Lisa Maszkiewicz

May 7th, 2020

Cryptology

Professor D. Zaret

Applying the LLL Lattice Reduction Attack to the Merkle-Hellman Knapsack Encryption Scheme

~ Merkle-Hellman Knapsack Cryptosystem ~

This paper documents my journey learning about lattice reduction attacks and how to break knapsack cryptosystems with them. I will first explain the sort of math knowledge I had to build up before digging into the actual implementation. Next I will go into details of how and why this cryptosystem works. After that, I will describe the libraries I used to demonstrate how the attack is executed. Along the way, there were many interesting detours, which I have documented at the end of my paper. Enjoy!

Before reading about lattices and LLL, I wanted to first remind myself of the Merkle-Hellman knapsack cryptosystem. I remember doing the Merkle-Hellman problem on problem set #3. It is a typical knapsack problem, specifically apparently the subset sum problem, which I hadn't really heard of until now. It is NP-complete, and reduces to the Traveling Salesman Problem, which is not a necessary fact to know, just refreshing my brain on the problem. As we discussed in class, the general idea of a Merkle-Hellman encryption/decryption scheme is to introduce a seemingly difficult/unfeasible subset sum problem into an easy subset problem, given a secret key. Basically, you want to transform an easy subset sum problem A = (a1, a2, a3 …) and M, into a much more difficult one. Then transmit the public key B = (b1, b2, b3 …) along with the private keys: M, W, which can be decrypted. Here is a reminder of the encryption/decryption scheme:

Encryption of (x1, x2, x2 …)

$$T = \sum_{i=1}^{n} b_i \, x_i$$

Decryption

1. Compute:

$$W^{-1}T \ (mod\ M) = \ W^{-1} \sum b_i x_i (mod\ M) = W^{-1} \sum W a_i x_i (mod\ M)$$
$$= \sum a_i x_i (mod\ M) = \ \sum a_i x_i$$

2. Solve easy subset sum problem.

In general, as I mentioned earlier, you disguise an easy subset sum problem, as a really difficult one, and hide the keys (W^{-1}, M) that make the problem simple. Very straightforward and elegant, cool, got it. All of this should be familiar, and a restatement of what we learned in class. Next, to learn about the lattices and LLL algorithm, which somehow avoids solving this NP-complete problem, but does break the encryption/decryption scheme.

Before learning about LLL I need to learn about lattices. What is a lattice? I needed to refresh some of my linear algebra knowledge, but happily, most of the terminology was still familiar to me. I never learned what a lattice is, but I know now they are the set of all linear combinations of a set of column vectors with integer coefficients. One example I find easy to visualize is that a lattice is a collection of distinct points in a plane that can be obtained by adding any linear combination of two linearly independent vectors (say c_0 and c_1). i.e. $\propto_0 * c_0 + \propto_1 * c_1$, where $\propto_0$ and $\propto_1$ are integers.

In this section, most of the following properties I'll describe and the algorithm outline I got from [1], which I found a really helpful text for this project. Just to be clear, a lot of the following is a re-phrasing (and at times barely that) of what is written in the text. This is crucial information for the algorithm.

The useful part of lattices is being able to solve for a matrix M^{-1} from M, in a reasonable amount of time. We already discussed what a lattice is, more formally written, it is:

$$L = \{a_1 v_1 + a_2 v_2 + \cdots + a_n v_n : \ a_1, a_2, \ldots, a_n \ \in Z\}$$

Any set of independent vectors that generate L is a basis. Say that that set of linearly independent vectors are a basis $v_1, v_2, \ldots, v_n$ for L. Now say there's another set $w_1, w_2, \ldots, w_n \ \in L$. Because we know that set of v vectors is a basis, there must be some way to write the following:

$$w_1 = a_{11} v_1 + a_{12} v_2 + \cdots + a_{1n} v_n,$$

$$w_2 = a_{21} v_1 + a_{22} v_2 + \cdots + a_{2n} v_n,$$

$$\ldots$$

$$w_n = a_{n1} v_1 + a_{n2} v_2 + \cdots + a_{nn} v_n,$$

And because we've been dealing with lattices, we know those a coefficients are integers. So now, to express the v's in terms of w, it involves inverting the matrix:

$$A = \begin{bmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \ldots & \ldots & & \ldots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}$$

Another important note, I remember in linear algebra lots of discussions on determinants, and here the case is no different. We need A^{-1} to have integer entries, and since we know that

A has integer values, the determinant of $A * A^{-1}$ needs to equal one. Therefore, det(A) = ± 1. Now we have this useful result!

Proposition 6.14 [1]: "Any two bases for a lattice L are related by a matrix having integer coefficients and determinant equal to ± 1."

This is a very important result. The integer property is key, hopefully you can see how it could lend itself to solving our problem of trying to find the integer coefficients for our particular equation. We start with vectors w, expressed in terms of v. Write that as a matrix, then find the inverse of that matrix. The coefficients in that matrix will be how to express vectors v, in terms of w.

The next important concept for Merkle-Hellman is that of a "short vector". A short vector is defined as the shortest nonzero vector in the lattice. An efficient algorithm for finding this vector is what made lattices such a useful. There are also close vectors, which we don't need, but those are defined as the vector in a lattice that is closes to a given non-lattice vector. Here's a formal definition from the text of the shortest vector fundamental lattice problem:

"The Shortest Vector Problem (SVP): Find a shortest nonzero vector in a lattice L, i.e., find a nonzero vector $v \in L$ that minimizes the Euclidean norm $||\boldsymbol{w} - \boldsymbol{v}||$."

There's also a special case of SVP called approximate SVP (apprSVP). I will discuss that briefly, I think that the LLL actually only breaks apprSVP, and only breaks SVP in small dimension cases.

Next, what is a lattice reduction? One article I read states: "This transformation of a bad basis into a better basis is known as lattice reduction," [4] That sums up the general idea. I will attempt to walk through the process.

Lattice reductions apparently started with Gauss, in only two dimensions. As stated in [1], "The underlying idea is to alternately subtract multiple of one basis vector from the other until further improvement is not possible. Suppose you have two basis vectors of L, v_1 and v_2. Say $||v_1|| < ||v_2||$, we can try to make v_2 smaller by subtracting an integer multiple of v_1 (an integer multiple will guarantee it's still in L). Repeat if $||v_1|| < ||v_2||$, or, if v_2 is now shorter than v_1, swap the two and repeat. This eventually terminates when both of the vectors can't get any smaller, and you can get a fairly short basis from this process. Here's a pseudocode of this short algorithm from Springer:

Loop
  If $\|\mathbf{v}_2\| < \|\mathbf{v}_1\|$, swap $\mathbf{v}_1$ and $\mathbf{v}_2$.
  Compute $m = \lfloor \mathbf{v}_1 \cdot \mathbf{v}_2 / \|\mathbf{v}_1\|^2 \rceil$.
  If $m = 0$, return the basis vectors $\mathbf{v}_1$ and $\mathbf{v}_2$.
  Replace $\mathbf{v}_2$ with $\mathbf{v}_2 - m\mathbf{v}_1$.
Continue Loop

Next I wanted to learn about the Lenstra-Lenstra-Lovasz (LLL) algorithm itself. Something I learned very quickly is that a lot of the papers assume you are a mathematician, so I struggled to find good papers to read that were accessible to me. I did like the text in [1], but I was hoping to also find something that didn't use any mathematical symbols or equations to describe it. To me, the general idea of LLL is to reduce a set of vectors until they are as short a length as possible.

We want to get the above inequality as close to an equality as possible. If we need to construct an improved basis, we need to create a basis. We can use Gram-Schmidt to do so! Wow I remember doing Gram-Schmidt in linear algebra about a billion years ago. It was long and tedious work, but I remember it being satisfying that you can construct these orthogonal vectors from an algorithm. Alright, so say we have a collection of vectors B* = $\{v^*_1, v^*_2, \dots, v^*_n\}$ that is an orthogonal basis for B = $\{v_1, v_2, \dots, v_n\}$. There's a definition in the text ([1] p. 408):

"Let B = $\{v_1, v_2, \dots, v_n\}$ be basis for a lattice L and let B* = $\{v^*_1, v^*_2, \dots, v^*_n\}$ be the associated Gram-Schmidt orthogonal basis as described in Theorem 6.13. The basis B is said to be *LLL reduced* if it satisfies the following two conditions:

$$\textbf{(Size Condition)} \quad |\mu_{i,j}| = \frac{|\mathbf{v}_i \cdot \mathbf{v}_j^*|}{\|\mathbf{v}_j^*\|^2} \le \frac{1}{2} \qquad \text{for all } 1 \le j < i \le n.$$

$$\textbf{(Lovász Condition)} \quad \|\mathbf{v}_i^*\|^2 \ge \left(\frac{3}{4} - \mu_{i,i-1}^2\right) \|\mathbf{v}_{i-1}^*\|^2 \quad \text{for all } 1 < i \le n.$$

It also states: "The fundamental result of Lenstra, Lenstra, and Lovasz says that an LLL reduced basis is a good basis and that it is possible to compute an LLL reduced basis in polynomial time." Hopefully the size condition makes sense from our previous conversation. For me, the Lovasz condition was a lot more confusing, I like this equivalent statement a little better:

$$\|\text{Projection of } \mathbf{v}_i \text{ onto } \text{Span}(\mathbf{v}_1, \dots, \mathbf{v}_{i-2})^\perp\| \ge \frac{3}{4}\|\text{Projection of } \mathbf{v}_{i-1} \text{ onto } \text{Span}(\mathbf{v}_1, \dots, \mathbf{v}_{i-2})^\perp\|.$$

To me, this is more readable than the shorter Lovasz condition, but I'm still not exactly sure where they got this value from, the math was very complex at this point. After reading a little more, it seems the idea is that swapping the order of the vectors can change how small they can get. And apparently, the Lovasz condition is close to the minimum size that the vectors can get. So, once the vectors are reduced a certain amount, see if they're as short as they can get, if not, swap the order a little and reduce some more, until the Lovasz condition is satisfied.

Here is the algorithm from the text:

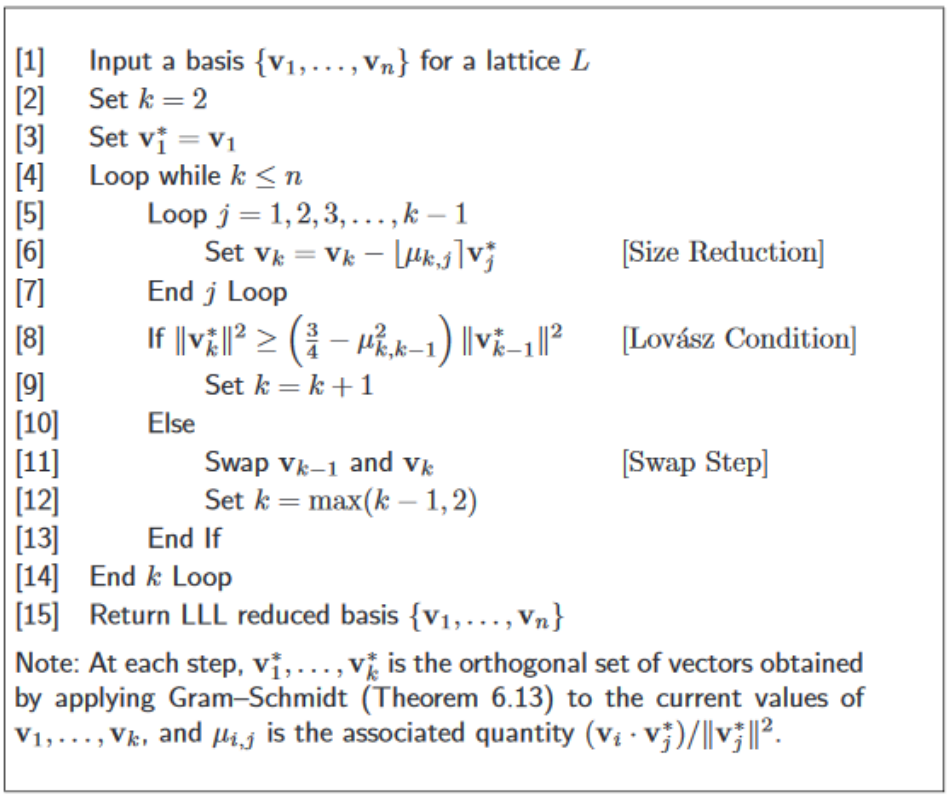

Figure 6.7: The LLL lattice reduction algorithm

You can see the steps we've talked about fairly clearly. You see that we're iterating through the vectors, reducing as much as possible. Then, check to see if we've got them as small as we can, if so, move on to the next vectors in the set. If not, swap the order of the two vectors in focus and reduce again. All of this is just to be able to solve that matrix problem we talked about way at the beginning of our lattices discussion. As said in [1]: "The goal of LLL is to produce a list of short vectors in increasing order of length." That should sound very familiar!!! But I'll get back to that a little later.

I wanted to discuss one more piece of the in-depth math before I go back up to a higher level. They did discuss for a bit how the algorithm is polynomial time. In the main chunk from lines [4]-[14], you can see there are additions to k and subtractions to k. There is a proof in the text that describes how, ***if*** the algorithm terminates, it will be in polynomial time. It is fairly complicated and relies on other theorems not discussed in this paper thus far, therefore, I will leave that proof as an exercise for the reader… ☺ The algorithm turns out to run in $O(n^2)$ time, if it terminates. This distinction is also where it fits in that LLL always solves apprSVP problems, and SVP problems in small dimensions. But the larger the dimension, the larger the intermediate values are, and it becomes very intractable to compute the solution. Any type of rounding can cause errors later on. There were some variations of LLL that tried to describe ways to reduce this problem. It doesn't seem like anything groundbreaking, but more so

chipping away at a large block to make some improvements in "output". I'm not exactly sure what that means without reading about these variations, but I'm assuming it means it's allowing you to solve larger problems, even if the run time is slower, it's possible.

Now to tie everything together, we have all the tools we need, we just have to reformat our knapsack problem as a lattice problem. Say you have values M = (89, 243, 212, 150, 245) and S=546 (example given in textbook). That can also be viewed as this matrix (sorry the formatting is not great):

$$A_{M,S} =$$

$$\begin{vmatrix} 2 & 0 & 0 & 0 & 0 & 89 \\ 0 & 2 & 0 & 0 & 0 & 243 \\ 0 & 0 & 2 & 0 & 0 & 212 \end{vmatrix}$$

$$\begin{vmatrix} 0 & 0 & 0 & 2 & 0 & 150 \\ 0 & 0 & 0 & 0 & 2 & 245 \\ 1 & 1 & 1 & 1 & 1 & 546 \end{vmatrix}$$

Then if you reduce this matrix using LLL, you get the following matrix:

$$\begin{vmatrix} -1 & 1 & -1 & 1 & -1 & 0 \\ 1 & -1 & -1 & 1 & -1 & -1 \\ -1 & -1 & -1 & 1 & 1 & 2 \end{vmatrix}$$

$$\begin{vmatrix} 1 & -1 & -1 & -1 & -1 & 2 \\ -2 & -2 & 4 & 0 & -2 & 0 \\ -6 & -4 & -6 & -6 & 0 & -3 \end{vmatrix}$$

Then, one last trick, take the top vector (-1, 1, -1, 1, -1,0) and write it as a linear combination of the original basis vectors given by the rows of $A_{M,S}$.

$$\text{(-1, 1, -1, 1, -1,0)} = \text{(-1, 0, -1, 0, -1, 1)}\, A_{M,S}$$

(-1, 0, -1, 0, -1, 1) is the solution to the knapsack problem. -1(89) + 0(243) + -1(212) + 0(150)+ -1(245) + 1(526) = 0 => 89 + 212 + 245 = 526 !

This lattice reduction seems as though it is a very powerful tool for solving problems quicker. [2] is a paper from 2008, that discusses lattice-based cryptography. It suggests that lattices are promising for post-quantum cryptography (wow, people were already worried about this in 2008???) because "they enjoy very strong security proofs based on worst-case hardness, relatively efficient implementations, as well as great simplicity." I see now how the portion on worst-case hardness is really useful. I don't know if I agree about the 'great simplicity' part. I found the linear algebra concepts pretty tricky to grasp, and I wouldn't really say any part of this is straightforward. The general idea is, maybe that's what this person was referring to. I am a little disappointed that I didn't get to know more about why lattice-based cryptography is secure against quantum computers, but maybe once I've recovered from the trauma of this math and let the information soak in a little more, I'll be open to reading up

more on that particular topic. I kind of wish I had made my project topic the one that more generally learned about lattices and how they affect a post-quantum world. I think I was intimidated because with this project, at least I understand Merkle-Hellman. If I had picked one of those, I'd have to understand the quantum-resistant portion, and the lattices portion, which was too intimidating for me to figure that out on my own. This sentence from [2] alone makes me nervous: "lattice problems seem like a natural candidate to attempt to solve using quantum algorithms: because they are believe not to be NP-hard for typical approximation factors, because of their periodic structure, and because the Fourier transform, which is used so successfully in quantum algorithms, is tightly related to the notion of lattice duality". There are quite a few intimidating words in that sentence, including 'periodic structure', 'Fourier transform', and 'lattice duality'. But who knows, maybe it wouldn't be so bad.

~ Implementation ~

Now to actually implement the thing.

- I tried to use this Python implementation of LLL reduction: https://github.com/orisano/olll
- Then I tried to use this Python implementation of LLL: https://github.com/fplll/fpylll, which might have been the most disappointing failure, because this is their mission statement: "Make implementing lattice-reduction strategies so easy that we can demand that people publish their code." I found this extremely frustrating because after following their installation steps to the letter, I could not get the library to cooperate.
- I briefly tried to implement LLL with a C++ library having not found any luck in Python thus far: https://github.com/Nimsay/LLL--Attack which did not go well because I have not used C++ in a long time
- Here's another C++ implementation I thought about trying: https://github.com/rnavares/BREAKING-Merkle-Hellman-Crytosystem-BAD-Ass-i
- But, due to the aforementioned lack of talent with C++, I first tried one last ditch attempt with this Python 2.7 library: https://github.com/rrkane/LLL

And gosh darn it I was not able to get it working!!! Overall, the implementation portion was not that exciting to me anyways. I found it incredibly frustrating trying to work with these libraries. So many people talk about how the world is moving towards open-source projects, and after an experience like this I am more skeptical than before. The math and approach were the interesting bits, especially learning how these lattice-based approaches will be future cryptosystem foundations. This surprised me, usually I really enjoy when everything comes together via code, but in this case, all it would really do is reduce my problem to a matrix, and outputted the inverse matrix with a solution vector as one of the columns.

I was very happy initially when I found the rrrkane implementation, because I found it easy to follow, and the steps were clearly laid out, just like in the paper. I was able to clearly identify the different steps, the man wrote comments and even has documentation strings! There is

even an option to have the steps printed out as you go. Hallelujah! But, I could not get the dang thing to run. I tried to debug a little, but I was really frustrated, especially when I went to the trouble of downloading Python 27! I had a casting error with the Numpy array that was used, I'm guessing this person used a specific version of Numpy, and it has changed since then. However, I'm hoping I have gone into depth enough in the math section above to convince you that I know how the attack works. If you take a peek at the code, you can see it follows a very similar format to the pseudocode I showed previously. There's the size reduction step, then Lovasz check/swap step. Helper methods include Lovasz, reduction, gram_schmidt, projection, and scaled projection. I also included my work for the two of the libraries that I attempted to get working.

~ Beginning of interesting notes section ~

There are several things I found interesting when writing this paper that I thought I would include, listed below in decreasing order of interest.

I was pleased to see that lattice-based cryptography rests on the assumption that well-studied lattice problems are NP hard. However, it does make me nervous. I know that most people believe NP hard problems will eventually be proven to not be solvable in polynomial time, but it's not a guarantee. And, I don't have extensive knowledge on the other types of cryptographic schemes for a post-quantum world. But, it seems risky to put all the marbles in this bin, when this bin seems to have such an uncertain future. I do think the subset sum problem will turn out to not be solvable in polynomial time, I'm just hesitant because it would be really catastrophic to the world if we build our future of cryptosystems based on the fact that P != NP, and then someone figures out that's not true. And, people have been trying to figure out the answer for P vs. NP for almost 70 years now and it's hard to imagine we're on the brink of solving it. It could happen all of a sudden yes, that would be very exciting, no matter what the result is, I hope it happens in my lifetime. Anyways, I'm rambling on, I just find it a very interesting idea that people are willing to base cryptographic techniques on the assumption that P != NP.

Something that really bothered me is how many different sources of information that had conflicting techniques. The LLL algorithm was fairly standard, but getting the Merkle-Hellman knapsack problem to fit into the matrix solution there were many different ways people suggested to do it. One site suggested the values should be on the bottom, one on the right vertical. One site suggested it should be a identity matrix * 2, while a lot of them simply suggested they should be an identity matrix. Some said to transpose the matrix before and after LLL. It was very confusing to me, I decided to stick with [1] as my holy grail during this project, but the conflicting sources did not help in my understanding of the content. It's nice to have one accurate source (thanks for providing it), but it didn't always have useful explanations, and it was hard to determine if different textbooks were actually saying the same thing.

It interests me that before lattice-based cryptography was seen as a solution, it was first used as a way to break these cryptographic schemes. It kind of makes sense, but it's almost like, people kept using lattices and making better techniques to try and attack different cryptosystems, and then they realized it was pretty hard. So they were like, huh, maybe we should try and make defensive techniques out of this since it's so strong and has a good worst-case.

Something fun I noticed when I was looking at lattice problems is that a paper that kind of started this work by Miklos Atjai, "Generating hard instances of lattice problems", was accepted at the Eighth Annual ACM Symposium on Theory of Computing. I almost went to this conference this year (I'm glad this is not the one I chose because I would not have been able to go due to COVID-19), I instead went to a Data Science Salon. However, I hope to go in the future and I also hope I learn enough from this project to attend talks like this and not be totally lost! That's why I decided in the end not to attend, I didn't want to go and have everything be way over my head. But I'm not getting any smarter so maybe I should go sooner rather than later… ☺

~ End of interesting notes section ~

Overall, I am immensely happy I decided to work on this project. Hopefully you got some understanding of how the LLL lattice reduction operates (if you didn't already understand it). I initially started it as a way of learning about lattice-based cryptography, so that I could add it to my knowledge of why it will work against quantum computers (to add to the knowledge I gained from sitting in on your quantum algorithms course). Although it will never be my field of work, I am very happy to stay informed and watch from a distance as the technology evolves. In addition to that, the project then transitioned to also being about NP hard problems, which I am very interested in. I took an algorithms course from Dr. Mount at UMD, College Park, and it was my absolute favorite course in college. I learned all about graph theory, NP hard problems and how you can reduce them to each other, and found those topics incredibly fascinating. This project helped to add depth in my knowledge in that area as well.

I also feel this is an appropriate time to mention how much I've enjoyed this course, as well as the quantum algorithms course that I've taken with you. Cryptography may not be one of my great passions, but you teach courses in a way that I thoroughly enjoy. You provide useful examples and anecdotes along the way, and bring in an appropriate amount of humor to keep me hanging on when the math gets complicated. Thank you very much!!! Stay great!

Sources

[1] J. Hoffstein, J. Pipher, J. Silverman, *An Introduction to Mathematical Cryptography*, Springer 2008. (https://link.springer.com/content/pdf/10.1007%2F978-0-387-77993-5.pdf)

[2] https://cims.nyu.edu/~regev/papers/pqc.pdf

[3] http://www.cs.sjsu.edu/faculty/stamp/papers/topics/topic16/Knapsack.pdf

[4] https://kel.bz/post/lll/

[5] https://algo.epfl.ch/_media/en/projects/bachelor_semester/rapportetiennehelfer.pdf

[6] https://github.com/rrkane/LLL