Using Lattices for Cryptanalysis

Nadia Heninger

UCSD

January 23, 2020

Talk outline: Breaking classical crypto with lattices

1. Knapsacks

2. NTRU

3. Univariate Coppersmith: small solutions of polynomials modulo integers
    - Breaking RSA with bad padding

4. Howgrave-Graham: solutions modulo divisors
    - RSA partial key recovery

5. Multivariate extensions
    - RSA short secret exponent
    - Approx-GCD

6. Hidden number problem
    - Breaking (EC)DSA

Warm-up 1: Solving knapsack problems with lattices

[Lagarias Odlyzko 1984]

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

Desired solution: $z_i \in \{0, 1\}$ such that $\sum_i a_i z_i = T$

Warm-up 1: Solving knapsack problems with lattices

[Lagarias Odlyzko 1984]

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

Desired solution: $z_i \in \{0, 1\}$ such that $\sum_i a_i z_i = T$

Generate lattice basis

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

A solution $\sum_i z_i a_i = T$ corresponds to a vector

$$v_z = (z_1, z_2, \ldots, 0)$$

- We know $|v_z| \leq \sqrt{n}$.
- If the a_i are large and random, then can use density argument to show that $|v_z|$ is likely shortest vector.

A few practical notes

Knapsack problem: Find $z_i \in \{0, 1\}$ such that $\sum_i a_i z_i = T$

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

- We can use weights w to try to “force” the z_i to be small.
- In the 80s when the original papers were written, they stopped at “we hope LLL will find the shortest vector”.
- Solvable dimensions were small enough that LLL usually found the shortest vector in practice. Not true anymore.

Practical note: Current feasible lattice reduction

- LLL: In practice on random lattices, get approximation factor of $(1.02)^{\dim L}$ [Nguyen Stehle]
    - 12-2019: “We were able to reduce matrices of dimension 4096 with 6675-bit integers in 4 days” [Kirchner Espitau Fouque 2019]
    - Implementation doesn’t seem to be public.

- BKZ/enumeration:
    - 2017: 250-dimensional reduced basis, pruned enumeration (from latticechallenge.org) [Aono Nguyen 2017]

fpLLL [Albrecht Bai Ducas Stehle Stevens Walter et al.] best open source implementation for LLL/BKZ

Finding small solutions to linear equations

Knapsack problem: Find $z_i \in \{0, 1\}$ such that $\sum_i a_i z_i - T = 0$

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

- We are asking for a particularly “short” integer solution to a linear equation.
- Finding *an* integer solution to the relation is trivial:
  1. If $\gcd(a_i, a_j) = 1$
  2. Then $c_1 a_i + c_2 a_j = 1$ for $c_1, c_2 \in \mathbb{Z}$.
  3. $Tc_1 a_i + Tc_2 a_j - T = 0$ is an integer solution.
- In practice, lattice algorithms are good at finding solutions we don’t want!

Warm-up 2: Lattice attacks on NTRU

[Coppersmith Shamir 1997]

Private Key

$f, g \in R_q = \mathbb{Z}_q[x]/(x^n + 1)$

$f_i, g_i \in (-1, 0, 1)$

$f(x) = f_{n-1}x^{n-1} + \cdots + f_1 x + f_0$

$g(x) = g_{n-1}x^{n-1} + \cdots + g_1 x + g_0$

Public Key

$h = gf^{-1}$

Key recovery problem: Given h, find f, g such that $fh = g$.

Warm-up 2: Lattice attacks on NTRU

[Coppersmith Shamir 1997]

Private Key

$f, g \in R_q = \mathbb{Z}_q[x]/(x^n + 1)$

$f_i, g_i \in (-1, 0, 1)$

$f(x) = f_{n-1}x^{n-1} + \cdots + f_1 x + f_0$

$g(x) = g_{n-1}x^{n-1} + \cdots + g_1 x + g_0$

Public Key

$h = gf^{-1}$

Key recovery problem: Given h, find f, g such that $fh = g$.

Let M_h be the matrix representing multiplication by h. Then

$$(f_0, \ldots, f_{n-1})M_h \bmod q \equiv (g_0, \ldots, g_{n-1})$$

If we construct the lattice basis

$$\begin{bmatrix} I_n & M_h \\ & qI_n \end{bmatrix}$$

then $(f_0, f_1, \ldots, f_{n-1}, g_0, g_1, \ldots, g_{n-1})$ is a vector in this lattice.

Lattices as a cryptanalytic tool

Many cryptanalysis problems can be formulated either as:

- Find a small solution to some polynomial/system of equations subject to some constraints, or
- Find a polynomial with small coefficients

Often these approaches are dual.

Manipulating polynomials with lattices

We have already seen a couple of representations of elements of polynomial rings (and friends):

$$f(x) = f_{n-1}x^{n-1} + f_{n-2}x^{n-2} + \cdots + f_1 x + f_0$$

Coefficient embedding:

$$(f_{n-1}, f_{n-2}, \ldots f_1, f_0)$$

Evaluation embedding:

$$(f(z_0), f(z_1), \ldots, f(z_{n-2}), f(z_{n-1}))$$

Manipulating polynomials with lattices

We have already seen a couple of representations of elements of polynomial rings (and friends):

$$f(x) = f_{n-1}x^{n-1} + f_{n-2}x^{n-2} + \cdots + f_1 x + f_0$$

Coefficient embedding:

$$(f_{n-1}, f_{n-2}, \ldots f_1, f_0)$$

Evaluation embedding:

$$(f(z_0), f(z_1), \ldots, f(z_{n-2}), f(z_{n-1}))$$

Both homomorphic under addition, so lattice preserves additive structure.

Lattices introduce new *geometric* structure (e.g. ℓ_2 norms).

Lattice algorithms give us geometric guarantees, which often do not map exactly onto algebraic structure of crypto problems.

Coppersmith's method for univariate polynomials

[Coppersmith 96]

Theorem (Coppersmith)

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

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

when $|r_i| < N^{1/d}$.

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 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: small roots r modulo N with $|r| < R$

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

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

Coppersmith's Algorithm Outline

Input: polynomial f, modulus N.
Output: small roots r modulo N with $|r| < R$

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

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

1. Ensure $Q(r) \equiv 0 \bmod N$ by construction.

   $f(r) \equiv 0 \bmod N$ and $N \equiv 0 \bmod N$ so any polynomial combination is as well. If

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

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

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

Coppersmith's Algorithm Outline

Input: polynomial f, modulus N.
Output: small roots r modulo N with $|r| < R$

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

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

1. Ensure $Q(r) \equiv 0 \bmod N$ by construction.
2. Find such a Q with $|Q(r)| < N$.

$$\begin{aligned} |Q(r)| &= |Q_d r^d + Q_{d-1} r^{d-1} + \cdots + Q_1 r + Q_0| \\ &\leq |Q_d| R^d + |Q_{d-1}| R^{d-1} + \cdots + |Q_1| R + |Q_0| \end{aligned}$$

Coppersmith's Algorithm Outline

Input: polynomial f, modulus N.
Output: small roots r modulo N with $|r| < R$

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

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

1. Ensure $Q(r) \equiv 0 \bmod N$ by construction.
2. Find such a Q with $|Q(r)| < N$.

$$\begin{aligned} |Q(r)| &= |Q_d r^d + Q_{d-1} r^{d-1} + \cdots + Q_1 r + Q_0| \\ &\leq |Q_d| R^d + |Q_{d-1}| R^{d-1} + \cdots + |Q_1| R + |Q_0| \end{aligned}$$

3. Compute integer roots of Q and output all small ones.

Concrete example of manipulating polynomials

Input: $f(x) = x^3 + f_2 x^2 + f_1 x + f_0$,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 $c_3, c_2, c_1, c_0 \in \mathbb{Z}$.

$$\begin{array}{cccccccccc}
 & c_3 & (x^3 & + & f_2 x^2 & + & f_1 x & + & f_0) \\
+ & c_2 & & & N x^2 & & & & \\
+ & c_1 & & & & & N x & & \\
+ & c_0 & & & & & & & N \\
\hline
 & & Q_3 x^3 & + & Q_2 x^2 & + & Q_1 x & + & Q_0
\end{array}$$

Concrete example of 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

$$Q(x) = c_3f(x) + c_2Nx^2 + c_1Nx + c_0N$$

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

Coefficient embedding lattice basis:

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

Then (Q_3, Q_2, Q_1, Q_0) is a vector in this lattice.

Concrete example of 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

$$Q(x) = c_3f(x) + c_2Nx^2 + c_1Nx + c_0N$$

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

We wanted to bound $|Q_3|R^3 + |Q_2|R^2 + |Q_1|R + |Q_0| < N$.
Rescale lattice basis for convenience.

$$\begin{bmatrix} R^3 & f_2R^2 & f_1R & f_0 \\ & NR^2 & & \\ & & NR & \\ & & & N \end{bmatrix}$$

We want a vector in this lattice with small ℓ_1 norm.

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. Construct lattice of scaled coefficient embedding of suitable polynomials.
3. Find short vector in lattice. If we use LLL, we want

$$|v|_1 \leq \sqrt{n}|v|_2 \leq 2^{(n-1)/4} \det L^{1/\dim L} < N$$

4. Factor polynomial corresponding to short vector to find integer roots.

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

1. Generate lattice from subset of $\langle f(x), N\rangle^k$.
2. Be clever about which of these polynomials you include in your lattice basis.
3. Allow higher degree polynomials.

- Interesting fact: The exponential approximation factor of LLL only results in a constant factor loss in the root size.

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

1. Generate lattice from subset of $\langle f(x), N \rangle^k$.
2. Be clever about which of these polynomials you include in your lattice basis.
3. Allow higher degree polynomials.

- Interesting fact: The exponential approximation factor of LLL only results in a constant factor loss in the root size.

Theorem (CHHS 2016)

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

Open problem: Eliminate other classes of approaches.

Open problem: General systematic description of which polynomials to include in basis.

Application: Breaking Textbook RSA

[Rivest Shamir Adleman 1977]

Public Key

$N = pq$ modulus

e encryption exponent

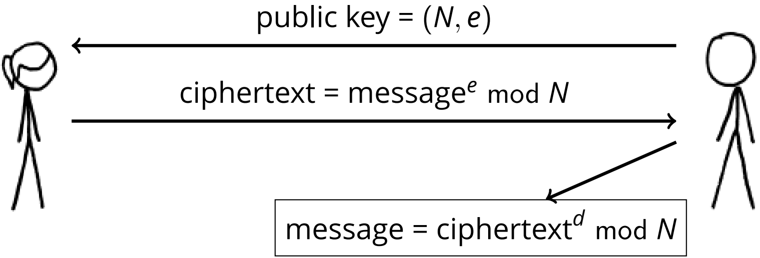
Private Key

p, q primes

d decryption exponent

$(d = e^{-1} \bmod (p-1)(q-1))$

Encryption



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'
```

Finding solutions modulo divisors

Theorem (Howgrave-Graham)

Given degree d polynomial f, integer N, we can in polynomial time find roots r modulo divisors B of N satisfying

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

for $|B| > N^{\beta}$, when $|r| < N^{\beta^2/d}$.

Proof.

Same as Coppersmith's univariate method, but find a vector in the lattice less than $N^{\beta} < B$.

□

Application: Factoring RSA with bits known

Theorem (Coppersmith)

Given half the bits (most or least significant) of a factor p, we can factor an RSA modulus $N = pq$ in polynomial time.

Proof.

Let $f(x) = x + a$ where a represents the most significant half of bits of p and r least significant bits, so $a + r = p$.

We have $f(r) \equiv 0 \bmod p > N^{1/2}$.

Apply theorem with degree $d = 1$ and $\beta = 1/2$, so $|r| < N^{\beta^2/d} = N^{1/4}$.

$\square$

```
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'
```

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
```

Partial key recovery example

Input: $f(x) = a + x, N$
Output: $r < R \quad \text{s.t.} \quad f(r) \equiv 0 \bmod p, \quad p|N, \quad p \geq N^{1/2}$

1. We chose the polynomial basis $(x + a)^2, (x + a), N$.

Partial key recovery example

Input: $f(x) = a + x, N$
Output: $r < R \quad \text{s.t.} \quad f(r) \equiv 0 \bmod p, \quad p | N, \quad p \geq N^{1/2}$

1. We chose the polynomial basis $(x+a)^2, (x+a), N$.
2. This corresponds to a lattice basis

$$\begin{bmatrix} R^2 & 2Ra & a^2 \\ 0 & R & a \\ & & N \end{bmatrix} \qquad \begin{array}{l} \dim L = 3 \\ \det L = R^3 N \end{array}$$

Partial key recovery example

Input: $f(x) = a + x, N$
Output: $r < R \quad \text{s.t.} \quad f(r) \equiv 0 \bmod p, \quad p|N, \quad p \geq N^{1/2}$

1. We chose the polynomial basis $(x+a)^2, (x+a), N$.
2. This corresponds to a lattice basis

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

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

3. LLL will find us a vector of size about $|v| \approx \det L^{1/\dim L}$.

Partial key recovery example

Input: $f(x) = a + x, N$
Output: $r < R \quad \text{s.t.} \quad f(r) \equiv 0 \bmod p, \quad p|N, \quad p \geq N^{1/2}$

1. We chose the polynomial basis $(x+a)^2, (x+a), N$.
2. This corresponds to a lattice basis

$$\begin{bmatrix} R^2 & 2Ra & a^2 \\ 0 & R & a \\ & & N \end{bmatrix} \qquad \begin{array}{l} \dim L = 3 \\ \det L = R^3 N \end{array}$$

3. LLL will find us a vector of size about $|v| \approx \det L^{1/\dim L}$.
4. The algorithm will find the root when we have

$$|Q(r)| \leq |v| \approx \det L^{1/\dim L} < p$$
$$(R^3 N)^{1/3} < N^{1/2}$$
$$R < N^{1/6}$$

We had $\lg r = 86$ and $\lg p = 512$.

Partial key recovery and related attacks

RSA particularly susceptible to partial key recovery attacks.

- Can factor given 1/2 bits of p. [Coppersmith 96]
- Can factor given 1/4 bits of d. [Boneh Durfee Frankel 98]
- Can factor given 1/2 bits of $d \bmod (p-1)$. [Blömer May 03]

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

d = random_prime(2^254)
e = inverse_mod(d,(p-1)*(q-1))
```

d is relatively small. (But not that small.)

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

d = random_prime(2^254)
e = inverse_mod(d,(p-1)*(q-1))



X = 2^764; Y = 2^254
M = matrix([[X, e*Y, -1], [0, Y*(N+1), 0], [0, 0, N+1]])

B = M.LLL()
```

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

d = random_prime(2^254)
e = inverse_mod(d,(p-1)*(q-1))


X = 2^764; Y = 2^254
M = matrix([[X, e*Y, -1], [0, Y*(N+1), 0], [0, 0, N+1]])

B = M.LLL()

sage: abs(B[0][0]/X) == d
True
```

Small RSA private exponent with lattices

Theorem (Wiener)
We can efficiently compute d when $d < N^{1/4}$.

The *RSA equation* is

$$ed \equiv 1 \bmod (p-1)(q-1)$$
$$ed = 1 + k(N - (p+q) + 1)$$

Small RSA private exponent with lattices

Theorem (Wiener)
We can efficiently compute d when $d < N^{1/4}$.

The *RSA equation* is

$$ed \equiv 1 \bmod (p-1)(q-1)$$
$$ed = 1 + k(N - (p+q) + 1)$$

Let $s = p + q$.

We would like to solve

$$ed = 1 - ks + k(N+1)$$

for d, k, s unknown.

We know $k \leq d$ and $s \approx \sqrt{N}$.

Small RSA private exponent with lattices

We would like to solve

$$ed = 1 - ks + k(N + 1)$$

for d, k, s unknown.

Can write as

$$ks + ed - 1 \equiv 0 \bmod (N + 1)$$

We would like to find small solutions $x = ks, y = d$ for

$$f(x, y) = x + ey - 1 \equiv 0 \bmod (N + 1).$$

Small RSA private exponent with lattices

Would like to solve equation

$$f(x,y) = x + ey - 1 \equiv 0 \bmod (N+1)$$

for solution $x = ks, y = d$. Bound $|d| < X$, $|ks| < Y$.

Create lattice basis

$$\begin{bmatrix} X & eY & -1 \\ & Y(N+1) & \\ & & (N+1) \end{bmatrix}$$

$$\dim L = 3$$
$$\det L = XY(N+1)^2$$

Corresponds to $x + ey - 1$, $y(N+1)$, $(N+1)$.
Lattice reduction is actually finding equation

$$dx + (ks - 1)y - d = 0$$

Theorem (Boneh Durfee)

We can efficiently compute d when $d < N^{0.292}$.

Boneh and Durfee use Coppersmith's method to find small solutions $x = k, y = (p + q)$ to

$$xy - (N + 1)x - 1 \equiv 0 \bmod e$$

Improvements: Use higher multiplicities and degree, be clever about choice of sublattice.

Open problem: Boneh and Durfee conjecture that their method can be improved to $d < N^{0.5}$.

Multivariate Coppersmith

Input: Multivariate polynomial $f(x_1, \ldots, x_m)$

Output: Integers $r_1, \ldots, r_m$ such that

$$f(r_1, \ldots, r_m) \equiv 0 \bmod N$$

Same approach works in this case, with some tweaks:

- To find solutions we solve a system of m equations taken from the short vectors in our lattice.
- May encounter algebraic independence issues: similar to Ring-LWE, additive lattice loses information about multiplicative structure of ideal.
- Theorems are generally heuristic; no totally generic solution is possible.
- Results are more ad hoc in general.

Open problem: Give a useful characterization of when multivariate Coppersmith method works.

Application: Approximate common divisors

[van Dijk Gentry Halevi Vaikuntanathan 2010]

Input: $a_1 = q_1 p + r_1, \ldots, a_m = q_m p + r_m$
(1-d Ring-LWE over $\mathbb{Z}$)

Problem: Find p, or equivalently the r_i.

Application: Approximate common divisors

[van Dijk Gentry Halevi Vaikuntanathan 2010]

Input: $a_1 = q_1 p + r_1, \ldots, a_m = q_m p + r_m$
(1-d Ring-LWE over $\mathbb{Z}$)

Problem: Find p, or equivalently the r_i.

Multivariate Coppersmith-type cryptanalysis:

1. Input $f_1(x) = a_1 - x_1, \ldots f_m(x) = a_m - x_m$.
2. Construct a lattice of polynomial combinations.
3. Find m short multivariate polynomials in this lattice.
4. Find the common roots.

- Works for some parameters, but fails for small p due to approximation factor of lattice reduction.
- Can be adapted to Ring-LWE, but results in huge-dimensional lattices.

Open problem: Is there some way to adapt Coppersmith-type amplification (multiplicity, higher degree) to Ring-LWE setting in a feasible way?

The hidden number problem

[Boneh Venkatesan 96]

Secret: Integer α. **Public parameter:** Integer n

Input: Pairs (t_i, a_i) where a_i are most significant bits of $t_i\alpha \bmod n$.

Desired Output: α

The hidden number problem

[Boneh Venkatesan 96]

Secret: Integer α. **Public parameter:** Integer n

Input: Pairs (t_i, a_i) where a_i are most significant bits of $t_i\alpha \bmod n$.

Desired Output: α

Can formulate system of equations in unknowns $r_1, \dots, r_m, \alpha$:

$$\begin{aligned} r_1 - t_1\alpha + a_1 &\equiv 0 \bmod n \\ r_2 - t_2\alpha + a_2 &\equiv 0 \bmod n \\ &\vdots \\ r_m - t_m\alpha + a_m &\equiv 0 \bmod n \end{aligned}$$

Here the r_i are small.

Solving the hidden number problem with CVP

Input:
$$r_1 - t_1\alpha + a_1 \equiv 0 \bmod n$$
$$\vdots$$
$$r_m - t_m\alpha + a_m \equiv 0 \bmod n$$

in unknowns $r_1, \dots, r_m, \alpha$, where $|r_i| < R$.

Construct the lattice basis

$$M = \begin{bmatrix} n & & & & \\ & n & & & \\ & & \ddots & & \\ & & & n & \\ t_1 & t_2 & \dots & t_m & \end{bmatrix}$$

Solve CVP with target vector $v_t = (a_1, a_2, \dots, a_m)$.

$v_k = (r_1, r_2, \dots, r_m)$ will be a close vector in this lattice.

SVP embedding

LLL, BKZ implementations easier to use as a black box than trying to implement CVP.

Input:

$$\begin{aligned} r_1 - t_1\alpha + a_1 &\equiv 0 \bmod n \\ &\vdots \\ r_m - t_m\alpha + a_m &\equiv 0 \bmod n \end{aligned}$$

in unknowns $r_1, \dots, r_m, \alpha$, where $|r_i| < R$.

Construct the lattice basis

$$M = \begin{bmatrix} n & & & & & \\ & n & & & & \\ & & \ddots & & & \\ & & & n & & \\ t_1 & t_2 & \dots & t_m & R/n & \\ a_1 & a_2 & \dots & a_m & & R \end{bmatrix}$$

$v_r = (r_1, r_2, \dots, r_m, R\alpha/n, R)$ is a short vector in this lattice.

SVP embedding

Construct the lattice

$$M = \begin{bmatrix} n & & & & & \\ & n & & & & \\ & & \ddots & & & \\ & & & n & & \\ t_1 & t_2 & \dots & t_m & R/n & \\ a_1 & a_2 & \dots & a_m & & R \end{bmatrix}$$

Want vector
$v_r = (r_1, r_2, \dots, r_m, R\alpha/n, R)$

We have:

- $\dim L = m + 2$ $\qquad$ $\det L = R^2 n^{m-1}$
- Ignoring approximation factors, LLL or BKZ will find a vector
$$|v| \leq (\det L)^{1/\dim L}$$
- We are searching for a vector with length $|v_r| \leq \sqrt{m+2}B$.
- Thus we expect to find v_r when
$$\log R \leq \lfloor \log n(m-1)/m - (\log m)/2 \rfloor$$

Solving the hidden number problem with lattices

We expect to find v_r when

$$\log R \leq \lfloor \log n(m-1)/m - (\log m)/2 \rfloor$$

Boneh and Venkatesan are interested in the limiting behavior:

Works for $m = \sqrt{\log n}$ and revealing $\sqrt{\log n}$ bits.

Possibly dumb but open question: Using higher multiplicities here doesn't improve the determinant bound. Why not?

Application: (EC)DSA Key Recovery

Global Parameters Group of order n with generator G.

Private Key Integer d

Public Key $Q = dG$

Signature Generation

Message Hash: h

Per-Signature “nonce”: Integer k

Signature on h: (r, s) $\quad r = x(kG) \quad s = k^{-1}(h + dr) \bmod n$

Application: (EC)DSA Key Recovery

Global Parameters Group of order n with generator G.

Private Key Integer d

Public Key $Q = dG$

Signature Generation

Message Hash: h

Per-Signature “nonce”: Integer k

Signature on h: (r, s) $\quad r = x(kG) \quad s = k^{-1}(h + dr) \bmod n$

Hidden number problem application:

Input k_i with known MSBs (assume 0 wlog, so k_i are “small”).

HNP instance:

$$
\begin{aligned}
k_1 - s_1^{-1} r_1 d - s_1^{-1} h_1 &\equiv 0 \bmod n \\
k_2 - s_2^{-1} r_2 d - s_2^{-1} h_2 &\equiv 0 \bmod n \\
&\vdots \\
k_m - s_m^{-1} r_m d - s_m^{-1} h_m &\equiv 0 \bmod n
\end{aligned}
$$

More Hidden Number Problem Open Problems

Open problem: There is also a Fourier analysis algorithm for the hidden number problem but it requires many more samples. Is there a smooth tradeoff that can be characterized between these two algorithms?

Open problem: The original Boneh Venkatesan application was to hardcore bits in Diffie-Hellman, but to my knowledge nobody has ever found a realistic scenario where this could be applied in the wild.

Summary

Numerous lattice constructions for cryptanalysis.

Open problem: Many of these applications feel like a “black art”. Is there a systematic way to characterize when various techniques work without manual calculation for every application? Examples:

- When does the approximation factor for LLL/BKZ matter and when does it not?
- When is the coefficient embedding better than evaluation? (It makes a small difference sometimes in practice.)
- When do amplification techniques like multiplicity work?
- Which polynomials in your ideal do you include in your lattice basis?


LLL ALL THE
KEYS!