

Lattice-based public-key cryptosystems

D. J. Bernstein

NIST post-quantum competition:
69 submissions in first round,
from hundreds of people.

(+13 submissions that NIST
declared incomplete or improper.)

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!

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

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

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

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.

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.

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:
```

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

```
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:
```

```
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:
```

```
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  

$$4x^2 + x + 3$$
  
sage: g = Zx([2,7,1])  
sage: g  

$$x^2 + 7x + 2$$
  
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:
```

```
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: 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: f*x      # built-in mul
```

$$4x^3 + x^2 + 3x$$

```
sage: f*x^2
```

$$4x^4 + x^3 + 3x^2$$

```
sage: f*2
```

$$8x^2 + 2x + 6$$

```
sage: f*(7*x)
```

$$28x^3 + 7x^2 + 21x$$

```
sage:
```

```
sage: f*x      # built-in mul
```

$$4x^3 + x^2 + 3x$$

```
sage: f*x^2
```

$$4x^4 + x^3 + 3x^2$$

```
sage: f*2
```

$$8x^2 + 2x + 6$$

```
sage: f*(7*x)
```

$$28x^3 + 7x^2 + 21x$$

```
sage: f*g
```

$$4x^4 + 29x^3 + 18x^2 + 23x + 6$$

```
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) % (xn-1)  
....:  
sage:
```

```
sage: # replace  $x^n$  with 1,  
sage: #  $x^{(n+1)}$  with  $x$ , etc.  
sage: def convolution(f,g):  
....:     return (f*g) % (xn-1)  
....:  
sage: n = 3    # global variable  
sage:
```

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

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

```
sage: def randompoly():
....:     f = list(randrange(3)-1
....:         for j in range(n))
....:     return Zx(f)
....:
sage: n = 7
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: 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: 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.

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.

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?

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.

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.

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:
```

```
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:
```

```
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:
```

```
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:
```

```
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 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 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 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 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)
```

$$6x^6 + 6x^5 + 3x^4 + 3x^3 + 3x^2 + 3x + 4$$

```
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`.

```
sage: n = 7
```

```
sage: q = 256
```

```
sage:
```

```
sage: n = 7
```

```
sage: q = 256
```

```
sage: f = randompoly()
```

```
sage:
```

```
sage: n = 7
```

```
sage: q = 256
```

```
sage: f = randompoly()
```

```
sage: f
```

```
-x^6 - x^4 + x^2 + x - 1
```

```
sage:
```

```
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:
```

```
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:
```

```
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:
```

```
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).

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

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.

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
```

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

```
sage:
```

```
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:
```

```
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:
```

```
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:
```

```
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:
```

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

```
sage: A
```

$$-126x^6 - 31x^5 - 118x^4 - 33x^3 + 73x^2 - 16x + 7$$

```
sage: d,d3 = secretkey
```

```
sage: d
```

$$-x^6 + x^5 - x^4 + x^3 - 1$$

```
sage: convolution(d,A)
```

$$-3x^6 + 253x^5 + 253x^3 - 253x^2 - 3x - 3$$

```
sage: balancedmod(_,q)
```

$$-3x^6 - 3x^5 - 3x^3 + 3x^2 - 3x - 3$$

```
sage:
```

NTRU encryption

One more parameter:

w , positive integer (e.g., 467).

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.

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:

```

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

```

$$21x^6 - 48x^5 + 31x^4 - 76x^3 - 77x^2 + 15x - 113$$

```

sage:

```

NTRU decryption

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

NTRU decryption

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

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

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 $q/2 - 1$.

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

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 .

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 .

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:
```

```
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:
```

```

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: n = 7
```

```
sage: w = 5
```

```
sage: q = 256
```

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

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

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

```
sage:
```

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

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

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

```
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:
```

```
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: 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: 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: 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: 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)
```

$$x^6 - x^5 - x^4 - 3x^3 - x^2 + x - 3$$

```
sage:
```

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)

$$x^6 - x^5 - x^4 - 3x^3 - x^2 + x - 3$$

sage: M(_,3)

$$x^6 - x^5 - x^4 - x^2 + x$$

sage:

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)

$$x^6 - x^5 - x^4 - 3x^3 - x^2 + x - 3$$

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.

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 .

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 .)

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 $4w$.

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

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

Same argument doesn't work.

$$a = b = c = d =$$

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

$$3ab + dc \text{ has a coeff } 4w > q/2.$$

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

Same argument doesn't work.

$$a = b = c = d =$$

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

$$3ab + dc \text{ has a coeff } 4w > q/2.$$

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

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

Same argument doesn't work.

$$a = b = c = d =$$

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

$3ab + dc$ 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”.

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

$$c_0 d_{n-1} + c_1 d_{n-2} + \dots + c_{n-1} d_0.$$

This coeff is large $\Leftrightarrow$

$c_0, c_1, \dots, c_{n-1}$ has

high correlation with

$d_{n-1}, d_{n-2}, \dots, d_0$.

Coeff of x^{n-1} in cd is

$$c_0 d_{n-1} + c_1 d_{n-2} + \dots + c_{n-1} d_0.$$

This coeff is large $\Leftrightarrow$

$c_0, c_1, \dots, c_{n-1}$ has
high correlation with
 $d_{n-1}, d_{n-2}, \dots, d_0$.

Some coeff is large $\Leftrightarrow$

$c_0, c_1, \dots, c_{n-1}$ has high
correlation with some rotation
of $d_{n-1}, d_{n-2}, \dots, d_0$.

Coeff of x^{n-1} in cd is

$$c_0 d_{n-1} + c_1 d_{n-2} + \dots + c_{n-1} d_0.$$

This coeff is large $\Leftrightarrow$

$c_0, c_1, \dots, c_{n-1}$ has
high correlation with
 $d_{n-1}, d_{n-2}, \dots, d_0$.

Some coeff is large $\Leftrightarrow$

$c_0, c_1, \dots, c_{n-1}$ has high
correlation with some rotation
of $d_{n-1}, d_{n-2}, \dots, d_0$.

i.e. c is correlated with

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

$$\text{rev}(d) = d_0 + d_1 x^{n-1} + \dots + d_{n-1} x.$$

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

Reasonable guesses given a
random decryption failure:

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

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

Reasonable guesses given a
random decryption failure:

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

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

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

Reasonable guesses given a random decryption failure:

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

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

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

Experimentally confirmed:

Average of $c \text{rev}(c)$

over some decryption failures

is close to $d \text{rev}(d)$.

Round to integers: $d \text{rev}(d)$.

Reasonable guesses given a random decryption failure:

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

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

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

Experimentally confirmed:

Average of $c \text{rev}(c)$

over some decryption failures
is close to $d \text{rev}(d)$.

Round to integers: $d \text{rev}(d)$.

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.

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

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

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

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

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

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

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

Then $3ab + dc + kd =$
 $\dots + (390 + k)x^{478} + \dots$.

Decryption fails for big k .

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

Then $3ab + dc + kd =$
 $\dots + (390 + k)x^{478} + \dots$.

Decryption fails for big k .

Search for smallest k that fails.

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

Then $3ab + dc + kd =$
 $\dots + (390 + k)x^{478} + \dots$.

Decryption fails for big k .

Search for smallest k that fails.

Does $3ab + dc + kxd$ also fail?

Yes *if* $xd = \dots + x^{478} + \dots$,
 i.e., if $d = \dots + x^{477} + \dots$.

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

Then $3ab + dc + kd =$
 $\dots + (390 + k)x^{478} + \dots$.

Decryption fails for big k .

Search for smallest k that fails.

Does $3ab + dc + kxd$ also fail?

Yes if $xd = \dots + x^{478} + \dots$,
 i.e., if $d = \dots + x^{477} + \dots$.

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.

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.

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.

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

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

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.

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: Compute all randomness that was used.

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

Solution: 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.”

Solution: 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.

Solution: 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.

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.

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} .

⋮

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.

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.

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?

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.

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 ?

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!

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.)

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.

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.

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.

$n = 701$, $w = 467$:

$$\binom{n}{w} 2^w \approx 2^{1106.09},$$

$$3^n \approx 2^{1111.06},$$

$$3^n/n \approx 2^{1101.61}.$$

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!

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 .

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 .

$$a = (A/3)d = (A/3)d_1 + (A/3)d_2$$

$$\text{so } a - (A/3)d_2 = (A/3)d_1.$$

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 .

$$a = (A/3)d = (A/3)d_1 + (A/3)d_2$$

$$\text{so } a - (A/3)d_2 = (A/3)d_1.$$

Eliminate a : almost certainly

$$H(-(A/3)d_2) = H((A/3)d_1) \text{ for}$$

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

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 .

$$a = (A/3)d = (A/3)d_1 + (A/3)d_2$$

$$\text{so } a - (A/3)d_2 = (A/3)d_1.$$

Eliminate a : almost certainly

$$H(-(A/3)d_2) = H((A/3)d_1) \text{ for}$$

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

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

Lattices

This is a lettuce:

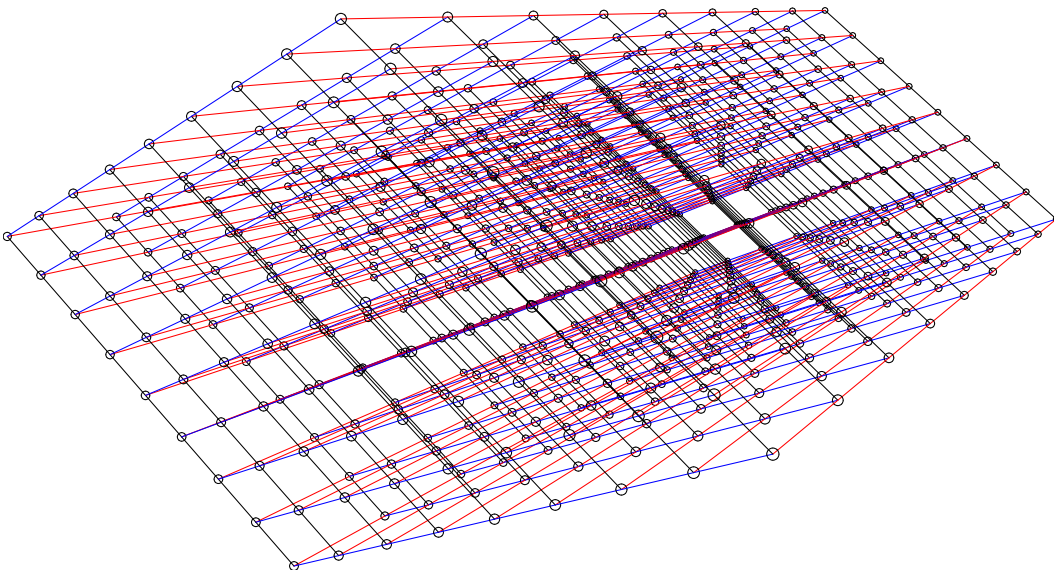


Lattices

This is a lettuce:



This is a lattice:



Lattices, mathematically

Assume that $b_1, b_2, \dots, b_k \in \mathbf{R}^n$
are $\mathbf{R}$ -linearly independent,

i.e., $\mathbf{R}b_1 + \dots + \mathbf{R}b_k =$

$\{r_1 b_1 + \dots + r_k b_k : r_1, \dots, r_k \in \mathbf{R}\}$

is a k -dimensional vector space.

Lattices, mathematically

Assume that $b_1, b_2, \dots, b_k \in \mathbf{R}^n$
are $\mathbf{R}$ -linearly independent,

i.e., $\mathbf{R}b_1 + \dots + \mathbf{R}b_k =$

$\{r_1 b_1 + \dots + r_k b_k : r_1, \dots, r_k \in \mathbf{R}\}$

is a k -dimensional vector space.

$\mathbf{Z}b_1 + \dots + \mathbf{Z}b_k =$

$\{r_1 b_1 + \dots + r_k b_k : r_1, \dots, r_k \in \mathbf{Z}\}$

is a rank- k length- n **lattice**.

Lattices, mathematically

Assume that $b_1, b_2, \dots, b_k \in \mathbf{R}^n$
are $\mathbf{R}$ -linearly independent,

i.e., $\mathbf{R}b_1 + \dots + \mathbf{R}b_k =$

$\{r_1 b_1 + \dots + r_k b_k : r_1, \dots, r_k \in \mathbf{R}\}$

is a k -dimensional vector space.

$\mathbf{Z}b_1 + \dots + \mathbf{Z}b_k =$

$\{r_1 b_1 + \dots + r_k b_k : r_1, \dots, r_k \in \mathbf{Z}\}$

is a rank- k length- n **lattice**.

$b_1, \dots, b_k$

is a **basis** of this lattice.

Short vectors in lattices

Given $b_1, b_2, \dots, b_k \in \mathbf{Z}^n$,
what is shortest vector
in $\mathbf{Z}b_1 + \dots + \mathbf{Z}b_k$?

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.

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?

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.

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 view of NTRU

Given public key $A = 3a/d$.

Compute $A/3 = a/d$.

Lattice view of NTRU

Given public key $A = 3a/d$.

Compute $A/3 = a/d$.

d is obtained from

$1, x, \dots, x^{n-1}$

by a few additions, subtractions.

Lattice view of NTRU

Given public key $A = 3a/d$.

Compute $A/3 = a/d$.

d is obtained from

$$1, x, \dots, x^{n-1}$$

by a few additions, subtractions.

$d(A/3)$ is obtained from

$$A/3, xA/3, \dots, x^{n-1}A/3$$

by a few additions, subtractions.

Lattice view of NTRU

Given public key $A = 3a/d$.

Compute $A/3 = a/d$.

d is obtained from

$$1, x, \dots, x^{n-1}$$

by a few additions, subtractions.

$d(A/3)$ is obtained from

$$A/3, xA/3, \dots, x^{n-1}A/3$$

by a few additions, subtractions.

a is obtained from

$$q, qx, qx^2, \dots, qx^{n-1},$$

$$A/3, xA/3, \dots, x^{n-1}A/3$$

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.

(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

$$H_0 + H_1x + \dots + H_{n-1}x^{n-1}.$$

$(a_0, a_1, \dots, a_{n-1}, d_0, d_1, \dots, d_{n-1})$

is obtained from

$(q, 0, \dots, 0, 0, 0, \dots, 0),$

$(0, q, \dots, 0, 0, 0, \dots, 0),$

$\vdots$

$(0, 0, \dots, q, 0, 0, \dots, 0),$

$(H_0, H_1, \dots, H_{n-1}, 1, 0, \dots, 0),$

$(H_{n-1}, H_0, \dots, H_{n-2}, 0, 1, \dots, 0),$

$\vdots$

$(H_1, H_2, \dots, H_0, 0, 0, \dots, 1)$

by a few additions, subtractions.

$(a_0, a_1, \dots, a_{n-1}, d_0, d_1, \dots, d_{n-1})$

is a surprisingly short vector

in lattice generated by

$(q, 0, \dots, 0, 0, 0, \dots, 0)$ etc.

$(a_0, a_1, \dots, a_{n-1}, d_0, d_1, \dots, d_{n-1})$

is a surprisingly short vector

in lattice generated by

$(q, 0, \dots, 0, 0, 0, \dots, 0)$ etc.

Attacker searches for short vector

in this lattice using LLL etc.

$(a_0, a_1, \dots, a_{n-1}, d_0, d_1, \dots, d_{n-1})$

is a surprisingly short vector

in lattice generated by

$(q, 0, \dots, 0, 0, 0, \dots, 0)$ 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 .

$(a_0, a_1, \dots, a_{n-1}, d_0, d_1, \dots, d_{n-1})$
is a surprisingly short vector
in lattice generated by
 $(q, 0, \dots, 0, 0, 0, \dots, 0)$ 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 .

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 .

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 .

“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 $q/2$.

“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 .