Information Security Lab
Autumn Semester 2023
Module 4, Week 1 – Elliptic Curve Cryptography + Intro to Lattices

Kenny Paterson (@kennyog)

Applied Cryptography Group

https://appliedcrypto.ethz.ch/

Overview of today's lecture

- Elliptic Curves
- Cryptography from Elliptic Curves
- Introduction to lattices

(Week 2: lattice cryptanalysis and breaking ECDSA with partially known nonces.

Week 3: Coppersmith's method, with more applications in cryptanalysis.)

Overview of this module's labs

- **Week 1**: you will be implementing ECDSA variants, starting from basic elliptic curve operations.
- You'll also see some simple attacks on weak ECDSA implementations.
- **Week 2:** you will see how vulnerable ECDSA is to side-channel attacks and implementation errors by breaking it using *lattice cryptanalysis.*
- There, we will allow you to use basic tools for performing operations on lattices, but the rest will be up to you!
- **Week 3:** you will use *Coppersmith's method* to solve various cryptanalytic challenges.

Lab setup

- We will operate labs in a different mode from previous weeks.
- **Only one room for everyone: ML D 28 (this room)**
  - All 5 TAs will be here.
  - It's a lecture hall, so make sure your laptop is charged.
- At the beginning of each Lab Session we will give a short presentation covering topics like:
  - Installation/setup of sagemath.
  - Important or helpful sagemath functions,
  - Potential pitfalls to be aware of.

Elliptic Curves

Classical Discrete Log Cryptography

- Recall: set p a large prime; q a prime divisor of p-1; g an element of order q mod p.
- So g generates G_q, a cyclic group of prime order q in the set of integers modulo p:

$$G_q = \{g^0 = 1, g^1, g^2, \ldots, g^{q-1}\}.$$

- (p,q,g) are **public parameters** for discrete log based crypto.
- E.g. Ephemeral Diffie-Hellman Key Exchange (EDHKE/DHE):
  - Alice and Bob agree on (p,q,g), e.g. get them from a standard.
  - Alice selects x uniformly at random from $\{0,1,\ldots,q-1\}$, and sends Bob $X = g^x \bmod p$.
  - Bob selects y uniformly at random from $\{0,1,\ldots,q-1\}$, and sends Alice $Y = g^y \bmod p$.
  - Alice and Bob can both now compute $Y^x = X^y = g^{xy} \bmod p$ and use this value to derive a shared key.

Classical Discrete Log Cryptography

Security in the classical discrete log setting depends on the Discrete Logarithm Problem and variants:

The discrete logarithm problem (DLP) in G_q:

Let (p, q, g) be as above. Given as input (p, q, g) and $\boldsymbol{y = g^x \bmod p}$, where x is a uniformly random value in {0,1,…,q-1}, **find x.**

Problem: making the DLP in G_q sufficiently hard in the face of the best known algorithms means using large q and p.

e.g. For "128-bit security", p should have 3072 bits and q should have 256 bits.

This makes cryptographic algorithms using the DLP in this setting slow and bandwidth hungry.

Elliptic Curves

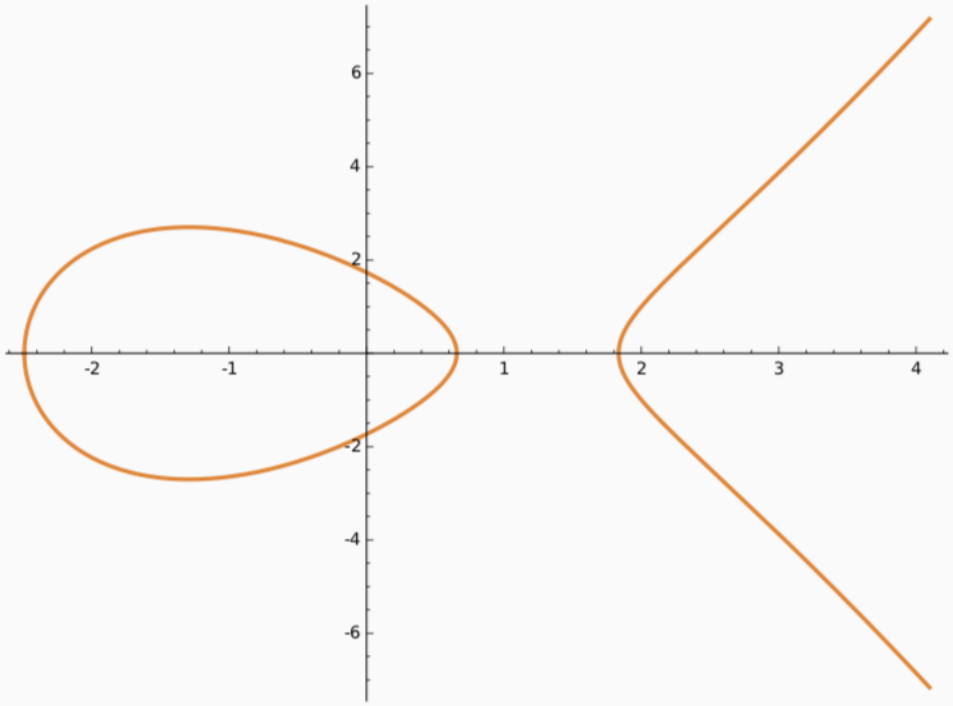
- An elliptic curve over a field F is a set of pairs $(x,y) \in F \times F$ called *points*, defined by some equation in x and y defined over F.
- Think of F as the integers mod p for some large prime p (typically 256 bits).
- A common form for the equation of an elliptic curve is

$$E = \{ (x,y) \in F \times F \mid y^2 = x^3 + ax + b \} \cup \{ O \}$$

  where $4a^3 + 27b^2 \neq 0$ over F.

- Here a and b are coefficients from F that can vary to give us different curves.
- Here "point at infinity" O is a special curve point that does **not** have a representation as a pair $(x,y) \in F \times F$.

Elliptic Curve over the **Rationals** with a = −5, b = 3




```
sage: E = EllipticCurve([-5, 3])
```

$y^2 = x^3 + 2x + 4$ modulo 5

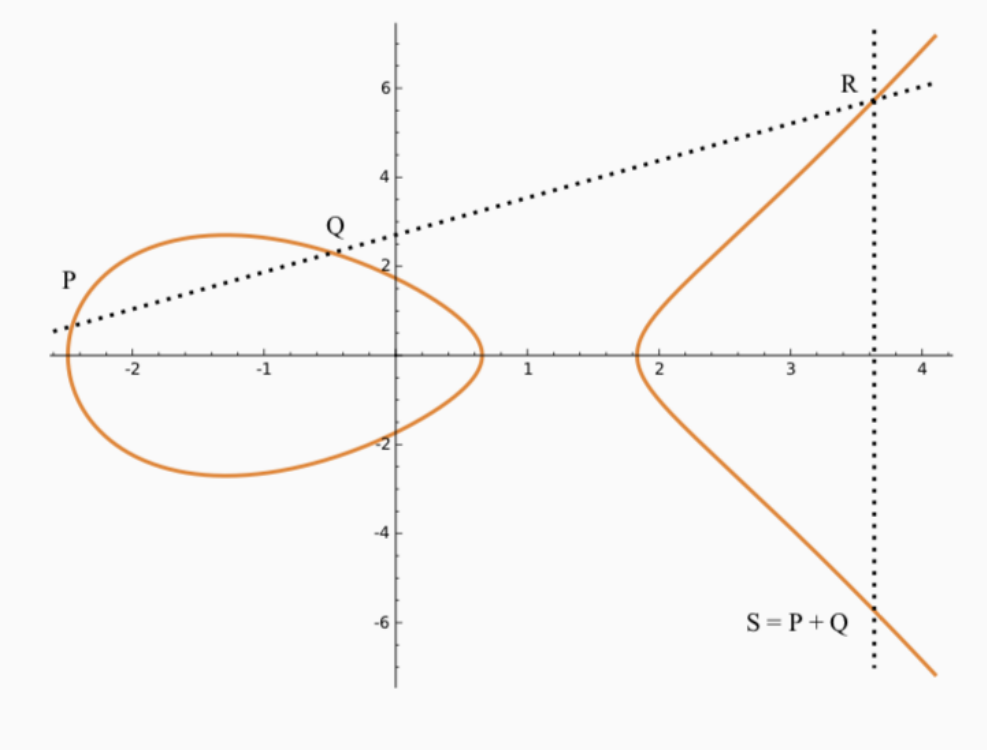
x	0	1	2	3	4
x^3	0	1	3	2	4
$2x$	0	2	4	1	3
4	4	4	4	4	4
y^2	4	2	1	2	1
y	2,3		1,4		1,4

- Here, we see fairly typical behaviour of elliptic curve over a finite field (using artificially small p=5).
- $x^3 + 2x + 4$ takes on 3 distinct values; of these 2 values have square roots mod 5, leading to points (0,2), (0,3), (2,1), (2,4), (4,1), (4,4).
- Including O, we get a total of 7 points on our curve E.

Addition of Points on an Elliptic Curve

- Any pair of points on an elliptic curve can be added to obtain a third point.
- The point at infinity O acts as an (additive) identity for this addition operation.
  - $P + O = O + P = P$ for all elliptic curve points P.
- Each point P has an (additive) inverse denoted $-P$.
  - O is its own inverse: $O + O = O$. (NB Symbols not numbers here!)
  - If $P = (x, y)$ then $-P = (x, -y)$.
  - $P + (-P) = O$.

Addition of Points on an Elliptic Curve



- There is a geometric interpretation of the addition process: to find $P + Q$, draw a straight line through P and Q, find the point of intersection with the curve, and project through the x-axis.
- Special case when $P=Q$: use tangent line at P.

Elliptic Curves as Groups

- This addition law turns the set of points on an elliptic curve over a field into a **group**.
- The group operation is written as "+", and we speak of **adding** two points.
- The identity in the group is the special point *O* (the point at infinity).
- The group **order** is the number of points on the curve.
- By carefully choosing *E*, we can ensure that the group has prime, or nearly prime order, good for doing cryptography.

Example: Elliptic Curves as Groups

- Recall the curve E with equation $y^2 = x^3 + 2x + 4$ over $F = F_5$.
- This curve has points O, (0,2), (0,3), (2,1), (2,4), (4,1), (4,4).
- So the group order is 7.
- Take P = (0,2).
- Then it so happens that P, $P+P$, $P+P+P$,… gives all 7 group elements.
- So P is a **generator** of the group of points on E.
- Compare with $\{1, g, g^2, \ldots g^{q-1}\}$ in the usual discrete logarithm setting (where we have a subgroup of order q mod p).

Scalar Multiplication

- We write $[k]P$ for the operation of adding P to itself k times.
- This is called *scalar multiplication by k*.
- This is the analogue of exponentiation in the usual discrete log setting, i.e. $[k]P$ on curve roughly equivalent to $g^x \bmod p$.
- In our example, P, $[2]P$, $[3]P$,… gives us the full set of points on the curve.
- NB1: $[7]P = O$, c.f. $g^q = 1 \bmod p$ in classical DL setting.
- NB2: If $P = (x,y)$, then $[k]P \neq (kx, ky)$ in general!!!

Scalar Multiplication

- To compute some scalar multiple $[k]P$ of a point P we use an analogue of square-and-multiply called *double-and-add.*
- Suppose we want to compute $[13]P$.
- $13_{10} = 1101_2$, so we compute $[13]P$ by the following chain:

1: P

1: Double and add: $[2]P + P = [3]P$

0: Double: $[6]P$

1: Double and add: $[12]P + P = [13]P$

Cryptography from Elliptic Curves

The Elliptic Curve Discrete Logarithm Problem

Recall the (classical) discrete logarithm problem:

The Discrete Logarithm Problem in G_q:

Let (p, q, g) be group parameters (so q divides $p-1$; g has order q mod p).

Set $y = g^x \bmod p$, where x is a uniformly random value in $\{0,1,\ldots,q-1\}$.

Given (p, q, g) and y, find x.

The Elliptic Curve Discrete Logarithm Problem (ECDLP):

Let E be an elliptic curve over the field F of prime order p.

Let P be a point of prime order q on E.

Set $Q = [x]P$ where x is a uniformly random value in $\{0,1,\ldots,q-1\}$.

Given E and points P, Q, find x.

The Elliptic Curve Discrete Logarithm Problem

- The essence of Elliptic Curve Cryptography is that, except for some special cases, the best algorithms for solving ECDLP run in time $O(q^{1/2})$ where q is the order of the generator P.
- These are in fact *generic* algorithms that work in any finite abelian group.
- The $O(q^{1/2})$ behaviour enables us to choose much smaller parameters than are needed in "traditional" discrete-log-based cryptography.
- This results in more compact keys, ciphertexts, etc, and faster cryptographic operations.
- For 128-bit security, we want $O(q^{1/2}) \approx 2^{128}$, so we need q (and hence p) to have 256 bits.

Cryptography from ECDLP

- Most schemes for the DLP setting can be translated easily into the ECDLP setting.
- Example: ECDHE (Elliptic Curve Diffie-Hellman Ephemeral).
  - Alice and Bob agree on a curve E and a base-point P of prime order q.
  - Alice chooses x uniformly at random from $\{0,1,..,q\text{-}1\}$, and sends Bob $[x]P$.
  - Bob chooses y uniformly at random from $\{0,1,..,q\text{-}1\}$, and sends Alice $[y]P$.
  - Both sides can now compute $[xy]P$: Alice computes $[x]([y]P)$ and Bob computes $[y]([x]P)$.
  - See https://curves.xargs.org/ for animated examples of this protocol.

ECC Setup

To set up a system for using elliptic curve cryptography:

- We need to decide on a field *F* (usually a prime field for some prime *p*).
- We need to decide on a curve *E* over that field.
- We need to find a base point *P* on the curve of known and large prime order *q.*
- **Solution** to all of the above: use curve standards, e.g. NIST or IETF.
- We also need to support the new arithmetic of scalar multiplication on our curve, in a fast and secure manner.
- **Solution**: use a cryptographic library.

An example standardised curve: NIST P-256

- $p = 2^{224}(2^{32} - 1) + 2^{192} + 2^{96} - 1$.
- $a = -3$
- b := 5ac635d8 aa3a93e7 b3ebbd55 769886bc 651d06b0 cc53b0f6 3bce3c3e 27d2604b.
- $h = 1$; q = FFFFFFFF 00000000 FFFFFFFF FFFFFFFF BCE6FAAD A7179E84 F3B9CAC2 FC632551.
- A base point P is also specified.
- NIST P-256 is a curve of prime order q; special *sparse form* of p potentially makes mod p arithmetic faster.
- Very widely supported in crypto libraries.
- p and q have 256 bits, so complexity of solving ECDLP is about 2^{128}.

An example standardised curve: Curve25519

- Introduced by Bernstein in 2005/2006.
- $p = 2^{255} - 19$, allowing very fast modular reduction.
- Curve equation: $y^2 = x^3 + 486662x^2 + x$.
- Curve has Montgomery form, allowing ECDH operations to be done using only x coordinates in a side-channel resistant manner.
- Group order: $8(2^{252} + 27742317777372353535851937790883648493)$ – co-factor of 8.
- "Minimal" curve satisfying various security/performance criteria.
- Offers a bit less than 128-bit security, improved speed compared to, e.g. NIST P-256.
- Adopted for use in TLS 1.3 (along with NIST P-256, NIST P-384, NIST P-521 and Curve448-Goldilocks).
- See https://cr.yp.to/ecdh/curve25519-20060209.pdf and RFC 7748 for further details.

ECDSA and friends

ECDSA

- ECDSA is a translation of DSA from the standard DL setting to the elliptic curve setting.
- ECDSA (and DSA) specified in: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf and ANSI X9.62 (not free!).
- Detailed description in Boneh-Shoup, Section 19.3, but using different notation: http://toc.cryptobook.us/book.pdf
- Signatures are pairs (r,s) where r and s are integers mod q, the order of base point P; hence 512 bit signatures at the 128-bit security level.
- Public key is a point Q on an elliptic curve, requiring 256 bits at 128-bit security level.
- This is pretty compact!

ECDSA – The Gory Details

Parameters: (E, p, n, q, h, P, H) defining a curve E over field F_p with $n = q \cdot h$ points, subgroup of prime order q and generator P of order q; H is a hash function, e.g. SHA-256 (here we assume output of H is at least bit-size of q).

KeyGen:

Set $Q = [x]P$ where x is uniformly random from $\{1,\ldots,q\text{-}1\}$.

Output verification key: $\boldsymbol{Q}$; signing key: $\boldsymbol{x}$.

Sign: Inputs (x, m) // x is private key; m is the message to be signed

h = bits2int($H(m)$) mod q. // take len(q) MSBs of $H(m)$, cast to BigInt, reduce mod q.

Do:

1. Select k uniformly at random from $\{1,\ldots,q\text{-}1\}$.
2. Compute r = x-coord($[k]P$) mod q. // $[k]P$ is a point on E; its x-coord is in F_p; we consider that as an integer and reduce mod q.
3. Compute $s = k^{-1}(h + xr)$ mod q.

Until $r \neq 0$ and $s \neq 0$. // works first try w.h.p.

Output (r,s).

ECDSA – The Gory Details

Verify: Inputs (Q, m, (r,s)) // Q is verification key; m is message; (r, s) is claimed signature.

1. check that $1 \le r \le q-1$ and $1 \le s \le q-1$.
2. compute $w = s^{-1} \bmod q$.
3. compute $h = \text{bits2int}(H(m)) \bmod q$.
4. compute $u_1 = w \cdot h \bmod q$ and $u_2 = w \cdot r \bmod q$.
5. compute $Z = [u_1]P + [u_2]Q$.
6. If (x-coord(Z) mod q == r) then output 1 else output 0.

Correctness:

Suppose (r,s) is a signature for message m under key Q. Then:

$$Z = [u_1]P + [u_2]Q = [s^{-1}h]P + [s^{-1}r]Q = [s^{-1}(h + xr)]P = [k]P.$$

Here we used $s = k^{-1}(h + xr) \bmod q$ from the signing algorithm to obtain $s^{-1}(h + xr) = k \bmod q$.

Recalling that $r = \text{x-coord}([k]P) \bmod q$ completes the argument.

ECDSA Security and Implementation Pitfalls

- Implementation requires:
  - Various fiddly conversions of bit-strings to integers, etc: bits2int() and conversion of mod p integers to mod q integers.
  - Uniform sampling of integers k in the range $\{1,\ldots, q\text{-}1\}$ – use rejection sampling (sample from $[0, 2^t]$ for t = bitsize(q), until result is in $\{1,\ldots, q\text{-}1\}$).
  - Computation of multiplicative inverses mod q: k^{-1}, s^{-1}.
  - Scalar multiplications: $Q = [x]P$; $[k]P$; $[u_1]P + [u_2]Q$.
  - Sanity checks on r, s.
- There are lots of ways to get some or all of this wrong!
  - Repeating k, or k being predictable due to bad RNG – **see labs this week**.
  - Sampling k wrongly, e.g. choose k from $[0, 2^t]$ where t is bit-size of q, and reduce mod q.
  - Leaking some or all of k through a side-channel attack, e.g. running time of $[k]P$ or computation of k^{-1} mod q – **see labs next week**.

ECDSA Variants

- ECDSA is very sensitive to randomness failures, e.g. Sony Playstation fail, various cryptocurrency incidents.
- The key issue: if the same *k* is ever used twice by a signer on two different messages, then an attacker can detect this and recover the private key.
- RFC 6979: de-randomisation technique for ECDSA.
- Similar trick used in EdDSA (along with a specific curve, and different signing and verification equations).

Lattices and Lattice Reduction

Lattices

Definition (full-rank lattice)

Let $\{\underline{b}_1, \underline{b}_2, \ldots, \underline{b}_n\}$ be n linearly independent vectors in $\mathbb{R}^n$.

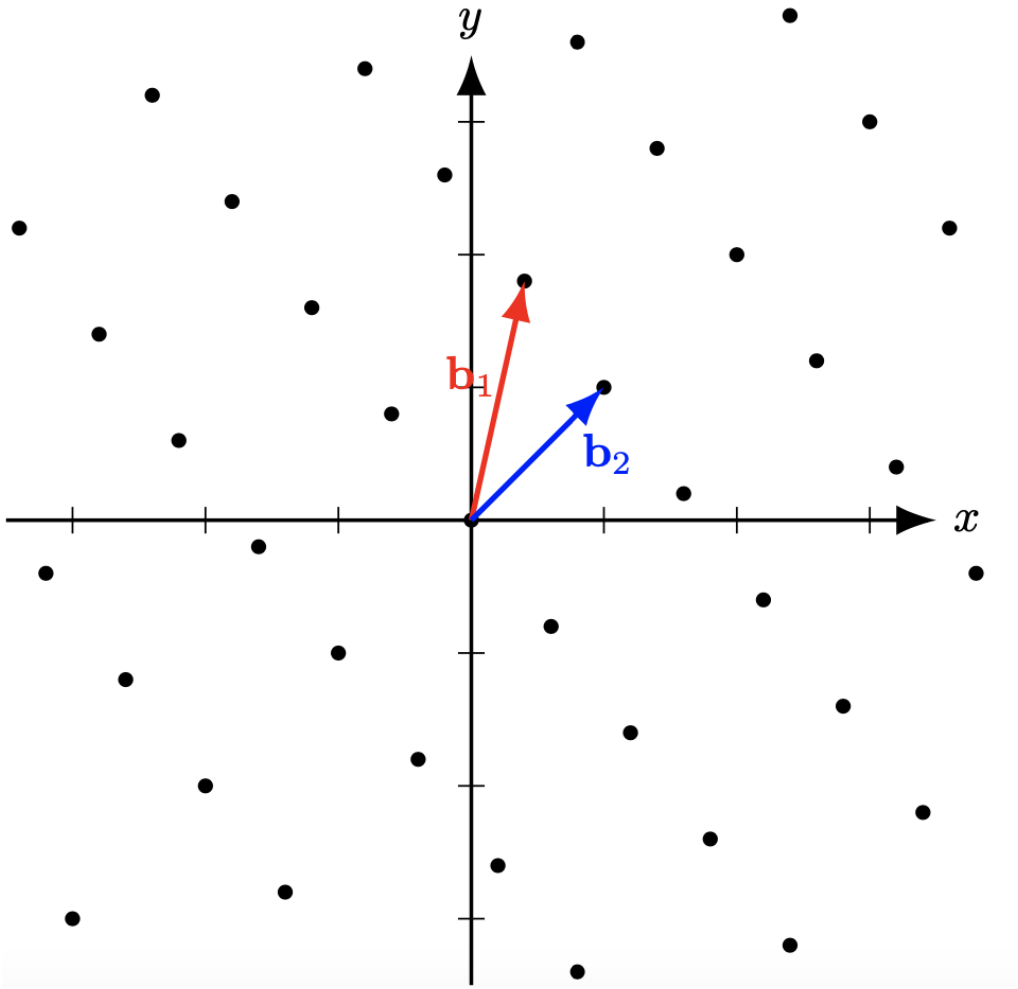
Then the lattice generated by $\{\underline{b}_1, \underline{b}_2, \ldots, \underline{b}_n\}$ is the set:

$$L := \{\sum_{i=1}^{n} l_i \, \underline{b}_i : l_i \in \mathbb{Z}\}$$

of **integer** linear combinations of the $\underline{b}_i$.

- If we allow real-valued linear combinations, we just get $L = \mathbb{R}^n$.
- Example: n=2; $\underline{b}_1$ = (1,0); $\underline{b}_2$ = (0,1), then $L = \mathbb{Z} \times \mathbb{Z}$.
- Example: n=2; $\underline{b}_1$ = (12.3,π); $\underline{b}_2$ = (-5,e), then L = ??.
- In general, the sums, differences etc, of lattice vectors are also lattice vectors.
- So a lattice is a *discrete subgroup* of $\mathbb{R}^n$.

Lattices – 2-d example



Lattice Bases

Definition (basis matrix of a full-rank lattice):

A basis matrix B of a lattice $L \subset \mathbb{R}^n$ is an $n \times n$ matrix formed by taking the rows to be basis vectors $\underline{b}_i$. Then

$$L = \{ \underline{x}B : \underline{x} \in \mathbb{Z}^n \}$$

- In cryptanalysis applications, our matrix entries will usually be integers.
- Two different, full-rank matrices B, B' can generate the same lattice.

Fact:

Two different matrices B, B' generate the same lattice L if and only if $B' = U B$ where U is an $n \times n$ matrix with integer entries and determinant ± 1.

Definition (determinant of a full-rank lattice):

The determinant of a full-rank lattice L is the absolute value of the determinant of a basis matrix B for the lattice.

Lattice Successive Minima

Definition (norm of a vector):

For $\underline{v} \in \mathbb{R}^n$, $||\underline{v}||$ denotes the Euclidean norm of $\underline{v}$, i.e.

$$||\underline{v}|| = (\sum_{i=1}^{n} \boldsymbol{v_i^2})^{1/2}.$$

Definition (successive minima of a lattice):

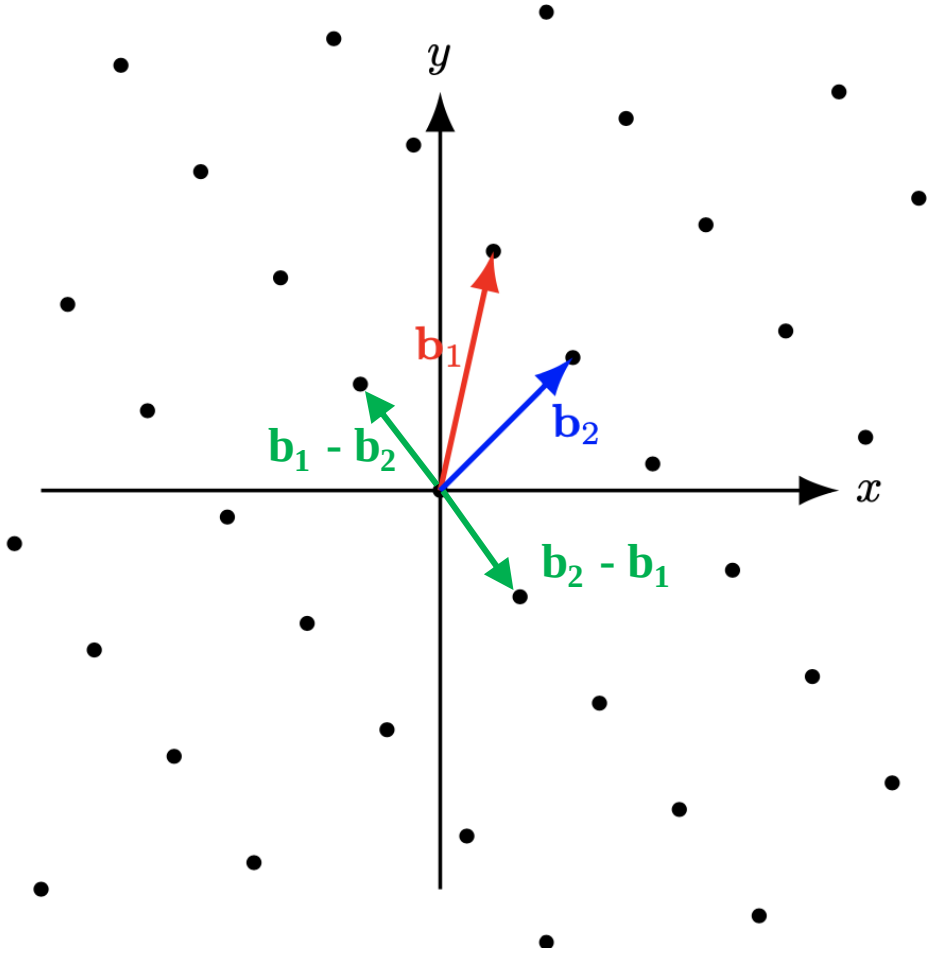
Let $L \subset \mathbb{R}^n$ be a full rank lattice. The successive minima of L are the values $\lambda_1, \ldots, \lambda_n \in \mathbb{R}$ such that:

For $1 \leq j \leq n$, λ_i is the smallest real value such that there exist i linearly independent vectors $\underline{v}_1, \ldots, \underline{v}_i$ with $||\underline{v}_j|| \leq \lambda_i$ for $1 \leq j \leq i$.

Special case:

λ_1 is the length of a **shortest** (in terms of Euclidean norm) non-zero vector in L.

The shortest vector need not be a basis vector!



The Gaussian Heuristic

There are many bounds on the size of the successive minima, particularly for λ_1.

The Gaussian Heuristic

Let $L \subset \mathbb{R}^n$ be a "random" full rank lattice. Then

$$\lambda_1 \approx (n/2\pi e)^{1/2} \cdot \det(L)^{1/n}.$$

NB: "*random lattice*" is not formally defined, but the ones we use in cryptanalysis can often be assumed to behave in this way.

Before the Lab

- Review this lecture, check that you understand what ECC is all about.
- Brush up your mod *p* arithmetic, revisit/learn how to do modular inversion mod *p* using Extended Euclidean Algorithm.
- Sharpen up your Python and SAGE programming skills.
- Read the lab material.
- Go program!