CSE 206A: Lattices Algorithms and Applications (Fall 2017)

Problem Set 3

Lecturer: Daniele Micciancio **Due:** *Thu November 2, 2017*

Cryptanalysis

You have intercepted a secret message. (You will receive the message over email soon!) You know the message as been encrypted using a truncated linear congruential generator with parameters

- $p = 258535798238006737310015446227842039611$
- $a = 558997263476390417180901441412060571 79$
- $b = 206062773649155181136659667127305895548$

(For your convenience, the parameters have been included in the email.) As a reminder, the linear congruential generator starts from a random secret seed $x_0 \in \mathbb{Z}_p$ (unknown to you), computes the sequence $x_{i+1} = ax_i + b \pmod p \in \mathbb{Z}_p$, and outputs the 8 least significant bits $y_i = x_i \pmod{256}$ from each value in the sequence.

You know these values $(y_0, y_1, \ldots)$ have been used as a one-time pad to encrypt the message by breaking the message into a sequence of characters $(m_0, m_1, \ldots)$, interpreting each character as an 8-bit number using the standard ASCII encoding, and computing the ciphertext as $c_i = m_i + k_i \pmod{256}$. You also know that the message begins with the string "CSE206A Cryptanalysis Challenge:". (Pay attention to upper and lower case.)

Decrypt the message, and include it in your answer together with a brief description of how you solved the problem.

As part of this problem, you will need to run a combination of LLL lattice basis reduction and the nearest plane algorithm to solve an instance of the bounded distance decoding problem. You are neither required nor expected to implement your own lattice algorithms as part of this problem. Instead, use one of the many available implementations linked on the course webpage. The easiest way is probably to use "fplll", which provides both a library and a command line interface. The command line interface should be enough to solve this problem. You can install fplll from github following the link on the course webpage. (It may also be available through your operating system package manager.) Then, if you want to solve a BDD or CVP instace, write the basis matrix and vector in a file and then run "fplll -a cvp < inputfile". For example, if the input file contains "[[3 0] [0 1]] [5 6]", the commad should output the vector "[6 6]". Here fplll first runs the LLL basis reduction algorithm (you don't

need to reduce the basis before calling fplll) and then finds the lattice vector closest to the target [5 6] using a variant of the nearest plane algorithm. (Specifically, it first runs the nearest plane algorithm, and it a suitable solution is not found, it performs an exhaustive search for the closest lattice vector.) In general, this may take a *lot of time* to produce a solution, because the closest vector problem is NP-complete, and the algorithm implemented by fplll is exponential time in the worst case. But for the BDD instances involved in this problem, the running time will be polynomial, and, in fact, pretty small, say just a fraction of a second (including the time to perform basis reduction.) If *fplll* is taking more than a few seconds to solve your problem, it probably means you got your basis or target wrong, and you should abort the computation and fix the input, rather than keep waiting for an answer (which, most likely, is incorrect anyway.)

If you want to see what an LLL reduced basis looks like, just run "fplll -a lll" on the lattice basis.