LatticeHacks

Daniel J. Bernstein, Nadia Heninger, Tanja Lange

https://latticehacks.cr.yp.to

OPINION

Military wants Holy Grail of secure encryption

NATALIE WOLCHOVER SCIENCE 09.19.15 7:00 AM

THE TRICKY ENCRYPTION THAT COULD STUMP QUANTUM COMPUTERS

COMPLETELY BROKEN —

Millions of high-security crypto keys crippled by newly discovered flaw

What do all these headlines have in common?

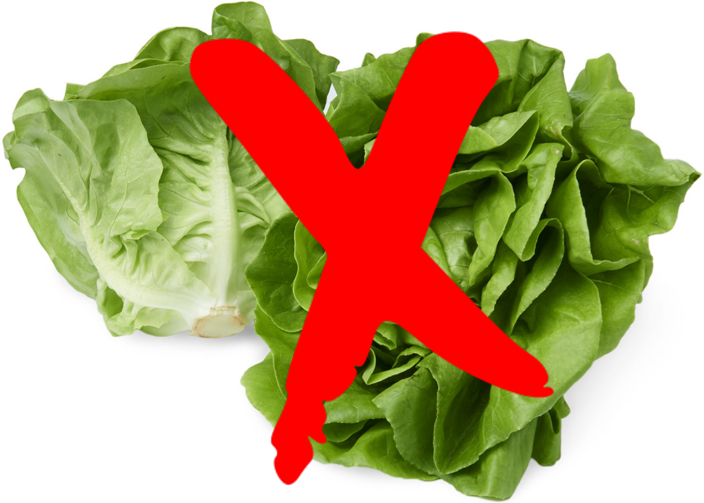
Lattices.

We can do two important things with lattices in cryptography:

- **We can break crypto.**
  - Knapsack cryptosystems
  - DSA nonce biases
  - Factoring RSA keys with bits known
  - Small RSA private exponents
  - Stereotyped messages with small RSA exponents
- **We can make crypto.**
  - Post-quantum cryptography
  - Fully homomorphic encryption
  - ...

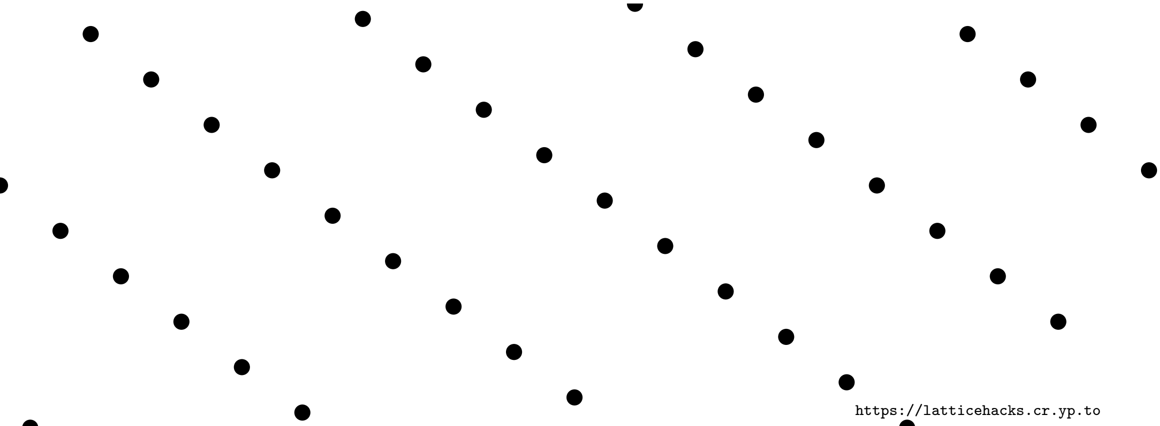
What is a lattice?

What is a lattice?



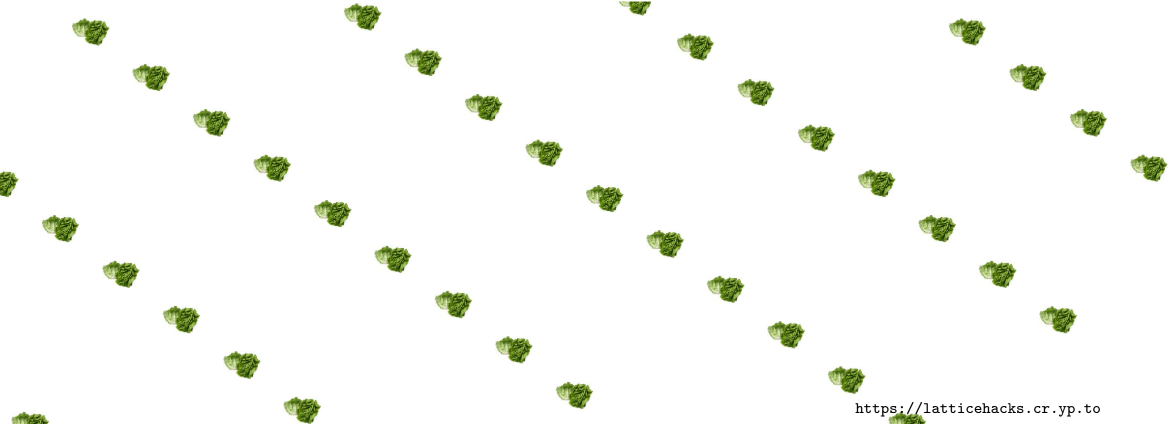
What is a lattice?

A lattice is a repeating grid of points in n-dimensional space.



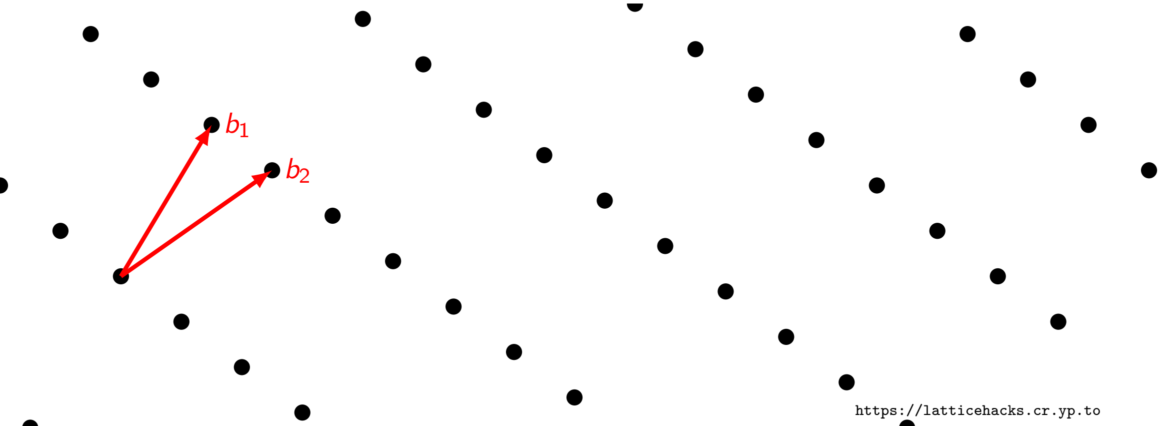
What is a lettuce lattice?

A lattice of lettuces is a repeating grid of lettuces in n-dimensional space.



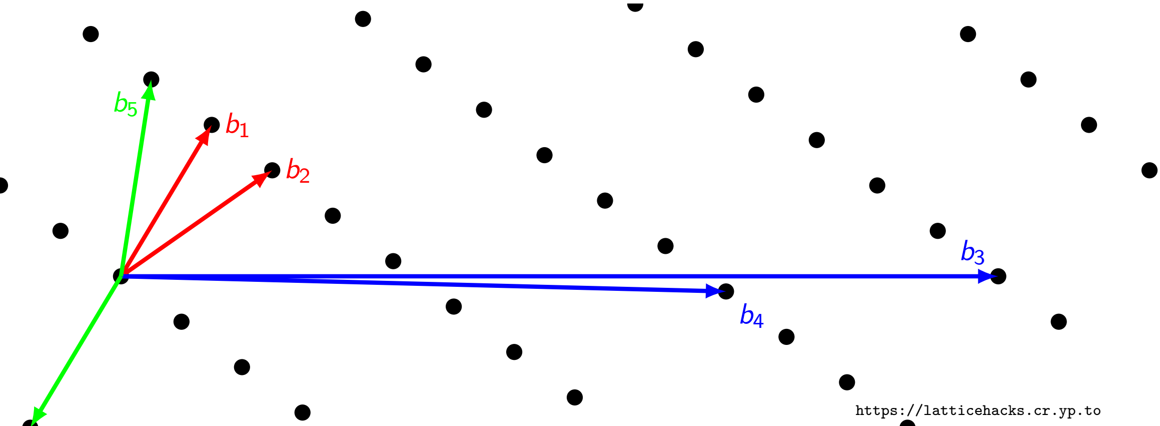
What is a lattice?

We can think of a lattice as being generated by integer multiples of some basis vectors.



What is a lattice?

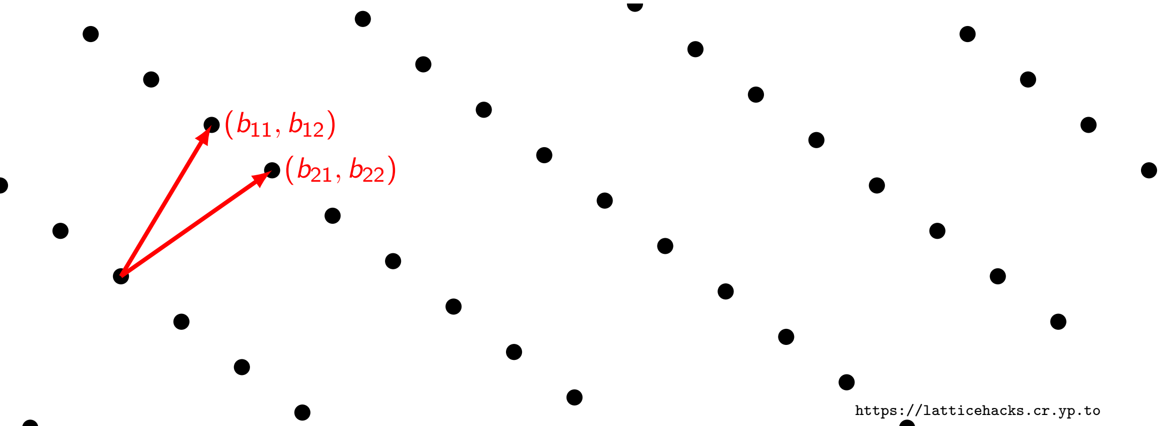
A lattice can have many different bases.



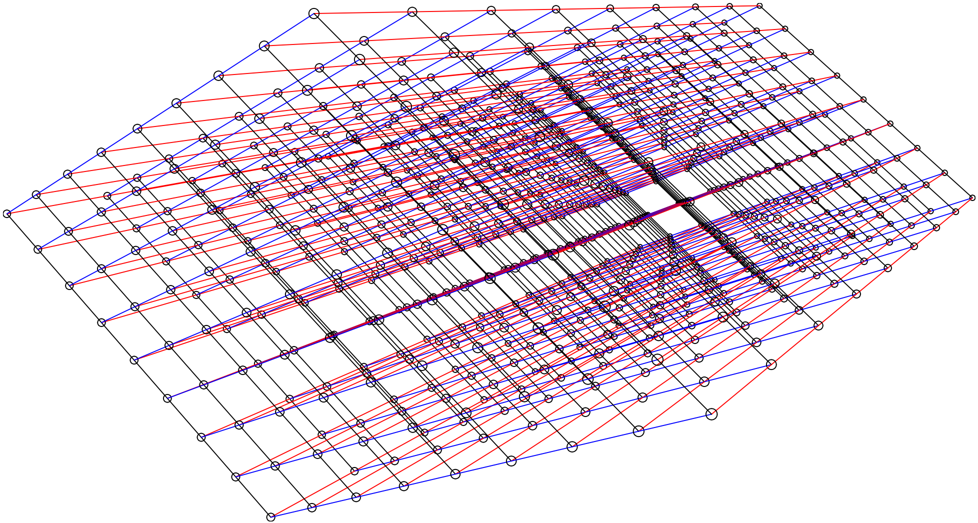
What is a lattice?

We can represent a lattice as a matrix of basis vector coefficients:

$$B = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}$$



Now that you've mastered two dimensions, here is a lattice in three.



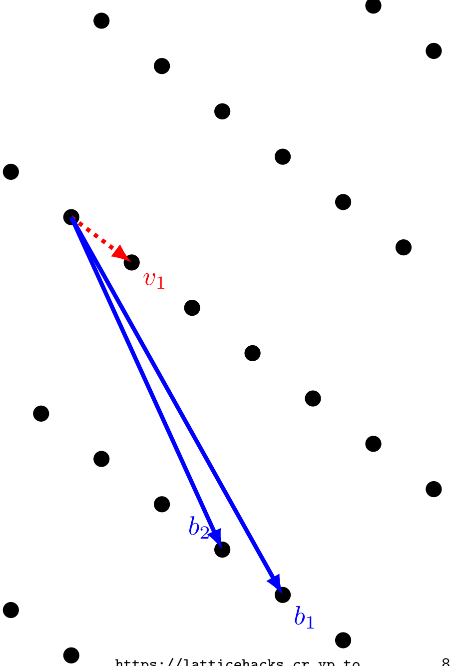
Do you really want us to keep going? Didn't think so.

The Shortest Vector Problem (SVP)

Shortest Vector Problem (SVP)

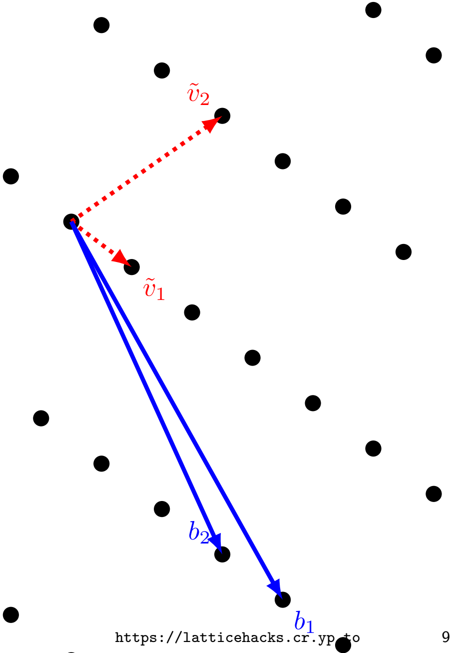
Given an arbitrary basis for L, find the shortest nonzero vector v_1 in L.

- Slow algorithm to compute exact solution. (Exponential time!)
- Fast algorithm to compute approximate solution. (Polynomial time!)



The LLL Algorithm

(Lenstra, Lenstra, Lovasz)



Input: A lattice basis B in n dimensions.

Output: A pretty short vector in the lattice.

Guarantee: Length $\leq 2^{n/2}|v_1|$

Using Sage

We wanted to give you working code examples. We're going to use Sage.

Sage is free open source mathematics software.
Download from `http://www.sagemath.org/`.

Sage is based on Python

```
sage: 2*3
6
```

Using Sage

We wanted to give you working code examples. We're going to use Sage.

Sage is free open source mathematics software.
Download from `http://www.sagemath.org/`.

Sage is based on Python, but there are a few differences:

^ is exponentiation, not xor

```
sage: 2^3
8
```

Using Sage

We wanted to give you working code examples. We're going to use Sage.

Sage is free open source mathematics software.
Download from `http://www.sagemath.org/`.

Sage is based on Python, but there are a few differences:

^ is exponentiation, not xor

```
sage: 2^3
8
```

It has lots of useful libraries:

```
sage: factor(x^2-9)
(x + 3)*(x - 3)
```

Using Sage

We wanted to give you working code examples. We're going to use Sage.

Sage is free open source mathematics software.
Download from `http://www.sagemath.org/`.

Sage is based on Python, but there are a few differences:

^ is exponentiation, not xor

```
sage: 2^3
8
```

It has lots of useful libraries:

```
sage: factor(x^2-9)
(x + 3)*(x - 3)
```

```
sage: (x^2-9).roots()
[(-3, 1), (3, 1)]
```

RSA Review

(This is pre-quantum, non-lattice-based crypto!)

```
p = random_prime(2^512)
q = random_prime(2^512)
```

RSA Review

(This is pre-quantum, non-lattice-based crypto!)

```
p = random_prime(2^512)
q = random_prime(2^512)
```

Public Key

```
N = p*q
e = 3
```

or 65537 or 35...

RSA Review

(This is pre-quantum, non-lattice-based crypto!)

```
p = random_prime(2^512)
q = random_prime(2^512)
```

Public Key

```
N = p*q
e = 3
```

or 65537 or 35...

```
message^e % N
```

Encryption

```
ciphertext = pow(message,e,N)
```

RSA Review

(This is pre-quantum, non-lattice-based crypto!)

```
p = random_prime(2^512)
q = random_prime(2^512)
```

Public Key

```
N = p*q
e = 3
```

or 65537 or 35...

```
message^e % N
```

Encryption

```
ciphertext = pow(message,e,N)
```

Warning: Please don't ever implement RSA like this.
Textbook RSA is insecure for many reasons.

Factoring is hard. Let's go shopping!

Factoring with Lattices

RSA is secure only if it is hard to factor. So how hard is factoring really?

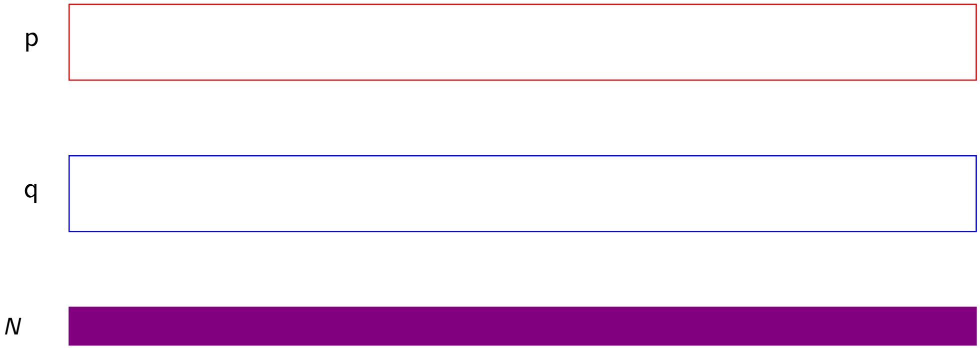
How hard is factoring?

Already-factored modulus: Trivial.

How hard is factoring?

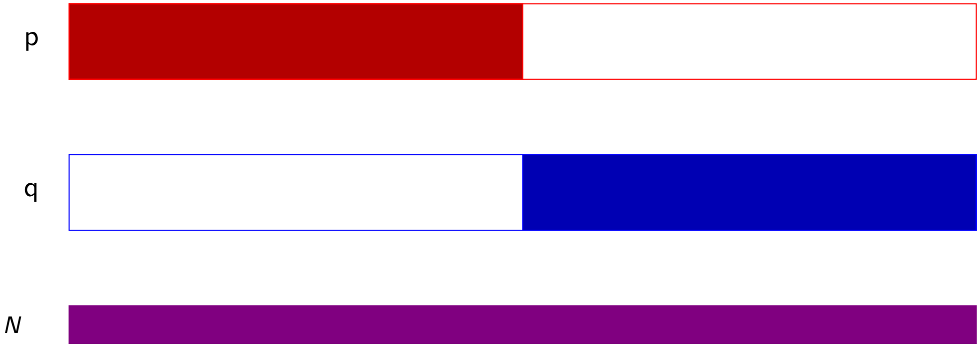
One factor known: Trivial. (Division)

How hard is factoring?



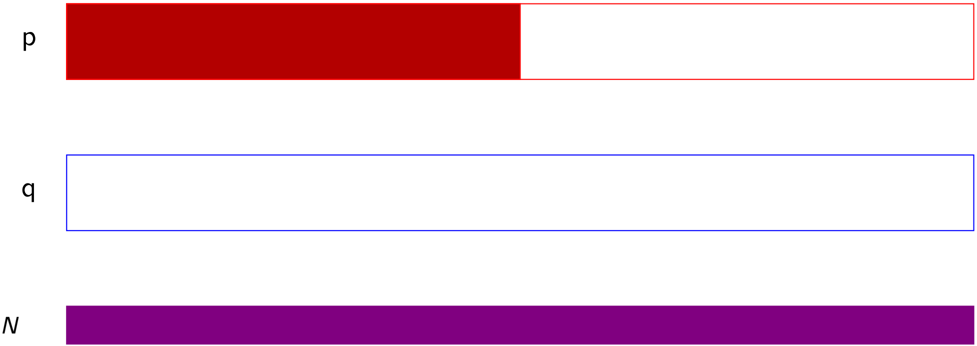
Neither factor known: Subexponential time.
(Nobody has factored the RSA-1024 challenge in public yet.)
See “FactHacks” (29C3) for more information.

Factoring with Partial Information

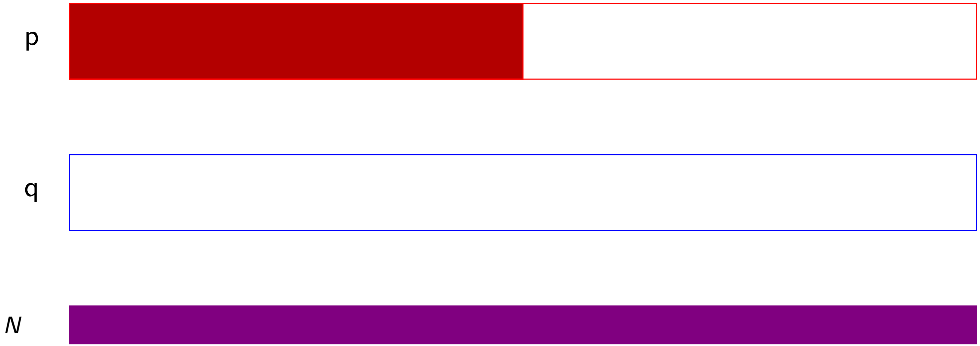


Trivial. (Division + fixing a few bits.)

Factoring with Partial Information



Factoring with Partial Information



Polynomial time. (With lattices!) [Coppersmith 96]

```
p = random_prime(2^512); q = random_prime(2^512)
N = p*q

a = p - (p % 2^86)
```

```
p = random_prime(2^512); q = random_prime(2^512)
N = p*q

a = p - (p % 2^86)
```

```
sage: hex(a)
'a9759e8c9fba8c0ec3e637d1e26e7b88befeb03ac199d1190
76e3294d16ffcaef629e2937a03592895b29b0ac708e79830
4330240bc000000000000000000000'
```

RSA key recovery from partial information.

```
p = random_prime(2^512); q = random_prime(2^512)
N = p*q

a = p - (p % 2^86)


X = 2^86
M = matrix([[X^2, 2*X*a, a^2], [0, X, a], [0, 0, N]])
B = M.LLL()
```

```
p = random_prime(2^512); q = random_prime(2^512)
N = p*q

a = p - (p % 2^86)


X = 2^86
M = matrix([[X^2, 2*X*a, a^2], [0, X, a], [0, 0, N]])
B = M.LLL()

Q = B[0][0]*x^2/X^2+B[0][1]*x/X+B[0][2]

sage: a+Q.roots(ring=ZZ)[0][0] == p
True
```

What is going on? Using lattices to factor.

Coppersmith/Howgrave-Graham

1. I wrote down a polynomial $f(x) = a + x$ that has a pretty small root

$$f(r) \equiv 0 \bmod p$$

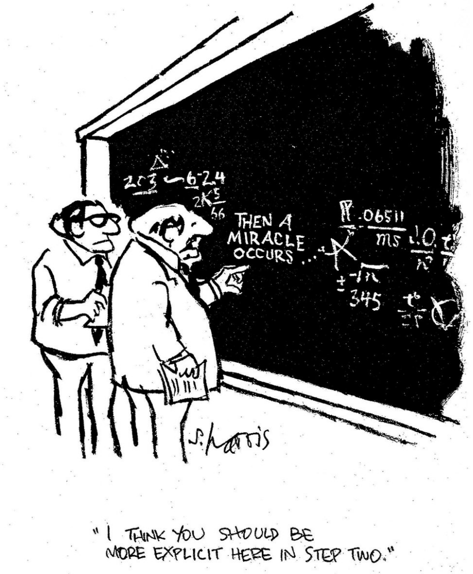
   I don't even know p, I only know N!

2. I constructed a lattice basis from coefficients of polynomials that vanish mod p:

$$\begin{bmatrix} 1 & 2a & a^2 \\ 0 & 1 & a \\ 0 & 0 & N \end{bmatrix}$$

3. I called the LLL algorithm to find a short vector.
4. My solution r was a root of the corresponding "small" polynomial.

At this point, we all kind of feel like this...



ROCA (Return of Coppersmith's Attack)

Nemec, Sys, Svenda, Klinec, and Matyas noticed that Infineon chips were generating RSA keys with primes of the form

$$p = kM + (g^a \bmod M)$$

where M, g are known and a is in a set small enough to brute force.

Exactly the same method as we just described works to factor these keys.
(With slightly bigger lattices.)

Oops!

Their paper covers tradeoffs between lattice dimension and search space.

When will quantum computers break RSA-2048?

Michele Mosca, November 2015: “I estimate a 1/7 chance of breaking RSA-2048 by 2026 and a 1/2 chance by 2031.”

When will quantum computers break RSA-2048?

Michele Mosca, November 2015: “I estimate a 1/7 chance of breaking RSA-2048 by 2026 and a 1/2 chance by 2031.”

Dario Gil, November 2017: “We have successfully built a 20-qubit and a 50-qubit quantum processor that works . . . We’re going to look back in history and say that [2016–2021] is when quantum computing emerged as a technology.”

When will quantum computers break RSA-2048?

Michele Mosca, November 2015: "I estimate a 1/7 chance of breaking RSA-2048 by 2026 and a 1/2 chance by 2031."

Dario Gil, November 2017: "We have successfully built a 20-qubit and a 50-qubit quantum processor that works . . . We're going to look back in history and say that [2016–2021] is when quantum computing emerged as a technology."

Shor's algorithm will break RSA-2048 using $\approx$4096 qubits
(and maybe it's possible to use even fewer qubits).

When will quantum computers break RSA-2048?

Michele Mosca, November 2015: "I estimate a 1/7 chance of breaking RSA-2048 by 2026 and a 1/2 chance by 2031."

Dario Gil, November 2017: "We have successfully built a 20-qubit and a 50-qubit quantum processor that works . . . We're going to look back in history and say that [2016–2021] is when quantum computing emerged as a technology."

Shor's algorithm will break RSA-2048 using ≈4096 qubits
(and maybe it's possible to use even fewer qubits).

Caveat: IBM has 50 **unreliable** qubits. Shor needs **reliable** qubits. Known error-correction methods use ≈1000 unreliable qubits (depending on error rate) to simulate one reliable qubit.

The NIST post-quantum competition

December 2016, after public feedback: NIST calls for submissions of post-quantum cryptosystems to standardize.

30 November 2017: NIST receives 82 submissions.

	Signatures	KEM/Encryption	Overall
Lattice-based	4	24	28
Code-based	5	19	24
Multi-variate	7	6	13
Hash-based	4		4
Other	3	10	13
Total	**23**	**59**	**82**

“Complete and proper” submissions

21 December 2017: NIST posts 69 submissions from 260 people.

BIG QUAKE. **BIKE**. **CFPKM**. **Classic McEliece**. **Compact LWE**. **CRYSTALS-DILITHIUM**. **CRYSTALS-KYBER**. **DAGS**. **Ding Key Exchange**. **DME**. **DRS**. **DualModeMS**. **Edon-K**. **EMBLEM and R.EMBLEM**. **FALCON**. **FrodoKEM**. **GeMSS**. **Giophantus**. **Gravity-SPHINCS**. **Guess Again**. **Gui**. **HILA5**. **HiMQ-3**. **HK17**. **HQC**. **KINDI**. **LAC**. **LAKE**. **LEDAkem**. **LEDApkc**. **Lepton**. **LIMA**. **Lizard**. **LOCKER**. **LOTUS**. **LUOV**. **McNie**. **Mersenne-756839**. **MQDSS**. **NewHope**. **NTRUEncrypt**. **pqNTRUSign**. **NTRU-HRSS-KEM**. **NTRU Prime**. **NTS-KEM**. **Odd Manhattan**. **OKCN/AKCN/CNKE**. **Ouroboros-R**. **Picnic**. **pqRSA encryption**. **pqRSA signature**. **pqsigRM**. **QC-MDPC KEM**. **qTESLA**. **RaCoSS**. **Rainbow**. **Ramstake**. **RankSign**. **RLCE-KEM**. **Round2**. **RQC**. **RVB**. **SABER**. **SIKE**. **SPHINCS+**. **SRTPI**. **Three Bears**. **Titanium**. **WalnutDSA**.

“Complete and proper” submissions

21 December 2017: NIST posts 69 submissions from 260 people.

BIG QUAKE. **BIKE**. **CFPKM**. **Classic McEliece**. **Compact LWE**. **CRYSTALS-DILITHIUM**. **CRYSTALS-KYBER**. **DAGS**. **Ding Key Exchange**. **DME**. **DRS**. **DualModeMS**. **Edon-K**. **EMBLEM and R.EMBLEM**. **FALCON**. **FrodoKEM**. **GeMSS**. **Giophantus**. **Gravity-SPHINCS**. **Guess Again**. **Gui**. **HILA5**. **HiMQ-3**. **HK17**. **HQC**. **KINDI**. **LAC**. **LAKE**. **LEDAkem**. **LEDApkc**. **Lepton**. **LIMA**. **Lizard**. **LOCKER**. **LOTUS**. **LUOV**. **McNie**. **Mersenne-756839**. **MQDSS**. **NewHope**. **NTRUEncrypt**. **pqNTRUSign**. **NTRU-HRSS-KEM**. **NTRU Prime**. **NTS-KEM**. **Odd Manhattan**. **OKCN/AKCN/CNKE**. **Ouroboros-R**. **Picnic**. **pqRSA encryption**. **pqRSA signature**. **pqsigRM**. **QC-MDPC KEM**. **qTESLA**. **RaCoSS**. **Rainbow**. **Ramstake**. **RankSign**. **RLCE-KEM**. **Round2**. **RQC**. **RVB**. **SABER**. **SIKE**. **SPHINCS+**. **SRTPI**. **Three Bears**. **Titanium**. **WalnutDSA**. Some attack scripts already posted.

Breaking RSA more with lattices

Coppersmith small exponent attacks

What's wrong with this RSA example?

```
message = Integer('squeamishossifrage',base=35)
N = random_prime(2^512)*random_prime(2^512)
c = message^3 % N
```

What's wrong with this RSA example?

```
message = Integer('squeamishossifrage',base=35)
N = random_prime(2^512)*random_prime(2^512)
c = message^3 % N

sage: Integer(c^(1/3)).str(base=35)
'squeamishossifrage'
```

What's wrong with this RSA example?

```
message = Integer('squeamishossifrage',base=35)
N = random_prime(2^512)*random_prime(2^512)
c = message^3 % N

sage: Integer(c^(1/3)).str(base=35)
'squeamishossifrage'
```

The message is small: $\text{message}^3 < N$, so the modular reduction does nothing.

```
N = random_prime(2^150)*random_prime(2^150)
message = Integer('thepasswordfortodayisswordfish',base=35)
c = message^3 % N
```

```
N = random_prime(2^150)*random_prime(2^150)
message = Integer('thepasswordfortodayisswordfish',base=35)
c = message^3 % N
```

```
sage: int(c^(1/3))==message
False
```

```
N = random_prime(2^150)*random_prime(2^150)
message = Integer('thepasswordfortodayisswordfish',base=35)
c = message^3 % N
```

This is a stereotyped message. We might be able to guess the format.

```
N = random_prime(2^150)*random_prime(2^150)
message = Integer('thepasswordfortodayisswordfish',base=35)
c = message^3 % N


a = Integer('thepasswordfortodayis000000000',base=35)
```

```
N = random_prime(2^150)*random_prime(2^150)
message = Integer('thepasswordfortodayisswordfish',base=35)
c = message^3 % N
```

```
a = Integer('thepasswordfortodayis000000000',base=35)

X = Integer('xxxxxxxxx',base=35)
M = matrix([[X^3, 3*X^2*a, 3*X*a^2, a^3-c],
            [0,N*X^2,0,0],[0,0,N*X,0],[0,0,0,N]])
```

```
N = random_prime(2^150)*random_prime(2^150)
message = Integer('thepasswordfortodayisswordfish',base=35)
c = message^3 % N
```

```
a = Integer('thepasswordfortodayis000000000',base=35)

X = Integer('xxxxxxxxx',base=35)
M = matrix([[X^3, 3*X^2*a, 3*X*a^2, a^3-c],
            [0,N*X^2,0,0],[0,0,N*X,0],[0,0,0,N]])

B = M.LLL()
Q = B[0][0]*x^3/X^3+B[0][1]*x^2/X^2+B[0][2]*x/X+B[0][3]
```

```
N = random_prime(2^150)*random_prime(2^150)
message = Integer('thepasswordfortodayisswordfish',base=35)
c = message^3 % N



a = Integer('thepasswordfortodayis000000000',base=35)

X = Integer('xxxxxxxxx',base=35)
M = matrix([[X^3, 3*X^2*a, 3*X*a^2, a^3-c],
            [0,N*X^2,0,0],[0,0,N*X,0],[0,0,0,N]])

B = M.LLL()
Q = B[0][0]*x^3/X^3+B[0][1]*x^2/X^2+B[0][2]*x/X+B[0][3]

sage: Q.roots(ring=ZZ)[0][0].str(base=35)
'swordfish'
```

What is going on? Coppersmith's method.

1. I wrote down a polynomial $f(x) = (a + x)^3 - c$ that has a pretty small root

$$f(\texttt{swordfish}) \equiv 0 \bmod N$$

2. I construct a lattice of polynomials that vanish mod N:

$$\begin{bmatrix} 1 & 3a & 3a^2 & a^3 - c \\ 0 & N & 0 & 0 \\ 0 & 0 & N & 0 \\ 0 & 0 & 0 & N \end{bmatrix}$$

3. I called the LLL algorithm to find a short vector.
4. My solution `swordfish` was a root of the corresponding "small" polynomial.

Countermeasures against Coppersmith padding attacks

- Don't use RSA.
- If you must use RSA, use a proper padding scheme.
- If you must use RSA, don't use $e < 65537$.

NTRU history

- Introduced by Hoffstein, Pipher, and Silverman in 1998.
- Presented as an alternative to RSA and ECC;
  higher speed but larger key size & ciphertext.
- Good amount of research into attacks during last 20 years.
  - NTRU signature scheme had a bit of a bumpy ride.

Silverman, Jan 2015 (NTRU and Lattice-Based Crypto)

Lattice-Based Digital Signatures 27

A Signature Scheme Disaster

“Luckily the crypto community was pretty forgiving about this mishap.”

NTRU history

- Introduced by Hoffstein, Pipher, and Silverman in 1998.
- Presented as an alternative to RSA and ECC; higher speed but larger key size & ciphertext.
- Good amount of research into attacks during last 20 years.
  - NTRU signature scheme had a bit of a bumpy ride.

NTRU history

- Introduced by Hoffstein, Pipher, and Silverman in 1998.
- Presented as an alternative to RSA and ECC;
  higher speed but larger key size & ciphertext.
- Good amount of research into attacks during last 20 years.
  - NTRU signature scheme had a bit of a bumpy ride.
  - NTRU encryption held up after first change of parameters.
- Far less research into efficient implementation and secure usage

NTRU history

- Introduced by Hoffstein, Pipher, and Silverman in 1998.
- Presented as an alternative to RSA and ECC;
  higher speed but larger key size & ciphertext.
- Good amount of research into attacks during last 20 years.
  - NTRU signature scheme had a bit of a bumpy ride.
  - NTRU encryption held up after first change of parameters.
- Far less research into efficient implementation and secure usage
  – why invest research effort into patented scheme...
- NTRU patent finally expired now.

NTRU operations

NTRU works with polynomials over the integers of degree less than some system parameter $250 < n < 2500$.

$$R = \{a_0 + a_1x + a_2x^2 + \cdots + a_{n-1}x^{n-1} | a_i \in \mathbb{Z}\}.$$

We add component wise

$$(a_0 + a_1x + a_2x^2 + \cdots + a_{n-1}x^{n-1}) + (b_0 + b_1x + b_2x^2 + \cdots + b_{n-1}x^{n-1}) =$$
$$(a_0 + b_0) + (a_1 + b_1)x + (a_2 + b_2)x^2 + \cdots + (a_{n-1} + b_{n-1})x^{n-1}$$

We also define some form of multiplication

$$(a_0 + a_1x + a_2x^2 + \cdots + a_{n-1}x^{n-1}) * (b_0 + b_1x + b_2x^2 + \cdots + b_{n-1}x^{n-1}) =$$
$$(a_0b_0 + a_1b_{n-1} + a_2b_{n-2} + \cdots + a_{n-1}b_1) + (a_0b_1 + a_1b_0 + a_2b_{n-1} + \cdots + a_{n-1}b_2)x +$$
$$\cdots + (a_0b_{n-1} + a_1b_{n-2} + a_2b_{n-3} + \cdots + a_{n-1}b_0)x^{n-1}$$

which stays within R. Operation $*$ called *cyclic convolution*.

NTRU operations (same slide in math)

NTRU works with polynomials over the integers of degree less than some system parameter $250 < n < 2500$.

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

We add component wise

$$\sum_{i=0}^{n-1} a_i x^i + \sum_{i=0}^{n-1} b_i x^i = \sum_{i=0}^{n-1} (a_i + b_i) x^i.$$

We also define some form of multiplication

$$\sum_{i=0}^{n-1} a_i x^i * \sum_{i=0}^{n-1} b_i x^i = \sum_{i=0}^{n-1} a_i x^i \cdot \sum_{i=0}^{n-1} b_i x^i \bmod (x^n - 1).$$

Regular multiplication in R.

```
sage: Zx.<x> = ZZ[]
sage:
```

```
sage: Zx.<x> = ZZ[]
sage: f = Zx([3,1,4])
sage:
```

```
sage: Zx.<x> = ZZ[]
sage: f = Zx([3,1,4])
sage: f
4*x^2 + x + 3
sage:
```

```
sage: Zx.<x> = ZZ[]
sage: f = Zx([3,1,4])
sage: f
4*x^2 + x + 3
sage: f*x
4*x^3 + x^2 + 3*x
sage:
```

```
sage: Zx.<x> = ZZ[]
sage: f = Zx([3,1,4])
sage: f
4*x^2 + x + 3
sage: f*x
4*x^3 + x^2 + 3*x
sage: g = Zx([2,7,1])
sage: g
x^2 + 7*x + 2
sage:
```

```
sage: Zx.<x> = ZZ[]
sage: f = Zx([3,1,4])
sage: f
4*x^2 + x + 3
sage: f*x
4*x^3 + x^2 + 3*x
sage: g = Zx([2,7,1])
sage: g
x^2 + 7*x + 2
sage: f*g
4*x^4 + 29*x^3 + 18*x^2 + 23*x + 6
sage:
```

```
sage: def convolution(f,g):
....:     return (f * g) % (x^n-1)
....:
sage: n = 3
sage:
```

```
sage: def convolution(f,g):
....:     return (f * g) % (x^n-1)
....:
sage: n = 3
sage: f
4*x^2 + x + 3
sage:
```

```
sage: def convolution(f,g):
....:     return (f * g) % (x^n-1)
....:
sage: n = 3
sage: f
4*x^2 + x + 3
sage: convolution(f,x)
x^2 + 3*x + 4
sage:
```

```
sage: def convolution(f,g):
....:     return (f * g) % (x^n-1)
....:
sage: n = 3
sage: f
4*x^2 + x + 3
sage: convolution(f,x)
x^2 + 3*x + 4
sage: convolution(f,x^2)
3*x^2 + 4*x + 1
sage:
```

```
sage: def convolution(f,g):
....:     return (f * g) % (x^n-1)
....:
sage: n = 3
sage: f
4*x^2 + x + 3
sage: convolution(f,x)
x^2 + 3*x + 4
sage: convolution(f,x^2)
3*x^2 + 4*x + 1
sage: f*g
4*x^4 + 29*x^3 + 18*x^2 + 23*x + 6
sage: convolution(f,g)
18*x^2 + 27*x + 35
sage:
```

Alternative: Use `R = Zx.quotient(x^n-1)` to create $R = \mathbb{Z}[x]/(x^n - 1)$.

More NTRU parameters

- NTRU specifies integer n (as above).
- Integer q, typically a power of 2.
  In any case, q must not be multiple of 3.
- Some computations reduce each polynomial coefficient modulo q.
  We use $\bmod q$ on the polynomial.
- Same for modulo 3.

More NTRU parameters

- NTRU specifies integer n (as above).
- Integer q, typically a power of 2.
  In any case, q must not be multiple of 3.
- Some computations reduce each polynomial coefficient modulo q.
  We use $\bmod q$ on the polynomial.
- Same for modulo 3.
- Pick $f, g \in R$ with coefficients in $\{-1, 0, 1\}$, almost all coefficients are zero (small fixed number are nonzero).
- Public key $h \in R$ with $h * f = 3g \bmod q$.
  If no such h exists, start over with new f.
- In math notation $h = 3g/f \bmod q$ in $\mathbb{Z}[x]/(x^n - 1)$.
- Private key f and f_3 with $f * f_3 = 1 \bmod 3$.

NTRU encryption (schoolbook version)

- Public key $h \in R$ with $h * f = 3g \bmod q$.
- Encryption of message $m \in R$, coefficients in $\{-1, 0, 1\}$:
  - Pick random $r \in R$, with coefficients in $\{-1, 0, 1\}$, almost all coefficients are zero (small fixed number are nonzero).
  - Compute
  $$c = r * h + m \bmod q.$$
  - Send ciphertext c.
- Decryption of $c \in R_q$:
  - Compute
  $$a = f * c = f * (r * h + m) = r * 3g + f * m \bmod q$$
  using $h * f = 3g \bmod q$.
  - Move all coefficients of a to $[-q/2, q/2]$.
  - If everything is small enough then a equals $r * 3g + f * m$ in R and
  $$m = a * f_3 \bmod 3,$$
  using $f * f_3 = 1 \bmod 3$.

```
sage: n = 7
sage: d = 5
sage: q = 256
sage: f = randomdpoly()
sage: f
-x^4 + x^3 + x^2 - x + 1
sage:
```

```
sage: n = 7
sage: d = 5
sage: q = 256
sage: f = randomdpoly()
sage: f
-x^4 + x^3 + x^2 - x + 1
sage: f3 = invertmodprime(f,3)
sage: f3
x^6 + 2*x^4 + x
sage:
```

```
sage: n = 7
sage: d = 5
sage: q = 256
sage: f = randomdpoly()
sage: f
-x^4 + x^3 + x^2 - x + 1
sage: f3 = invertmodprime(f,3)
sage: f3
x^6 + 2*x^4 + x
sage: convolution(f,f3)
3*x^6 - 3*x^5 + 3*x^4 + 1
sage:
```

```
sage: fq = invertmodpowerof2(f,q)
sage: convolution(f,fq)
-256*x^6 + 256*x^4 - 256*x^2 + 257
sage:
```

```
sage: fq = invertmodpowerof2(f,q)
sage: convolution(f,fq)
-256*x^6 + 256*x^4 - 256*x^2 + 257
sage: g = randomdpoly()
sage:
```

```
sage: fq = invertmodpowerof2(f,q)
sage: convolution(f,fq)
-256*x^6 + 256*x^4 - 256*x^2 + 257
sage: g = randomdpoly()
sage: h = (3 * convolution(fq,g)) % q
sage: h
174*x^6 + 118*x^5 + 162*x^4 + 108*x^3 - 186*x^2 + 134*x + 5
sage:
```

```
sage: fq = invertmodpowerof2(f,q)
sage: convolution(f,fq)
-256*x^6 + 256*x^4 - 256*x^2 + 257
sage: g = randomdpoly()
sage: h = (3 * convolution(fq,g)) % q
sage: h
174*x^6 + 118*x^5 + 162*x^4 + 108*x^3 - 186*x^2 + 134*x + 5
sage: h = balancedmod(3 * convolution(fq,g),q)
sage: h
-82*x^6 + 118*x^5 - 94*x^4 + 108*x^3 + 70*x^2 - 122*x + 5
sage:
```

```
sage: fq = invertmodpowerof2(f,q)
sage: convolution(f,fq)
-256*x^6 + 256*x^4 - 256*x^2 + 257
sage: g = randomdpoly()
sage: h = (3 * convolution(fq,g)) % q
sage: h
174*x^6 + 118*x^5 + 162*x^4 + 108*x^3 - 186*x^2 + 134*x + 5
sage: h = balancedmod(3 * convolution(fq,g),q)
sage: h
-82*x^6 + 118*x^5 - 94*x^4 + 108*x^3 + 70*x^2 - 122*x + 5
sage: balancedmod(convolution(f,h),q)
-3*x^4 + 3*x^3 - 3*x^2 + 3*x + 3
sage: 3 * g
-3*x^4 + 3*x^3 - 3*x^2 + 3*x + 3
sage:
```

```
sage: m = randommessage()
sage: m
-x^6 - x^4 + x^2 + 1
sage:
```

```
sage: m = randommessage()
sage: m
-x^6 - x^4 + x^2 + 1
sage: r = randomdpoly()
sage: c = balancedmod(convolution(h,r) + m,q)
sage: c
-66*x^6 + 37*x^5 + 115*x^4 - 15*x^3 - 6*x^2 - 89*x + 27
sage:
```

```
sage: m = randommessage()
sage: m
-x^6 - x^4 + x^2 + 1
sage: r = randomdpoly()
sage: c = balancedmod(convolution(h,r) + m,q)
sage: c
-66*x^6 + 37*x^5 + 115*x^4 - 15*x^3 - 6*x^2 - 89*x + 27
sage: a = balancedmod(convolution(f,c),q)
sage: a
3*x^6 - 10*x^5 + 8*x^4 - 5*x^3 + 7*x^2 - 4*x + 4
sage: 3*convolution(g,r) + convolution(f,m)
3*x^6 - 10*x^5 + 8*x^4 - 5*x^3 + 7*x^2 - 4*x + 4
sage:
```

```
sage: m = randommessage()
sage: m
-x^6 - x^4 + x^2 + 1
sage: r = randomdpoly()
sage: c = balancedmod(convolution(h,r) + m,q)
sage: c
-66*x^6 + 37*x^5 + 115*x^4 - 15*x^3 - 6*x^2 - 89*x + 27
sage: a = balancedmod(convolution(f,c),q)
sage: a
3*x^6 - 10*x^5 + 8*x^4 - 5*x^3 + 7*x^2 - 4*x + 4
sage: 3*convolution(g,r) + convolution(f,m)
3*x^6 - 10*x^5 + 8*x^4 - 5*x^3 + 7*x^2 - 4*x + 4
sage: balancedmod(convolution(a,f3),3)
-x^6 - x^4 + x^2 + 1
sage:
```

Didn't your mom tell you not to mix mod p and mod q?

Decryption failures

Decryption of c wants that

$$a = f * c = r * 3g + f * m \bmod q,$$

has the integer factor 3 in the first part, even after reduction modulo q.
This works if the computed a equals $r * 3g + f * m$ in R, i.e., without reduction modulo q.
This works if everything is small enough compared to q.
For d non-zero coefficients in f and r the maximum coefficient of $r * 3g + f * m$ is

$$3d + d,$$

and typically much smaller.
Can choose q such that $q/2 > 4d$ – or hope for the best and expect coefficients not to collude.

Breaking NTRU with lattices

NTRU – translation to lattices

- Public key h with $h * f = 3g \bmod q$.
- Can see this as lattice with basis matrix $$B = \begin{pmatrix} q\,I_n & 0 \\ H & I_n \end{pmatrix},$$ where H corresponds to multiplication $*$ by $h/3$ in R.
- So

$$((1, 0, 0, \ldots, 0), (3, 0, 0, \ldots, 0)) \begin{pmatrix} q\,I_n & 0 \\ H & I_n \end{pmatrix}$$
$$= ((q, 0, 0, \ldots, 0) + (h_0, h_1, \ldots, h_{n-1}), (3, 0, 0, \ldots, 0)).$$

NTRU – translation to lattices

- Public key h with $h * f = 3g \bmod q$.
- Can see this as lattice with basis matrix
$$B = \begin{pmatrix} q\, I_n & 0 \\ H & I_n \end{pmatrix},$$
where H corresponds to multiplication $*$ by $h/3$ in R.
- So
$$((1,0,0,\ldots,0),(3,0,0,\ldots,0)) \begin{pmatrix} q\, I_n & 0 \\ H & I_n \end{pmatrix}$$
$$= ((q,0,0,\ldots,0) + (h_0, h_1, \ldots, h_{n-1}), (3,0,0,\ldots,0)).$$
- (g, f) is a short vector in the lattice as result of
$$(k, f)B = (kq + f * h/3, f) = (g, f)$$
for some $k \in R$ (from $h * f = 3g \bmod q$, i.e., $h * f = 3g + 3kq$).

```
sage: Integers(q)(1/3)
171
sage:
```

```
sage: Integers(q)(1/3)
171
sage: h3 = (171*h)%q
sage: h3
58*x^6 + 210*x^5 + 54*x^4 + 36*x^3 + 194*x^2 + 130*x + 87
sage:
```

```
sage: Integers(q)(1/3)
171
sage: h3 = (171*h)%q
sage: h3
58*x^6 + 210*x^5 + 54*x^4 + 36*x^3 + 194*x^2 + 130*x + 87
sage: convolution(h3,x)
210*x^6 + 54*x^5 + 36*x^4 + 194*x^3 + 130*x^2 + 87*x + 58
sage:
```

```
sage: Integers(q)(1/3)
171
sage: h3 = (171*h)%q
sage: h3
58*x^6 + 210*x^5 + 54*x^4 + 36*x^3 + 194*x^2 + 130*x + 87
sage: convolution(h3,x)
210*x^6 + 54*x^5 + 36*x^4 + 194*x^3 + 130*x^2 + 87*x + 58
sage: convolution(h3,x^2)
54*x^6 + 36*x^5 + 194*x^4 + 130*x^3 + 87*x^2 + 58*x + 210
sage: convolution(h3,x^3)
36*x^6 + 194*x^5 + 130*x^4 + 87*x^3 + 58*x^2 + 210*x + 54
sage: convolution(h3,x^4)
194*x^6 + 130*x^5 + 87*x^4 + 58*x^3 + 210*x^2 + 54*x + 36
sage: convolution(h3,x^5)
130*x^6 + 87*x^5 + 58*x^4 + 210*x^3 + 54*x^2 + 36*x + 194
sage: convolution(h3,x^6)
87*x^6 + 58*x^5 + 210*x^4 + 54*x^3 + 36*x^2 + 194*x + 130
sage:
```

```
sage: M = matrix(2*n)
sage: for i in range(n): M[i,i] = q
sage: for i in range(n,2*n): M[i,i] = 1
sage: for i in range(n):
....:     for j in range(n):
....:         M[i+n,j] = convolution(h3,x^i)[j]
....:
sage:
```

```
sage: M
[256   0   0   0   0   0   0   0   0   0   0   0   0   0]
[  0 256   0   0   0   0   0   0   0   0   0   0   0   0]
[  0   0 256   0   0   0   0   0   0   0   0   0   0   0]
[  0   0   0 256   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0 256   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0 256   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0 256   0   0   0   0   0   0   0]
[ 87 130 194  36  54 210  58   1   0   0   0   0   0   0]
[ 58  87 130 194  36  54 210   0   1   0   0   0   0   0]
[210  58  87 130 194  36  54   0   0   1   0   0   0   0]
[ 54 210  58  87 130 194  36   0   0   0   1   0   0   0]
[ 36  54 210  58  87 130 194   0   0   0   0   1   0   0]
[194  36  54 210  58  87 130   0   0   0   0   0   1   0]
[130 194  36  54 210  58  87   0   0   0   0   0   0   1]
sage:
```

```
sage: M.LLL()
[ -1  -1   1  -1   1   0   0  -1   1  -1  -1   1   0   0]
[  0  -1  -1   1  -1   1   0   0  -1   1  -1  -1   1   0]
[  1  -1   1  -1   0   0   1  -1   1   1  -1   0   0   1]
[ -1   1  -1   0   0   1   1   1   1  -1   0   0   1  -1]
[  1  -1   0   0   1   1  -1   1  -1   0   0   1  -1   1]
[  1   1   1   1   1   1   1   1   1   1   1   1   1   1]
[  0   0   1   1  -1   1  -1   0   0   1  -1   1   1  -1]
[ 39 -28  19  12  11 -48  -4  47   6 -31 -20 -19  36 -18]
[ -5 -34 -14  -3   9 -39 -43  47  54  22   1 -17  19   1]
[  4 -39  28 -19 -12 -11  48  18 -47  -6  31  20  19 -36]
[  9 -40 -43  -5 -32 -13  -1 -17  20   1  47  54  23   3]
[ -1   9 -40 -43  -5 -32 -13   3 -17  20   1  47  54  23]
[ 14   3  -9  40  43   4  32 -22  -3  17 -18  -1 -48 -54]
[ 28 -19 -12 -11  48   4 -39  -6  31  20  19 -36  18 -47]
sage:
```

```
sage: M.LLL()[0][n:]
(-1, 1, -1, -1, 1, 0, 0)
sage:
```

```
sage: M.LLL()[0][n:]
(-1, 1, -1, -1, 1, 0, 0)
sage: Zx(list(_))
x^4 - x^3 - x^2 + x - 1
sage:
```

```
sage: M.LLL()[0][n:]
(-1, 1, -1, -1, 1, 0, 0)
sage: Zx(list(_))
x^4 - x^3 - x^2 + x - 1
sage: f
-x^4 + x^3 + x^2 - x + 1
sage:
```

Conclusion: This attack breaks NTRU with $n = 7$, $d = 5$, $q = 256$.

```
sage: M.LLL()[0][n:]
(-1, 1, -1, -1, 1, 0, 0)
sage: Zx(list(_))
x^4 - x^3 - x^2 + x - 1
sage: f
-x^4 + x^3 + x^2 - x + 1
sage:
```

Conclusion: This attack breaks NTRU with $n = 7$, $d = 5$, $q = 256$.

The secrets were too small for security anyway.
Scale up: NTRU with $n = 150$, $d = 101$, $q = 2^{32}$. Now $>2^{200}$ choices of f.

```
sage: M.LLL()[0][n:]
(-1, 1, -1, -1, 1, 0, 0)
sage: Zx(list(_))
x^4 - x^3 - x^2 + x - 1
sage: f
-x^4 + x^3 + x^2 - x + 1
sage:
```

Conclusion: This attack breaks NTRU with $n = 7$, $d = 5$, $q = 256$.

The secrets were too small for security anyway.
Scale up: NTRU with $n = 150$, $d = 101$, $q = 2^{32}$. Now $>2^{200}$ choices of f.

Try running same lattice attack against a random public key.
Instead of f, attacker finds the following polynomial . . .

```
x^108 + x^103 - 2*x^102 - 2*x^101 - x^100 - x^99 - 2*x^98 - 3*x^97 +
4*x^96 - x^95 - 5*x^94 - 3*x^93 + 8*x^92 + 5*x^91 + 10*x^90 - 2*x^89 +
5*x^88 - 7*x^87 - x^86 + 6*x^85 - 11*x^84 + 4*x^83 + 14*x^82 - 13*x^81 +
2*x^80 + 3*x^79 - x^78 + 3*x^77 - 5*x^76 + 7*x^74 - 8*x^73 - 23*x^72 -
15*x^71 - 23*x^70 + 33*x^69 - 11*x^68 - 22*x^67 - 20*x^66 + 17*x^65 -
24*x^64 - 9*x^63 - 21*x^62 + 27*x^61 - 22*x^59 - 15*x^58 - 2*x^57 - x^56
+ x^55 + x^54 + 6*x^53 + 3*x^52 - 8*x^51 + x^50 - 12*x^49 + 15*x^48 -
5*x^45 + 13*x^44 - 12*x^43 + 9*x^42 + 23*x^41 - 45*x^40 + 25*x^39 -
17*x^38 + 18*x^37 + 2*x^36 - 15*x^35 + 5*x^34 + 9*x^33 - 31*x^32 +
10*x^31 + 16*x^30 - 38*x^29 + 36*x^28 + 5*x^27 + 3*x^26 - 15*x^25 +
18*x^24 + 17*x^23 - 6*x^22 + 18*x^21 - 9*x^20 + 5*x^19 - 14*x^18 +
17*x^17 - 17*x^16 + 20*x^15 + 26*x^14 - 16*x^13 - x^12 + 21*x^11 +
25*x^10 - 21*x^9 + 8*x^8 + 23*x^7 + 8*x^6 - 38*x^5 + 14*x^4 - 11*x^3 +
10*x^2 - 10*x + 4
```

This isn't f, but **it is small enough to successfully decrypt messages**.

```
x^108 + x^103 - 2*x^102 - 2*x^101 - x^100 - x^99 - 2*x^98 - 3*x^97 +
4*x^96 - x^95 - 5*x^94 - 3*x^93 + 8*x^92 + 5*x^91 + 10*x^90 - 2*x^89 +
5*x^88 - 7*x^87 - x^86 + 6*x^85 - 11*x^84 + 4*x^83 + 14*x^82 - 13*x^81 +
2*x^80 + 3*x^79 - x^78 + 3*x^77 - 5*x^76 + 7*x^74 - 8*x^73 - 23*x^72 -
15*x^71 - 23*x^70 + 33*x^69 - 11*x^68 - 22*x^67 - 20*x^66 + 17*x^65 -
24*x^64 - 9*x^63 - 21*x^62 + 27*x^61 - 22*x^59 - 15*x^58 - 2*x^57 - x^56
+ x^55 + x^54 + 6*x^53 + 3*x^52 - 8*x^51 + x^50 - 12*x^49 + 15*x^48 -
5*x^45 + 13*x^44 - 12*x^43 + 9*x^42 + 23*x^41 - 45*x^40 + 25*x^39 -
17*x^38 + 18*x^37 + 2*x^36 - 15*x^35 + 5*x^34 + 9*x^33 - 31*x^32 +
10*x^31 + 16*x^30 - 38*x^29 + 36*x^28 + 5*x^27 + 3*x^26 - 15*x^25 +
18*x^24 + 17*x^23 - 6*x^22 + 18*x^21 - 9*x^20 + 5*x^19 - 14*x^18 +
17*x^17 - 17*x^16 + 20*x^15 + 26*x^14 - 16*x^13 - x^12 + 21*x^11 +
25*x^10 - 21*x^9 + 8*x^8 + 23*x^7 + 8*x^6 - 38*x^5 + 14*x^4 - 11*x^3 +
10*x^2 - 10*x + 4
```

This isn't f, but **it is small enough to successfully decrypt messages**.
For better security, **decrease** q, and **increase** n.

Attacks on NTRU

Mathematical attacks

- LLL works if q is too large compared to n and d; accept some decryption failures to avoid LLL?
- More powerful lattice-basis reduction (see next part), choose large enough n to avoid.
- Meet-in-the-middle attack.
- Hybrid attack, combining both.
- Attacks using the structure of R, incl. quantum attacks.

Crypto attacks (certainly don't use schoolbook version; best use some KEM)

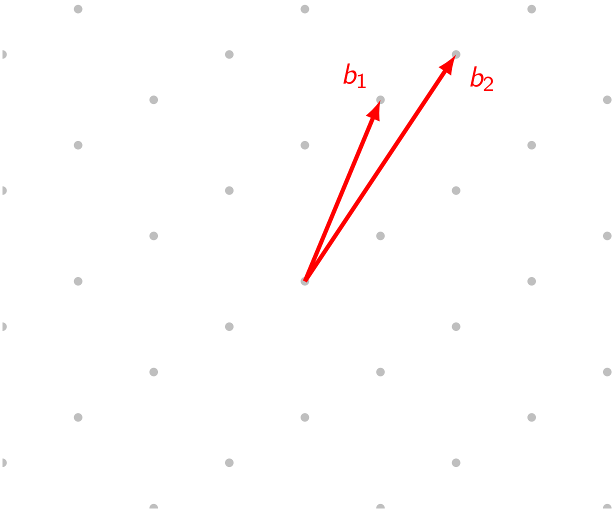
- Evaluation-at-1 attack;
- Chosen-ciphertext attacks;
- Decryption-failure attacks;
- Complicated padding systems.

LLL is just the beginning

Many more attacks

- Block Korkine-Zolotarev (BKZ)
  - Assumes we can solve SVP exactly in small dimension m.
  - Projects m vectors to smaller space,
    solves SVP there, lifts back.
  - Chains these in a way and interleaves with LLL
    to obtain short basis.
  - Quality depends heavily on m.
- Enumeration algorithms
  - Search for absolutely shortest, with some smart ideas.
  - Finds shortest vector.
  - Can balance time and quality of basis by stopping early/pruning.
- Sieving algorithms
  - Asymptotically faster than enumeration;
    better than BKZ.
  - Needs more space.
  - No guarantee that short vector found is shortest.
  - Balances time and quality of basis.

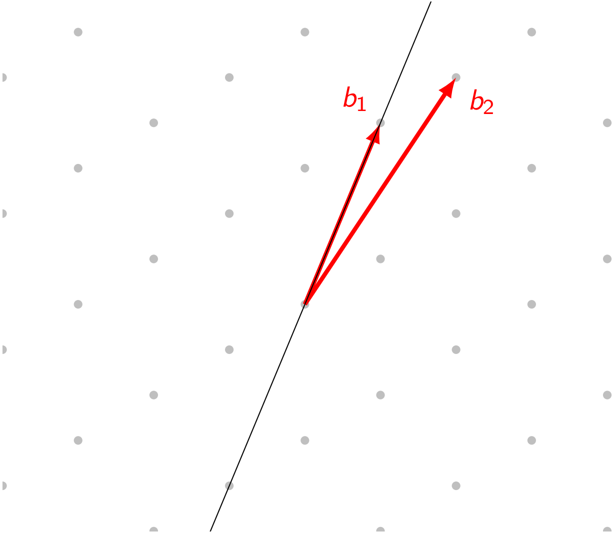
Enumeration



Visualization idea: Thijs Laarhoven.

Enumeration

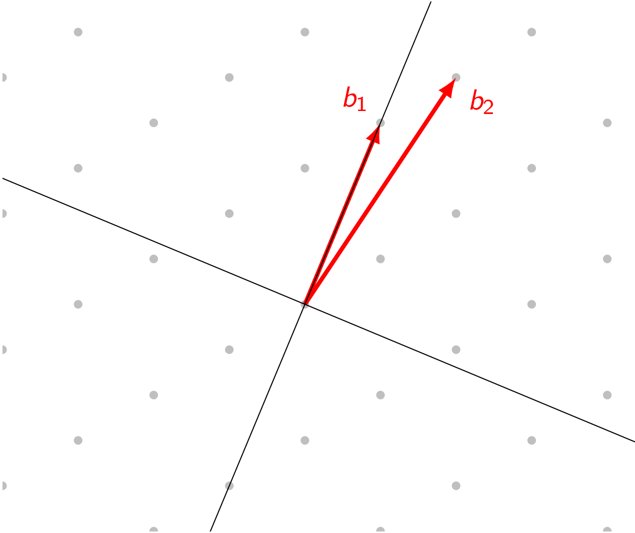
- Pick one direction, here b_1.



Visualization idea: Thijs Laarhoven.

Enumeration

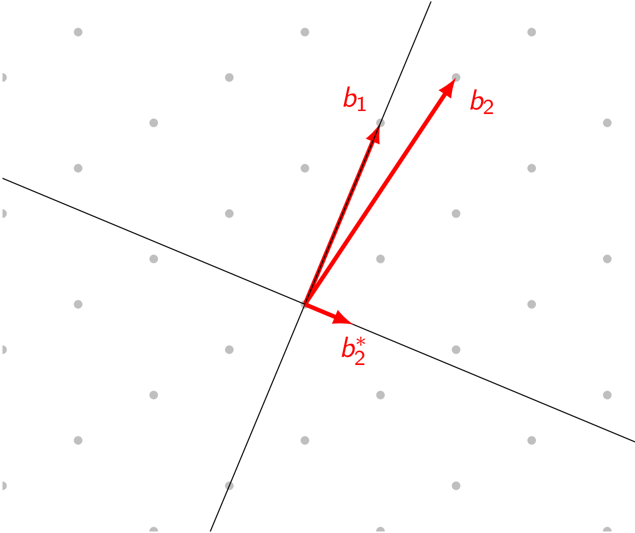
- Pick one direction, here b_1.
- Consider directions orthogonal to it.



Visualization idea: Thijs Laarhoven.

Enumeration

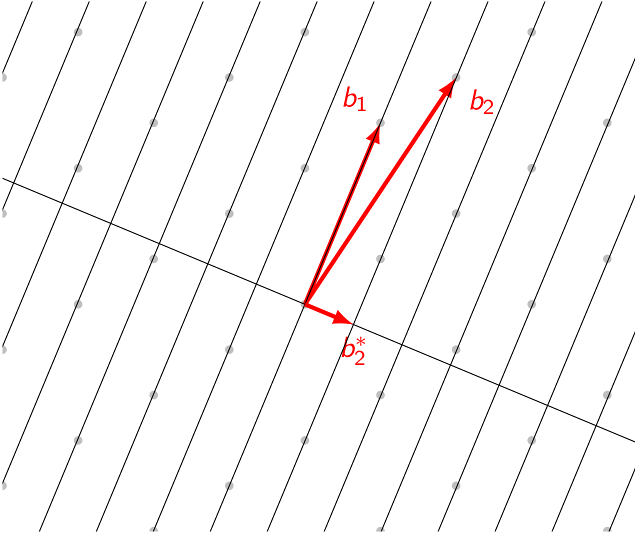
- Pick one direction, here b_1.
- Consider directions orthogonal to it.
- Project the other vector(s) on this orthogonal part.



Visualization idea: Thijs Laarhoven.

Enumeration

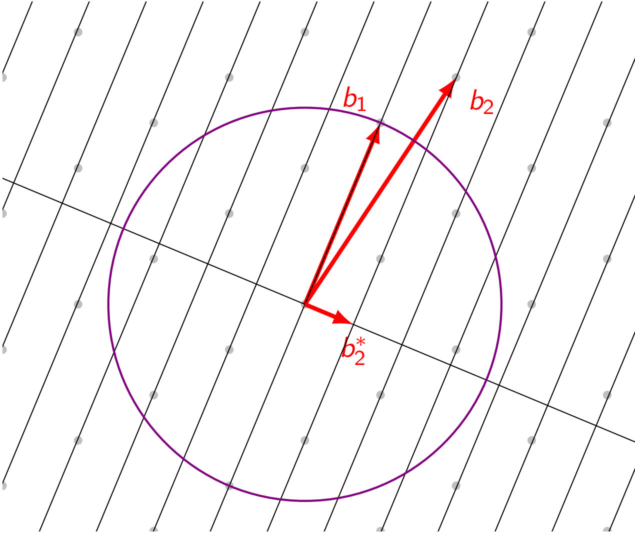
- Pick one direction, here b_1.
- Consider directions orthogonal to it.
- Project the other vector(s) on this orthogonal part.
- Make a grid parallel to b_1 spaced by the length of B_2^*.
- Consider points within the sphere of radius $||b_1||$.



Visualization idea: Thijs Laarhoven.

Enumeration

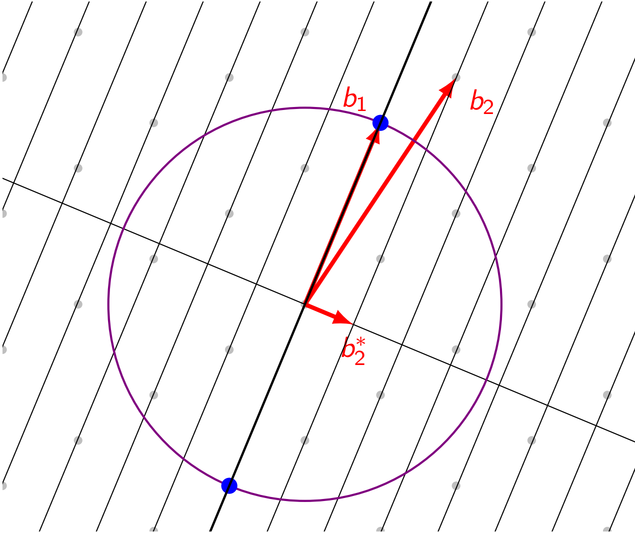
- Pick one direction, here b_1.
- Consider directions orthogonal to it.
- Project the other vector(s) on this orthogonal part.
- Make a grid parallel to b_1 spaced by the length of B_2^*.
- Consider points within the sphere of radius $||b_1||$.
- For each multiple of $||b_2^*||$ find all lattice points on that line.



Visualization idea: Thijs Laarhoven.

Enumeration

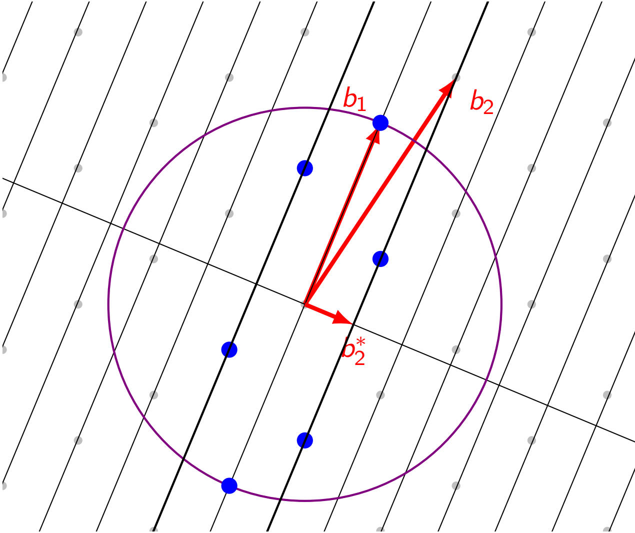
- Pick one direction, here b_1.
- Consider directions orthogonal to it.
- Project the other vector(s) on this orthogonal part.
- Make a grid parallel to b_1 spaced by the length of B_2^*.
- Consider points within the sphere of radius $||b_1||$.
- For each multiple of $||b_2^*||$ find all lattice points on that line.



Visualization idea: Thijs Laarhoven.

Enumeration

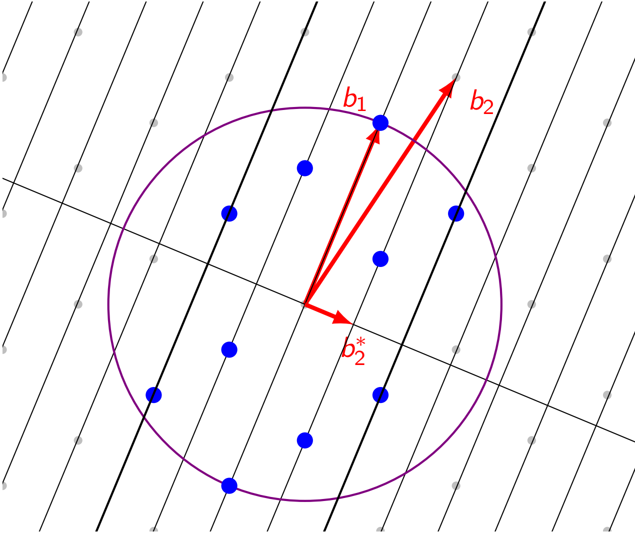
- Pick one direction, here b_1.
- Consider directions orthogonal to it.
- Project the other vector(s) on this orthogonal part.
- Make a grid parallel to b_1 spaced by the length of B_2^*.
- Consider points within the sphere of radius $||b_1||$.
- For each multiple of $||b_2^*||$ find all lattice points on that line.



Visualization idea: Thijs Laarhoven.

Enumeration

- Pick one direction, here b_1.
- Consider directions orthogonal to it.
- Project the other vector(s) on this orthogonal part.
- Make a grid parallel to b_1 spaced by the length of B_2^*.
- Consider points within the sphere of radius $||b_1||$.
- For each multiple of $||b_2^*||$ find all lattice points on that line.

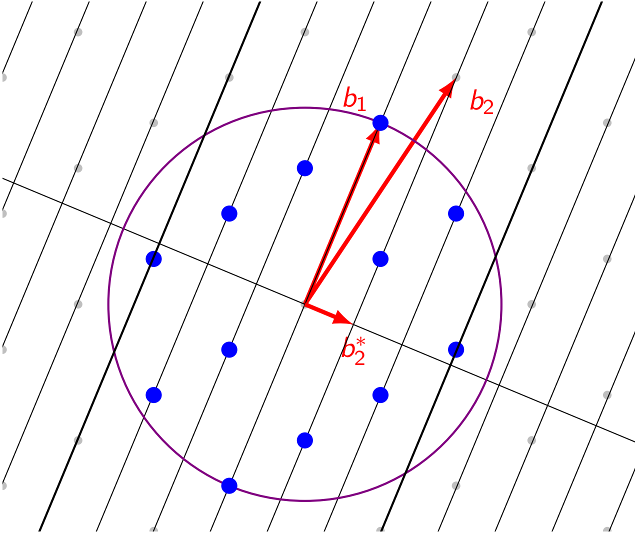


Visualization idea: Thijs Laarhoven.

Enumeration

- Pick one direction, here b_1.
- Consider directions orthogonal to it.
- Project the other vector(s) on this orthogonal part.
- Make a grid parallel to b_1 spaced by the length of B_2^*.
- Consider points within the sphere of radius $||b_1||$.
- For each multiple of $||b_2^*||$ find all lattice points on that line.
- Output the shortest vector in the sphere.

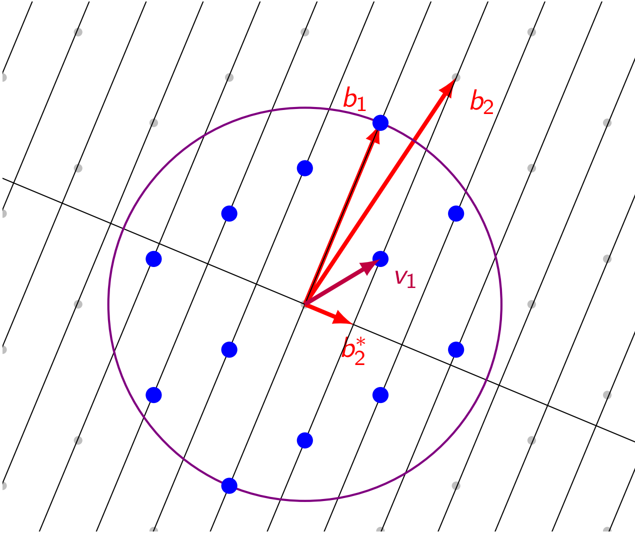
Visualization idea: Thijs Laarhoven.



Enumeration

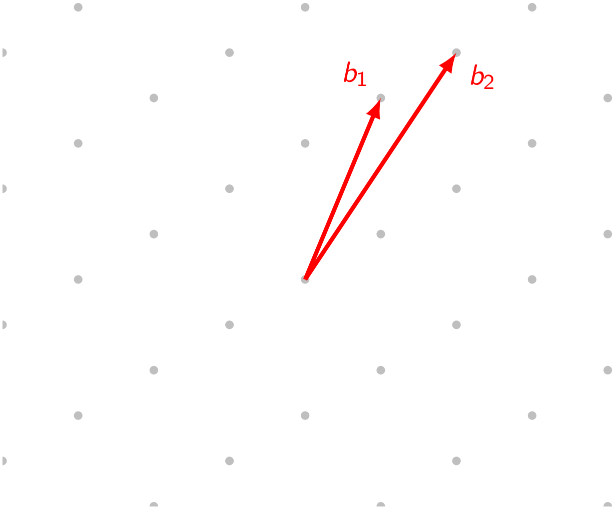
- Pick one direction, here b_1.
- Consider directions orthogonal to it.
- Project the other vector(s) on this orthogonal part.
- Make a grid parallel to b_1 spaced by the length of B_2^*.
- Consider points within the sphere of radius $||b_1||$.
- For each multiple of $||b_2^*||$ find all lattice points on that line.
- Output the shortest vector in the sphere.

Visualization idea: Thijs Laarhoven.



Enumeration with pruning

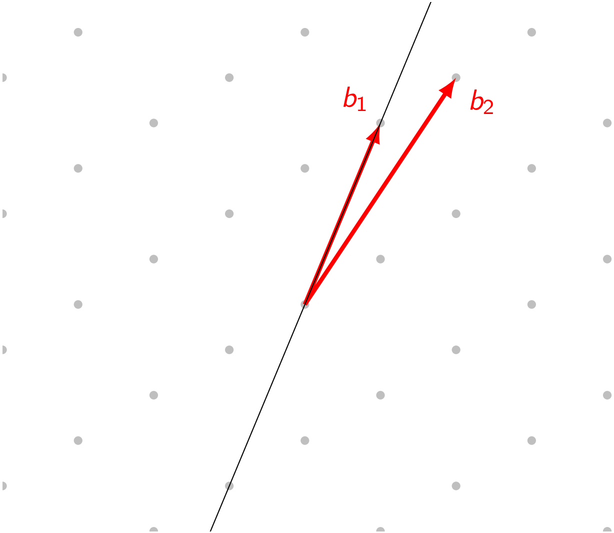
- Follow the steps for enumeration.



Visualization idea: Thijs Laarhoven.

Enumeration with pruning

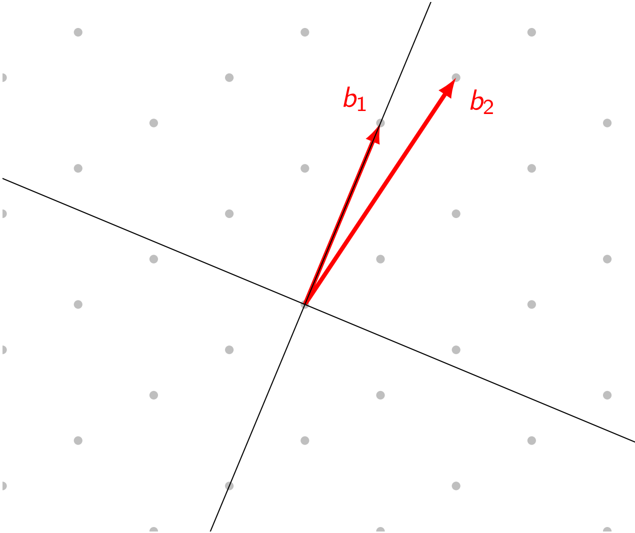
- Follow the steps for enumeration.



Visualization idea: Thijs Laarhoven.

Enumeration with pruning

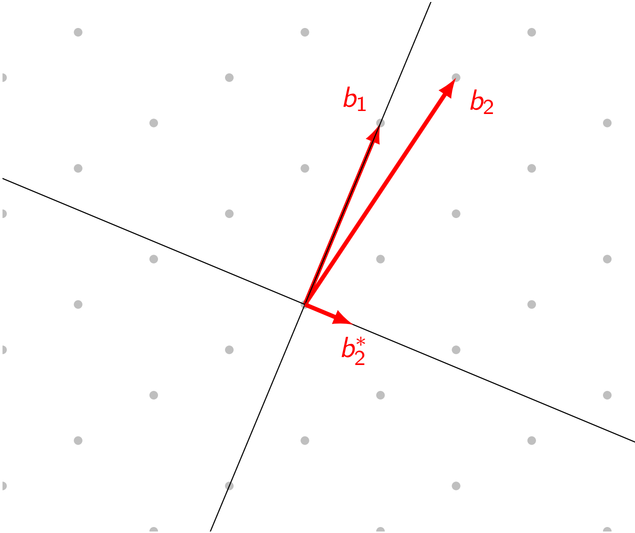
- Follow the steps for enumeration.



Visualization idea: Thijs Laarhoven.

Enumeration with pruning

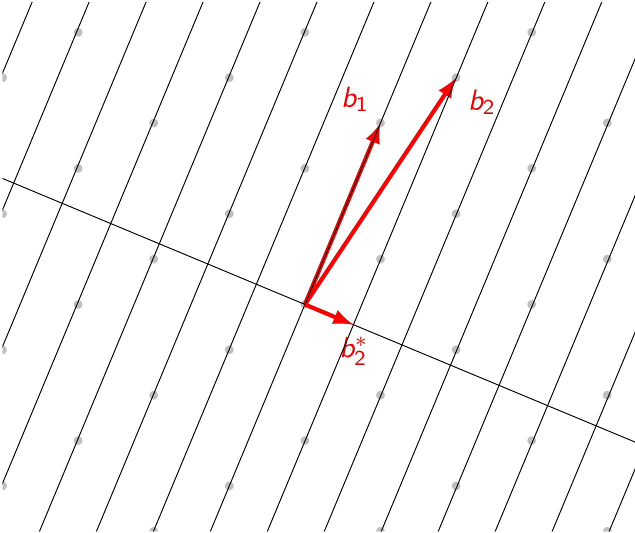
- Follow the steps for enumeration.



Visualization idea: Thijs Laarhoven.

Enumeration with pruning

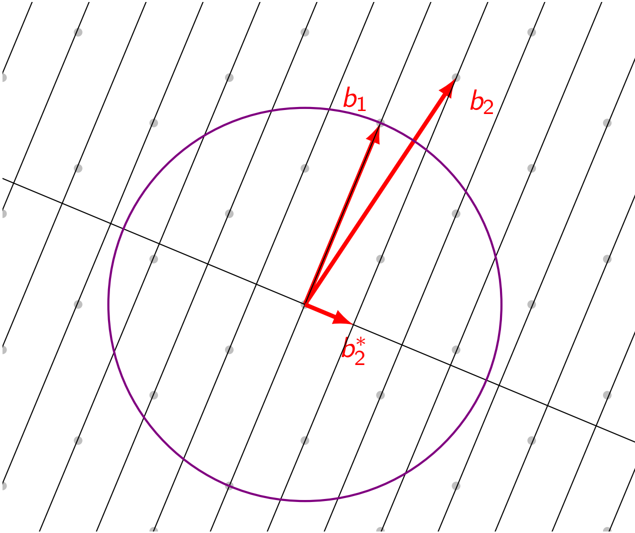
- Follow the steps for enumeration.



Visualization idea: Thijs Laarhoven.

Enumeration with pruning

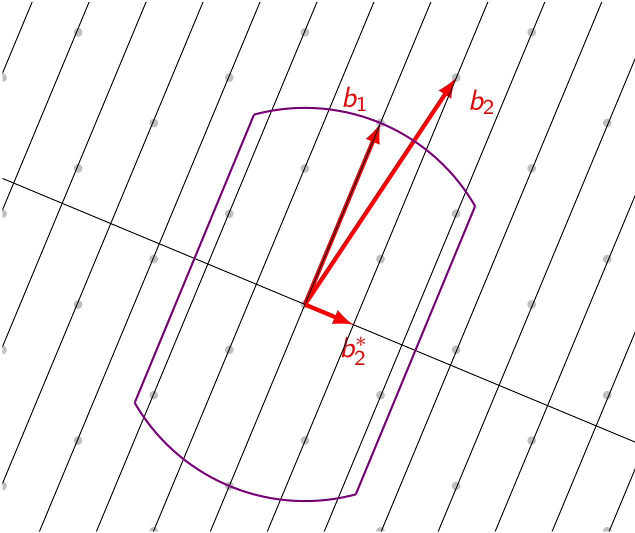
- Follow the steps for enumeration.
- Restrict the multiples of b_2^*.



Visualization idea: Thijs Laarhoven.

Enumeration with pruning

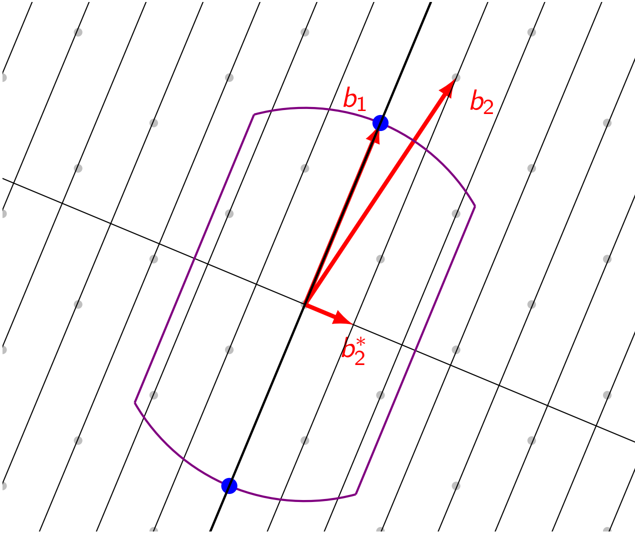
- Follow the steps for enumeration.
- Restrict the multiples of b_2^*.
- Continue as in enumeration



Visualization idea: Thijs Laarhoven.

Enumeration with pruning

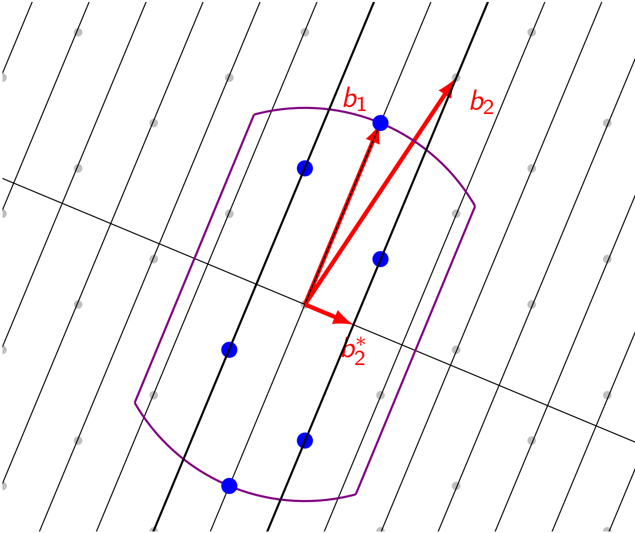
- Follow the steps for enumeration.
- Restrict the multiples of b_2^*.
- Continue as in enumeration



Visualization idea: Thijs Laarhoven.

Enumeration with pruning

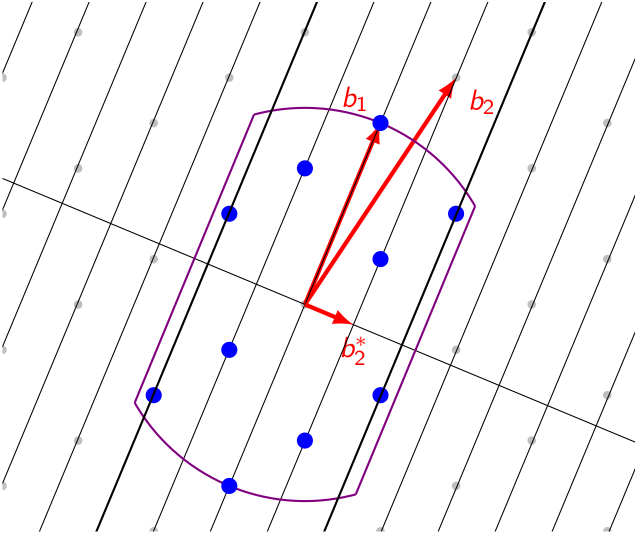
- Follow the steps for enumeration.
- Restrict the multiples of b_2^*.
- Continue as in enumeration



Visualization idea: Thijs Laarhoven.

Enumeration with pruning

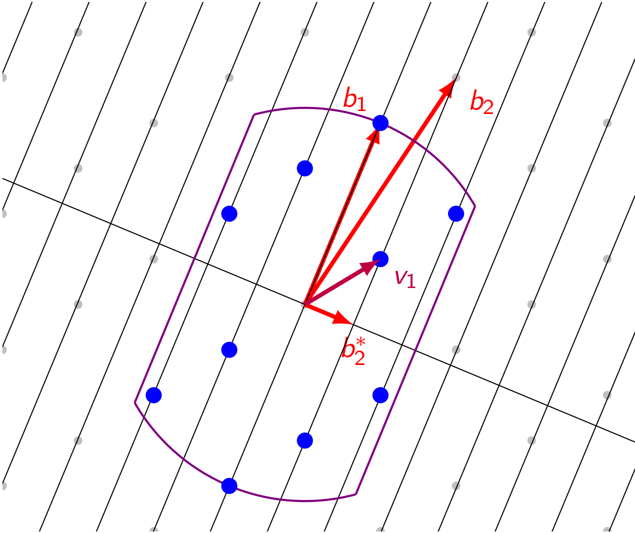
- Follow the steps for enumeration.
- Restrict the multiples of b_2^*.
- Continue as in enumeration



Visualization idea: Thijs Laarhoven.

Enumeration with pruning

- Follow the steps for enumeration.
- Restrict the multiples of b_2^*.
- Continue as in enumeration
- Output the shortest vector in the sphere.

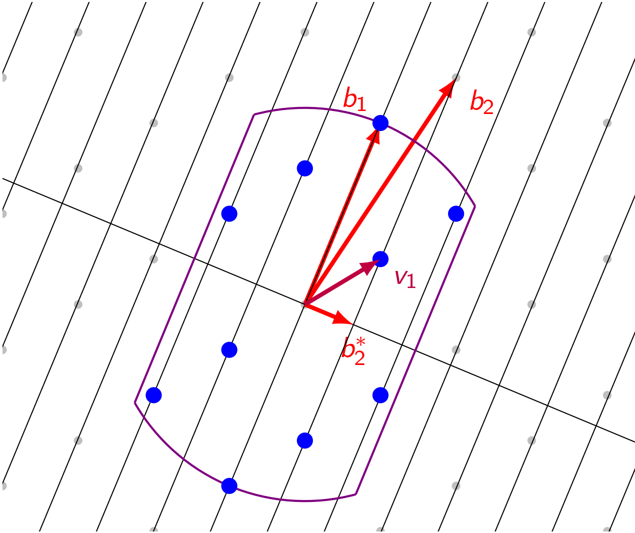


Visualization idea: Thijs Laarhoven.

Enumeration with pruning

- Follow the steps for enumeration.
- Restrict the multiples of b_2^*.
- Continue as in enumeration
- Output the shortest vector in the sphere.
- Benefit is that search space gets smaller;
  usually shortest vector is in pruned space.

Visualization idea: Thijs Laarhoven.



You can try this at home!

- Every NIST submission has a reference implementation.
  - `https://csrc.nist.gov/projects/post-quantum-cryptography/round-1-submissions`
  - (More than 90% of them have survived a week of cryptanalysis!)
- Contribute to the Open Quantum Safe project:
  - `https://github.com/open-quantum-safe/`
  - Caveat: Schemes might become obsolete due to cryptanalytic advances.
- Break stuff! Analyze proposals new and old, check the implementations. This needs more eyes, hands, computer power, . . .
- If you feel like turning on post quantum cryptography in your own projects, we would recommend a hybrid approach with ECC. (Like what Google did with CECPQ1.)