Lattice-based cryptography, part 1: simplicity

D. J. Bernstein

University of Illinois at Chicago;
Ruhr University Bochum

2000 Cohen cryptosystem

Public key: vector of integers
$K = (K_1, \ldots, K_N) \in \{-X, \ldots, X\}^N$.

Encryption:

1. Input message $m \in \{0, 1\}$.

2. Generate $r_1, \ldots, r_N \in \{0, 1\}$.
i.e. $r = (r_1, \ldots, r_N) \in \{0, 1\}^N$.

(Cohen says pick "half of the integers in the public key at random": I guess this means $N \in 2\mathbf{Z}$ and $\sum r_i = N/2$.)

3. Compute and send ciphertext
$C = (-1)^m(r_1K_1 + \cdots + r_NK_N)$.

ased cryptography,
simplicity

ernstein

ty of Illinois at Chicago;
iversity Bochum

2000 Cohen cryptosystem

Public key: vector of integers
$K = (K_1, \ldots, K_N) \in \{-X, \ldots, X\}^N$.

Encryption:

1. Input message $m \in \{0, 1\}$.

2. Generate $r_1, \ldots, r_N \in \{0, 1\}$.
i.e. $r = (r_1, \ldots, r_N) \in \{0, 1\}^N$.

(Cohen says pick "half of the integers in the public key at random": I guess this means $N \in 2\mathbf{Z}$ and $\sum r_i = N/2$.)

3. Compute and send ciphertext
$C = (-1)^m (r_1 K_1 + \cdots + r_N K_N)$.

How car

Key gen
Generate

$u_1, \ldots,$

$K_i \in (u_i$

Decrypti
$m = 0$ if
otherwis

Why thi
K_i mod
$r_1 K_1 + \cdot$

(Be care

tography,

is at Chicago;

ochum

2000 Cohen cryptosystem

Public key: vector of integers
$K = (K_1, \ldots, K_N) \in \{-X, \ldots, X\}^N$.

Encryption:

1. Input message $m \in \{0, 1\}$.

2. Generate $r_1, \ldots, r_N \in \{0, 1\}$.
i.e. $r = (r_1, \ldots, r_N) \in \{0, 1\}^N$.

(Cohen says pick "half of the integers in the public key at random": I guess this means $N \in 2\mathbf{Z}$ and $\sum r_i = N/2$.)

3. Compute and send ciphertext
$C = (-1)^m (r_1 K_1 + \cdots + r_N K_N)$.

How can receiver

Key generation:
Generate $s \in \{1, .$
$u_1, \ldots, u_N \in \{0, .$
$K_i \in (u_i + s\mathbf{Z}) \cap$

Decryption:
$m = 0$ if C mod s
otherwise $m = 1$.

Why this works:
K_i mod $s = u_i \leq$
$r_1 K_1 + \cdots + r_N K_N$

(Be careful! What

ago;

2000 Cohen cryptosystem

Public key: vector of integers $K = (K_1, \ldots, K_N) \in \{-X, \ldots, X\}^N$.

Encryption:

1. Input message $m \in \{0, 1\}$.

2. Generate $r_1, \ldots, r_N \in \{0, 1\}$. i.e. $r = (r_1, \ldots, r_N) \in \{0, 1\}^N$.

(Cohen says pick "half of the integers in the public key at random": I guess this means $N \in 2\mathbf{Z}$ and $\sum r_i = N/2$.)

3. Compute and send ciphertext $C = (-1)^m (r_1 K_1 + \cdots + r_N K_N)$.

How can receiver decrypt?

Key generation:

Generate $s \in \{1, \ldots, Y\}$;

$u_1, \ldots, u_N \in \left\{0, \ldots, \left\lfloor \frac{s-1}{2N} \right.\right.$

$K_i \in (u_i + s\mathbf{Z}) \cap \{-X, \ldots,$

Decryption:

$m = 0$ if $C \bmod s \le (s-1)$

otherwise $m = 1$.

Why this works:

$K_i \bmod s = u_i \le (s-1)/2N$

$r_1 K_1 + \cdots + r_N K_N \bmod s \le$

(Be careful! What if all $r_i =$

2000 Cohen cryptosystem

Public key: vector of integers $K = (K_1, \ldots, K_N) \in \{-X, \ldots, X\}^N$.

Encryption:

1. Input message $m \in \{0, 1\}$.

2. Generate $r_1, \ldots, r_N \in \{0, 1\}$.
i.e. $r = (r_1, \ldots, r_N) \in \{0, 1\}^N$.

(Cohen says pick "half of the integers in the public key at random": I guess this means $N \in 2\mathbf{Z}$ and $\sum r_i = N/2$.)

3. Compute and send ciphertext $C = (-1)^m (r_1 K_1 + \cdots + r_N K_N)$.

How can receiver decrypt?

Key generation:
Generate $s \in \{1, \ldots, Y\}$;
$u_1, \ldots, u_N \in \left\{0, \ldots, \left\lfloor \frac{s-1}{2N} \right\rfloor \right\}$;
$K_i \in (u_i + s\mathbf{Z}) \cap \{-X, \ldots, X\}$.

Decryption:
$m = 0$ if $C \bmod s \leq (s-1)/2$;
otherwise $m = 1$.

Why this works:
$K_i \bmod s = u_i \leq (s-1)/2N$ so
$r_1 K_1 + \cdots + r_N K_N \bmod s \leq \frac{s-1}{2}$.

(Be careful! What if all $r_i = 0$?)

hen cryptosystem

ey: vector of integers

$_1, \ldots, K_N) \in \{-X, \ldots, X\}^N$.

on:

message $m \in \{0, 1\}$.

rate $r_1, \ldots, r_N \in \{0, 1\}$.

$(r_1, \ldots, r_N) \in \{0, 1\}^N$.

says pick “half of the

in the public key at

’: I guess this means

and $\sum r_i = N/2$.)

pute and send ciphertext

$1)^m(r_1 K_1 + \cdots + r_N K_N)$.

How can receiver decrypt?

Key generation:

Generate $s \in \{1, \ldots, Y\}$;

$u_1, \ldots, u_N \in \left\{0, \ldots, \left\lfloor \frac{s-1}{2N} \right\rfloor\right\}$;

$K_i \in (u_i + s\mathbf{Z}) \cap \{-X, \ldots, X\}$.

Decryption:

$m = 0$ if $C \bmod s \le (s-1)/2$;

otherwise $m = 1$.

Why this works:

$K_i \bmod s = u_i \le (s-1)/2N$ so

$r_1 K_1 + \cdots + r_N K_N \bmod s \le \frac{s-1}{2}$.

(Be careful! What if all $r_i = 0$?)

Let’s try

Debian:

Fedora:

Source:

Web (us

sagecel

Sage is

+ many

+ a few

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

osystem

of integers

$) \in \{-X, \dots, X\}^N$.

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

$\dots, r_N \in \{0, 1\}$.

$) \in \{0, 1\}^N$.

“half of the

lic key at

this means

$= N/2$.)

end ciphertext

$+ \dots + r_N K_N)$.

How can receiver decrypt?

Key generation:

Generate $s \in \{1, \dots, Y\}$;

$$u_1, \dots, u_N \in \left\{0, \dots, \left\lfloor \frac{s-1}{2N} \right\rfloor \right\};$$

$K_i \in (u_i + s\mathbf{Z}) \cap \{-X, \dots, X\}$.

Decryption:

$m = 0$ if $C \bmod s \le (s-1)/2$;

otherwise $m = 1$.

Why this works:

$K_i \bmod s = u_i \le (s-1)/2N$ so

$$r_1 K_1 + \dots + r_N K_N \bmod s \le \frac{s-1}{2}.$$

(Be careful! What if all $r_i = 0$?)

Let’s try this on th

Debian: `apt inst`

Fedora: `dnf inst`

Source: `www.sage`

Web (use `print(X`

`sagecell.sagema`

Sage is Python 3

+ many math libra

+ a few syntax dif

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

rs

$\ldots, X\}^N$.

$\}$.

$, 1\}$.

^{N}.

e

s

rtext

$K_N)$.

How can receiver decrypt?

Key generation:

Generate $s \in \{1, \ldots, Y\}$;

$$u_1, \ldots, u_N \in \left\{0, \ldots, \left\lfloor \frac{s-1}{2N} \right\rfloor \right\};$$

$K_i \in (u_i + s\mathbf{Z}) \cap \{-X, \ldots, X\}$.

Decryption:

$m = 0$ if $C \bmod s \le (s-1)/2$;

otherwise $m = 1$.

Why this works:

$K_i \bmod s = u_i \le (s-1)/2N$ so

$$r_1 K_1 + \cdots + r_N K_N \bmod s \le \frac{s-1}{2}.$$

(Be careful! What if all $r_i = 0$?)

Let's try this on the comput

Debian: `apt install sage`

Fedora: `dnf install sage`

Source: `www.sagemath.org`

Web (use `print(X)` to see

`sagecell.sagemath.org`

Sage is Python 3

+ many math libraries

+ a few syntax differences:

```
sage: 10^6 # power, not x
1000000
sage: factor(31415926535
317213509 * 990371647
sage:
```

How can receiver decrypt?

Key generation:

Generate $s \in \{1, \ldots, Y\}$;

$u_1, \ldots, u_N \in \left\{0, \ldots, \left\lfloor \frac{s-1}{2N} \right\rfloor \right\}$;

$K_i \in (u_i + s\mathbf{Z}) \cap \{-X, \ldots, X\}$.

Decryption:

$m = 0$ if $C \bmod s \le (s-1)/2$;

otherwise $m = 1$.

Why this works:

$K_i \bmod s = u_i \le (s-1)/2N$ so

$r_1K_1 + \cdots + r_NK_N \bmod s \le \frac{s-1}{2}$.

(Be careful! What if all $r_i = 0$?)

Let's try this on the computer.

Debian: `apt install sagemath`

Fedora: `dnf install sagemath`

Source: `www.sagemath.org`

Web (use `print(X)` to see `X`):

`sagecell.sagemath.org`

Sage is Python 3

+ many math libraries

+ a few syntax differences:

```
sage: 10^6 # power, not xor
1000000
sage: factor(31415926535897932 3)
```
317213509 * 990371647

sage:

n receiver decrypt?

eration:

e $s \in \{1, \dots, Y\}$;

$u_N \in \left\{0, \dots, \left\lfloor \frac{s-1}{2N} \right\rfloor \right\}$;

$+ s\mathbf{Z}) \cap \{-X, \dots, X\}$.

ion:

f $C \bmod s \le (s-1)/2$;

e $m = 1$.

s works:

$s = u_i \le (s-1)/2N$ so

$\cdots + r_N K_N \bmod s \le \frac{s-1}{2}$.

eful! What if all $r_i = 0$?)

Let's try this on the computer.

Debian: `apt install sagemath`

Fedora: `dnf install sagemath`

Source: `www.sagemath.org`

Web (use `print(X)` to see `X`):

`sagecell.sagemath.org`

Sage is Python 3

+ many math libraries

+ a few syntax differences:

```
sage: 10^6 # power, not xor
1000000
sage: factor(31415926535879323)
317213509 * 990371647
sage:
```

For integ

Sage's "

outputs

Matches

C mod s

Warning

C < 0 pr

in lower-

nonzero

Warning

Sage can

decrypt?

$\ldots, Y\}$;

$\ldots, \left\lfloor \frac{s-1}{2N} \right\rfloor \Big\}$;

$\{-X, \ldots, X\}$.

$\leq (s-1)/2$;

$(s-1)/2N$ so

$\bmod s \leq \frac{s-1}{2}$.

if all $r_i = 0$?)

Let's try this on the computer.

Debian: `apt install sagemath`

Fedora: `dnf install sagemath`

Source: `www.sagemath.org`

Web (use `print(X)` to see `X`):

`sagecell.sagemath.org`

Sage is Python 3

+ many math libraries

+ a few syntax differences:

```
sage: 10^6 # power, not xor
1000000
sage: factor(31415926535879323)
317213509 * 990371647
sage:
```

For integers `C`, `s` w

Sage's "`C%s`" alwa

outputs between `0`

Matches standard

$C \bmod s = C - \lfloor C$

Warning: Typically

`C` < 0 produces `C%`

in lower-level langu

nonzero output lea

Warning: For poly

Sage can make th

$\rfloor\}$;

$X\}$.

/2;

V so

$\frac{s-1}{2}$.

= 0?)

Let's try this on the computer.

Debian: `apt install sagemath`

Fedora: `dnf install sagemath`

Source: www.sagemath.org

Web (use `print(X)` to see `X`):

sagecell.sagemath.org

Sage is Python 3

+ many math libraries

+ a few syntax differences:

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

For integers `C`, `s` with `s` > 0

Sage's "`C%s`" always produc

outputs between 0 and `s` $-$

Matches standard math defi

$C \bmod s = C - \lfloor C/s \rfloor s$.

Warning: Typically

`C` < 0 produces `C%s` < 0

in lower-level languages, so

nonzero output leaks input s

Warning: For polynomials `C`

Sage can make the same mi

Let's try this on the computer.

Debian: `apt install sagemath`
Fedora: `dnf install sagemath`
Source: `www.sagemath.org`
Web (use `print(X)` to see `X`):
`sagecell.sagemath.org`

Sage is Python 3
+ many math libraries
+ a few syntax differences:

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


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

Matches standard math definition:
$C \bmod s = C - \lfloor C/s \rfloor s$.

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

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

y this on the computer.

```
apt install sagemath
dnf install sagemath
www.sagemath.org
```

se print(X) to see X):

ll.sagemath.org

Python 3

math libraries

syntax differences:

```
0^6 # power, not xor

actor(31415926535897932 3)
09 * 990371647
```


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

Matches standard math definition:
$C \bmod s = C - \lfloor C/s \rfloor s$.

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

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

```
sage: N
sage: X
sage: Y
sage: Y
1048576
sage: s
sage: s
359512
sage: u
....:
....:
sage: u
[14485,
 10493,
 8213,
```

ne computer.

```
all sagemath
all sagemath
emath.org
```

X) to see X):

```
ath.org
```

aries

fferences:

```
er, not xor
15926535897932 3)
71647
```


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

Matches standard math definition:
$C \bmod s = C - \lfloor C/s \rfloor s$.

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

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


```
sage: N=10
sage: X=2^50
sage: Y=2^20
sage: Y
1048576
sage: s=randrang
sage: s
359512
sage: u=[randran
....:       (s-1)
....:     for i i
sage: u
[14485, 7039, 69
 10493, 17333, 1
 8213, 6370]
```

er.

math

math

g

X):

or

979323)

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

Matches standard math definition:
$C \bmod s = C - \lfloor C/s \rfloor s$.

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

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


```
sage: N=10
sage: X=2^50
sage: Y=2^20
sage: Y
1048576
sage: s=randrange(1,Y+1)
sage: s
359512
sage: u=[randrange(
....:       (s-1)//(2*N)+1
....:     for i in range(N
sage: u
[14485, 7039, 6945, 15890
 10493, 17333, 1397, 8656
 8213, 6370]
```

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

Matches standard math definition: $C \bmod s = C - \lfloor C/s \rfloor s$.

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

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


```
sage: N=10
sage: X=2^50
sage: Y=2^20
sage: Y
1048576
sage: s=randrange(1,Y+1)
sage: s
359512
sage: u=[randrange(
....:      (s-1)//(2*N)+1)
....:    for i in range(N)]
sage: u
[14485, 7039, 6945, 15890,
 10493, 17333, 1397, 8656,
 8213, 6370]
```

gers C, s with s > 0,
"C%s" always produces
between 0 and s − 1.

s standard math definition:
$s = C - \lfloor C/s \rfloor s$.

g: Typically
roduces C%s < 0
-level languages, so
output leaks input sign.

g: For polynomials C,
n make the same mistake.


```
sage: N=10
sage: X=2^50
sage: Y=2^20
sage: Y
1048576
sage: s=randrange(1,Y+1)
sage: s
359512
sage: u=[randrange(
....:       (s-1)//(2*N)+1)
....:    for i in range(N)]
sage: u
[14485, 7039, 6945, 15890,
 10493, 17333, 1397, 8656,
 8213, 6370]
```

```
sage: K
....:
....:
....:
sage: K
[870056
 822006
 -29476
 -66927
 528958
 426006
 -64194
 501543
 -58306
 461093
```

vith s > 0,
ys produces
) and s − 1.

math definition:
C/s⌋s.

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

ynomials C,
e same mistake.


```
sage: N=10
sage: X=2^50
sage: Y=2^20
sage: Y
1048576
sage: s=randrange(1,Y+1)
sage: s
359512
sage: u=[randrange(
....:        (s-1)//(2*N)+1)
....:     for i in range(N)]
sage: u
[14485, 7039, 6945, 15890,
 10493, 17333, 1397, 8656,
 8213, 6370]
```

```
sage: K=[ui+s*ra
....:        ceil(
....:        floor
....:     for ui
sage: K
[870056918917829
 822006576592695
 -29476554434581
 -66927510008098
 528958455221029
 426006001074157
 -64194017608053
 501543495923784
 -58306407539258
 4610939024383 4]
```

,

es

1.

nition:

sign.

,

stake.


```
sage: N=10
sage: X=2^50
sage: Y=2^20
sage: Y
1048576
sage: s=randrange(1,Y+1)
sage: s
359512
sage: u=[randrange(
....:        (s-1)//(2*N)+1)
....:     for i in range(N)]
sage: u
[14485, 7039, 6945, 15890,
 10493, 17333, 1397, 8656,
 8213, 6370]
```



```
sage: K=[ui+s*randrange(
....:       ceil(-(X+ui)/s
....:       floor((X-ui)/s
....:     for ui in u]
sage: K
[870056918917829,
 822006576592695,
 -294765544345815,
 -669275100080982,
 528958455221029,
 426006001074157,
 -641940176080531,
 501543495923784,
 -583064075392587,
 461093902438344]
```

```
sage: N=10
sage: X=2^50
sage: Y=2^20
sage: Y
1048576
sage: s=randrange(1,Y+1)
sage: s
359512
sage: u=[randrange(
....:        (s-1)//(2*N)+1)
....:     for i in range(N)]
sage: u
[14485, 7039, 6945, 15890,
 10493, 17333, 1397, 8656,
 8213, 6370]
```



```
sage: K=[ui+s*randrange(
....:        ceil(-(X+ui)/s),
....:        floor((X-ui)/s)+1)
....:     for ui in u]
sage: K
[870056918917829,
 822006576592695,
 -294765544345815,
 -669275100080982,
 528958455221029,
 426006001074157,
 -641940176080531,
 501543495923784,
 -583064075392587,
 46109390243834]
```

```
=10
=2^50
=2^20

=randrange(1,Y+1)

=[randrange(
    (s-1)//(2*N)+1)
  for i in range(N)]

 7039, 6945, 15890,
 17333, 1397, 8656,
6370]
```



```
sage: K=[ui+s*randrange(
....:       ceil(-(X+ui)/s),
....:       floor((X-ui)/s)+1)
....:     for ui in u]
sage: K
[870056918917829,
 822006576592695,
 -294765544345815,
 -669275100080982,
 528958455221029,
 426006001074157,
 -641940176080531,
 501543495923784,
 -583064075392587,
 461093902438341]
```

```
sage: [
[14485,
 10493,
 8213,
sage: u
[14485,
 10493,
 8213,
sage: su
96821
sage: su
96821
sage: s/
179756
sage:
```

```
e(1,Y+1)

ge(
//(2*N)+1)
n range(N)]

45, 15890,
397, 8656,
```



```
sage: K=[ui+s*randrange(
....:        ceil(-(X+ui)/s),
....:        floor((X-ui)/s)+1)
....:      for ui in u]
sage: K
[870056918917829,
 822006576592695,
 -294765544345815,
 -669275100080982,
 528958455221029,
 426006001074157,
 -641940176080531,
 501543495923784,
 -583064075392587,
 461093902438341]
```



```
sage: [Ki%s for
[14485, 7039, 69
 10493, 17333, 1
 8213, 6370]
sage: u
[14485, 7039, 69
 10493, 17333, 1
 8213, 6370]
sage: sum(K)%s
96821
sage: sum(u)
96821
sage: s//2
179756
sage:
```

```
sage: K=[ui+s*randrange(
....:       ceil(-(X+ui)/s),
....:       floor((X-ui)/s)+1)
....:     for ui in u]
sage: K
[870056918917829,
 822006576592695,
 -294765544345815,
 -669275100080982,
 528958455221029,
 426006001074157,
 -641940176080531,
 501543495923784,
 -583064075392587,
 46109390243834]
```



```
sage: [Ki%s for Ki in K]
[14485, 7039, 6945, 15890
 10493, 17333, 1397, 8656
 8213, 6370]
sage: u
[14485, 7039, 6945, 15890
 10493, 17333, 1397, 8656
 8213, 6370]
sage: sum(K)%s
96821
sage: sum(u)
96821
sage: s//2
179756
sage:
```

```
sage: K=[ui+s*randrange(
....:       ceil(-(X+ui)/s),
....:       floor((X-ui)/s)+1)
....:     for ui in u]
sage: K
[870056918917829,
 822006576592695,
 -294765544345815,
 -669275100080982,
 528958455221029,
 426006001074157,
 -641940176080531,
 501543495923784,
 -583064075392587,
 461093902438341]
```



```
sage: [Ki%s for Ki in K]
[14485, 7039, 6945, 15890,
 10493, 17333, 1397, 8656,
 8213, 6370]
sage: u
[14485, 7039, 6945, 15890,
 10493, 17333, 1397, 8656,
 8213, 6370]
sage: sum(K)%s
96821
sage: sum(u)
96821
sage: s//2
179756
sage:
```

```
=[ui+s*randrange(
    ceil(-(X+ui)/s),
    floor((X-ui)/s)+1)
  for ui in u]

918917829,
576592695,
5544345815,
5100080982,
455221029,
001074157,
0176080531,
495923784,
4075392587,
90243834]
```



```
sage: [Ki%s for Ki in K]
[14485, 7039, 6945, 15890,
 10493, 17333, 1397, 8656,
 8213, 6370]
sage: u
[14485, 7039, 6945, 15890,
 10493, 17333, 1397, 8656,
 8213, 6370]
sage: sum(K)%s
96821
sage: sum(u)
96821
sage: s//2
179756
sage:
```

```
sage: m
sage: r
....:
sage: C
....:
sage: C
-202215
sage: C
47024
sage: m
0
sage: su
....:
47024
sage:
```

```
ndrange(
-(X+ui)/s),
((X-ui)/s)+1)
in u]

,
,
5,
2,
,
,
1,
,
7,
```



```
sage: [Ki%s for Ki in K]
[14485, 7039, 6945, 15890,
 10493, 17333, 1397, 8656,
 8213, 6370]
sage: u
[14485, 7039, 6945, 15890,
 10493, 17333, 1397, 8656,
 8213, 6370]
sage: sum(K)%s
96821
sage: sum(u)
96821
sage: s//2
179756
sage:
```

```
sage: m=randrang
sage: r=[randran
....:     for i i
sage: C=(-1)^m*s
....:    for i in
sage: C
-202215856043576
sage: C%s
47024
sage: m
0
sage: sum(r[i]*u
....:      for i
47024
sage:
```

```
),
)+1)
```



```
sage: [Ki%s for Ki in K]
[14485, 7039, 6945, 15890,
 10493, 17333, 1397, 8656,
 8213, 6370]
sage: u
[14485, 7039, 6945, 15890,
 10493, 17333, 1397, 8656,
 8213, 6370]
sage: sum(K)%s
96821
sage: sum(u)
96821
sage: s//2
179756
sage:
```



```
sage: m=randrange(2)
sage: r=[randrange(2)
....:      for i in range(N
sage: C=(-1)^m*sum(r[i]*K
....:     for i in range(N)
sage: C
-20221585604357б
sage: C%s
47024
sage: m
0
sage: sum(r[i]*u[i]
....:      for i in range(
47024
sage:
```

```
sage: [Ki%s for Ki in K]
[14485, 7039, 6945, 15890,
 10493, 17333, 1397, 8656,
 8213, 6370]
sage: u
[14485, 7039, 6945, 15890,
 10493, 17333, 1397, 8656,
 8213, 6370]
sage: sum(K)%s
96821
sage: sum(u)
96821
sage: s//2
179756
sage:
```



```
sage: m=randrange(2)
sage: r=[randrange(2)
....:     for i in range(N)]
sage: C=(-1)^m*sum(r[i]*K[i]
....:    for i in range(N))
sage: C
-202215856043576
sage: C%s
47024
sage: m
0
sage: sum(r[i]*u[i]
....:      for i in range(N))
47024
sage:
```

```
Ki%s for Ki in K]
 7039, 6945, 15890,
 17333, 1397, 8656,
6370]

 7039, 6945, 15890,
 17333, 1397, 8656,
6370]
um(K)%s

um(u)

//2
```



```
sage: m=randrange(2)
sage: r=[randrange(2)
....:    for i in range(N)]
sage: C=(-1)^m*sum(r[i]*K[i]
....:   for i in range(N))
sage: C
-202215856043576
sage: C%s
47024
sage: m
0
sage: sum(r[i]*u[i]
....:     for i in range(N))
47024
sage:
```

Some pr

1. Funct
System
that hav

2. Secur
We wan
“chosen-
where at
decrypti

Chosen-
against
Decrypt

(Works

```
Ki in K]
45, 15890,
397, 8656,

45, 15890,
397, 8656,
```



```
sage: m=randrange(2)
sage: r=[randrange(2)
....:    for i in range(N)]
sage: C=(-1)^m*sum(r[i]*K[i]
....:   for i in range(N))
sage: C
-202215856043576
sage: C%s
47024
sage: m
0
sage: sum(r[i]*u[i]
....:      for i in range(N))
47024
sage:
```

Some problems wi

1. Functionality p
System can't encr
that have more th

2. Security proble
We want cryptosy
"chosen-ciphertext
where attacker ca
decryptions of oth

Chosen-ciphertext
against this syste
Decrypt $-C$. Flip

(Works whenever

```
sage: m=randrange(2)
sage: r=[randrange(2)
....:    for i in range(N)]
sage: C=(-1)^m*sum(r[i]*K[i]
....:   for i in range(N))
sage: C
-202215856043576
sage: C%s
47024
sage: m
0
sage: sum(r[i]*u[i]
....:     for i in range(N))
47024
sage:
```


Some problems with cryptos

1. Functionality problem:
System can't encrypt messa
that have more than 1 bit.

2. Security problem:
We want cryptosystems to r
"chosen-ciphertext attacks"
where attacker can see
decryptions of other cipherte

Chosen-ciphertext attack
against this system:
Decrypt $-C$. Flip result.

(Works whenever $C \neq 0$.)

```
sage: m=randrange(2)
sage: r=[randrange(2)
....:    for i in range(N)]
sage: C=(-1)^m*sum(r[i]*K[i]
....:   for i in range(N))
sage: C
-202215856043576
sage: C%s
47024
sage: m
0
sage: sum(r[i]*u[i]
....:     for i in range(N))
47024
sage:
```


Some problems with cryptosystem

1. Functionality problem:
System can't encrypt messages that have more than 1 bit.

2. Security problem:
We want cryptosystems to resist "chosen-ciphertext attacks" where attacker can see decryptions of other ciphertexts.

Chosen-ciphertext attack against this system:
Decrypt $-C$. Flip result.

(Works whenever $C \neq 0$.)

```
=randrange(2)
=[randrange(2)
  for i in range(N)]
=(-1)^m*sum(r[i]*K[i]
 for i in range(N))

856043576
%s



um(r[i]*u[i]
    for i in range(N))
```


Some problems with cryptosystem

1. Functionality problem:
System can't encrypt messages
that have more than 1 bit.

2. Security problem:
We want cryptosystems to resist
"chosen-ciphertext attacks"
where attacker can see
decryptions of other ciphertexts.

Chosen-ciphertext attack
against this system:
Decrypt $-C$. Flip result.

(Works whenever $C \neq 0$.)

2000 Co
fixing bo

1. Trans
into mul
encrypti
Use new

B-bit inp
$m = (m$
For each
Generate

Cipherte
$(-1)^{m_1}($
. . . ,
$(-1)^{m_B}($

```
e(2)
ge(2)
n range(N)]
um(r[i]*K[i]
 range(N))

[i]
in range(N))
```


Some problems with cryptosystem

1. Functionality problem:
System can't encrypt messages that have more than 1 bit.

2. Security problem:
We want cryptosystems to resist "chosen-ciphertext attacks" where attacker can see decryptions of other ciphertexts.

Chosen-ciphertext attack against this system:
Decrypt $-C$. Flip result.

(Works whenever $C \neq 0$.)

2000 Cohen: cryp
fixing both of thes

1. Transform 1-bi
into multi-bit encr
encrypting each bi
Use new randomn

B-bit input messa
$m = (m_1, \ldots, m_B$
For each $i \in \{1, \ldots$
Generate $r_{i,1}, \ldots,$

Ciphertext C:
$(-1)^{m_1}(r_{1,1}K_1 +$
$\ldots,$
$(-1)^{m_B}(r_{B,1}K_1 +$

)]
[i]
)
N))

Some problems with cryptosystem

1. Functionality problem:
System can't encrypt messages that have more than 1 bit.

2. Security problem:
We want cryptosystems to resist "chosen-ciphertext attacks" where attacker can see decryptions of other ciphertexts.

Chosen-ciphertext attack against this system:
Decrypt $-C$. Flip result.

(Works whenever $C \neq 0$.)

2000 Cohen: cryptosystem fixing both of these problem

1. Transform 1-bit encryptio into multi-bit encryption by encrypting each bit separate
Use new randomness for eac

B-bit input message
$m = (m_1, \ldots, m_B) \in \{0, 1\}^{B}$
For each $i \in \{1, \ldots, B\}$:
Generate $r_{i,1}, \ldots, r_{i,N} \in \{0,$

Ciphertext C:
$(-1)^{m_1}(r_{1,1}K_1 + \cdots + r_{1,N}K$
$\ldots,$
$(-1)^{m_B}(r_{B,1}K_1 + \cdots + r_{B,N}$

Some problems with cryptosystem

1. Functionality problem:
System can't encrypt messages that have more than 1 bit.

2. Security problem:
We want cryptosystems to resist "chosen-ciphertext attacks" where attacker can see decryptions of other ciphertexts.

Chosen-ciphertext attack against this system:
Decrypt $-C$. Flip result.

(Works whenever $C \neq 0$.)

2000 Cohen: cryptosystem fixing both of these problems.

1. Transform 1-bit encryption into multi-bit encryption by encrypting each bit separately. Use new randomness for each bit.

B-bit input message
$m = (m_1, \ldots, m_B) \in \{0, 1\}^B$.
For each $i \in \{1, \ldots, B\}$:
Generate $r_{i,1}, \ldots, r_{i,N} \in \{0, 1\}$.

Ciphertext C:
$(-1)^{m_1}(r_{1,1}K_1 + \cdots + r_{1,N}K_N)$,
$\ldots$,
$(-1)^{m_B}(r_{B,1}K_1 + \cdots + r_{B,N}K_N)$.

oblems with cryptosystem

tionality problem:
can't encrypt messages
ve more than 1 bit.

rity problem:
t cryptosystems to resist
-ciphertext attacks"
ttacker can see
ons of other ciphertexts.

ciphertext attack
this system:
$-C$. Flip result.

whenever $C \neq 0$.)

2000 Cohen: cryptosystem fixing both of these problems.

1. Transform 1-bit encryption into multi-bit encryption by encrypting each bit separately. Use new randomness for each bit.

B-bit input message
$m = (m_1, \ldots, m_B) \in \{0, 1\}^B$.
For each $i \in \{1, \ldots, B\}$:
Generate $r_{i,1}, \ldots, r_{i,N} \in \{0, 1\}$.

Ciphertext C:
$(-1)^{m_1}(r_{1,1}K_1 + \cdots + r_{1,N}K_N)$,
$\ldots$,
$(-1)^{m_B}(r_{B,1}K_1 + \cdots + r_{B,N}K_N)$.

2. Derar
reencryp

This is a
1999 Fu

Derando
as crypto
using sta
(Watch

Decrypti
1. Input
2. Decry
3. Recor
4. Recor
5. Abort

th cryptosystem

roblem:

ypt messages

an 1 bit.

m:

stems to resist

t attacks”

n see

er ciphertexts.

attack

n:

result.

$C \neq 0$.)

2000 Cohen: cryptosystem fixing both of these problems.

1. Transform 1-bit encryption into multi-bit encryption by encrypting each bit separately. Use new randomness for each bit.

B-bit input message $m = (m_1, \ldots, m_B) \in \{0,1\}^B$.
For each $i \in \{1, \ldots, B\}$:
Generate $r_{i,1}, \ldots, r_{i,N} \in \{0,1\}$.

Ciphertext C:
$(-1)^{m_1}(r_{1,1}K_1 + \cdots + r_{1,N}K_N)$,
$\ldots$,
$(-1)^{m_B}(r_{B,1}K_1 + \cdots + r_{B,N}K_N)$.

2. Derandomize e

reencrypt during d

This is an exampl

1999 Fujisaki–Oka

Derandomization:

as cryptographic h

using standard has

(Watch out: Is m

Decryption with re

1. Input C'. (May

2. Decrypt to obt

3. Recompute r' =

4. Recompute C''

5. Abort if $C'' \neq$

ystem

ges

esist

exts.

2000 Cohen: cryptosystem fixing both of these problems.

1. Transform 1-bit encryption into multi-bit encryption by encrypting each bit separately. Use new randomness for each bit.

B-bit input message
$m = (m_1, \ldots, m_B) \in \{0,1\}^B$.
For each $i \in \{1, \ldots, B\}$:
Generate $r_{i,1}, \ldots, r_{i,N} \in \{0,1\}$.

Ciphertext C:
$(-1)^{m_1}(r_{1,1}K_1 + \cdots + r_{1,N}K_N)$,
$\ldots$,
$(-1)^{m_B}(r_{B,1}K_1 + \cdots + r_{B,N}K_N)$.

2. Derandomize encryption,
reencrypt during decryption.

This is an example of "FO",
1999 Fujisaki–Okamoto tran

Derandomization: Generate
as cryptographic hash $H(m)$
using standard hash functio
(Watch out: Is m guessable

Decryption with reencryptio
1. Input C'. (Maybe $C' \neq C$
2. Decrypt to obtain m'.
3. Recompute $r' = H(m')$.
4. Recompute C'' from m',
5. Abort if $C'' \neq C'$.

2000 Cohen: cryptosystem fixing both of these problems.

1. Transform 1-bit encryption into multi-bit encryption by encrypting each bit separately. Use new randomness for each bit.

B-bit input message
$m = (m_1, \ldots, m_B) \in \{0,1\}^B$.
For each $i \in \{1, \ldots, B\}$:
Generate $r_{i,1}, \ldots, r_{i,N} \in \{0,1\}$.

Ciphertext C:
$(-1)^{m_1}(r_{1,1}K_1 + \cdots + r_{1,N}K_N)$,
$\ldots$,
$(-1)^{m_B}(r_{B,1}K_1 + \cdots + r_{B,N}K_N)$.

2. Derandomize encryption, and reencrypt during decryption.

This is an example of "FO", the 1999 Fujisaki–Okamoto transform.

Derandomization: Generate r as cryptographic hash $H(m)$, using standard hash function H. (Watch out: Is m guessable?)

Decryption with reencryption:
1. Input C'. (Maybe $C' \neq C$.)
2. Decrypt to obtain m'.
3. Recompute $r' = H(m')$.
4. Recompute C'' from m', r'.
5. Abort if $C'' \neq C'$.

phen: cryptosystem
oth of these problems.

sform 1-bit encryption
ti-bit encryption by
ng each bit separately.
randomness for each bit.

put message
$_1, \ldots, m_B) \in \{0,1\}^B$.
n $i \in \{1, \ldots, B\}$:
e $r_{i,1}, \ldots, r_{i,N} \in \{0,1\}$.

xt C:
$(r_{1,1}K_1 + \cdots + r_{1,N}K_N)$,

$(r_{B,1}K_1 + \cdots + r_{B,N}K_N)$.

2. Derandomize encryption, and reencrypt during decryption.

This is an example of "FO", the 1999 Fujisaki–Okamoto transform.

Derandomization: Generate r as cryptographic hash $H(m)$, using standard hash function H. (Watch out: Is m guessable?)

Decryption with reencryption:
1. Input C'. (Maybe $C' \ne C$.)
2. Decrypt to obtain m'.
3. Recompute $r' = H(m')$.
4. Recompute C'' from m', r'.
5. Abort if $C'' \ne C'$.

Subset-s

Attacker
for $(r_1, .$
checks r
against

This tak
e.g. 102

"This fi

— This
applicati
encryptio

— Also,
to find a

tosystem
se problems.

t encryption
ryption by
t separately.
ess for each bit.

ge
$) \in \{0,1\}^B$.
$. ., B\}$:
$r_{i,N} \in \{0,1\}$.

$\cdots + r_{1,N}K_N)$,

$\cdots + r_{B,N}K_N)$.

2. Derandomize encryption, and reencrypt during decryption.

This is an example of "FO", the 1999 Fujisaki–Okamoto transform.

Derandomization: Generate r as cryptographic hash $H(m)$, using standard hash function H. (Watch out: Is m guessable?)

Decryption with reencryption:
1. Input C'. (Maybe $C' \neq C$.)
2. Decrypt to obtain m'.
3. Recompute $r' = H(m')$.
4. Recompute C'' from m', r'.
5. Abort if $C'' \neq C'$.

Subset-sum attack

Attacker searches
for $(r_1, \ldots, r_N)$,
checks $r_1K_1 + \cdots$
against $\pm C_1$.

This takes 2^N easy
e.g. 1024 operatio

"This finds only o

— This is a proble
applications. Shou
encryption to leak

— Also, can easily
to find all bits of r

s.

on

ly.

ch bit.

^{B}.

$1\}$.

$K_N)$,

$_N K_N)$.

2. Derandomize encryption, and reencrypt during decryption.

This is an example of "FO", the 1999 Fujisaki–Okamoto transform.

Derandomization: Generate r as cryptographic hash $H(m)$, using standard hash function H. (Watch out: Is m guessable?)

Decryption with reencryption:
1. Input C'. (Maybe $C' \neq C$.)
2. Decrypt to obtain m'.
3. Recompute $r' = H(m')$.
4. Recompute C'' from m', r'.
5. Abort if $C'' \neq C'$.

Subset-sum attacks

Attacker searches all possibi
for $(r_1, \ldots, r_N)$,
checks $r_1 K_1 + \cdots + r_N K_N$
against $\pm C_1$.

This takes 2^N easy operatio
e.g. 1024 operations for $N =$

"This finds only one bit m_1.

— This is a problem in som
applications. Should design
encryption to leak *no* inform

— Also, can easily modify a
to find all bits of message.

2. Derandomize encryption, and reencrypt during decryption.

This is an example of “FO”, the 1999 Fujisaki–Okamoto transform.

Derandomization: Generate r as cryptographic hash $H(m)$, using standard hash function H. (Watch out: Is m guessable?)

Decryption with reencryption:
1. Input C'. (Maybe $C' \neq C$.)
2. Decrypt to obtain m'.
3. Recompute $r' = H(m')$.
4. Recompute C'' from m', r'.
5. Abort if $C'' \neq C'$.

Subset-sum attacks

Attacker searches all possibilities for $(r_1, \ldots, r_N)$, checks $r_1 K_1 + \cdots + r_N K_N$ against $\pm C_1$.

This takes 2^N easy operations: e.g. 1024 operations for $N = 10$.

“This finds only one bit m_1.”

— This is a problem in some applications. Should design encryption to leak *no* information.

— Also, can easily modify attack to find all bits of message.

ndomize encryption, and
pt during decryption.

an example of "FO", the
jisaki–Okamoto transform.

omization: Generate r
ographic hash $H(m)$,
andard hash function H.
out: Is m guessable?)

ion with reencryption:
C'. (Maybe $C' \neq C$.)
ypt to obtain m'.
mpute $r' = H(m')$.
mpute C'' from m', r'.
t if $C'' \neq C'$.

<u>Subset-sum attacks</u>

Attacker searches all possibilities for $(r_1, \ldots, r_N)$, checks $r_1K_1 + \cdots + r_NK_N$ against $\pm C_1$.

This takes 2^N easy operations: e.g. 1024 operations for $N = 10$.

"This finds only one bit m_1."

— This is a problem in some applications. Should design encryption to leak *no* information.

— Also, can easily modify attack to find all bits of message.

Modifie
For each
$r_1K_1 +$
containi

Multi-ta
Apply th
one mes
message

Finding
total 2^N

Finding
message
total 0.0

ncryption, and
lecryption.

e of “FO”, the
moto transform.

Generate r
ash $H(m)$,
sh function H.
guessable?)

eencryption:
ybe $C' \neq C$.)
ain m'.
$= H(m')$.
from m', r'.
C'.

Subset-sum attacks

Attacker searches all possibilities for $(r_1, \ldots, r_N)$, checks $r_1K_1 + \cdots + r_NK_N$ against $\pm C_1$.

This takes 2^N easy operations: e.g. 1024 operations for $N = 10$.

“This finds only one bit m_1.”

— This is a problem in some applications. Should design encryption to leak *no* information.

— Also, can easily modify attack to find all bits of message.

Modified attack:
For each $(r_1, \ldots, r$
$r_1K_1 + \cdots + r_NK$
containing $\pm C_1, \pm$

Multi-target attac
Apply this not just
one message, but
messages sent to t

Finding all bits in
total 2^N operation

Finding 1% of all
messages, huge in
total $0.01 \cdot 2^N$ ope

and

, the
nsform.

r

,

n H.

?)

n:

C.)

r'.

Subset-sum attacks

Attacker searches all possibilities for $(r_1, \ldots, r_N)$,
checks $r_1 K_1 + \cdots + r_N K_N$
against $\pm C_1$.

This takes 2^N easy operations: e.g. 1024 operations for $N = 10$.

"This finds only one bit m_1."

— This is a problem in some applications. Should design encryption to leak *no* information.

— Also, can easily modify attack to find all bits of message.

Modified attack:
For each $(r_1, \ldots, r_N)$, look u
$r_1 K_1 + \cdots + r_N K_N$ in hash
containing $\pm C_1, \pm C_2, \ldots, \pm$

Multi-target attack:
Apply this not just to B bits
one message, but all bits in
messages sent to this key.

Finding all bits in all messag
total 2^N operations.

Finding 1% of all bits in all
messages, huge information
total $0.01 \cdot 2^N$ operations.

Subset-sum attacks

Attacker searches all possibilities for $(r_1, \dots, r_N)$, checks $r_1 K_1 + \cdots + r_N K_N$ against $\pm C_1$.

This takes 2^N easy operations: e.g. 1024 operations for $N = 10$.

"This finds only one bit m_1."

— This is a problem in some applications. Should design encryption to leak *no* information.

— Also, can easily modify attack to find all bits of message.

Modified attack:
For each $(r_1, \dots, r_N)$, look up $r_1 K_1 + \cdots + r_N K_N$ in hash table containing $\pm C_1, \pm C_2, \dots, \pm C_B$.

Multi-target attack:
Apply this not just to B bits in one message, but all bits in all messages sent to this key.

Finding all bits in all messages: total 2^N operations.

Finding 1% of all bits in all messages, huge information leak: total $0.01 \cdot 2^N$ operations.

sum attacks

r searches all possibilities
$\ldots, r_N)$,
$_1K_1 + \cdots + r_NK_N$
$\pm C_1$.

es 2^N easy operations:
4 operations for $N = 10$.

nds only one bit m_1.”

is a problem in some
ions. Should design
on to leak *no* information.

can easily modify attack
ll bits of message.

Modified attack:
For each $(r_1, \ldots, r_N)$, look up $r_1K_1 + \cdots + r_NK_N$ in hash table containing $\pm C_1, \pm C_2, \ldots, \pm C_B$.

Multi-target attack:
Apply this not just to B bits in one message, but all bits in all messages sent to this key.

Finding all bits in all messages: total 2^N operations.

Finding 1% of all bits in all messages, huge information leak: total $0.01 \cdot 2^N$ operations.

“We can
$N = 128$
day, and
transform

— Stand
take only
to find (
with r_1K

Make ha
$C - r_{N/2}$
for all (r

Look up
hash tab

ks

all possibilities

$+ r_N K_N$

y operations:

ns for $N = 10$.

ne bit m_1."

em in some

uld design

no information.

y modify attack

message.

Modified attack:

For each $(r_1, \ldots, r_N)$, look up $r_1 K_1 + \cdots + r_N K_N$ in hash table containing $\pm C_1, \pm C_2, \ldots, \pm C_B$.

Multi-target attack:

Apply this not just to B bits in one message, but all bits in all messages sent to this key.

Finding all bits in all messages: total 2^N operations.

Finding 1% of all bits in all messages, huge information leak: total $0.01 \cdot 2^N$ operations.

"We can stop atta

$N = 128$, and cha

day, and applying

transform to each

— Standard subse

take only $2^{N/2}$ ope

to find $(r_1, \ldots, r_N$

with $r_1 K_1 + \cdots +$

Make hash table c

$C - r_{N/2+1} K_{N/2+1}$

for all $(r_{N/2+1}, \ldots$

Look up $r_1 K_1 + \cdot$

hash table for eac

lities

ns:

= 10.

”

e

ation.

ttack

Modified attack:

For each $(r_1, \ldots, r_N)$, look up $r_1 K_1 + \cdots + r_N K_N$ in hash table containing $\pm C_1, \pm C_2, \ldots, \pm C_B$.

Multi-target attack:

Apply this not just to B bits in one message, but all bits in all messages sent to this key.

Finding all bits in all messages: total 2^N operations.

Finding 1% of all bits in all messages, huge information leak: total $0.01 \cdot 2^N$ operations.

“We can stop attacks by tak

$N = 128$, and changing keys

day, and applying all-or-noth

transform to each message.”

— Standard subset-sum atta

take only $2^{N/2}$ operations

to find $(r_1, \ldots, r_N) \in \{0, 1\}$

with $r_1 K_1 + \cdots + r_N K_N =$

Make hash table containing

$C - r_{N/2+1} K_{N/2+1} - \cdots -$

for all $(r_{N/2+1}, \ldots, r_N)$.

Look up $r_1 K_1 + \cdots + r_{N/2}$

hash table for each $(r_1, \ldots,$

Modified attack:
For each $(r_1, \ldots, r_N)$, look up $r_1 K_1 + \cdots + r_N K_N$ in hash table containing $\pm C_1, \pm C_2, \ldots, \pm C_B$.

Multi-target attack:
Apply this not just to B bits in one message, but all bits in all messages sent to this key.

Finding all bits in all messages: total 2^N operations.

Finding 1% of all bits in all messages, huge information leak: total $0.01 \cdot 2^N$ operations.

"We can stop attacks by taking $N = 128$, and changing keys every day, and applying all-or-nothing transform to each message."

— Standard subset-sum attacks take only $2^{N/2}$ operations to find $(r_1, \ldots, r_N) \in \{0, 1\}^N$ with $r_1 K_1 + \cdots + r_N K_N = C$.

Make hash table containing $C - r_{N/2+1} K_{N/2+1} - \cdots - r_N K_N$ for all $(r_{N/2+1}, \ldots, r_N)$.

Look up $r_1 K_1 + \cdots + r_{N/2} K_{N/2}$ in hash table for each $(r_1, \ldots, r_{N/2})$.

d attack:
n $(r_1, \ldots, r_N)$, look up
$\cdots + r_N K_N$ in hash table
ng $\pm C_1, \pm C_2, \ldots, \pm C_B$.

rget attack:
his not just to B bits in
sage, but all bits in all
s sent to this key.

all bits in all messages:
operations.

1% of all bits in all
s, huge information leak:
01 $\cdot\ 2^N$ operations.

"We can stop attacks by taking $N = 128$, and changing keys every day, and applying all-or-nothing transform to each message."

— Standard subset-sum attacks take only $2^{N/2}$ operations to find $(r_1, \ldots, r_N) \in \{0, 1\}^N$ with $r_1 K_1 + \cdots + r_N K_N = C$.

Make hash table containing $C - r_{N/2+1} K_{N/2+1} - \cdots - r_N K_N$ for all $(r_{N/2+1}, \ldots, r_N)$.

Look up $r_1 K_1 + \cdots + r_{N/2} K_{N/2}$ in hash table for each $(r_1, \ldots, r_{N/2})$.

These at
structure
one targ

(Actually
$\pm C_1, \ldots$
Convert
total B^1
to find a
have mo

There ar
exploit t

1981 Sc
$2^{N/2}$ ope

r_N), look up
$_N$ in hash table
$\pm C_2, \ldots, \pm C_B$.

k:
t to B bits in
all bits in all
this key.

all messages:
ns.

bits in all
formation leak:
erations.

"We can stop attacks by taking $N = 128$, and changing keys every day, and applying all-or-nothing transform to each message."

— Standard subset-sum attacks take only $2^{N/2}$ operations to find $(r_1, \ldots, r_N) \in \{0,1\}^N$ with $r_1 K_1 + \cdots + r_N K_N = C$.

Make hash table containing $C - r_{N/2+1} K_{N/2+1} - \cdots - r_N K_N$ for all $(r_{N/2+1}, \ldots, r_N)$.

Look up $r_1 K_1 + \cdots + r_{N/2} K_{N/2}$ in hash table for each $(r_1, \ldots, r_{N/2})$.

These attacks exp
structure of proble
one target C into

(Actually have $2B$
$\pm C_1, \ldots, \pm C_B$ for
Convert into $B^{1/2}$
total $B^{1/2} 2^{N/2}$ op
to find all B bits.
have more messag

There are even mo
exploit the linear s

1981 Schroeppel–S
$2^{N/2}$ operations, s

up

table

C_B.

s in

all

ges:

leak:

“We can stop attacks by taking $N = 128$, and changing keys every day, and applying all-or-nothing transform to each message.”

— Standard subset-sum attacks take only $2^{N/2}$ operations to find $(r_1, \ldots, r_N) \in \{0, 1\}^N$ with $r_1 K_1 + \cdots + r_N K_N = C$.

Make hash table containing $C - r_{N/2+1} K_{N/2+1} - \cdots - r_N K_N$ for all $(r_{N/2+1}, \ldots, r_N)$.

Look up $r_1 K_1 + \cdots + r_{N/2} K_{N/2}$ in hash table for each $(r_1, \ldots, r_{N/2})$.

These attacks exploit linear structure of problem to conv one target C into many targ

(Actually have $2B$ targets $\pm C_1, \ldots, \pm C_B$ for one mess Convert into $B^{1/2} 2^{N/2}$ targe total $B^{1/2} 2^{N/2}$ operations to find all B bits. Also, may have more messages to atta

There are even more ways to exploit the linear structure.

1981 Schroeppel–Shamir: $2^{N/2}$ operations, space $2^{N/4}$

“We can stop attacks by taking $N = 128$, and changing keys every day, and applying all-or-nothing transform to each message.”

— Standard subset-sum attacks take only $2^{N/2}$ operations to find $(r_1, \ldots, r_N) \in \{0, 1\}^N$ with $r_1 K_1 + \cdots + r_N K_N = C$.

Make hash table containing $C - r_{N/2+1} K_{N/2+1} - \cdots - r_N K_N$ for all $(r_{N/2+1}, \ldots, r_N)$.

Look up $r_1 K_1 + \cdots + r_{N/2} K_{N/2}$ in hash table for each $(r_1, \ldots, r_{N/2})$.

These attacks exploit linear structure of problem to convert one target C into many targets.

(Actually have $2B$ targets $\pm C_1, \ldots, \pm C_B$ for one message. Convert into $B^{1/2} 2^{N/2}$ targets: total $B^{1/2} 2^{N/2}$ operations to find all B bits. Also, maybe have more messages to attack.)

There are even more ways to exploit the linear structure.

1981 Schroeppel–Shamir: $2^{N/2}$ operations, space $2^{N/4}$.

n stop attacks by taking
3, and changing keys every
applying all-or-nothing
m to each message."

dard subset-sum attacks
y $2^{N/2}$ operations
$r_1, \ldots, r_N) \in \{0,1\}^N$
$K_1 + \cdots + r_N K_N = C$.

ash table containing
$_{2+1} K_{N/2+1} - \cdots - r_N K_N$
$r_{N/2+1}, \ldots, r_N)$.

$r_1 K_1 + \cdots + r_{N/2} K_{N/2}$ in
ble for each $(r_1, \ldots, r_{N/2})$.

These attacks exploit linear structure of problem to convert one target C into many targets.

(Actually have $2B$ targets $\pm C_1, \ldots, \pm C_B$ for one message. Convert into $B^{1/2} 2^{N/2}$ targets: total $B^{1/2} 2^{N/2}$ operations to find all B bits. Also, maybe have more messages to attack.)

There are even more ways to exploit the linear structure.

1981 Schroeppel–Shamir: $2^{N/2}$ operations, space $2^{N/4}$.

2010 Ho
claimed
May–Me

2011 Be
$2^{0.291N}$

2016 Oz

2019 Es
operatio

2020 Bo
Schrotte

Quantur

Multi-ta

acks by taking
nging keys every
all-or-nothing
message.”

t-sum attacks
erations
$) \in \{0, 1\}^N$
$r_N K_N = C$.

ontaining
$_1 - \cdots - r_N K_N$
$, r_N)$.

$\cdots + r_{N/2} K_{N/2}$ in
n $(r_1, \ldots, r_{N/2})$.

These attacks exploit linear structure of problem to convert one target C into many targets.

(Actually have $2B$ targets $\pm C_1, \ldots, \pm C_B$ for one message. Convert into $B^{1/2} 2^{N/2}$ targets: total $B^{1/2} 2^{N/2}$ operations to find all B bits. Also, maybe have more messages to attack.)

There are even more ways to exploit the linear structure.

1981 Schroeppel–Shamir: $2^{N/2}$ operations, space $2^{N/4}$.

2010 Howgrave-Gr
claimed $2^{0.311N}$ op
May–Meurer corre

2011 Becker–Coro
$2^{0.291N}$ operations

2016 Ozerov: $2^{0.2}$

2019 Esser–May:
operations, but wi

2020 Bonnetain–B
Schrottenloher–Sh

Quantum attacks:

Multi-target speed

king
s every
ning

acks

N

C.

$r_N K_N$

$K_{N/2}$ in
$r_{N/2}$).

These attacks exploit linear structure of problem to convert one target C into many targets.

(Actually have $2B$ targets $\pm C_1, \ldots, \pm C_B$ for one message. Convert into $B^{1/2} 2^{N/2}$ targets: total $B^{1/2} 2^{N/2}$ operations to find all B bits. Also, maybe have more messages to attack.)

There are even more ways to exploit the linear structure.

1981 Schroeppel–Shamir: $2^{N/2}$ operations, space $2^{N/4}$.

2010 Howgrave-Graham–Jou
claimed $2^{0.311N}$ operations.
May–Meurer correction: $2^{0.3}$

2011 Becker–Coron–Joux:
$2^{0.291N}$ operations.

2016 Ozerov: $2^{0.287N}$ opera

2019 Esser–May: claimed 2^{0}
operations, but withdrew cla

2020 Bonnetain–Bricout–
Schrottenloher–Shen: $2^{0.283}$

Quantum attacks: various p

Multi-target speedups: prob

These attacks exploit linear structure of problem to convert one target C into many targets.

(Actually have $2B$ targets $\pm C_1, \ldots, \pm C_B$ for one message. Convert into $B^{1/2}2^{N/2}$ targets: total $B^{1/2}2^{N/2}$ operations to find all B bits. Also, maybe have more messages to attack.)

There are even more ways to exploit the linear structure.

1981 Schroeppel–Shamir: $2^{N/2}$ operations, space $2^{N/4}$.

2010 Howgrave-Graham–Joux: claimed $2^{0.311N}$ operations. 2011 May–Meurer correction: $2^{0.337N}$.

2011 Becker–Coron–Joux: $2^{0.291N}$ operations.

2016 Ozerov: $2^{0.287N}$ operations.

2019 Esser–May: claimed $2^{0.255N}$ operations, but withdrew claim.

2020 Bonnetain–Bricout–Schrottenloher–Shen: $2^{0.283N}$.

Quantum attacks: various papers.

Multi-target speedups: probably!

ttacks exploit linear
e of problem to convert
et C into many targets.

y have $2B$ targets
, $\pm C_B$ for one message.
into $B^{1/2}2^{N/2}$ targets:
$/2 2^{N/2}$ operations
ll B bits. Also, maybe
re messages to attack.)

re even more ways to
he linear structure.

hroeppel–Shamir:
erations, space $2^{N/4}$.

2010 Howgrave-Graham–Joux: claimed $2^{0.311N}$ operations. 2011 May–Meurer correction: $2^{0.337N}$.

2011 Becker–Coron–Joux: $2^{0.291N}$ operations.

2016 Ozerov: $2^{0.287N}$ operations.

2019 Esser–May: claimed $2^{0.255N}$ operations, but withdrew claim.

2020 Bonnetain–Bricout–Schrottenloher–Shen: $2^{0.283N}$.

Quantum attacks: various papers.

Multi-target speedups: probably!

Variants

2003 Re
(without
$(-1)^m(r$
$m(K_1/2$

To make
modify k
and $(K_1$
Also be

2009 va
Vaikunta
$C = m +$
$m = (C$
Be caref

loit linear
em to convert
many targets.

targets
one message.
$2^{N/2}$ targets:
erations
Also, maybe
es to attack.)

ore ways to
structure.

Shamir:
pace $2^{N/4}$.

2010 Howgrave-Graham–Joux: claimed $2^{0.311N}$ operations. 2011 May–Meurer correction: $2^{0.337N}$.

2011 Becker–Coron–Joux: $2^{0.291N}$ operations.

2016 Ozerov: $2^{0.287N}$ operations.

2019 Esser–May: claimed $2^{0.255N}$ operations, but withdrew claim.

2020 Bonnetain–Bricout–Schrottenloher–Shen: $2^{0.283N}$.

Quantum attacks: various papers.

Multi-target speedups: probably!

Variants of crypto

2003 Regev: Cohe
(without credit), b
$(-1)^m(r_1K_1 + \cdots$
$m(K_1/2) + r_1K_1$ –

To make this work
modify keygen to
and $(K_1 - u_1)/s$ ∈
Also be careful wit

2009 van Dijk–Ge
Vaikuntanathan:
$C = m + r_1K_1 + \cdot$
$m = (C \bmod s)$ m
Be careful to take

vert
gets.

age.
ets:

ybe
ck.)

o

.

2010 Howgrave-Graham–Joux: claimed $2^{0.311N}$ operations. 2011 May–Meurer correction: $2^{0.337N}$.

2011 Becker–Coron–Joux: $2^{0.291N}$ operations.

2016 Ozerov: $2^{0.287N}$ operations.

2019 Esser–May: claimed $2^{0.255N}$ operations, but withdrew claim.

2020 Bonnetain–Bricout–Schrottenloher–Shen: $2^{0.283N}$.

Quantum attacks: various papers.

Multi-target speedups: probably!

Variants of cryptosystem

2003 Regev: Cohen cryptosy
(without credit), but replace
$(-1)^m(r_1K_1 + \cdots + r_NK_N)$
$m(K_1/2) + r_1K_1 + \cdots + r_N$

To make this work,
modify keygen to force $K_1 \in$
and $(K_1 - u_1)/s \in 1 + 2\mathbf{Z}$.
Also be careful with u_i boun

2009 van Dijk–Gentry–Halev
Vaikuntanathan: $K_i \in 2u_i +$
$C = m + r_1K_1 + \cdots + r_NK_N$
$m = (C \bmod s) \bmod 2$.
Be careful to take $s \in 1 + 2$

2010 Howgrave-Graham–Joux: claimed $2^{0.311N}$ operations. 2011 May–Meurer correction: $2^{0.337N}$.

2011 Becker–Coron–Joux: $2^{0.291N}$ operations.

2016 Ozerov: $2^{0.287N}$ operations.

2019 Esser–May: claimed $2^{0.255N}$ operations, but withdrew claim.

2020 Bonnetain–Bricout–Schrottenloher–Shen: $2^{0.283N}$.

Quantum attacks: various papers.

Multi-target speedups: probably!

Variants of cryptosystem

2003 Regev: Cohen cryptosystem (without credit), but replace $(-1)^m(r_1K_1 + \cdots + r_NK_N)$ with $m(K_1/2) + r_1K_1 + \cdots + r_NK_N$.

To make this work,
modify keygen to force $K_1 \in 2\mathbf{Z}$
and $(K_1 - u_1)/s \in 1 + 2\mathbf{Z}$.
Also be careful with u_i bounds.

2009 van Dijk–Gentry–Halevi–Vaikuntanathan: $K_i \in 2u_i + s\mathbf{Z}$;
$C = m + r_1K_1 + \cdots + r_NK_N$;
$m = (C \bmod s) \bmod 2$.
Be careful to take $s \in 1 + 2\mathbf{Z}$.

owgrave-Graham–Joux:
$2^{0.311N}$ operations. 2011
eurer correction: $2^{0.337N}$.

cker–Coron–Joux:
operations.

erov: $2^{0.287N}$ operations.

ser–May: claimed $2^{0.255N}$
ns, but withdrew claim.

onnetain–Bricout–
enloher–Shen: $2^{0.283N}$.

m attacks: various papers.

rget speedups: probably!

Variants of cryptosystem

2003 Regev: Cohen cryptosystem (without credit), but replace $(-1)^m(r_1K_1 + \cdots + r_NK_N)$ with $m(K_1/2) + r_1K_1 + \cdots + r_NK_N$.

To make this work,
modify keygen to force $K_1 \in 2\mathbf{Z}$ and $(K_1 - u_1)/s \in 1 + 2\mathbf{Z}$.
Also be careful with u_i bounds.

2009 van Dijk–Gentry–Halevi–Vaikuntanathan: $K_i \in 2u_i + s\mathbf{Z}$;
$C = m + r_1K_1 + \cdots + r_NK_N$;
$m = (C \bmod s) \bmod 2$.
Be careful to take $s \in 1 + 2\mathbf{Z}$.

Homomo

If u_i/s is
DGHV s

Take tw
$C = m +$
$C' = m'$
with sma

$C + C' =$
$s(q + q'$
$m + m'$

$CC' = n$
$s(\cdots)$.
mm' if e

raham–Joux:

perations. 2011

ction: $2^{0.337N}$.

n–Joux:

s.

^{87N} operations.

claimed $2^{0.255N}$

thdrew claim.

Bricout–

en: $2^{0.283N}$.

various papers.

lups: probably!

Variants of cryptosystem

2003 Regev: Cohen cryptosystem (without credit), but replace $(-1)^m(r_1K_1 + \cdots + r_NK_N)$ with $m(K_1/2) + r_1K_1 + \cdots + r_NK_N$.

To make this work,
modify keygen to force $K_1 \in 2\mathbf{Z}$
and $(K_1 - u_1)/s \in 1 + 2\mathbf{Z}$.
Also be careful with u_i bounds.

2009 van Dijk–Gentry–Halevi–Vaikuntanathan: $K_i \in 2u_i + s\mathbf{Z}$;
$C = m + r_1K_1 + \cdots + r_NK_N$;
$m = (C \bmod s) \bmod 2$.
Be careful to take $s \in 1 + 2\mathbf{Z}$.

Homomorphic enc

If u_i/s is small enc

DGHV system is h

Take two ciphertex

$C = m + 2\epsilon + sq,$

$C' = m' + 2\epsilon' + s$

with small $\epsilon, \epsilon' \in$

$C + C' = m + m'$

$s(q + q')$. This de

$m + m'$ mod 2 if ϵ

$CC' = mm' + 2(\epsilon m$

$s(\cdots)$. This decry

mm' if $\epsilon m' + \epsilon' m$

ux:

2011

$337N$.

tions.

$0.255N$

aim.

N .

apers.

ably!

Variants of cryptosystem

2003 Regev: Cohen cryptosystem (without credit), but replace $(-1)^m(r_1K_1 + \cdots + r_NK_N)$ with $m(K_1/2) + r_1K_1 + \cdots + r_NK_N$.

To make this work, modify keygen to force $K_1 \in 2\mathbf{Z}$ and $(K_1 - u_1)/s \in 1 + 2\mathbf{Z}$. Also be careful with u_i bounds.

2009 van Dijk–Gentry–Halevi–Vaikuntanathan: $K_i \in 2u_i + s\mathbf{Z}$; $C = m + r_1K_1 + \cdots + r_NK_N$; $m = (C \bmod s) \bmod 2$. Be careful to take $s \in 1 + 2\mathbf{Z}$.

Homomorphic encryption

If u_i/s is small enough then DGHV system is homomorp

Take two ciphertexts: $C = m + 2\epsilon + sq$, $C' = m' + 2\epsilon' + sq'$ with small $\epsilon, \epsilon' \in \mathbf{Z}$.

$C + C' = m + m' + 2(\epsilon + \epsilon'$ $s(q + q')$. This decrypts to $m + m' \bmod 2$ if $\epsilon + \epsilon'$ is sm

$CC' = mm' + 2(\epsilon m' + \epsilon' m + 2$ $s(\cdots)$. This decrypts to mm' if $\epsilon m' + \epsilon' m + 2\epsilon\epsilon'$ is s

Variants of cryptosystem

2003 Regev: Cohen cryptosystem (without credit), but replace $(-1)^m(r_1K_1 + \cdots + r_NK_N)$ with $m(K_1/2) + r_1K_1 + \cdots + r_NK_N$.

To make this work, modify keygen to force $K_1 \in 2\mathbf{Z}$ and $(K_1 - u_1)/s \in 1 + 2\mathbf{Z}$. Also be careful with u_i bounds.

2009 van Dijk–Gentry–Halevi–Vaikuntanathan: $K_i \in 2u_i + s\mathbf{Z}$; $C = m + r_1K_1 + \cdots + r_NK_N$; $m = (C \bmod s) \bmod 2$. Be careful to take $s \in 1 + 2\mathbf{Z}$.

Homomorphic encryption

If u_i/s is small enough then 2009 DGHV system is homomorphic.

Take two ciphertexts:
$C = m + 2\epsilon + sq$,
$C' = m' + 2\epsilon' + sq'$
with small $\epsilon, \epsilon' \in \mathbf{Z}$.

$C + C' = m + m' + 2(\epsilon + \epsilon') + s(q + q')$. This decrypts to $m + m' \bmod 2$ if $\epsilon + \epsilon'$ is small.

$CC' = mm' + 2(\epsilon m' + \epsilon' m + 2\epsilon\epsilon') + s(\cdots)$. This decrypts to mm' if $\epsilon m' + \epsilon' m + 2\epsilon\epsilon'$ is small.

of cryptosystem

gev: Cohen cryptosystem
credit), but replace
$r_1 K_1 + \cdots + r_N K_N$) with
$) + r_1 K_1 + \cdots + r_N K_N$.

e this work,
keygen to force $K_1 \in 2\mathbf{Z}$
$- u_1)/s \in 1 + 2\mathbf{Z}$.
careful with u_i bounds.

n Dijk–Gentry–Halevi–
anathan: $K_i \in 2u_i + s\mathbf{Z}$;
$+ r_1 K_1 + \cdots + r_N K_N$;
mod s) mod 2.
ful to take $s \in 1 + 2\mathbf{Z}$.

Homomorphic encryption

If u_i/s is small enough then 2009 DGHV system is homomorphic.

Take two ciphertexts:
$C = m + 2\epsilon + sq$,
$C' = m' + 2\epsilon' + sq'$
with small $\epsilon, \epsilon' \in \mathbf{Z}$.

$C + C' = m + m' + 2(\epsilon + \epsilon') + s(q + q')$. This decrypts to $m + m'$ mod 2 if $\epsilon + \epsilon'$ is small.

$CC' = mm' + 2(\epsilon m' + \epsilon' m + 2\epsilon\epsilon') + s(\cdots)$. This decrypts to mm' if $\epsilon m' + \epsilon' m + 2\epsilon\epsilon'$ is small.

```
sage: N=
sage: E=
sage: Y=
sage: X=
sage: s=
sage: s
98488730
sage: u=
....:
sage: u
[247, 41
 772, 20
sage:
```

system

en cryptosystem
out replace
$+ r_N K_N)$ with
$+ \cdots + r_N K_N$.

k,
force $K_1 \in 2\mathbf{Z}$
$\in 1 + 2\mathbf{Z}$.
th u_i bounds.

ntry–Halevi–
$K_i \in 2u_i + s\mathbf{Z}$;
$\cdots + r_N K_N$;
od 2.
$s \in 1 + 2\mathbf{Z}$.

Homomorphic encryption

If u_i/s is small enough then 2009 DGHV system is homomorphic.

Take two ciphertexts:
$C = m + 2\epsilon + sq$,
$C' = m' + 2\epsilon' + sq'$
with small $\epsilon, \epsilon' \in \mathbf{Z}$.

$C + C' = m + m' + 2(\epsilon + \epsilon') + s(q + q')$. This decrypts to $m + m'$ mod 2 if $\epsilon + \epsilon'$ is small.

$CC' = mm' + 2(\epsilon m' + \epsilon' m + 2\epsilon\epsilon') + s(\cdots)$. This decrypts to mm' if $\epsilon m' + \epsilon' m + 2\epsilon\epsilon'$ is small.

```
sage: N=10
sage: E=2^10
sage: Y=2^50
sage: X=2^80
sage: s=1+2*rand
sage: s
984887308997925
sage: u=[randran
....:    for i i
sage: u
[247, 418, 365,
 772, 209, 673,
sage:
```

ystem
e
with
K_N.

$\in 2\mathbf{Z}$

nds.

vi–
sZ;
N;

$\mathbf{Z}$.

Homomorphic encryption

If u_i/s is small enough then 2009 DGHV system is homomorphic.

Take two ciphertexts:
$C = m + 2\epsilon + sq$,
$C' = m' + 2\epsilon' + sq'$
with small $\epsilon, \epsilon' \in \mathbf{Z}$.

$C + C' = m + m' + 2(\epsilon + \epsilon') + s(q + q')$. This decrypts to $m + m'$ mod 2 if $\epsilon + \epsilon'$ is small.

$CC' = mm' + 2(\epsilon m' + \epsilon' m + 2\epsilon\epsilon') + s(\cdots)$. This decrypts to mm' if $\epsilon m' + \epsilon' m + 2\epsilon\epsilon'$ is small.

```
sage: N=10
sage: E=2^10
sage: Y=2^50
sage: X=2^80
sage: s=1+2*randrange(Y/4
sage: s
984887308997925
sage: u=[randrange(E)
....:    for i in range(N
sage: u
[247, 418, 365, 738, 123,
 772, 209, 673, 47]
sage:
```

Homomorphic encryption

If u_i/s is small enough then 2009 DGHV system is homomorphic.

Take two ciphertexts:
$C = m + 2\epsilon + sq$,
$C' = m' + 2\epsilon' + sq'$
with small $\epsilon, \epsilon' \in \mathbf{Z}$.

$C + C' = m + m' + 2(\epsilon + \epsilon') + s(q + q')$. This decrypts to $m + m'$ mod 2 if $\epsilon + \epsilon'$ is small.

$CC' = mm' + 2(\epsilon m' + \epsilon' m + 2\epsilon\epsilon') + s(\cdots)$. This decrypts to mm' if $\epsilon m' + \epsilon' m + 2\epsilon\epsilon'$ is small.


```
sage: N=10
sage: E=2^10
sage: Y=2^50
sage: X=2^80
sage: s=1+2*randrange(Y/4,Y/2)
sage: s
984887308997925
sage: u=[randrange(E)
....:    for i in range(N)]
sage: u
[247, 418, 365, 738, 123, 735,
 772, 209, 673, 47]
sage:
```

orphic encryption

s small enough then 2009
ystem is homomorphic.

o ciphertexts:
$+ 2\epsilon + sq$,
$+ 2\epsilon' + sq'$
all $\epsilon, \epsilon' \in \mathbf{Z}$.

$= m + m' + 2(\epsilon + \epsilon') +$
). This decrypts to
mod 2 if $\epsilon + \epsilon'$ is small.

$mm' + 2(\epsilon m' + \epsilon' m + 2\epsilon\epsilon') +$
This decrypts to
$\epsilon m' + \epsilon' m + 2\epsilon\epsilon'$ is small.


```
sage: N=10
sage: E=2^10
sage: Y=2^50
sage: X=2^80
sage: s=1+2*randrange(Y/4,Y/2)
sage: s
984887308997925
sage: u=[randrange(E)
....:    for i in range(N)]
sage: u
[247, 418, 365, 738, 123, 735,
 772, 209, 673, 47]
sage:
```

```
sage: K
....:
....:
....:
sage: K
[587473
 -11115
 794301
 688178
 742362
 102334
 -35716
 112142
 -11096
 -23562
```

ryption

ough then 2009
homomorphic.

xts:

q'
$\mathbf{Z}$.

$+ 2(\epsilon + \epsilon') +$
ecrypts to
$+ \epsilon'$ is small.

$n' + \epsilon' m + 2\epsilon\epsilon') +$
pts to
$+ 2\epsilon\epsilon'$ is small.


```
sage: N=10
sage: E=2^10
sage: Y=2^50
sage: X=2^80
sage: s=1+2*randrange(Y/4,Y/2)
sage: s
984887308997925
sage: u=[randrange(E)
....:    for i in range(N)]
sage: u
[247, 418, 365, 738, 123, 735,
 772, 209, 673, 47]
sage:
```

```
sage: K=[2*ui+s*
....:       ceil(
....:       floor
....:    for ui
sage: K
[587473338058640
 -11115391791007
 79430145953378З
 688178021083749
 742362470968200
 102334582783153
 -35716867939855
 112142161911996
 -11096748622762
 -23562893778500
```

2009
hic.

) +

nall.

$2\epsilon\epsilon'$) +

small.


```
sage: N=10
sage: E=2^10
sage: Y=2^50
sage: X=2^80
sage: s=1+2*randrange(Y/4,Y/2)
sage: s
984887308997925
sage: u=[randrange(E)
....:     for i in range(N)]
sage: u
[247, 418, 365, 738, 123, 735,
 772, 209, 673, 47]
sage:
```

```
sage: K=[2*ui+s*randrange
....:        ceil(-(X+2*ui)
....:        floor((X-2*ui)
....:     for ui in u]
sage: K
[587473338058640662659869
 -11115391791007200837703
 794301459533783434896055
 68817802108374958901751,
 742362470968200823035396
 102334582783153951505479
 -35716867939855887673000
 112142161911996460105144
 -11096748622762224955871
 -23562893778500377052338
```

```
sage: N=10
sage: E=2^10
sage: Y=2^50
sage: X=2^80
sage: s=1+2*randrange(Y/4,Y/2)
sage: s
984887308997925
sage: u=[randrange(E)
....:    for i in range(N)]
sage: u
[247, 418, 365, 738, 123, 735,
 772, 209, 673, 47]
sage:
```



```
sage: K=[2*ui+s*randrange(
....:       ceil(-(X+2*ui)/s),
....:       floor((X-2*ui)/s)+1)
....:     for ui in u]
sage: K
[587473338058640662659869,
 -1111539179100720083770339,
 794301459533783434896055,
 688178021083749589017751,
 742362470968200823035396,
 1023345827831539515054795,
 -357168679398558876730006,
 1121421619119964601051443,
 -1109674862276222495587129,
 -235628937785003770523381]
```

```
=10
=2^10
=2^50
=2^80
=1+2*randrange(Y/4,Y/2)

08997925
=[randrange(E)
  for i in range(N)]

18, 365, 738, 123, 735,
09, 673, 47]
```



```
sage: K=[2*ui+s*randrange(
....:       ceil(-(X+2*ui)/s),
....:       floor((X-2*ui)/s)+1)
....:     for ui in u]
sage: K
[587473338058640662659869,
 -1111539179100720083770339,
 794301459533783434896055,
 68817802108374958901751,
 742362470968200823035396,
 1023345827831539515054795,
 -357168679398558876730006,
 1121421619119964601051443,
 -1109674862276222495587129,
 -235628937785003770523381]
```

```
sage: m
sage: r
....:
sage: C
....:
sage: C
2094088
sage: C
2703
sage: (
1
sage: m
1
sage:
```

```
range(Y/4,Y/2)

ge(E)
n range(N)]

738, 123, 735,
47]
```



```
sage: K=[2*ui+s*randrange(
....:       ceil(-(X+2*ui)/s),
....:       floor((X-2*ui)/s)+1)
....:     for ui in u]
sage: K
[587473338058640662659869,
 -1111539179100720083770339,
 794301459533783434896055,
 68817802108374958901751,
 742362470968200823035396,
 1023345827831539515054795,
 -357168679398558876730006,
 1121421619119964601051443,
 -1109674862276222495587129,
 -235628937785003770523381]
```



```
sage: m=randrang
sage: r=[randran
....:     for i i
sage: C=m+sum(r[
....:     for i i
sage: C
2094088748748247
sage: C%s
2703
sage: (C%s)%2
1
sage: m
1
sage:
```

```
,Y/2)
)]
 735,
```

```
sage: K=[2*ui+s*randrange(
....:      ceil(-(X+2*ui)/s),
....:      floor((X-2*ui)/s)+1)
....:    for ui in u]
sage: K
[587473338058640662659869,
 -1111539179100720083770339,
 794301459533783434896055,
 68817802108374958901751,
 742362470968200823035396,
 1023345827831539515054795,
 -357168679398558876730006,
 1121421619119964601051443,
 -1109674862276222495587129,
 -235628937785003770523381]
```



```
sage: m=randrange(2)
sage: r=[randrange(2)
....:    for i in range(N
sage: C=m+sum(r[i]*K[i]
....:    for i in range(N
sage: C
2094088748748247210016703
sage: C%s
2703
sage: (C%s)%2
1
sage: m
1
sage:
```

```
sage: K=[2*ui+s*randrange(
....:      ceil(-(X+2*ui)/s),
....:      floor((X-2*ui)/s)+1)
....:    for ui in u]
sage: K
[587473338058640662659869,
 -1111539179100720083770339,
 794301459533783434896055,
 68817802108374958901751,
 742362470968200823035396,
 1023345827831539515054795,
 -357168679398558876730006,
 1121421619119964601051443,
 -1109674862276222495587129,
 -235628937785003770523381]
```



```
sage: m=randrange(2)
sage: r=[randrange(2)
....:    for i in range(N)]
sage: C=m+sum(r[i]*K[i]
....:    for i in range(N))
sage: C
2094088748748247210016703
sage: C%s
2703
sage: (C%s)%2
1
sage: m
1
sage:
```

```
=[2*ui+s*randrange(
    ceil(-(X+2*ui)/s),
    floor((X-2*ui)/s)+1)
  for ui in u]

33805864066265986 9,
39179100720083770339,
45953378343489605 5,
02108374958901751,
47096820082303539 6,
58278315395150547 95,
86793985588767300 06,
16191199646010514 43,
74862276222495587129,
89377850037052338 1]
```



```
sage: m=randrange(2)
sage: r=[randrange(2)
....:    for i in range(N)]
sage: C=m+sum(r[i]*K[i]
....:    for i in range(N))
sage: C
2094088748748247210016703
sage: C%s
2703
sage: (C%s)%2
1
sage: m
1
sage:
```

```
sage: m
sage: r
....:
sage: C
....:
sage: C
-517223
sage: C
4971
sage: (
1
sage: m
1
sage:
```

```
randrange(
-(X+2*ui)/s),
((X-2*ui)/s)+1)
in u]

662659869,
20083770339,
434896055,
58901751,
823035396,
9515054795,
8876730006,
4601051443,
22495587129,
3770523381]
```



```
sage: m=randrange(2)
sage: r=[randrange(2)
....:     for i in range(N)]
sage: C=m+sum(r[i]*K[i]
....:     for i in range(N))
sage: C
20940887487482472100167032
sage: C%s
2703
sage: (C%s)%2
1
sage: m
1
sage:
```

```
sage: m2=randran
sage: r2=[randra
....:     for i
sage: C2=m2+sum(
....:     for i
sage: C2
-517223537379827
sage: C2%s
4971
sage: (C2%s)%2
1
sage: m2
1
sage:
```

```
(
/s),
/s)+1)

,
39,
,

,
5,
6,
3,
29,
1]
```



```
sage: m=randrange(2)
sage: r=[randrange(2)
....:     for i in range(N)]
sage: C=m+sum(r[i]*K[i]
....:     for i in range(N))
sage: C
20940887487482472100167O3
sage: C%s
2703
sage: (C%s)%2
1
sage: m
1
sage:
```

```
sage: m2=randrange(2)
sage: r2=[randrange(2)
....:     for i in range(
sage: C2=m2+sum(r2[i]*K[i
....:     for i in range(
sage: C2
-5172235373798273727012 9
sage: C2%s
4971
sage: (C2%s)%2
1
sage: m2
1
sage:
```

```
sage: m=randrange(2)
sage: r=[randrange(2)
....:     for i in range(N)]
sage: C=m+sum(r[i]*K[i]
....:     for i in range(N))
sage: C
20940887487482472100016703
sage: C%s
2703
sage: (C%s)%2
1
sage: m
1
sage:
```



```
sage: m2=randrange(2)
sage: r2=[randrange(2)
....:     for i in range(N)]
sage: C2=m2+sum(r2[i]*K[i]
....:     for i in range(N))
sage: C2
-51722353737982737270129
sage: C2%s
4971
sage: (C2%s)%2
1
sage: m2
1
sage:
```

```
=randrange(2)
=[randrange(2)
 for i in range(N)]
=m+sum(r[i]*K[i]
 for i in range(N))

74874824721001670з
%s

C%s)%2
```



```
sage: m2=randrange(2)
sage: r2=[randrange(2)
....:     for i in range(N)]
sage: C2=m2+sum(r2[i]*K[i]
....:     for i in range(N))
sage: C2
-51722353737982737270129
sage: C2%s
4971
sage: (C2%s)%2
1
sage: m2
1
sage:
```

```
sage: (C
7674
sage: (C
13436613
sage:
```

Because
are smal
have C –
(C' mod
(C mod

Refineme
to ciphe
Gentry)

```
e(2)
ge(2)
n range(N)]
i]*K[i]
n range(N))

210016703
```



```
sage: m2=randrange(2)
sage: r2=[randrange(2)
....:     for i in range(N)]
sage: C2=m2+sum(r2[i]*K[i]
....:     for i in range(N))
sage: C2
-51722353737982737270129
sage: C2%s
4971
sage: (C2%s)%2
1
sage: m2
1
sage:
```

```
sage: (C+C2)%s
7674
sage: (C*C2)%s
13436613
sage:
```

Because C mod s
are small enough
have $C + C'$ mod s
(C' mod s) and C
(C mod s)(C' mod

Refinements: add
to ciphertexts, boo
Gentry) to control

```
sage: m2=randrange(2)
sage: r2=[randrange(2)
....:     for i in range(N)]
sage: C2=m2+sum(r2[i]*K[i]
....:     for i in range(N))
sage: C2
-51722353737982737270129
sage: C2%s
4971
sage: (C2%s)%2
1
sage: m2
1
sage:
```



```
sage: (C+C2)%s
7674
sage: (C*C2)%s
13436613
sage:
```

Because C mod s and C' mo
are small enough compared
have $C + C'$ mod $s = (C$ mo
$(C'$ mod $s)$ and CC' mod s
$(C$ mod $s)(C'$ mod $s)$.

Refinements: add more nois
to ciphertexts, bootstrap (20
Gentry) to control noise, etc

```
sage: m2=randrange(2)
sage: r2=[randrange(2)
....:     for i in range(N)]
sage: C2=m2+sum(r2[i]*K[i]
....:     for i in range(N))
sage: C2
-5172235373798273727O129
sage: C2%s
4971
sage: (C2%s)%2
1
sage: m2
1
sage:
```



```
sage: (C+C2)%s
7674
sage: (C*C2)%s
13436613
sage:
```

Because C mod s and C' mod s are small enough compared to s, have $C + C'$ mod $s = (C$ mod $s) +$ $(C'$ mod $s)$ and CC' mod $s =$ $(C$ mod $s)(C'$ mod $s)$.

Refinements: add more noise to ciphertexts, bootstrap (2009 Gentry) to control noise, etc.

```
2=randrange(2)
2=[randrange(2)
   for i in range(N)]
2=m2+sum(r2[i]*K[i]
   for i in range(N))
2
53737982737270129
2%s

C2%s)%2

2
```



```
sage: (C+C2)%s
7674
sage: (C*C2)%s
13436613
sage:
```

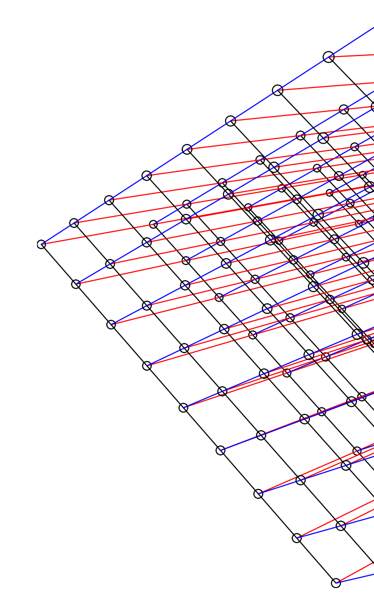
Because C mod s and C' mod s are small enough compared to s, have $C + C'$ mod $s = (C$ mod $s) + (C'$ mod $s)$ and CC' mod s $=$ $(C$ mod $s)(C'$ mod $s)$.

Refinements: add more noise to ciphertexts, bootstrap (2009 Gentry) to control noise, etc.

Lattices

This is a

This is a



```
ge(2)
nge(2)
in range(N)]
r2[i]*K[i]
in range(N))

37270129
```



```
sage: (C+C2)%s
7674
sage: (C*C2)%s
13436613
sage:
```

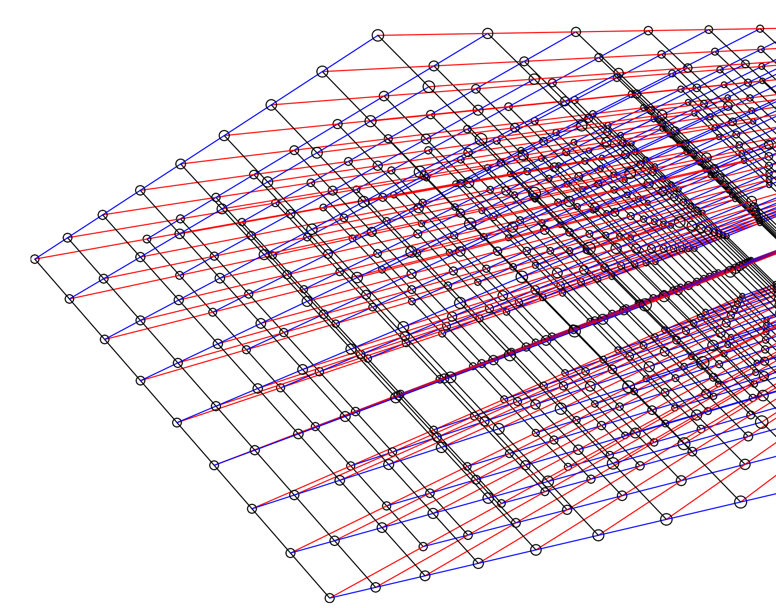
Because C mod s and C' mod s are small enough compared to s, have $C + C'$ mod $s = (C$ mod $s) + (C'$ mod $s)$ and CC' mod $s = (C$ mod $s)(C'$ mod $s)$.

Refinements: add more noise to ciphertexts, bootstrap (2009 Gentry) to control noise, etc.

Lattices

This is a lettuce:

This is a lattice:



```
N)]
]
N))
```



```
sage: (C+C2)%s
7674
sage: (C*C2)%s
13436613
sage:
```

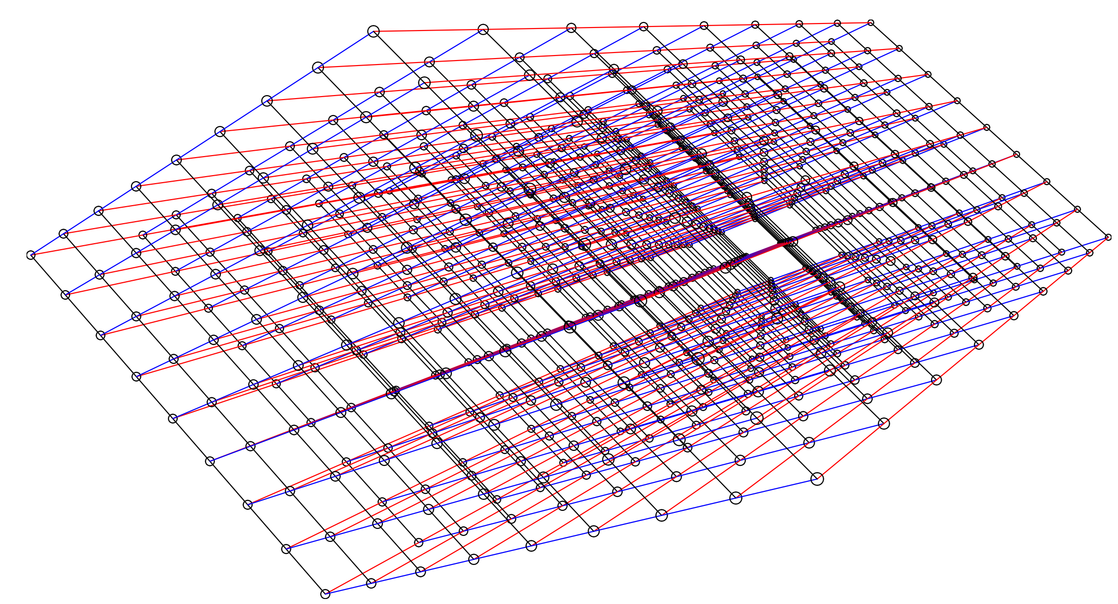
Because C mod s and C' mod s are small enough compared to s, have $C + C'$ mod $s = (C$ mod $s) + (C'$ mod $s)$ and CC' mod $s = (C$ mod $s)(C'$ mod $s)$.

Refinements: add more noise to ciphertexts, bootstrap (2009 Gentry) to control noise, etc.

Lattices

This is a lettuce:

This is a lattice:



```
sage: (C+C2)%s
7674
sage: (C*C2)%s
13436613
sage:
```

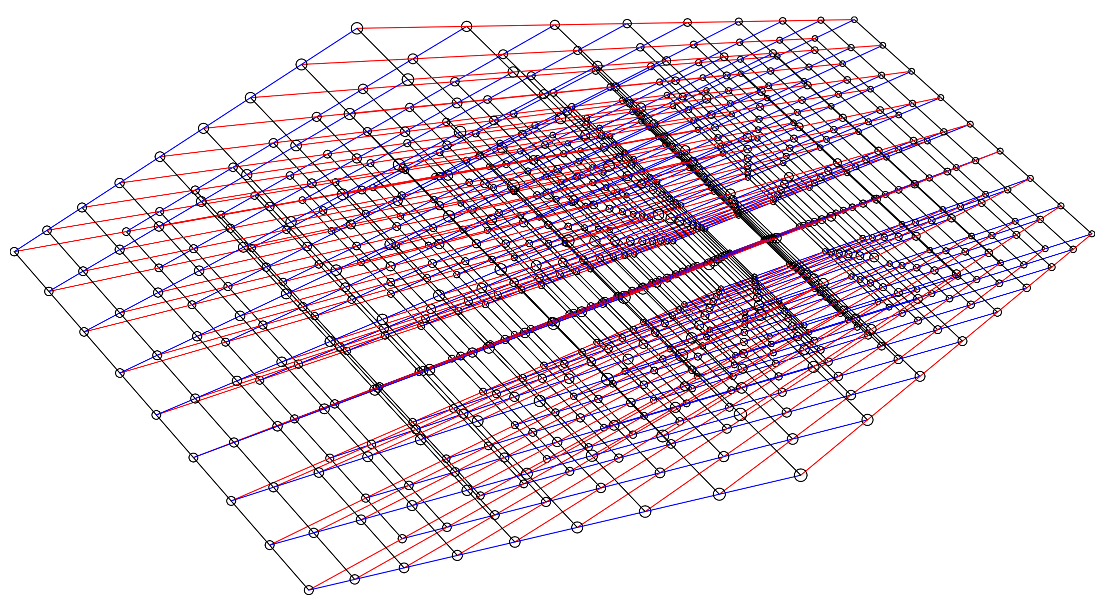
Because C mod s and C' mod s are small enough compared to s, have $C + C'$ mod $s = (C$ mod $s) + (C'$ mod $s)$ and CC' mod $s = (C$ mod $s)(C'$ mod $s)$.

Refinements: add more noise to ciphertexts, bootstrap (2009 Gentry) to control noise, etc.

Lattices

This is a lettuce:

This is a lattice:



```
C+C2)%s
```

```
C*C2)%s
3
```

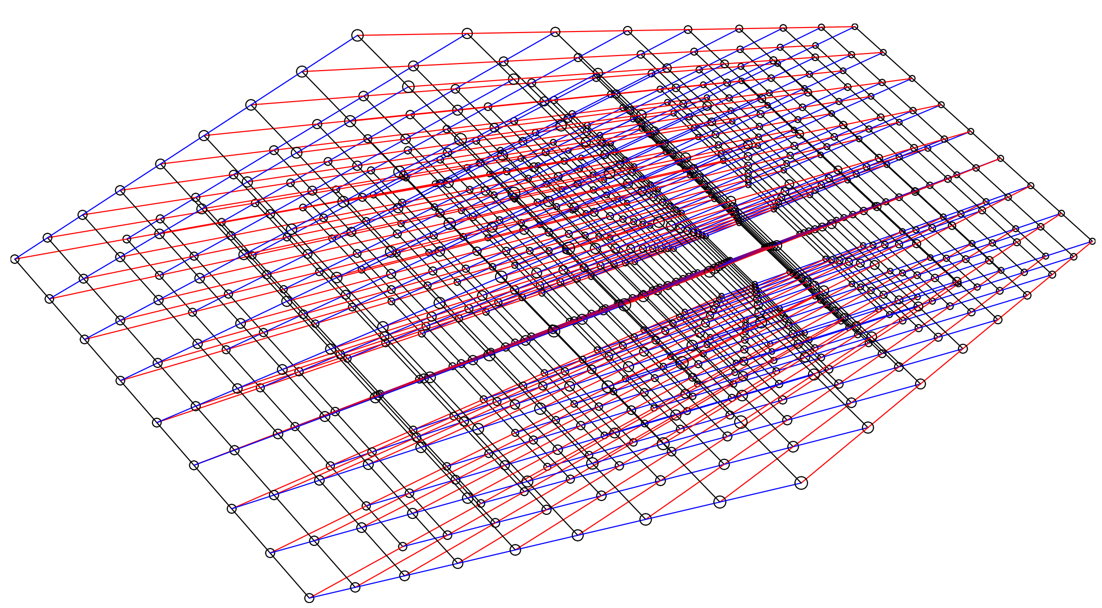
C mod s and C' mod s
l enough compared to s,
$+\,C'$ mod $s = (C$ mod $s) +$
$s)$ and CC' mod s $=$
$s)(C'$ mod $s)$.

ents: add more noise
rtexts, bootstrap (2009
to control noise, etc.

Lattices

This is a lettuce:

This is a lattice:



Lattices,

Assume
are **R**-lin
i.e., $\mathbf{R}V_1$
$\{r_1V_1 +$
is a D-d

$\mathbf{Z}V_1 + \cdots$
$\{r_1V_1 +$
is a rank

$V_1, \ldots, V$
is a **basi**

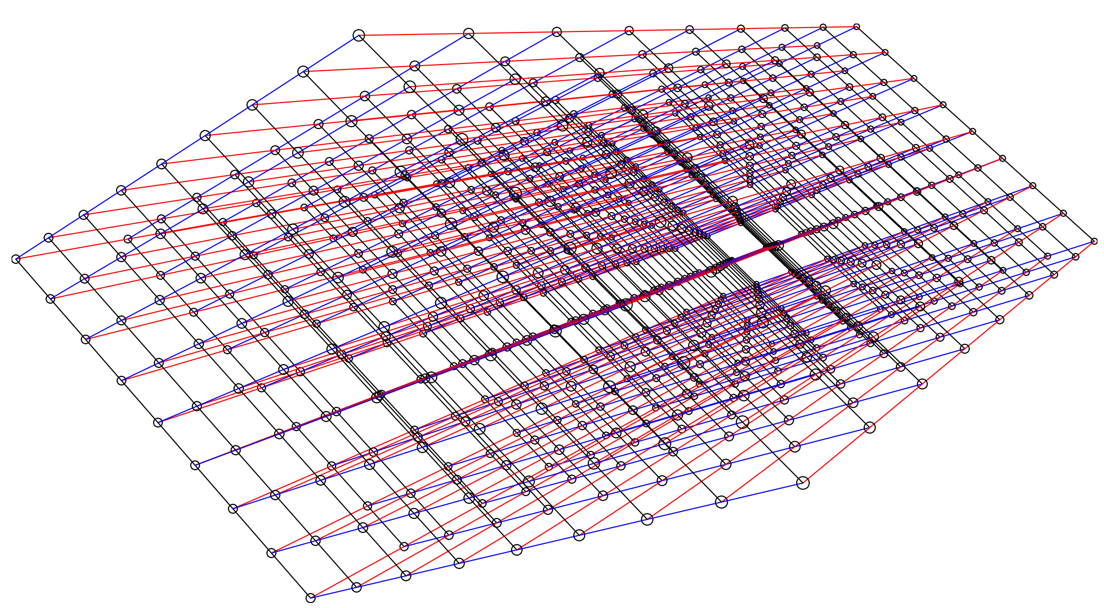
and C' mod s
compared to s,
$s = (C \bmod s) +$
$C' \bmod s =$
s).

more noise
otstrap (2009
noise, etc.

Lattices

This is a lettuce:

This is a lattice:



Lattices, mathema

Assume that $V_1, \ldots$
are **R**-linearly inde
i.e., $\mathbf{R}V_1 + \cdots + \mathbf{R}$
$\{r_1V_1 + \cdots + r_DV$
is a D-dimensiona

$\mathbf{Z}V_1 + \cdots + \mathbf{Z}V_D =$
$\{r_1V_1 + \cdots + r_DV$
is a rank-D length

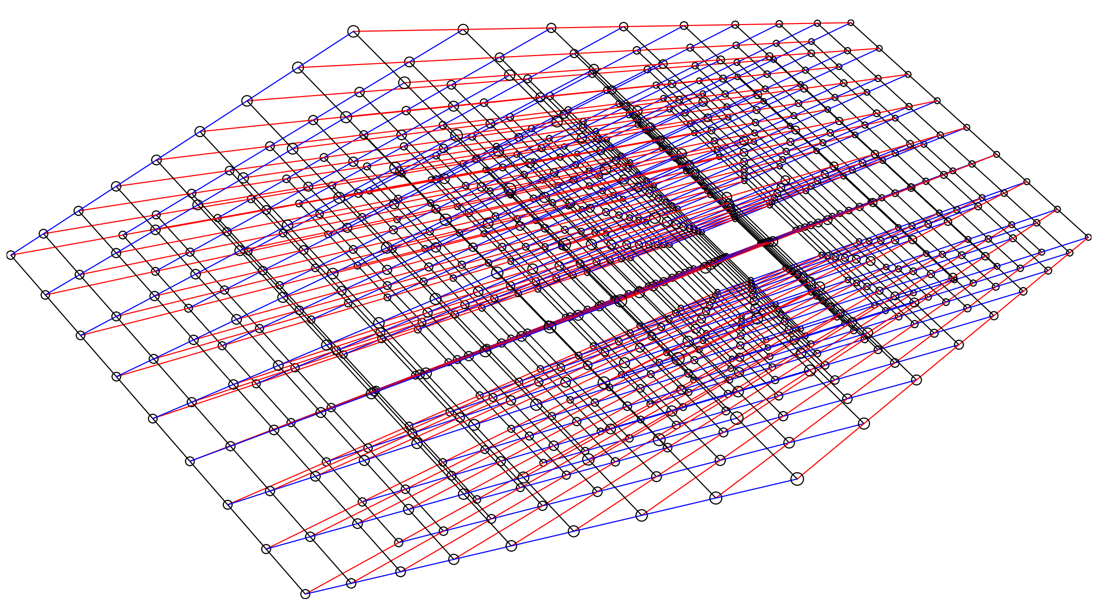
$V_1, \ldots, V_D$
is a **basis** of this l

od s
to s,
od $s) +$
$=$
e
009
c.

Lattices

This is a lettuce:

This is a lattice:



Lattices, mathematically

Assume that $V_1, \ldots, V_D \in \mathbf{R}$
are $\mathbf{R}$-linearly independent,
i.e., $\mathbf{R}V_1 + \cdots + \mathbf{R}V_D =$
$\{r_1V_1 + \cdots + r_DV_D : r_1, \ldots,$
is a D-dimensional vector sp

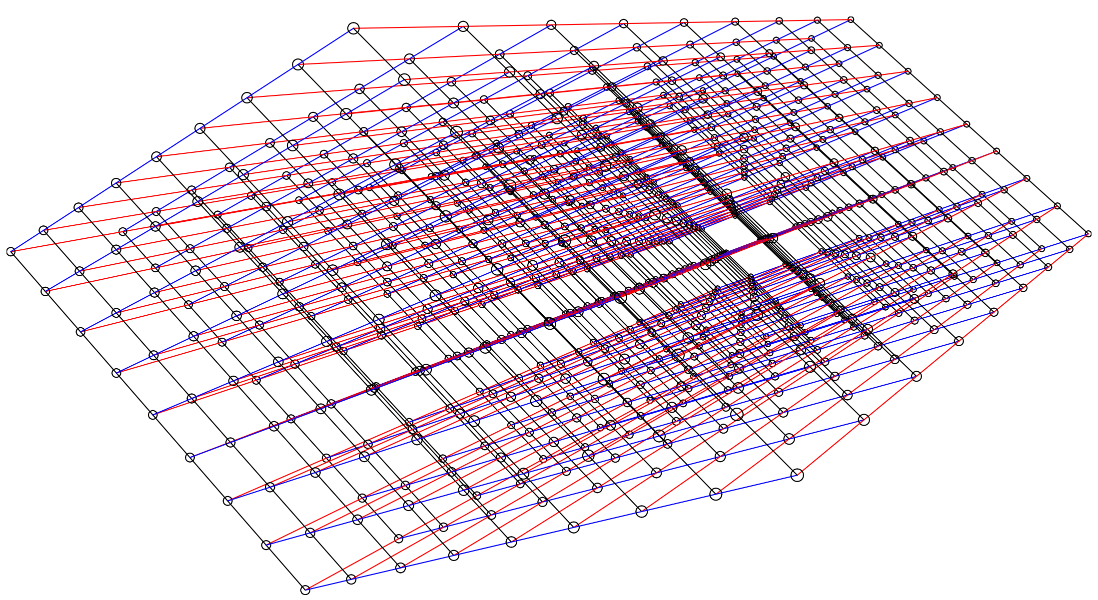
$\mathbf{Z}V_1 + \cdots + \mathbf{Z}V_D =$
$\{r_1V_1 + \cdots + r_DV_D : r_1, \ldots,$
is a rank-D length-N **lattice**

$V_1, \ldots, V_D$
is a **basis** of this lattice.

Lattices

This is a lettuce:

This is a lattice:



Lattices, mathematically

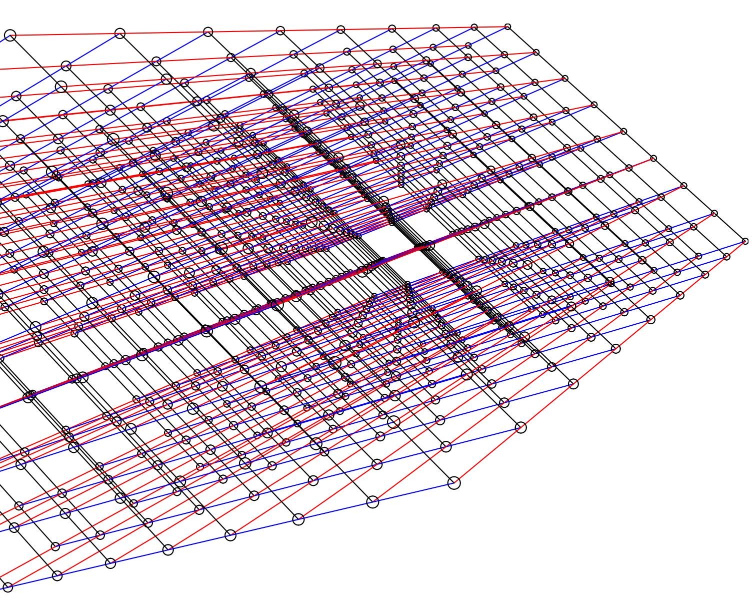
Assume that $V_1, \dots, V_D \in \mathbf{R}^N$ are $\mathbf{R}$-linearly independent, i.e., $\mathbf{R}V_1 + \cdots + \mathbf{R}V_D = \{r_1V_1 + \cdots + r_DV_D : r_1, \dots, r_D \in \mathbf{R}\}$ is a D-dimensional vector space.

$\mathbf{Z}V_1 + \cdots + \mathbf{Z}V_D = \{r_1V_1 + \cdots + r_DV_D : r_1, \dots, r_D \in \mathbf{Z}\}$ is a rank-D length-N **lattice**.

$V_1, \dots, V_D$ is a **basis** of this lattice.

n lettuce:

n lattice:



Lattices, mathematically

Assume that $V_1, \ldots, V_D \in \mathbf{R}^N$
are $\mathbf{R}$-linearly independent,
i.e., $\mathbf{R}V_1 + \cdots + \mathbf{R}V_D =$
$\{r_1V_1 + \cdots + r_DV_D : r_1, \ldots, r_D \in \mathbf{R}\}$
is a D-dimensional vector space.

$\mathbf{Z}V_1 + \cdots + \mathbf{Z}V_D =$
$\{r_1V_1 + \cdots + r_DV_D : r_1, \ldots, r_D \in \mathbf{Z}\}$
is a rank-D length-N **lattice**.

$V_1, \ldots, V_D$
is a **basis** of this lattice.

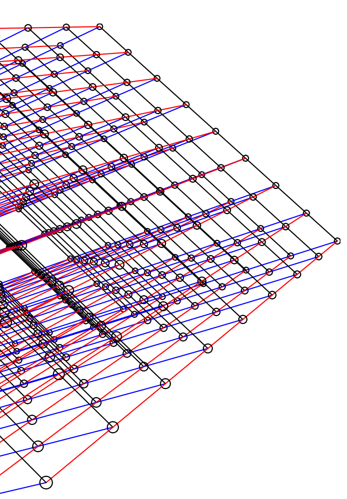
Short ve

Given V_1
what is s
in $L = \mathbf{Z}$

0.

"SVP: s
What is

1982 Le
(LLL) al
compute
with len
length o
Typically



Lattices, mathematically

Assume that $V_1, \ldots, V_D \in \mathbf{R}^N$ are $\mathbf{R}$-linearly independent, i.e., $\mathbf{R}V_1 + \cdots + \mathbf{R}V_D =$ $\{r_1V_1 + \cdots + r_DV_D : r_1, \ldots, r_D \in \mathbf{R}\}$ is a D-dimensional vector space.

$\mathbf{Z}V_1 + \cdots + \mathbf{Z}V_D =$ $\{r_1V_1 + \cdots + r_DV_D : r_1, \ldots, r_D \in \mathbf{Z}\}$ is a rank-D length-N **lattice**.

$V_1, \ldots, V_D$ is a **basis** of this lattice.

Short vectors in la

Given $V_1, V_2, \ldots, V$
what is shortest ve
in $L = \mathbf{Z}V_1 + \cdots$

0.

"SVP: shortest-ve
What is shortest n

1982 Lenstra–Lens
(LLL) algorithm ru
computes a nonze
with length at mo
length of shortest
Typically $\approx 1.02^D$

Lattices, mathematically

Assume that $V_1, \ldots, V_D \in \mathbf{R}^N$
are $\mathbf{R}$-linearly independent,
i.e., $\mathbf{R}V_1 + \cdots + \mathbf{R}V_D =$
$\{r_1V_1 + \cdots + r_DV_D : r_1, \ldots, r_D \in \mathbf{R}\}$
is a D-dimensional vector space.

$\mathbf{Z}V_1 + \cdots + \mathbf{Z}V_D =$
$\{r_1V_1 + \cdots + r_DV_D : r_1, \ldots, r_D \in \mathbf{Z}\}$
is a rank-D length-N **lattice**.

$V_1, \ldots, V_D$
is a **basis** of this lattice.

Short vectors in lattices

Given $V_1, V_2, \ldots, V_D \in \mathbf{Z}^N$,
what is shortest vector
in $L = \mathbf{Z}V_1 + \cdots + \mathbf{Z}V_D$?

0.

"SVP: shortest-vector proble
What is shortest nonzero ve

1982 Lenstra–Lenstra–Lovás
(LLL) algorithm runs in poly
computes a nonzero vector i
with length at most $2^{D/2}$ ti
length of shortest nonzero v
Typically $\approx 1.02^D$ instead of

Lattices, mathematically

Assume that $V_1, \ldots, V_D \in \mathbf{R}^N$
are **R**-linearly independent,
i.e., $\mathbf{R}V_1 + \cdots + \mathbf{R}V_D =$
$\{r_1V_1 + \cdots + r_DV_D : r_1, \ldots, r_D \in \mathbf{R}\}$
is a D-dimensional vector space.

$\mathbf{Z}V_1 + \cdots + \mathbf{Z}V_D =$
$\{r_1V_1 + \cdots + r_DV_D : r_1, \ldots, r_D \in \mathbf{Z}\}$
is a rank-D length-N **lattice**.

$V_1, \ldots, V_D$
is a **basis** of this lattice.

Short vectors in lattices

Given $V_1, V_2, \ldots, V_D \in \mathbf{Z}^N$,
what is shortest vector
in $L = \mathbf{Z}V_1 + \cdots + \mathbf{Z}V_D$?

0.

"SVP: shortest-vector problem":
What is shortest nonzero vector?

1982 Lenstra–Lenstra–Lovász
(LLL) algorithm runs in poly time,
computes a nonzero vector in L
with length at most $2^{D/2}$ times
length of shortest nonzero vector.
Typically $\approx 1.02^D$ instead of $2^{D/2}$.

, mathematically

that $V_1, \ldots, V_D \in \mathbf{R}^N$
nearly independent,
$+ \cdots + \mathbf{R}V_D =$
$\cdots + r_D V_D : r_1, \ldots, r_D \in \mathbf{R}\}$
imensional vector space.

$\cdots + \mathbf{Z}V_D =$
$\cdots + r_D V_D : r_1, \ldots, r_D \in \mathbf{Z}\}$
k-D length-N **lattice**.

V_D
is of this lattice.

Short vectors in lattices

Given $V_1, V_2, \ldots, V_D \in \mathbf{Z}^N$, what is shortest vector in $L = \mathbf{Z}V_1 + \cdots + \mathbf{Z}V_D$?

0.

"SVP: shortest-vector problem": What is shortest nonzero vector?

1982 Lenstra–Lenstra–Lovász (LLL) algorithm runs in poly time, computes a nonzero vector in L with length at most $2^{D/2}$ times length of shortest nonzero vector. Typically $\approx 1.02^D$ instead of $2^{D/2}$.

Subset-s

One way
where C

Choose
$V_0 = (-$
$V_1 = (K$
$V_2 = (K$
$\ldots,$
$V_N = (K$

Define L
L contai
$V_0 + r_1 V$
$(0, r_1\lambda, \ldots$

tically

$\ldots, V_D \in \mathbf{R}^N$

pendent,

$\mathbf{R}V_D =$

$_D : r_1, \ldots, r_D \in \mathbf{R}\}$

l vector space.

$=$

$_D : r_1, \ldots, r_D \in \mathbf{Z}\}$

-N **lattice**.

attice.

Short vectors in lattices

Given $V_1, V_2, \ldots, V_D \in \mathbf{Z}^N$, what is shortest vector in $L = \mathbf{Z}V_1 + \cdots + \mathbf{Z}V_D$?

0.

"SVP: shortest-vector problem": What is shortest nonzero vector?

1982 Lenstra–Lenstra–Lovász (LLL) algorithm runs in poly time, computes a nonzero vector in L with length at most $2^{D/2}$ times length of shortest nonzero vector. Typically $\approx 1.02^D$ instead of $2^{D/2}$.

Subset-sum lattice

One way to find (

where $C = r_1 K_1 +$

Choose λ. Define

$V_0 = (-C, 0, 0, \ldots$

$V_1 = (K_1, \lambda, 0, \ldots$

$V_2 = (K_2, 0, \lambda, \ldots$

$\ldots,$

$V_N = (K_N, 0, 0, \ldots$

Define $L = \mathbf{Z}V_0 +$

L contains the sho

$V_0 + r_1 V_1 + \cdots +$

$(0, r_1\lambda, \ldots, r_N\lambda)$.

^{N}

$r_D \in \mathbf{R}\}$
ace.

$r_D \in \mathbf{Z}\}$
e.

Short vectors in lattices

Given $V_1, V_2, \ldots, V_D \in \mathbf{Z}^N$, what is shortest vector in $L = \mathbf{Z}V_1 + \cdots + \mathbf{Z}V_D$?

0.

"SVP: shortest-vector problem": What is shortest nonzero vector?

1982 Lenstra–Lenstra–Lovász (LLL) algorithm runs in poly time, computes a nonzero vector in L with length at most $2^{D/2}$ times length of shortest nonzero vector. Typically $\approx 1.02^D$ instead of $2^{D/2}$.

Subset-sum lattices

One way to find $(r_1, \ldots, r_N)$ where $C = r_1 K_1 + \cdots + r_N K$

Choose λ. Define
$V_0 = (-C, 0, 0, \ldots, 0)$,
$V_1 = (K_1, \lambda, 0, \ldots, 0)$,
$V_2 = (K_2, 0, \lambda, \ldots, 0)$,
$\ldots$,
$V_N = (K_N, 0, 0, \ldots, \lambda)$.

Define $L = \mathbf{Z}V_0 + \cdots + \mathbf{Z}V$
L contains the short vector
$V_0 + r_1 V_1 + \cdots + r_N V_N =$
$(0, r_1\lambda, \ldots, r_N\lambda)$.

Short vectors in lattices

Given $V_1, V_2, \ldots, V_D \in \mathbf{Z}^N$, what is shortest vector in $L = \mathbf{Z}V_1 + \cdots + \mathbf{Z}V_D$?

0.

"SVP: shortest-vector problem": What is shortest nonzero vector?

1982 Lenstra–Lenstra–Lovász (LLL) algorithm runs in poly time, computes a nonzero vector in L with length at most $2^{D/2}$ times length of shortest nonzero vector. Typically $\approx 1.02^D$ instead of $2^{D/2}$.

Subset-sum lattices

One way to find $(r_1, \ldots, r_N)$ where $C = r_1K_1 + \cdots + r_NK_N$:

Choose λ. Define
$V_0 = (-C, 0, 0, \ldots, 0)$,
$V_1 = (K_1, \lambda, 0, \ldots, 0)$,
$V_2 = (K_2, 0, \lambda, \ldots, 0)$,
$\ldots$,
$V_N = (K_N, 0, 0, \ldots, \lambda)$.

Define $L = \mathbf{Z}V_0 + \cdots + \mathbf{Z}V_N$. L contains the short vector $V_0 + r_1V_1 + \cdots + r_NV_N = (0, r_1\lambda, \ldots, r_N\lambda)$.

ectors in lattices

, $V_2, \dots, V_D \in \mathbf{Z}^N$,
shortest vector
$\mathbf{Z}V_1 + \cdots + \mathbf{Z}V_D$?

hortest-vector problem":
shortest nonzero vector?

nstra–Lenstra–Lovász
gorithm runs in poly time,
es a nonzero vector in L
gth at most $2^{D/2}$ times
f shortest nonzero vector.
$\approx 1.02^D$ instead of $2^{D/2}$.

Subset-sum lattices

One way to find $(r_1, \dots, r_N)$
where $C = r_1 K_1 + \cdots + r_N K_N$:

Choose λ. Define
$V_0 = (-C, 0, 0, \dots, 0)$,
$V_1 = (K_1, \lambda, 0, \dots, 0)$,
$V_2 = (K_2, 0, \lambda, \dots, 0)$,
$\dots$,
$V_N = (K_N, 0, 0, \dots, \lambda)$.

Define $L = \mathbf{Z}V_0 + \cdots + \mathbf{Z}V_N$.
L contains the short vector
$V_0 + r_1 V_1 + \cdots + r_N V_N =$
$(0, r_1\lambda, \dots, r_N\lambda)$.

LLL is fa
finds thi

1991 Sc
algorithr
LLL find
lattice.
vs.-short

2012 Sc
that mo
subset-s
2011 Be

Is this tr
exponen

ttices

$V_D \in \mathbf{Z}^N$,

ector

$+ \mathbf{Z}V_D$?

ctor problem":

onzero vector?

stra–Lovász

uns in poly time,

ro vector in L

st $2^{D/2}$ times

nonzero vector.

instead of $2^{D/2}$.

Subset-sum lattices

One way to find $(r_1, \ldots, r_N)$ where $C = r_1K_1 + \cdots + r_NK_N$:

Choose λ. Define

$V_0 = (-C, 0, 0, \ldots, 0)$,
$V_1 = (K_1, \lambda, 0, \ldots, 0)$,
$V_2 = (K_2, 0, \lambda, \ldots, 0)$,
$\ldots$,
$V_N = (K_N, 0, 0, \ldots, \lambda)$.

Define $L = \mathbf{Z}V_0 + \cdots + \mathbf{Z}V_N$.
L contains the short vector
$V_0 + r_1V_1 + \cdots + r_NV_N =$
$(0, r_1\lambda, \ldots, r_N\lambda)$.

LLL is fast but alm

finds this short vec

1991 Schnorr–Euc

algorithm spends r

LLL finding shorte

lattice. Many subs

vs.-shortness impro

2012 Schnorr–She

that modern form

subset-sum proble

2011 Becker–Coro

Is this true? Open

exponent of this a

em":
ctor?
sz
time,
n L
mes
ector.
$2^{D/2}$.

Subset-sum lattices

One way to find $(r_1, \ldots, r_N)$ where $C = r_1 K_1 + \cdots + r_N K_N$:

Choose λ. Define
$V_0 = (-C, 0, 0, \ldots, 0)$,
$V_1 = (K_1, \lambda, 0, \ldots, 0)$,
$V_2 = (K_2, 0, \lambda, \ldots, 0)$,
$\ldots$,
$V_N = (K_N, 0, 0, \ldots, \lambda)$.

Define $L = \mathbf{Z}V_0 + \cdots + \mathbf{Z}V_N$.
L contains the short vector
$V_0 + r_1 V_1 + \cdots + r_N V_N = (0, r_1\lambda, \ldots, r_N\lambda)$.

LLL is fast but almost never
finds this short vector in L.

1991 Schnorr–Euchner "BKZ
algorithm spends more time
LLL finding shorter vectors i
lattice. Many subsequent ti
vs.-shortness improvements.

2012 Schnorr–Shevchenko c
that modern form of BKZ so
subset-sum problems faster
2011 Becker–Coron–Joux.

Is this true? Open: What's
exponent of this algorithm?

Subset-sum lattices

One way to find $(r_1, \ldots, r_N)$ where $C = r_1 K_1 + \cdots + r_N K_N$:

Choose λ. Define
$V_0 = (-C, 0, 0, \ldots, 0)$,
$V_1 = (K_1, \lambda, 0, \ldots, 0)$,
$V_2 = (K_2, 0, \lambda, \ldots, 0)$,
$\ldots$,
$V_N = (K_N, 0, 0, \ldots, \lambda)$.

Define $L = \mathbf{Z}V_0 + \cdots + \mathbf{Z}V_N$.
L contains the short vector
$V_0 + r_1 V_1 + \cdots + r_N V_N = (0, r_1\lambda, \ldots, r_N\lambda)$.

LLL is fast but almost never finds this short vector in L.

1991 Schnorr–Euchner "BKZ" algorithm spends more time than LLL finding shorter vectors in any lattice. Many subsequent time-vs.-shortness improvements.

2012 Schnorr–Shevchenko claim that modern form of BKZ solves subset-sum problems faster than 2011 Becker–Coron–Joux.

Is this true? Open: What's the exponent of this algorithm?

sum lattices

y to find $(r_1, \ldots, r_N)$

$= r_1 K_1 + \cdots + r_N K_N$:

λ. Define

$C, 0, 0, \ldots, 0)$,

$_1, \lambda, 0, \ldots, 0)$,

$_2, 0, \lambda, \ldots, 0)$,

$K_N, 0, 0, \ldots, \lambda)$.

$= \mathbf{Z}V_0 + \cdots + \mathbf{Z}V_N$.

ns the short vector

$_1 + \cdots + r_N V_N =$

$\ldots, r_N \lambda)$.

LLL is fast but almost never finds this short vector in L.

1991 Schnorr–Euchner "BKZ" algorithm spends more time than LLL finding shorter vectors in any lattice. Many subsequent time-vs.-shortness improvements.

2012 Schnorr–Shevchenko claim that modern form of BKZ solves subset-sum problems faster than 2011 Becker–Coron–Joux.

Is this true? Open: What's the exponent of this algorithm?

Lattice a

Recall K

Each u_i

Note q_j

Define

$V_1 = (E$

$V_2 = (0,$

$V_3 = (0,$

$\ldots;$

$V_N = (0$

Define L

L contai

$(q_1 E, q_1$

$(q_1 E, 2q$

es

$r_1, \ldots, r_N)$

$\cdots + r_N K_N$:

$, 0),$

$, 0),$

$, 0),$

$., \lambda).$

$\cdots + \mathbf{Z}V_N.$

rt vector

$r_N V_N =$

LLL is fast but almost never finds this short vector in L.

1991 Schnorr–Euchner "BKZ" algorithm spends more time than LLL finding shorter vectors in any lattice. Many subsequent time-vs.-shortness improvements.

2012 Schnorr–Shevchenko claim that modern form of BKZ solves subset-sum problems faster than 2011 Becker–Coron–Joux.

Is this true? Open: What's the exponent of this algorithm?

Lattice attacks on

Recall $K_i = 2u_i +$

Each u_i is small:

Note $q_j K_i - q_i K_j$

Define

$V_1 = (E, K_2, K_3, \ldots$

$V_2 = (0, -K_1, 0, \ldots$

$V_3 = (0, 0, -K_1, \ldots$

$\ldots;$

$V_N = (0, 0, 0, \ldots,$

Define $L = \mathbf{Z}V_1 +$

L contains $q_1 V_1 +$

$(q_1 E, q_1 K_2 - q_2 K$

$(q_1 E, 2q_1 u_2 - 2q_2$

LLL is fast but almost never finds this short vector in L.

1991 Schnorr–Euchner "BKZ" algorithm spends more time than LLL finding shorter vectors in any lattice. Many subsequent time-vs.-shortness improvements.

2012 Schnorr–Shevchenko claim that modern form of BKZ solves subset-sum problems faster than 2011 Becker–Coron–Joux.

Is this true? Open: What's the exponent of this algorithm?

Lattice attacks on DGHV ke

Recall $K_i = 2u_i + sq_i \approx sq_i$
Each u_i is small: $u_i < E$.
Note $q_j K_i - q_i K_j = 2q_j u_i -$

Define
$V_1 = (E, K_2, K_3, \ldots, K_N)$;
$V_2 = (0, -K_1, 0, \ldots, 0)$;
$V_3 = (0, 0, -K_1, \ldots, 0)$;
$\ldots$;
$V_N = (0, 0, 0, \ldots, -K_1)$.

Define $L = \mathbf{Z}V_1 + \cdots + \mathbf{Z}V_N$
L contains $q_1 V_1 + \cdots + q_N V$
$(q_1 E, q_1 K_2 - q_2 K_1, \ldots) =$
$(q_1 E, 2q_1 u_2 - 2q_2 u_1, \ldots)$.

LLL is fast but almost never finds this short vector in L.

1991 Schnorr–Euchner "BKZ" algorithm spends more time than LLL finding shorter vectors in any lattice. Many subsequent time-vs.-shortness improvements.

2012 Schnorr–Shevchenko claim that modern form of BKZ solves subset-sum problems faster than 2011 Becker–Coron–Joux.

Is this true? Open: What's the exponent of this algorithm?

Lattice attacks on DGHV keys

Recall $K_i = 2u_i + sq_i \approx sq_i$.
Each u_i is small: $u_i < E$.
Note $q_j K_i - q_i K_j = 2q_j u_i - 2q_i u_j$.

Define
$V_1 = (E, K_2, K_3, \ldots, K_N)$;
$V_2 = (0, -K_1, 0, \ldots, 0)$;
$V_3 = (0, 0, -K_1, \ldots, 0)$;
$\ldots$;
$V_N = (0, 0, 0, \ldots, -K_1)$.

Define $L = \mathbf{Z}V_1 + \cdots + \mathbf{Z}V_N$.
L contains $q_1 V_1 + \cdots + q_N V_N =$
$(q_1 E, q_1 K_2 - q_2 K_1, \ldots) =$
$(q_1 E, 2q_1 u_2 - 2q_2 u_1, \ldots)$.

ast but almost never
s short vector in L.

hnorr–Euchner "BKZ"
m spends more time than
ling shorter vectors in any
Many subsequent time-
tness improvements.

hnorr–Shevchenko claim
dern form of BKZ solves
um problems faster than
cker–Coron–Joux.

rue? Open: What's the
t of this algorithm?

Lattice attacks on DGHV keys

Recall $K_i = 2u_i + sq_i \approx sq_i$.
Each u_i is small: $u_i < E$.
Note $q_j K_i - q_i K_j = 2q_j u_i - 2q_i u_j$.

Define
$V_1 = (E, K_2, K_3, \ldots, K_N)$;
$V_2 = (0, -K_1, 0, \ldots, 0)$;
$V_3 = (0, 0, -K_1, \ldots, 0)$;
$\ldots$;
$V_N = (0, 0, 0, \ldots, -K_1)$.

Define $L = \mathbf{Z}V_1 + \cdots + \mathbf{Z}V_N$.
L contains $q_1 V_1 + \cdots + q_N V_N =$
$(q_1 E, q_1 K_2 - q_2 K_1, \ldots) =$
$(q_1 E, 2q_1 u_2 - 2q_2 u_1, \ldots)$.

```
sage: V=
sage: V=
sage: V+
sage: V+
sage: V
sage: q(
sage: q(
5964878
sage: r(
9848873(
sage: s
9848873(
sage:
```

most never
ctor in L.

hner "BKZ"
more time than
r vectors in any
sequent time-
ovements.

vchenko claim
of BKZ solves
ms faster than
n–Joux.

: What's the
lgorithm?

Lattice attacks on DGHV keys

Recall $K_i = 2u_i + sq_i \approx sq_i$.
Each u_i is small: $u_i < E$.
Note $q_j K_i - q_i K_j = 2q_j u_i - 2q_i u_j$.

Define
$V_1 = (E, K_2, K_3, \ldots, K_N)$;
$V_2 = (0, -K_1, 0, \ldots, 0)$;
$V_3 = (0, 0, -K_1, \ldots, 0)$;
$\ldots$;
$V_N = (0, 0, 0, \ldots, -K_1)$.

Define $L = \mathbf{Z}V_1 + \cdots + \mathbf{Z}V_N$.
L contains $q_1 V_1 + \cdots + q_N V_N =$
$(q_1 E, q_1 K_2 - q_2 K_1, \ldots) =$
$(q_1 E, 2q_1 u_2 - 2q_2 u_1, \ldots)$.

```
sage: V=matrix.i
sage: V=-K[0]*V
sage: Vtop=copy(
sage: Vtop[0]=E
sage: V[0]=Vtop
sage: q0=V.LLL()
sage: q0
596487875
sage: round(K[0]
984887308997925
sage: s
984887308997925
sage:
```

r

Z"

than

n any

me-

laim

olves

than

the

Lattice attacks on DGHV keys

Recall $K_i = 2u_i + sq_i \approx sq_i$.
Each u_i is small: $u_i < E$.
Note $q_j K_i - q_i K_j = 2q_j u_i - 2q_i u_j$.

Define
$V_1 = (E, K_2, K_3, \ldots, K_N)$;
$V_2 = (0, -K_1, 0, \ldots, 0)$;
$V_3 = (0, 0, -K_1, \ldots, 0)$;
$\ldots$;
$V_N = (0, 0, 0, \ldots, -K_1)$.

Define $L = \mathbf{Z}V_1 + \cdots + \mathbf{Z}V_N$.
L contains $q_1 V_1 + \cdots + q_N V_N =$
$(q_1 E, q_1 K_2 - q_2 K_1, \ldots) =$
$(q_1 E, 2q_1 u_2 - 2q_2 u_1, \ldots)$.

```
sage: V=matrix.identity(N
sage: V=-K[0]*V
sage: Vtop=copy(K)
sage: Vtop[0]=E
sage: V[0]=Vtop
sage: q0=V.LLL()[0][0]/E
sage: q0
596487875
sage: round(K[0]/q0)
984887308997925
sage: s
984887308997925
sage:
```

Lattice attacks on DGHV keys

Recall $K_i = 2u_i + sq_i \approx sq_i$.
Each u_i is small: $u_i < E$.
Note $q_j K_i - q_i K_j = 2q_j u_i - 2q_i u_j$.

Define
$V_1 = (E, K_2, K_3, \ldots, K_N)$;
$V_2 = (0, -K_1, 0, \ldots, 0)$;
$V_3 = (0, 0, -K_1, \ldots, 0)$;
$\ldots$;
$V_N = (0, 0, 0, \ldots, -K_1)$.

Define $L = \mathbf{Z}V_1 + \cdots + \mathbf{Z}V_N$.
L contains $q_1 V_1 + \cdots + q_N V_N =$
$(q_1 E, q_1 K_2 - q_2 K_1, \ldots) =$
$(q_1 E, 2q_1 u_2 - 2q_2 u_1, \ldots)$.


```
sage: V=matrix.identity(N)
sage: V=-K[0]*V
sage: Vtop=copy(K)
sage: Vtop[0]=E
sage: V[0]=Vtop
sage: q0=V.LLL()[0][0]/E
sage: q0
596487875
sage: round(K[0]/q0)
984887308997925
sage: s
984887308997925
sage:
```

attacks on DGHV keys

$K_i = 2u_i + sq_i \approx sq_i$.

is small: $u_i < E$.

$K_i - q_iK_j = 2q_ju_i - 2q_iu_j$.

$, K_2, K_3, \ldots, K_N)$;

$-K_1, 0, \ldots, 0)$;

$0, -K_1, \ldots, 0)$;

$, 0, 0, \ldots, -K_1)$.

$= \mathbf{Z}V_1 + \cdots + \mathbf{Z}V_N$.

ns $q_1V_1 + \cdots + q_NV_N =$

$K_2 - q_2K_1, \ldots) =$

$_1u_2 - 2q_2u_1, \ldots)$.


```
sage: V=matrix.identity(N)
sage: V=-K[0]*V
sage: Vtop=copy(K)
sage: Vtop[0]=E
sage: V[0]=Vtop
sage: q0=V.LLL()[0][0]/E
sage: q0
596487875
sage: round(K[0]/q0)
984887308997925
sage: s
984887308997925
sage:
```

```
sage: V
(1024,
 -11115
 794301
 688178
 742362
 102334
 -35716
 112142
 -11096
 -23562
sage: V
(0, -58
 0, 0, 
sage:
```

DGHV keys

$sq_i \approx sq_i$.

$u_i < E$.

$= 2q_j u_i - 2q_i u_j$.

$\ldots, K_N)$;

$\ldots, 0)$;

$\ldots, 0)$;

$-K_1)$.

$\cdots + \mathbf{Z}V_N$.

$\cdots + q_N V_N =$

$_1, \ldots) =$

$u_1, \ldots)$.


```
sage: V=matrix.identity(N)
sage: V=-K[0]*V
sage: Vtop=copy(K)
sage: Vtop[0]=E
sage: V[0]=Vtop
sage: q0=V.LLL()[0][0]/E
sage: q0
596487875
sage: round(K[0]/q0)
984887308997925
sage: s
984887308997925
sage:
```

```
sage: V[0]
(1024,
 -11115391791007
 79430145953378З
 68817802108374
 74236247096820
 10233458278315
 -3571686793985
 11214216191199
 -1109674862276
 -2356289377850
sage: V[1]
(0, -5874733380
 0, 0, 0, 0, 0,
sage:
```

eys

.

$2q_i u_j$.

N.
$/_N =$

```
sage: V=matrix.identity(N)
sage: V=-K[0]*V
sage: Vtop=copy(K)
sage: Vtop[0]=E
sage: V[0]=Vtop
sage: q0=V.LLL()[0][0]/E
sage: q0
596487875
sage: round(K[0]/q0)
984887308997925
sage: s
984887308997925
sage:
```



```
sage: V[0]
(1024,
 -1111539179100720083770З
 79430145953378343489605‍5
 68817802108374958901751,
 742362470968200823035396
 102334582783153951505479
 -357168679398558876730000
 112142161911996460105144
 -110967486227622249558871
 -235628937785003770523338
sage: V[1]
(0, -587473338058640662 65
 0, 0, 0, 0, 0, 0, 0, 0)
sage:
```

```
sage: V=matrix.identity(N)
sage: V=-K[0]*V
sage: Vtop=copy(K)
sage: Vtop[0]=E
sage: V[0]=Vtop
sage: q0=V.LLL()[0][0]/E
sage: q0
596487875
sage: round(K[0]/q0)
984887308997925
sage: s
984887308997925
sage:
```



```
sage: V[0]
(1024,
 -1111539179100720083770339,
 794301459533783434896055,
 68817802108374958901751,
 742362470968200823035396,
 1023345827831539515054795,
 -357168679398558876730006,
 1121421619119964601051443,
 -1109674862276222495587129,
 -235628937785003770523381)
sage: V[1]
(0, -587473338058640662659869,
 0, 0, 0, 0, 0, 0, 0, 0)
sage:
```

```
=matrix.identity(N)
=-K[0]*V
top=copy(K)
top[0]=E
[0]=Vtop
0=V.LLL()[0][0]/E
0
75
ound(K[0]/q0)
08997925

08997925
```



```
sage: V[0]
(1024,
 -11115391791007200837703339,
 7943014595337834348960 55,
 6881780210837495890175 1,
 7423624709682008230353 96,
 10233458278315395150547 95,
 -35716867939855887673000 6,
 11214216191199646010514 43,
 -11096748622762224955871 29,
 -23562893778500377052338 1)
sage: V[1]
(0, -58747333805864066265986 9,
 0, 0, 0, 0, 0, 0, 0, 0)
sage:
```

```
sage: V
(610803
 370302
 -22561
 110012
 135946
sage: q
sage: q
6108035
sage: q
1056189
sage: q
1742566
sage:
```

```
dentity(N)

K)

[0][0]/E

/q0)
```



```
sage: V[0]
(1024,
 -1115391791007200837703339,
 794301459533783434896055,
 688178021083749589017751,
 742362470968200823035396,
 1023345827831539515054795,
 -357168679398558876730006,
 1121421619119964601051443,
 -1109674862276222495587129,
 -235628937785003770523381)
sage: V[1]
(0, -587473338058640662659869,
 0, 0, 0, 0, 0, 0, 0, 0)
sage:
```

```
sage: V.LLL()[0]
(610803584000, 1
 37030242384, 84
 -225618319442,
 1100126026284,
 1359463649048,
sage: q=[Ki//s f
sage: q[0]*E
610803584000
sage: q[0]*K[1]-
1056189937254
sage: q[0]*K[9]-
174256676348
sage:
```

])


```
sage: V[0]
(1024,
 -1111539179100720083770339,
 794301459533783434896055,
 68817802108374958901751,
 742362470968200823035396,
 1023345827831539515054795,
 -357168679398558876730006,
 1121421619119964601051443,
 -1109674862276222495587129,
 -235628937785003770523381)
sage: V[1]
(0, -587473338058640662659869,
 0, 0, 0, 0, 0, 0, 0, 0)
sage:
```



```
sage: V.LLL()[0]
(610803584000, 1056189937
 37030242384, 84589845469
 -225618319442, 363547143
 1100126026284, -31315097
 1359463649048, 174256676
sage: q=[Ki//s for Ki in
sage: q[0]*E
610803584000
sage: q[0]*K[1]-q[1]*K[0]
1056189937254
sage: q[0]*K[9]-q[9]*K[0]
174256676348
sage:
```

```
sage: V[0]
(1024,
 -1111539179100720083770339,
 794301459533783434896055,
 688178021083749589017751,
 742362470968200823035396,
 1023345827831539515054795,
 -357168679398558876730006,
 1121421619119964601051443,
 -1109674862276222495587129,
 -2356289377850037705233881)
sage: V[1]
(0, -5874733380586406626598869,
 0, 0, 0, 0, 0, 0, 0, 0)
sage:
```



```
sage: V.LLL()[0]
(610803584000, 1056189937254,
 37030242384, 845898454698,
 -225618319442, 363547143644,
 1100126026284, -313150978512,
 1359463649048, 174256676348)
sage: q=[Ki//s for Ki in K]
sage: q[0]*E
610803584000
sage: q[0]*K[1]-q[1]*K[0]
1056189937254
sage: q[0]*K[9]-q[9]*K[0]
174256676348
sage:
```

```
[0]

39179100720083770339,
459533783434896055,
02108374958901751,
470968200823035396,
5827831539515054795,
8679398558876730006,
1619119964601051443,
74862276222495587129,
893778500370523381)
[1]
7473338058640662659869,
0, 0, 0, 0, 0, 0)
```



```
sage: V.LLL()[0]
(610803584000, 1056189937254,
 37030242384, 845898454698,
 -225618319442, 363547143644,
 1100126026284, -313150978512,
 1359463649048, 174256676348)
sage: q=[Ki//s for Ki in K]
sage: q[0]*E
610803584000
sage: q[0]*K[1]-q[1]*K[0]
1056189937254
sage: q[0]*K[9]-q[9]*K[0]
174256676348
sage:
```

2009 DG
can choo
these lat

2011 Co
Tibouch
by modi
shows th
encrypti
with a si

e.g. all a
with pub

2012 Ch
Need big

```
20083770339,
434896055,
58901751,
823035396,
9515054795,
8876730006,
4601051443,
22495587129,
3770523381)

8640662659869,
0, 0, 0)
```



```
sage: V.LLL()[0]
(610803584000, 1056189937254,
 37030242384, 845898454698,
 -225618319442, 363547143644,
 1100126026284, -313150978512,
 1359463649048, 174256676348)
sage: q=[Ki//s for Ki in K]
sage: q[0]*E
610803584000
sage: q[0]*K[1]-q[1]*K[0]
1056189937254
sage: q[0]*K[9]-q[9]*K[0]
174256676348
sage:
```

2009 DGHV analy
can choose key siz
these lattice attac

2011 Coron–Mand
Tibouchi: reduce
by modifying DGH
shows that fully ho
encryption can be
with a simple sche

e.g. all attacks tak
with public keys o

2012 Chen–Nguye
Need bigger DGH

```
339,
5,
3,
05,
06,
13,
129,
31)
59869,
```

```
sage: V.LLL()[0]
(610803584000, 1056189937254,
 37030242384, 845898454698,
 -225618319442, 363547143644,
 1100126026284, -313150978512,
 1359463649048, 174256676348)
sage: q=[Ki//s for Ki in K]
sage: q[0]*E
610803584000
sage: q[0]*K[1]-q[1]*K[0]
1056189937254
sage: q[0]*K[9]-q[9]*K[0]
174256676348
sage:
```


2009 DGHV analysis:
can choose key sizes where
these lattice attacks fail.

2011 Coron–Mandal–Naccac
Tibouchi: reduce key sizes
by modifying DGHV. "This
shows that fully homomorph
encryption can be implemen
with a simple scheme."

e.g. all attacks take $\geq 2^{72}$ cy
with public keys only 802ME

2012 Chen–Nguyen: faster a
Need bigger DGHV/CMNT

```
sage: V.LLL()[0]
(610803584000, 1056189937254,
 37030242384, 845898454698,
 -225618319442, 363547143644,
 1100126026284, -313150978512,
 1359463649048, 174256676348)
sage: q=[Ki//s for Ki in K]
sage: q[0]*E
610803584000
sage: q[0]*K[1]-q[1]*K[0]
1056189937254
sage: q[0]*K[9]-q[9]*K[0]
174256676348
sage:
```


2009 DGHV analysis: can choose key sizes where these lattice attacks fail.

2011 Coron–Mandal–Naccache–Tibouchi: reduce key sizes by modifying DGHV. "This shows that fully homomorphic encryption can be implemented with a simple scheme."

e.g. all attacks take $\geq 2^{72}$ cycles with public keys only 802MB.

2012 Chen–Nguyen: faster attack. Need bigger DGHV/CMNT keys.

```
.LLL()[0]
584000, 1056189937254,
42384, 845898454698,
3319442, 363547143644,
6026284, -313150978512,
3649048, 174256676348)
=[Ki//s for Ki in K]
[0]*E
84000
[0]*K[1]-q[1]*K[0]
937254
[0]*K[9]-q[9]*K[0]
76348
```


2009 DGHV analysis: can choose key sizes where these lattice attacks fail.

2011 Coron–Mandal–Naccache–Tibouchi: reduce key sizes by modifying DGHV. "This shows that fully homomorphic encryption can be implemented with a simple scheme."

e.g. all attacks take $\geq 2^{72}$ cycles with public keys only 802MB.

2012 Chen–Nguyen: faster attack. Need bigger DGHV/CMNT keys.

Big atta

1991 Ch
Pfitzma
define C
for suita

Simple,
Very eas
finding C
computi

Typical
C is "pr
"cryptog
"security
mathem

```
056189937254,
5898454698,
363547143644,
-313150978512,
174256676348)
or Ki in K]

q[1]*K[0]

q[9]*K[0]
```


2009 DGHV analysis: can choose key sizes where these lattice attacks fail.

2011 Coron–Mandal–Naccache–Tibouchi: reduce key sizes by modifying DGHV. "This shows that fully homomorphic encryption can be implemented with a simple scheme."

e.g. all attacks take $\geq 2^{72}$ cycles with public keys only 802MB.

2012 Chen–Nguyen: faster attack. Need bigger DGHV/CMNT keys.

Big attack surface

1991 Chaum–van

Pfitzmann: choose

define $C(x, y) = 4$

for suitable ranges

Simple, beautiful,

Very easy security

finding C collision

computing a discr

Typical exaggerati

C is "provably sec

"cryptographically

"security follows fr

mathematical proc

/254,
98,
3644,
/8512,
3348)
K]

2009 DGHV analysis: can choose key sizes where these lattice attacks fail.

2011 Coron–Mandal–Naccache–Tibouchi: reduce key sizes by modifying DGHV. "This shows that fully homomorphic encryption can be implemented with a simple scheme."

e.g. all attacks take $\geq 2^{72}$ cycles with public keys only 802MB.

2012 Chen–Nguyen: faster attack. Need bigger DGHV/CMNT keys.

Big attack surfaces are dang

1991 Chaum–van Heijst–Pfitzmann: choose p sensibl define $C(x, y) = 4^x 9^y \bmod$ for suitable ranges of x and

Simple, beautiful, structured Very easy security reduction finding C collision implies computing a discrete logarit

Typical exaggerations: C is "provably secure"; C is "cryptographically collision-f "security follows from rigoro mathematical proofs".

2009 DGHV analysis:
can choose key sizes where
these lattice attacks fail.

2011 Coron–Mandal–Naccache–Tibouchi: reduce key sizes by modifying DGHV. "This shows that fully homomorphic encryption can be implemented with a simple scheme."

e.g. all attacks take $\geq 2^{72}$ cycles with public keys only 802MB.

2012 Chen–Nguyen: faster attack. Need bigger DGHV/CMNT keys.

Big attack surfaces are dangerous

1991 Chaum–van Heijst–Pfitzmann: choose p sensibly; define $C(x, y) = 4^x 9^y \bmod p$ for suitable ranges of x and y.

Simple, beautiful, structured. Very easy security reduction: finding C collision implies computing a discrete logarithm.

Typical exaggerations:
C is "provably secure"; C is "cryptographically collision-free"; "security follows from rigorous mathematical proofs".

GHV analysis:
ose key sizes where
ttice attacks fail.

oron–Mandal–Naccache–
i: reduce key sizes
fying DGHV. "This
hat fully homomorphic
on can be implemented
imple scheme."

attacks take $\geq 2^{72}$ cycles
blic keys only 802MB.

en–Nguyen: faster attack.
gger DGHV/CMNT keys.

Big attack surfaces are dangerous

1991 Chaum–van Heijst–Pfitzmann: choose p sensibly; define $C(x, y) = 4^x 9^y \bmod p$ for suitable ranges of x and y.

Simple, beautiful, structured. Very easy security reduction: finding C collision implies computing a discrete logarithm.

Typical exaggerations:
C is "provably secure"; C is "cryptographically collision-free"; "security follows from rigorous mathematical proofs".

Security
1922 Kr
1986 Co
Schroep
1993 Go
1993 Sc
1994 Sh
many su
from pe
pre-quan

C is very
No matt
is, obtai
"unstruc
function

sis:
zes where
ks fail.

lal–Naccache–
key sizes
HV. “This
omomorphic
implemented
eme.”

ke $\geq 2^{72}$ cycles
nly 802MB.

n: faster attack.
V/CMNT keys.

Big attack surfaces are dangerous

1991 Chaum–van Heijst–Pfitzmann: choose p sensibly; define $C(x, y) = 4^x 9^y \bmod p$ for suitable ranges of x and y.

Simple, beautiful, structured. Very easy security reduction: finding C collision implies computing a discrete logarithm.

Typical exaggerations:
C is “provably secure”; C is “cryptographically collision-free”; “security follows from rigorous mathematical proofs”.

Security losses in C
1922 Kraitchik (in
1986 Coppersmith
Schroeppel (NFS p
1993 Gordon (gene
1993 Schirokauer
1994 Shor (quantu
many subsequent
from people who c
pre-quantum secur

C is very bad cryp
No matter what u
is, obtain better se
“unstructured” co
function designs su

che–

ic
ted

ycles
3.

ttack.
keys.

Big attack surfaces are dangerous

1991 Chaum–van Heijst–Pfitzmann: choose p sensibly; define $C(x, y) = 4^x 9^y \bmod p$ for suitable ranges of x and y.

Simple, beautiful, structured. Very easy security reduction: finding C collision implies computing a discrete logarithm.

Typical exaggerations: C is "provably secure"; C is "cryptographically collision-free"; "security follows from rigorous mathematical proofs".

Security losses in C include
1922 Kraitchik (index calcul
1986 Coppersmith–Odlyzko–
Schroeppel (NFS predecesso
1993 Gordon (general DL N
1993 Schirokauer (faster NF
1994 Shor (quantum poly ti
many subsequent attack spe
from people who care about
pre-quantum security.

C is very bad cryptography.
No matter what user's cost
is, obtain better security wit
"unstructured" compression-
function designs such as BL

Big attack surfaces are dangerous

1991 Chaum–van Heijst–Pfitzmann: choose p sensibly; define $C(x, y) = 4^x 9^y \bmod p$ for suitable ranges of x and y.

Simple, beautiful, structured. Very easy security reduction: finding C collision implies computing a discrete logarithm.

Typical exaggerations: C is "provably secure"; C is "cryptographically collision-free"; "security follows from rigorous mathematical proofs".

Security losses in C include
1922 Kraitchik (index calculus);
1986 Coppersmith–Odlyzko–Schroeppel (NFS predecessor);
1993 Gordon (general DL NFS);
1993 Schirokauer (faster NFS);
1994 Shor (quantum poly time);
many subsequent attack speedups from people who care about pre-quantum security.

C is very bad cryptography. No matter what user's cost limit is, obtain better security with "unstructured" compression-function designs such as BLAKE.

ck surfaces are dangerous

aum–van Heijst–
nn: choose p sensibly;
$(x, y) = 4^x 9^y \bmod p$
ble ranges of x and y.

beautiful, structured.
y security reduction:
C collision implies
ng a discrete logarithm.

exaggerations:
ovably secure"; C is
graphically collision-free";
follows from rigorous
atical proofs".

Security losses in C include
1922 Kraitchik (index calculus);
1986 Coppersmith–Odlyzko–
Schroeppel (NFS predecessor);
1993 Gordon (general DL NFS);
1993 Schirokauer (faster NFS);
1994 Shor (quantum poly time);
many subsequent attack speedups
from people who care about
pre-quantum security.

C is very bad cryptography.
No matter what user's cost limit
is, obtain better security with
"unstructured" compression-
function designs such as BLAKE.

For publ
Some m
seems to
but purs
often lea

Pre-quan
simpler t
suffered
than EC
attacks a

2013 Ba
Thomé:
break of

s are dangerous

Heijst–
e p sensibly;
$^x g^y$ mod p
of x and y.

structured.
reduction:
implies
ete logarithm.

ons:
ure"; C is
collision-free";
rom rigorous
ofs".

Security losses in C include
1922 Kraitchik (index calculus);
1986 Coppersmith–Odlyzko–Schroeppel (NFS predecessor);
1993 Gordon (general DL NFS);
1993 Schirokauer (faster NFS);
1994 Shor (quantum poly time);
many subsequent attack speedups from people who care about pre-quantum security.

C is very bad cryptography. No matter what user's cost limit is, obtain better security with "unstructured" compression-function designs such as BLAKE.

For public-key enc
Some mathematic
seems to be unavo
but pursuing simp
often leads to secu

Pre-quantum exam
simpler than ECD
suffered many mor
than ECDH. State
attacks are very c

2013 Barbulescu–
Thomé: pre-quant
break of small-cha

gerous

ly;
p
y.
l.
:

hm.

free";
ous

Security losses in C include
1922 Kraitchik (index calculus);
1986 Coppersmith–Odlyzko–Schroeppel (NFS predecessor);
1993 Gordon (general DL NFS);
1993 Schirokauer (faster NFS);
1994 Shor (quantum poly time);
many subsequent attack speedups from people who care about pre-quantum security.

C is very bad cryptography. No matter what user's cost limit is, obtain better security with "unstructured" compression-function designs such as BLAKE.

For public-key encryption:
Some mathematical structur
seems to be unavoidable,
but pursuing simple structur
often leads to security disast

Pre-quantum example: DH
simpler than ECDH, but DH
suffered many more security
than ECDH. State-of-the-ar
attacks are very complicatec

2013 Barbulescu–Gaudry–Jo
Thomé: pre-quantum quasi-
break of small-characteristic

Security losses in C include
1922 Kraitchik (index calculus);
1986 Coppersmith–Odlyzko–Schroeppel (NFS predecessor);
1993 Gordon (general DL NFS);
1993 Schirokauer (faster NFS);
1994 Shor (quantum poly time);
many subsequent attack speedups from people who care about pre-quantum security.

C is very bad cryptography. No matter what user's cost limit is, obtain better security with "unstructured" compression-function designs such as BLAKE.

For public-key encryption: Some mathematical structure seems to be unavoidable, but pursuing simple structures often leads to security disasters.

Pre-quantum example: DH is simpler than ECDH, but DH has suffered many more security losses than ECDH. State-of-the-art DH attacks are very complicated.

2013 Barbulescu–Gaudry–Joux–Thomé: pre-quantum quasi-poly break of small-characteristic DH.

losses in C include
aitchik (index calculus);
ppersmith–Odlyzko–
pel (NFS predecessor);
rdon (general DL NFS);
hirokauer (faster NFS);
or (quantum poly time);
bsequent attack speedups
ople who care about
ntum security.

y bad cryptography.
ter what user's cost limit
n better security with
ctured" compression-
designs such as BLAKE.

For public-key encryption:
Some mathematical structure
seems to be unavoidable,
but pursuing simple structures
often leads to security disasters.

Pre-quantum example: DH is
simpler than ECDH, but DH has
suffered many more security losses
than ECDH. State-of-the-art DH
attacks are very complicated.

2013 Barbulescu–Gaudry–Joux–
Thomé: pre-quantum quasi-poly
break of small-characteristic DH.

The stat
against
are muc
than the

Lattice-b
advertise
simple",
"linear c
Attacks

For effic
cryptosy
features
surface e
rings and

C include
dex calculus);
–Odlyzko–
predecessor);
eral DL NFS);
(faster NFS);
um poly time);
attack speedups
care about
rity.

tography.
ser's cost limit
ecurity with
mpression-
uch as BLAKE.

For public-key encryption:
Some mathematical structure seems to be unavoidable, but pursuing simple structures often leads to security disasters.

Pre-quantum example: DH is simpler than ECDH, but DH has suffered many more security losses than ECDH. State-of-the-art DH attacks are very complicated.

2013 Barbulescu–Gaudry–Joux–Thomé: pre-quantum quasi-poly break of small-characteristic DH.

The state-of-the-a
against Cohen's cr
are much more co
than the cryptosys

Lattice-based cryp
advertised as "alg
simple", consisting
"linear operations
Attacks exploit thi

For efficiency, latti
cryptosystems usu
features that expa
surface even more
rings and decrypti

us);
-
or);
FS);
S);
me);
eedups

limit
h
-
AKE.

For public-key encryption:
Some mathematical structure seems to be unavoidable, but pursuing simple structures often leads to security disasters.

Pre-quantum example: DH is simpler than ECDH, but DH has suffered many more security losses than ECDH. State-of-the-art DH attacks are very complicated.

2013 Barbulescu–Gaudry–Joux–Thomé: pre-quantum quasi-poly break of small-characteristic DH.

The state-of-the-art attacks
against Cohen's cryptosyster
are much more complicated
than the cryptosystem is. S

Lattice-based cryptosystems
advertised as "algorithmicall
simple", consisting mainly o
"linear operations on vector
Attacks exploit this structur

For efficiency, lattice-based
cryptosystems usually have
features that expand the att
surface even more: e.g.,
rings and decryption failures

For public-key encryption:
Some mathematical structure seems to be unavoidable, but pursuing simple structures often leads to security disasters.

Pre-quantum example: DH is simpler than ECDH, but DH has suffered many more security losses than ECDH. State-of-the-art DH attacks are very complicated.

2013 Barbulescu–Gaudry–Joux–Thomé: pre-quantum quasi-poly break of small-characteristic DH.

The state-of-the-art attacks against Cohen's cryptosystem are much more complicated than the cryptosystem is. Scary!

Lattice-based cryptosystems are advertised as "algorithmically simple", consisting mainly of "linear operations on vectors". Attacks exploit this structure!

For efficiency, lattice-based cryptosystems usually have features that expand the attack surface even more: e.g., rings and decryption failures.