CSE 207B: Applied Cryptography

Nadia Heninger

UCSD

Fall 2023 Lecture 15

Announcements

1. HW 8 is available!

Last time:

- Authenticated key exchange

This time:

- Lattice-based cryptanalysis

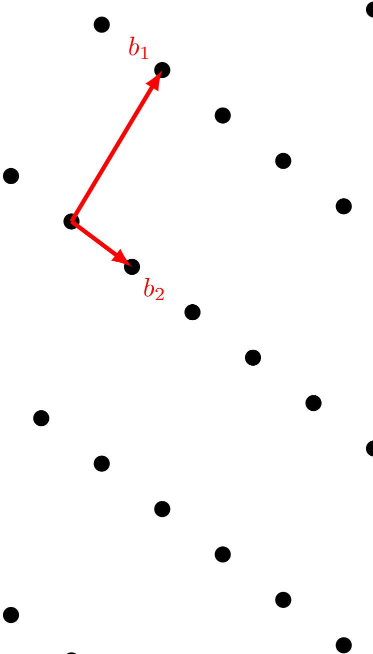
What is a lattice?

Definition

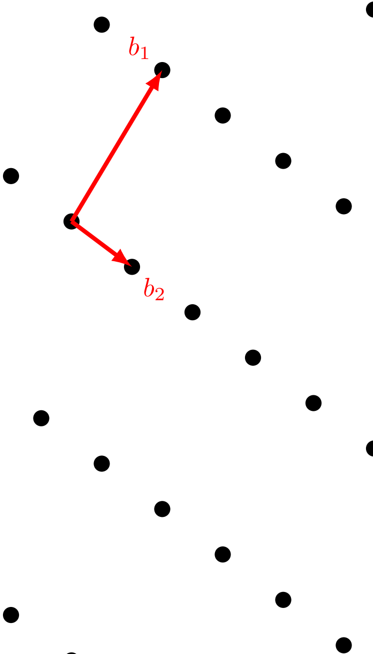
A **lattice** is a subset of $\mathbb{R}^n$ generated by integer linear combinations of some linearly independent basis $\{b_1, \ldots, b_n\}$.

Can represent as Cartesian coordinates: origin $(0, 0, \ldots, 0)$ $b_i = (z_1, \ldots, z_n)$.

- Has algebraic properties (it's a group under addition).
- Has geometric properties (it lives in $\mathbb{R}^n$ so has dot product, distance).



What is a lattice?



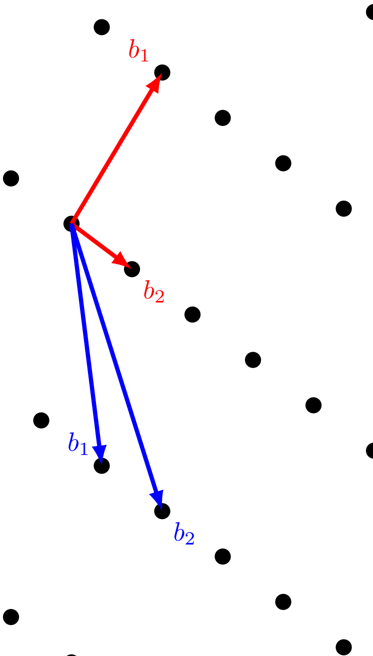
Definition

A **lattice** is a discrete additive subgroup of $\mathbb{R}^n$.

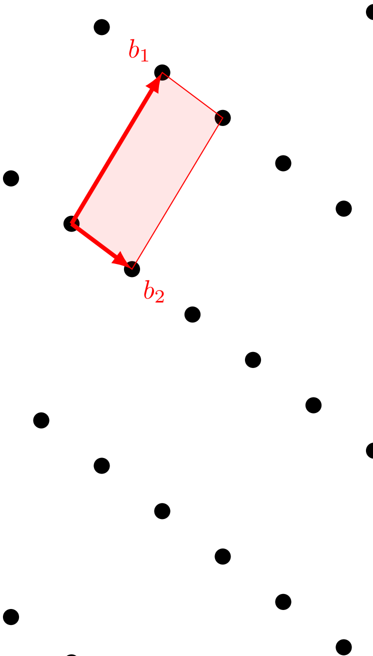
- Discrete: $\exists \delta > 0$ s.t. $|v_i - v_j| > \delta$ $\forall\ v_i, v_j \in L(B)$.
- Additive subgroup: closed under addition.

Properties of lattices: Bases

- In n dimensions a lattice has a basis of size at most n.
- The basis is not unique.
- Let $L(B)$ be the lattice generated by basis B. Deciding if $L(B) = L(B')$ for $B \neq B'$ is efficient. The Hermite Normal Form is unique and efficient to compute. Check if HNF(B) = HNF(B').



Properties of lattices: Determinant



Definition

The **determinant** of a lattice with a basis matrix B is $|\det B|$.

- The determinant is invariant for a given lattice.
- Gives volume of fundamental parallelepiped.

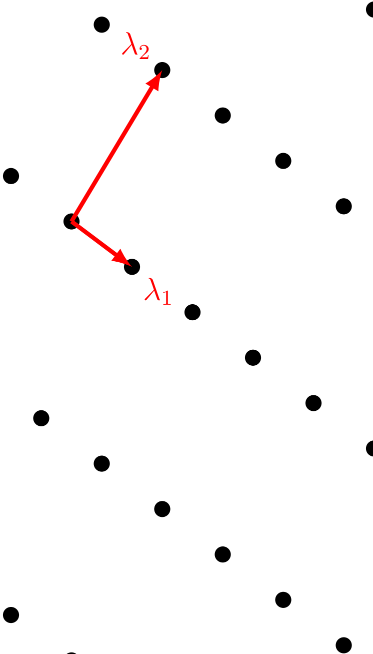
Properties of lattices: Minima

Let $\lambda_1 > 0$ be the length of the shortest vector in the lattice.

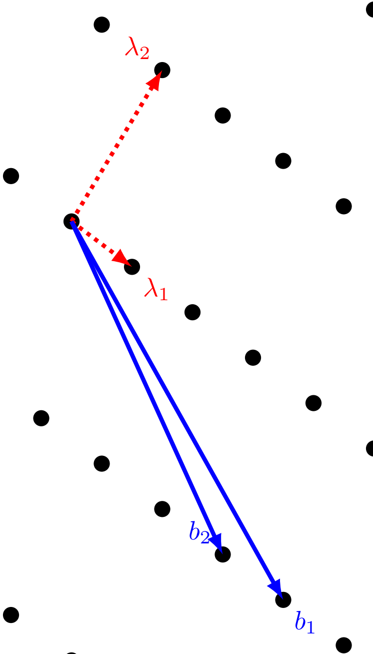
Theorem (Minkowski)

$\lambda_1(L) < \sqrt{n} \det L^{1/n}$

Can define *successive minima* λ_i, the length of the shortest vector linearly independent to the vectors achieving the $i-1$ successive minima.



Computational problems on lattices: SVP

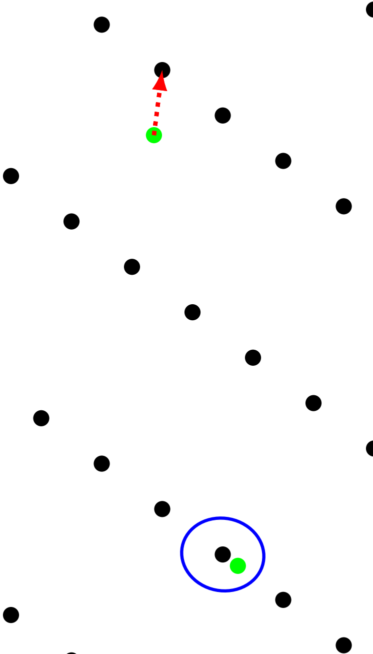


Shortest Vector Problem (SVP)

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

- SVP is NP-hard.

Computational problems on lattices: CVP



Closest Vector Problem (CVP)

Given an arbitrary basis for L, and a point x find the vector in L closest to x.

- CVP is NP-hard.

Approximation results for SVP

Input: Lattice basis B.
Desired output: Vector of length $\gamma\lambda_1(L(B))$.

$$\longleftarrow \quad \gamma \quad \longrightarrow$$

1 $\quad \sqrt{n}$ $\quad O(n \log n)$ $\quad n^{O(1)}$ $\quad 2^{O(n \log\log n / \log n)}$

1: NP-hard

$\sqrt{n}$: not NP-hard (NP$\cap$co-NP)

$O(n \log n)$: worst case$\rightarrow$average case reduction

$n^{O(1)}$: cryptography

$2^{O(n \log\log n / \log n)}$: polynomial time algorithm

Algorithmic results for SVP

Lenstra Lenstra Lovasz (LLL)

Given a basis for a lattice can in polynomial time find a *reduced* basis $\{b_i\}$ s.t.

$$|b_i| \leq 2^{(n-1)/2} \lambda_i$$

Algorithmic results for SVP

Lenstra Lenstra Lovasz (LLL)

Given a basis for a lattice can in polynomial time find a *reduced* basis $\{b_i\}$ s.t.

$$|b_i| \leq 2^{(n-1)/2} \lambda_i$$

Theorem (LLL (Simplified Version))

We can find a vector of length

$$|v| < 2^{\dim L} (\det L)^{1/\dim L}$$

Algorithmic results for SVP

Lenstra Lenstra Lovasz (LLL)

Given a basis for a lattice can in polynomial time find a *reduced* basis $\{b_i\}$ s.t.

$$|b_i| \leq 2^{(n-1)/2}\lambda_i$$

Theorem (LLL (Simplified Version))

We can find a vector of length

$$|v| < 2^{\dim L}(\det L)^{1/\dim L}$$

- In practice on random lattices, LLL finds $v = 1.02^n(\det L)^{1/\dim L}$. [Nguyen,Stehle]

Algorithmic results for SVP

Lenstra Lenstra Lovasz (LLL)

Given a basis for a lattice can in polynomial time find a *reduced* basis $\{b_i\}$ s.t.

$$|b_i| \leq 2^{(n-1)/2} \lambda_i$$

Theorem (LLL (Simplified Version))

We can find a vector of length

$$|v| < 2^{\dim L} (\det L)^{1/\dim L}$$

- In practice on random lattices, LLL finds $v = 1.02^n (\det L)^{1/\dim L}$. [Nguyen,Stehle]

BKZ

Given a lattice basis, can in time $2^{O(k)}$ find a reduced basis s.t. $|b_i| \leq k^{O(n/k)} \lambda_i$.

The "two faces" of lattices in cryptography

- **Cryptanalysis:** Can use approximation algorithms for SVP in lattices to cryptanalyze a wide variety of classical cryptography:
  - Attacks on low public exponent RSA
  - Factoring with partial knowledge
  - (EC)DSA with partial information about nonces
  - Knapsack-based cryptosystems
- **Cryptographic constructions:**
  - Lattice problems appear to be hard to solve for quantum computers, so lattice-based cryptosystems among most promising candidates for post-quantum cryptography.
  - Algebraic structure of lattices leads to many interesting cryptographic constructions that may someday be practical, like fully homomorphic encryption, identity-based encryption, etc.

History: Lattices and cryptography

1910	Minkowski's geometry of numbers
1973/1977	Public-key cryptography invented (GCHQ/RSA)
1978	Knapsack cryptography invented (Merkle-Hellmann)
1982	CVP NP-hard (van Emde Boas)
1982	LLL lattice basis reduction algorithm (Lenstra-Lenstra-Lovasz)
1983	LLL algorithm used against knapsack cryptosystems (Lagarias-Odlyzko)
1996	Lattice-based cryptosystems invented (Ajtai-Dwork)
1997	SVP NP-hard (Ajtai)
2005	LWE problem (Regev)
2009	Fully homomorphic encryption using ideal lattices (Gentry)

Historical Interlude: Subset Sum

Subset Sum Problem

Input: Integers $a_1, \ldots, a_n$, target integer T.

Goal: Find a subset $\sum_S a_i = T$.

- NP-hard
- First attempt to base cryptography off of NP-hardness.
- All schemes have a "trapdoor" that lets the decrypter solve the problem. (e.g. super-increasing sequence working modulo some number.)

Solving subset sum with lattices

Input: Integers $a_1, \ldots, a_n$, target integer T.

Generate lattice from rows of matrix

$$\begin{bmatrix} 1 & & & a_1 \\ & 1 & & a_2 \\ & & \ddots & \vdots \\ & & & -T \end{bmatrix}$$

Solving subset sum with lattices

Input: Integers $a_1, \ldots, a_n$, target integer T.

Generate lattice from rows of matrix

$$\begin{bmatrix} 1 & & & a_1 \\ & 1 & & a_2 \\ & & \ddots & \vdots \\ & & & -T \end{bmatrix}$$

A solution $\sum_i b_i a_i = T \;\; b_i \in \{0, 1\}$ determines a vector

$$v = (b_1, b_2, \ldots, 0) \qquad |v|_2 \approx \sqrt{n/2}$$

Solving subset sum with lattices

Input: Integers $a_1, \ldots, a_n$, target integer T.

Generate lattice from rows of matrix

$$\begin{bmatrix} 1 & & & a_1 \\ & 1 & & a_2 \\ & & \ddots & \vdots \\ & & & -T \end{bmatrix}$$

A solution $\sum_i b_i a_i = T \;\; b_i \in \{0, 1\}$ determines a vector

$$v = (b_1, b_2, \ldots, 0) \qquad |v|_2 \approx \sqrt{n/2}$$

- $\det L = T$, $\dim L = n + 1$; expect random non-solution vectors to have size $\sqrt{n} \det L^{1/\dim L}$; LLL has approximation factor 1.02^n.
- LLL or BKZ might find short v when $|v| = \sqrt{n/2} < T^{1/(n+1)}$
- Proposed knapsack cryptosystems contained "trapdoors" that made problem easier to solve.

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 too small.

This is why we use padding.

```
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's going on here? Coppersmith's method.

Theorem (Coppersmith)

We can efficiently compute up to $1/e$-fraction of the bits of an RSA-encrypted message with public exponent e if we know the rest of the plaintext.

```
sage: N.nbits()
296
sage: Integer('swordfish',base=35).nbits()
46
```

What's going on here? Coppersmith's method.

Theorem (Coppersmith)

Given a polynomial f of degree d and N, we can efficiently find all roots r_i satisfying

$$f(r_i) \equiv 0 \bmod N$$

when $|r_i| < N^{1/d}$ in time polynomial in $\log N$ and d.

In our case, our input polynomial looks like

$$f(x) = (a + x)^3 - c \equiv 0 \bmod N$$

We are looking for a root $r = \texttt{swordfish}$ satisfying

$$f(r) = (a + \texttt{swordfish})^3 - c \equiv 0 \bmod N$$

Why is this an interesting theorem?

1. A general method to solve polynomials mod N would break RSA: If c is a ciphertext,

$$x^e - c \equiv 0 \bmod N$$

   has a root $x = m$ for m our original message.

2. There is an efficient algorithm to solve equations mod primes.
   - For a composite, factor into primes, solve mod each prime, and use Chinese remainder theorem and Hensel lifting to lift solution mod N.

3. By accepting a bound on solution size, Coppersmith's method lets us solve equations without factoring N.

Coppersmith's Algorithm Outline

Input: polynomial f, modulus N.
Output: a root r modulo N.

In our example, we have $f(x) = (x+a)^3 - c$.

We will construct a new polynomial $Q(x)$ so that

$$Q(r) = 0 \qquad \text{over the integers.}$$

If we construct $Q(x)$ as

$$Q(x) = s(x)f(x) + t(x)N$$

with $s(x), t(x) \in \mathbb{Z}[x]$, then by construction

$$Q(r) \equiv 0 \bmod N$$

(In other words, $Q(x) \in \langle f(x), N \rangle$ over $\mathbb{Z}[x]$.)

Manipulating polynomials

Input: $f(x) = x^3 + f_2x^2 + f_1x + f_0$,N
Output: $Q(x) \in \langle f(x), N \rangle$ over $\mathbb{Z}[x]$.

If we only care about polynomials Q of degree 3, then

$$Q(x) = c_3 f(x) + c_2 N x^2 + c_1 N x + c_0 N$$

with $c_3, c_2, c_1, c_0 \in \mathbb{Z}$.

$$\begin{array}{cccccccccc}
 & c_3 & (x^3 & + & f_2x^2 & + & f_1x & + & f_0) \\
+ & c_2 & & & Nx^2 & & & & \\
+ & c_1 & & & & & Nx & & \\
+ & c_0 & & & & & & & N \\
\hline
 & & Q_3x^3 & + & Q_2x^2 & + & Q_1x & + & Q_0
\end{array}$$

Manipulating polynomials as coefficient vectors

We can represent elements of $\mathbb{Z}[x]$ as coefficient vectors:

$$g_d x^d + g_{d-1} x^{d-1} + \cdots + g_0 \qquad \leftrightarrow \qquad (g_d, g_{d-1}, \ldots, g_0)$$

If we construct the matrix

$$\begin{bmatrix} 1 & f_2 & f_1 & f_0 \\ & N & & \\ & & N & \\ & & & N \end{bmatrix}$$

Then the coefficient vector representing our polynomial

$$Q(x) = c_3 f(x) + c_2 N x^2 + c_1 N x + c_0 N$$

is an integer combination of the rows of this matrix.

Polynomial coefficient vectors and lattices

The set of vectors generated by integer combinations of the rows of our matrix

$$\begin{bmatrix} 1 & f_2 & f_1 & f_0 \\ & N & & \\ & & N & \\ & & & N \end{bmatrix}$$

is a *lattice*.

Coppersmith's method outline

Input: $f(x) \in \mathbb{Z}[x]$, $N \in \mathbb{Z}$. **Output:** r s.t. $f(r) \equiv 0 \bmod N$.

Intermediate output: $Q(x)$ such that $Q(r) = 0$ over $\mathbb{Z}$.

1. $Q(x) \in \langle f(x), N \rangle$ so $Q(r) \equiv 0 \bmod N$ by construction.

2. If $|r| < R$, then we can bound

$$\begin{aligned} |Q(r)| &= |Q_3 r^3 + Q_2 r^2 + Q_1 r + Q_0| \\ &\leq |Q_3|R^3 + |Q_2|R^2 + |Q_1|R + |Q_0| \end{aligned}$$

3. If $|Q(r)| < N$ and $Q(r) \equiv 0 \bmod N$ then $Q(r) = 0$.

We want a Q in our lattice with short coefficient vector!

Coppersmith's method outline

1. Construct a matrix of coefficient vectors of elements of $\langle f(x), N \rangle$.

2. Run a lattice basis reduction algorithm on this matrix.

3. Construct a polynomial Q from the shortest vector output.

4. Factor Q to find its roots.

Running Coppersmith's method on our example

Input: $f(x) = (x+a)^3 - c$, N
Output: $r < R$ such that $f(r) \equiv 0 \bmod N$.

1. Construct lattice basis

$$\begin{bmatrix} R^3 & 3aR^2 & 3a^2R & a^3 - c \\ & NR^2 & & \\ & & NR & \\ & & & N \end{bmatrix}$$

$\dim L = 4$

$\det L = R^6 N^3$

Factor of R is so that $Q(r) \leq |v|$ for $v \in L$.

Running Coppersmith's method on our example

Input: $f(x) = (x+a)^3 - c$, N
Output: $r < R$ such that $f(r) \equiv 0 \bmod N$.

1. Construct lattice basis

$$\begin{bmatrix} R^3 & 3aR^2 & 3a^2R & a^3 - c \\ & NR^2 & & \\ & & NR & \\ & & & N \end{bmatrix}$$

$$\dim L = 4$$

$$\det L = R^6 N^3$$

   Factor of R is so that $Q(r) \le |v|$ for $v \in L$.

2. Ignoring approximation factor, we can solve when

$$\begin{aligned} |Q(r)| \le |v_1| \le \det L^{1/\dim L} &< N \\ (R^6 N^3)^{1/4} &< N \\ R &< N^{1/6} \end{aligned}$$

In my example I chose $\lg N = 296$, $\lg r = 46$.

Achieving the Coppersmith bound $r < N^{1/d}$

1. Generate lattice from subset of $\langle f(x), N \rangle^k$.
2. Allow higher degree polynomials.

Theorem (CHHS 2016)

It is not possible to solve for $r > N^{1/d}$ with any method that constructs auxiliary polynomial $Q(x)$.

Countermeasures for real-world RSA

- Must use padding scheme with cryptographically secure randomized padding for RSA.
  - PKCS#1v1.5 widely used in practice, not CCA-secure.
  - OAEP is CCA-secure but not widely used.

- Current recommendation: Use RSA exponent $e \geq 65537$.