**CSE 206A: Lattice Algorithms and Applications** Spring 2007

# Lecture 1: Introduction

*Lecturer: Daniele Micciancio* *Scribe: Daniele Micciancio*

## 1 Introduction

Lattices are geometric objects that have been used to solve many problems in mathematics and computer science. In this course we will study the theory of lattices from an algorithmic point of view, motivated by their applications in cryptology as well as other areas of computer science.

A cautionary note about the word *lattice*: There are two quite different mathematical structures that are usually called lattices. One has to do with partially ordered sets and it is *not* the one we will be talking about in this class. The other one, which is the main subject of this course, can be pictorially described as the set of intersection points of a regular (but not necessarily orthogonal) n-dimensional grid (see figure 1).[1]

Three dimensional lattices arise naturally in crystallography and sphere packing problems (e.g., stacking oranges). Historically, lattices in $n$-dimensional space were studied (in the equivalent language of quadratic forms) since the 18th century by mathematicians such as Lagrange, Gauss, and later Minkowski, as a bridge between geometry and number theory. (We will illustrate some of the most basic applications of lattices in number theory later in this course.) Still, no satisfactory algorithm was known to solve many lattice problems till the early 80', when the *lattice reduction* algorithm of Lenstra, Lenstra and Lovasz (most commonly known as the LLL algorithm) was discovered [LLL82]. Since the discovery of the LLL algorithm, lattices have found numerous applications in computer science, ranging from computer algebra, to coding theory, and most notably cryptology. Quite peculiarly, in cryptology, lattices have found application both to cryptanalysis (using lattices to break cryptosystems) and cryptography (using hard lattice problems to design secure cryptosystems).

In this course you will learn about lattices, algorithms to solve lattice problems, and their many applications in computer science.

**Background** I will assume familiarity with basic material from various area of mathematics, like linear algebra, basic number theory, and probability, and occasionally some calculus, as typically covered in most computer science undergraduate curricula.

[1] This second kind of objects is sometimes called *point lattice* (or *integer lattice* when the coordinates of the intersection points are all integers) in order to distinguish it from the first kind. However, they are more commonly called just *lattices*. The two notions are so different that they seldom occur in the same context (paper, book, etc.), so the ambiguity caused by the same name is usually not much of a problem. All the material handed out in this class will be about point lattices. However, if you look for other material (in the library, on the web, etc.) be prepared to find many references on lattice theory that have nothing to do with the subject of this class.

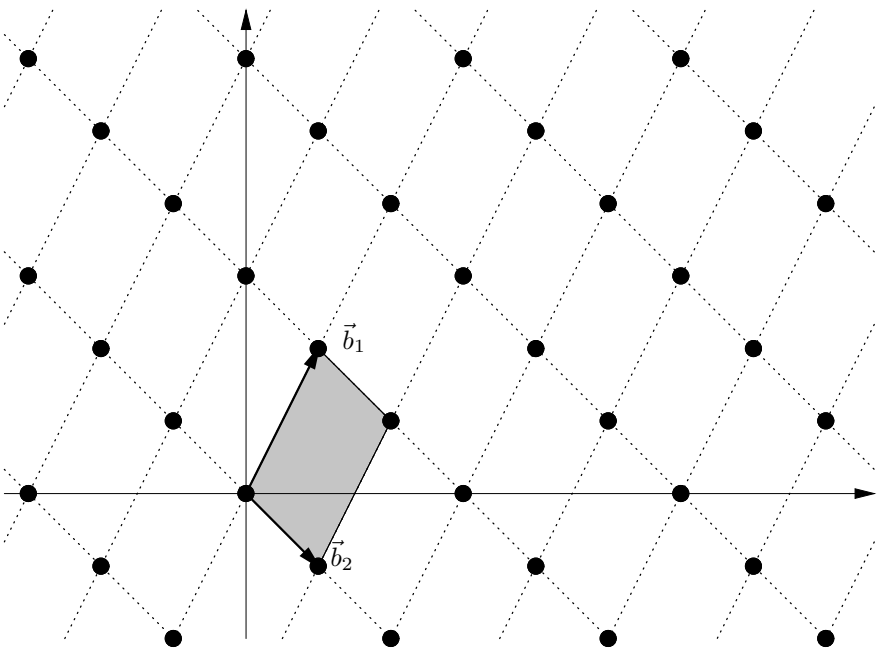


Figure 1: A two dimensional lattice

# 2 Some Problems in Cryptanalysis and Coding

In order to motivate the development of lattice theory, we will first describe some non-trivial problems in cryptology and coding theory that will be solved later in the course with the help of lattices. Within cryptology, we will first focus on cryptanalysis, both for historical reasons (cryptanalytic applications were the first to be discovered) and because they use lattices in a more direct way than the construction of secure cryptographic functions.

## 2.1 Chinese Remainder Codes

Recall the Chinese remainder theorem:

**Theorem 1** *Let $p_1 < \ldots < p_k$ be $k$ relatively prime positive integers, and let $P = \prod p_i$ their product. Then, the ring $\mathbb{Z}_P$ of integers modulo $P$ is isomorphic to the product ring $\prod_i \mathbb{Z}_{p_i}$, with isomorphism $\phi\colon \mathbb{Z}_P \to \prod_i \mathbb{Z}_{p_i}$ given by*

$$\phi(x) = (x \bmod p_1, \ldots, x \bmod p_k).$$

*Moreover, the isomorphism $\phi$ is efficiently computable and invertible. In particular, for any $a_1, \ldots, a_k$, there exists a unique $a \in \mathbb{Z}_P$ such that $a_i = a \bmod p_i$ for all $i$, and such $a$ can be efficiently computed from $a_1, \ldots, a_k$.*

The Chinese remainder theorem gives a way to encode an integer $a$ in the range $\{1, \ldots, P\}$ as a tuple of smaller integers $a_1, \ldots, a_k$ in such a way that $a$ can be uniquely recovered from $a_1, \ldots, a_k$.

Now assume the smaller integers are transmitted over a noisy channel that may corrupt some of the $a_i$, say to $a'_i$. In general this will result in the wrong number $a$ being recovered. In order to avoid such "decoding errors", we can encode $a$ in a redundant way using a longer sequence of moduli $p_1 < \ldots < p_n$, where $a < \prod_{i=1}^{k} p_i$ and $k < n$. It follows from the Chinese remainder theorem that if less than $d = (n-k)/2$ remainders $a_i$ are corrupted, then $a$ is the unique integer in the range $\{1, \ldots, P\}$ such that $a \bmod p_i \neq a'_i$ for at most $d$ values of $i$. To see this, consider any message $m$ in the range $\{1, \ldots, P\}$ such that $m \neq a'_i \pmod{p_i}$ for at most $d$ values of $i$. Let $S$ be the set of indexes such that $a = a'_i \pmod{p_i}$ and $m = a'_i \pmod{p_i}$. By assumption, the set $S$ has size at least $n - 2d = k$. Since $a = m \pmod{p_i}$ for all $i \in S$, it follows by the Chinese Remainder Theorem that $a$ and $m$ are congruent modulo $\prod S \geq \prod_i^k p_i = P$. This proves that $a = m$ because $a$ and $m$ are both in the range $\{1, \ldots, P\}$.

So, if the number of errors is bounded by $d = (n-k)/2$, then $a$ can still be uniquely recovered, at least in principle. The question is: how can we efficiently find $a$ without knowning which positions are corrupted?

## 2.2 Low-exponent RSA

The first kind of cryptanalysis problems we will consider is attacking the so-called low-exponent RSA function. RSA is a family of cryptographic functions parametrized by a modulus $N = pq$ (where $p$ and $q$ are prime numbers) and an encryption exponent $e$ relatively prime to $\varphi(N)$. In practice, for efficiency reasons, the exponent is often fixed to some small number, say $e = 3$, giving the low-exponent RSA function

$$\text{loRSA}_N(m) = m^3 \bmod N.$$

The loRSA function can be easily computed (given $N$), but no efficient algorithm is known to invert it ... unless the factorization of $N$ is known.[2] So, loRSA can be used as a public key encryption scheme as follows:

- Each party (say, Bob, Charlie and Dave) picks a pair of large random prime numbers $(p_X, q_X)$ (for $X = B, C, D$), computes their product $N_X = p_X q_X$, and publishes it in a directory (e.g., telephone book)
- When Alice wants to send a message to Bob, she looks up his *public key* $N_B$ in the directory, and sends the encrypted message $\text{loRSA}_{N_B}(m)$ to Bob
- Bob recovers the original message $m$ from the ciphertext $\text{loRSA}_{N_B}(m)$ using his knowledge of $p_B$ and $q_B$.
- Since Bob is the only one to know the factorization of $N_B$, no adversary intercepting the ciphertext can recover $m$

[2] Given the factorization of $N$, one can compute Euler's function $\varphi(N) = (p-1)(q-1)$, and the inverse $d$ of 3 modulo $\varphi(N)$. We recall that $\varphi(N)$ is the order of the multiplicative group $\mathbb{Z}_N^*$. So, for any $a \in \mathbb{Z}_N^*$, we have $a^{\varphi(N)} = 1 \pmod{N}$. Using $d$, $m$ can be easily recovered from $\text{loRSA}(m) = m^3 \pmod{N}$ raising $m^3$ to the power $d$ to yield $m^{3d} = m^{1+k\varphi(N)} = m \cdot (m^k)^{\varphi}(N) = m \pmod{N}$.

Now consider a broadcast application in which Alice sends the same message to Bob, Charlie and Dave encrypted with their respective loRSA public keys $N_B, N_C, N_D$. The ciphertexts are $c_B = m^3 \bmod N_B$, $c_C = m^3 \bmod N_C$ and $c_D = m^3 \bmod N_D$. Well, an adversary that sees all the ciphertexts can recover the entire message as follows: use the Chinese Remainder Theorem to compute $c = m^3 \bmod N_B N_C N_D$. Then notice that since $m < N_B, N_C, N_D$, then $m^3 < N_B N_C N_D$ and $c = m^3$ (no modular reduction performed!). The message $m$ can be easily recovered computing the cubic root of $c$ over the integers.

It is clear that loRSA has some serious weaknesses whenever the same message is encrypted more than once (even with different public keys). So, let's fix the weaknesses by forcing the messages to be different using various tricks. Consider the following scenarios in which loRSA is modified in order to avoid the previously described attacks:

- In the broadcast example, append the name of the recipient to each message. Alice send message "m;B" to Bob, "m;C" to Charlie and "m;D" to Dave. The CRT attack no longer works, so this seems to solve the problem.
- If Alice is sending several (possibly identical) messages to Bob, append a timestamp to the messages. I.e., send the encryption of "m;t" where $t$ is the current time, or a counter that is incremented after each transmission. Is this secure?
- Maybe loRSA can be used in very simple applications where it is used only once. Consider for example Alice sending to Bob a single message $m$=(Let's use RSA only once. From now on let's use triple DES with key "secretword"). Once the secret key is established Alice and Bob can keep talking to each other using a symmetric cipher (e.g., triple DES) assumed to be secure.
- Transform loRSA into a probabilistic function using a random pad. Each time Alice wants to send a message $m$, she generates a radom string $r$ and encrypts $m;r$. The security of this scheme obviously depends on the length of the random pad (e.g., if $r$ is 10 bits, then one can easily try all possibilities). So, let's consider some specific examples:
  - $N = 1000$ bits and $r = 100$ bits. These are typical parameters, as 1000 bits moduli are currently considered infeasible to factor, and trying $2^{100}$ possible values for $r$ would take an infeasible amount of time with current technology.
  - Let's forget efficiency for a moment, and take $N = 10000$ bits and $r = 1000$. Is the encryption function secure now?

In this course we will develop general techniques that can be used to attack all of these problems.

## 2.3 Linear Congruential Generators

A second class of attacks that we will consider are attacks to a popular kind of pseudo-random generator: the linear congruential generator (LCG). A LCG is defined by three integer parameters $M, a, b$. Given a seed $x_0$ the generator outputs a sequence $G(x_0) = x_1 x_2 x_3 \ldots$ where the $x_i$'s are defined by the recurrence $x_{i+1} = a \cdot x_i + b \bmod M$.

LCG are very popular because they are fast and enjoy good statistical properties (if the parameters are accurately chosen). However they are not cryptographocally secure. If the parameters are known (but the seed $x_0$ is kept secret) the generator is *predictable*: as soon as the adversary sees $x_1$, she can start predicting the rest of the sequence applying the recurrence relation to $x_1$.

We consider several different ways to "fix" the generator.

- Keep the parameters $M, a, b$ secret. It can be shown that even in this case the generator is predictable. (You don't need lattices for this attack.)
- Hide part of the $x_i$'s, e.g., output only the first half of the bits of each $x_i$. This kind of generator is called *truncated LCG*. The simple prediction algorithm cannot be applied any more (even if the parameters are known) because you do not get to see any $x_i$ entirely.
- Apply a one-way function (i.e., a function that is computationally hard to invert) to $x_i$ before it is output. This might already be done by some cryptographic applications. For example, the DSS digital signature algorithm takes as input a message $m$ to be signed and some randomness $r$, but does not output $r$ as part of the signature. Rather it outputs the value $g^r \pmod p$. Again, the simple prediction algorithm cannot be applied because it is not known how to recover $r$ from $g^r \pmod p$.

Next time we will start studying lattices and lattice approximation algorithms, and in a few lectures you will find out how all these cryptographic applications can be broken using lattices.

## References

[LLL82] Arjen K. Lenstra, Hendrik W. Lenstra, Jr., and László Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261:513–534, 1982.