

yte

Let's try NTRU on the computer.

Debian: `apt install sagemath`
Fedora: `yum install sagemath`
Source: `www.sagemath.org`
Web: `sagecell.sagemath.org`

Sage is Python 2
+ many math libraries
+ a few syntax differences:

```
sage: 10^6 # power, not xor
1000000
sage: factor(314159265358979323)
317213509 * 990371647
sage:
```

```
sage: Zx.<x> = ZZ[]
sage: # now Zx is a class
sage: # Zx objects are po
sage: # in x with int coe
sage:
```

Let's try NTRU on the computer.

Debian: `apt install sagemath`

Fedora: `yum install sagemath`

Source: `www.sagemath.org`

Web: `sagecell.sagemath.org`

Sage is Python 2

+ many math libraries

+ a few syntax differences:

```
sage: 10^6 # power, not xor
1000000
sage: factor(314159265358979323)
317213509 * 990371647
sage:
```



```
sage: Zx.<x> = ZZ[]
sage: # now Zx is a class
sage: # Zx objects are polys
sage: # in x with int coeffs
sage:
```

Let's try NTRU on the computer.

Debian: `apt install sagemath`

Fedora: `yum install sagemath`

Source: `www.sagemath.org`

Web: `sagecell.sagemath.org`

Sage is Python 2

+ many math libraries

+ a few syntax differences:

```
sage: 10^6 # power, not xor
1000000
sage: factor(314159265358979323)
317213509 * 990371647
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:
```

Let's try NTRU on the computer.

Debian: `apt install sagemath`
Fedora: `yum install sagemath`
Source: www.sagemath.org
Web: sagecell.sagemath.org

Sage is Python 2
+ many math libraries
+ a few syntax differences:

```
sage: 10^6 # power, not xor
1000000
sage: factor(314159265358979323)
317213509 * 990371647
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:
```

Let's try NTRU on the computer.

Debian: `apt install sagemath`

Fedora: `yum install sagemath`

Source: `www.sagemath.org`

Web: `sagecell.sagemath.org`

Sage is Python 2

+ many math libraries

+ a few syntax differences:

```
sage: 10^6 # power, not xor
1000000
sage: factor(314159265358979323)
317213509 * 990371647
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:
```

Let's try NTRU on the computer.

Debian: `apt install sagemath`
Fedora: `yum install sagemath`
Source: `www.sagemath.org`
Web: `sagecell.sagemath.org`

Sage is Python 2
+ many math libraries
+ a few syntax differences:

```
sage: 10^6 # power, not xor
1000000
sage: factor(314159265358979323)
317213509 * 990371647
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:
```

Let's try NTRU on the computer.

Debian: `apt install sagemath`
Fedora: `yum install sagemath`
Source: `www.sagemath.org`
Web: `sagecell.sagemath.org`

Sage is Python 2
+ many math libraries
+ a few syntax differences:

```
sage: 10^6 # power, not xor
1000000
sage: factor(314159265358979323)
317213509 * 990371647
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:
```

NTRU on the computer.

```
apt install sagemath
yum install sagemath
www.sagemath.org
agecell.sagemath.org
```

Python 2

math libraries

syntax differences:

```
0^6 # power, not xor
```

```
actor(31415926535897932 3)
09 * 990371647
```



```
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:
```

n the computer.

:all sagemath

all sagemath

emath.org

sagemath.org

aries

fferences:

er, not xor

159265358979323)

71647


```
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:
```

puter.

math

math

g

.org

or

979323)


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

Will use

Some ch

in submi

$n = 701$

$n = 743$

$n = 761$

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

Will use bigger n f

Some choices of n
in submissions to N

$n = 701$ for NTRU
$n = 743$ for NTRU
$n = 761$ for sntru

1,
etc.
):
^n-1)

riable


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

Will use bigger n for security

Some choices of n
in submissions to NIST:

$n = 701$ for NTRU HRSS.
$n = 743$ for NTRUEncrypt.
$n = 761$ for `sntrup4591761`

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


Will use bigger n for security.

Some choices of n
in submissions to NIST:

$n = 701$ for NTRU HRSS.
$n = 743$ for NTRUEncrypt.
$n = 761$ for `sntrup4591761`.

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


Will use bigger n for security.

Some choices of n
in submissions to NIST:

$n = 701$ for NTRU HRSS.
$n = 743$ for NTRUEncrypt.
$n = 761$ for `sntrup4591761`.

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

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


Will use bigger n for security.

Some choices of n
in submissions to NIST:

$n = 701$ for NTRU HRSS.
$n = 743$ for NTRUEncrypt.
$n = 761$ for `sntrup4591761`.

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

Can we find better algorithms?

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


Will use bigger n for security.

Some choices of n
in submissions to NIST:

$n = 701$ for NTRU HRSS.
$n = 743$ for NTRUEncrypt.
$n = 761$ for `sntrup4591761`.

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

Can we find better algorithms?

1998 NTRU paper took $n = 503$.

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

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


Will use bigger n for security.

Some choices of n
in submissions to NIST:

$n = 701$ for NTRU HRSS.
$n = 743$ for NTRUEncrypt.
$n = 761$ for `sntrup4591761`.

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

Can we find better algorithms?

1998 NTRU paper took $n = 503$.

Modular

For integ
Sage's "
outputs

Matches

```
poly():
(randrange(3)-1
in range(n))
x(f)



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


Will use bigger n for security.

Some choices of n
in submissions to NIST:

$n = 701$ for NTRU HRSS.
$n = 743$ for NTRUEncrypt.
$n = 761$ for `sntrup4591761`.

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

Can we find better algorithms?

1998 NTRU paper took $n = 503$.

Modular reduction

For integers `u`, `q` w
Sage's "`u%q`" alwa
outputs between 0

Matches standard

```
ge(3)-1
(n))
```

```
x^2 +
```

Will use bigger n for security.

Some choices of n
in submissions to NIST:

$n = 701$ for NTRU HRSS.
$n = 743$ for NTRUEncrypt.
$n = 761$ for `sntrup4591761`.

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

Can we find better algorithms?

1998 NTRU paper took $n = 503$.

Modular reduction

For integers `u`, `q` with `q` $>$ 0
Sage's "`u%q`" always produc
outputs between 0 and `q` $-$

Matches standard math defi

Will use bigger n for security.

Some choices of n
in submissions to NIST:

$n = 701$ for NTRU HRSS.
$n = 743$ for NTRUEncrypt.
$n = 761$ for `sntrup4591761`.

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

Can we find better algorithms?

1998 NTRU paper took $n = 503$.

Modular reduction

For integers `u`, `q` with $\mathtt{q} > 0$, Sage's "`u%q`" always produces outputs between 0 and $\mathtt{q} - 1$.

Matches standard math definition.

Will use bigger n for security.

Some choices of n
in submissions to NIST:

$n = 701$ for NTRU HRSS.
$n = 743$ for NTRUEncrypt.
$n = 761$ for `sntrup4591761`.

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

Can we find better algorithms?

1998 NTRU paper took $n = 503$.

Modular reduction

For integers `u`, `q` with `q` > 0, Sage's "`u%q`" always produces outputs between 0 and `q` $- 1$.

Matches standard math definition.

Warning: Typically
`u` < 0 produces `u%q` < 0
in lower-level languages, so nonzero output leaks input sign.

Will use bigger n for security.

Some choices of n
in submissions to NIST:

$n = 701$ for NTRU HRSS.
$n = 743$ for NTRUEncrypt.
$n = 761$ for `sntrup4591761`.

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

Can we find better algorithms?

1998 NTRU paper took $n = 503$.

Modular reduction

For integers `u`, `q` with $\mathtt{q} > 0$, Sage's "`u%q`" always produces outputs between 0 and $\mathtt{q} - 1$.

Matches standard math definition.

Warning: Typically
$\mathtt{u} < 0$ produces $\mathtt{u\%q} < 0$
in lower-level languages, so nonzero output leaks input sign.

Warning: For polynomials `u`,
Sage can make the same mistake.

bigger n for security.

oices of n
issions to NIST:

for NTRU HRSS.
for NTRUEncrypt.
for `sntrup4591761`.

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

find better algorithms?

RU paper took $n = 503$.

Modular reduction

For integers `u`, `q` with `q` > 0,
Sage's "`u%q`" always produces
outputs between 0 and `q` $- 1$.

Matches standard math definition.

Warning: Typically
`u` < 0 produces `u%q` < 0
in lower-level languages, so
nonzero output leaks input sign.

Warning: For polynomials `u`,
Sage can make the same mistake.

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

for security.

NIST:

J HRSS.

JEncrypt.

ıp4591761.

ttack algorithms
n for future
ntum computer.

r algorithms?

r took $n = 503$.

Modular reduction

For integers u, q with $q > 0$,
Sage's "u%q" always produces
outputs between 0 and $q - 1$.

Matches standard math definition.

Warning: Typically
$u < 0$ produces $u\%q < 0$
in lower-level languages, so
nonzero output leaks input sign.

Warning: For polynomials u,
Sage can make the same mistake.

```
sage: def balanc
sage:   g=list((
sage:   -q//2 fo
sage:   return Z
sage:
sage:
```

y.

1.

rithms
e
puter.

ns?

= 503.

Modular reduction

For integers u, q with $q > 0$, Sage's "u%q" always produces outputs between 0 and $q - 1$.

Matches standard math definition.

Warning: Typically $u < 0$ produces $u\%q < 0$ in lower-level languages, so nonzero output leaks input sign.

Warning: For polynomials u, Sage can make the same mistake.

```
sage: def balancedmod(f,q
sage:    g=list(((f[i]+q//
sage:    -q//2 for i in ra
sage:    return Zx(g)
sage:
sage:
```

Modular reduction

For integers u, q with $q > 0$, Sage's "u%q" always produces outputs between 0 and $q - 1$.

Matches standard math definition.

Warning: Typically $u < 0$ produces $u\%q < 0$ in lower-level languages, so nonzero output leaks input sign.

Warning: For polynomials u, Sage can make the same mistake.


```
sage: def balancedmod(f,q):
sage:   g=list(((f[i]+q//2)%q)
sage:   -q//2 for i in range(n))
sage:   return Zx(g)
sage:
sage:
```

Modular reduction

For integers u, q with $q > 0$, Sage's "u%q" always produces outputs between 0 and $q - 1$.

Matches standard math definition.

Warning: Typically $u < 0$ produces $u\%q < 0$ in lower-level languages, so nonzero output leaks input sign.

Warning: For polynomials u, Sage can make the same mistake.


```
sage: def balancedmod(f,q):
sage:   g=list(((f[i]+q//2)%q)
sage:   -q//2 for i in range(n))
sage:   return Zx(g)
sage:
sage: u = 314-159*x
sage:
```

Modular reduction

For integers u, q with $q > 0$, Sage's "u%q" always produces outputs between 0 and $q - 1$.

Matches standard math definition.

Warning: Typically $u < 0$ produces $u\%q < 0$ in lower-level languages, so nonzero output leaks input sign.

Warning: For polynomials u, Sage can make the same mistake.


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

Modular reduction

For integers u, q with $q > 0$, Sage's "u%q" always produces outputs between 0 and $q - 1$.

Matches standard math definition.

Warning: Typically $u < 0$ produces $u\%q < 0$ in lower-level languages, so nonzero output leaks input sign.

Warning: For polynomials u, Sage can make the same mistake.


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

Modular reduction

For integers u, q with $q > 0$, Sage's "u%q" always produces outputs between 0 and $q - 1$.

Matches standard math definition.

Warning: Typically $u < 0$ produces $u\%q < 0$ in lower-level languages, so nonzero output leaks input sign.

Warning: For polynomials u, Sage can make the same mistake.


```
sage: def balancedmod(f,q):
sage:   g=list(((f[i]+q//2)%q)
sage:   -q//2 for i in range(n))
sage:   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:
```

reduction

gers u, q with $q > 0$,
"u%q" always produces
between 0 and $q - 1$.

standard math definition.

: Typically
roduces $u\%q < 0$
-level languages, so
output leaks input sign.

: For polynomials u,
make the same mistake.


```
sage: def balancedmod(f,q):
sage:   g=list(((f[i]+q//2)%q)
sage:   -q//2 for i in range(n))
sage:   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:
```

n

with q > 0,
ays produces
0 and q − 1.

math definition.

y
%q < 0
uages, so
aks input sign.

ynomials u,
e same mistake.


```
sage: def balancedmod(f,q):
sage:   g=list(((f[i]+q//2)%q)
sage:   -q//2 for i in range(n))
sage:   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:
```

,
es
1.
nition.

sign.

,
stake.


```
sage: def balancedmod(f,q):
sage:   g=list(((f[i]+q//2)%q)
sage:   -q//2 for i in range(n))
sage:   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):
sage:   g=list(((f[i]+q//2)%q)
sage:   -q//2 for i in range(n))
sage:   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):
sage:   g=list(((f[i]+q//2)%q)
sage:   -q//2 for i in range(n))
sage:   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):
sage:   g=list(((f[i]+q//2)%q)
sage:   -q//2 for i in range(n))
sage:   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 = randompoly()
sage:
```

```
sage: def balancedmod(f,q):
sage:   g=list(((f[i]+q//2)%q)
sage:   -q//2 for i in range(n))
sage:   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 = randompoly()
sage: f3 = invertmodprime(f,3)
sage:
```

```
sage: def balancedmod(f,q):
sage:   g=list(((f[i]+q//2)%q)
sage:   -q//2 for i in range(n))
sage:   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 = randompoly()
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 = randompoly()
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
  C = c
  while
    r =
    if
    g =
```

Exercise

invertm

Hint: Co

```
edmod(f,q):
(f[i]+q//2)%q)
r 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 = randompoly()
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
  C = convolutio
  while True:
    r = M(C(g,f)
    if r == 1: r
    g = M(C(g,2-
```

Exercise: Figure o

`invertmodpowerc`

Hint: Compare `r`

```
):
2)%q)
nge(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 = randompoly()
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
  C = convolution
  while True:
    r = M(C(g,f),q)
    if r == 1: return g
    g = M(C(g,2-r),q)
```

Exercise: Figure out how `invertmodpowerof2` works.

Hint: Compare `r` to previous

```
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 = randompoly()
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
  C = convolution
  while True:
    r = M(C(g,f),q)
    if r == 1: return g
    g = M(C(g,2-r),q)
```

Exercise: Figure out how `invertmodpowerof2` works.

Hint: Compare `r` to previous `r`.

```
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
 = randompoly()
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
  C = convolution
  while True:
    r = M(C(g,f),q)
    if r == 1: return g
    g = M(C(g,2-r),q)
```

Exercise: Figure out how `invertmodpowerof2` works.

Hint: Compare `r` to previous `r`.

```
sage: n
sage: q
sage:
```

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


poly()
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
  C = convolution
  while True:
    r = M(C(g,f),q)
    if r == 1: return g
    g = M(C(g,2-r),q)
```

Exercise: Figure out how `invertmodpowerof2` works.

Hint: Compare `r` to previous `r`.

```
sage: n = 7
sage: q = 256
sage:
```

```
(f,p):

ing(Fp)
x^n-1)
T(f)))

(f,3)

*x^3 +
```



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

Exercise: Figure out how `invertmodpowerof2` works.

Hint: Compare `r` to previous `r`.

```
sage: n = 7
sage: q = 256
sage:
```

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

Exercise: Figure out how `invertmodpowerof2` works.

Hint: Compare `r` to previous `r`.


```
sage: n = 7
sage: q = 256
sage:
```

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

Exercise: Figure out how `invertmodpowerof2` works.

Hint: Compare `r` to previous `r`.


```
sage: n = 7
sage: q = 256
sage: f = randompoly()
sage:
```

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

Exercise: Figure out how `invertmodpowerof2` works.

Hint: Compare `r` to previous `r`.


```
sage: n = 7
sage: q = 256
sage: f = randompoly()
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
  C = convolution
  while True:
    r = M(C(g,f),q)
    if r == 1: return g
    g = M(C(g,2-r),q)
```

Exercise: Figure out how `invertmodpowerof2` works.

Hint: Compare `r` to previous `r`.


```
sage: n = 7
sage: q = 256
sage: f = randompoly()
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
  C = convolution
  while True:
    r = M(C(g,f),q)
    if r == 1: return g
    g = M(C(g,2-r),q)
```

Exercise: Figure out how `invertmodpowerof2` works.

Hint: Compare `r` to previous `r`.


```
sage: n = 7
sage: q = 256
sage: f = randompoly()
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
  C = convolution
  while True:
    r = M(C(g,f),q)
    if r == 1: return g
    g = M(C(g,2-r),q)
```

Exercise: Figure out how `invertmodpowerof2` works.
Hint: Compare `r` to previous `r`.


```
sage: n = 7
sage: q = 256
sage: f = randompoly()
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
  C = convolution
  while True:
    r = M(C(g,f),q)
    if r == 1: return g
    g = M(C(g,2-r),q)
```

Exercise: Figure out how `invertmodpowerof2` works.

Hint: Compare `r` to previous `r`.


```
sage: n = 7
sage: q = 256
sage: f = randompoly()
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
onvolution
 True:
 M(C(g,f),q)
r == 1: return g
 M(C(g,2-r),q)
```

: Figure out how
odpowerof2 works.
ompare r to previous r.


```
sage: n = 7
sage: q = 256
sage: f = randompoly()
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:
```

NTRU k

Paramet
n, positi
q, powe

```
erof2(f,q):
wer_of(2)
rime(f,2)
d
n

,q)
eturn g
r),q)
```

ut how
of2 works.
to previous r.


```
sage: n = 7
sage: q = 256
sage: f = randompoly()
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:
```

NTRU key genera

Parameters:

n, positive integer

q, power of 2 (e.g

q):

.

s r.


```
sage: n = 7
sage: q = 256
sage: f = randompoly()
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:
```

NTRU key generation

Parameters:

n, positive integer (e.g., 701

q, power of 2 (e.g., 4096).

```
sage: n = 7
sage: q = 256
sage: f = randompoly()
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:
```


NTRU key generation

Parameters:

n, positive integer (e.g., 701);

q, power of 2 (e.g., 4096).

```
sage: n = 7
sage: q = 256
sage: f = randompoly()
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:
```


NTRU key generation

Parameters:

n, positive integer (e.g., 701);
q, power of 2 (e.g., 4096).

Secret key:

random n-coeff polynomial a;
random n-coeff polynomial d;
all coefficients in $\{-1, 0, 1\}$.

```
sage: n = 7
sage: q = 256
sage: f = randompoly()
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:
```


NTRU key generation

Parameters:

n, positive integer (e.g., 701);

q, power of 2 (e.g., 4096).

Secret key:

random n-coeff polynomial a;

random n-coeff polynomial d;

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

Require d invertible mod q.

Require d invertible mod 3.

```
sage: n = 7
sage: q = 256
sage: f = randompoly()
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:
```


NTRU key generation

Parameters:
n, positive integer (e.g., 701);
q, power of 2 (e.g., 4096).

Secret key:
random n-coeff polynomial a;
random n-coeff polynomial d;
all coefficients in $\{-1, 0, 1\}$.

Require d invertible mod q.
Require d invertible mod 3.

Public key: $A = 3a/d$ in the ring $R_q = (\mathbf{Z}/q)[x]/(x^n - 1)$.

```
= 7
= 256
= randompoly()

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)
```


NTRU key generation

Parameters:
n, positive integer (e.g., 701);
q, power of 2 (e.g., 4096).

Secret key:
random n-coeff polynomial a;
random n-coeff polynomial d;
all coefficients in $\{-1, 0, 1\}$.

Require d invertible mod q.
Require d invertible mod 3.

Public key: $A = 3a/d$ in the ring $R_q = (\mathbf{Z}/q)[x]/(x^n - 1)$.

```
def key
  while
    try
      d
      d
      d
      b
    exc
      p
  a = r
  publi

  secre
  retur
```

```
poly()

 + x - 1
modpowerof2(f,q)

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


NTRU key generation

Parameters:
n, positive integer (e.g., 701);
q, power of 2 (e.g., 4096).

Secret key:
random n-coeff polynomial a;
random n-coeff polynomial d;
all coefficients in $\{-1, 0, 1\}$.

Require d invertible mod q.
Require d invertible mod 3.

Public key: $A = 3a/d$ in the ring
$R_q = (\mathbf{Z}/q)[x]/(x^n - 1)$.

```
def keypair():
  while True:
    try:
      d = random
      d3 = inver
      dq = inver
      break
    except:
      pass
  a = randompoly
  publickey = ba
              con
  secretkey = d,
  return publick
```

```
f2(f,q)

 -
 61

x + 257
```


NTRU key generation

Parameters:

n, positive integer (e.g., 701);

q, power of 2 (e.g., 4096).

Secret key:

random n-coeff polynomial a;

random n-coeff polynomial d;

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

Require d invertible mod q.

Require d invertible mod 3.

Public key: $A = 3a/d$ in the ring $R_q = (\mathbf{Z}/q)[x]/(x^n - 1)$.


```
def keypair():
  while True:
    try:
      d = randompoly()
      d3 = invertmodprime
      dq = invertmodpower
      break
    except:
      pass
  a = randompoly()
  publickey = balancedmod
              convolution(
  secretkey = d,d3
  return publickey,secret
```

NTRU key generation

Parameters:

n, positive integer (e.g., 701);

q, power of 2 (e.g., 4096).

Secret key:

random n-coeff polynomial a;

random n-coeff polynomial d;

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

Require d invertible mod q.

Require d invertible mod 3.

Public key: $A = 3a/d$ in the ring $R_q = (\mathbf{Z}/q)[x]/(x^n - 1)$.


```
def keypair():
  while True:
    try:
      d = randompoly()
      d3 = invertmodprime(d,3)
      dq = invertmodpowerof2(d,q)
      break
    except:
      pass
  a = randompoly()
  publickey = balancedmod(3 *
                convolution(a,dq),q)
  secretkey = d,d3
  return publickey,secretkey
```

key generation

ers:

ve integer (e.g., 701);

r of 2 (e.g., 4096).

ey:

n-coeff polynomial a;

n-coeff polynomial d;

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

d invertible mod q.

d invertible mod 3.

ey: $A = 3a/d$ in the ring

$\mathbb{Z}/q)[x]/(x^n - 1)$.


```
def keypair():
  while True:
    try:
      d = randompoly()
      d3 = invertmodprime(d,3)
      dq = invertmodpowerof2(d,q)
      break
    except:
      pass
  a = randompoly()
  publickey = balancedmod(3 *
                convolution(a,dq),q)
  secretkey = d,d3
  return publickey,secretkey
```

```
sage: A
sage:
```

tion

(e.g., 701);
., 4096).

lynomial a;
lynomial d;
$\{-1, 0, 1\}$.

le mod q.
le mod 3.

$3a/d$ in the ring
$^n - 1)$.


```
def keypair():
  while True:
    try:
      d = randompoly()
      d3 = invertmodprime(d,3)
      dq = invertmodpowerof2(d,q)
      break
    except:
      pass
  a = randompoly()
  publickey = balancedmod(3 *
                convolution(a,dq),q)
  secretkey = d,d3
  return publickey,secretkey
```

```
sage: A,secretke
sage:
```

);

a;

d;

e ring


```
def keypair():
  while True:
    try:
      d = randompoly()
      d3 = invertmodprime(d,3)
      dq = invertmodpowerof2(d,q)
      break
    except:
      pass
  a = randompoly()
  publickey = balancedmod(3 *
              convolution(a,dq),q)
  secretkey = d,d3
  return publickey,secretkey
```

```
sage: A,secretkey = keypa
sage:
```

```
def keypair():
  while True:
    try:
      d = randompoly()
      d3 = invertmodprime(d,3)
      dq = invertmodpowerof2(d,q)
      break
    except:
      pass
  a = randompoly()
  publickey = balancedmod(3 *
              convolution(a,dq),q)
  secretkey = d,d3
  return publickey,secretkey
```



```
sage: A,secretkey = keypair()
sage:
```

```
def keypair():
  while True:
    try:
      d = randompoly()
      d3 = invertmodprime(d,3)
      dq = invertmodpowerof2(d,q)
      break
    except:
      pass
  a = randompoly()
  publickey = balancedmod(3 *
              convolution(a,dq),q)
  secretkey = d,d3
  return publickey,secretkey
```



```
sage: A,secretkey = keypair()
sage: A
-126*x^6 - 31*x^5 - 118*x^4 -
 33*x^3 + 73*x^2 - 16*x + 7
sage:
```

```
def keypair():
  while True:
    try:
      d = randompoly()
      d3 = invertmodprime(d,3)
      dq = invertmodpowerof2(d,q)
      break
    except:
      pass
  a = randompoly()
  publickey = balancedmod(3 *
              convolution(a,dq),q)
  secretkey = d,d3
  return publickey,secretkey
```



```
sage: A,secretkey = keypair()
sage: A
-126*x^6 - 31*x^5 - 118*x^4 -
 33*x^3 + 73*x^2 - 16*x + 7
sage: d,d3 = secretkey
sage:
```

```
def keypair():
  while True:
    try:
      d = randompoly()
      d3 = invertmodprime(d,3)
      dq = invertmodpowerof2(d,q)
      break
    except:
      pass
  a = randompoly()
  publickey = balancedmod(3 *
               convolution(a,dq),q)
  secretkey = d,d3
  return publickey,secretkey
```



```
sage: A,secretkey = keypair()
sage: A
-126*x^6 - 31*x^5 - 118*x^4 -
 33*x^3 + 73*x^2 - 16*x + 7
sage: d,d3 = secretkey
sage: d
-x^6 + x^5 - x^4 + x^3 - 1
sage:
```

```
def keypair():
  while True:
    try:
      d = randompoly()
      d3 = invertmodprime(d,3)
      dq = invertmodpowerof2(d,q)
      break
    except:
      pass
  a = randompoly()
  publickey = balancedmod(3 *
              convolution(a,dq),q)
  secretkey = d,d3
  return publickey,secretkey
```



```
sage: A,secretkey = keypair()
sage: A
-126*x^6 - 31*x^5 - 118*x^4 -
 33*x^3 + 73*x^2 - 16*x + 7
sage: d,d3 = secretkey
sage: d
-x^6 + x^5 - x^4 + x^3 - 1
sage: convolution(d,A)
-3*x^6 + 253*x^5 + 253*x^3 -
 253*x^2 - 3*x - 3
sage:
```

```
def keypair():
  while True:
    try:
      d = randompoly()
      d3 = invertmodprime(d,3)
      dq = invertmodpowerof2(d,q)
      break
    except:
      pass
  a = randompoly()
  publickey = balancedmod(3 *
              convolution(a,dq),q)
  secretkey = d,d3
  return publickey,secretkey
```



```
sage: A,secretkey = keypair()
sage: A
-126*x^6 - 31*x^5 - 118*x^4 -
 33*x^3 + 73*x^2 - 16*x + 7
sage: d,d3 = secretkey
sage: d
-x^6 + x^5 - x^4 + x^3 - 1
sage: convolution(d,A)
-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:
:
 = randompoly()
3 = invertmodprime(d,3)
q = invertmodpowerof2(d,q)
reak
ept:
ass
andompoly()
ckey = balancedmod(3 *
      convolution(a,dq),q)
tkey = d,d3
n publickey,secretkey
```



```
sage: A,secretkey = keypair()
sage: A
-126*x^6 - 31*x^5 - 118*x^4 -
 33*x^3 + 73*x^2 - 16*x + 7
sage: d,d3 = secretkey
sage: d
-x^6 + x^5 - x^4 + x^3 - 1
sage: convolution(d,A)
-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:
```

NTRU e

One mo

w, posit

```
poly()
tmodprime(d,3)
tmodpowerof2(d,q)
()
lancedmod(3 *
volution(a,dq),q)
d3
ey,secretkey
```



```
sage: A,secretkey = keypair()
sage: A
-126*x^6 - 31*x^5 - 118*x^4 -
 33*x^3 + 73*x^2 - 16*x + 7
sage: d,d3 = secretkey
sage: d
-x^6 + x^5 - x^4 + x^3 - 1
sage: convolution(d,A)
-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:
```

NTRU encryption

One more paramet

w, positive intege

```
(d,3)
of2(d,q)
(3 *
a,dq),q)
key
```



```
sage: A,secretkey = keypair()
sage: A
-126*x^6 - 31*x^5 - 118*x^4 -
 33*x^3 + 73*x^2 - 16*x + 7
sage: d,d3 = secretkey
sage: d
-x^6 + x^5 - x^4 + x^3 - 1
sage: convolution(d,A)
-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:
```

NTRU encryption

One more parameter:

w, positive integer (e.g., 46

```
sage: A,secretkey = keypair()
sage: A
-126*x^6 - 31*x^5 - 118*x^4 -
 33*x^3 + 73*x^2 - 16*x + 7
sage: d,d3 = secretkey
sage: d
-x^6 + x^5 - x^4 + x^3 - 1
sage: convolution(d,A)
-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:
```


NTRU encryption

One more parameter:
w, positive integer (e.g., 467).

```
sage: A,secretkey = keypair()
sage: A
-126*x^6 - 31*x^5 - 118*x^4 -
 33*x^3 + 73*x^2 - 16*x + 7
sage: d,d3 = secretkey
sage: d
-x^6 + x^5 - x^4 + x^3 - 1
sage: convolution(d,A)
-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:
```


NTRU encryption

One more parameter:
w, positive integer (e.g., 467).

Message for encryption:
n-coeff weight-w polynomial c
with all coeffs in $\{-1, 0, 1\}$.

"Weight w": w nonzero coeffs,
$n - w$ zero coeffs.

```
sage: A,secretkey = keypair()
sage: A
-126*x^6 - 31*x^5 - 118*x^4 -
 33*x^3 + 73*x^2 - 16*x + 7
sage: d,d3 = secretkey
sage: d
-x^6 + x^5 - x^4 + x^3 - 1
sage: convolution(d,A)
-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:
```


NTRU encryption

One more parameter:
w, positive integer (e.g., 467).

Message for encryption:
n-coeff weight-w polynomial c
with all coeffs in $\{-1, 0, 1\}$.

"Weight w": w nonzero coeffs,
$n - w$ zero coeffs.

Ciphertext: $C = Ab + c$ in R_q
where b is chosen randomly
from the set of messages.

```
,secretkey = keypair()

6 - 31*x^5 - 118*x^4 -
 + 73*x^2 - 16*x + 7
,d3 = secretkey

x^5 - x^4 + x^3 - 1
onvolution(d,A)
+ 253*x^5 + 253*x^3 -
2 - 3*x - 3
alancedmod(_,q)
- 3*x^5 - 3*x^3 + 3*x^2
- 3
```


NTRU encryption

One more parameter:
w, positive integer (e.g., 467).

Message for encryption:
n-coeff weight-w polynomial c
with all coeffs in $\{-1, 0, 1\}$.

"Weight w": w nonzero coeffs,
$n - w$ zero coeffs.

Ciphertext: $C = Ab + c$ in R_q
where b is chosen randomly
from the set of messages.

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

```
y = keypair()

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

 + x^3 - 1
n(d,A)
 + 253*x^3 -
 3
d(_,q)
 3*x^3 + 3*x^2
```


NTRU encryption

One more parameter:
w, positive integer (e.g., 467).

Message for encryption:
n-coeff weight-w polynomial c
with all coeffs in $\{-1, 0, 1\}$.

"Weight w": w nonzero coeffs,
$n - w$ zero coeffs.

Ciphertext: $C = Ab + c$ in R_q
where b is chosen randomly
from the set of messages.

```
sage: def random
....:   R = rand
....:   assert w
....:   c = n*[0
....:   for j in
....:     while
....:       r =
....:       if n
....:     c[r] =
....:   return Z
....:
sage: w = 5
sage: randommess
-x^6 - x^5 + x^4
sage:
```

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


NTRU encryption

One more parameter:
w, positive integer (e.g., 467).

Message for encryption:
n-coeff weight-w polynomial c with all coeffs in $\{-1, 0, 1\}$.

"Weight w": w nonzero coeffs, $n - w$ zero coeffs.

Ciphertext: $C = Ab + c$ in R_q where b is chosen randomly from the set of messages.

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

NTRU encryption

One more parameter:
w, positive integer (e.g., 467).

Message for encryption:
n-coeff weight-w polynomial c
with all coeffs in $\{-1, 0, 1\}$.

"Weight w": w nonzero coeffs,
$n - w$ zero coeffs.

Ciphertext: $C = Ab + c$ in R_q
where b is chosen randomly
from the set of messages.


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

encryption

re parameter:
ive integer (e.g., 467).

for encryption:
weight-w polynomial c
coeffs in $\{-1, 0, 1\}$.

w": w nonzero coeffs,
ero coeffs.

ext: $C = Ab + c$ in R_q
is chosen randomly
e set of messages.


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

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

ter:
r (e.g., 467).

ption:
polynomial c
$\{-1, 0, 1\}$.

onzero coeffs,

$Ab + c$ in R_q
randomly
essages.


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

```
sage: def encryp
....:   b = rand
....:   Ab = con
....:   C = bala
....:   return C
....:
sage:
```

7).

l c

effs,

R_q


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



```
sage: def encrypt(c,A):
....:   b = randommessage
....:   Ab = convolution(
....:   C = balancedmod(A
....:   return C
....:
sage:
```

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



```
sage: def encrypt(c,A):
....:   b = randommessage()
....:   Ab = convolution(A,b)
....:   C = balancedmod(Ab + c,q)
....:   return C
....:
sage:
```

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



```
sage: def encrypt(c,A):
....:   b = randommessage()
....:   Ab = convolution(A,b)
....:   C = balancedmod(Ab + c,q)
....:   return C
....:
sage: A,secretkey = keypair()
sage:
```

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



```
sage: def encrypt(c,A):
....:   b = randommessage()
....:   Ab = convolution(A,b)
....:   C = balancedmod(Ab + c,q)
....:   return C
....:
sage: A,secretkey = keypair()
sage: c = randommessage()
sage:
```

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



```
sage: def encrypt(c,A):
....:   b = randommessage()
....:   Ab = convolution(A,b)
....:   C = balancedmod(Ab + c,q)
....:   return C
....:
sage: A,secretkey = keypair()
sage: c = randommessage()
sage: C = encrypt(c,A)
sage:
```

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



```
sage: def encrypt(c,A):
....:   b = randommessage()
....:   Ab = convolution(A,b)
....:   C = balancedmod(Ab + c,q)
....:   return C
....:
sage: A,secretkey = keypair()
sage: c = randommessage()
sage: C = encrypt(c,A)
sage: C
21*x^6 - 48*x^5 + 31*x^4 -
 76*x^3 - 77*x^2 + 15*x - 113
sage:
```

```
ef randommessage():
 R = randrange
 assert w <= n
 c = n*[0]
 for j in range(w):
   while True:
     r = R(n)
     if not c[r]: break
   c[r] = 1-2*R(2)
 return Zx(c)

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



```
sage: def encrypt(c,A):
....:   b = randommessage()
....:   Ab = convolution(A,b)
....:   C = balancedmod(Ab + c,q)
....:   return C
....:
sage: A,secretkey = keypair()
sage: c = randommessage()
sage: C = encrypt(c,A)
sage: C
21*x^6 - 48*x^5 + 31*x^4 -
 76*x^3 - 77*x^2 + 15*x - 113
sage:
```

NTRU d

Comput

```
message():
range
 <= n
]
 range(w):
True:
R(n)
ot c[r]: break
 1-2*R(2)
x(c)


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



```
sage: def encrypt(c,A):
....:     b = randommessage()
....:     Ab = convolution(A,b)
....:     C = balancedmod(Ab + c,q)
....:     return C
....:
sage: A,secretkey = keypair()
sage: c = randommessage()
sage: C = encrypt(c,A)
sage: C
21*x^6 - 48*x^5 + 31*x^4 -
 76*x^3 - 77*x^2 + 15*x - 113
sage:
```

NTRU decryption

Compute $dC = 3a$

:

:

break

x^2


```
sage: def encrypt(c,A):
....:     b = randommessage()
....:     Ab = convolution(A,b)
....:     C = balancedmod(Ab + c,q)
....:     return C
....:
sage: A,secretkey = keypair()
sage: c = randommessage()
sage: C = encrypt(c,A)
sage: C
21*x^6 - 48*x^5 + 31*x^4 -
 76*x^3 - 77*x^2 + 15*x - 113
sage:
```

NTRU decryption

Compute $dC = 3ab + dc$ in

```
sage: def encrypt(c,A):
....:   b = randommessage()
....:   Ab = convolution(A,b)
....:   C = balancedmod(Ab + c,q)
....:   return C
....:
sage: A,secretkey = keypair()
sage: c = randommessage()
sage: C = encrypt(c,A)
sage: C
21*x^6 - 48*x^5 + 31*x^4 -
 76*x^3 - 77*x^2 + 15*x - 113
sage:
```


NTRU decryption

Compute $dC = 3ab + dc$ in R_q.

```
sage: def encrypt(c,A):
....:   b = randommessage()
....:   Ab = convolution(A,b)
....:   C = balancedmod(Ab + c,q)
....:   return C
....:
sage: A,secretkey = keypair()
sage: c = randommessage()
sage: C = encrypt(c,A)
sage: C
21*x^6 - 48*x^5 + 31*x^4 -
 76*x^3 - 77*x^2 + 15*x - 113
sage:
```


NTRU decryption

Compute $dC = 3ab + dc$ in R_q.

a, b, c, d have small coeffs, so $3ab + dc$ is not very big.

```
sage: def encrypt(c,A):
....:   b = randommessage()
....:   Ab = convolution(A,b)
....:   C = balancedmod(Ab + c,q)
....:   return C
....:
sage: A,secretkey = keypair()
sage: c = randommessage()
sage: C = encrypt(c,A)
sage: C
21*x^6 - 48*x^5 + 31*x^4 -
 76*x^3 - 77*x^2 + 15*x - 113
sage:
```


NTRU decryption

Compute $dC = 3ab + dc$ in R_q.

a, b, c, d have small coeffs, so $3ab + dc$ is not very big. **Assume** that coeffs of $3ab + dc$ are between $-q/2$ and $q/2 - 1$.

```
sage: def encrypt(c,A):
....:     b = randommessage()
....:     Ab = convolution(A,b)
....:     C = balancedmod(Ab + c,q)
....:     return C
....:
sage: A,secretkey = keypair()
sage: c = randommessage()
sage: C = encrypt(c,A)
sage: C
21*x^6 - 48*x^5 + 31*x^4 -
 76*x^3 - 77*x^2 + 15*x - 113
sage:
```


NTRU decryption

Compute $dC = 3ab + dc$ in R_q.

a, b, c, d have small coeffs, so $3ab + dc$ is not very big. **Assume** that coeffs of $3ab + dc$ are between $-q/2$ and $q/2 - 1$.

Then $3ab + dc$ in R_q reveals $3ab + dc$ in $R = \mathbf{Z}[x]/(x^n - 1)$.

```
sage: def encrypt(c,A):
....:   b = randommessage()
....:   Ab = convolution(A,b)
....:   C = balancedmod(Ab + c,q)
....:   return C
....:
sage: A,secretkey = keypair()
sage: c = randommessage()
sage: C = encrypt(c,A)
sage: C
21*x^6 - 48*x^5 + 31*x^4 -
 76*x^3 - 77*x^2 + 15*x - 113
sage:
```


NTRU decryption

Compute $dC = 3ab + dc$ in R_q.

a, b, c, d have small coeffs, so $3ab + dc$ is not very big. **Assume** that coeffs of $3ab + dc$ are between $-q/2$ and $q/2 - 1$.

Then $3ab + dc$ in R_q reveals $3ab + dc$ in $R = \mathbf{Z}[x]/(x^n - 1)$. Reduce modulo 3: dc in R_3.

```
sage: def encrypt(c,A):
....:     b = randommessage()
....:     Ab = convolution(A,b)
....:     C = balancedmod(Ab + c,q)
....:     return C
....:
sage: A,secretkey = keypair()
sage: c = randommessage()
sage: C = encrypt(c,A)
sage: C
21*x^6 - 48*x^5 + 31*x^4 -
 76*x^3 - 77*x^2 + 15*x - 113
sage:
```


NTRU decryption

Compute $dC = 3ab + dc$ in R_q.

a, b, c, d have small coeffs, so $3ab + dc$ is not very big. **Assume** that coeffs of $3ab + dc$ are between $-q/2$ and $q/2 - 1$.

Then $3ab + dc$ in R_q reveals $3ab + dc$ in $R = \mathbf{Z}[x]/(x^n - 1)$. Reduce modulo 3: dc in R_3.

Multiply by $1/d$ in R_3 to recover message c in R_3.

```
sage: def encrypt(c,A):
....:     b = randommessage()
....:     Ab = convolution(A,b)
....:     C = balancedmod(Ab + c,q)
....:     return C
....:
sage: A,secretkey = keypair()
sage: c = randommessage()
sage: C = encrypt(c,A)
sage: C
21*x^6 - 48*x^5 + 31*x^4 -
 76*x^3 - 77*x^2 + 15*x - 113
sage:
```


NTRU decryption

Compute $dC = 3ab + dc$ in R_q.

a, b, c, d have small coeffs, so $3ab + dc$ is not very big. **Assume** that coeffs of $3ab + dc$ are between $-q/2$ and $q/2 - 1$.

Then $3ab + dc$ in R_q reveals $3ab + dc$ in $R = \mathbf{Z}[x]/(x^n - 1)$. Reduce modulo 3: dc in R_3.

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

```
ef encrypt(c,A):
 b = randommessage()
 Ab = convolution(A,b)
 C = balancedmod(Ab + c,q)
 return C

,secretkey = keypair()
 = randommessage()
 = encrypt(c,A)

- 48*x^5 + 31*x^4 -
 - 77*x^2 + 15*x - 113
```


NTRU decryption

Compute $dC = 3ab + dc$ in R_q.

a, b, c, d have small coeffs, so $3ab + dc$ is not very big. **Assume** that coeffs of $3ab + dc$ are between $-q/2$ and $q/2 - 1$.

Then $3ab + dc$ in R_q reveals $3ab + dc$ in $R = \mathbf{Z}[x]/(x^n - 1)$. Reduce modulo 3: dc in R_3.

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

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

```
t(c,A):
ommessage()
volution(A,b)
ncedmod(Ab + c,q)


y = keypair()
message()
t(c,A)


+ 31*x^4 -
 + 15*x - 113
```


NTRU decryption

Compute $dC = 3ab + dc$ in R_q.

a, b, c, d have small coeffs, so $3ab + dc$ is not very big. **Assume** that coeffs of $3ab + dc$ are between $-q/2$ and $q/2 - 1$.

Then $3ab + dc$ in R_q reveals $3ab + dc$ in $R = \mathbf{Z}[x]/(x^n - 1)$. Reduce modulo 3: dc in R_3.

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

```
sage: def decryp
....:     M = ba
....:     f,r =
....:     u=M(co
....:     c=M(co
....:     return
....:
sage:
```

```
()
A,b)
b + c,q)

ir()

-
 113
```

NTRU decryption

Compute $dC = 3ab + dc$ in R_q.

a, b, c, d have small coeffs, so $3ab + dc$ is not very big. **Assume** that coeffs of $3ab + dc$ are between $-q/2$ and $q/2 - 1$.

Then $3ab + dc$ in R_q reveals $3ab + dc$ in $R = \mathbf{Z}[x]/(x^n - 1)$. Reduce modulo 3: dc in R_3.

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


```
sage: def decrypt(C,secre
....:     M = balancedmod
....:     f,r = secretkey
....:     u=M(convolution
....:     c=M(convolution
....:     return c
....:
sage:
```

NTRU decryption

Compute $dC = 3ab + dc$ in R_q.

a, b, c, d have small coeffs, so $3ab + dc$ is not very big. **Assume** that coeffs of $3ab + dc$ are between $-q/2$ and $q/2 - 1$.

Then $3ab + dc$ in R_q reveals $3ab + dc$ in $R = \mathbf{Z}[x]/(x^n - 1)$. Reduce modulo 3: dc in R_3.

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


```
sage: def decrypt(C,secretkey):
....:     M = balancedmod
....:     f,r = secretkey
....:     u=M(convolution(C,f),q)
....:     c=M(convolution(u,r),3)
....:     return c
....:
sage:
```

NTRU decryption

Compute $dC = 3ab + dc$ in R_q.

a, b, c, d have small coeffs, so $3ab + dc$ is not very big. **Assume** that coeffs of $3ab + dc$ are between $-q/2$ and $q/2 - 1$.

Then $3ab + dc$ in R_q reveals $3ab + dc$ in $R = \mathbf{Z}[x]/(x^n - 1)$. Reduce modulo 3: dc in R_3.

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


```
sage: def decrypt(C,secretkey):
....:     M = balancedmod
....:     f,r = secretkey
....:     u=M(convolution(C,f),q)
....:     c=M(convolution(u,r),3)
....:     return c
....:
sage: c
x^5 + x^4 - x^3 + x + 1
sage:
```

NTRU decryption

Compute $dC = 3ab + dc$ in R_q.

a, b, c, d have small coeffs, so $3ab + dc$ is not very big. **Assume** that coeffs of $3ab + dc$ are between $-q/2$ and $q/2 - 1$.

Then $3ab + dc$ in R_q reveals $3ab + dc$ in $R = \mathbf{Z}[x]/(x^n - 1)$. Reduce modulo 3: dc in R_3.

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


```
sage: def decrypt(C,secretkey):
....:     M = balancedmod
....:     f,r = secretkey
....:     u=M(convolution(C,f),q)
....:     c=M(convolution(u,r),3)
....:     return c
....:
sage: c
x^5 + x^4 - x^3 + x + 1
sage: decrypt(C,secretkey)
x^5 + x^4 - x^3 + x + 1
sage:
```

lecryption

e $dC = 3ab + dc$ in R_q.

/ have small coeffs,

- dc is not very big.

e that coeffs of $3ab + dc$

veen $-q/2$ and $q/2 - 1$.

$b + dc$ in R_q reveals

c in $R = \mathbf{Z}[x]/(x^n - 1)$.

modulo 3: dc in R_3.

by $1/d$ in R_3

er message c in R_3.

re between -1 and 1,

er c in R.


```
sage: def decrypt(C,secretkey):
....:     M = balancedmod
....:     f,r = secretkey
....:     u=M(convolution(C,f),q)
....:     c=M(convolution(u,r),3)
....:     return c
....:
sage: c
x^5 + x^4 - x^3 + x + 1
sage: decrypt(C,secretkey)
x^5 + x^4 - x^3 + x + 1
sage:
```

```
sage: n
sage: w
sage: q
sage:
```

$ab + dc$ in R_q.

ll coeffs,
t very big.
fs of $3ab + dc$
and $q/2 - 1$.

R_q reveals
$\mathbf{Z}[x]/(x^n - 1)$.
dc in R_3.

n R_3
e c in R_3.
n -1 and 1,


```
sage: def decrypt(C,secretkey):
....:     M = balancedmod
....:     f,r = secretkey
....:     u=M(convolution(C,f),q)
....:     c=M(convolution(u,r),3)
....:     return c
....:
sage: c
x^5 + x^4 - x^3 + x + 1
sage: decrypt(C,secretkey)
x^5 + x^4 - x^3 + x + 1
sage:
```

```
sage: n = 7
sage: w = 5
sage: q = 256
sage:
```

R_q.

$+ dc$

$- 1$.

s

$- 1)$.

.

1,


```
sage: def decrypt(C,secretkey):
....:     M = balancedmod
....:     f,r = secretkey
....:     u=M(convolution(C,f),q)
....:     c=M(convolution(u,r),3)
....:     return c
....:
sage: c
x^5 + x^4 - x^3 + x + 1
sage: decrypt(C,secretkey)
x^5 + x^4 - x^3 + x + 1
sage:
```

```
sage: n = 7
sage: w = 5
sage: q = 256
sage:
```

```
sage: def decrypt(C,secretkey):
....:     M = balancedmod
....:     f,r = secretkey
....:     u=M(convolution(C,f),q)
....:     c=M(convolution(u,r),3)
....:     return c
....:
sage: c
x^5 + x^4 - x^3 + x + 1
sage: decrypt(C,secretkey)
x^5 + x^4 - x^3 + x + 1
sage:
```



```
sage: n = 7
sage: w = 5
sage: q = 256
sage:
```

```
sage: def decrypt(C,secretkey):
....:     M = balancedmod
....:     f,r = secretkey
....:     u=M(convolution(C,f),q)
....:     c=M(convolution(u,r),3)
....:     return c
....:
sage: c
x^5 + x^4 - x^3 + x + 1
sage: decrypt(C,secretkey)
x^5 + x^4 - x^3 + x + 1
sage:
```



```
sage: n = 7
sage: w = 5
sage: q = 256
sage: A,secretkey = keypair()
sage:
```

```
sage: def decrypt(C,secretkey):
....:     M = balancedmod
....:     f,r = secretkey
....:     u=M(convolution(C,f),q)
....:     c=M(convolution(u,r),3)
....:     return c
....:
sage: c
x^5 + x^4 - x^3 + x + 1
sage: decrypt(C,secretkey)
x^5 + x^4 - x^3 + x + 1
sage:
```



```
sage: n = 7
sage: w = 5
sage: q = 256
sage: A,secretkey = keypair()
sage: A
-101*x^6 - 76*x^5 - 90*x^4 -
 83*x^3 + 40*x^2 + 108*x - 54
sage:
```

```
sage: def decrypt(C,secretkey):
....:     M = balancedmod
....:     f,r = secretkey
....:     u=M(convolution(C,f),q)
....:     c=M(convolution(u,r),3)
....:     return c
....:
sage: c
x^5 + x^4 - x^3 + x + 1
sage: decrypt(C,secretkey)
x^5 + x^4 - x^3 + x + 1
sage:
```



```
sage: n = 7
sage: w = 5
sage: q = 256
sage: A,secretkey = keypair()
sage: A
-101*x^6 - 76*x^5 - 90*x^4 -
 83*x^3 + 40*x^2 + 108*x - 54
sage: d,d3 = secretkey
sage:
```

```
sage: def decrypt(C,secretkey):
....:     M = balancedmod
....:     f,r = secretkey
....:     u=M(convolution(C,f),q)
....:     c=M(convolution(u,r),3)
....:     return c
....:
sage: c
x^5 + x^4 - x^3 + x + 1
sage: decrypt(C,secretkey)
x^5 + x^4 - x^3 + x + 1
sage:
```



```
sage: n = 7
sage: w = 5
sage: q = 256
sage: A,secretkey = keypair()
sage: A
-101*x^6 - 76*x^5 - 90*x^4 -
 83*x^3 + 40*x^2 + 108*x - 54
sage: d,d3 = secretkey
sage: d
x^5 + x^4 - x^3 + x - 1
sage:
```

```
sage: def decrypt(C,secretkey):
....:     M = balancedmod
....:     f,r = secretkey
....:     u=M(convolution(C,f),q)
....:     c=M(convolution(u,r),3)
....:     return c
....:
sage: c
x^5 + x^4 - x^3 + x + 1
sage: decrypt(C,secretkey)
x^5 + x^4 - x^3 + x + 1
sage:
```



```
sage: n = 7
sage: w = 5
sage: q = 256
sage: A,secretkey = keypair()
sage: A
-101*x^6 - 76*x^5 - 90*x^4 -
 83*x^3 + 40*x^2 + 108*x - 54
sage: d,d3 = secretkey
sage: d
x^5 + x^4 - x^3 + x - 1
sage: conv = convolution
sage:
```

```
sage: def decrypt(C,secretkey):
....:     M = balancedmod
....:     f,r = secretkey
....:     u=M(convolution(C,f),q)
....:     c=M(convolution(u,r),3)
....:     return c
....:
sage: c
x^5 + x^4 - x^3 + x + 1
sage: decrypt(C,secretkey)
x^5 + x^4 - x^3 + x + 1
sage:
```



```
sage: n = 7
sage: w = 5
sage: q = 256
sage: A,secretkey = keypair()
sage: A
-101*x^6 - 76*x^5 - 90*x^4 -
 83*x^3 + 40*x^2 + 108*x - 54
sage: d,d3 = secretkey
sage: d
x^5 + x^4 - x^3 + x - 1
sage: conv = convolution
sage: M = balancedmod
sage:
```

```
sage: def decrypt(C,secretkey):
....:     M = balancedmod
....:     f,r = secretkey
....:     u=M(convolution(C,f),q)
....:     c=M(convolution(u,r),3)
....:     return c
....:
sage: c
x^5 + x^4 - x^3 + x + 1
sage: decrypt(C,secretkey)
x^5 + x^4 - x^3 + x + 1
sage:
```



```
sage: n = 7
sage: w = 5
sage: q = 256
sage: A,secretkey = keypair()
sage: A
-101*x^6 - 76*x^5 - 90*x^4 -
 83*x^3 + 40*x^2 + 108*x - 54
sage: d,d3 = secretkey
sage: d
x^5 + x^4 - x^3 + x - 1
sage: conv = convolution
sage: M = balancedmod
sage: a3 = M(conv(d,A),q)
sage:
```

```
sage: def decrypt(C,secretkey):
....:     M = balancedmod
....:     f,r = secretkey
....:     u=M(convolution(C,f),q)
....:     c=M(convolution(u,r),3)
....:     return c
....:
sage: c
x^5 + x^4 - x^3 + x + 1
sage: decrypt(C,secretkey)
x^5 + x^4 - x^3 + x + 1
sage:
```



```
sage: n = 7
sage: w = 5
sage: q = 256
sage: A,secretkey = keypair()
sage: A
-101*x^6 - 76*x^5 - 90*x^4 -
 83*x^3 + 40*x^2 + 108*x - 54
sage: d,d3 = secretkey
sage: d
x^5 + x^4 - x^3 + x - 1
sage: conv = convolution
sage: M = balancedmod
sage: a3 = M(conv(d,A),q)
sage: a3
3*x^2 - 3*x
```

```
ef decrypt(C,secretkey):
   M = balancedmod
   f,r = secretkey
   u=M(convolution(C,f),q)
   c=M(convolution(u,r),3)
   return c


^4 - x^3 + x + 1
ecrypt(C,secretkey)
^4 - x^3 + x + 1
```



```
sage: n = 7
sage: w = 5
sage: q = 256
sage: A,secretkey = keypair()
sage: A
-101*x^6 - 76*x^5 - 90*x^4 -
 83*x^3 + 40*x^2 + 108*x - 54
sage: d,d3 = secretkey
sage: d
x^5 + x^4 - x^3 + x - 1
sage: conv = convolution
sage: M = balancedmod
sage: a3 = M(conv(d,A),q)
sage: a3
3*x^2 - 3*x
```

```
sage: c
sage:
```

```
t(C,secretkey):
lancedmod
secretkey
nvolution(C,f),q)
nvolution(u,r),3)
 c


+ x + 1
secretkey)
+ x + 1
```



```
sage: n = 7
sage: w = 5
sage: q = 256
sage: A,secretkey = keypair()
sage: A
-101*x^6 - 76*x^5 - 90*x^4 -
 83*x^3 + 40*x^2 + 108*x - 54
sage: d,d3 = secretkey
sage: d
x^5 + x^4 - x^3 + x - 1
sage: conv = convolution
sage: M = balancedmod
sage: a3 = M(conv(d,A),q)
sage: a3
3*x^2 - 3*x
```

```
sage: c = random
sage:
```

```
tkey):
.
.
(C,f),q)
(u,r),3)

)
```



```
sage: n = 7
sage: w = 5
sage: q = 256
sage: A,secretkey = keypair()
sage: A
-101*x^6 - 76*x^5 - 90*x^4 -
 83*x^3 + 40*x^2 + 108*x - 54
sage: d,d3 = secretkey
sage: d
x^5 + x^4 - x^3 + x - 1
sage: conv = convolution
sage: M = balancedmod
sage: a3 = M(conv(d,A),q)
sage: a3
3*x^2 - 3*x
```

```
sage: c = randommessage()
sage:
```

```
sage: n = 7
sage: w = 5
sage: q = 256
sage: A,secretkey = keypair()
sage: A
-101*x^6 - 76*x^5 - 90*x^4 -
 83*x^3 + 40*x^2 + 108*x - 54
sage: d,d3 = secretkey
sage: d
x^5 + x^4 - x^3 + x - 1
sage: conv = convolution
sage: M = balancedmod
sage: a3 = M(conv(d,A),q)
sage: a3
3*x^2 - 3*x
```



```
sage: c = randommessage()
sage:
```

```
sage: n = 7
sage: w = 5
sage: q = 256
sage: A,secretkey = keypair()
sage: A
-101*x^6 - 76*x^5 - 90*x^4 -
 83*x^3 + 40*x^2 + 108*x - 54
sage: d,d3 = secretkey
sage: d
x^5 + x^4 - x^3 + x - 1
sage: conv = convolution
sage: M = balancedmod
sage: a3 = M(conv(d,A),q)
sage: a3
3*x^2 - 3*x
```



```
sage: c = randommessage()
sage: b = randommessage()
sage:
```

```
sage: n = 7
sage: w = 5
sage: q = 256
sage: A,secretkey = keypair()
sage: A
-101*x^6 - 76*x^5 - 90*x^4 -
 83*x^3 + 40*x^2 + 108*x - 54
sage: d,d3 = secretkey
sage: d
x^5 + x^4 - x^3 + x - 1
sage: conv = convolution
sage: M = balancedmod
sage: a3 = M(conv(d,A),q)
sage: a3
3*x^2 - 3*x
```



```
sage: c = randommessage()
sage: b = randommessage()
sage: C = M(conv(A,b)+c,q)
sage:
```

```
sage: n = 7
sage: w = 5
sage: q = 256
sage: A,secretkey = keypair()
sage: A
-101*x^6 - 76*x^5 - 90*x^4 -
 83*x^3 + 40*x^2 + 108*x - 54
sage: d,d3 = secretkey
sage: d
x^5 + x^4 - x^3 + x - 1
sage: conv = convolution
sage: M = balancedmod
sage: a3 = M(conv(d,A),q)
sage: a3
3*x^2 - 3*x
```



```
sage: c = randommessage()
sage: b = randommessage()
sage: C = M(conv(A,b)+c,q)
sage: C
-57*x^6 + 28*x^5 + 114*x^4 +
 72*x^3 - 37*x^2 + 16*x + 119
sage:
```

```
sage: n = 7
sage: w = 5
sage: q = 256
sage: A,secretkey = keypair()
sage: A
-101*x^6 - 76*x^5 - 90*x^4 -
 83*x^3 + 40*x^2 + 108*x - 54
sage: d,d3 = secretkey
sage: d
x^5 + x^4 - x^3 + x - 1
sage: conv = convolution
sage: M = balancedmod
sage: a3 = M(conv(d,A),q)
sage: a3
3*x^2 - 3*x
```



```
sage: c = randommessage()
sage: b = randommessage()
sage: C = M(conv(A,b)+c,q)
sage: C
-57*x^6 + 28*x^5 + 114*x^4 +
 72*x^3 - 37*x^2 + 16*x + 119
sage: u = M(conv(C,d),q)
sage:
```

```
sage: n = 7
sage: w = 5
sage: q = 256
sage: A,secretkey = keypair()
sage: A
-101*x^6 - 76*x^5 - 90*x^4 -
 83*x^3 + 40*x^2 + 108*x - 54
sage: d,d3 = secretkey
sage: d
x^5 + x^4 - x^3 + x - 1
sage: conv = convolution
sage: M = balancedmod
sage: a3 = M(conv(d,A),q)
sage: a3
3*x^2 - 3*x
```



```
sage: c = randommessage()
sage: b = randommessage()
sage: C = M(conv(A,b)+c,q)
sage: C
-57*x^6 + 28*x^5 + 114*x^4 +
 72*x^3 - 37*x^2 + 16*x + 119
sage: u = M(conv(C,d),q)
sage: u
-8*x^6 + 2*x^5 + 4*x^4 - x^3 -
 4*x^2 + 5*x + 1
sage:
```

```
sage: n = 7
sage: w = 5
sage: q = 256
sage: A,secretkey = keypair()
sage: A
-101*x^6 - 76*x^5 - 90*x^4 -
 83*x^3 + 40*x^2 + 108*x - 54
sage: d,d3 = secretkey
sage: d
x^5 + x^4 - x^3 + x - 1
sage: conv = convolution
sage: M = balancedmod
sage: a3 = M(conv(d,A),q)
sage: a3
3*x^2 - 3*x
```



```
sage: c = randommessage()
sage: b = randommessage()
sage: C = M(conv(A,b)+c,q)
sage: C
-57*x^6 + 28*x^5 + 114*x^4 +
 72*x^3 - 37*x^2 + 16*x + 119
sage: u = M(conv(C,d),q)
sage: u
-8*x^6 + 2*x^5 + 4*x^4 - x^3 -
 4*x^2 + 5*x + 1
sage: conv(a3,b)+conv(c,d)
-8*x^6 + 2*x^5 + 4*x^4 - x^3 -
 4*x^2 + 5*x + 1
```

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

6 - 76*x^5 - 90*x^4 -
 + 40*x^2 + 108*x - 54
,d3 = secretkey

^4 - x^3 + x - 1
onv = convolution
 = balancedmod
3 = M(conv(d,A),q)
3
 3*x
```



```
sage: c = randommessage()
sage: b = randommessage()
sage: C = M(conv(A,b)+c,q)
sage: C
-57*x^6 + 28*x^5 + 114*x^4 +
 72*x^3 - 37*x^2 + 16*x + 119
sage: u = M(conv(C,d),q)
sage: u
-8*x^6 + 2*x^5 + 4*x^4 - x^3 -
 4*x^2 + 5*x + 1
sage: conv(a3,b)+conv(c,d)
-8*x^6 + 2*x^5 + 4*x^4 - x^3 -
 4*x^2 + 5*x + 1
```

```
sage: M
x^6 - x
 + 1
sage:
```

```
y = keypair()

5 - 90*x^4 -
 + 108*x - 54
retkey

+ x - 1
volution
edmod
v(d,A),q)
```



```
sage: c = randommessage()
sage: b = randommessage()
sage: C = M(conv(A,b)+c,q)
sage: C
-57*x^6 + 28*x^5 + 114*x^4 +
 72*x^3 - 37*x^2 + 16*x + 119
sage: u = M(conv(C,d),q)
sage: u
-8*x^6 + 2*x^5 + 4*x^4 - x^3 -
 4*x^2 + 5*x + 1
sage: conv(a3,b)+conv(c,d)
-8*x^6 + 2*x^5 + 4*x^4 - x^3 -
 4*x^2 + 5*x + 1
```

```
sage: M(u,3)
x^6 - x^5 + x^4
 + 1
sage:
```

```
ir()

4 -

- 54
```

```
sage: c = randommessage()
sage: b = randommessage()
sage: C = M(conv(A,b)+c,q)
sage: C
-57*x^6 + 28*x^5 + 114*x^4 +
 72*x^3 - 37*x^2 + 16*x + 119
sage: u = M(conv(C,d),q)
sage: u
-8*x^6 + 2*x^5 + 4*x^4 - x^3 -
 4*x^2 + 5*x + 1
sage: conv(a3,b)+conv(c,d)
-8*x^6 + 2*x^5 + 4*x^4 - x^3 -
 4*x^2 + 5*x + 1
```



```
sage: M(u,3)
x^6 - x^5 + x^4 - x^3 - x
 + 1
sage:
```

```
sage: c = randommessage()
sage: b = randommessage()
sage: C = M(conv(A,b)+c,q)
sage: C
-57*x^6 + 28*x^5 + 114*x^4 +
 72*x^3 - 37*x^2 + 16*x + 119
sage: u = M(conv(C,d),q)
sage: u
-8*x^6 + 2*x^5 + 4*x^4 - x^3 -
 4*x^2 + 5*x + 1
sage: conv(a3,b)+conv(c,d)
-8*x^6 + 2*x^5 + 4*x^4 - x^3 -
 4*x^2 + 5*x + 1
```



```
sage: M(u,3)
x^6 - x^5 + x^4 - x^3 - x^2 - x
 + 1
sage:
```

```
sage: c = randommessage()
sage: b = randommessage()
sage: C = M(conv(A,b)+c,q)
sage: C
-57*x^6 + 28*x^5 + 114*x^4 +
 72*x^3 - 37*x^2 + 16*x + 119
sage: u = M(conv(C,d),q)
sage: u
-8*x^6 + 2*x^5 + 4*x^4 - x^3 -
 4*x^2 + 5*x + 1
sage: conv(a3,b)+conv(c,d)
-8*x^6 + 2*x^5 + 4*x^4 - x^3 -
 4*x^2 + 5*x + 1
```



```
sage: M(u,3)
x^6 - x^5 + x^4 - x^3 - x^2 - x
 + 1
sage: M(conv(c,d),3)
x^6 - x^5 + x^4 - x^3 - x^2 - x
 + 1
sage:
```

```
sage: c = randommessage()
sage: b = randommessage()
sage: C = M(conv(A,b)+c,q)
sage: C
-57*x^6 + 28*x^5 + 114*x^4 +
 72*x^3 - 37*x^2 + 16*x + 119
sage: u = M(conv(C,d),q)
sage: u
-8*x^6 + 2*x^5 + 4*x^4 - x^3 -
 4*x^2 + 5*x + 1
sage: conv(a3,b)+conv(c,d)
-8*x^6 + 2*x^5 + 4*x^4 - x^3 -
 4*x^2 + 5*x + 1
```



```
sage: M(u,3)
x^6 - x^5 + x^4 - x^3 - x^2 - x
 + 1
sage: M(conv(c,d),3)
x^6 - x^5 + x^4 - x^3 - x^2 - x
 + 1
sage: conv(M(u,3),d3)
x^6 - x^5 - x^4 - 3*x^3 - x^2 +
 x - 3
sage:
```

```
sage: c = randommessage()
sage: b = randommessage()
sage: C = M(conv(A,b)+c,q)
sage: C
-57*x^6 + 28*x^5 + 114*x^4 +
 72*x^3 - 37*x^2 + 16*x + 119
sage: u = M(conv(C,d),q)
sage: u
-8*x^6 + 2*x^5 + 4*x^4 - x^3 -
 4*x^2 + 5*x + 1
sage: conv(a3,b)+conv(c,d)
-8*x^6 + 2*x^5 + 4*x^4 - x^3 -
 4*x^2 + 5*x + 1
```



```
sage: M(u,3)
x^6 - x^5 + x^4 - x^3 - x^2 - x
 + 1
sage: M(conv(c,d),3)
x^6 - x^5 + x^4 - x^3 - x^2 - x
 + 1
sage: conv(M(u,3),d3)
x^6 - x^5 - x^4 - 3*x^3 - x^2 +
 x - 3
sage: M(_,3)
x^6 - x^5 - x^4 - x^2 + x
sage:
```

```
sage: c = randommessage()
sage: b = randommessage()
sage: C = M(conv(A,b)+c,q)
sage: C
-57*x^6 + 28*x^5 + 114*x^4 +
 72*x^3 - 37*x^2 + 16*x + 119
sage: u = M(conv(C,d),q)
sage: u
-8*x^6 + 2*x^5 + 4*x^4 - x^3 -
 4*x^2 + 5*x + 1
sage: conv(a3,b)+conv(c,d)
-8*x^6 + 2*x^5 + 4*x^4 - x^3 -
 4*x^2 + 5*x + 1
```



```
sage: M(u,3)
x^6 - x^5 + x^4 - x^3 - x^2 - x
 + 1
sage: M(conv(c,d),3)
x^6 - x^5 + x^4 - x^3 - x^2 - x
 + 1
sage: conv(M(u,3),d3)
x^6 - x^5 - x^4 - 3*x^3 - x^2 +
 x - 3
sage: M(_,3)
x^6 - x^5 - x^4 - x^2 + x
sage: c
x^6 - x^5 - x^4 - x^2 + x
sage:
```

```
 = randommessage()
 = randommessage()
 = M(conv(A,b)+c,q)

 + 28*x^5 + 114*x^4 +
 - 37*x^2 + 16*x + 119
 = M(conv(C,d),q)

+ 2*x^5 + 4*x^4 - x^3 -
+ 5*x + 1
onv(a3,b)+conv(c,d)
+ 2*x^5 + 4*x^4 - x^3 -
+ 5*x + 1
```



```
sage: M(u,3)
x^6 - x^5 + x^4 - x^3 - x^2 - x
 + 1
sage: M(conv(c,d),3)
x^6 - x^5 + x^4 - x^3 - x^2 - x
 + 1
sage: conv(M(u,3),d3)
x^6 - x^5 - x^4 - 3*x^3 - x^2 +
 x - 3
sage: M(_,3)
x^6 - x^5 - x^4 - x^2 + x
sage: c
x^6 - x^5 - x^4 - x^2 + x
sage:
```

Does de

All coeff

All coeff

and exac

```
message()
message()
(A,b)+c,q)

 + 114*x^4 +
 + 16*x + 119
(C,d),q)

 4*x^4 - x^3 -

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



```
sage: M(u,3)
x^6 - x^5 + x^4 - x^3 - x^2 - x
 + 1
sage: M(conv(c,d),3)
x^6 - x^5 + x^4 - x^3 - x^2 - x
 + 1
sage: conv(M(u,3),d3)
x^6 - x^5 - x^4 - 3*x^3 - x^2 +
 x - 3
sage: M(_,3)
x^6 - x^5 - x^4 - x^2 + x
sage: c
x^6 - x^5 - x^4 - x^2 + x
sage:
```

Does decryption a

All coeffs of a are

All coeffs of b are

and exactly w are

)

4 +

119

x^3 -

)

x^3 -


```
sage: M(u,3)
x^6 - x^5 + x^4 - x^3 - x^2 - x
 + 1
sage: M(conv(c,d),3)
x^6 - x^5 + x^4 - x^3 - x^2 - x
 + 1
sage: conv(M(u,3),d3)
x^6 - x^5 - x^4 - 3*x^3 - x^2 +
 x - 3
sage: M(_,3)
x^6 - x^5 - x^4 - x^2 + x
sage: c
x^6 - x^5 - x^4 - x^2 + x
sage:
```

Does decryption always work

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

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

and exactly w are nonzero.

```
sage: M(u,3)
x^6 - x^5 + x^4 - x^3 - x^2 - x
 + 1
sage: M(conv(c,d),3)
x^6 - x^5 + x^4 - x^3 - x^2 - x
 + 1
sage: conv(M(u,3),d3)
x^6 - x^5 - x^4 - 3*x^3 - x^2 +
 x - 3
sage: M(_,3)
x^6 - x^5 - x^4 - x^2 + x
sage: c
x^6 - x^5 - x^4 - x^2 + x
sage:
```


Does decryption always work?

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

```
sage: M(u,3)
x^6 - x^5 + x^4 - x^3 - x^2 - x
 + 1
sage: M(conv(c,d),3)
x^6 - x^5 + x^4 - x^3 - x^2 - x
 + 1
sage: conv(M(u,3),d3)
x^6 - x^5 - x^4 - 3*x^3 - x^2 +
 x - 3
sage: M(_,3)
x^6 - x^5 - x^4 - x^2 + x
sage: c
x^6 - x^5 - x^4 - x^2 + x
sage:
```


Does decryption always work?

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

Each coeff of ab in R
has absolute value at most w.

```
sage: M(u,3)
x^6 - x^5 + x^4 - x^3 - x^2 - x
 + 1
sage: M(conv(c,d),3)
x^6 - x^5 + x^4 - x^3 - x^2 - x
 + 1
sage: conv(M(u,3),d3)
x^6 - x^5 - x^4 - 3*x^3 - x^2 +
 x - 3
sage: M(_,3)
x^6 - x^5 - x^4 - x^2 + x
sage: c
x^6 - x^5 - x^4 - x^2 + x
sage:
```


Does decryption always work?

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

Each coeff of ab in R
has absolute value at most w.
(Same argument would work for
b of any weight, a of weight w.)

```
sage: M(u,3)
x^6 - x^5 + x^4 - x^3 - x^2 - x
 + 1
sage: M(conv(c,d),3)
x^6 - x^5 + x^4 - x^3 - x^2 - x
 + 1
sage: conv(M(u,3),d3)
x^6 - x^5 - x^4 - 3*x^3 - x^2 +
 x - 3
sage: M(_,3)
x^6 - x^5 - x^4 - x^2 + x
sage: c
x^6 - x^5 - x^4 - x^2 + x
sage:
```


Does decryption always work?

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

Each coeff of ab in R
has absolute value at most w.
(Same argument would work for
b of any weight, a of weight w.)

Similar comments for d, c.
Each coeff of $3ab + dc$ in R
has absolute value at most $4w$.

```
sage: M(u,3)
x^6 - x^5 + x^4 - x^3 - x^2 - x
 + 1
sage: M(conv(c,d),3)
x^6 - x^5 + x^4 - x^3 - x^2 - x
 + 1
sage: conv(M(u,3),d3)
x^6 - x^5 - x^4 - 3*x^3 - x^2 +
 x - 3
sage: M(_,3)
x^6 - x^5 - x^4 - x^2 + x
sage: c
x^6 - x^5 - x^4 - x^2 + x
sage:
```


Does decryption always work?

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

Each coeff of ab in R
has absolute value at most w.
(Same argument would work for
b of any weight, a of weight w.)

Similar comments for d, c.
Each coeff of $3ab + dc$ in R
has absolute value at most $4w$.

e.g. $w = 467$: at most 1868.
Decryption works for $q = 4096$.

```
(u,3)
^5 + x^4 - x^3 - x^2 - x

(conv(c,d),3)
^5 + x^4 - x^3 - x^2 - x

onv(M(u,3),d3)
^5 - x^4 - 3*x^3 - x^2 +

(_,3)
^5 - x^4 - x^2 + x

^5 - x^4 - x^2 + x
```


Does decryption always work?

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

Each coeff of ab in R
has absolute value at most w.
(Same argument would work for
b of any weight, a of weight w.)

Similar comments for d, c.
Each coeff of $3ab + dc$ in R
has absolute value at most $4w$.

e.g. $w = 467$: at most 1868.
Decryption works for $q = 4096$.

What ab

```
- x^3 - x^2 - x

),3)
- x^3 - x^2 - x

),d3)
- 3*x^3 - x^2 +

- x^2 + x

- x^2 + x
```


Does decryption always work?

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

Each coeff of ab in R
has absolute value at most w.
(Same argument would work for
b of any weight, a of weight w.)

Similar comments for d, c.
Each coeff of $3ab + dc$ in R
has absolute value at most $4w$.

e.g. $w = 467$: at most 1868.
Decryption works for $q = 4096$.

What about $w =$

x^2 - x

x^2 - x

x^2 +

Does decryption always work?

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

Each coeff of ab in R has absolute value at most w.
(Same argument would work for b of any weight, a of weight w.)

Similar comments for d, c.
Each coeff of $3ab + dc$ in R has absolute value at most $4w$.

e.g. $w = 467$: at most 1868.
Decryption works for $q = 4096$.

What about $w = 467$, $q = 2$

Does decryption always work?

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

Each coeff of ab in R
has absolute value at most w.
(Same argument would work for
b of any weight, a of weight w.)

Similar comments for d, c.
Each coeff of $3ab + dc$ 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 a are in $\{-1, 0, 1\}$.
All coeffs of b are in $\{-1, 0, 1\}$,
and exactly w are nonzero.

Each coeff of ab in R
has absolute value at most w.
(Same argument would work for
b of any weight, a of weight w.)

Similar comments for d, c.
Each coeff of $3ab + dc$ 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.
$a = b = c = d =$
$1 + x + x^2 + \cdots + x^{w-1}$:
$3ab + dc$ has a coeff $4w > q/2$.

Does decryption always work?

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

Each coeff of ab in R has absolute value at most w.
(Same argument would work for b of any weight, a of weight w.)

Similar comments for d, c.
Each coeff of $3ab + dc$ 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.
$a = b = c = d = 1 + x + x^2 + \cdots + x^{w-1}$:
$3ab + dc$ has a coeff $4w > q/2$.

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

Does decryption always work?

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

Each coeff of ab in R
has absolute value at most w.
(Same argument would work for
b of any weight, a of weight w.)

Similar comments for d, c.
Each coeff of $3ab + dc$ 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.
$a = b = c = d =$
$1 + x + x^2 + \cdots + x^{w-1}$:
$3ab + dc$ 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 a are in $\{-1, 0, 1\}$.
fs of b are in $\{-1, 0, 1\}$,
ctly w are nonzero.

eff of ab in R
olute value at most w.
rgument would work for
weight, a of weight w.)

comments for d, c.
eff of $3ab + dc$ 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.
$a = b = c = d =$
$1 + x + x^2 + \cdots + x^{w-1}$:
$3ab + dc$ 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
decryptio
security

Decrypti
"all the
for vario
not be v

lways work?

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

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

for d, c.
$+ dc$ in R
at most $4w$.

most 1868.
for $q = 4096$.

What about $w = 467$, $q = 2048$?

Same argument doesn't work.
$a = b = c = d =$
$1 + x + x^2 + \cdots + x^{w-1}$:
$3ab + dc$ 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
not be valid after

x?

1}.

, 1},

w.

x for

w.)

?

4w.

.

096.

What about $w = 467$, $q = 2048$?

Same argument doesn't work.
$a = b = c = d =$
$1 + x + x^2 + \cdots + x^{w-1}$:
$3ab + dc$ 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
not be valid after all".

What about $w = 467$, $q = 2048$?

Same argument doesn't work.
$a = b = c = d =$
$1 + x + x^2 + \cdots + x^{w-1}$:
$3ab + dc$ 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.
$a = b = c = d =$
$1 + x + x^2 + \cdots + x^{w-1}$:
$3ab + dc$ 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.

$c = d =$

$x^2 + \cdots + x^{w-1}$:

c has a coeff $4w > q/2$.

ffs are usually <1024

d are chosen randomly.

TRU handout mentioned

yption-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

$c_0 d_{n-1}$ -

This coe

$c_0, c_1, \ldots$

high cor

d_{n-1}, d_n

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 c

$c_0 d_{n-1} + c_1 d_{n-2}$ –

This coeff is large

$c_0, c_1, \ldots, c_{n-1}$ ha

high correlation w

$d_{n-1}, d_{n-2}, \ldots, d_0$

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 cd is $c_0 d_{n-1} + c_1 d_{n-2} + \ldots + c_{n-}$

This coeff is large $\Leftrightarrow$ $c_0, c_1, \ldots, c_{n-1}$ 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 cd is $c_0 d_{n-1} + c_1 d_{n-2} + \ldots + c_{n-1} d_0$.

This coeff is large $\Leftrightarrow$ $c_0, c_1, \ldots, c_{n-1}$ 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 cd is $c_0 d_{n-1} + c_1 d_{n-2} + \ldots + c_{n-1} d_0$.

This coeff is large $\Leftrightarrow$ $c_0, c_1, \ldots, c_{n-1}$ has high correlation with $d_{n-1}, d_{n-2}, \ldots, d_0$.

Some coeff is large $\Leftrightarrow$ $c_0, c_1, \ldots, c_{n-1}$ 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 cd is $c_0 d_{n-1} + c_1 d_{n-2} + \ldots + c_{n-1} d_0$.

This coeff is large $\Leftrightarrow$ $c_0, c_1, \ldots, c_{n-1}$ has high correlation with $d_{n-1}, d_{n-2}, \ldots, d_0$.

Some coeff is large $\Leftrightarrow$ $c_0, c_1, \ldots, c_{n-1}$ has high correlation with some rotation of $d_{n-1}, d_{n-2}, \ldots, d_0$.

i.e. c 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 may
valid after all".

rse: Attacker who sees
ndom decryption failures
re out the secret key!

Coeff of x^{n-1} in cd is
$c_0 d_{n-1} + c_1 d_{n-2} + \ldots + c_{n-1} d_0$.

This coeff is large $\Leftrightarrow$
$c_0, c_1, \ldots, c_{n-1}$ has
high correlation with
$d_{n-1}, d_{n-2}, \ldots, d_0$.

Some coeff is large $\Leftrightarrow$
$c_0, c_1, \ldots, c_{n-1}$ has high
correlation with some rotation
of $d_{n-1}, d_{n-2}, \ldots, d_0$.

i.e. c 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
c correla

grave-Graham–
val–Proos–
-Whyte

. on the
encryption":

s imply that
roofs known . . .
paddings may
all".

cker who sees
ryption failures
secret key!

Coeff of x^{n-1} in cd is
$c_0 d_{n-1} + c_1 d_{n-2} + \ldots + c_{n-1} d_0$.

This coeff is large $\Leftrightarrow$
$c_0, c_1, \ldots, c_{n-1}$ has
high correlation with
$d_{n-1}, d_{n-2}, \ldots, d_0$.

Some coeff is large $\Leftrightarrow$
$c_0, c_1, \ldots, c_{n-1}$ has high
correlation with some rotation
of $d_{n-1}, d_{n-2}, \ldots, d_0$.

i.e. c 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 decryption
c correlated with s

nam–

-

n":

at

vn . . .

may

ees

lures

y!

Coeff of x^{n-1} in cd is
$c_0 d_{n-1} + c_1 d_{n-2} + \ldots + c_{n-1} d_0$.

This coeff is large $\Leftrightarrow$
$c_0, c_1, \ldots, c_{n-1}$ has
high correlation with
$d_{n-1}, d_{n-2}, \ldots, d_0$.

Some coeff is large $\Leftrightarrow$
$c_0, c_1, \ldots, c_{n-1}$ has high
correlation with some rotation
of $d_{n-1}, d_{n-2}, \ldots, d_0$.

i.e. c 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:
c correlated with some x^i re

Coeff of x^{n-1} in cd is
$c_0 d_{n-1} + c_1 d_{n-2} + \ldots + c_{n-1} d_0$.

This coeff is large $\Leftrightarrow$
$c_0, c_1, \ldots, c_{n-1}$ has
high correlation with
$d_{n-1}, d_{n-2}, \ldots, d_0$.

Some coeff is large $\Leftrightarrow$
$c_0, c_1, \ldots, c_{n-1}$ has high
correlation with some rotation
of $d_{n-1}, d_{n-2}, \ldots, d_0$.

i.e. c 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:
c correlated with some $x^i \operatorname{rev}(d)$.

Coeff of x^{n-1} in cd is
$c_0 d_{n-1} + c_1 d_{n-2} + \ldots + c_{n-1} d_0$.

This coeff is large $\Leftrightarrow$
$c_0, c_1, \ldots, c_{n-1}$ has
high correlation with
$d_{n-1}, d_{n-2}, \ldots, d_0$.

Some coeff is large $\Leftrightarrow$
$c_0, c_1, \ldots, c_{n-1}$ has high
correlation with some rotation
of $d_{n-1}, d_{n-2}, \ldots, d_0$.

i.e. c 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:
c correlated with some $x^i \operatorname{rev}(d)$.
$\operatorname{rev}(c)$ correlated with $x^{-i} d$.

Coeff of x^{n-1} in cd is
$c_0 d_{n-1} + c_1 d_{n-2} + \ldots + c_{n-1} d_0$.

This coeff is large $\Leftrightarrow$
$c_0, c_1, \ldots, c_{n-1}$ has
high correlation with
$d_{n-1}, d_{n-2}, \ldots, d_0$.

Some coeff is large $\Leftrightarrow$
$c_0, c_1, \ldots, c_{n-1}$ has high
correlation with some rotation
of $d_{n-1}, d_{n-2}, \ldots, d_0$.

i.e. c 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:
c correlated with some $x^i \operatorname{rev}(d)$.
$\operatorname{rev}(c)$ correlated with $x^{-i} d$.
$c \operatorname{rev}(c)$ correlated with $d \operatorname{rev}(d)$.

Coeff of x^{n-1} in cd is
$c_0 d_{n-1} + c_1 d_{n-2} + \ldots + c_{n-1} d_0$.

This coeff is large $\Leftrightarrow$
$c_0, c_1, \ldots, c_{n-1}$ has
high correlation with
$d_{n-1}, d_{n-2}, \ldots, d_0$.

Some coeff is large $\Leftrightarrow$
$c_0, c_1, \ldots, c_{n-1}$ has high
correlation with some rotation
of $d_{n-1}, d_{n-2}, \ldots, d_0$.

i.e. c 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:
c correlated with some $x^i \operatorname{rev}(d)$.
$\operatorname{rev}(c)$ correlated with $x^{-i} d$.
$c \operatorname{rev}(c)$ correlated with $d \operatorname{rev}(d)$.

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

Coeff of x^{n-1} in cd is
$c_0 d_{n-1} + c_1 d_{n-2} + \ldots + c_{n-1} d_0$.

This coeff is large $\Leftrightarrow$
$c_0, c_1, \ldots, c_{n-1}$ has
high correlation with
$d_{n-1}, d_{n-2}, \ldots, d_0$.

Some coeff is large $\Leftrightarrow$
$c_0, c_1, \ldots, c_{n-1}$ has high
correlation with some rotation
of $d_{n-1}, d_{n-2}, \ldots, d_0$.

i.e. c 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:
c correlated with some $x^i \operatorname{rev}(d)$.
$\operatorname{rev}(c)$ correlated with $x^{-i} d$.
$c \operatorname{rev}(c)$ correlated with $d \operatorname{rev}(d)$.

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

Eurocrypt 2002 Gentry–Szydlo
algorithm then finds d.

x^{n-1} in cd is

$+ c_1 d_{n-2} + \ldots + c_{n-1} d_0$.

eff is large $\Leftrightarrow$

$., c_{n-1}$ has

relation with

$_{-2}, \ldots, d_0$.

oeff is large $\Leftrightarrow$

$., c_{n-1}$ 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:

c correlated with some $x^i \operatorname{rev}(d)$.

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

$c \operatorname{rev}(c)$ correlated with $d \operatorname{rev}(d)$.

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

Eurocrypt 2002 Gentry–Szydlo algorithm then finds d.

1999 Ha

2000 Ja

Hoffstei

Fluhrer,

using inv

d is

$+ \ldots + c_{n-1} d_0$.

$\Leftrightarrow$

ns

ith

n.

e $\Leftrightarrow$

ns high

ome rotation

d_0.

with

i, where

$^{n-1} + \cdots + d_{n-1} x$.

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

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

Eurocrypt 2002 Gentry–Szydlo algorithm then finds d.

1999 Hall–Goldber

2000 Jaulmes–Jou

Hoffstein–Silverma

Fluhrer, etc.: Ever

using invalid mess

$_{-1} d_0$.

on

$d_{n-1} x$.

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

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

Eurocrypt 2002 Gentry–Szydlo algorithm then finds d.

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

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

Eurocrypt 2002 Gentry–Szydlo algorithm then finds d.

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

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

Eurocrypt 2002 Gentry–Szydlo algorithm then finds d.

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

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

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

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

Eurocrypt 2002 Gentry–Szydlo algorithm then finds d.

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

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

This changes $3ab + dc$: adds
$\pm d$, $\pm xd$, ..., $\pm x^{n-1} d$;
$\pm 2d$, $\pm 2xd$, ..., $\pm 2x^{n-1} d$;
$\pm 3d$, etc.

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

entally confirmed:
of c rev(c)
ne decryption failures
to d rev(d).
o integers: d rev(d).

pt 2002 Gentry–Szydlo
m then finds d.

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

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

This changes $3ab + dc$: adds
$\pm d$, $\pm xd$, ..., $\pm x^{n-1}d$;
$\pm 2d$, $\pm 2xd$, ..., $\pm 2x^{n-1}d$;
$\pm 3d$, etc.

e.g. $3ab$
all other
and $d =$

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

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

entry–Szydlo
ds d.

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

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

This changes $3ab + dc$: adds
$\pm d$, $\pm xd$, ..., $\pm x^{n-1}d$;
$\pm 2d$, $\pm 2xd$, ..., $\pm 2x^{n-1}d$;
$\pm 3d$, etc.

e.g. $3ab+dc = \cdots$
all other coeffs in
and $d = \cdots + x^{47}$

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 c to
$c \pm 1$, $c \pm x$, . . . , $c \pm x^{n-1}$;
$c \pm 2$, $c \pm 2x$, . . . , $c \pm 2x^{n-1}$;
$c \pm 3$, etc.

This changes $3ab + dc$: adds
$\pm d$, $\pm xd$, . . . , $\pm x^{n-1}d$;
$\pm 2d$, $\pm 2xd$, . . . , $\pm 2x^{n-1}d$;
$\pm 3d$, etc.

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

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

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

This changes $3ab + dc$: adds
$\pm d$, $\pm xd$, ..., $\pm x^{n-1}d$;
$\pm 2d$, $\pm 2xd$, ..., $\pm 2x^{n-1}d$;
$\pm 3d$, etc.

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

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

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

This changes $3ab + dc$: adds
$\pm d$, $\pm xd$, ..., $\pm x^{n-1}d$;
$\pm 2d$, $\pm 2xd$, ..., $\pm 2x^{n-1}d$;
$\pm 3d$, etc.

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

Then $3ab + dc + kd =$
$\cdots + (390 + k)x^{478} + \cdots$.
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 c to
$c \pm 1$, $c \pm x$, ..., $c \pm x^{n-1}$;
$c \pm 2$, $c \pm 2x$, ..., $c \pm 2x^{n-1}$;
$c \pm 3$, etc.

This changes $3ab + dc$: adds
$\pm d$, $\pm xd$, ..., $\pm x^{n-1}d$;
$\pm 2d$, $\pm 2xd$, ..., $\pm 2x^{n-1}d$;
$\pm 3d$, etc.

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

Then $3ab + dc + kd =$
$\cdots + (390 + k)x^{478} + \cdots$.
Decryption fails for big k.

Search for smallest k that falis.

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

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

This changes $3ab + dc$: adds
$\pm d$, $\pm xd$, ..., $\pm x^{n-1}d$;
$\pm 2d$, $\pm 2xd$, ..., $\pm 2x^{n-1}d$;
$\pm 3d$, etc.

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

Then $3ab + dc + kd =$
$\cdots + (390 + k)x^{478} + \cdots$.
Decryption fails for big k.

Search for smallest k that falis.

Does $3ab + dc + kxd$ also fail?
Yes *if* $xd = \cdots + x^{478} + \cdots$,
i.e., if $d = \cdots + x^{477} + \cdots$.

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

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

This changes $3ab + dc$: adds
$\pm d$, $\pm xd$, ..., $\pm x^{n-1}d$;
$\pm 2d$, $\pm 2xd$, ..., $\pm 2x^{n-1}d$;
$\pm 3d$, etc.

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

Then $3ab + dc + kd =$
$\cdots + (390 + k)x^{478} + \cdots$.
Decryption fails for big k.

Search for smallest k that falis.

Does $3ab + dc + kxd$ also fail?
Yes *if* $xd = \cdots + x^{478} + \cdots$,
i.e., if $d = \cdots + x^{477} + \cdots$.

Try x^2kd, x^3kd, etc.
See pattern of d coeffs.

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

changes c to
$\pm x, \ldots, c \pm x^{n-1}$;
$\pm 2x, \ldots, c \pm 2x^{n-1}$;
tc.

anges $3ab + dc$: adds
$d, \ldots, \pm x^{n-1}d$;
$2xd, \ldots, \pm 2x^{n-1}d$;
c.

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

Then $3ab + dc + kd =$
$\cdots + (390 + k)x^{478} + \cdots$.
Decryption fails for big k.

Search for smallest k that falis.

Does $3ab + dc + kxd$ also fail?
Yes *if* $xd = \cdots + x^{478} + \cdots$,
i.e., if $d = \cdots + x^{477} + \cdots$.

Try x^2kd, x^3kd, etc.
See pattern of d coeffs.

How to

Approac
constant

For each
generate
Use sign
that nob

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

c to
$c \pm x^{n-1}$;
, $c \pm 2x^{n-1}$;

$+ dc$: adds
$x^{n-1}d$;
$\pm 2x^{n-1}d$;

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

Then $3ab + dc + kd =$
$\cdots + (390 + k)x^{478} + \cdots$.
Decryption fails for big k.

Search for smallest k that falis.

Does $3ab + dc + kxd$ also fail?
Yes *if* $xd = \cdots + x^{478} + \cdots$,
i.e., if $d = \cdots + x^{477} + \cdots$.

Try x^2kd, x^3kd, etc.
See pattern of d coeffs.

How to handle inv

Approach 1: Tell u
constantly switch

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

er,

tacks

$^{-1}$;

ds

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

Then $3ab + dc + kd =$
$\cdots + (390 + k)x^{478} + \cdots$.
Decryption fails for big k.

Search for smallest k that falis.

Does $3ab + dc + kxd$ also fail?
Yes *if* $xd = \cdots + x^{478} + \cdots$,
i.e., if $d = \cdots + x^{477} + \cdots$.

Try x^2kd, x^3kd, etc.
See pattern of d 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. $3ab+dc = \cdots+390x^{478}+\cdots$,
all other coeffs in $[-389, 389]$;
and $d = \cdots + x^{478} + \cdots$.

Then $3ab + dc + kd =$
$\cdots + (390 + k)x^{478} + \cdots$.
Decryption fails for big k.

Search for smallest k that falis.

Does $3ab + dc + kxd$ also fail?
Yes *if* $xd = \cdots + x^{478} + \cdots$,
i.e., if $d = \cdots + x^{477} + \cdots$.

Try x^2kd, x^3kd, etc.
See pattern of d 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. $3ab+dc = \cdots+390x^{478}+\cdots$,
all other coeffs in $[-389, 389]$;
and $d = \cdots + x^{478} + \cdots$.

Then $3ab + dc + kd =$
$\cdots + (390 + k)x^{478} + \cdots$.
Decryption fails for big k.

Search for smallest k that falis.

Does $3ab + dc + kxd$ also fail?
Yes *if* $xd = \cdots + x^{478} + \cdots$,
i.e., if $d = \cdots + x^{477} + \cdots$.

Try x^2kd, x^3kd, etc.
See pattern of d 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. original “IND-CPA” version of New Hope; Ding.

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

Then $3ab + dc + kd =$
$\cdots + (390 + k)x^{478} + \cdots$.
Decryption fails for big k.

Search for smallest k that falis.

Does $3ab + dc + kxd$ also fail?
Yes *if* $xd = \cdots + x^{478} + \cdots$,
i.e., if $d = \cdots + x^{477} + \cdots$.

Try x^2kd, x^3kd, etc.
See pattern of d 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. original "IND-CPA" version of New Hope; Ding.

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

$b+dc = \cdots+390x^{478}+\cdots$,
coeffs in $[-389, 389]$;
$\cdots + x^{478} + \cdots$.

$b + dc + kd =$
$90 + k)x^{478} + \cdots$.
ion fails for big k.

or smallest k that falis.

$b + dc + kxd$ also fail?
$d = \cdots + x^{478} + \cdots$,
$= \cdots + x^{477} + \cdots$.

d, x^3kd, etc.
ern of d 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. original “IND-CPA” version of New Hope; Ding.

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

Approac
encrypti
eliminat

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

$[-389, 389]$;

$^{8} + \cdots$.

$kd =$

$^{78} + \cdots$.

r big k.

t k that falis.

kxd also fail?

$x^{478} + \cdots$,

$^{477} + \cdots$.

etc.

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. original "IND-CPA" version of New Hope; Ding.

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

Approach 2: Modi
encryption and de
eliminate invalid m

$^{8}+\cdots,$

9];

alis.

fail?

$\cdot,$

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. original "IND-CPA" version of New Hope; Ding.

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

Approach 2: Modify encryption and decryption to eliminate invalid 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.

e.g. original "IND-CPA" version of New Hope; Ding.

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

Approach 2: Modify encryption and decryption to eliminate invalid 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.

e.g. original "IND-CPA" version of New Hope; Ding.

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

Approach 2: Modify encryption and decryption to eliminate invalid messages.

e.g. "IND-CCA" New Hope submission; most submissions.

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. original "IND-CPA" version of New Hope; Ding.

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

Approach 2: Modify encryption and decryption to eliminate invalid messages.

e.g. "IND-CCA" New Hope submission; most submissions.

Basic idea, from Crypto 1999 Fujisaki–Okamoto: After decrypting message, check whether (1) message is valid and (2) ciphertext matches reencryption of message.

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. original “IND-CPA” version of New Hope; Ding.

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

Approach 2: Modify encryption and decryption to eliminate invalid messages.

e.g. “IND-CCA” New Hope submission; most submissions.

Basic idea, from Crypto 1999 Fujisaki–Okamoto: After decrypting message, check whether (1) message is valid and (2) ciphertext matches reencryption of message.

But encryption is randomized! Reencryption won’t match.

handle invalid messages

h 1: Tell user to
tly switch keys.

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

inal “IND-CPA” version
Hope; Ding.

euses a key:
ser for the attacks.

Approach 2: Modify encryption and decryption to eliminate invalid messages.

e.g. “IND-CCA” New Hope submission; most submissions.

Basic idea, from Crypto 1999 Fujisaki–Okamoto: After decrypting message, check whether (1) message is valid and (2) ciphertext matches reencryption of message.

But encryption is randomized! Reencryption won't match.

Solution
all rand

e.g. afte
compute

alid messages

user to
keys.

ler,
ic key.
ensure
uses key.

-CPA" version
g.

y:
e attacks.

Approach 2: Modify encryption and decryption to eliminate invalid messages.

e.g. "IND-CCA" New Hope submission; most submissions.

Basic idea, from Crypto 1999 Fujisaki–Okamoto: After decrypting message, check whether (1) message is valid and (2) ciphertext matches reencryption of message.

But encryption is randomized! Reencryption won't match.

Solution: In decry
all randomness tha

e.g. after computi
compute *b* from 3

ages

sion

Approach 2: Modify encryption and decryption to eliminate invalid messages.

e.g. “IND-CCA” New Hope submission; most submissions.

Basic idea, from Crypto 1999 Fujisaki–Okamoto: After decrypting message, check whether (1) message is valid and (2) ciphertext matches reencryption of message.

But encryption is randomized! Reencryption won’t match.

Solution: In decryption, com

all randomness that was use

e.g. after computing c in NT

compute b from $3ab + dc$.

Approach 2: Modify encryption and decryption to eliminate invalid messages.

e.g. "IND-CCA" New Hope submission; most submissions.

Basic idea, from Crypto 1999 Fujisaki–Okamoto: After decrypting message, check whether (1) message is valid and (2) ciphertext matches reencryption of message.

But encryption is randomized! Reencryption won't match.

Solution: In decryption, compute all randomness that was used.

e.g. after computing c in NTRU, compute b from $3ab + dc$.

Approach 2: Modify encryption and decryption to eliminate invalid messages.

e.g. "IND-CCA" New Hope submission; most submissions.

Basic idea, from Crypto 1999 Fujisaki–Okamoto: After decrypting message, check whether (1) message is valid and (2) ciphertext matches reencryption of message.

But encryption is randomized! Reencryption won't match.

Solution: In decryption, compute all randomness that was used.

e.g. after computing c in NTRU, compute b from $3ab + dc$.

Can view (b, c) as message, no further randomness. "Deterministic encryption."

Approach 2: Modify encryption and decryption to eliminate invalid messages.

e.g. “IND-CCA” New Hope submission; most submissions.

Basic idea, from Crypto 1999 Fujisaki–Okamoto: After decrypting message, check whether (1) message is valid and (2) ciphertext matches reencryption of message.

But encryption is randomized! Reencryption won’t match.

Solution: In decryption, compute all randomness that was used.

e.g. after computing c in NTRU, compute b from $3ab + dc$.

Can view (b, c) as message, no further randomness. “Deterministic encryption.”

“Product NTRU” variant is not naturally deterministic.

Approach 2: Modify encryption and decryption to eliminate invalid messages.

e.g. "IND-CCA" New Hope submission; most submissions.

Basic idea, from Crypto 1999 Fujisaki–Okamoto: After decrypting message, check whether (1) message is valid and (2) ciphertext matches reencryption of message.

But encryption is randomized! Reencryption won't match.

Solution: In decryption, compute all randomness that was used.

e.g. after computing c in NTRU, compute b from $3ab + dc$.

Can view (b, c) as message, no further randomness. "Deterministic encryption."

"Product NTRU" variant is not naturally deterministic.

Generic Fujisaki–Okamoto solution: Require sender to compute randomness as standard hash of message.

h 2: Modify
on and decryption to
e invalid messages.

D-CCA" New Hope
on; most submissions.

ea, from Crypto 1999
-Okamoto: After
ng message, check
(1) message is valid
ciphertext matches
ption of message.

ryption is randomized!
ption won't match.

Solution: In decryption, compute all randomness that was used.

e.g. after computing c in NTRU, compute b from $3ab + dc$.

Can view (b, c) as message, no further randomness. "Deterministic encryption."

"Product NTRU" variant is not naturally deterministic.

Generic Fujisaki–Okamoto solution: Require sender to compute randomness as standard hash of message.

How to

Eliminat
not enou
using de
random

ify
cryption to
nessages.

New Hope
submissions.

Crypto 1999
: After
ge, check
ge is valid
matches
essage.

randomized!
't match.

Solution: In decryption, compute all randomness that was used.

e.g. after computing c in NTRU, compute b from $3ab + dc$.

Can view (b, c) as message, no further randomness. "Deterministic encryption."

"Product NTRU" variant is not naturally deterministic.

Generic Fujisaki–Okamoto solution: Require sender to compute randomness as standard hash of message.

How to handle dec

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

Solution: In decryption, compute all randomness that was used.

e.g. after computing c in NTRU, compute b from $3ab + dc$.

Can view (b, c) as message, no further randomness. "Deterministic encryption."

"Product NTRU" variant is not naturally deterministic.

Generic Fujisaki–Okamoto solution: Require sender to compute randomness as standard hash of message.

How to handle decryption fa

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

Solution: In decryption, compute all randomness that was used.

e.g. after computing c in NTRU, compute b from $3ab + dc$.

Can view (b, c) as message, no further randomness. "Deterministic encryption."

"Product NTRU" variant is not naturally deterministic.

Generic Fujisaki–Okamoto solution: Require sender to compute randomness as standard hash of message.

How to handle decryption failures

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

Solution: In decryption, compute all randomness that was used.

e.g. after computing c in NTRU, compute b from $3ab + dc$.

Can view (b, c) as message, no further randomness. "Deterministic encryption."

"Product NTRU" variant is not naturally deterministic.

Generic Fujisaki–Okamoto solution: Require sender to compute randomness as standard hash of message.

How to handle decryption failures

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

NIST encryption submissions vary in failure rates.

NTRU HRSS, NTRU Prime, Odd Manhattan choose q to eliminate decryption failures.

Solution: In decryption, compute all randomness that was used.

e.g. after computing c in NTRU, compute b from $3ab + dc$.

Can view (b, c) as message, no further randomness. "Deterministic encryption."

"Product NTRU" variant is not naturally deterministic.

Generic Fujisaki–Okamoto solution: Require sender to compute randomness as standard hash of message.

How to handle decryption failures

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

NIST encryption submissions vary in failure rates.

NTRU HRSS, NTRU Prime, Odd Manhattan choose q to eliminate decryption failures.

LIMA tried to eliminate decryption failures, but failed.

: In decryption, compute
omness that was used.

r computing c in NTRU,
e b from $3ab + dc$.

v (b, c) as message,
er randomness.
inistic encryption."

t NTRU" variant
aturally deterministic.

Fujisaki–Okamoto
: Require sender to
e randomness as
l hash of message.

How to handle decryption failures

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

NIST encryption submissions vary in failure rates.

NTRU HRSS, NTRU Prime, Odd Manhattan choose q to eliminate decryption failures.

LIMA tried to eliminate decryption failures, but failed.

More cla
LOTUS:
New Ho
KINDI: 2
⋮
NTRUE
KCL: ≈2
Ding: ≈

Current
what de
is small
decrypti
were cal

ption, compute
at was used.

ng c in NTRU,
$ab + dc$.

message,
ness.
cryption."

variant
terministic.

Okamoto
sender to
ess as
message.

How to handle decryption failures

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

NIST encryption submissions vary in failure rates.

NTRU HRSS, NTRU Prime, Odd Manhattan choose q to eliminate decryption failures.

LIMA tried to eliminate decryption failures, but failed.

More claimed failu
LOTUS: $<2^{-256}$.
New Hope submis
KINDI: 2^{-165}.
⋮
NTRUEncrypt: <
KCL: $\approx 2^{-60}$.
Ding: $\approx 2^{-60}$, only

Current debates a
what decryption fa
is small enough; w
decryption failure
were calculated co

npute
d.
TRU,
c.

How to handle decryption failures

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

NIST encryption submissions vary in failure rates.

NTRU HRSS, NTRU Prime, Odd Manhattan choose q to eliminate decryption failures.

LIMA tried to eliminate decryption failures, but failed.

More claimed failure rates:
LOTUS: $<2^{-256}$.
New Hope submission: $<2^{-}$
KINDI: 2^{-165}.
$\vdots$
NTRUEncrypt: $<2^{-80}$.
KCL: $\approx 2^{-60}$.
Ding: $\approx 2^{-60}$, only IND-CPA

Current debates about what decryption failure prob is small enough; whether decryption failure probabiliti were calculated correctly; et

How to handle decryption failures

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

NIST encryption submissions vary in failure rates.

NTRU HRSS, NTRU Prime, Odd Manhattan choose q to eliminate decryption failures.

LIMA tried to eliminate decryption failures, but failed.

More claimed failure rates:

LOTUS: $<2^{-256}$.

New Hope submission: $<2^{-213}$.

KINDI: 2^{-165}.

$\vdots$

NTRUEncrypt: $<2^{-80}$.

KCL: $\approx 2^{-60}$.

Ding: $\approx 2^{-60}$, only IND-CPA.

Current debates about what decryption failure probability is small enough; whether decryption failure probabilities were calculated correctly; etc.

handle decryption failures

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

cryption submissions
ailure rates.

HRSS, NTRU Prime,
nhattan choose q to
e decryption failures.

ied to eliminate
on failures, but failed.

More claimed failure rates:

LOTUS: $<2^{-256}$.

New Hope submission: $<2^{-213}$.

KINDI: 2^{-165}.

$\vdots$

NTRUEncrypt: $<2^{-80}$.

KCL: $\approx 2^{-60}$.

Ding: $\approx 2^{-60}$, only IND-CPA.

Current debates about what decryption failure probability is small enough; whether decryption failure probabilities were calculated correctly; etc.

How to

If messa
Attacker
a guess

cryption failures

l messages is
mber attack
ailures for
sages.

ubmissions
es.

RU Prime,
hoose q to
on failures.

ninate
, but failed.

More claimed failure rates:

LOTUS: $<2^{-256}$.

New Hope submission: $<2^{-213}$.

KINDI: 2^{-165}.

$\vdots$

NTRUEncrypt: $<2^{-80}$.

KCL: $\approx 2^{-60}$.

Ding: $\approx 2^{-60}$, only IND-CPA.

Current debates about what decryption failure probability is small enough; whether decryption failure probabilities were calculated correctly; etc.

How to randomize

If message is gues
Attacker can chec
a guess matches a

ilures

s is

ck

s

,

o

.

d.

More claimed failure rates:

LOTUS: $<2^{-256}$.

New Hope submission: $<2^{-213}$.

KINDI: 2^{-165}.

⋮

NTRUEncrypt: $<2^{-80}$.

KCL: $\approx 2^{-60}$.

Ding: $\approx 2^{-60}$, only IND-CPA.

Current debates about what decryption failure probability is small enough; whether decryption failure probabilities were calculated correctly; etc.

How to randomize messages

If message is guessable:

Attacker can check whether a guess matches a ciphertex

More claimed failure rates:

LOTUS: $<2^{-256}$.

New Hope submission: $<2^{-213}$.

KINDI: 2^{-165}.

$\vdots$

NTRUEncrypt: $<2^{-80}$.

KCL: $\approx 2^{-60}$.

Ding: $\approx 2^{-60}$, only IND-CPA.

Current debates about what decryption failure probability is small enough; whether decryption failure probabilities were calculated correctly; etc.

How to randomize messages

If message is guessable: Attacker can check whether a guess matches a ciphertext.

More claimed failure rates:

LOTUS: $<2^{-256}$.

New Hope submission: $<2^{-213}$.

KINDI: 2^{-165}.

$\vdots$

NTRUEncrypt: $<2^{-80}$.

KCL: $\approx 2^{-60}$.

Ding: $\approx 2^{-60}$, only IND-CPA.

Current debates about what decryption failure probability is small enough; whether decryption failure probabilities were calculated correctly; etc.

How to randomize messages

If message is guessable: Attacker can check whether a guess matches a ciphertext.

Also various attacks using guesses of portion of message.

More claimed failure rates:

LOTUS: $<2^{-256}$.

New Hope submission: $<2^{-213}$.

KINDI: 2^{-165}.

$\vdots$

NTRUEncrypt: $<2^{-80}$.

KCL: $\approx 2^{-60}$.

Ding: $\approx 2^{-60}$, only IND-CPA.

Current debates about what decryption failure probability is small enough; whether decryption failure probabilities were calculated correctly; etc.

How to randomize messages

If message is guessable: Attacker can check whether a guess matches a ciphertext.

Also various attacks using guesses of portion of message.

Modern "KEM-DEM" solution, from Eurocrypt 2000 Shoup: Choose random message. Use hash of message as (e.g.) AES-256-GCM key to encrypt and authenticate user data.

aimed failure rates:
$<2^{-256}$.
pe submission: $<2^{-213}$.
2^{-165}.

ncrypt: $<2^{-80}$.
2^{-60}.
2^{-60}, only IND-CPA.

debates about
cryption failure probability
enough; whether
on failure probabilities
culated correctly; etc.

How to randomize messages

If message is guessable:
Attacker can check whether
a guess matches a ciphertext.

Also various attacks using
guesses of portion of message.

Modern “KEM-DEM” solution,
from Eurocrypt 2000 Shoup:
Choose random message.
Use hash of message as (e.g.)
AES-256-GCM key to encrypt
and authenticate user data.

Central
Can atta
a rando
public k

ure rates:

sion: $<2^{-213}$.

2^{-80}.

y IND-CPA.

bout
ailure probability
vhether
probabilities
orrectly; etc.

How to randomize messages

If message is guessable:
Attacker can check whether
a guess matches a ciphertext.

Also various attacks using
guesses of portion of message.

Modern "KEM-DEM" solution,
from Eurocrypt 2000 Shoup:
Choose random message.
Use hash of message as (e.g.)
AES-256-GCM key to encrypt
and authenticate user data.

Central "one-wayn
Can attacker figur
a random message
public key and cip

-213.

A.

ability

es
c.

How to randomize messages

If message is guessable:
Attacker can check whether
a guess matches a ciphertext.

Also various attacks using
guesses of portion of message.

Modern “KEM-DEM” solution,
from Eurocrypt 2000 Shoup:
Choose random message.
Use hash of message as (e.g.)
AES-256-GCM key to encrypt
and authenticate user data.

Central “one-wayness” quest
Can attacker figure out
a random message given
public key and ciphertext?

How to randomize messages

If message is guessable:
Attacker can check whether
a guess matches a ciphertext.

Also various attacks using
guesses of portion of message.

Modern "KEM-DEM" solution,
from Eurocrypt 2000 Shoup:
Choose random message.
Use hash of message as (e.g.)
AES-256-GCM key to encrypt
and authenticate user data.

Central "one-wayness" question:
Can attacker figure out
a random message given
public key and ciphertext?

How to randomize messages

If message is guessable:
Attacker can check whether
a guess matches a ciphertext.

Also various attacks using
guesses of portion of message.

Modern "KEM-DEM" solution,
from Eurocrypt 2000 Shoup:
Choose random message.
Use hash of message as (e.g.)
AES-256-GCM key to encrypt
and authenticate user data.

Central "one-wayness" question:
Can attacker figure out
a random message given
public key and ciphertext?

Fujisaki–Okamoto and many
newer papers try to prove that all
chosen-ciphertext distinguishers
("IND-CCA attacks") are as
difficult as breaking one-wayness.

How to randomize messages

If message is guessable:
Attacker can check whether
a guess matches a ciphertext.

Also various attacks using
guesses of portion of message.

Modern "KEM-DEM" solution,
from Eurocrypt 2000 Shoup:
Choose random message.
Use hash of message as (e.g.)
AES-256-GCM key to encrypt
and authenticate user data.

Central "one-wayness" question:
Can attacker figure out
a random message given
public key and ciphertext?

Fujisaki–Okamoto and many
newer papers try to prove that all
chosen-ciphertext distinguishers
("IND-CCA attacks") are as
difficult as breaking one-wayness.

Many limitations to proofs: bugs;
looseness; assumptions of "ROM"
or "QROM" attacks; assumptions
on failure probability; etc.

randomize messages

ge is guessable:
r can check whether
matches a ciphertext.

ious attacks using
of portion of message.

"KEM-DEM" solution,
rocrypt 2000 Shoup:
random message.
n of message as (e.g.)
6-GCM key to encrypt
henticate user data.

Central "one-wayness" question:
Can attacker figure out
a random message given
public key and ciphertext?

Fujisaki–Okamoto and many newer papers try to prove that all chosen-ciphertext distinguishers ("IND-CCA attacks") are as difficult as breaking one-wayness.

Many limitations to proofs: bugs; looseness; assumptions of "ROM" or "QROM" attacks; assumptions on failure probability; etc.

Brute-fo

Attacker
$A = 3a/$
Can atta

e messages

sable:
k whether
ciphertext.

ks using
of message.

EM" solution,
000 Shoup:
essage.
ge as (e.g.)
y to encrypt
user data.

Central "one-wayness" question: Can attacker figure out a random message given public key and ciphertext?

Fujisaki–Okamoto and many newer papers try to prove that all chosen-ciphertext distinguishers ("IND-CCA attacks") are as difficult as breaking one-wayness.

Many limitations to proofs: bugs; looseness; assumptions of "ROM" or "QROM" attacks; assumptions on failure probability; etc.

Brute-force search

Attacker is given p
$A = 3a/d$, ciphert
Can attacker find

Central "one-wayness" question: Can attacker figure out a random message given public key and ciphertext?

Fujisaki–Okamoto and many newer papers try to prove that all chosen-ciphertext distinguishers ("IND-CCA attacks") are as difficult as breaking one-wayness.

Many limitations to proofs: bugs; looseness; assumptions of "ROM" or "QROM" attacks; assumptions on failure probability; etc.

Brute-force search

Attacker is given public key $A = 3a/d$, ciphertext $C = A$

Can attacker find c?

Central "one-wayness" question:
Can attacker figure out a random message given public key and ciphertext?

Fujisaki–Okamoto and many newer papers try to prove that all chosen-ciphertext distinguishers ("IND-CCA attacks") are as difficult as breaking one-wayness.

Many limitations to proofs: bugs; looseness; assumptions of "ROM" or "QROM" attacks; assumptions on failure probability; etc.

Brute-force search

Attacker is given public key $A = 3a/d$, ciphertext $C = Ab + c$.
Can attacker find c?

Central "one-wayness" question: Can attacker figure out a random message given public key and ciphertext?

Fujisaki–Okamoto and many newer papers try to prove that all chosen-ciphertext distinguishers ("IND-CCA attacks") are as difficult as breaking one-wayness.

Many limitations to proofs: bugs; looseness; assumptions of "ROM" or "QROM" attacks; assumptions on failure probability; etc.

Brute-force search

Attacker is given public key $A = 3a/d$, ciphertext $C = Ab + c$. Can attacker find c?

Search $\binom{n}{w}2^w$ choices of b. If $c = C - Ab$ is small: done!

Central "one-wayness" question: Can attacker figure out a random message given public key and ciphertext?

Fujisaki–Okamoto and many newer papers try to prove that all chosen-ciphertext distinguishers ("IND-CCA attacks") are as difficult as breaking one-wayness.

Many limitations to proofs: bugs; looseness; assumptions of "ROM" or "QROM" attacks; assumptions on failure probability; etc.

Brute-force search

Attacker is given public key $A = 3a/d$, ciphertext $C = Ab + c$. Can attacker find c?

Search $\binom{n}{w}2^w$ choices of b. If $c = C - Ab$ is small: done!

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

Central "one-wayness" question: Can attacker figure out a random message given public key and ciphertext?

Fujisaki–Okamoto and many newer papers try to prove that all chosen-ciphertext distinguishers ("IND-CCA attacks") are as difficult as breaking one-wayness.

Many limitations to proofs: bugs; looseness; assumptions of "ROM" or "QROM" attacks; assumptions on failure probability; etc.

Brute-force search

Attacker is given public key $A = 3a/d$, ciphertext $C = Ab + c$. Can attacker find c?

Search $\binom{n}{w}2^w$ choices of b. If $c = C - Ab$ is small: done!

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

Or search 3^n choices of d. If $a = dA/3$ is small, use (a, d) to decrypt. Slightly slower but can be reused for many ciphertexts.

"one-wayness" question:
acker figure out
n message given
ey and ciphertext?

-Okamoto and many
apers try to prove that all
ciphertext distinguishers
CCA attacks") are as
as breaking one-wayness.

mitations to proofs: bugs;
s; assumptions of "ROM"
OM" attacks; assumptions
e probability; etc.

Brute-force search

Attacker is given public key $A = 3a/d$, ciphertext $C = Ab + c$. Can attacker find c?

Search $\binom{n}{w}2^w$ choices of b. If $c = C - Ab$ is small: done!

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

Or search 3^n choices of d. If $a = dA/3$ is small, use (a, d) to decrypt. Slightly slower but can be reused for many ciphertexts.

Equivale

Secret k
secret ke
secret ke

ness" question:
e out
e given
hertext?

and many
o prove that all
distinguishers
ks") are as
g one-wayness.

o proofs: bugs;
tions of "ROM"
ks; assumptions
ity; etc.

Brute-force search

Attacker is given public key $A = 3a/d$, ciphertext $C = Ab + c$. Can attacker find c?

Search $\binom{n}{w}2^w$ choices of b. If $c = C - Ab$ is small: done!

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

Or search 3^n choices of d. If $a = dA/3$ is small, use (a, d) to decrypt. Slightly slower but can be reused for many ciphertexts.

Equivalent keys

Secret key (a, d) i
secret key $(xa, xd$
secret key $(x^2a, x^2$

tion:

/
nat all
ners
s
yness.

bugs;
ROM"
ptions

Brute-force search

Attacker is given public key $A = 3a/d$, ciphertext $C = Ab + c$. Can attacker find c?

Search $\binom{n}{w}2^w$ choices of b. If $c = C - Ab$ is small: done!

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

Or search 3^n choices of d. If $a = dA/3$ is small, use (a, d) to decrypt. Slightly slower but can be reused for many ciphertexts.

Equivalent keys

Secret key (a, d) is equivaler
secret key (xa, xd),
secret key (x^2a, x^2d), etc.

Brute-force search

Attacker is given public key $A = 3a/d$, ciphertext $C = Ab + c$. Can attacker find c?

Search $\binom{n}{w}2^w$ choices of b. If $c = C - Ab$ is small: done!

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

Or search 3^n choices of d. If $a = dA/3$ is small, use (a, d) to decrypt. Slightly slower but can be reused for many ciphertexts.

Equivalent keys

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

Brute-force search

Attacker is given public key $A = 3a/d$, ciphertext $C = Ab + c$. Can attacker find c?

Search $\binom{n}{w}2^w$ choices of b. If $c = C - Ab$ is small: done!

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

Or search 3^n choices of d. If $a = dA/3$ is small, use (a, d) to decrypt. Slightly slower but can be reused for many ciphertexts.

Equivalent keys

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

Search only about $3^n/n$ choices.

Brute-force search

Attacker is given public key $A = 3a/d$, ciphertext $C = Ab + c$. Can attacker find c?

Search $\binom{n}{w}2^w$ choices of b. If $c = C - Ab$ is small: done!

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

Or search 3^n choices of d. If $a = dA/3$ is small, use (a, d) to decrypt. Slightly slower but can be reused for many ciphertexts.

Equivalent keys

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

Search only about $3^n/n$ choices.

$n = 701$, $w = 467$:

$$\binom{n}{w}2^w \approx 2^{1106.09};$$
$$3^n \approx 2^{1111.06};$$
$$3^n/n \approx 2^{1101.61}.$$

Brute-force search

Attacker is given public key $A = 3a/d$, ciphertext $C = Ab + c$. Can attacker find c?

Search $\binom{n}{w}2^w$ choices of b. If $c = C - Ab$ is small: done!

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

Or search 3^n choices of d. If $a = dA/3$ is small, use (a, d) to decrypt. Slightly slower but can be reused for many ciphertexts.

Equivalent keys

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

Search only about $3^n/n$ choices.

$n = 701$, $w = 467$:

$$\binom{n}{w}2^w \approx 2^{1106.09};$$
$$3^n \approx 2^{1111.06};$$
$$3^n/n \approx 2^{1101.61}.$$

Exercise: Find more equivalences!

Brute-force search

Attacker is given public key $A = 3a/d$, ciphertext $C = Ab + c$. Can attacker find c?

Search $\binom{n}{w}2^w$ choices of b. If $c = C - Ab$ is small: done!

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

Or search 3^n choices of d. If $a = dA/3$ is small, use (a, d) to decrypt. Slightly slower but can be reused for many ciphertexts.

Equivalent keys

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

Search only about $3^n/n$ choices.

$n = 701$, $w = 467$:

$$\binom{n}{w}2^w \approx 2^{1106.09};$$
$$3^n \approx 2^{1111.06};$$
$$3^n/n \approx 2^{1101.61}.$$

Exercise: Find more equivalences!

But if w is chosen smaller then $\binom{n}{w}2^w$ search will be faster.

rce search

r is given public key
d, ciphertext $C = Ab + c$.
acker find c?

$\binom{n}{w}2^w$ choices of b.
$- Ab$ is small: done!

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

ch 3^n choices of d.
$A/3$ is small, use (a, d) to
Slightly slower but can
d for many ciphertexts.

Equivalent keys

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

Search only about $3^n/n$ choices.

$n = 701$, $w = 467$:

$$\binom{n}{w}2^w \approx 2^{1106.09};$$
$$3^n \approx 2^{1111.06};$$
$$3^n/n \approx 2^{1101.61}.$$

Exercise: Find more equivalences!

But if w is chosen smaller then
$\binom{n}{w}2^w$ search will be faster.

Collision

Write d
$d_1 =$ bo
$d_2 =$ rer

ublic key
ext $C = Ab + c$.
c?

ices of b.
mall: done!

different
kely. This would
e decryption.)

ces of d.
all, use (a, d) to
lower but can
y ciphertexts.

Equivalent keys

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

Search only about $3^n/n$ choices.

$n = 701$, $w = 467$:

$$\binom{n}{w}2^w \approx 2^{1106.09};$$
$$3^n \approx 2^{1111.06};$$
$$3^n/n \approx 2^{1101.61}.$$

Exercise: Find more equivalences!

But if w is chosen smaller then
$\binom{n}{w}2^w$ search will be faster.

Collision attacks

Write d as $d_1 + d$
d_1 = bottom $\lceil n/2$
d_2 = remaining te

$b + c$.

e!

would

on.)

, d) to

can

xts.

Equivalent keys

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

Search only about $3^n/n$ choices.

$n = 701$, $w = 467$:

$$\binom{n}{w}2^w \approx 2^{1106.09};$$
$$3^n \approx 2^{1111.06};$$
$$3^n/n \approx 2^{1101.61}.$$

Exercise: Find more equivalences!

But if w is chosen smaller then $\binom{n}{w}2^w$ search will be faster.

Collision attacks

Write d as $d_1 + d_2$ where
d_1 = bottom $\lceil n/2 \rceil$ terms of
d_2 = remaining terms of d.

Equivalent keys

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

Search only about $3^n/n$ choices.

$n = 701$, $w = 467$:

$$\binom{n}{w}2^w \approx 2^{1106.09};$$
$$3^n \approx 2^{1111.06};$$
$$3^n/n \approx 2^{1101.61}.$$

Exercise: Find more equivalences!

But if w is chosen smaller then $\binom{n}{w}2^w$ search will be faster.

Collision attacks

Write d as $d_1 + d_2$ where
$d_1 =$ bottom $\lceil n/2 \rceil$ terms of d,
$d_2 =$ remaining terms of d.

Equivalent keys

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

Search only about $3^n/n$ choices.

$n = 701$, $w = 467$:

$$\binom{n}{w}2^w \approx 2^{1106.09};$$
$$3^n \approx 2^{1111.06};$$
$$3^n/n \approx 2^{1101.61}.$$

Exercise: Find more equivalences!

But if w is chosen smaller then $\binom{n}{w}2^w$ search will be faster.

Collision attacks

Write d as $d_1 + d_2$ where $d_1 =$ bottom $\lceil n/2 \rceil$ terms of d, $d_2 =$ remaining terms of d.

$a = (A/3)d = (A/3)d_1 + (A/3)d_2$ so $a - (A/3)d_2 = (A/3)d_1$.

Equivalent keys

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

Search only about $3^n/n$ choices.

$n = 701$, $w = 467$:

$$\binom{n}{w}2^w \approx 2^{1106.09};$$
$$3^n \approx 2^{1111.06};$$
$$3^n/n \approx 2^{1101.61}.$$

Exercise: Find more equivalences!

But if w is chosen smaller then $\binom{n}{w}2^w$ search will be faster.

Collision attacks

Write d as $d_1 + d_2$ where $d_1 =$ bottom $\lceil n/2 \rceil$ terms of d, $d_2 =$ remaining terms of d.

$a = (A/3)d = (A/3)d_1 + (A/3)d_2$ so $a - (A/3)d_2 = (A/3)d_1$. Eliminate a: almost certainly $H(-(A/3)d_2) = H((A/3)d_1)$ for $H(f) = ([f_0 < 0], \ldots, [f_{k-1} < 0])$.

Equivalent keys

Secret key (a, d) is equivalent to secret key (xa, xd), secret key $(x^2 a, x^2 d)$, etc.

Search only about $3^n/n$ choices.

$n = 701$, $w = 467$:

$$\binom{n}{w} 2^w \approx 2^{1106.09};$$
$$3^n \approx 2^{1111.06};$$
$$3^n/n \approx 2^{1101.61}.$$

Exercise: Find more equivalences!

But if w is chosen smaller then $\binom{n}{w} 2^w$ search will be faster.

Collision attacks

Write d as $d_1 + d_2$ where $d_1 =$ bottom $\lceil n/2 \rceil$ terms of d, $d_2 =$ remaining terms of d.

$a = (A/3)d = (A/3)d_1 + (A/3)d_2$ so $a - (A/3)d_2 = (A/3)d_1$.
Eliminate a: almost certainly $H(-(A/3)d_2) = H((A/3)d_1)$ for $H(f) = ([f_0 < 0], \ldots, [f_{k-1} < 0])$.

Enumerate all $H(-(A/3)d_2)$.
Enumerate all $H((A/3)d_1)$.
Search for collisions.
Only about $3^{n/2}$ computations; but beware cost of memory.

ent keys

ey (a, d) is equivalent to
ey (xa, xd),
ey (x^2a, x^2d), etc.

only about $3^n/n$ choices.

, $w = 467$:

$$\binom{n}{w}2^w \approx 2^{1106.09};$$
$$3^n \approx 2^{1111.06};$$
$$3^n/n \approx 2^{1101.61}.$$

: Find more equivalences!

is chosen smaller then
earch will be faster.

Collision attacks

Write d as $d_1 + d_2$ where
d_1 = bottom $\lceil n/2 \rceil$ terms of d,
d_2 = remaining terms of d.

$a = (A/3)d = (A/3)d_1 + (A/3)d_2$
so $a - (A/3)d_2 = (A/3)d_1$.
Eliminate a: almost certainly
$H(-(A/3)d_2) = H((A/3)d_1)$ for
$H(f) = ([f_0 < 0], \ldots, [f_{k-1} < 0])$.

Enumerate all $H(-(A/3)d_2)$.
Enumerate all $H((A/3)d_1)$.
Search for collisions.
Only about $3^{n/2}$ computations;
but beware cost of memory.

Lattices

s equivalent to
),
d), etc.

$3^n/n$ choices.

:
$)2^w \approx 2^{1106.09}$;
$3^n \approx 2^{1111.06}$;
$3^n/n \approx 2^{1101.61}$.

re equivalences!

smaller then
be faster.

Collision attacks

Write d as $d_1 + d_2$ where
$d_1 =$ bottom $\lceil n/2 \rceil$ terms of d,
$d_2 =$ remaining terms of d.

$a = (A/3)d = (A/3)d_1 + (A/3)d_2$
so $a - (A/3)d_2 = (A/3)d_1$.
Eliminate a: almost certainly
$H(-(A/3)d_2) = H((A/3)d_1)$ for
$H(f) = ([f_0 < 0], \ldots, [f_{k-1} < 0])$.

Enumerate all $H(-(A/3)d_2)$.
Enumerate all $H((A/3)d_1)$.
Search for collisions.
Only about $3^{n/2}$ computations;
but beware cost of memory.

Lattices

nt to

ces.

1106.09;

1111.06;

1101.61.

ences!

hen

Collision attacks

Write d as $d_1 + d_2$ where
d_1 = bottom $\lceil n/2 \rceil$ terms of d,
d_2 = remaining terms of d.

$a = (A/3)d = (A/3)d_1 + (A/3)d_2$
so $a - (A/3)d_2 = (A/3)d_1$.
Eliminate a: almost certainly
$H(-(A/3)d_2) = H((A/3)d_1)$ for
$H(f) = ([f_0 < 0], \ldots, [f_{k-1} < 0])$.

Enumerate all $H(-(A/3)d_2)$.
Enumerate all $H((A/3)d_1)$.
Search for collisions.
Only about $3^{n/2}$ computations;
but beware cost of memory.

Lattices

Collision attacks

Write d as $d_1 + d_2$ where
$d_1 =$ bottom $\lceil n/2 \rceil$ terms of d,
$d_2 =$ remaining terms of d.

$a = (A/3)d = (A/3)d_1 + (A/3)d_2$
so $a - (A/3)d_2 = (A/3)d_1$.
Eliminate a: almost certainly
$H(-(A/3)d_2) = H((A/3)d_1)$ for
$H(f) = ([f_0 < 0], \ldots, [f_{k-1} < 0])$.

Enumerate all $H(-(A/3)d_2)$.
Enumerate all $H((A/3)d_1)$.
Search for collisions.
Only about $3^{n/2}$ computations;
but beware cost of memory.

Lattices

Collision attacks

Write d as $d_1 + d_2$ where
d_1 = bottom $\lceil n/2 \rceil$ terms of d,
d_2 = remaining terms of d.

$a = (A/3)d = (A/3)d_1 + (A/3)d_2$
so $a - (A/3)d_2 = (A/3)d_1$.
Eliminate a: almost certainly
$H(-(A/3)d_2) = H((A/3)d_1)$ for
$H(f) = ([f_0 < 0], \ldots, [f_{k-1} < 0])$.

Enumerate all $H(-(A/3)d_2)$.
Enumerate all $H((A/3)d_1)$.
Search for collisions.
Only about $3^{n/2}$ computations;
but beware cost of memory.

Lattices

This is a lettuce:

Collision attacks

Write d as $d_1 + d_2$ where
d_1 = bottom $\lceil n/2 \rceil$ terms of d,
d_2 = remaining terms of d.

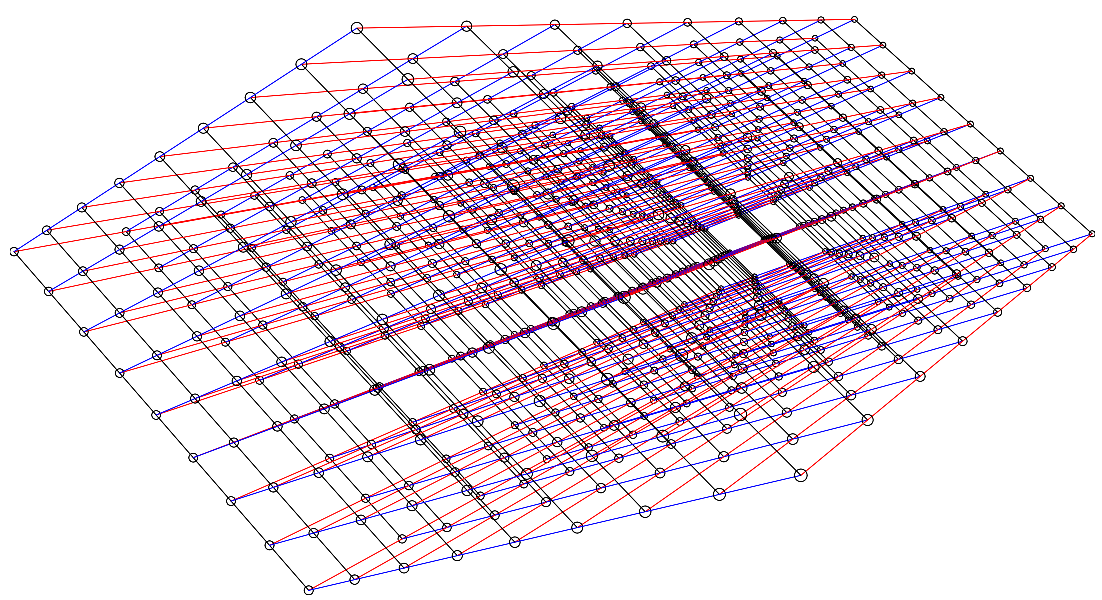
$a = (A/3)d = (A/3)d_1 + (A/3)d_2$
so $a - (A/3)d_2 = (A/3)d_1$.
Eliminate a: almost certainly
$H(-(A/3)d_2) = H((A/3)d_1)$ for
$H(f) = ([f_0 < 0], \ldots, [f_{k-1} < 0])$.

Enumerate all $H(-(A/3)d_2)$.
Enumerate all $H((A/3)d_1)$.
Search for collisions.
Only about $3^{n/2}$ computations;
but beware cost of memory.

Lattices

This is a lettuce:

This is a lattice:



attacks

as $d_1 + d_2$ where
ttom $\lceil n/2 \rceil$ terms of d,
maining terms of d.

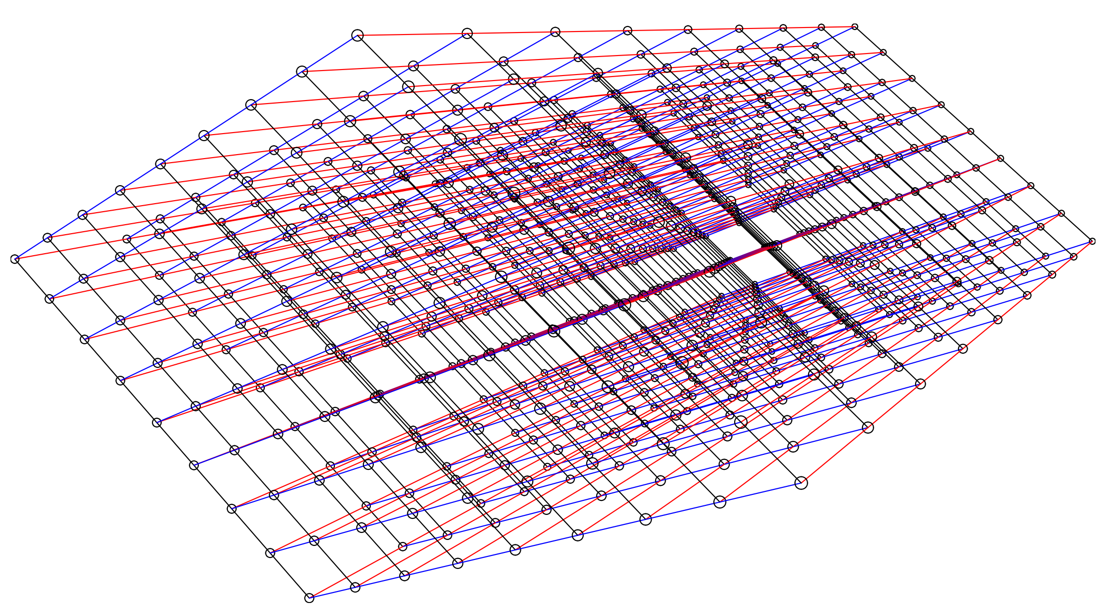
$3)d = (A/3)d_1 + (A/3)d_2$
$A/3)d_2 = (A/3)d_1$.
e a: almost certainly
$3)d_2) = H((A/3)d_1)$ for
$([f_0 < 0], \ldots, [f_{k-1} < 0])$.

ate all $H(-(A/3)d_2)$.
ate all $H((A/3)d_1)$.
or collisions.
out $3^{n/2}$ computations;
are cost of memory.

Lattices

This is a lettuce:

This is a lattice:



Lattices,

Assume
are **R**-lin
i.e., $\mathbf{R}b_1$
$\{r_1 b_1 +$
is a k-di

$_2$ where

$2\rceil$ terms of d,

rms of d.

$/3)d_1 + (A/3)d_2$

$(A/3)d_1$.

st certainly

$H((A/3)d_1)$ for

$\ldots, [f_{k-1} < 0])$.

$-(A/3)d_2)$.

$(A/3)d_1)$.

ns.

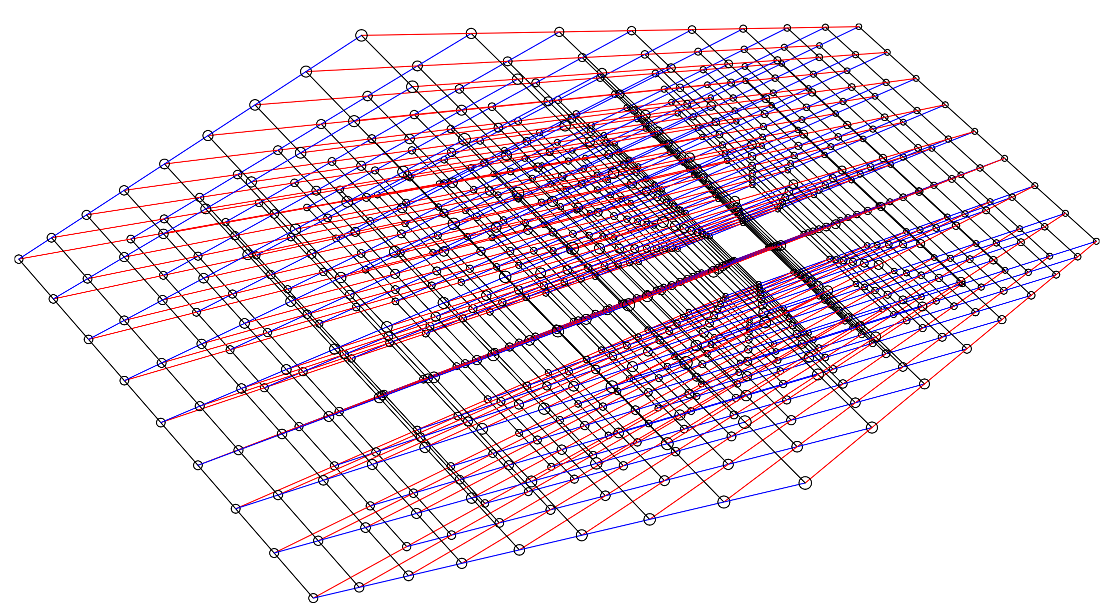
omputations;

f memory.

Lattices

This is a lettuce:

This is a lattice:



Lattices, mathema

Assume that b_1, b

are **R**-linearly inde

i.e., $\mathbf{R}b_1 + \ldots + \mathbf{R}$

$\{r_1 b_1 + \ldots + r_k b_k$

is a k-dimensional

f d,

$/3)d_2$

y

) for

$< 0]$).

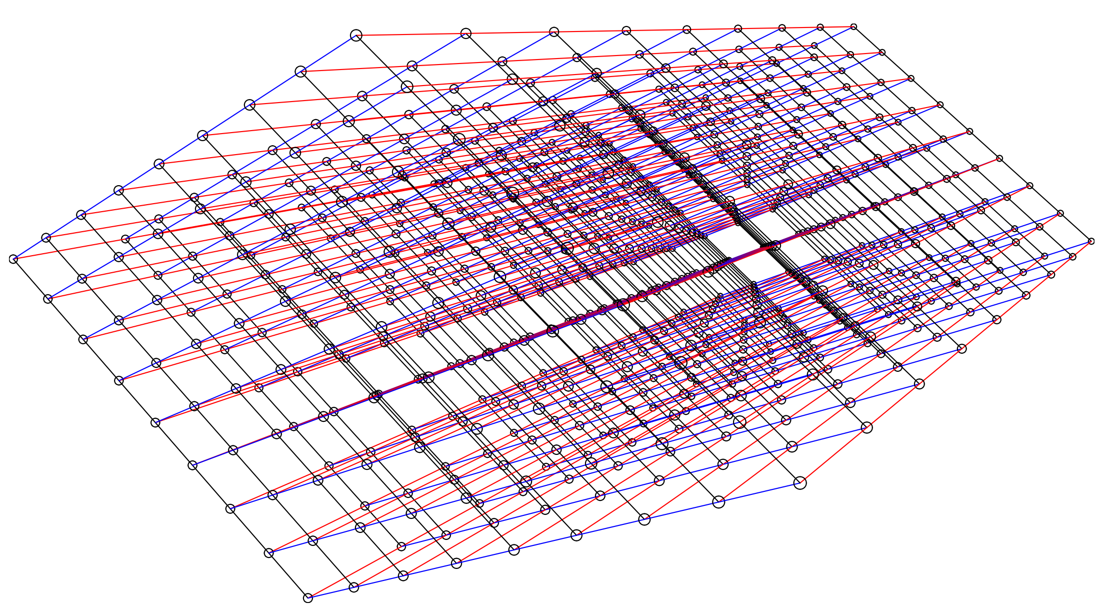
).

ons;

Lattices

This is a lettuce:

This is a lattice:



Lattices, mathematically

Assume that $b_1, b_2, \ldots, b_k$

are $\mathbf{R}$-linearly independent,

i.e., $\mathbf{R}b_1 + \ldots + \mathbf{R}b_k =$

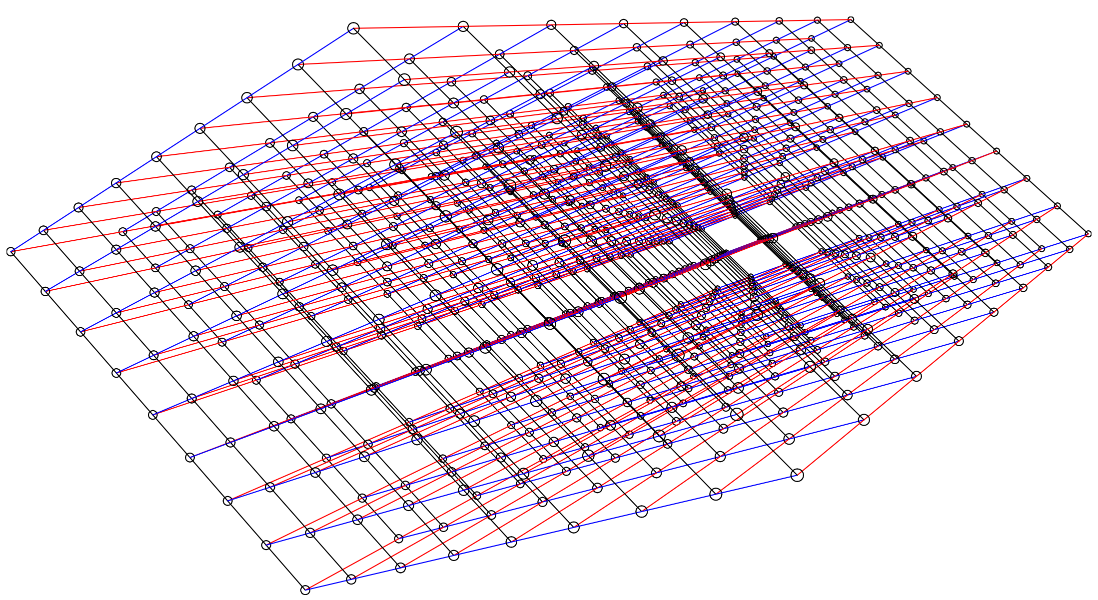
$\{r_1 b_1 + \ldots + r_k b_k : r_1, \ldots,$

is a k-dimensional vector spa

Lattices

This is a lettuce:

This is a lattice:



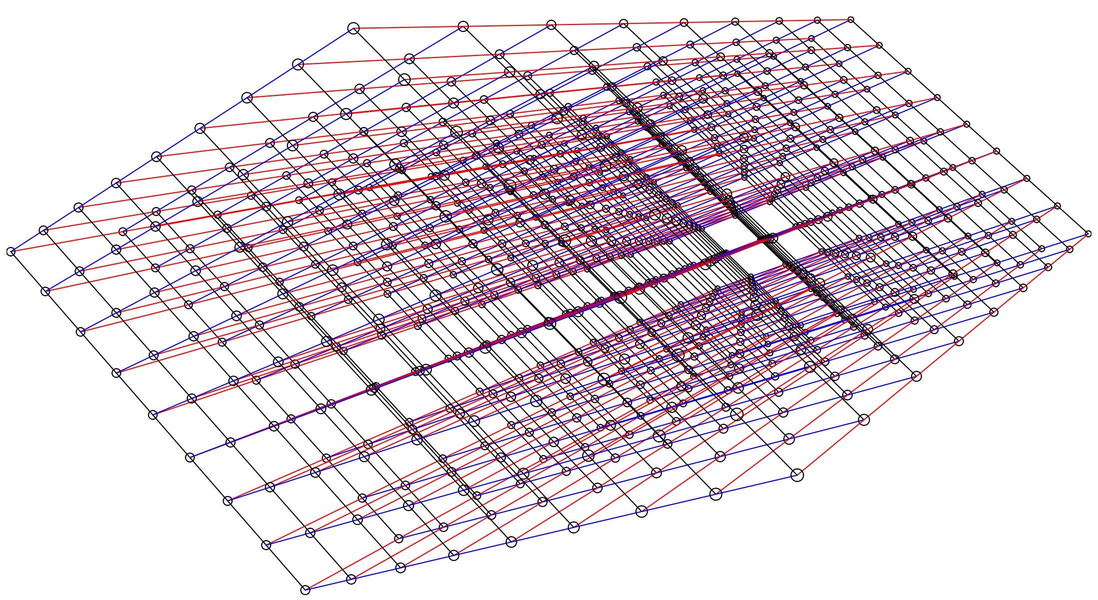
Lattices, mathematically

Assume that $b_1, b_2, \ldots, b_k \in \mathbf{R}^n$ are $\mathbf{R}$-linearly independent, i.e., $\mathbf{R}b_1 + \ldots + \mathbf{R}b_k = \{r_1 b_1 + \ldots + r_k b_k : r_1, \ldots, r_k \in \mathbf{R}\}$ is a k-dimensional vector space.

Lattices

This is a lettuce:

This is a lattice:



Lattices, mathematically

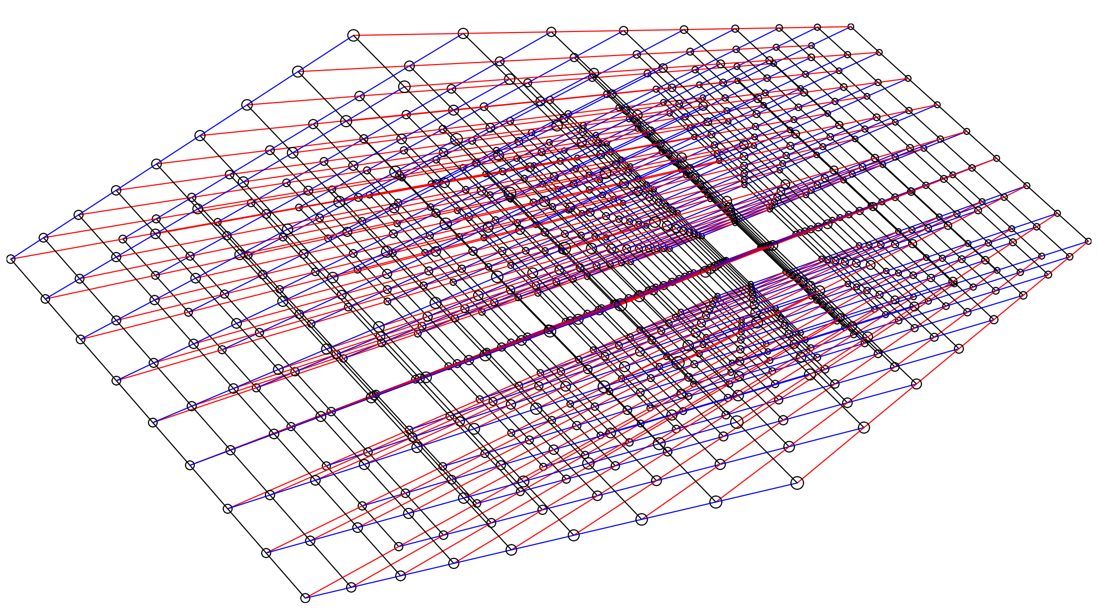
Assume that $b_1, b_2, \ldots, b_k \in \mathbf{R}^n$ are $\mathbf{R}$-linearly independent, i.e., $\mathbf{R}b_1 + \ldots + \mathbf{R}b_k = \{r_1 b_1 + \ldots + r_k b_k : r_1, \ldots, r_k \in \mathbf{R}\}$ is a k-dimensional vector space.

$\mathbf{Z}b_1 + \ldots + \mathbf{Z}b_k = \{r_1 b_1 + \ldots + r_k b_k : r_1, \ldots, r_k \in \mathbf{Z}\}$ is a rank-k length-n **lattice**.

Lattices

This is a lettuce:

This is a lattice:



Lattices, mathematically

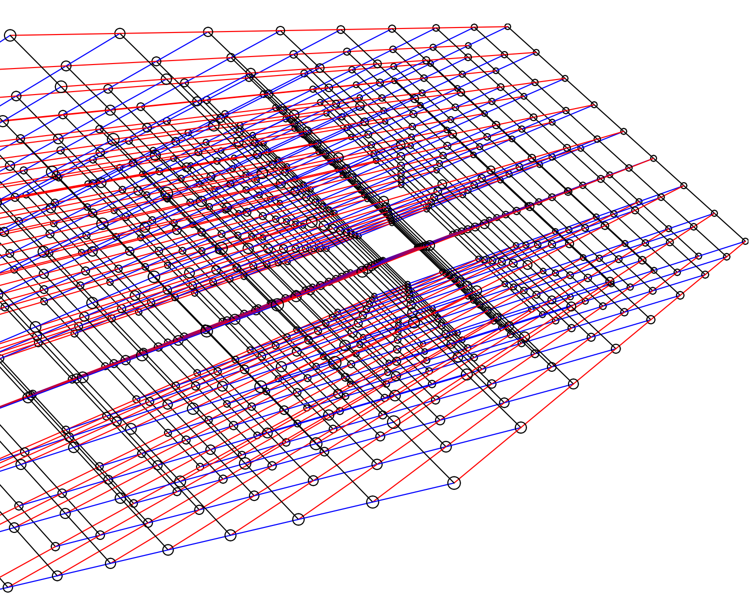
Assume that $b_1, b_2, \ldots, b_k \in \mathbf{R}^n$
are **R**-linearly independent,
i.e., $\mathbf{R}b_1 + \ldots + \mathbf{R}b_k =$
$\{r_1 b_1 + \ldots + r_k b_k : r_1, \ldots, r_k \in \mathbf{R}\}$
is a k-dimensional vector space.

$\mathbf{Z}b_1 + \ldots + \mathbf{Z}b_k =$
$\{r_1 b_1 + \ldots + r_k b_k : r_1, \ldots, r_k \in \mathbf{Z}\}$
is a rank-k length-n **lattice**.

$b_1, \ldots, b_k$
is a **basis** of this lattice.

lettuce:

lattice:



Lattices, mathematically

Assume that $b_1, b_2, \ldots, b_k \in \mathbf{R}^n$ are $\mathbf{R}$-linearly independent, i.e., $\mathbf{R}b_1 + \ldots + \mathbf{R}b_k =$ $\{r_1 b_1 + \ldots + r_k b_k : r_1, \ldots, r_k \in \mathbf{R}\}$ is a k-dimensional vector space.

$\mathbf{Z}b_1 + \ldots + \mathbf{Z}b_k =$ $\{r_1 b_1 + \ldots + r_k b_k : r_1, \ldots, r_k \in \mathbf{Z}\}$ is a rank-k length-n **lattice**.

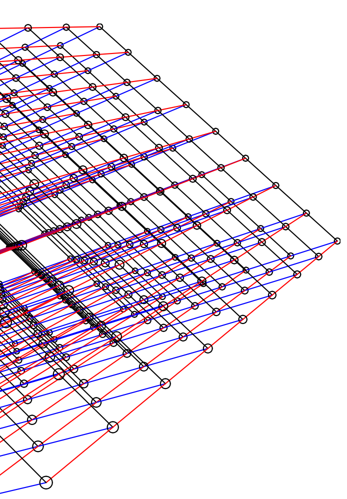
$b_1, \ldots, b_k$ is a **basis** of this lattice.

Short ve

Given b_1

what is s

in $\mathbf{Z}b_1 +$



Lattices, mathematically

Assume that $b_1, b_2, \ldots, b_k \in \mathbf{R}^n$
are $\mathbf{R}$-linearly independent,
i.e., $\mathbf{R}b_1 + \ldots + \mathbf{R}b_k =$
$\{r_1 b_1 + \ldots + r_k b_k : r_1, \ldots, r_k \in \mathbf{R}\}$
is a k-dimensional vector space.

$\mathbf{Z}b_1 + \ldots + \mathbf{Z}b_k =$
$\{r_1 b_1 + \ldots + r_k b_k : r_1, \ldots, r_k \in \mathbf{Z}\}$
is a rank-k length-n **lattice**.

$b_1, \ldots, b_k$
is a **basis** of this lattice.

Short vectors in la

Given $b_1, b_2, \ldots,$
what is shortest ve
in $\mathbf{Z}b_1 + \ldots + \mathbf{Z}b$

Lattices, mathematically

Assume that $b_1, b_2, \ldots, b_k \in \mathbf{R}^n$
are $\mathbf{R}$-linearly independent,
i.e., $\mathbf{R}b_1 + \ldots + \mathbf{R}b_k =$
$\{r_1 b_1 + \ldots + r_k b_k : r_1, \ldots, r_k \in \mathbf{R}\}$
is a k-dimensional vector space.

$\mathbf{Z}b_1 + \ldots + \mathbf{Z}b_k =$
$\{r_1 b_1 + \ldots + r_k b_k : r_1, \ldots, r_k \in \mathbf{Z}\}$
is a rank-k length-n **lattice**.

$b_1, \ldots, b_k$
is a **basis** of this lattice.

Short vectors in lattices

Given $b_1, b_2, \ldots, b_k \in \mathbf{Z}^n$,
what is shortest vector
in $\mathbf{Z}b_1 + \ldots + \mathbf{Z}b_k$?

Lattices, mathematically

Assume that $b_1, b_2, \ldots, b_k \in \mathbf{R}^n$
are $\mathbf{R}$-linearly independent,
i.e., $\mathbf{R}b_1 + \ldots + \mathbf{R}b_k =$
$\{r_1 b_1 + \ldots + r_k b_k : r_1, \ldots, r_k \in \mathbf{R}\}$
is a k-dimensional vector space.

$\mathbf{Z}b_1 + \ldots + \mathbf{Z}b_k =$
$\{r_1 b_1 + \ldots + r_k b_k : r_1, \ldots, r_k \in \mathbf{Z}\}$
is a rank-k length-n **lattice**.

$b_1, \ldots, b_k$
is a **basis** of this lattice.

Short vectors in lattices

Given $b_1, b_2, \ldots, b_k \in \mathbf{Z}^n$,
what is shortest vector
in $\mathbf{Z}b_1 + \ldots + \mathbf{Z}b_k$?

Lattices, mathematically

Assume that $b_1, b_2, \ldots, b_k \in \mathbf{R}^n$
are $\mathbf{R}$-linearly independent,
i.e., $\mathbf{R}b_1 + \ldots + \mathbf{R}b_k =$
$\{r_1 b_1 + \ldots + r_k b_k : r_1, \ldots, r_k \in \mathbf{R}\}$
is a k-dimensional vector space.

$\mathbf{Z}b_1 + \ldots + \mathbf{Z}b_k =$
$\{r_1 b_1 + \ldots + r_k b_k : r_1, \ldots, r_k \in \mathbf{Z}\}$
is a rank-k length-n **lattice**.

$b_1, \ldots, b_k$
is a **basis** of this lattice.

Short vectors in lattices

Given $b_1, b_2, \ldots, b_k \in \mathbf{Z}^n$,
what is shortest vector
in $\mathbf{Z}b_1 + \ldots + \mathbf{Z}b_k$?

0.

Lattices, mathematically

Assume that $b_1, b_2, \ldots, b_k \in \mathbf{R}^n$
are $\mathbf{R}$-linearly independent,
i.e., $\mathbf{R}b_1 + \ldots + \mathbf{R}b_k =$
$\{r_1 b_1 + \ldots + r_k b_k : r_1, \ldots, r_k \in \mathbf{R}\}$
is a k-dimensional vector space.

$\mathbf{Z}b_1 + \ldots + \mathbf{Z}b_k =$
$\{r_1 b_1 + \ldots + r_k b_k : r_1, \ldots, r_k \in \mathbf{Z}\}$
is a rank-k length-n **lattice**.

$b_1, \ldots, b_k$
is a **basis** of this lattice.

Short vectors in lattices

Given $b_1, b_2, \ldots, b_k \in \mathbf{Z}^n$,
what is shortest vector
in $\mathbf{Z}b_1 + \ldots + \mathbf{Z}b_k$?

0.

What is shortest nonzero vector?

Lattices, mathematically

Assume that $b_1, b_2, \ldots, b_k \in \mathbf{R}^n$
are **R**-linearly independent,
i.e., $\mathbf{R}b_1 + \ldots + \mathbf{R}b_k =$
$\{r_1 b_1 + \ldots + r_k b_k : r_1, \ldots, r_k \in \mathbf{R}\}$
is a k-dimensional vector space.

$\mathbf{Z}b_1 + \ldots + \mathbf{Z}b_k =$
$\{r_1 b_1 + \ldots + r_k b_k : r_1, \ldots, r_k \in \mathbf{Z}\}$
is a rank-k length-n **lattice**.

$b_1, \ldots, b_k$
is a **basis** of this lattice.

Short vectors in lattices

Given $b_1, b_2, \ldots, b_k \in \mathbf{Z}^n$,
what is shortest vector
in $\mathbf{Z}b_1 + \ldots + \mathbf{Z}b_k$?

0.

What is shortest nonzero vector?

LLL algorithm runs in poly time,
computes a vector whose length
is at most $2^{n/2}$ times
length of shortest nonzero vector.

Lattices, mathematically

Assume that $b_1, b_2, \ldots, b_k \in \mathbf{R}^n$ are $\mathbf{R}$-linearly independent, i.e., $\mathbf{R}b_1 + \ldots + \mathbf{R}b_k = \{r_1 b_1 + \ldots + r_k b_k : r_1, \ldots, r_k \in \mathbf{R}\}$ is a k-dimensional vector space.

$\mathbf{Z}b_1 + \ldots + \mathbf{Z}b_k = \{r_1 b_1 + \ldots + r_k b_k : r_1, \ldots, r_k \in \mathbf{Z}\}$ is a rank-k length-n **lattice**.

$b_1, \ldots, b_k$ is a **basis** of this lattice.

Short vectors in lattices

Given $b_1, b_2, \ldots, b_k \in \mathbf{Z}^n$, what is shortest vector in $\mathbf{Z}b_1 + \ldots + \mathbf{Z}b_k$?

0.

What is shortest nonzero vector?

LLL algorithm runs in poly time, computes a vector whose length is at most $2^{n/2}$ times length of shortest nonzero vector.

Fancier algorithms (e.g., BKZ) compute shorter vectors at surprisingly high speed.

mathematically

that $b_1, b_2, \dots, b_k \in \mathbf{R}^n$
nearly independent,
$+ \dots + \mathbf{R} b_k =$
$\dots + r_k b_k : r_1, \dots, r_k \in \mathbf{R}\}$
mensional vector space.

$\dots + \mathbf{Z} b_k =$
$\dots + r_k b_k : r_1, \dots, r_k \in \mathbf{Z}\}$
-k length-n **lattice**.

b_k
is of this lattice.

Short vectors in lattices

Given $b_1, b_2, \dots, b_k \in \mathbf{Z}^n$,
what is shortest vector
in $\mathbf{Z} b_1 + \dots + \mathbf{Z} b_k$?

0.

What is shortest nonzero vector?

LLL algorithm runs in poly time,
computes a vector whose length
is at most $2^{n/2}$ times
length of shortest nonzero vector.

Fancier algorithms (e.g., BKZ)
compute shorter vectors
at surprisingly high speed.

Lattice v

Given pu
Comput

tically

$_2, \ldots, b_k \in \mathbf{R}^n$

pendent,

$\mathbf{R}b_k =$

$_k : r_1, \ldots, r_k \in \mathbf{R}\}$

vector space.

$=$

$_k : r_1, \ldots, r_k \in \mathbf{Z}\}$

-n **lattice**.

attice.

Short vectors in lattices

Given $b_1, b_2, \ldots, b_k \in \mathbf{Z}^n$,
what is shortest vector
in $\mathbf{Z}b_1 + \ldots + \mathbf{Z}b_k$?

0.

What is shortest nonzero vector?

LLL algorithm runs in poly time,
computes a vector whose length
is at most $2^{n/2}$ times
length of shortest nonzero vector.

Fancier algorithms (e.g., BKZ)
compute shorter vectors
at surprisingly high speed.

Lattice view of NT

Given public key A

Compute $A/3 = a$

$\in \mathbf{R}^n$

$r_k \in \mathbf{R}\}$
ace.

$r_k \in \mathbf{Z}\}$

Short vectors in lattices

Given $b_1, b_2, \ldots, b_k \in \mathbf{Z}^n$,
what is shortest vector
in $\mathbf{Z}b_1 + \ldots + \mathbf{Z}b_k$?

0.

What is shortest nonzero vector?

LLL algorithm runs in poly time,
computes a vector whose length
is at most $2^{n/2}$ times
length of shortest nonzero vector.

Fancier algorithms (e.g., BKZ)
compute shorter vectors
at surprisingly high speed.

Lattice view of NTRU

Given public key $A = 3a/d$.
Compute $A/3 = a/d$.

Short vectors in lattices

Given $b_1, b_2, \ldots, b_k \in \mathbf{Z}^n$,
what is shortest vector
in $\mathbf{Z}b_1 + \ldots + \mathbf{Z}b_k$?

0.

What is shortest nonzero vector?

LLL algorithm runs in poly time, computes a vector whose length is at most $2^{n/2}$ times length of shortest nonzero vector.

Fancier algorithms (e.g., BKZ) compute shorter vectors at surprisingly high speed.

Lattice view of NTRU

Given public key $A = 3a/d$.
Compute $A/3 = a/d$.

Short vectors in lattices

Given $b_1, b_2, \ldots, b_k \in \mathbf{Z}^n$,
what is shortest vector
in $\mathbf{Z}b_1 + \ldots + \mathbf{Z}b_k$?

0.

What is shortest nonzero vector?

LLL algorithm runs in poly time, computes a vector whose length is at most $2^{n/2}$ times length of shortest nonzero vector.

Fancier algorithms (e.g., BKZ) compute shorter vectors at surprisingly high speed.

Lattice view of NTRU

Given public key $A = 3a/d$.
Compute $A/3 = a/d$.

d is obtained from
$1, x, \ldots, x^{n-1}$
by a few additions, subtractions.

Short vectors in lattices

Given $b_1, b_2, \ldots, b_k \in \mathbf{Z}^n$,
what is shortest vector
in $\mathbf{Z}b_1 + \ldots + \mathbf{Z}b_k$?

0.

What is shortest nonzero vector?

LLL algorithm runs in poly time,
computes a vector whose length
is at most $2^{n/2}$ times
length of shortest nonzero vector.

Fancier algorithms (e.g., BKZ)
compute shorter vectors
at surprisingly high speed.

Lattice view of NTRU

Given public key $A = 3a/d$.
Compute $A/3 = a/d$.

d is obtained from
$1, x, \ldots, x^{n-1}$
by a few additions, subtractions.

$d(A/3)$ is obtained from
$A/3, xA/3, \ldots, x^{n-1}A/3$
by a few additions, subtractions.

Short vectors in lattices

Given $b_1, b_2, \ldots, b_k \in \mathbf{Z}^n$,
what is shortest vector
in $\mathbf{Z}b_1 + \ldots + \mathbf{Z}b_k$?

0.

What is shortest nonzero vector?

LLL algorithm runs in poly time,
computes a vector whose length
is at most $2^{n/2}$ times
length of shortest nonzero vector.

Fancier algorithms (e.g., BKZ)
compute shorter vectors
at surprisingly high speed.

Lattice view of NTRU

Given public key $A = 3a/d$.
Compute $A/3 = a/d$.

d is obtained from
$1, x, \ldots, x^{n-1}$
by a few additions, subtractions.

$d(A/3)$ is obtained from
$A/3, xA/3, \ldots, x^{n-1}A/3$
by a few additions, subtractions.

a is obtained from
$q, qx, qx^2, \ldots, qx^{n-1}$,
$A/3, xA/3, \ldots, x^{n-1}A/3$
by a few additions, subtractions.

ectors in lattices

, $b_2, \ldots, b_k \in \mathbf{Z}^n$,
shortest vector
$+ \ldots + \mathbf{Z} b_k$?

shortest nonzero vector?

orithm runs in poly time,
es a vector whose length
st $2^{n/2}$ times
f shortest nonzero vector.

algorithms (e.g., BKZ)
e shorter vectors
singly high speed.

Lattice view of NTRU

Given public key $A = 3a/d$.
Compute $A/3 = a/d$.

d is obtained from
$1, x, \ldots, x^{n-1}$
by a few additions, subtractions.

$d(A/3)$ is obtained from
$A/3, xA/3, \ldots, x^{n-1}A/3$
by a few additions, subtractions.

a is obtained from
$q, qx, qx^2, \ldots, qx^{n-1}$,
$A/3, xA/3, \ldots, x^{n-1}A/3$
by a few additions, subtractions.

(a, d) is
$(q, 0)$,
$(qx, 0)$,
$\vdots$
$(qx^{n-1},$
$(A/3, 1)$
$(xA/3, x$
$\vdots$
$(x^{n-1}A/$
by a few

ttices

$b_k \in \mathbf{Z}^n$,
ector
$_k$?

onzero vector?

s in poly time,
whose length
mes
nonzero vector.

s (e.g., BKZ)
ectors
n speed.

Lattice view of NTRU

Given public key $A = 3a/d$.
Compute $A/3 = a/d$.

d is obtained from
$1, x, \ldots, x^{n-1}$
by a few additions, subtractions.

$d(A/3)$ is obtained from
$A/3, xA/3, \ldots, x^{n-1}A/3$
by a few additions, subtractions.

a is obtained from
$q, qx, qx^2, \ldots, qx^{n-1}$,
$A/3, xA/3, \ldots, x^{n-1}A/3$
by a few additions, subtractions.

(a, d) is obtained
$(q, 0)$,
$(qx, 0)$,
$\vdots$
$(qx^{n-1}, 0)$,
$(A/3, 1)$,
$(xA/3, x)$,
$\vdots$
$(x^{n-1}A/3, x^{n-1})$
by a few additions

ctor?

time,

ngth

ector.

Z)

Lattice view of NTRU

Given public key $A = 3a/d$.
Compute $A/3 = a/d$.

d is obtained from
$1, x, \ldots, x^{n-1}$
by a few additions, subtractions.

$d(A/3)$ is obtained from
$A/3, xA/3, \ldots, x^{n-1}A/3$
by a few additions, subtractions.

a is obtained from
$q, qx, qx^2, \ldots, qx^{n-1}$,
$A/3, xA/3, \ldots, x^{n-1}A/3$
by a few additions, subtractions.

(a, d) is obtained from
$(q, 0)$,
$(qx, 0)$,
$\vdots$
$(qx^{n-1}, 0)$,
$(A/3, 1)$,
$(xA/3, x)$,
$\vdots$
$(x^{n-1}A/3, x^{n-1})$
by a few additions, subtracti

Lattice view of NTRU

Given public key $A = 3a/d$.
Compute $A/3 = a/d$.

d is obtained from
$1, x, \ldots, x^{n-1}$
by a few additions, subtractions.

$d(A/3)$ is obtained from
$A/3, xA/3, \ldots, x^{n-1}A/3$
by a few additions, subtractions.

a is obtained from
$q, qx, qx^2, \ldots, qx^{n-1}$,
$A/3, xA/3, \ldots, x^{n-1}A/3$
by a few additions, subtractions.

(a, d) is obtained from
$(q, 0)$,
$(qx, 0)$,
$\vdots$
$(qx^{n-1}, 0)$,
$(A/3, 1)$,
$(xA/3, x)$,
$\vdots$
$(x^{n-1}A/3, x^{n-1})$
by a few additions, subtractions.

Lattice view of NTRU

Given public key $A = 3a/d$.
Compute $A/3 = a/d$.

d is obtained from
$1, x, \ldots, x^{n-1}$
by a few additions, subtractions.

$d(A/3)$ is obtained from
$A/3, xA/3, \ldots, x^{n-1}A/3$
by a few additions, subtractions.

a is obtained from
$q, qx, qx^2, \ldots, qx^{n-1}$,
$A/3, xA/3, \ldots, x^{n-1}A/3$
by a few additions, subtractions.

(a, d) is obtained from
$(q, 0)$,
$(qx, 0)$,
$\vdots$
$(qx^{n-1}, 0)$,
$(A/3, 1)$,
$(xA/3, x)$,
$\vdots$
$(x^{n-1}A/3, x^{n-1})$
by a few additions, subtractions.

Write $A/3$ as
$H_0 + H_1x + \ldots + H_{n-1}x^{n-1}$.

view of NTRU

ublic key $A = 3a/d$.

e $A/3 = a/d$.

ained from

$, x^{n-1}$

additions, subtractions.

is obtained from

$/3, \ldots, x^{n-1}A/3$

additions, subtractions.

ained from

$x^2, \ldots, qx^{n-1}$,

$/3, \ldots, x^{n-1}A/3$

additions, subtractions.

(a, d) is obtained from

$(q, 0)$,

$(qx, 0)$,

$\vdots$

$(qx^{n-1}, 0)$,

$(A/3, 1)$,

$(xA/3, x)$,

$\vdots$

$(x^{n-1}A/3, x^{n-1})$

by a few additions, subtractions.

Write $A/3$ as

$H_0 + H_1 x + \ldots + H_{n-1} x^{n-1}$.

$(a_0, a_1, \ldots$

is obtain

$(q, 0, \ldots$

$(0, q, \ldots$

$\vdots$

$(0, 0, \ldots$

$(H_0, H_1,$

$(H_{n-1}, H$

$\vdots$

$(H_1, H_2,$

by a few

TRU

$A = 3a/d$.

$/d$.

n

, subtractions.

d from

$^{-1}A/3$

, subtractions.

$^{n-1}$,

$^{-1}A/3$

, subtractions.

(a, d) is obtained from

$(q, 0)$,

$(qx, 0)$,

$\vdots$

$(qx^{n-1}, 0)$,

$(A/3, 1)$,

$(xA/3, x)$,

$\vdots$

$(x^{n-1}A/3, x^{n-1})$

by a few additions, subtractions.

Write $A/3$ as

$H_0 + H_1 x + \ldots + H_{n-1}x^{n-1}$.

$(a_0, a_1, \ldots, a_{n-1},$

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

ions.

ions.

ions.

(a, d) is obtained from

$(q, 0)$,

$(qx, 0)$,

$\vdots$

$(qx^{n-1}, 0)$,

$(A/3, 1)$,

$(xA/3, x)$,

$\vdots$

$(x^{n-1}A/3, x^{n-1})$

by a few additions, subtractions.

Write $A/3$ as

$H_0 + H_1x + \ldots + H_{n-1}x^{n-1}$.

$(a_0, a_1, \ldots, a_{n-1}, d_0, d_1, \ldots,$

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

(a, d) is obtained from

$(q, 0)$,

$(qx, 0)$,

$\vdots$

$(qx^{n-1}, 0)$,

$(A/3, 1)$,

$(xA/3, x)$,

$\vdots$

$(x^{n-1}A/3, x^{n-1})$

by a few additions, subtractions.

Write $A/3$ as

$H_0 + H_1 x + \ldots + H_{n-1} x^{n-1}$.

$(a_0, a_1, \ldots, a_{n-1}, d_0, d_1, \ldots, d_{n-1})$

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.

obtained from

$0)$,

,

$x)$,

$/3, x^{n-1})$

additions, subtractions.

$/3$ as

$x + \ldots + H_{n-1}x^{n-1}$.

$(a_0, a_1, \ldots, a_{n-1}, d_0, d_1, \ldots, d_{n-1})$

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.

$(a_0, a_1, \ldots$

is a surp

in lattic

$(q, 0, \ldots$

from

, subtractions.

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

$(a_0, a_1, \ldots, a_{n-1}, d_0, d_1, \ldots, d_{n-1})$
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.

$(a_0, a_1, \ldots, a_{n-1},$
is a surprisingly sh
in lattice generate
$(q, 0, \ldots, 0, 0, 0, \ldots$

ions.

1
.

$(a_0, a_1, \ldots, a_{n-1}, d_0, d_1, \ldots, d_{n-1})$
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.

$(a_0, a_1, \ldots, a_{n-1}, d_0, d_1, \ldots,$
is a surprisingly short vector
in lattice generated by
$(q, 0, \ldots, 0, 0, 0, \ldots, 0)$ etc.

$(a_0, a_1, \ldots, a_{n-1}, d_0, d_1, \ldots, d_{n-1})$
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.

$(a_0, a_1, \ldots, a_{n-1}, d_0, d_1, \ldots, d_{n-1})$
is a surprisingly short vector
in lattice generated by
$(q, 0, \ldots, 0, 0, 0, \ldots, 0)$ etc.

$(a_0, a_1, \ldots, a_{n-1}, d_0, d_1, \ldots, d_{n-1})$
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.

$(a_0, a_1, \ldots, a_{n-1}, d_0, d_1, \ldots, d_{n-1})$
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 LLL etc.

$(a_0, a_1, \ldots, a_{n-1}, d_0, d_1, \ldots, d_{n-1})$
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.

$(a_0, a_1, \ldots, a_{n-1}, d_0, d_1, \ldots, d_{n-1})$
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 LLL etc.

1997 Coppersmith–Shamir
balancing: e.g., set up lattice
to contain $(10a, d)$
if d is chosen $10\times$ larger than a.

$(a_0, a_1, \ldots, a_{n-1}, d_0, d_1, \ldots, d_{n-1})$
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.

$(a_0, a_1, \ldots, a_{n-1}, d_0, d_1, \ldots, d_{n-1})$
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 LLL etc.

1997 Coppersmith–Shamir
balancing: e.g., set up lattice
to contain $(10a, d)$
if d is chosen $10\times$ larger than a.

Exercise: Describe search for
(b, c) as a problem of finding
a vector close to a lattice.

$\ldots, a_{n-1}, d_0, d_1, \ldots, d_{n-1})$
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.

$(a_0, a_1, \ldots, a_{n-1}, d_0, d_1, \ldots, d_{n-1})$
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 LLL etc.

1997 Coppersmith–Shamir balancing: e.g., set up lattice to contain $(10a, d)$ if d is chosen $10\times$ larger than a.

Exercise: Describe search for (b, c) as a problem of finding a vector close to a lattice.

Quotient

"Quotie
is the st

Alice ge
for smal
i.e., dA

$d_0, d_1, \ldots, d_{n-1})$

$., 0),$

$., 0),$

$., 0),$

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

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

$, 0, \ldots, 1)$

, subtractions.

$(a_0, a_1, \ldots, a_{n-1}, d_0, d_1, \ldots, d_{n-1})$
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 LLL etc.

1997 Coppersmith–Shamir
balancing: e.g., set up lattice
to contain $(10a, d)$
if d is chosen $10\times$ larger than a.

Exercise: Describe search for
(b, c) as a problem of finding
a vector close to a lattice.

Quotient NTRU v

"Quotient NTRU"
is the structure we

Alice generates A
for small random
i.e., $dA - 3a = 0$

$d_{n-1})$

0),

. , 0),

ions.

$(a_0, a_1, \ldots, a_{n-1}, d_0, d_1, \ldots, d_{n-1})$
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 LLL etc.

1997 Coppersmith–Shamir
balancing: e.g., set up lattice
to contain $(10a, d)$
if d is chosen $10\times$ larger than a.

Exercise: Describe search for
(b, c) as a problem of finding
a vector close to a lattice.

Quotient NTRU vs. product

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

Alice generates $A = 3a/d$ in
for small random a, d:
i.e., $dA - 3a = 0$ in R_q.

$(a_0, a_1, \ldots, a_{n-1}, d_0, d_1, \ldots, d_{n-1})$
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 LLL etc.

1997 Coppersmith–Shamir
balancing: e.g., set up lattice
to contain $(10a, d)$
if d is chosen $10\times$ larger than a.

Exercise: Describe search for
(b, c) as a problem of finding
a vector close to a lattice.

Quotient NTRU vs. product NTRU

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

Alice generates $A = 3a/d$ in R_q
for small random a, d:
i.e., $dA - 3a = 0$ in R_q.

$(a_0, a_1, \ldots, a_{n-1}, d_0, d_1, \ldots, d_{n-1})$ 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 LLL etc.

1997 Coppersmith–Shamir balancing: e.g., set up lattice to contain $(10a, d)$ if d is chosen $10\times$ larger than a.

Exercise: Describe search for (b, c) as a problem of finding a vector close to a lattice.

Quotient NTRU vs. product NTRU

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

Alice generates $A = 3a/d$ in R_q for small random a, d: i.e., $dA - 3a = 0$ in R_q.

Bob sends $C = Ab + c$ in R_q.
Alice computes dC in R_q, i.e., $3ab + dc$ in R_q.

$(a_0, a_1, \ldots, a_{n-1}, d_0, d_1, \ldots, d_{n-1})$
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 LLL etc.

1997 Coppersmith–Shamir
balancing: e.g., set up lattice
to contain $(10a, d)$
if d is chosen $10\times$ larger than a.

Exercise: Describe search for
(b, c) as a problem of finding
a vector close to a lattice.

Quotient NTRU vs. product NTRU

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

Alice generates $A = 3a/d$ in R_q
for small random a, d:
i.e., $dA - 3a = 0$ in R_q.

Bob sends $C = Ab + c$ in R_q.
Alice computes dC in R_q,
i.e., $3ab + dc$ in R_q.

Alice reconstructs $3ab + dc$ in R,
using smallness of a, b, d, c.
Alice computes dc in R_3,
deduces c, deduces b.

$\ldots, a_{n-1}, d_0, d_1, \ldots, d_{n-1})$
risingly short vector
e generated by
$, 0, 0, 0, \ldots, 0)$ etc.

r searches for short vector
attice using LLL etc.

ppersmith–Shamir
g: e.g., set up lattice
in $(10a, d)$
hosen $10\times$ larger than a.

: Describe search for
s a problem of finding
close to a lattice.

Quotient NTRU vs. product NTRU

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

Alice generates $A = 3a/d$ in R_q
for small random a, d:
i.e., $dA - 3a = 0$ in R_q.

Bob sends $C = Ab + c$ in R_q.
Alice computes dC in R_q,
i.e., $3ab + dc$ in R_q.

Alice reconstructs $3ab + dc$ in R,
using smallness of a, b, d, c.
Alice computes dc in R_3,
deduces c, deduces b.

"Produc
2010 Ly

Everyone
Alice ge
for smal

$d_0, d_1, \ldots, d_{n-1})$
ort vector
d by
$\ldots, 0)$ etc.

for short vector
g LLL etc.

–Shamir
et up lattice
)
larger than a.

e search for
n of finding
n lattice.

Quotient NTRU vs. product NTRU

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

Alice generates $A = 3a/d$ in R_q
for small random a, d:
i.e., $dA - 3a = 0$ in R_q.

Bob sends $C = Ab + c$ in R_q.
Alice computes dC in R_q,
i.e., $3ab + dc$ in R_q.

Alice reconstructs $3ab + dc$ in R,
using smallness of a, b, d, c.
Alice computes dc in R_3,
deduces c, deduces b.

"Product NTRU"
2010 Lyubashevsk

Everyone knows ra
Alice generates A
for small random a

$d_{n-1})$

vector

ce

an a.

r

g

Quotient NTRU vs. product NTRU

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

Alice generates $A = 3a/d$ in R_q for small random a, d: i.e., $dA - 3a = 0$ in R_q.

Bob sends $C = Ab + c$ in R_q. Alice computes dC in R_q, i.e., $3ab + dc$ in R_q.

Alice reconstructs $3ab + dc$ in R, using smallness of a, b, d, c. Alice computes dc in R_3, deduces c, deduces b.

"Product NTRU" (new nam

2010 Lyubashevsky–Peikert–

Everyone knows random $G \in$

Alice generates $A = aG + d$ for small random a, d.

Quotient NTRU vs. product NTRU

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

Alice generates $A = 3a/d$ in R_q for small random a, d: i.e., $dA - 3a = 0$ in R_q.

Bob sends $C = Ab + c$ in R_q. Alice computes dC in R_q, i.e., $3ab + dc$ in R_q.

Alice reconstructs $3ab + dc$ in R, using smallness of a, b, d, c. Alice computes dc in R_3, deduces c, deduces b.

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

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

Quotient NTRU vs. product NTRU

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

Alice generates $A = 3a/d$ in R_q for small random a, d: i.e., $dA - 3a = 0$ in R_q.

Bob sends $C = Ab + c$ in R_q. Alice computes dC in R_q, i.e., $3ab + dc$ in R_q.

Alice reconstructs $3ab + dc$ in R, using smallness of a, b, d, c. Alice computes dc in R_3, deduces c, deduces b.

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

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

Bob sends $B = Gb + e$ in R_q and $C = m + Ab + c$ in R_q where b, c, e are small and each coefficient of m is 0 or $q/2$.

Quotient NTRU vs. product NTRU

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

Alice generates $A = 3a/d$ in R_q for small random a, d: i.e., $dA - 3a = 0$ in R_q.

Bob sends $C = Ab + c$ in R_q. Alice computes dC in R_q, i.e., $3ab + dc$ in R_q.

Alice reconstructs $3ab + dc$ in R, using smallness of a, b, d, c. Alice computes dc in R_3, deduces c, deduces b.

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

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

Bob sends $B = Gb + e$ in R_q and $C = m + Ab + c$ in R_q where b, c, e are small and each coefficient of m is 0 or $q/2$.

Alice computes $C - aB$ in R_q, i.e., $m + db + c - ae$ in R_q. Alice reconstructs m, using smallness of d, b, c, a, e.