Practical lattice reductions

for CTF challenges

Robin Jadoul

2024/02/23

Bootcamp ICC Team Europe, Athens, Greece

Outline

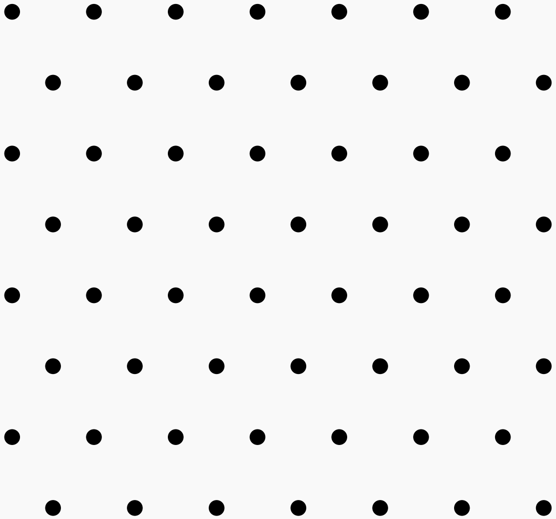
Lattices?

Lattices

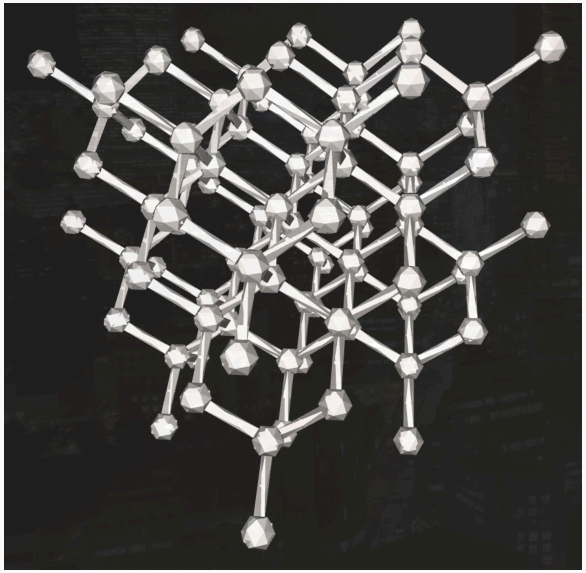
- "A lattice $\mathcal{L}$ of dimension n is a discrete additive subgroup of $\mathbb{R}^n$."
- It's a group $\Rightarrow$ addition, scalar mult
- Discrete $\Rightarrow$ we can map it to $\mathbb{Q}^n$ or $\mathbb{Z}^n$
- A group is a $\mathbb{Z}$-module, so think vector spaces
- Pick a basis $\boldsymbol{B} = (\boldsymbol{b}_1, ..., \boldsymbol{b}_n) \in \mathbb{R}^n$
  - We usually write $\boldsymbol{b}_i$ as rows of $\boldsymbol{B}$
- $\mathcal{L} = \{\sum a_i \cdot \boldsymbol{b}_i \mid \boldsymbol{a} \in \mathbb{Z}^n\}$
- Many choices of $\boldsymbol{B}$

Lattices

https://www.redwoodstore.com/spectre-fan-lattice

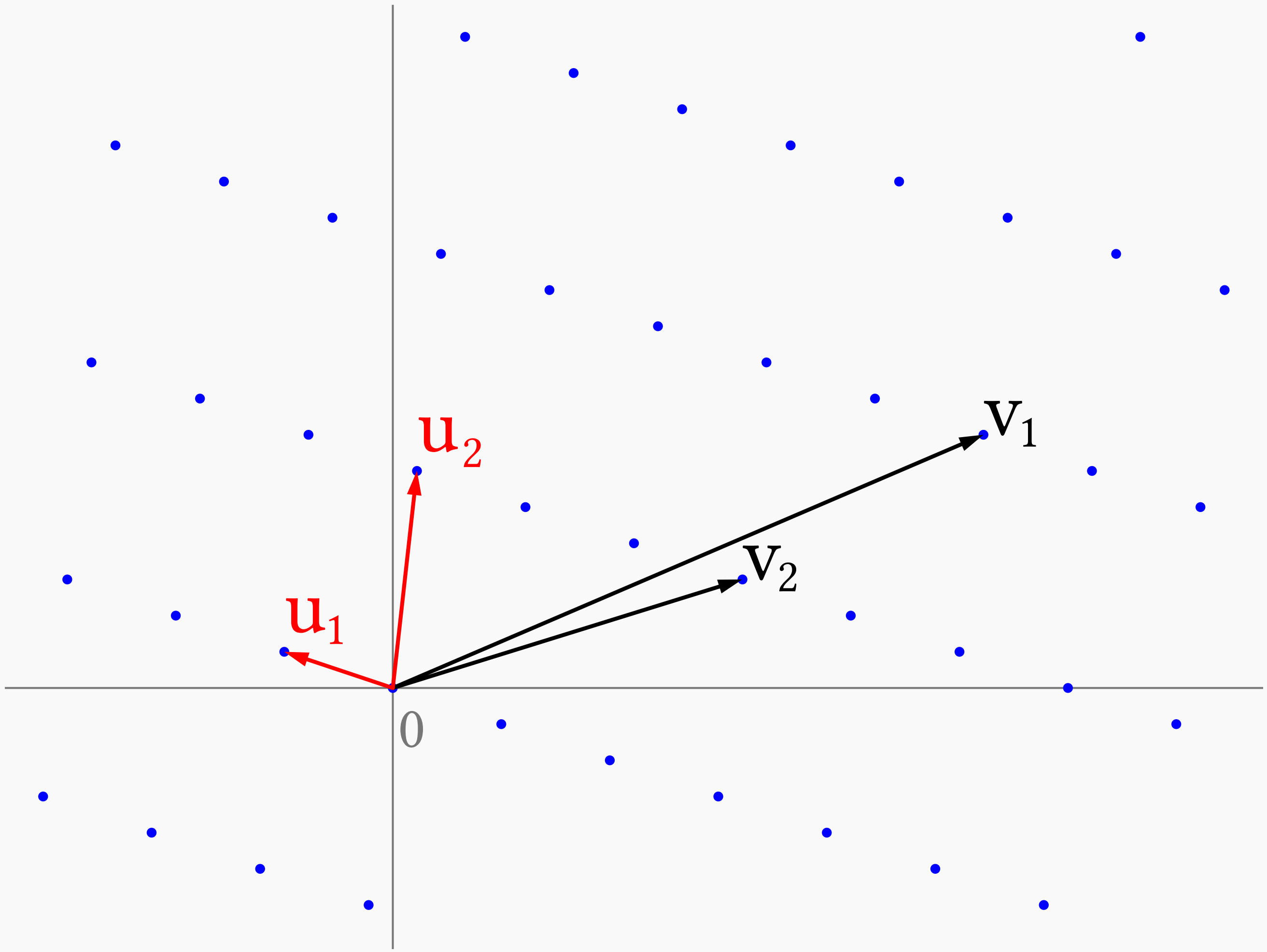


https://en.wikipedia.org/wiki/Lattice_%28group%29



https://en.wikipedia.org/wiki/File:Diamond_lattice.stl

Lattices

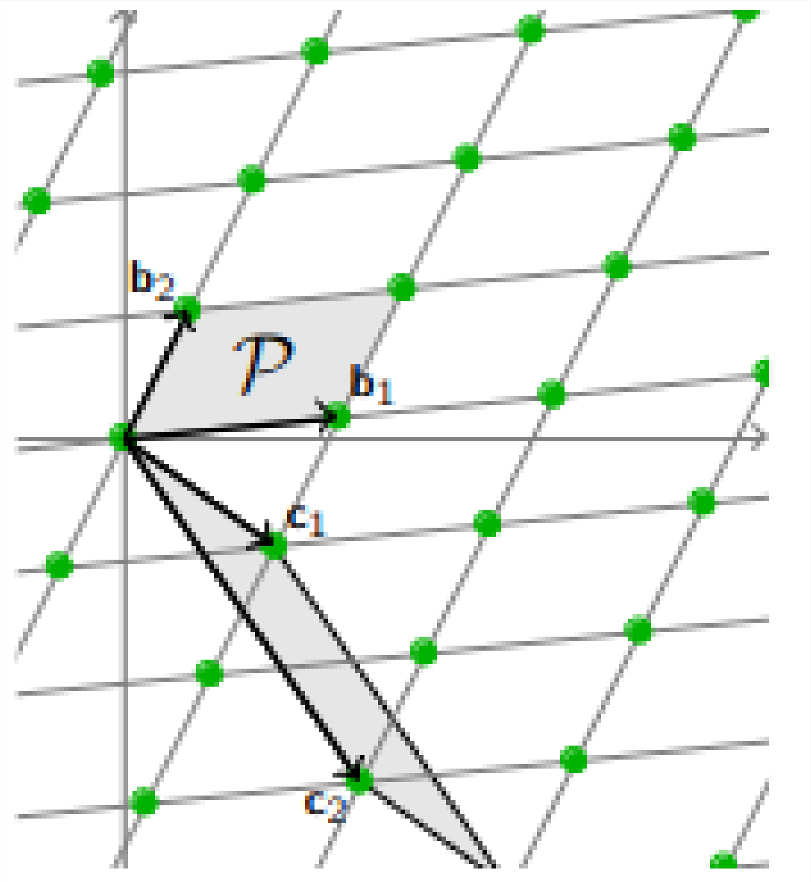


https://en.wikipedia.org/wiki/Lattice_reduction

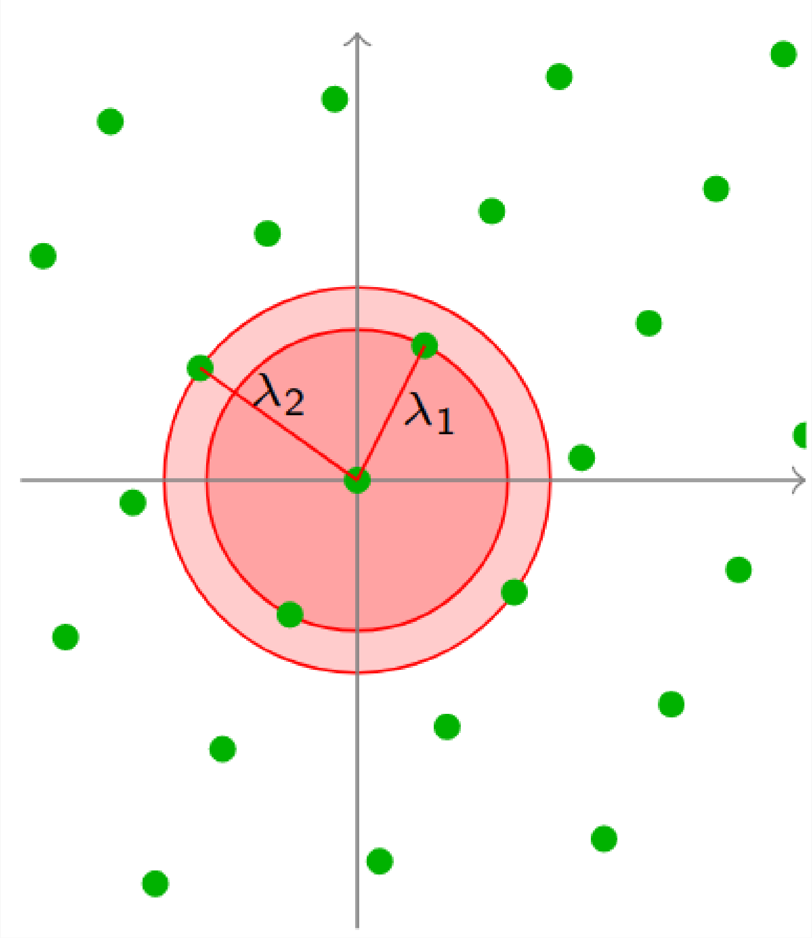
Lattice properties

- "Fundamental parallelepiped" $\mathcal{P}(\boldsymbol{B})$: a single "enclosed region"
  - $\mathcal{P}(\boldsymbol{B}) = \{\sum a_i \cdot \boldsymbol{b}_i \mid \boldsymbol{a} \in [0,1)\}$
  - $\mathbb{R}^n$ is tiled by $\mathcal{P}(\boldsymbol{B})$
- $\det(\mathcal{L}) = \mathrm{vol}(\mathcal{P}(\boldsymbol{B})) = |\det(\boldsymbol{B})|$
  - Invariant, independent of $\boldsymbol{B}$
  - Base change: invertible and unimodular
- Successive minima: $\lambda_i(\mathcal{L})$
  - $\lambda_1(\mathcal{L})$ length of shortest vector
  - $\lambda_1(\mathcal{L}) \leq \sqrt{n}|\det(\mathcal{L})|^{\frac{1}{n}}$
  - $\mathrm{GM}(\lambda_1, \ldots, \lambda_n) = \left(\prod \lambda_i\right)^{\frac{1}{n}} \leq \sqrt{n}|\det(\mathcal{L})|^{\frac{1}{n}}$
- Distance $\mu(\boldsymbol{t}, \mathcal{L}) = \min_{\boldsymbol{v} \in \mathcal{L}} \|\boldsymbol{t} - \boldsymbol{v}\|$

Lattice properties



https://simons.berkeley.edu/sites/default/files/docs/14953/intro.pdf



https://simons.berkeley.edu/sites/default/files/docs/14953/intro.pdf

Sage

```
from sage.modules.free_module_integer import IntegerLattice
B = Matrix(QQ, [[1, 0], [0, 2]])/2
L = IntegerLattice(B.denominator() * B, lll_reduce=False)
# Warning, may be slow :)
L.shortest_vector() # (1, 0)
L.closest_vector((123/42, 345/12)) # (3, 28)

L.volume() # 2
```

Why lattices?

Why lattices? Part 1: construction

- Many things in lattices are hard
- SVP: “shortest vector problem”
- CVP: “closest vector problem”
- SIS: “short integer solutions”
- $\Rightarrow$ build trapdoors, e.g. LWE
- Hopefully post-quantum too
- See later

Why lattices? Part 2: destruction

- Many things are “small”
- Many things are discrete
- e.g. some instances of integer programming
- RSA: $pq - \varphi(pq) = p + q - 1$ is “small” wrt. pq
- Breaking lattice schemes
- Generally: linear structure
- Think about **linear systems** with small solutions

Why lattices? Part 2: destruction

- Known to break many crypto “weaknesses”
- Some bias in your RNG? Lattices will break it.
- Chose your RSA private key wrong? Lattices will break it.
- Lost some precision in your floating points calculations? Lattices might help.
- Some think they even might break factoring :)

How lattices?

Improving a basis

- Starting point: *some* basis $\boldsymbol{B}$
- Goal: *good* basis $\boldsymbol{B}'$
- But what is good?
- And how do we find it?

Good lattices

- Goal: find a better basis
- Good basis?
  - Shorter basis vectors
  - Close to orthogonal
  - Find some short vectors
- Great basis?
  - Read off $\lambda_1(\mathcal{L})$
  - Read off all $\lambda_i(\mathcal{L})$?

Intuition from linear algebra

- We know $\det(\mathcal{L})$ is constant
- Want: shorter $\boldsymbol{b}_i$
- So we need wider angles between all $\boldsymbol{b}_i$ to have more area
- Hence, more orthogonal

- Gram-Schmidt orthogonalization
- But breaks the lattice
- Use it as a guideline

Lattice reduction algorithms

- LLL (Lenstra–Lenstra–Lovász)
  - Polynomial time $\mathcal{O}\left(n^6 \log^3 \|\boldsymbol{B}\|_\infty\right)$
  - $\|\boldsymbol{b}_1'\| \leq 2^{\frac{n-1}{2}} \lambda_1(\mathcal{L})$
- HKZ (Hermite–Korkine–Zolotarev)
  - Exponential time
  - $\|\boldsymbol{b}_1'\| = \lambda_1(\mathcal{L})$
- BKZ (Block (H)KZ)
  - Parametrized by block size β
  - Larger β: slower
  - Smaller β: worse basis
- Sieving and other costly approaches

Basis reduction in 2D

- In two dimensions, exact is easy
- Provides some basic intuition for LLL
- Looks like GCD

```
def gauss_reduction(v1, v2):
  while True:
      if v2.norm() < v1.norm():
          v1, v2 = v2, v1 # swap step
      m = round( (v1 * v2) / (v1 * v1) )
      if m == 0:
          return (v1, v2)
      v2 = v2 - m*v1 # reduction step
```

A brief look at LLL

```python
def LLL(B, delta):
    Q = gram_schmidt(B)

    def mu(i,j):
        v = B[i]
        u = Q[j]
        return (v*u) / (u*u)

    n, k = B.nrows(), 1
    while k < n:

        # length reduction step
        for j in reversed(range(k)):
            if abs(mu(k,j)) > .5:
                B[k] = B[k] - round(mu(k,j))*B[j]
                Q = gram_schmidt(B)

        # swap step
        if Q[k]*Q[k] >= (delta - mu(k,k-1)**2)*(Q[k-1]*Q[k-1]):
            k = k + 1
        else:
            B[k], B[k-1] = B[k-1], B[k]
            Q = gram_schmidt(B)
            k = max(k-1, 1)

   return B
```

Lattice tips and tricks

Weights

$$\boldsymbol{z} = a_1\boldsymbol{v}_1 + \ldots + a_m\boldsymbol{v}_m$$

- $m > n$
- All a_i small
- Linear system with a small solution
  - But underdetermined
  - So not just linear algebra
- (approximate) SVP would find a short solution

Weights

$$\begin{pmatrix} \boldsymbol{z} & 0 & 0 & \dots & 0 \\ -\boldsymbol{v}_1 & 1 & 0 & \dots & 0 \\ -\boldsymbol{v}_2 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -\boldsymbol{v}_m & 0 & 0 & \dots & 1 \end{pmatrix}$$

- **But**, what if we have some remaining *short* $\boldsymbol{r} = \boldsymbol{z} - \sum a_i \boldsymbol{v}_i$?
- To LLL, short is short
- Assign higher weights W to first columns
  - $\Rightarrow W\boldsymbol{r}$ not so short anymore
  - Could even vary W_i for element of $\boldsymbol{z}$
- Note: short vectors can also be the negation of what you search

Weights

$$\begin{pmatrix} W\boldsymbol{z} & 0 & 0 & \dots & 0 \\ -W\boldsymbol{v}_1 & 1 & 0 & \dots & 0 \\ -W\boldsymbol{v}_2 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -W\boldsymbol{v}_m & 0 & 0 & \dots & 1 \end{pmatrix}$$

```
z = vector(ZZ, [...])
v = Matrix(ZZ, m, n, [[...], ...])
L = Matrix.block([[Matrix(z), 0], [v, 1]])
W = Matrix.diagonal([w1, w2, ..., wn, 1, 1, ..., 1])
L = (L * W).LLL() / W
```

How to determine weights

1. Wing it and hope for the best :)
    - Works fairly often
    - Can finetune with synthetic data
    - Wiggling weights can even help finding different solutions
2. Investigate expected weights
    - Mostly when different columns have different expected weights in the target vector
    - Observe: outliers in $\|\boldsymbol{v}\| = \sqrt{v_1^2 + \ldots + v_n^2}$ weigh more
    - So the goal is: make all v_i roughly equal
        - Investigate expected result in target vector
        - Modify weights per column so target vector is all 1 (or arbitrary constant like 2^{128})

CVP

- Sometimes, switch view to CVP
- Rather than solving a linear system, you're close to some lattice point
- e.g. integer multiple + random noise (see LWE later)
- So looking for a lattice vector close to our target
  - Either close vector is final goal
  - Or just solve with linear algebra after
- If you have a range of values, put the target in the center

Kannan embedding

$$\begin{pmatrix} \boldsymbol{B} & 0 \\ \boldsymbol{t} & q \end{pmatrix}$$

- Embed CVP into an SVP instance
- Likely close to what you started with
- Short vector: $(\boldsymbol{t} - \boldsymbol{B}\boldsymbol{c}, q)$
- $q \sim$ a weight, matters for results

How to CVP

Babai’s closest plane

- Uses a reduced basis for the original lattice
- Greedy algorithm
- Iteratively project each coordinate onto the closest hyperplane
- In sage, over $\mathbb{Q}$: GS step is generally slow
    - Exact numbers that grow fast-ish

```
def Babai_CVP(mat, target):
    M = IntegerLattice(mat, lll_reduce=True).reduced_basis
    G = M.gram_schmidt()[0]
    diff = target
    for i in reversed(range(G.nrows())):
       diff -=  M[i] * ((diff * G[i]) / (G[i] * G[i])).round()
    return target - diff
```

Try different reductions

- When LLL is fast enough, but gives no results
- Consider trying BKZ instead
- Experiment with block size β, synthetic data is good
- $(\beta = n) \equiv \text{HKZ}$
- For more speed (especially coppersmith): consider `flatter`

Magic tricks for fast exploration

- Got a linear system and some bounds?
- Why not try asking nicely
- `rkm0959` made a tool/library: rkm0959/Inequality_Solving_with_CVP
- Could even make a wrapper for convenience
- Good for first exploration, not always foolproof

Enumeration

- Your target is not always the shortest
- Or even in the basis for that matter
- It's still short though
  - It's a small linear combination of basis vectors
  - Try bruteforce
  - Or random combinations
- fp(y)lll also has structured enumeration
- Optionally with extra pruning
- badly documented
  - https://fpylll.readthedocs.io/en/latest/modules.html

fpylll enumeration

```python
from fpylll import IntegerMatrix
from fpylll.fplll.gso import MatGSO
from fpylll.fplll.enumeration import (Enumeration,
                                       EvaluatorStrategy)
A = IntegerMatrix.from_matrix(M.LLL())
count = 2000
G = MatGSO(A)
G.update_gso()
enum = Enumeration(G, nr_solutions=count,
          strategy=EvaluatorStrategy.BEST_N_SOLUTIONS)
n = M.nrows()
for vec, length in enum.enumerate(0, n, max_dist, 0, t):
  ...
```

Polynomials

- Polynomials form a vector space
  - If degree is bounded/fixed
  - Over $\mathbb{R}$ or otherwise
- So we can take a discrete additive subgroup of them
- Look, ma, it's a lattice!
- Basis for coppersmith's method and ring-LWE (see later)

Codes

- Hmm, I have a lattice over $\mathbb{F}_2$
- While that may be true, "small" mostly breaks down
- Have a look at coding theory instead
- Techniques like *ISD* can be very powerful here
- Can also apply over
  - $\mathbb{F}_3$ or other small fields
  - Fields of small characteristic ($\mathbb{F}_{2^k}$ etc)

Common lattice problems

Making things modular

- Instead of working over $\mathbb{Z}$, we now want $\mathbb{Z}/q\mathbb{Z}$
- Keep thinking about linear systems
- $\sum a_i x_i \equiv y \pmod q$
- $\sum a_i x_i = y + kq$
- Repeat a few times
- Stack $q \cdot I_m$ under your matrix

Knapsack and Subset Sum

- Given:
  - A set $S = \{s_1, ..., s_n\}$
  - A value $v = \sum b_i s_i$, with $b_i \in \{0, 1\}$
- Find appropriate b_i
- Often called a knapsack problem
- More accurately it's a subset sum problem
  - there are no values attached
- Known public key cryptosystem
  - Merkle-Damgård
  - Broken by lattices (low density)

Knapsack and Subset Sum

$$\begin{pmatrix} v & 0 & 0 & \dots & 0 \\ -s_1 & 1 & 0 & \dots & 0 \\ -s_2 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -s_n & 0 & 0 & \dots & 1 \end{pmatrix}$$

- Short vector $(0, b_1, b_2, \dots, b_n)$
- Rephrase as CVP
  - Leave out first row
  - Target: $(v, 0, 0, \dots, 0)$
- Optimize CVP:
  - Want $b_i \in \{0, 1\}$, so centered around $\frac{1}{2}$
  - Can do the same trick in the original lattice

Knapsack and Subset Sum

Think about it:

- What about a more general version
  - $b_i \in \mathcal{X}$
  - Knapsack: optimize for some value t_i
  - Dealing with negative numbers
  - Parallel instances
  - Modular
  - …
- *Hidden* Subset Sum problem

- Don't forget to look at the negatives in your reduced basis!

Approximate GCD

- Given: samples $x_i = q_i p + r_i$, with small r_i
- Target: Find p, the gcd of the samples, up to errors r_i
- *Partial* AGCD: $r_0 = 0$
  - i.e. $x_0 = q_0 p$
  - e.g. RSA with extra information

AGCD: SDA

- $\frac{x_i}{x_0} \approx \frac{q_i}{q_0}$
- Find candidate q_0
- Recover p from $x_0, q + 0$
- Short vector: $(Wq_0, q_0 r_1 - q_1 r_0, \ldots)$

$$\begin{pmatrix} W & x_1 & x_2 & \ldots & x_n \\ 0 & x_0 & 0 & \ldots & 0 \\ 0 & 0 & x_0 & \ldots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \ldots & x_0 \end{pmatrix}$$

AGCD: Orthogonal lattice

- Orthogonal lattice
  - $\mathcal{L}^{\perp} = \{\boldsymbol{v} \mid \forall b \in \mathcal{L}, \langle \boldsymbol{v}, \boldsymbol{b} \rangle = 0\}$
  - Observe: $\mathcal{L} \subseteq \left(\mathcal{L}^{\perp}\right)^{\perp}$
  - Useful for some problems, including hidden subset sum
- General idea: find short vector *orthogonal* to some target
  - Often just to one vector or a lattice with 2 basis vectors
  - Then derive some useful quantity
- $\mathcal{L}(\boldsymbol{q}, \boldsymbol{r})^{\perp} \subseteq \mathcal{L}(\boldsymbol{x})^{\perp}$, and short
- reduce $\mathcal{L}(\boldsymbol{x})^{\perp}$ to find a sub-basis spanning $\mathcal{L}(\boldsymbol{q}, \boldsymbol{r})$
- recover $\boldsymbol{q}, \boldsymbol{r}$

Hidden Number Problem

$$\alpha_i x + \rho_i k_i \equiv \beta_i \pmod N$$

- k_i bounded
- See (EC)DSA biased nonce attacks
  - $\alpha_i = -r_i$, $\rho_i = s_i$, $\beta_i = H - 2^t \text{MSB}_{\text{nonce}}$, $k_i = \text{LSB}_{\text{nonce}}$
- Key realization: $k_i \equiv \frac{\beta_i - \alpha_i x}{\rho_i}$ is bounded/small
  - Try to build the lattice ;)
  - Or read biased nonce papers
- Generalization: EHNP
  - Support multiple "holes"
  - Formulation gets complex
  - "Just" implementing the paper is feasible

Coppersmith's Method

- Shift in focus
  - No longer linear systems… one polynomial
- $f(x) \equiv 0 \pmod{N}$, monic, $x < X$ bounded
  - Or even mod $d \approx N^{\beta}$ with $d \mid N$
- Sage has `f.small_roots()`, with some parameters
  - or use implementation from `kiona`/`defund`/…
  - `flatter` usually works very well for these
- $X < N^{\frac{\beta^2}{\deg(f)} - \varepsilon}$
  - ε is a useful parameter for sage
  - Smaller ε is slower, maybe brute some bits
- Multivariate (heuristic) generalizations

Coppersmith intuition

- Generate polynomials sharing roots $\pmod N$
  - $x^k f(x)$
  - $N^k f(x)$
  - $f^k(x)$
  - $\Rightarrow x^i N^j f^k(x)$
- Find small f' over $\mathbb{Z}$
  - Lattice reduction
  - Factor over $\mathbb{Z}$
  - Check results $\pmod N$
- Multivariate
  - Extract roots from multiple polynomials
  - Gröbner basis, resultants, …

Coppersmith attacks

- RSA: stereotyped message
  - $f(x) = (K + x)^e - c \equiv 0 \pmod N$, x small
- RSA: partially known factor
  - $f(x) = \left(p_{\text{high}} + x\right) \equiv 0 \pmod p$, $x < N^{\frac{1}{4}}$, $p \mid N$
- Boneh-Durfee
  - $f(x, y) = x((N + 1) - y) \equiv 0 \pmod e$
  - $y = -(p + q)$, x modular "wraps"
- AGCD
  - Multivariate

Building with lattices

Learning With Errors

$$\begin{aligned} \boldsymbol{s} &\leftarrow \chi_{\mathrm{k}}^{n} \\ \boldsymbol{a_i} &\leftarrow \mathbb{Z}_q^n \\ e_i &\leftarrow \chi_{\mathrm{e}} \\ b_i &= \langle \boldsymbol{s}, \boldsymbol{a}_i \rangle + e_i \end{aligned}$$

$$\boldsymbol{b} = \boldsymbol{A}\boldsymbol{s} + \boldsymbol{e}$$

LWE

- Distinguishing
- Key recovery
- Embedding messages
  - $b = \langle \boldsymbol{s}, \boldsymbol{a} \rangle + e + \frac{q}{p} \cdot m$
  - $b = \langle \boldsymbol{s}, \boldsymbol{a} \rangle + p \cdot e + m$

Ring-LWE

$$
\begin{aligned}
R &= \mathbb{Z}[X]/f(X), f \text{ monic irreducible}, \deg(f) = N \\
R_q &= R/qR \\
s(X) &\leftarrow \chi_{\mathrm{k}}^N \in R_q \\
e(X) &\leftarrow \chi_{\mathrm{e}}^N \in R_q \\
b_i(X) &= a_i(X) \cdot s(X) + e_i(X)
\end{aligned}
$$

Ring-LWE

$$b_i(X) = a_i(X) \cdot s(X) + e_i(X)$$

$$\boldsymbol{b_i} = \begin{pmatrix} a_{i,1} & (X^{-1}a)_1 & \dots & (X^{-N+1}a)_1 \\ a_{i,2} & (X^{-1}a)_2 & \dots & (X^{-N+1}a)_2 \\ \vdots & \vdots & \ddots & \vdots \\ a_{i,N} & (X^{-1}a)_N & \dots & (X^{-N+1}a)_N \end{pmatrix} \cdot \boldsymbol{s} + \boldsymbol{e_i}$$

NTRU

$$R = \mathbb{Z}[X]/(X^N \pm 1)$$

$$R_q = R/qR$$

$$f \leftarrow \chi^N \in R_q, \exists f^{-1}$$

$$g \leftarrow \chi^N \in R_q$$

$$h = \frac{g}{f}$$

NTRU

- Distinguishing
- Recover f
- Embedding messages
  - $f = p \cdot f' + 1$
  - $c = \frac{g}{f} + \frac{q}{p} \cdot m$
  - $c \cdot f \equiv g + \frac{q}{p} \cdot m \cdot p \cdot f' + \frac{q}{p} \cdot m$
- Alternative:
  - $h = p \cdot \frac{g}{f}, c = r \cdot h + m$
- Parameters matter: NTRU fatigue/overstretched NTRU

NTRU lattice

$$\begin{pmatrix} 1 & h \\ 0 & q \end{pmatrix}$$

- Matrix form for h (and 0, 1, q)
    - (anti-)circulant matrix
- short vector: $(f, g) = f \cdot (1, h) + k \cdot (0, q)$

Estimating difficulty

- https://github.com/malb/lattice-estimator
  - Viability
  - Ideas for attacks to look at
  - Making sure your own lattices are safe?
- https://github.com/WvanWoerden/NTRUFatigue
  - Specifically for NTRU
  - Fatigue/overstretched regime

Breaking things

Linear algebra

- Basis is linear algebra + noise
- Sometimes the noise is not there
- Or not enough
- So just throw matrices at it

Lattice reduction

- AKA primal attack
- The straightforward thing

Weak structure

- RLWE/NTRU/… in a weird ring
  - Composite modulus
  - Reducible polynomial
  - …
- Chinese remainder theorem
  - AKA working mod a factor
- Depends on end goal

Breaking things

Linearization

- Arora-Ge attack
- Consider $e \in \{-1, 0, 1\}$
- Write $v = \langle \boldsymbol{s}, \boldsymbol{a} \rangle - b$
- $v \cdot (v - 1) \cdot (v + 1) \equiv 0$
- Max degree: 3
- Enough samples $\rightarrow$ each monomial becomes 1 linear variable
- Linear algebra

Some resources

- https://eprint.iacr.org/2023/032.pdf
- https://eprint.iacr.org/2020/1506.pdf
- https://kel.bz/post/lll/
- https://github.com/rkm0959/Inequality_Solving_with_CVP
- https://github.com/jvdsn/crypto-attacks
- https://github.com/kionactf/coppersmith
- https://gist.github.com/RobinJadoul/796857fa33b118c17a4e54ff1b7ccfbe
- https://doi.org/10.1007/3-540-44670-2_12

Get your hands dirty

ImaginaryCTF (round 25)

A multiplicative knapsack, kinda.

Author Robin_Jadoul
Flag format `ictf{...}`

ImaginaryCTF 2023

`tan(x)` is a broken hash function in terms of collision resistance and second preimage resistance. But you surely can't find the preimage of `tan(flag)`, right?

Author maple3142
Flag format `ictf{...}`

ImaginaryCTF 2023

I threw in a bit of source-given rev, because why not.

> I hate crypto and rev because both are math

Sorry to people who feel like this and even say so in the ictf discord ;)

Author Robin_Jadoul
Flag format `ictf{...}`

ECSC 2023

Champagne for my real friends, real pain for my sham friends.

Author Robin_Jadoul
Flag format `ECSC{...}`

Unbalanced

ICC 2022

I want to keep my private key small, but I've heard this is dangerous. I think I've found a way around this though!

Author jack
Flag format `ICC{...}`

pbctf 2020

I know there's a famous attack on biased nonces. Then, how about this?

Author rbtree
Flag format `pbctf{...}`
Extra note (try the harder approach, just for fun)

pbctf 2021

I came up with this fun game that only lucky people can win. Do you feel lucky?

Author UnblvR
Flag format `pbctf{...}`

SECCON finals 2022

Recently, I learned that this random number generator is called "MRG".

Author Xornet
Flag format `SECCON{...}`

SEETF 2023

How to bypass this line?

```
assert __import__('re').fullmatch(r'SEE{\w{23}}', flag:=input()) and not int.from_bytes(flag.encode(), 'big') % 13**37
```

Author Neobeo

Flag format `SEE{...}`

TSJ CTF 2022

I encrypted the flag and messages by xoring them with a random number generator again. But it should be harder to break this time.

Author maple3142
Flag format `TSJ{...}`

Random Shuffling Algorithm

HITCON CTF 2023

I think you already know what is this challenge about after seeing the challenge name :)

Author maple3142
Flag format `hitcon{...}`

Reality (remake)

<Mostly new>

Based upon the challenge `reality` from google ctf 2019

Author Robin_Jadoul

Flag format `flag{...}`