Lattice based cryptography and Learning with Errors

October 8, 2024

1 What is a lattice

A lattice in two dimensions includes probably exactly what you imagine: a Cartesian product of points, in a grid structure. In n-dimensions, a lattice is a set of points in n-dimensional space with a periodic structure. Mathematically a lattice is a n-dimensional vector space over $\mathbb{Z}$, so it is defined by a basis $b_1, \ldots, b_n \in \mathbb{R}^n$; the lattice is given by

$$\mathcal{L}(b_1, \ldots, b_n) := \{\sum_{i=1}^{n} a_i b_i : a_i \in \mathbb{Z}\}.$$

A somewhat boring example of a lattice is $\mathbb{Z}^n$.

2 One-way functions from lattices

For lattices to be a candidate for cryptography, we must be able to associate them to a one-way function. This is a function that is easy to calculate, but computing the input of a given output is hard. Ajtai REF introduced a lattice-based one-way function assuming the hardness of the Shortest Vector Problem, specfically the n^c-approximate Shortest Vector Problem.

The Shortest Vector Problem (SVP) is to find the shortest vector for a given lattice. In the α-approximate SVP, it is enough to find a vector whose length is at most a factor of α greater than the length of the shortest vector.

Ajtai's one-way function is over $\mathbb{Z}_q$; it's actually a hash function:

$$\begin{aligned} f_A\{0, \ldots, d-1\}^m &\to \mathbb{Z}_q^n \\ y &\mapsto Ay \pmod{q} \end{aligned}$$

where A is a matrix chosen uniformly randomly from $\mathbb{Z}_q^{n \times m}$. For example, we choose $d = 2$, $q = n^2$ and $m > n \log(n^2)$.

A collision (which would violate the hash function assumption) means that $Ay = Ay'$, and so $y - y'$ is a short non-zero vector of the lattice defined by $\lambda_q^{\perp}(A) := \{y \in \mathbb{Z}^m : Ay = 0 \mod q\}$.

Exercise: show that $\lambda_q^\perp(A)$ is a lattice.

Notice that the function f_A is simple to implement (because it's just matrix multiplication and modular arithmetic), but the 'key size' (the size of A) grows at least quardratically. We could make this construction more efficient by giving A a special structure.

3 LWE

3.1 Learning from parity with errors

For $n \geq 1$ and $\epsilon \geq 0$ the *learning from parity with error* problem is to find $s \in \mathbb{Z}_2^n$ given a list of 'equations with errors':

$$\begin{aligned} < s, a_1 > &\approx_\epsilon b_1 \pmod 2 \\ < s, a_2 > &\approx_\epsilon b_2 \pmod 2 \\ &\vdots \end{aligned}$$

where $a_i \in \mathbb{Z}_2^n$ is chosen uniformly and independently and $\approx_\epsilon$ means that each equation is correct independently with probability $1 - \epsilon$.

Q. How hard is this problem when $\epsilon = 0$? What value of ϵ makes this problem the hardest?

Q. Imagine that we want to just recover the first bit of s: we could pick a random set of n equations. What probability do we have that our guess for the first bit of s is correct? From this probability, how many times do we need to iterate to be confident of the first bit of s? How many times do we need to iterate to get all of s with a high confidence?

3.2 Learning with errors

We can extend the Learning from Parity with error problem in many ways: firstly, why limit ourselves to working mod 2? Instead, let's work mod p. Secondly, instead of our errors being chosen uniformly, let's choose errors from a probability distribution χ.

This gives us the *learning with error problem*: for $n \geq 1$ and $p = p(n) \leq poly(n)$ prime, find $s \in \mathbb{Z}_p^n$ given

$$\begin{aligned} < s, a_1 > &= b_1 + e_1 \pmod p \\ < s, a_2 > &= b_2 + e_2 \pmod p \\ &\vdots \end{aligned}$$

where $a_i \in \mathbb{Z}_p^n$ are chosen uniformly and independently and $e_i \in \mathbb{Z}_p$ is chosen independently according to χ. We refer to this as $\mathrm{LWE}_{p,\chi}$

Q. As a quick exercise to become more familiar with the notation, how would we write the Learning from Parity with Noise problem using the notation $\mathrm{LWE}_{p,\chi}$?

The Decision LWE Problem is as follows: given m independent samples $(\mathbf{a}_i, b_i) \in \mathbb{Z}_p^n \times \mathbb{Z}_p$, where either all the samples are of the form $b_i = \mathbf{a}_i\mathbf{s} + e_i$ (where s is the LWE secret, and e_i the error distributed according to χ), or all the samples are of the from $b_i = u_i$ for some uniformly sampled $u_i \in \mathbb{Z}_p$, decide (with non-negligible advantage) which case we are in.

The Search-LWE problem is as follows: given m independent samples $(\mathbf{a}_i, b_i) \in \mathbb{Z}_p^n \times \mathbb{Z}_p$, where $b_i = \mathbf{a}_i\mathbf{s} + e_i$ (where s is the LWE secret, and e_i the error distributed according to χ), recover $\mathbf{s}$.

- Note that without the error term, both the search and decision problems are easy.
- We can also work with a single matrix equation $(A, As + e)$ instead of considering m samples
- LWE is somewhat similar to code-based cryptography: this time, instead of recovering a codeword, we're recovering a lattice point.
- The decision and search LWE problems are equivalent (up to a polynomial difference in the size of m between the two problems)
- Although I've been writing p a prime, actually it can be a prime power (where typically the notation is $q = p^r$); typically q is a power of 2.

3.3 Ring and Module Learning with errors

We can extend this to Ring or Module LWE by working over a ring or a module instead.

A ring is a group under addition that has a 'multiplication' operation. However, unlike a field, there is not necessarily inverses for non-zero elements. For example $\mathbb{Z}[x]$, the polynomial ring over the integers, is a ring.

A module is a generalisation of a vector space. However, the field of scalars (often $\mathbb{C}$) is replaced by a ring. For example, Consider the module $\mathcal{M}$ with basis elements given by $e_1 = (x, 0), e_2 = (0, x+1)$ and scalar multiplication over $\mathbb{Z}[x]$. Then any element of $\mathcal{M}$ looks like $a_1e_1 + a_2e_2$ where $a_1, a_2 = \mathbb{Z}[x]$.

3.4 SVP

The shortest vector problem is essentially the core hard problem underlying LWE. There's actually a couple of different reductions that reduce the hardness of LWE to very similar problems.

The shortest vector problem (SVP) is to find the shortest vector in the lattice.

This is related to the approximate-SVP problem: find a short vector that is at most a (given) factor larger than the shortest vector.

Similarly, we have the shortest independent vectors problem (SIVP): find a collection of independent vectors that form a basis of the lattice that are as short as possible (in the sense that if you line up the vectors from smallest to

largest, the largest vector has a minimal size over all possible sets of independent vectors). This is related to the approximate-SIVP problem, in which you're allowed to be a given factor away from the shortest possible independent set.

The hardness of lattice based cryptogrpahy is based on these types of problems.

4 Cryptosystem

There are a number of different ways that we can encrypt using LWE. They're all somewhat similar, and the differences are usually to take advantages of various efficiencies.

4.1 Encrypting and decrypting

In this subsection, we'll describe a system that encrypts a message. The goal is for decryption to recover that same message.

Our cryptosystem is parameterised by integers n (the security parameter), m (number of equations), q (modulus), and χ (noise distribution).

- The private key is $\mathbf{s} \in \mathbb{Z}_q^n$, chosen uniformly
- The public key is $(A, \mathbf{t} = A\mathbf{s} + \mathbf{e}) \in \mathbb{Z}_q^{m \times n} \times \mathbb{Z}_q^m$, where $\mathbf{e} \in \mathbb{Z}^m$ is the error vector chosen so that each element is independently chosen randomly according to the probability distribution χ.
- We will encrypt a message in $\{0,1\}^*$. For each bit of the message, chose a random set S uniformly among all 2^m subsets of $[m]$. The encryption is $(\sum_{i \in S} \mathbf{a}_i, \sum_{i \in S} t_i)$ if the message bit is 0, and $(\sum_{i \in S} \mathbf{a}_i, \lfloor \frac{q}{2} \rfloor + \sum_{i \in S} t_i)$ if the message bit is 1. Here, $\mathbf{a}_i$ is the ith row of the matrix A, and t_i is the ith element of the vector $\mathbf{t}$.
- To decrypt a pair $(\mathbf{a}, b)$, output 0 if $b - \langle \mathbf{a}, \mathbf{s} \rangle$ is closer to 0 than to $\lfloor \frac{q}{2} \rfloor$ (modulo q) and 1 otherwise.

Note that this system is quite inefficient.

Question: how big is the public key? By how much does the ciphertext increase for each message bit that we send?

Correctness A major question that we need to ask, is whether this cryptosystem work, in the sense that decryption recovers the message.

Exercise: show that decryption works if there is no error.

For S associated to a given message bit, decryption works as long as $\sum_{i \in S} e_i \leq q/4$. Therefore, the probability of a decryption failure is the probability that this inequality is *not* satisfied. The probability of a decryption failure therefore depends on the error distribution; typically we will choose errors following a discrete Gaussian distribution and we use standard probability bounds (or even just direct calculation, depending on the size of q) to bound this probability. We want the failure probability to be small.

Security The other major question with which we must concern ourselves is whether this cryptosystem is secure, in the sense that the public data (public key and ciphertext) does not reveal anything about the message. In reality, we will be satisfied if only a negligible amount of information about the message is revealed.

We will sketch the security proof against chosen plaintext attacks.

Suppose that there exists an efficient algorithm that, given a public key $(A, \mathbf{t})$ as above, can correctly guess the encrypted bit with probability at least $\frac{1}{2} + \frac{1}{poly(n)}$. Now, let us input into the algorithm $(A, \mathbf{u})$, where $\mathbf{u} \in \mathbb{Z}_q^m$ is chosen uniformly, and a random bit encrypted using $(A, \mathbf{u})$. Then, it follows [1] that with very high probability that the distribution $(\sum_{i \in S} \mathbf{a}_i, \sum_{i \in S} t_i)$ is close to uniform. Consequently, encryptions of 0 and 1 are pretty much identically distributed and so the algorithm cannot distinguish the encrypted bit beyond random guessing. So either the algorithm *can* correctly guess the value of an encrypted bit (if it's given a 'proper' sample) or it cannot (if it's given a random sample); this means it can distinguish LWE samples from uniform samples. This violates the decision LWE problem.

4.2 Key Encapsulation - FrodoKEM

Encapsulation is very related to encryption: we want two (or more) parties to agree a random secret. For example, Alice could send to Bob an encrypted secret that she has chosen. However, when encapsulating in general, Alice does not necessarily have control over the randomness generated. As typical encryption involves first using public key cryptography to exchange a key, and then symmetric cryptography.

The Key Encapsulation Mechanism (KEM) that we describe is basically FrodoKEM. It is defined over $\mathbb{Z}_q$, where $q = 2^N$ is a power of 2.

We'll first introduce some notation: let $r \to \chi^{n \times m}$ to denote that every element of $r \in \mathbb{Z}_q^{n \times m}$ is sampled uniformly at random from $\mathbb{Z}_q$ according to the probability distribution χ.

Let $B < N - 1$ and $\bar{B} = N - B$. We define the *rounding* function as

$$\lfloor \cdot \rceil_{2^B} : v \mapsto \lfloor 2^{-\bar{B}} v \rceil \pmod 2^B,$$

where $v \in \mathbb{Z}_q$ is represented as an integer in $[0, q)$. The rounding function outputs the B most significant bits of $v + 2^{\bar{B}-1}$, essentially partioning $\mathbb{Z}_q$ into 2^B intervals.

We will define the *cross-rounding function* from $\mathbb{Z}_q \to \mathbb{Z}_2$ as follows:

$$\langle \cdot \rangle_{2^B} : v \mapsto \lfloor 2^{-\bar{B}+1} v \rfloor \mod 2$$

where $\bar{B} = (\log_2 q) - B$. The cross rounding function partitions $\mathbb{Z}_q$ according to the $(B+1)$th most significant bit.

[1] This claim needs proof. For example, you could use the Leftover Hash Lemma.

- Alice generates $A \in \mathbb{Z}_q^{n \times n}$ uniformly at random. She samples $S, E \to \chi^{n \times \bar{n}}$ and sends $(A, AS + E) \in \mathbb{Z}_q^{n \times n} \times \mathbb{Z}_q^{n \times \bar{n}}$.
- Bob samples $S', E' \to \chi^{\bar{n} \times n}$ and $E'' \to \chi^{\bar{n} \times \bar{n}}$. He calculates $B' = S'A + E'$ and $V = S'B + E''$.

  Bob sends $(B', \langle V \rangle_{2^B} \in \mathbb{Z}_q^{\bar{n} \times n}) \times \mathbb{Z}_2^{\bar{n} \times \bar{n}}$.
- Alice calculates $B'S$; for each element x_{ij} in this matrix, she outputs $\lfloor v \rceil_{2^B}$, where v is the closest element to x_{ij} so that $\langle v \rangle_{2^B} = C_{ij}$. She calls this output matrix K_A.
- Bob calculates $\lfloor V \rceil_{2^B}$, calling this matrix K_B.

Correctness With high probability, $K_A = K_B$. Hence Alice and Bob agree on a secret. This of course depends on the choice of the error distribution. The error distribution is an approximation to the rounded Gaussian.