Lattices

A **Lattice** is a discrete subgroup of the additive group of n-dimensional space $\mathbf{R}^n$.

Lattices have many uses in cryptography. They may be used to define cryptosystems and to break other ciphers. They can attack RSA with poor parameter choices.

Let L be a lattice. The definition says that it has these properties:

- If $x \in L$, then $-x \in L$.
- 0 is the identity for L.
- If $x \in L$ and $y \in L$, then $x + y \in L$.
- There is a small n-dimensional ball around the origin which contains no nonzero element of L.

The elements of L are vectors.

The *norm* of a vector is its length.

There exists a vector v_1 in L with minimum norm, the shortest vector. Let $\lambda_1(L)$ be the norm of v_1. Every vector $w \in L$ which is linearly dependent on v_1 must be $w = nv_1$ for some integer n.

A *basis* for a lattice L is a set of linearly independent vectors v_1, ..., v_r such that any vector in L can be written as a linear combination $a_1v_1 + \cdots + a_rv_r$ of the basis vectors with INTEGER coefficients a_i.

Every lattice has a basis. Every basis of a given lattice L has the same size r, called the *rank* of L. The rank r is the same as the dimension of the vector space (a subspace of $\mathbf{R}^n$) spanned by any basis of L.

Alternate definition of a lattice: The lattice generated by linearly independent vectors v_1, ..., v_r is the set of all linear combinations $a_1v_1 + \cdots + a_rv_r$ with INTEGERS a_i.

Recall that if v_1, ..., v_r and w_1, ..., w_r are two bases for the same vector space, then each w_i can be written as a linear combination of the v_j.

Likewise, if v_1, ..., v_r and w_1, ..., w_r are two bases for the same lattice, then each w_i can be written as a linear combination of the v_j with INTEGER coefficients.

Write each basis as the row vectors of an $r \times n$ matrix. The rows of B are v_1, ..., v_r and those of B' are w_1, ..., w_r.

There is an $r \times r$ matrix U that changes basis: $B' = UB$. Likewise, there is a matrix U' with $B = U'B'$. Since $B' = UU'B'$, U and U' are inverses: $UU' = I$. Therefore $\det(U) = 1/\det(U')$.

But U and U' have integer coordinates, so each must have determinant 1 or -1, the only two integers whose reciprocal is an integer.

A *unimodular* matrix is one with determinant ± 1.

It is easy to show that

$$\det(B'B'^T) = \det(BB^T) > 0$$

when B and B' are two bases for L. Define the *determinant* of a lattice L as

$$\det(L) = \sqrt{\det(BB^T)},$$

where B is any basis of L.

If $v = (v_1, v_2, \ldots, v_n)$ and $w = (w_1, w_2, \ldots, w_n)$ are vectors, then their dot product or scalar product is the sum

$$v \cdot w = v_1 w_1 + v_2 w_2 + \cdots + v_n w_n.$$

Note that for any vector v, $v \cdot v \geq 0$ since it is the sum of squares.

The *norm* or *length* of a vector v is $||v|| = \sqrt{v \cdot v}$. The zero vector 0 is the only vector with length 0.

In a lattice L there is a shortest positive length of a nonzero vector. This length is denoted $\lambda_1(L)$. At least two vector have this minimum positive length since the norm of $-v$ equals the norm of v.

We generalize $\lambda_1(L)$ as follows. Let $\lambda_k(L)$ be the smallest positive real number so that there is at least one set of k linearly independent vectors of L, with each vector having length $\le \lambda_k(L)$.

This defines a sequence

$$\lambda_1(L) \le \lambda_2(L) \le \lambda_3(L) \le \cdots .$$

Note that we do not count vectors in this definition.

M's Theorem. For every integer $r > 1$, there is a positive constant γ_r so that for every lattice L of rank r and for all $1 \le k \le r$, we have

$$\left(\prod_{i=1}^{k} \lambda_i(L)\right)^{1/k} \le \sqrt{\gamma_r}\,\mathrm{det}(L)^{1/r}.$$

A lattice is often presented by giving a basis for it. This basis may consist of very long vectors and they may be nearly parallel.

Sometimes it is more useful to have a basis with shorter vectors which are closer to being orthogonal to each other. The process of finding such a basis from a poor one is called *reducing* the lattice or *lattice reduction*.

The ideal basis $\{v_1, v_2, \ldots, v_r\}$ for L would have the length of v_i be $\lambda_i(L)$ for each i.

It turns out to be NP-hard to find the ideal basis.

The Gram-Schmidt process constructs an orthogonal basis for a vector space V, given any basis for it. It is fast and simple.

Input: A basis $B = \{v_1, v_2, \ldots, v_r\}$ for V.

Output: Orthogonal basis $B' = \{w_1, w_2, \ldots, w_r\}$ for V and a matrix M that takes B into B'.

```
for ($i = 1$ to $r$) {
        $w_i = v_i$
        for ($j = 1$ to $i - 1$) {
                $m_{i,j} = (v_i \cdot w_j)/(w_j \cdot w_j)$
                $w_i = w_i - m_{i,j} w_j$
                }
        }
```

Example of Gram-Schmidt process:

In R^4, let V be the subspace with basis $\{v_1, v_2, v_3\}$, where $v_1 = (1,2,3,0)$, $v_2 = (1,2,0,0)$ and $v_3 = (1,0,0,1)$.

We will find an orthogonal basis $\{w_1, w_2, w_3\}$ for V.

$w_1 = v_1 = (1,2,3,0)$.

$w_2 = v_2 - ((v_2 \cdot w_1)/(w_1 \cdot w_1))w_1$
$= (1/14)(9,18,-15,0)$. We simplify the calculation by replacing w_2 with $14w_2 = (9,18,-15,0)$.

$w_3 = v_3 - ((v_3 \cdot w_1)/(w_1 \cdot w_1))w_1$
$- ((v_3 \cdot w_2)/(w_2 \cdot w_2))w_2 = (1/5)(4,-2,0,5)$.
We could replace w_3 with $5w_3 = (4,-2,0,5)$.

The orthogonal basis is: $w_1 = (1,2,3,0)$, $w_2 = (9,18,-15,0)$, $w_3 = (4,-2,0,5)$.

Check: $1 \cdot 9 + 2 \cdot 18 + 3 \cdot (-15) + 0 \cdot 0 = 0$, etc.

The Gram-Schmidt process does not work for lattices because the $m_{i,j}$ are usually not integers, so the new basis vectors are not INTEGER linear combinations of the original vectors.

An ideal basis $B = \{v_1, v_2, \ldots, v_r\}$ for a lattice L would have $||v_i|| = \lambda_i(L)$. To see that one cannot achieve this goal, consider the lattice generated by the vectors $(2,0,0,0,0)$, $(0,2,0,0,0)$, $(0,0,2,0,0)$, $(0,0,0,2,0)$, $(1,1,1,1,1)$.

Show that if g, h, i, j, k are even integers, then the vector (g,h,i,j,k) is in L.

We have $\lambda_1(L) = \lambda_2(L) = \lambda_3(L) = \lambda_4(L) = \lambda_5(L) = 2$.

These minima are realized for the vectors $(2,0,0,0,0)$, $(0,2,0,0,0)$, $(0,0,2,0,0)$, $(0,0,0,2,0)$, $(0,0,0,0,2)$, which are all in L. But these vectors are not a basis for L because $(1,1,1,1,1)$ is not an INTEGER linear combination of these vectors.

There are several definitions of reduced basis and they are not equivalent.

LLL [1982] gave the following definition:

Let δ be a parameter in $1/4 < \delta < 1$. A basis $B = \{v_1, v_2, \dots, v_r\}$ of a lattice L is called δ-LLL *reduced* if the following two conditions are satisfied, where $B' = \{w_1, w_2, \dots, w_r\}$ is the result of applying Gram-Schmidt to B.

1. For all $1 \le i < j \le r$, $v_j \cdot w_i \le (w_i \cdot w_i)/2$.

2. For all $1 < i \le r$,
$\delta ||w_{i-1}||^2 \le ||w_i||^2 + (v_i \cdot w_{i-1})/||w_{i-1}||^2$.

Property 1 guarantees the length reduction of the basis.

Larger values of δ give more reduction (but the LLL algorithm runs slower).

The LLL algorithm computes δ-LLL reduced bases for lattices. The δ-LLL reduced bases it produces approximate the shortest vectors possible. There are absolute constants $c_i > 1$ such that $||v_i|| < c_i\lambda_i(L)$ for each i. The constants c_i depend only on δ and not on L.

More about the constants γ_r in Minkowski's theorem above: $\gamma_r \le (4/3)^{(r-1)/2}$ and $\gamma_r \le 1 + r/4$ for $r \ge 1$. For
$r = 1, 2, 3, 4, 5, 6, 7, 8$, we have
$\gamma_r = 1, 2/\sqrt{3}, \sqrt[3]{2}, \sqrt{2}, \sqrt[5]{8}, \sqrt[6]{64/3}\ \sqrt[7]{64}, 2$.
These numbers are approximately
1.00, 1,15, 1.26, 1.41, 1.52, 1.67, 1.81, 2.00.

Letting $k = 1$ in Minkowski's theorem gives

$$\lambda_1(L) \le \sqrt{\gamma_r}\,\det(L)^{1/r}.$$

The definition of δ-LLL *reduced* is often used with $\delta = 3/4$. With this value, the LLL algorithm produces a basis $B = \{v_1, v_2, \ldots, v_r\}$ of L having

$$||v_1|| \leq 2^{(r-1)/4} \det(L)^{1/r}.$$

Application to Knapsacks

Factoring and DLP are thought to be hard because no one has found a polynomial time algorithm to solve them. Factoring is in both NP and co-NP. It is very likely not NP-complete because this would imply co-NP = NP, which nobody believes.

Wouldn't it be great if a cipher could be based on an NP-hard problem? Merkle and Hellman made the first attempt at this goal in 1979. They used knapsacks and the subset sum problem.

The subset sum problem is NP-complete: Given a set of n positive integers $M = \{M_1, \ldots, M_n\}$ and an integer S, find a subset of M whose sum is S.

Application to Knapsacks

To build a public-key cipher, let M be public. Encipher a message $x \in \{0, 1\}^n$ of bits x_1, ..., x_n as $S = \sum_{i=1}^{n} x_i M_i = x \cdot M$.

Problem: How to decipher?

Answer: Use the greedy algorithm and a superincreasing list of integers: $r = \{r_1, \ldots, r_n\}$ with $r_{i+1} > 2r_i$ for $1 \leq i < n$. It is easy to recover x from $S = x \cdot r$.

Of course, an eavesdropper can easily recover x from $S = x \cdot r$, too. To prevent this, M and H chose two large random integers A, B with $B > 2r_n$ and $\gcd(A, B) = 1$. Keep A, B, r secret. Define the public key M by $M_i = Ar_i \bmod B$. Encrypt x as the ciphertext $S = x \cdot M$.

The recipient who knows A, B and r can decipher S by computing $S' = A^{-1}S \bmod B$. Then $S' = x \cdot r$. Recover x from S' and r by the greedy algorithm, as above.

How to choose parameters for the knapsack cipher?

There is an easy attack if r_1 is very small, so typically $r_1 \approx 2^n$ and $r_n \approx 2^{2n}$. The public key is a list of n integers of length about $2n$, so takes $2n^2$ bits.

There is a meet-in-the-middle attack on the subset sum problem, so the security of a knapsack with size n is about $O(2^{n/2})$.

Choosing $n = 160$ takes effort 2^{80} to crack and a public key of length $2n^2 = 51200$ bits.

Compare with RSA or ElGamal with public key of about 1000 bits (and effort 2^{80} to crack).

Enter LLL. Form a lattice L of dimension $n+1$ spanned by vectors $v_1 = (1, 0, 0, \ldots, 0, M_1)$, $v_2 = (0, 1, 0, \ldots, 0, M_2)$, ...,
$v_n = (0, 0, 0, \ldots, 1, M_n)$, $v_{n+1} = (0, 0, 0, \ldots, 0, S)$.

We have $\det(L) = S$.

The fact that a subset of the M_i sums to S implies that L contains a very short vector

$$t = \sum_{i=1}^{n} x_i v_i - v_{n+1} = (x_1, x_2, x_3, \ldots, x_n, 0).$$

We have $||t|| \le \sqrt{n}$ and probably $||t|| \approx \sqrt{n/2}$. On the other hand, for each i, $||v_i||$ is between 2^n and 2^{2n}.

LLL finds w_1 in L with $||w_1|| \le 2^{(n-1)/4} \det(L)^{1/n}$

$$\approx 2^{n/4} S^{1/n} \approx 2^{n/4} 2^{2n/n} = 4 \cdot 2^{n/4}.$$

w_1 is almost certainly the shortest vector in L.

Thus, LLL can easily find the shortest vector in L when $n \le 300$. For $n = 300$, the public key size is $2n^2 \approx 180000$ bits, too big.

Cryptosystems using Lattices

GGH = Goldreich, Goldwasser, Halevi.

The secret key is a special small, reduced basis R for a lattice L. The owner constructs a public key B, a random basis for L, by multiplying R by a few random unimodular matrices. Let B and R also represent the $n \times n$ matrices whose rows are the basis vectors. Let U be the product of the unimodular matrices so that $B = UR$.

A plaintext is a vector x of n integers. It is enciphered using the public key B as $e = xB + r$, where r is a random vector of n small integers. Thus, xB is in L while r and e are not in L.

If r is short enough, then xB is the closest vector in L to e.

Someone who knows the secret reduced basis R for L can compute xB from e as follows.

First compute eR^{-1} (in the vector space, not in L). Then round each component to the nearest integer. If r is sufficiently small and if R is sufficiently short and close to being orthogonal, then the result of the rounding process will be xU. Finally, get x from xU by solving a linear system of n equations in n unknowns.

Without knowledge of R, it would appear that breaking GGH was equivalent to solving a general Closest Vector Problem (CVP) in a lattice L. G, G and H conjectured in 1997 that this CVP was intractable when $n > 300$.

Later improvements in LLL allowed Nguyen in 1999 to break the GGH challenge problems with $n = 300$ and 350. Larger keys with $n > 350$ are impractical.

NTRUEncrypt, 1996 by Hoffstein, Pipher, Silverman + Lieman.

The original reason LLL developed their lattice reduction algorithm was to factor polynomials with integer or rational number coefficients. Their surprising result was that polynomials can be factored in polynomial time. Lattice reduction was just a tool to reach this goal. Since their work, many applications of lattice reduction have been discovered throughout mathematics and cryptography.

NTRUEncrypt could be broken if one could factor certain kinds of polynomials. Cryptanalysis of NTRUEncrypt usually assumes that the LLL lattice reduction gives the fastest way to factor polynomials. It is much faster than earlier known methods, and no one knows a faster way to factor polynomials.

NTRUEncrypt is faster than other public-key ciphers and has this advantage: Quantum computer algorithms are known for factoring integers and the DLP, including the ECDLP. No quantum computer algorithm is known for lattice reduction. (Suggested thesis topic.)

We describe here a very simple version of NTRUEncrypt.

NTRUEncrypt uses three integer parameters, a prime N, an odd number p and a power of 2, q. (Other choices for p and q are possible, but we must have $\gcd(N,q)=1$, $\gcd(p,q)=1$ and q much larger than p.)

We will use polynomials in $\mathbf{Z}[X]/(X^N-1)$ with convolution as multiplication:

$$f * g(x) = \sum_{k=0}^{N-1} \left(\sum_{i+j \equiv k \pmod{N}} f_i g_j \right) x^k.$$

where

$$f(x) = \sum_{i=0}^{N-1} f_i x^i \quad \text{and} \quad g(x) = \sum_{j=0}^{N-1} g_j x^j.$$

(We have already seen convolution (with $N = 4$) in AES multiplication of 32-bit words.)

Alice generates her public and private keys as follows.

She chooses two polynomials $f(x)$ and $g(x)$ of degree $\leq N-1$ with coefficients in $\{-1, 0, 1\}$ (trinary polynomials). $f(x)$ must have an inverse $f_p(x)$ modulo p and an inverse $f_q(x)$ modulo q, which may be computed by the Euclidean algorithm for polynomials. This means that $f(x) * f_p(x) = 1 \pmod{p}$ and $f(x) * f_q(x) = 1 \pmod{q}$ as polynomials in x.

Alice's private key is the pair of polynomials $f(x)$, $f_p(x)$.

Alice's public key is the polynomial

$$h(x) = p f_q(x) * g(x) \pmod{q}.$$

Encryption

Bob wants to send a secret message to Alice using NTRUEncrypt. He encodes the message as a trinary polynomial m. He chooses a random polynomial r with small coefficients, but not necessarily trinary. He computes the ciphertext as

$$e = r * h + m \pmod{q},$$

where h is Alice's public key.

The random polynomial r is discarded after encryption. Anyone who knows r can easily find m.

Decryption

Alice receives the ciphertext e. Using her private key f, she computes

$$a = f * e \pmod{q}.$$

This polynomial is

$$a = f*(r*h+m) = f*(r*pf_q*g+m) \pmod{q}$$

which reduces to

$$a = pr * g + f * m \pmod{q}$$

since $f * f_q = 1 \pmod{q}$. Next, Alice computes

$$b = a \bmod p = f * m \pmod{p},$$

since $pr * g = 0 \pmod{p}$. Finally, Alice computes

$$f_p * b = f_p * f * m = m \pmod{p},$$

since $f * f_p = 1 \pmod{p}$.

Attacks on RSA using LLL

Recall RSA: $N = pq$ hard to factor. Choose e with $\gcd(e, \phi(N)) = 1$, where $\phi(N) = (p-1)(q-1)$. Via extended Euclid, find d with $ed \equiv 1 \pmod{\phi(N)}$. Discard p and q. Public key is N, e. Private key is d. Encipher m as $c = m^e \bmod N$. Decipher c as $m = c^d \bmod N$.

1. RSA problem: Given N, e, c, find m.

2. Compute d: Given N, e, find d.

3. Factor N: Given N, find p and q.

Clearly, $3 \to 2 \to 1$.

In fact, $3 \equiv 2$. It is not known whether $3 \equiv 1$. All three problems seem hard, although Shor showed that one can solve 3 quickly on a quantum computer.

Relax the problems. Assume we know some hint about p or q or d, either because they are limited or we have an oracle for them.

Coppersmith proved that if $f(x)$ is a monic polynomial and b is an unknown factor of a given integer N, then one can find all “small” solutions x to $f(x) \equiv 0 \pmod{b}$ quickly via LLL.

Let $N = bc$, where $b > c$. Suppose we know N and a bit more than half of the high-order bits of b. (Oracle or limit on b.)

Specifically, suppose we are given N and $\tilde{b}$ with $|b - \tilde{b}| < N^{5/28}/2$.

Let $f(x) = x + \tilde{b}$. Then $x_0 = b - \tilde{b}$ is a small zero of $f(x) \equiv 0 \pmod{b}$ so we can find it with Coppersmith's method.

Define (eight) polynomials for $i = 0$ to 3 by

$$f_i(x) = N^i f^{4-i}(x) = N^i (x + \tilde{b})^{4-i}$$

$$f_{4+i}(x) = x^i f^4(x) = x^i (x + \tilde{b})^4.$$

Then b^4 divides all $f_i(x_0)$, $i = 0, 1, \ldots, 7$.

Let $X = N^{5/28}/2$.

The coefficient vector of a polynomial $h(x) = a_0 + a_1x + a_2x^2 + \cdots a_nx^n$ is the vector $v = (a_0, a_1, \ldots, a_n)$.

Define a lattice L of dimension 8 by the basis of coefficient vectors of $f_i(xX)$.

Apply LLL to get a reduced basis, including a shortest vector $v =$ the coefficient vector of a polynomial $g(xX)$.

We have $|g(x_0)| < b^4$ and b^4 divides $g(x_0)$. Therefore, $g(x_0) = 0$. Solve $g(x) = 0$ in integer x. Then $b = x_0 + \tilde{b}$.

In a similar way, one can find d, given N and e provided d is small, say, $0 < d < N^{1/4}$.

Write $ed = 1 + k\phi(N)$. Since $\phi(N) = N - (p + q) + 1$, we can write $\phi(N) = N - z$ with $z <$ a small multiple of $\sqrt{N}$. This leads to

$$ed - kN - kz - 1 = 0.$$

Since kz is much smaller than ed or kN, we have $e/N \approx k/d$. Continued fractions allow us to recover k and d from e and N when $d < \sqrt[4]{N}$.

Rewrite $ed - kN - kz - 1 = 0$ as $kN + kz + 1 \equiv 0 \pmod{e}$. A variation of Coppersmith's method using lattices and LLL lets one find solutions to this congruence with z near $\sqrt{N}$ and k near N^{α} for α up to about 0.29.