Lattice-based cryptography, day 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 $C = (-1)^m(r_1K_1 + \cdots + r_NK_N)$.

based cryptography,
implicity

ernstein

ty of Illinois at Chicago;
niversity 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

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

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?

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?

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

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 $C \bmod s \le (s-1)/2$;
otherwise $m = 1$.

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 $C \bmod s \le (s-1)/2$;
otherwise $m = 1$.

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

2000 Cohen cryptosystem

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

Encryption:

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

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

(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
$C = (-1)^m (r_1 K_1 + \dots + r_N K_N)$.

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

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

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

oute 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: f
3172135
sage:
```

osystem

of integers

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

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

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

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

“half of the
blic key at
this means
$= N/2$.)

end ciphertext
$+ \cdots + r_N K_N)$.

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

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

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

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

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

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

Let's try this on the comput

Debian: `apt install sage`

Fedora: `dnf install sager`

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(314159265358
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

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

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

Let's try this on the 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:
```

n receiver decrypt?

eration:

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

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

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

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(31415926535897932 3)
317213509 * 990371647
sage:
```

For integ

Sage's “

outputs

Matches

C mod s

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

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

$\rfloor \rfloor \}$;

$X\}$.

$/2$;

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(31415926535879323)
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

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

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:
```


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

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

outputs between 0 and `s` $- 1$.

Matches standard math definition:

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

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.

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` 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.

this on the computer.

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

e `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:
```

ne computer.

```
all sagemath
all sagemath
```

emath.org

X) to see X):

ath.org

aries

fferences:

```
er, not xor

159265358979323)
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:
```

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:
```

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

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

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

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


```
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.


```
sage: N=10
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.


```
sage: N=10
sage: X=2^50
sage:
```

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

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

Warning: Typically
$\mathtt{C} < 0$ produces $\mathtt{C\%s} < 0$
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:
```

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:
```

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

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

Warning: Typically $\mathtt{C} < 0$ produces $\mathtt{C\%s} < 0$ 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:
```

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:
```

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:
```

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

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

: 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:
```

vith s > 0,
ys produces
) and s $- 1$.

math definition:
$C/s\rfloor s$.

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

nomials 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:
```

,
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:
```

```
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:
```

```
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,
 461093902438341]
```

6

```
=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]
```

7

```
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: [
[14485,
 10493,
 8213, 
sage: u
[14485,
 10493,
 8213, 
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,
 46109390243834]
```

```
sage: [Ki%s for
[14485, 7039, 69
 10493, 17333, 1
 8213, 6370]
sage: u
[14485, 7039, 69
 10493, 17333, 1
 8213, 6370]
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:
```

```
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:
```

```
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:
```

```
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:
```

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

918917829,
576592695,
554434581 5,
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:
```

```
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:
```

),

)+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:
```

```
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:
```

```
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:
```

```
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:
```

```
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:
```

```
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:
```

```
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:
```

```
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
-2022158560435 76
sage: C%s
47024
sage: m
0
sage: sum(r[i]*u[i]
....:      for i in range(N))
47024
sage:
```

8

```
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
```

9

```
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

```
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

```
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.

```
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.

```
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.

```
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
-20221585604357б
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

```
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

```
)]
[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

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.

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. Dera
reencryp

This is a
1999 Fu

th cryptosystem

roblem:
ypt messages
an 1 bit.

m:
stems to resist
: 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 example
1999 Fujisaki–Oka

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

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.

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?)

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
$_1, \dots, m_B) \in \{0, 1\}^B$.
$i \in \{1, \dots, B\}$:
e $r_{i,1}, \dots, r_{i,N} \in \{0, 1\}$.

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

Subset-s

Attacker
for $(r_1, .$
checks r
against

This tak
e.g. 102

tosystem
se problems.

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

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

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

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

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 $r_1 K_1 + \cdots$
against $\pm C_1$.

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

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 $r_1K_1 + \cdots + r_NK_N$ against $\pm C_1$.

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

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 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 $N = 10$.

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."

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.

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
ot 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.

Modifiec
For each
$r_1K_1 + \cdot$
containi

ncryption, and
lecryption.

e of “FO”, the
moto transform.

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

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

Subset-sum attacks

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

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 $\pm C_1, \pm C_2, \ldots, \pm$

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

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 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.

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 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.

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 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.

sum attacks

r searches all possibilities

$\ldots, r_N),$

$_1 K_1 + \cdots + r_N K_N$

$\pm C_1.$

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 $r_1 K_1 + \cdots + r_N K_N$ 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

$N = 128$

day, and

transform

ks

all possibilities

$+ r_N K_N$

y operations:

ns for $N = 10$.

ne bit m_1.”

em in some

uld design

no information.

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

lities

ns:

= 10.

”

e

nation.

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.”

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."

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 $r_1 K_1 + \cdots + r_N K_N = C$.

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 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 $r_1K_1 + \cdots + r_NK_N = C$.

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

Look up $r_1K_1 + \cdots + r_{N/2}K_{N/2}$ in hash table for each $(r_1, \ldots, r_{N/2})$.

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

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 $(r_1, \dots, r_N) \in \{0,1\}^N$ 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 $(r_{N/2+1}, \dots, r_N)$.

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

These at
structure
one targ

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

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

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

"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.

"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 $r_1K_1 + \cdots + r_NK_N = C$.

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

Look up $r_1K_1 + \cdots + r_{N/2}K_{N/2}$ 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.)

"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

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

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

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

$\cdot + r_{N/2} K_{N/2}$ 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

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: $2^{0.3}$

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: $2^{0.337N}$.

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.

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.

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.

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: $2^{0.283N}$.

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.

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
ore 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
$(-1)^m($
$m(K_1/2$

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 cryptos

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

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
$(-1)^m(r_1K_1 + \cdots + r_NK_N)$
$m(K_1/2) + r_1K_1 + \cdots + r_N$

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 $m(K_1/2) + r_1K_1 + \cdots + r_NK_N$.

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.

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

raham–Joux:
perations. 2011
ction: $2^{0.337N}$.

n–Joux:
.

^{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

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 homomorph

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.

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 $\epsilon, \epsilon' \in \mathbf{Z}$.

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.

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
$) + r_1 K_1 + \cdots + r_N K_N$.

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:
```

system

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

x,
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:
```

ystem
e
with
K_N.

$\in 2\mathbf{Z}$

nds.

vi–
$+ s\mathbf{Z}$;
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:
```

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:
```

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:
```

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:
```

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:
```

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:
```

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:
```

ryption

ough then 2009
omomorphic.

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:
```

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:
```

```
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:
```

```
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:
```

```
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,
 -3571686793985588767300006,
 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:
```

```
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,
 -111153917910072008377033 9,
 794301459533783434896055,
 688178021083749589017 51,
 742362470968200823035396,
 1023345827831539515054795,
 -3571686793985588767300 06,
 1121421619119964601051443,
 -1109674862276222495587129,
 -2356289377850037705233 81]
```

```
sage: m=randrang
sage: r=[randran
....:     for i i
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
[58747333805864066265986 9,
 -11115391791007200837703 39,
 79430145953378343489605 5,
 68817802108374958901751,
 74236247096820082303539 6,
 10233458278315395150547 95,
 -35716867939855887673000 6,
 11214216191199646010514 43,
 -11096748622762224955871 29,
 -23562893778500377052338 1]
```

```
sage: m=randrange(2)
sage: r=[randrange(2)
....:     for i in range(N
sage:
```

```
sage: K=[2*ui+s*randrange(
....:      ceil(-(X+2*ui)/s),
....:      floor((X-2*ui)/s)+1)
....:    for ui in u]
sage: K
[5874733380586406626 59869,
 -11115391791007200837 70339,
 79430145953378343489 6055,
 68817802108374958901 751,
 74236247096820082303 5396,
 10233458278315395150 54795,
 -35716867939855887673 0006,
 11214216191199646010 51443,
 -11096748622762224955 87129,
 -23562893778500377052 3381]
```



```
sage: m=randrange(2)
sage: r=[randrange(2)
....:    for i in range(N)]
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:
```

```
sage: K=[2*ui+s*randrange(
....:      ceil(-(X+2*ui)/s),
....:      floor((X-2*ui)/s)+1)
....:    for ui in u]
sage: K
[58747338058640662659869,
 -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:
```

```
sage: K=[2*ui+s*randrange(
....:        ceil(-(X+2*ui)/s),
....:        floor((X-2*ui)/s)+1)
....:     for ui in u]
sage: K
[58747338058640662659869,
 -1111539179100720083770339,
 79430145953783434896055,
 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:
```

```
sage: K=[2*ui+s*randrange(
....:      ceil(-(X+2*ui)/s),
....:      floor((X-2*ui)/s)+1)
....:    for ui in u]
sage: K
[58747338058640662659869,
 -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]

338058640662659869,
39179100720083770339,
459533783434896055,
02108374958901751,
470968200823035396,
5827831539515054795,
8679398558876730006,
1619119964601051443,
74862276222495587129,
893778500370523381]
```



```
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:
```

```
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
2094088748748247210016703
sage: C%s
2703
sage: (C%s)%2
1
sage: m
1
sage:
```

```
sage: m2=randran
sage: r2=[randra
....:     for i
sage:
```

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

),
39,
5,

3,
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:
```

```
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: m2=randrange(2)
sage: r2=[randrange(2)
....:     for i in range(N)]
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
2094088748748247210016703
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:
```

```
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(N)]
sage: C2=m2+sum(r2[i]*K[i]
....:      for i in range(N))
sage: C2
-5172235373798273727O129
sage: C2%s
4971
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
2094088748748247210016703
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
-5172253737982737270129
sage: C2%s
4971
sage: (C2%s)%2
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
2094088748748247210016703
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
-5172235373798273727O129
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: (
7674
sage: (
1343661
sage:
```

```
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:
```

```
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:
```

```
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:
```

```
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 $s = (C \bmod s) + (C' \bmod s)$ and CC' mod $s = (C \bmod s)(C' \bmod s)$.

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

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
537379827372701 29
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

```
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

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



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

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

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

Lattices

```
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 $s = (C \bmod s) + (C' \bmod s)$ and CC' mod $s = (C \bmod s)(C' \bmod s)$.

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

Lattices

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

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

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

Lattices

This is a lettuce:

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

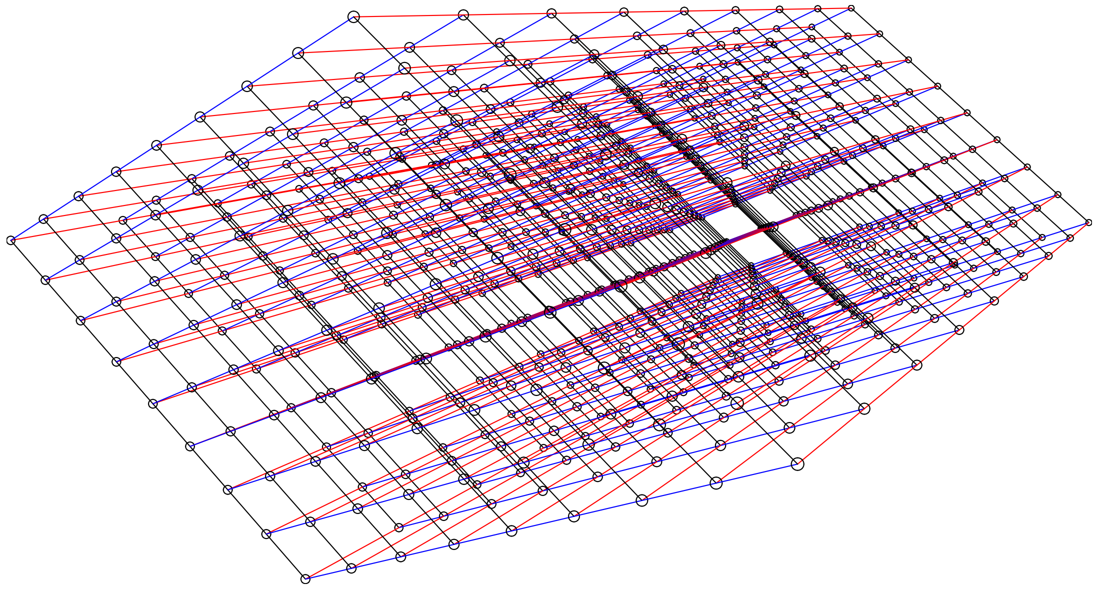
Because $C \bmod s$ and $C' \bmod s$ are small enough compared to s, have $C + C' \bmod s = (C \bmod s) + (C' \bmod s)$ and $CC' \bmod s = (C \bmod s)(C' \bmod s)$.

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
```

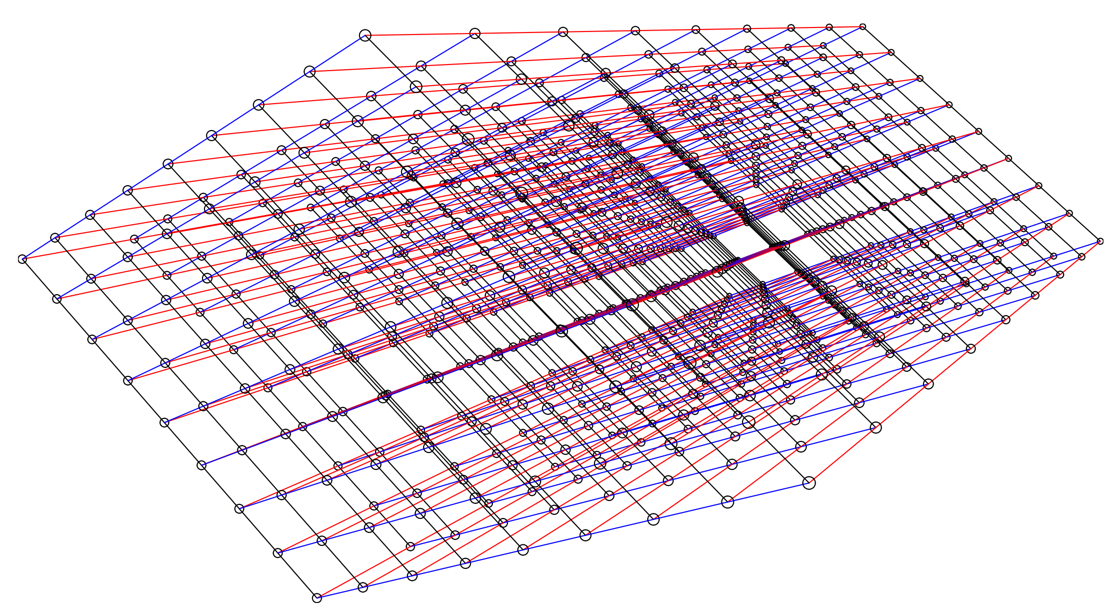
C mod s and C' mod s
l enough compared to s,
$+ C'$ mod $s = (C$ mod $s) +$
$s)$ and CC' mod s $=$
$s)(C'$ mod $s)$.

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

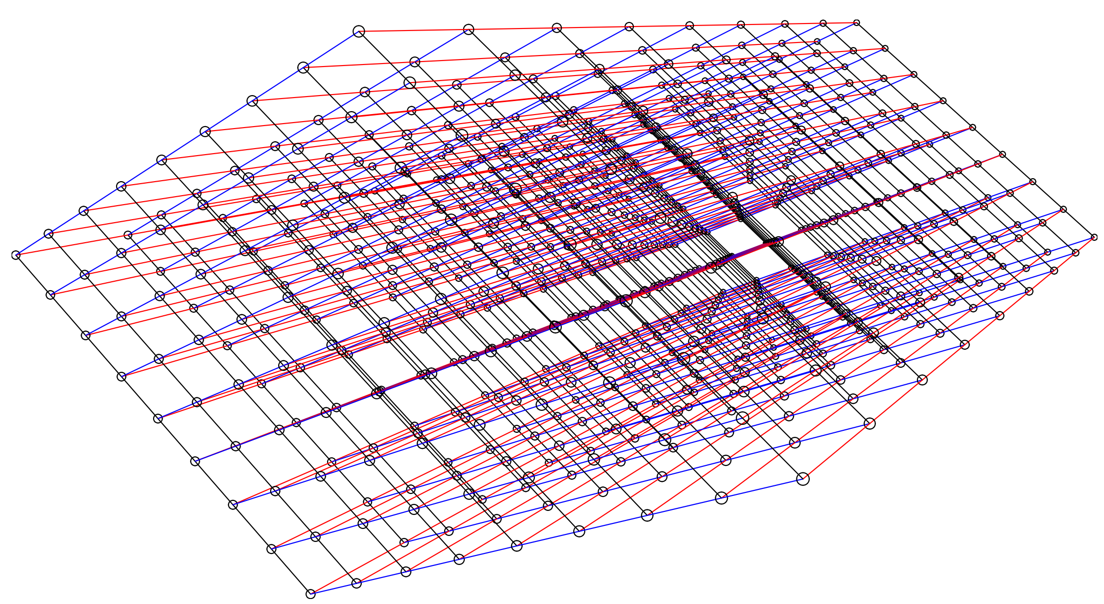
and C' mod s
compared to s,
$s = (C \bmod s) +$
$C' \bmod s =$
d s).

more noise
otstrap (2009
noise, etc.

Lattices

This is a lettuce:

This is a lattice:



Lattices, mathema

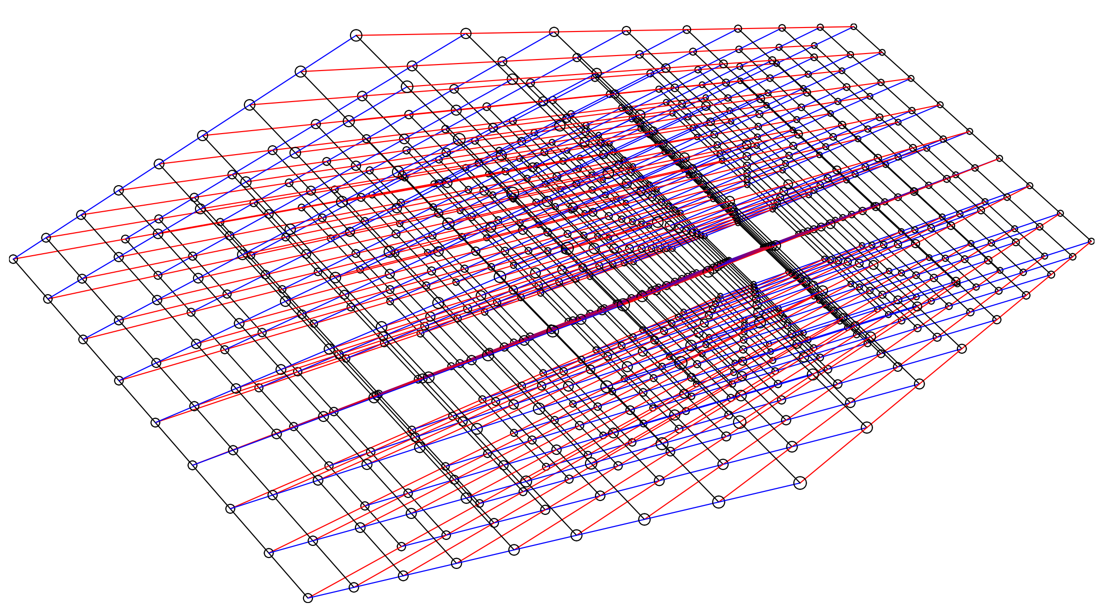
Assume that $V_1, \ldots$
are $\mathbf{R}$-linearly inde
i.e., $\mathbf{R}V_1 + \cdots + \mathbf{R}$
$\{r_1V_1 + \cdots + r_DV_D$
is a D-dimensiona

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

Lattices

This is a lettuce:

This is a lattice:



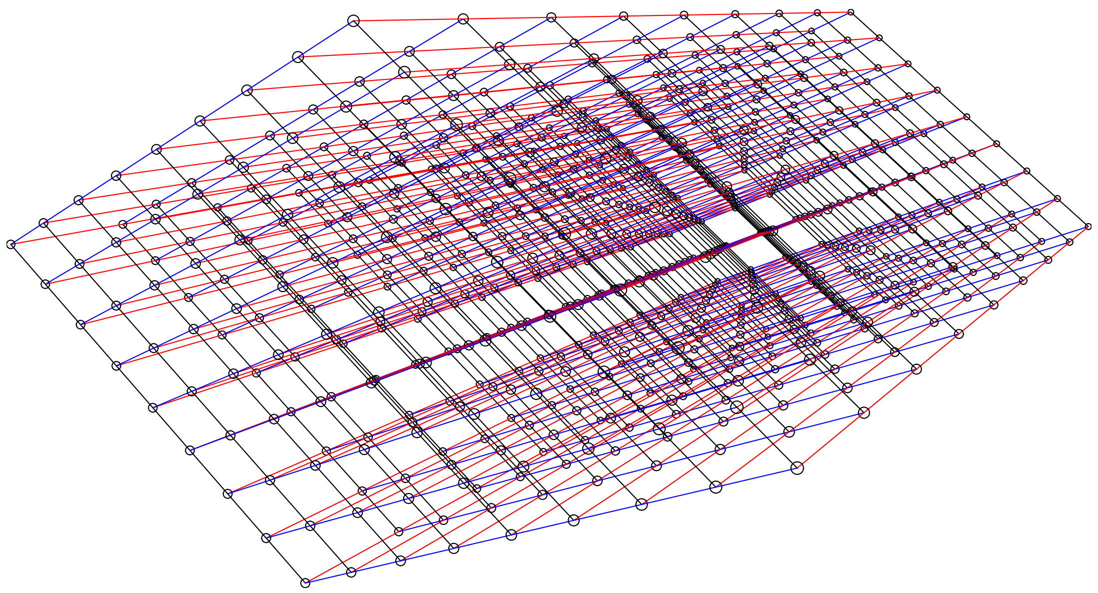
Lattices, mathematically

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

Lattices

This is a lettuce:

This is a lattice:



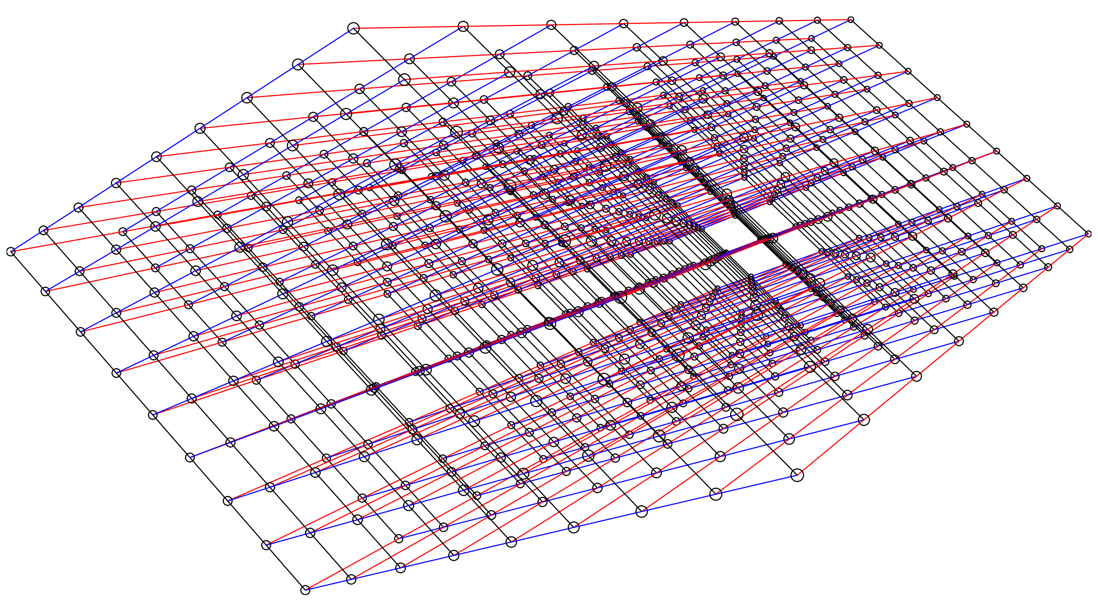
Lattices, mathematically

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.

Lattices

This is a lettuce:

This is a lattice:



Lattices, mathematically

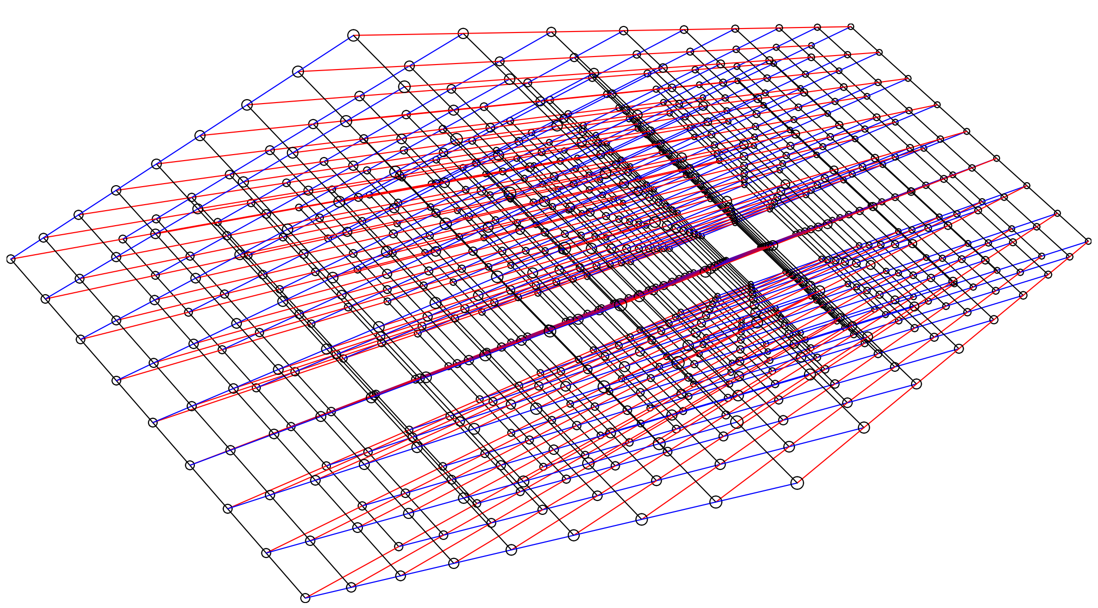
Assume that $V_1, \ldots, V_D \in \mathbf{R}^N$ are $\mathbf{R}$-linearly independent, i.e., $\mathbf{R}V_1 + \cdots + \mathbf{R}V_D = \{r_1V_1 + \cdots + r_DV_D : r_1, \ldots, r_D \in \mathbf{R}\}$ is a D-dimensional vector space.

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

Lattices

This is a lettuce:

This is a lattice:



Lattices, mathematically

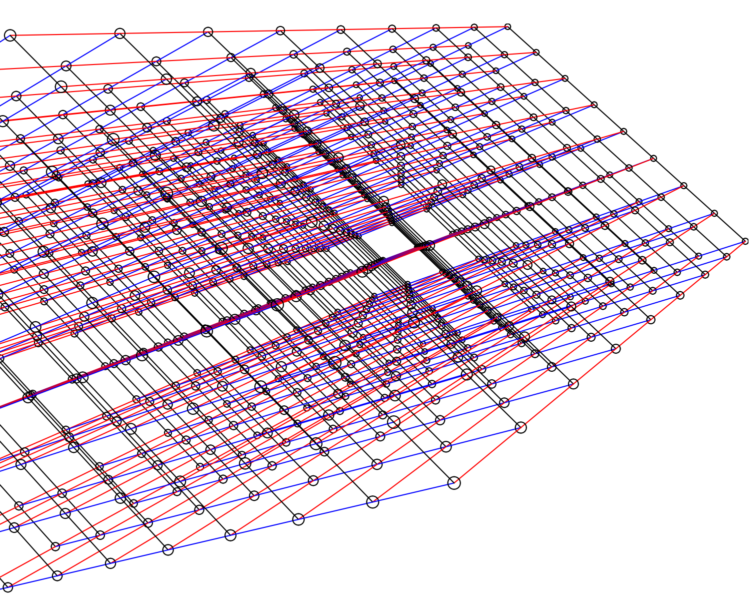
Assume that $V_1, \ldots, V_D \in \mathbf{R}^N$
are $\mathbf{R}$-linearly independent,
i.e., $\mathbf{R}V_1 + \cdots + \mathbf{R}V_D =$
$\{r_1V_1 + \cdots + r_DV_D : r_1, \ldots, r_D \in \mathbf{R}\}$
is a D-dimensional vector space.

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

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

lettuce:

lattice:



Lattices, mathematically

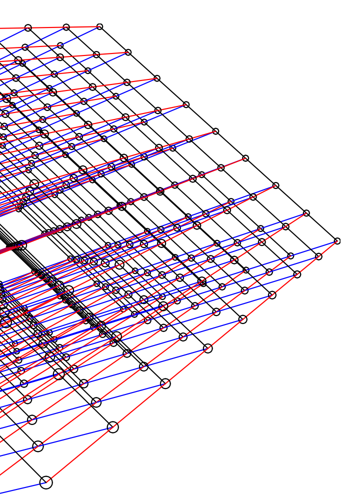
Assume that $V_1, \ldots, V_D \in \mathbf{R}^N$
are $\mathbf{R}$-linearly independent,
i.e., $\mathbf{R}V_1 + \cdots + \mathbf{R}V_D =$
$\{r_1V_1 + \cdots + r_DV_D : r_1, \ldots, r_D \in \mathbf{R}\}$
is a D-dimensional vector space.

$\mathbf{Z}V_1 + \cdots + \mathbf{Z}V_D =$
$\{r_1V_1 + \cdots + r_DV_D : r_1, \ldots, r_D \in \mathbf{Z}\}$
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 $L = \mathbf{Z}$



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

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 $L = \mathbf{Z}V_1 + \cdots + \mathbf{Z}V_D$?

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 $L = \mathbf{Z}V_1 + \cdots + \mathbf{Z}V_D$?

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.

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 problem":
What is shortest nonzero vector?

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

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 $C = r_1 K_1 +$

^{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 $C = r_1K_1 + \cdots + r_N$

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

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

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

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

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
$V_0 + r_1 V_1 + \cdots + r_N V_N =$
$(0, r_1\lambda, \ldots, r_N\lambda)$.

ectors in lattices

$, V_2, \ldots, V_D \in \mathbf{Z}^N$,
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.
y $\approx 1.02^D$ instead of $2^{D/2}$.

Subset-sum lattices

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

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 fa
finds thi

ttices

$V_D \in \mathbf{Z}^N$,
ector
$+ \mathbf{Z}V_D$?

ctor problem":
nonzero 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 $C = r_1 K_1 + \cdots + r_N K_N$:

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 alr
finds this short vec

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.

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.

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.

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.

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

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

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

λ. Define

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

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

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

${}_N, 0, 0, \ldots, \lambda)$.

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

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

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 $q_j K_i - q_i K_j$

)

K_N:

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 DGHV ke

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

Each u_i is small: $u_i < E$.

Note $q_j K_i - q_i K_j = 2q_j u_i -$

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 $q_j K_i - q_i K_j = 2q_j u_i - 2q_i u_j$.

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

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:
```

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:
```

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:
```

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:
```

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:
```

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:
```

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:
```

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:
```

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:
```

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:
```

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 $q_1V_1 + \cdots + q_NV_N =$

$K_2 - q_2K_1, \ldots) =$

$q_1u_2 - 2q_2u_1, \ldots)$.


```
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:
```

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
 688178021083749
 74236247096820O
 10233458278315З
 -35716867939855
 11214216191199б
 -11096748622762
 -23562893778500
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
 79430145953378343489605
 6881780210837495890175l,
 74236247096820082303539б
 10233458278315395150547
 -35716867939855887673000
 11214216191199646010514
 -11096748622762224955871
 -23562893778500377052338
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,
 688178021083749589017 51,
 742362470968200823035396,
 1023345827831539515054795,
 -357168679398558876730006,
 1121421619119964601051443,
 -1109674862276222495587129,
 -235628937785003770523381)
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,
 -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
(610803
 370302
 -22561
 110012
 135946
sage:
```

```
dentity(N)

K)

[0][0]/E

/q0)
```



```
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, 1
 37030242384, 84
 -225618319442,
 1100126026284,
 1359463649048,
sage:
```

)


```
sage: V[0]
(1024,
 -1111539179100720083770339,
 794301459533783434896055,
 68817802108374958901751,
 742362470968200823035396,
 1023345827831539515054795,
 -357168679398558876730006,
 1121421619119964601051443,
 -1109674862276222495587129,
 -235628937785003770523381)
sage: V[1]
(0, -58747338058640662659869,
 0, 0, 0, 0, 0, 0, 0, 0)
sage:
```



```
sage: V.LLL()[0]
(610803584000, 1056189937
 37030242384, 84589845469
 -225618319442, 363547143
 1100126026284, -31315097
 1359463649048, 174256676
sage:
```

```
sage: V[0]
(1024,
 -1111539179100720083770339,
 794301459533783434896055,
 688178021083749589017S1,
 742362470968200823035396,
 1023345827831539515054795,
 -3571686793985588767300O6,
 1121421619119964601051443,
 -1109674862276222495587129,
 -235628937785003770523381)
sage: V[1]
(0, -5874733380586406626598 69,
 0, 0, 0, 0, 0, 0, 0, 0)
sage:
```



```
sage: V.LLL()[0]
(610803584000, 1056189937254,
 37030242384, 845898454698,
 -225618319442, 363547143644,
 1100126026284, -313150978512,
 1359463649048, 174256676348)
sage:
```

```
sage: V[0]
(1024,
 -1111539179100720083770339,
 7943014595337834348960 55,
 6881780210837495890175 1,
 7423624709682008230353 96,
 1023345827831539515054795,
 -3571686793985588767300 06,
 1121421619119964601051443,
 -1109674862276222495587129,
 -2356289377850037705233 81)
sage: V[1]
(0, -5874733380586406626598 69,
 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:
```

```
sage: V[0]
(1024,
 -1111539179100720083770339,
 794301459533783434896055,
 688178021083749589017516,
 742362470968200823035396,
 1023345827831539515054795,
 -3571686793985588767300006,
 1121421619119964601051443,
 -1109674862276222495587129,
 -235628937785003770523381)
sage: V[1]
(0, -587473338058640662659869,
 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:
```

```
sage: V[0]
(1024,
 -1111539179100720083770339,
 7943014595337834348960 55,
 6881780210837495890175 1,
 7423624709682008230353 96,
 1023345827831539515054795,
 -3571686793985588767300 06,
 1121421619119964601051443,
 -1109674862276222495587129,
 -2356289377850037705233 81)
sage: V[1]
(0, -5874733380586406626598 69,
 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:
```

```
sage: V[0]
(1024,
 -1111539179100720083770339,
 794301459533783434896055,
 68817802108374958901751,
 742362470968200823035396,
 1023345827831539515054795,
 -357168679398558876730006,
 1121421619119964601051443,
 -1109674862276222495587129,
 -235628937785003770523381)
sage: V[1]
(0, -58747338058640662659869,
 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,
8937785003770523381)
[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

```
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

```
339,
,
,
05,
06,
13,
29,
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.

```
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.

```
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.”

```
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

```
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.

```
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,
8319442, 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
Pfitzmar
define C
for suita

Simple,
Very eas
finding C
computi

```
056189937254,
5898454698,
363547143644,
-31315097851 2,
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

/254,
08,
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

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.

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.

ron–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-quar

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

cles
3.

attack.
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.
sy 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

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

gerous

y;
p
y.
l.
:

hm.

free";
us

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

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.

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.

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

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

us);
-
or);
FS);
'S);
me);
eedups

limit
th
-
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

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!

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 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.

ic-key encryption:
athematical structure
› be unavoidable,
suing simple structures
ads to security disasters.

ntum example: DH is
than ECDH, but DH has
many more security losses
DH. State-of-the-art DH
are very complicated.

rbulescu–Gaudry–Joux–
pre-quantum quasi-poly
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.

NISTPQ

NIST re
69 subm
from hu
22 signa
47 encry

ryption:
al structure
oidable,
le structures
urity disasters.

mple: DH is
H, but DH has
re security losses
e-of-the-art DH
omplicated.

Gaudry–Joux–
tum quasi-poly
racteristic 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.

NISTPQC

NIST received 82
69 submissions in
from hundreds of
22 signature subm
47 encryption sub

re

res
ters.

is
l has
losses
t DH
d.

ux–
-poly
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.

NISTPQC

NIST received 82 submissio
69 submissions in round 1,
from hundreds of people;
22 signature submissions,
47 encryption submissions.

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.

NISTPQC

NIST received 82 submissions.
69 submissions in round 1, from hundreds of people;
22 signature submissions,
47 encryption submissions.

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.

NISTPQC

NIST received 82 submissions.
69 submissions in round 1, from hundreds of people;
22 signature submissions,
47 encryption submissions.

26 submissions in round 2:
9 signature submissions;
17 encryption submissions.

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.

NISTPQC

NIST received 82 submissions.
69 submissions in round 1, from hundreds of people;
22 signature submissions,
47 encryption submissions.

26 submissions in round 2:
9 signature submissions;
17 encryption submissions.

Round 3 starting soon.
My guesses: NIST will announce short list of planned standards + short backup list; and will overemphasize speed.

te-of-the-art attacks
Cohen's cryptosystem
h more complicated
e cryptosystem is. Scary!

pased cryptosystems are
ed as "algorithmically
consisting mainly of
operations on vectors".
exploit this structure!

iency, lattice-based
stems usually have
that expand the attack
even more: e.g.,
d decryption failures.

NISTPQC

NIST received 82 submissions.
69 submissions in round 1,
from hundreds of people;
22 signature submissions,
47 encryption submissions.

26 submissions in round 2:
9 signature submissions;
17 encryption submissions.

Round 3 starting soon.
My guesses: NIST will announce
short list of planned standards
+ short backup list; and will
overemphasize speed.

Lattice-b

- Dilithi
- DRS:
- FALCO
- pqNTI
- qTESL
  "theor
  param

rt attacks
ryptosystem
mplicated
stem is. Scary!

tosystems are
orithmically
g mainly of
on vectors".
is structure!

ice-based
ally have
nd the attack
: e.g.,
on failures.

NISTPQC

NIST received 82 submissions.
69 submissions in round 1,
from hundreds of people;
22 signature submissions,
47 encryption submissions.

26 submissions in round 2:
9 signature submissions;
17 encryption submissions.

Round 3 starting soon.
My guesses: NIST will announce
short list of planned standards
+ short backup list; and will
overemphasize speed.

Lattice-based sign

- Dilithium: round
- DRS: **broken**; el
- FALCON☢: rou
- pqNTRUSign☢:
- qTESLA: mistak
  "theorems"; rou
  parameters bro

m

cary!

are

y

f

s".

e!

ack

.

NISTPQC

NIST received 82 submissions.
69 submissions in round 1,
from hundreds of people;
22 signature submissions,
47 encryption submissions.

26 submissions in round 2:
9 signature submissions;
17 encryption submissions.

Round 3 starting soon.
My guesses: NIST will announce
short list of planned standards
+ short backup list; and will
overemphasize speed.

Lattice-based signature subr

- Dilithium: round 2.
- DRS: **broken**; eliminated.
- FALCON☢: round 2.
- pqNTRUSign☢: eliminate
- qTESLA: mistaken securit "theorems"; round 2; **som parameters broken**.

NISTPQC

NIST received 82 submissions.
69 submissions in round 1,
from hundreds of people;
22 signature submissions,
47 encryption submissions.

26 submissions in round 2:
9 signature submissions;
17 encryption submissions.

Round 3 starting soon.
My guesses: NIST will announce
short list of planned standards
+ short backup list; and will
overemphasize speed.

Lattice-based signature submissions:

- Dilithium: round 2.
- DRS: **broken**; eliminated.
- FALCON☢: round 2.
- pqNTRUSign☢: eliminated.
- qTESLA: mistaken security "theorems"; round 2; **some parameters broken**.

NISTPQC

NIST received 82 submissions.
69 submissions in round 1,
from hundreds of people;
22 signature submissions,
47 encryption submissions.

26 submissions in round 2:
9 signature submissions;
17 encryption submissions.

Round 3 starting soon.
My guesses: NIST will announce short list of planned standards + short backup list; and will overemphasize speed.

Lattice-based signature submissions:

- Dilithium: round 2.
- DRS: **broken**; eliminated.
- FALCON☢: round 2.
- pqNTRUSign☢: eliminated.
- qTESLA: mistaken security "theorems"; round 2; **some parameters broken**.

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

QC

ceived 82 submissions.
issions in round 1,
ndreds of people;
ture submissions,
ption submissions.

issions in round 2:
ure submissions;
ption submissions.

3 starting soon.
ses: NIST will announce
t of planned standards
backup list; and will
hasize speed.

Lattice-based signature submissions:

- Dilithium: round 2.
- DRS: **broken**; eliminated.
- FALCON☢: round 2.
- pqNTRUSign☢: eliminated.
- qTESLA: mistaken security "theorems"; round 2; **some parameters broken**.

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

Lattice-b
submissi
Kyber, L
NTRU P
ThreeBe

submissions.
round 1,
people;
issions,
missions.

round 2:
ssions;
missions.

soon.
T will announce
ed standards
st; and will
eed.

Lattice-based signature submissions:

- Dilithium: round 2.
- DRS: **broken**; eliminated.
- FALCON☢: round 2.
- pqNTRUSign☢: eliminated.
- qTESLA: mistaken security "theorems"; round 2; **some parameters broken**.

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

Lattice-based encr
submissions in rou
Kyber, LAC, NewH
NTRU Prime, Rou
ThreeBears (≈latt

ns.

unce

ds

l

Lattice-based signature submissions:

- Dilithium: round 2.
- DRS: **broken**; eliminated.
- FALCON☢: round 2.
- pqNTRUSign☢: eliminated.
- qTESLA: mistaken security "theorems"; round 2; **some parameters broken**.

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

Lattice-based encryption submissions in round 2: Fro Kyber, LAC, NewHope, NTI NTRU Prime, Round5☢, SA ThreeBears (≈lattice).

Lattice-based signature submissions:

- Dilithium: round 2.
- DRS: **broken**; eliminated.
- FALCON☢: round 2.
- pqNTRUSign☢: eliminated.
- qTESLA: mistaken security "theorems"; round 2; **some parameters broken**.

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

Lattice-based encryption submissions in round 2: Frodo, Kyber, LAC, NewHope, NTRU, NTRU Prime, Round5☢, SABER, ThreeBears (≈lattice).

Lattice-based signature submissions:

- Dilithium: round 2.
- DRS: **broken**; eliminated.
- FALCON☢: round 2.
- pqNTRUSign☢: eliminated.
- qTESLA: mistaken security "theorems"; round 2; **some parameters broken**.

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

Lattice-based encryption submissions in round 2: Frodo, Kyber, LAC, NewHope, NTRU, NTRU Prime, Round5☢, SABER, ThreeBears (≈lattice).

Other round-1 lattice-based encryption submissions: Compact LWE☢ (**broken**), Ding☢, EMBLEM, KINDI, LIMA, Lizard☢, LOTUS, Mersenne (≈lattice, big keys), Odd Manhattan (big keys), OKCN/AKCN/CNKE/KCL☢, Ramstake (≈lattice, big keys), Titanium.

pased signature submissions:

um: round 2.

broken; eliminated.

ON☢: round 2.

RUSign☢: eliminated.

LA: mistaken security
ems"; round 2; **some**
neters broken.

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

Lattice-based encryption submissions in round 2: Frodo, Kyber, LAC, NewHope, NTRU, NTRU Prime, Round5☢, SABER, ThreeBears (≈lattice).

Other round-1 lattice-based encryption submissions:
Compact LWE☢ (**broken**),
Ding☢, EMBLEM, KINDI,
LIMA, Lizard☢, LOTUS,
Mersenne (≈lattice, big keys),
Odd Manhattan (big keys),
OKCN/AKCN/CNKE/KCL☢,
Ramstake (≈lattice, big keys),
Titanium.

NTRU is
with NT

ature submissions:

d 2.

liminated.

ınd 2.

: eliminated.

ken security
nd 2; **some**
ken.

ms patent on
Varning: even
ssion could be
patents!

Lattice-based encryption submissions in round 2: Frodo, Kyber, LAC, NewHope, NTRU, NTRU Prime, Round5☢, SABER, ThreeBears (≈lattice).

Other round-1 lattice-based encryption submissions: Compact LWE☢ (**broken**), Ding☢, EMBLEM, KINDI, LIMA, Lizard☢, LOTUS, Mersenne (≈lattice, big keys), Odd Manhattan (big keys), OKCN/AKCN/CNKE/KCL☢, Ramstake (≈lattice, big keys), Titanium.

NTRU is merge of
with NTRU HRSS

missions:

ed.

y

ne

on

even

d be

Lattice-based encryption submissions in round 2: Frodo, Kyber, LAC, NewHope, NTRU, NTRU Prime, Round5☢, SABER, ThreeBears (≈lattice).

Other round-1 lattice-based encryption submissions: Compact LWE☢ (**broken**), Ding☢, EMBLEM, KINDI, LIMA, Lizard☢, LOTUS, Mersenne (≈lattice, big keys), Odd Manhattan (big keys), OKCN/AKCN/CNKE/KCL☢, Ramstake (≈lattice, big keys), Titanium.

NTRU is merge of NTRUEn with NTRU HRSS.

Lattice-based encryption submissions in round 2: Frodo, Kyber, LAC, NewHope, NTRU, NTRU Prime, Round5☢, SABER, ThreeBears (≈lattice).

Other round-1 lattice-based encryption submissions: Compact LWE☢ (**broken**), Ding☢, EMBLEM, KINDI, LIMA, Lizard☢, LOTUS, Mersenne (≈lattice, big keys), Odd Manhattan (big keys), OKCN/AKCN/CNKE/KCL☢, Ramstake (≈lattice, big keys), Titanium.

NTRU is merge of NTRUEncrypt with NTRU HRSS.

Lattice-based encryption submissions in round 2: Frodo, Kyber, LAC, NewHope, NTRU, NTRU Prime, Round5☢, SABER, ThreeBears (≈lattice).

Other round-1 lattice-based encryption submissions: Compact LWE☢ (**broken**), Ding☢, EMBLEM, KINDI, LIMA, Lizard☢, LOTUS, Mersenne (≈lattice, big keys), Odd Manhattan (big keys), OKCN/AKCN/CNKE/KCL☢, Ramstake (≈lattice, big keys), Titanium.

NTRU is merge of NTRUEncrypt with NTRU HRSS.

Round5☢ is merge of HILA5 with Round2☢. HILA5 **CCA security claim broken**. First Round5 version **broken** before round 2 began. Round2 **broken** after round 2 began.

Lattice-based encryption submissions in round 2: Frodo, Kyber, LAC, NewHope, NTRU, NTRU Prime, Round5☢, SABER, ThreeBears (≈lattice).

Other round-1 lattice-based encryption submissions: Compact LWE☢ (**broken**), Ding☢, EMBLEM, KINDI, LIMA, Lizard☢, LOTUS, Mersenne (≈lattice, big keys), Odd Manhattan (big keys), OKCN/AKCN/CNKE/KCL☢, Ramstake (≈lattice, big keys), Titanium.

NTRU is merge of NTRUEncrypt with NTRU HRSS.

Round5☢ is merge of HILA5 with Round2☢. HILA5 **CCA security claim broken**. First Round5 version **broken** before round 2 began. Round2 **broken** after round 2 began.

Mistaken security "theorems" have been identified for Frodo, Kyber, NewHope, Round5.

Lattice-based encryption submissions in round 2: Frodo, Kyber, LAC, NewHope, NTRU, NTRU Prime, Round5☢, SABER, ThreeBears (≈lattice).

Other round-1 lattice-based encryption submissions: Compact LWE☢ (**broken**), Ding☢, EMBLEM, KINDI, LIMA, Lizard☢, LOTUS, Mersenne (≈lattice, big keys), Odd Manhattan (big keys), OKCN/AKCN/CNKE/KCL☢, Ramstake (≈lattice, big keys), Titanium.

NTRU is merge of NTRUEncrypt with NTRU HRSS.

Round5☢ is merge of HILA5 with Round2☢. HILA5 **CCA security claim broken**. First Round5 version **broken** before round 2 began. Round2 **broken** after round 2 began.

Mistaken security “theorems” have been identified for Frodo, Kyber, NewHope, Round5.

All lattice submissions have suffered security losses.

based encryption
ons in round 2: Frodo,
LAC, NewHope, NTRU,
Prime, Round5☢, SABER,
ears (≈lattice).

ound-1 lattice-based
on submissions:
t LWE☢ (**broken**),
EMBLEM, KINDI,
Lizard☢, LOTUS,
e (≈lattice, big keys),
nhattan (big keys),
AKCN/CNKE/KCL☢,
ke (≈lattice, big keys),
n.

NTRU is merge of NTRUEncrypt with NTRU HRSS.

Round5☢ is merge of HILA5 with Round2☢. HILA5 **CCA security claim broken**. First Round5 version **broken** before round 2 began. Round2 **broken** after round 2 began.

Mistaken security "theorems" have been identified for Frodo, Kyber, NewHope, Round5.

All lattice submissions have suffered security losses.

Example
after beg

2018 La
"betwee
sieving,
SVP att

2018 Ba
variant,
for the "

2018 Ao
quantum
cryptogr
than siev

yption
nd 2: Frodo,
Hope, NTRU,
und5☢, SABER,
ice).

tice-based
sions:
(**broken**),
I, KINDI,
LOTUS,
e, big keys),
big keys),
IKE/KCL☢,
ce, big keys),

NTRU is merge of NTRUEncrypt with NTRU HRSS.

Round5☢ is merge of HILA5 with Round2☢. HILA5 **CCA security claim broken**. First Round5 version **broken** before round 2 began. Round2 **broken** after round 2 began.

Mistaken security "theorems" have been identified for Frodo, Kyber, NewHope, Round5.

All lattice submissions have suffered security losses.

Examples of attac
after beginning of

2018 Laarhoven–M
"between a factor
sieving, asymptoti
SVP attack known

2018 Bai–Stehlé–W
variant, "bases of
for the "same cost

2018 Aono–Nguye
quantum enumera
cryptographic size
than sieving in so

do,
RU,
ABER,

s),

☢,
s),

NTRU is merge of NTRUEncrypt with NTRU HRSS.

Round5☢ is merge of HILA5 with Round2☢. HILA5 **CCA security claim broken**. First Round5 version **broken** before round 2 began. Round2 **broken** after round 2 began.

Mistaken security "theorems" have been identified for Frodo, Kyber, NewHope, Round5.

All lattice submissions have suffered security losses.

Examples of attack improve
after beginning of round 1:

2018 Laarhoven–Mariano: s
"between a factor 20 to 40"
sieving, asymptotically faste
SVP attack known.

2018 Bai–Stehlé–Wen: new
variant, "bases of better qu
for the "same cost" of SVP.

2018 Aono–Nguyen–Shen:
quantum enumeration. For
cryptographic sizes, costs les
than sieving in some cost m

NTRU is merge of NTRUEncrypt with NTRU HRSS.

Round5☢ is merge of HILA5 with Round2☢. HILA5 **CCA security claim broken**. First Round5 version **broken** before round 2 began. Round2 **broken** after round 2 began.

Mistaken security "theorems" have been identified for Frodo, Kyber, NewHope, Round5.

All lattice submissions have suffered security losses.

Examples of attack improvements after beginning of round 1:

2018 Laarhoven–Mariano: saves "between a factor 20 to 40" in sieving, asymptotically fastest SVP attack known.

2018 Bai–Stehlé–Wen: new BKZ variant, "bases of better quality" for the "same cost" of SVP.

2018 Aono–Nguyen–Shen: quantum enumeration. For cryptographic sizes, costs less than sieving in some cost metrics.

s merge of NTRUEncrypt
TRU HRSS.

☢ is merge of HILA5
und2☢. HILA5 **CCA**
claim broken. First
version **broken** before
began. Round2 **broken**
und 2 began.

n security "theorems"
en identified for Frodo,
NewHope, Round5.

ce submissions have
l security losses.

Examples of attack improvements after beginning of round 1:

2018 Laarhoven–Mariano: saves "between a factor 20 to 40" in sieving, asymptotically fastest SVP attack known.

2018 Bai–Stehlé–Wen: new BKZ variant, "bases of better quality" for the "same cost" of SVP.

2018 Aono–Nguyen–Shen: quantum enumeration. For cryptographic sizes, costs less than sieving in some cost metrics.

2018 An
Verbauw
significa
(Ring/M
schemes
high fail

Frodo, K
Round5,
have no

For LAC
2^{48} time
Failure r
first vers

f NTRUEncrypt
.

e of HILA5
HILA5 **CCA**
oken. First
roken before
ound2 **broken**
an.

“theorems”
ed for Frodo,
Round5.

ssions have
losses.

Examples of attack improvements after beginning of round 1:

2018 Laarhoven–Mariano: saves “between a factor 20 to 40” in sieving, asymptotically fastest SVP attack known.

2018 Bai–Stehlé–Wen: new BKZ variant, “bases of better quality” for the “same cost” of SVP.

2018 Aono–Nguyen–Shen: quantum enumeration. For cryptographic sizes, costs less than sieving in some cost metrics.

2018 Anvers–Verc
Verbauwhede: “ar
significantly reduc
(Ring/Module)-LW
schemes that have
high failure rate”.

Frodo, Kyber, LAC
Round5, SABER,
have nonzero failu

For LAC-128, “the
2^{48} times bigger t
Failure rate is also
first version of Ro

crypt

5

A

st

ore

oken

s”

do,

ve

Examples of attack improvements after beginning of round 1:

2018 Laarhoven–Mariano: saves “between a factor 20 to 40” in sieving, asymptotically fastest SVP attack known.

2018 Bai–Stehlé–Wen: new BKZ variant, “bases of better quality” for the “same cost” of SVP.

2018 Aono–Nguyen–Shen: quantum enumeration. For cryptographic sizes, costs less than sieving in some cost metrics.

2018 Anvers–Vercauteren–
Verbauwhede: “an attacker
significantly reduce the secu
(Ring/Module)-LWE/LWR b
schemes that have a relative
high failure rate”.

Frodo, Kyber, LAC, NewHo
Round5, SABER, ThreeBear
have nonzero failure rates.

For LAC-128, “the failure ra
2^{48} times bigger than estima
Failure rate is also what bro
first version of Round5.

Examples of attack improvements after beginning of round 1:

2018 Laarhoven–Mariano: saves "between a factor 20 to 40" in sieving, asymptotically fastest SVP attack known.

2018 Bai–Stehlé–Wen: new BKZ variant, "bases of better quality" for the "same cost" of SVP.

2018 Aono–Nguyen–Shen: quantum enumeration. For cryptographic sizes, costs less than sieving in some cost metrics.

2018 Anvers–Vercauteren–Verbauwhede: "an attacker can significantly reduce the security of (Ring/Module)-LWE/LWR based schemes that have a relatively high failure rate".

Frodo, Kyber, LAC, NewHope, Round5, SABER, ThreeBears have nonzero failure rates.

For LAC-128, "the failure rate is 2^{48} times bigger than estimated". Failure rate is also what broke first version of Round5.

es of attack improvements
ginning of round 1:

arhoven–Mariano: saves
n a factor 20 to 40" in
asymptotically fastest
ack known.

i–Stehlé–Wen: new BKZ
"bases of better quality"
'same cost" of SVP.

no–Nguyen–Shen:
n enumeration. For
aphic sizes, costs less
ving in some cost metrics.

2018 Anvers–Vercauteren–Verbauwhede: "an attacker can significantly reduce the security of (Ring/Module)-LWE/LWR based schemes that have a relatively high failure rate".

Frodo, Kyber, LAC, NewHope, Round5, SABER, ThreeBears have nonzero failure rates.

For LAC-128, "the failure rate is 2^{48} times bigger than estimated". Failure rate is also what broke first version of Round5.

2019 Alb
Kirshano
Stevens:
the SVP
found 40
time rep
challeng

2019 Pe
broke cla
approxin
number–
Ideal-SV
cycloton
FHE in

k improvements
round 1:

Mariano: saves
20 to 40" in
cally fastest
n.

Wen: new BKZ
better quality"
t" of SVP.

n–Shen:
tion. For
s, costs less
me cost metrics.

2018 Anvers–Vercauteren–Verbauwhede: "an attacker can significantly reduce the security of (Ring/Module)-LWE/LWR based schemes that have a relatively high failure rate".

Frodo, Kyber, LAC, NewHope, Round5, SABER, ThreeBears have nonzero failure rates.

For LAC-128, "the failure rate is 2^{48} times bigger than estimated". Failure rate is also what broke first version of Round5.

2019 Albrecht–Du
Kirshanova–Postle
Stevens: "Our sol
the SVP-151 chall
found 400 times fa
time reported for t
challenge, the prev

2019 Pellet-Mary–
broke claimed half
approximation-fact
number-theoretic a
Ideal-SVP. (These
cyclotomic STOC
FHE in quantum p

ments

aves
in
st

BKZ
ality"

ss
etrics.

2018 Anvers–Vercauteren–Verbauwhede: "an attacker can significantly reduce the security of (Ring/Module)-LWE/LWR based schemes that have a relatively high failure rate".

Frodo, Kyber, LAC, NewHope, Round5, SABER, ThreeBears have nonzero failure rates.

For LAC-128, "the failure rate is 2^{48} times bigger than estimated". Failure rate is also what broke first version of Round5.

2019 Albrecht–Ducas–Herol
Kirshanova–Postlethwaite–
Stevens: "Our solution for
the SVP-151 challenge was
found 400 times faster than
time reported for the SVP-1
challenge, the previous recor

2019 Pellet-Mary–Hanrot–St
broke claimed half-exponent
approximation-factor barrier
number-theoretic attacks ag
Ideal-SVP. (These attacks b
cyclotomic STOC 2009 Gen
FHE in quantum poly time.)

2018 Anvers–Vercauteren–Verbauwhede: “an attacker can significantly reduce the security of (Ring/Module)-LWE/LWR based schemes that have a relatively high failure rate”.

Frodo, Kyber, LAC, NewHope, Round5, SABER, ThreeBears have nonzero failure rates.

For LAC-128, “the failure rate is 2^{48} times bigger than estimated”. Failure rate is also what broke first version of Round5.

2019 Albrecht–Ducas–Herold–Kirshanova–Postlethwaite–Stevens: “Our solution for the SVP-151 challenge was found 400 times faster than the time reported for the SVP-150 challenge, the previous record.”

2019 Pellet-Mary–Hanrot–Stehlé broke claimed half-exponential approximation-factor barrier for number-theoretic attacks against Ideal-SVP. (These attacks broke cyclotomic STOC 2009 Gentry FHE in quantum poly time.)

vers–Vercauteren–
vhede: "an attacker can
ntly reduce the security of
lodule)-LWE/LWR based
that have a relatively
ure rate".

(yber, LAC, NewHope,
SABER, ThreeBears
nzero failure rates.

-128, "the failure rate is
es bigger than estimated".
rate is also what broke
sion of Round5.

2019 Albrecht–Ducas–Herold–Kirshanova–Postlethwaite–Stevens: "Our solution for the SVP-151 challenge was found 400 times faster than the time reported for the SVP-150 challenge, the previous record."

2019 Pellet-Mary–Hanrot–Stehlé broke claimed half-exponential approximation-factor barrier for number-theoretic attacks against Ideal-SVP. (These attacks broke cyclotomic STOC 2009 Gentry FHE in quantum poly time.)

2019 Gu
faster at
systems
to reduc
(Violate

2020 Da
Gong–R
attacks
secrets (

2020 Alb
Kirchner
exponen
quantum

auteren–
attacker can
e the security of
VE/LWR based
e a relatively

C, NewHope,
ThreeBears
re rates.

e failure rate is
han estimated".
what broke
und5.

2019 Albrecht–Ducas–Herold–Kirshanova–Postlethwaite–Stevens: "Our solution for the SVP-151 challenge was found 400 times faster than the time reported for the SVP-150 challenge, the previous record."

2019 Pellet-Mary–Hanrot–Stehlé broke claimed half-exponential approximation-factor barrier for number-theoretic attacks against Ideal-SVP. (These attacks broke cyclotomic STOC 2009 Gentry FHE in quantum poly time.)

2019 Guo–Johans
faster attacks aga
systems that use
to reduce decrypt
(Violates security

2020 Dachman-So
Gong–Rossi: sligh
attacks against co
secrets (LAC, NT

2020 Albrecht–Ba
Kirchner–Stehlé–V
exponent for enum
quantum enumera

can
rity of
based
ely

pe,
rs

te is
ated".
ke

2019 Albrecht–Ducas–Herold–Kirshanova–Postlethwaite–Stevens: "Our solution for the SVP-151 challenge was found 400 times faster than the time reported for the SVP-150 challenge, the previous record."

2019 Pellet-Mary–Hanrot–Stehlé broke claimed half-exponential approximation-factor barrier for number-theoretic attacks against Ideal-SVP. (These attacks broke cyclotomic STOC 2009 Gentry FHE in quantum poly time.)

2019 Guo–Johansson–Yang:
faster attacks against some
systems that use error correc
to reduce decryption failures
(Violates security claims for

2020 Dachman-Soled–Ducas
Gong–Rossi: slightly faster
attacks against constant-sur
secrets (LAC, NTRU, Round

2020 Albrecht–Bai–Fouque–
Kirchner–Stehlé–Wen: bette
exponent for enumeration ar
quantum enumeration.

2019 Albrecht–Ducas–Herold–Kirshanova–Postlethwaite–Stevens: "Our solution for the SVP-151 challenge was found 400 times faster than the time reported for the SVP-150 challenge, the previous record."

2019 Pellet-Mary–Hanrot–Stehlé broke claimed half-exponential approximation-factor barrier for number-theoretic attacks against Ideal-SVP. (These attacks broke cyclotomic STOC 2009 Gentry FHE in quantum poly time.)

2019 Guo–Johansson–Yang: faster attacks against some systems that use error correction to reduce decryption failures. (Violates security claims for LAC.)

2020 Dachman-Soled–Ducas–Gong–Rossi: slightly faster attacks against constant-sum secrets (LAC, NTRU, Round5).

2020 Albrecht–Bai–Fouque–Kirchner–Stehlé–Wen: better exponent for enumeration and quantum enumeration.

precht–Ducas–Herold–
ova–Postlethwaite–
"Our solution for
-151 challenge was
00 times faster than the
orted for the SVP-150
e, the previous record."

llet-Mary–Hanrot–Stehlé
aimed half-exponential
nation-factor barrier for
theoretic attacks against
P. (These attacks broke
hic STOC 2009 Gentry
quantum poly time.)

2019 Guo–Johansson–Yang: faster attacks against some systems that use error correction to reduce decryption failures. (Violates security claims for LAC.)

2020 Dachman-Soled–Ducas–Gong–Rossi: slightly faster attacks against constant-sum secrets (LAC, NTRU, Round5).

2020 Albrecht–Bai–Fouque–Kirchner–Stehlé–Wen: better exponent for enumeration and quantum enumeration.

2020 Do
de Wege
methods
vector p
dimensio

cas–Herold–
ethwaite–
ution for
enge was
aster than the
the SVP-150
vious record.”

-Hanrot–Stehlé
-exponential
tor barrier for
attacks against
attacks broke
2009 Gentry
poly time.)

2019 Guo–Johansson–Yang: faster attacks against some systems that use error correction to reduce decryption failures. (Violates security claims for LAC.)

2020 Dachman-Soled–Ducas–Gong–Rossi: slightly faster attacks against constant-sum secrets (LAC, NTRU, Round5).

2020 Albrecht–Bai–Fouque–Kirchner–Stehlé–Wen: better exponent for enumeration and quantum enumeration.

2020 Doulgerakis–
de Weger: “faster
methods for solvin
vector problem (S
dimensional lattice

---

d–

the
50
rd.”

tehlé
ial
for
ainst
roke
try
)

2019 Guo–Johansson–Yang: faster attacks against some systems that use error correction to reduce decryption failures. (Violates security claims for LAC.)

2020 Dachman-Soled–Ducas–Gong–Rossi: slightly faster attacks against constant-sum secrets (LAC, NTRU, Round5).

2020 Albrecht–Bai–Fouque–Kirchner–Stehlé–Wen: better exponent for enumeration and quantum enumeration.

2020 Doulgerakis–Laarhover
de Weger: “faster [sieving]
methods for solving the shor
vector problem (SVP) on hig
dimensional lattices”.

2019 Guo–Johansson–Yang: faster attacks against some systems that use error correction to reduce decryption failures. (Violates security claims for LAC.)

2020 Dachman-Soled–Ducas–Gong–Rossi: slightly faster attacks against constant-sum secrets (LAC, NTRU, Round5).

2020 Albrecht–Bai–Fouque–Kirchner–Stehlé–Wen: better exponent for enumeration and quantum enumeration.

2020 Doulgerakis–Laarhoven–de Weger: "faster [sieving] methods for solving the shortest vector problem (SVP) on high-dimensional lattices".

2019 Guo–Johansson–Yang: faster attacks against some systems that use error correction to reduce decryption failures. (Violates security claims for LAC.)

2020 Dachman-Soled–Ducas–Gong–Rossi: slightly faster attacks against constant-sum secrets (LAC, NTRU, Round5).

2020 Albrecht–Bai–Fouque–Kirchner–Stehlé–Wen: better exponent for enumeration and quantum enumeration.

2020 Doulgerakis–Laarhoven–de Weger: "faster [sieving] methods for solving the shortest vector problem (SVP) on high-dimensional lattices".

"Conservative lower bound" on cost of BKZ was claimed in various submission documents in 2017 (round 1), 2019 (round 2).

2019 Guo–Johansson–Yang: faster attacks against some systems that use error correction to reduce decryption failures. (Violates security claims for LAC.)

2020 Dachman-Soled–Ducas–Gong–Rossi: slightly faster attacks against constant-sum secrets (LAC, NTRU, Round5).

2020 Albrecht–Bai–Fouque–Kirchner–Stehlé–Wen: better exponent for enumeration and quantum enumeration.

2020 Doulgerakis–Laarhoven–de Weger: "faster [sieving] methods for solving the shortest vector problem (SVP) on high-dimensional lattices".

"Conservative lower bound" on cost of BKZ was claimed in various submission documents in 2017 (round 1), 2019 (round 2). This "bound" was broken in 2018 for high-dimensional lattices.

2019 Guo–Johansson–Yang: faster attacks against some systems that use error correction to reduce decryption failures. (Violates security claims for LAC.)

2020 Dachman-Soled–Ducas–Gong–Rossi: slightly faster attacks against constant-sum secrets (LAC, NTRU, Round5).

2020 Albrecht–Bai–Fouque–Kirchner–Stehlé–Wen: better exponent for enumeration and quantum enumeration.

2020 Doulgerakis–Laarhoven–de Weger: "faster [sieving] methods for solving the shortest vector problem (SVP) on high-dimensional lattices".

"Conservative lower bound" on cost of BKZ was claimed in various submission documents in 2017 (round 1), 2019 (round 2). This "bound" was broken in 2018 for high-dimensional lattices. Apparently nobody noticed until I pointed this out in 2020.

uo–Johansson–Yang:
ttacks against some
that use error correction
e decryption failures.
s security claims for LAC.)

chman-Soled–Ducas–
ossi: slightly faster
against constant-sum
(LAC, NTRU, Round5).

brecht–Bai–Fouque–
r–Stehlé–Wen: better
t for enumeration and
n enumeration.

2020 Doulgerakis–Laarhoven–de Weger: "faster [sieving] methods for solving the shortest vector problem (SVP) on high-dimensional lattices".

"Conservative lower bound" on cost of BKZ was claimed in various submission documents in 2017 (round 1), 2019 (round 2). This "bound" was broken in 2018 for high-dimensional lattices. Apparently nobody noticed until I pointed this out in 2020.

Lattice r

"Strong
worst-ca
that "ha
some of
and com
back at

son–Yang:
inst some
error correction
on failures.
claims for LAC.)

oled–Ducas–
tly faster
nstant-sum
RU, Round5).

i–Fouque–
Ven: better
neration and
tion.

2020 Doulgerakis–Laarhoven–de Weger: "faster [sieving] methods for solving the shortest vector problem (SVP) on high-dimensional lattices".

"Conservative lower bound" on cost of BKZ was claimed in various submission documents in 2017 (round 1), 2019 (round 2). This "bound" was broken in 2018 for high-dimensional lattices. Apparently nobody noticed until I pointed this out in 2020.

Lattice marketing

"Strong security g
worst-case hardnes
that "have been d
some of the great
and computer scie
back at least to G

ction
s.
LAC.)
s–
m
d5).
er
nd

2020 Doulgerakis–Laarhoven–de Weger: "faster [sieving] methods for solving the shortest vector problem (SVP) on high-dimensional lattices".

"Conservative lower bound" on cost of BKZ was claimed in various submission documents in 2017 (round 1), 2019 (round 2). This "bound" was broken in 2018 for high-dimensional lattices. Apparently nobody noticed until I pointed this out in 2020.

Lattice marketing

"Strong security guarantees *worst-case* hardness" of prob that "have been deeply stud some of the great mathemat and computer scientists goin back at least to Gauss".

2020 Doulgerakis–Laarhoven–de Weger: "faster [sieving] methods for solving the shortest vector problem (SVP) on high-dimensional lattices".

"Conservative lower bound" on cost of BKZ was claimed in various submission documents in 2017 (round 1), 2019 (round 2). This "bound" was broken in 2018 for high-dimensional lattices. Apparently nobody noticed until I pointed this out in 2020.

Lattice marketing

"Strong security guarantees from *worst-case* hardness" of problems that "have been deeply studied by some of the great mathematicians and computer scientists going back at least to Gauss".

2020 Doulgerakis–Laarhoven–de Weger: “faster [sieving] methods for solving the shortest vector problem (SVP) on high-dimensional lattices”.

“Conservative lower bound” on cost of BKZ was claimed in various submission documents in 2017 (round 1), 2019 (round 2). This “bound” was broken in 2018 for high-dimensional lattices. Apparently nobody noticed until I pointed this out in 2020.

Lattice marketing

“Strong security guarantees from *worst-case* hardness” of problems that “have been deeply studied by some of the great mathematicians and computer scientists going back at least to Gauss”. Plus: fully homomorphic encryption.

2020 Doulgerakis–Laarhoven–de Weger: "faster [sieving] methods for solving the shortest vector problem (SVP) on high-dimensional lattices".

"Conservative lower bound" on cost of BKZ was claimed in various submission documents in 2017 (round 1), 2019 (round 2). This "bound" was broken in 2018 for high-dimensional lattices. Apparently nobody noticed until I pointed this out in 2020.

Lattice marketing

"Strong security guarantees from *worst-case* hardness" of problems that "have been deeply studied by some of the great mathematicians and computer scientists going back at least to Gauss". Plus: fully homomorphic encryption.

Facts: No NISTPQC submissions are homomorphic.

2020 Doulgerakis–Laarhoven–de Weger: "faster [sieving] methods for solving the shortest vector problem (SVP) on high-dimensional lattices".

"Conservative lower bound" on cost of BKZ was claimed in various submission documents in 2017 (round 1), 2019 (round 2). This "bound" was broken in 2018 for high-dimensional lattices. Apparently nobody noticed until I pointed this out in 2020.

Lattice marketing

"Strong security guarantees from *worst-case* hardness" of problems that "have been deeply studied by some of the great mathematicians and computer scientists going back at least to Gauss". Plus: fully homomorphic encryption.

Facts: No NISTPQC submissions are homomorphic. Gauss never attacked these problems.

2020 Doulgerakis–Laarhoven–de Weger: "faster [sieving] methods for solving the shortest vector problem (SVP) on high-dimensional lattices".

"Conservative lower bound" on cost of BKZ was claimed in various submission documents in 2017 (round 1), 2019 (round 2). This "bound" was broken in 2018 for high-dimensional lattices. Apparently nobody noticed until I pointed this out in 2020.

Lattice marketing

"Strong security guarantees from *worst-case* hardness" of problems that "have been deeply studied by some of the great mathematicians and computer scientists going back at least to Gauss". Plus: fully homomorphic encryption.

Facts: No NISTPQC submissions are homomorphic. Gauss never attacked these problems. Our attacks keep getting better.

2020 Doulgerakis–Laarhoven–de Weger: "faster [sieving] methods for solving the shortest vector problem (SVP) on high-dimensional lattices".

"Conservative lower bound" on cost of BKZ was claimed in various submission documents in 2017 (round 1), 2019 (round 2). This "bound" was broken in 2018 for high-dimensional lattices. Apparently nobody noticed until I pointed this out in 2020.

Lattice marketing

"Strong security guarantees from *worst-case* hardness" of problems that "have been deeply studied by some of the great mathematicians and computer scientists going back at least to Gauss". Plus: fully homomorphic encryption.

Facts: No NISTPQC submissions are homomorphic. Gauss never attacked these problems. Our attacks keep getting better. The guarantees do not apply to any NISTPQC submissions.