Lattice-Based Cryptography & The Shortest Vector Problem

Julian Schennach

August 18, 2024

1 Introduction

Many modern cryptosystems, such as ElGammal or RSA encryption, are based on computationally difficult number-theoretic problems, such as factoring or the discrete logarithm problem. However, with the development of quantum computers, some quantum algorithms have been found that solve these problems far more quickly — factoring, for instance, can be done in polynomial time. Hence, new cryptosystems that are not easily breakable with quantum computers are needed, and fortunately in the past 30 years, researchers have constructed cryptosystems based on *lattices*.

Definition 1. An n-dimensional *lattice* is defined to be $\Lambda = \{a_1\mathbf{b}_1 + a_2\mathbf{b}_2 + \cdots + a_n\mathbf{b}_n : a_i \in \mathbb{Z}\}$, where the $\mathbf{b}_i$'s are linearly independent vectors in $\mathbb{R}^n$. We refer to the set $\{\mathbf{b}_1, \mathbf{b}_2, \cdots, \mathbf{b}_n\}$ as the *basis* of Λ. Each element of the lattice is known as a *lattice point*.

Some of the most common problems used in lattice-based cryptography exploit the difficulty of searching for vectors with their Euclidean norm (i.e. length) minimized. Three such problems are the following:

Problem 1 (Shortest Vector Problem (SVP))**.** Given a basis for a lattice, find the shortest vector from one lattice point to another.

Problem 2 (Shortest Independent Vector Problem (SIVP))**.** Given a basis for a lattice, find another basis such that the length of the longest basis vector is minimized.

Problem 3 (Closest Vector Problem (CVP))**.** Given a basis for a lattice, and a point in $\mathbb{R}^n$, find the closest lattice point to that point.

As of yet, there is no known method for solving these problems in general. Unfortunately, there *is* a fairly efficient algorithm, the LLL algorithm, for finding a basis consisting of fairly short, orthogonal vectors when already given a basis for the lattice, and such "reduced" bases provide opportunities for solving lattice problems. This basis-reduction algorithm is due to Lenstra, Lenstra, and Lovász in 1982, although it was originally applied to factoring rational polynomials, an initially computationally difficult problem. We will be discussing

and providing motivation for the steps of the algorithm, and will also present a brute-force method for solving the SVP with the LLL algorithm.

Interestingly, even with the LLL algorithm being one of the best modern attacks on the SVP, Ajtai was able to show in 1998 that the problem was NP-hard. In this paper we will define "NP-hardness" and explain the importance of this result. Furthermore, an application of the SVP in public key cryptosystems were developed by him and Dwork in 1997, relying on the worst possible cases of the SVP, and we will be explaining how this system works and how messages can be encrypted and decrypted (although the explicit proof that the system is secure is beyond the scope of this paper).

2 Examples of Lattice Problems

The most fundamental definition is the one for *lattices*:

Definition 1. An n-dimensional *lattice* is defined to be $\Lambda = \{a_1\mathbf{b}_1 + a_2\mathbf{b}_2 + \cdots + a_n\mathbf{b}_n : a_i \in \mathbb{Z}\}$, where the $\mathbf{b}_i$'s are linearly independent vectors in $\mathbb{R}^n$. We refer to the set $\{\mathbf{b}_1, \mathbf{b}_2, \cdots, \mathbf{b}_n\}$ as the *basis* of Λ. Each element of the lattice can be known as a *lattice point*.

Alternatively, we can embed a d-dimensional lattice in $\mathbb{R}^n$ by having a basis with only d vectors. Given this definition, there are 3 very common and general lattice problems we can state, the most fundamental one being the SVP.

Problem 1 (Shortest Vector Problem (SVP))**.** Given a basis for a lattice, find the shortest vector from one lattice point to another.

Problem 2 (Shortest Independent Vector Problem (SIVP))**.** Given a basis for a lattice, find n linearly independent vectors such that their maximum length is less than or equal to the length of the longest vector in a basis such that said longest vector is as short as possible.

There also exists a seemingly more difficult generalization of the SVP that focus on finding certain lattice points rather than vectors:

Problem 3 (Closest Vector Problem (CVP))**.** Given a basis for a lattice, and a point $\mathbf{p}$ in $\mathbb{R}^n$, find the closest lattice point to $\mathbf{p}$.

If one can solve the CVP in n dimensions, the SVP in n dimensions can be solved as well, as we can select the origin as the point in $\mathbb{R}^n$. Then, the CVP (if formulated so that it doesn't give the trivial solution of the origin, which is 0 units away from itself) outputs the lattice point that is closest to the origin, which is precisely the query of the SVP.

It turns out that if one can solve the SIVP in n dimensions, one can also solve the CVP in n dimensions, and vice-versa, through a polynomial-time algorithm. If the reader wishes to see a proof of this equivalence, they can consult Micciancio's paper [4]. Hence, if we solve the CVP or SIVP in n dimensions, then we can solve the SVP in n dimensions too. Note that it is an open problem whether the SVP in n dimensions is equivalent to the SIVP (or the CVP) in n dimensions. However, if the SVP in $n+1$ dimensions can be solved, then the CVP and SIVP in n dimensions can be solved too.

Oftentimes, it is also sufficiently difficult to solve approximate versions of these problems. Then, for a certain parameter we want to minimize in a problem, the approximate version asks us to find a solution such that the parameter is less than or equal to $\gamma \geq 1$ times the minimal value of the parameter. This parameter is the length of a vector in the SVP and CVP, and the maximum length of a set of vectors in the SVP. While these versions of the SVP, SIVP, and CVP may of course seem less secure, there is currently no known polynomial-time algorithm for solving any of these exact or approximate lattice problems *in general*. There are polynomial-time algorithms when $n = 2, 3$, though, but for higher dimensions, these lattice problems are very difficult to solve. However, there are some known approaches to solving these problems, including the LLL algorithm.

3 The LLL Algorithm

The Lenstra-Lenstra-Lovász Algorithm, developed by its namesakes Arjen Lenstra, Hendrik Lenstra, and László Lovász in a paper 1982, is one method to approach the SIVP, and can often even solve the approximate version of the SIVP. The algorithm was actually originally applied to factoring polynomials with rational coefficients, as was the intended application in the paper presenting the LLL algorithm, but currently many people are interested in its potential for solving the SVP. (Interestingly, during talks on the LLL algorithm in Caen, a colleague of the authors, Peter van Emde Boas, asked whether people thought that the polynomial factoring or the basis reduction applications were more notable. Those asked all responded basis reduction.)

We now define the LLL algorithm rigorously, following the treatment in the book [5]. Generally, a *reduced basis* is a basis that has rather short and almost orthogonal vectors, and the LLL algorithm defines a particular kind of reduced basis.

Algorithm 1. (The LLL Algorithm)

Given a basis $\{\mathbf{b}_1, \mathbf{b}_2, \cdots, \mathbf{b}_d\}$, calculate the orthogonalized basis $\{\mathbf{b}_1{}^*, \mathbf{b}_2{}^*, \cdots, \mathbf{b}_d{}^*\}$ using the Gram-Schmidt method. Let $\mu_{i,j} = \frac{\mathbf{b}_i \bullet \mathbf{b}_j{}^*}{\mathbf{b}_j{}^* \bullet \mathbf{b}_j{}^*}$ for $1 \leq j < i \leq d$ (where $\mathbf{b}_i{}^* = \mathbf{b}_i - \sum_{j=1}^{j=i-1} \mu_{i,j}\mathbf{b}_j{}^*$ — the $\mu_{i,j}$ are the "Gram-Schmidt coefficients"). Verify whether the original basis is *LLL-reduced*, meaning that the orthogonal basis satisfies the following conditions:

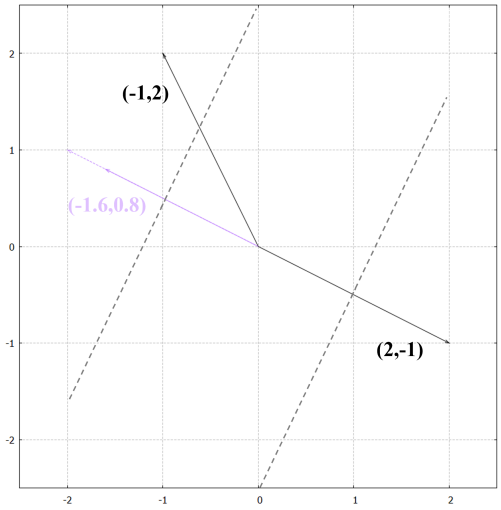
- The basis is *size-reduced*, meaning that $|\mu_{i,j}| \leq \frac{1}{2}$ for each $1 \leq j < i \leq d$. If the orthogonal basis is not size-reduced for a certain $\mu_{i,j}$, then redefine $\mathbf{b}_i$ as $\mathbf{b}_i - [\mu_{i,j}]\mathbf{b}_j$ in the original basis, where $[x]$ is x rounded to the closest integer. Then recompute the orthogonal basis.

- The basis satisfies the *Lovász condition*: For some parameter δ in $[\frac{1}{4}, 1]$, we have $\delta\|\mathbf{b}_i{}^*\|^2 \leq \|\mathbf{b}_{i+1}{}^* + \mu_{i+1,i}\mathbf{b}_i{}^*\|^2 = \|\mathbf{b}_{i+1}{}^*\|^2 + \|\mu_{i+1,i}\mathbf{b}_i{}^*\|^2$ for all $1 \leq i < d$ — these two forms are equivalent, because the $\mathbf{b}_i{}^*$'s are orthogonal. (We can also rewrite the condition as $(\delta - \mu_{i+1,i}^2)\|\mathbf{b}_i{}^*\|^2 \leq \|\mathbf{b}_{i+1}{}^*\|^2$.) If the orthogonal basis does not satisfy the Lovász condition for a certain i, then swap $\mathbf{b}_i$ and $\mathbf{b}_{i+1}$ in the original basis. Again, recompute the orthogonal basis.

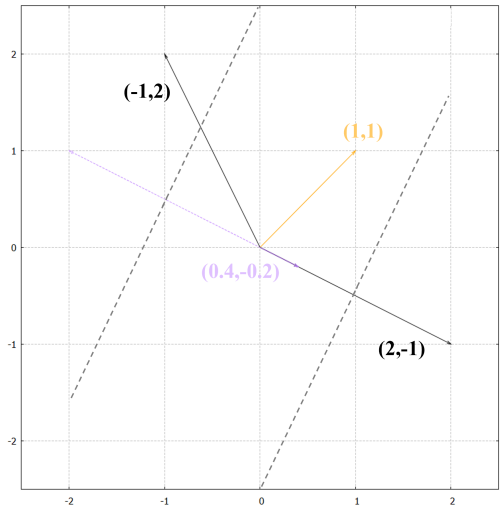
If the recomputed orthogonal basis satisfies all conditions, the modified original basis is LLL-reduced. Otherwise, repeat the modifications noted above.

In practice, one should only recompute the orthogonal basis after checking the conditions for all valid i, j.

The conditions for an LLL-reduced basis might be somewhat unintuitive, so we motivate them using a two dimensional example. First, as the name suggests, the size-reduced condition wishes to minimize the length of Suppose we want to size-reduce the basis $\{\binom{2}{-1}, \binom{-1}{2}\}$. The orthogonal basis is $\{\binom{2}{-1}, \binom{\frac{3}{5}}{\frac{6}{5}}\}$, and $\mu_{2,1} = -\frac{4}{5}$. $\mu_{2,1}$ is not within the interval $[-\frac{1}{2}, \frac{1}{2}]$ — what does this imply? Well, that implies that the projection of $\binom{-1}{2}$ onto $\binom{2}{-1}$ is longer than half the length of $\binom{2}{-1}$, as can be seen in this picture:



The second basis vector $\binom{-1}{2}$ is not as short as it could be — if we remove some integer scalar multiple (for size-reduction, $[-\frac{4}{5}] = -1$) of $\binom{2}{-1}$ until the projection of the resulting vector onto $\binom{2}{-1}$ becomes less than half as long as $\binom{2}{-1}$, then we will obtain a shorter vector (in this case, $\binom{-1}{2} - (-1)\binom{2}{-1} = \binom{1}{1}$) that serves the exact same purpose as our original one:



Notice that the projection onto $\binom{\frac{3}{5}}{\frac{6}{5}}$ (the vector orthogonal to $\binom{2}{-1}$) remains the same, and the projection onto $\binom{2}{-1}$ is shortened, resulting in the full length of our vector being

shortened too. Importantly, size-reduction uses the interval $[-\frac{1}{2}, \frac{1}{2}]$ because we can only remove integer scalar multiples of $\binom{2}{-1}$ from $\binom{-1}{2}$, so the projection onto $\binom{2}{-1}$ can only be within half the length of $\binom{2}{-1}$ from $\mathbf{0}$ (so the "error" from $\mathbf{0}$ is $[-\frac{1}{2}, \frac{1}{2}]$, an interval of length 1 centered at 0).

Next, the Lovász condition aims to make the LLL-reduced basis vectors almost orthogonal — specifically, by modifying the original basis according to that condition (so swapping any necessary pairs of vectors) and then size-reducing the basis, we should obtain a shorter and more orthogonal basis. δ is the measure for how reduced this basis is — the closer δ is to 1, the more the basis is LLL-reduced.

Note that the LLL algorithm keeps the lattice generated by the reduced basis the same as the original lattice. When size-reducing, each modified vector is an *integer* linear combination of the original basis vectors, so the lattice (also an integer linear combination of the basis vectors) remains the same. And changing the order of the basis evidently does not change the lattice.

How efficient is the LLL algorithm? If w define B to be the length of the longest vector in our original basis, the time complexity when $\delta \neq 1$ turns out to be $O(d^5 n \log^3 B)$ for that maximum length B. In order to maximize the time required for approximately solving the SVP, it is clear that one should use a lattice with an n-dimensional basis. Then $d = n$ and the LLL algorithm runs in $O(n^6 \log^3 B)$ time. We could also try to find original bases that are very large, so that $\log^3 B$ is very large as well. Importantly $\delta = 1$, which would result in the "most" LLL-reduced basis, is excluded, and indeed we do not know whether the LLL algorithm runs in polynomial time for that value of δ. The paper introducing the algorithm used $\delta = \frac{3}{4}$, and that is the standard value for δ.

Importantly, even though the LLL algorithm runs in polynomial time with respect to d, n, and B, it does *not* precisely solve the SVP, but can solve the *approximate* SVP. (This is because the shortest vector in an LLL-reduced basis is at most the length of the shortest vector times a certain constant c.) Of course, we may be extremely lucky after using LLL and find the shortest vector in our basis, but we do need to verify that one of the basis vectors is the solution to the SVP in our basis. And usually, we will not have found a shortest vector yet, nor a sufficiently short one. (And usually the reduced basis is not yet an optimal solution to SIVP either.) A simple algorithm for finding the shortest vector after LLL-reducing the known basis is by brute-force search.

Let $\mathbf{x}$ be the desired shortest vector, and express it as $\sum_{i=1}^{d} x_i \mathbf{b}_i$, where $\{\mathbf{b}_1, \mathbf{b}_2, \cdots, \mathbf{b}_d\}$ is an LLL-reduced basis (reordered from shortest to longest vectors). The x_i are integers. Clearly, $\|\mathbf{x}\| \leq B'$, where B' is $\|\mathbf{b}_1\|$ ($\mathbf{x}$ is either $\mathbf{b}_1$ itself, or some shorter vector). We rewrite $\mathbf{x}$ as a linear combination of $\{\mathbf{b}_1{}^*, \mathbf{b}_2{}^*, \cdots, \mathbf{b}_d{}^*\}$:

$$\mathbf{x} = \sum_{i=1}^{d} x_i \left(\mathbf{b}_i{}^* + \sum_{j=1}^{i-1} \mu_{i,j} \mathbf{b}_j{}^* \right) = \sum_{i=1}^{d} \left(x_i + \sum_{j=i+1}^{d} \mu_{j,i} x_j \right) \mathbf{b}_i{}^*.$$

Now we compute the projection of $\mathbf{x}$ onto the subspace with basis $\{\mathbf{b}_k{}^*, \mathbf{b}_{k+1}{}^*, \cdots, \mathbf{b}_d{}^* +\}$, which is:

$$\sum_{i=k}^{d} \left(x_i + \sum_{j=i+1}^{d} \mu_{j,i} x_j \right) \mathbf{b}_i{}^*.$$

A rather loose upper bound on the magnitude of this projection is B' itself. Hence:

$$\left\| \sum_{i=k}^{d} \left(x_i + \sum_{j=i+1}^{d} \mu_{j,i} x_j \right) \mathbf{b}_i^* \right\|^2 = \sum_{i=k}^{d} \left| x_i + \sum_{j=i+1}^{d} \mu_{j,i} x_j \right|^2 \|\mathbf{b}_i^*\|^2 \leq \|B'\|^2.$$

If we were to know the values for x_{k+1}, x_{k+2}, $\cdots$, x_d, then we obtain a linear inequality with the only unknown variable being x_k, so we can easily check through brute-force all possibilities for x_k (remember, the x_i are integers, so we only have a finite number of possibilities). Explicitly, the inequality is:

$$\left| x_k + \sum_{j=k+1}^{d} \mu_{i,k} x_i \right| \leq \frac{\sqrt{B'^2 - \sum_{i=k+1}^{d} \left| x_i + \sum_{j=i+1}^{d} \mu_{j,i} x_j \right|^2 \|\mathbf{b}_i^*\|^2}}{\|b_k^*\|}.$$

The solutions for x_k is simply a bounded, closed interval. Using this method, we are able to find every x_i by brute force: In the inequality above, first let $k = d$ and find all possibilities for x_d. Then, in each possibility, use the inequalities with $k = d - 1$ and then find the possible values for x_{d-1}. In each case, find the possibilities for x_{d-2}, and continue.

Clearly, this algorithm is very inefficient — it is a brute-force algorithm after all. Indeed, the time complexity for this method after the LLL-algorithm has been used is $2^{O(d^2)}$ (in addition to some negligible polynomial terms). If we were to use a stricter (but by itself less efficient) basis-reduction algorithm, as Ravidran Kannan used in 1983, then the time complexity becomes $2^{O(d \log d)}$. In 2001, Miklós Ajtai found a sieving algorithm for solving the SVP in $2^{O(d)}$ time complexity. This is still not polynomial time, so at this point the LLL or other basis-reduction algorithms have not yet completely solved the SVP. Perhaps there *may* be an explanation: Unlike the discrete logarithm or factoring problems, which are only "NP," the SVP can be shown to be "NP-hard."

4 NP-Hardness Of The SVP

In computer science, there are many different problem classifications. These include NP, NP-hard, and NP-complete. The most important of these classifications for our purposes is the notion of NP-hardness.

Definition 2. (NP-Hardness)

A problem H is NP-hard if for any NP problem G, there exists a polynomial-time algorithm that reduces G to H — in other words, there exists a polynomial-time algorithm for solving all NP problems when knowing the solution for H from an oracle.

Theorem 1. The Short Vector Problem is NP-hard under "randomized reductions" — i.e. the algorithms that reduce NP problems to the SVP are *probabilistic* and not *deterministic* (so each time we run the reduction algorithm for a particular NP problem, we obtain an SVP with different parameters, such as a different lattice basis).

To prove this theorem in his 1998 paper [1], Miklós Ajtai proves that the restricted subset sum problem, an *NP-complete* problem, can be reduced to the SVP. The specific definition of NP-completeness is not essential here. The key is that any NP problem can be reduced to an NP-complete problem, implying that if we can prove that an NP-complete problem (like the restricted SSP) reduces to the SVP, then we prove that all NP problems reduce to the SVP too, meaning that SVP is NP-hard. The restricted subset problem queries whether a subset of a certain set of (not necessarily distinct) integers can sum to a chosen target. The proof of the NP-hardness of the SVP is quite complicated, requiring many other lemmas regarding both the restricted subset sum problem and "hypergraphs," so if the reader wishes to see the proof, they can consult Ajtai's full paper.

As an example of an NP problem that can be solved with the SVP, we consider the factoring problem.

Theorem 2. If we have an oracle for solving the SVP (for any dimension), we can solve the factoring problem within polynomial time.

We present the rough framework for this proof — again, to ensure that we can actually follow these steps, we need to prove many foundational results. Let the positive integer we want to factor be N. Let $p_1, p_2, \cdots, p_k$ be the first k primes, all bounded by $(\log N)^a$ for some real a. Then construct the following lattice (where each column is a basis vector for the lattice) for some real b:

$$\begin{pmatrix} \sqrt{\log p_1} & 0 & \cdots & 0 \\ 0 & \sqrt{\log p_2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sqrt{\log p_k} \\ N^b \log p_1 & N^b \log p_2 & \cdots & N^b \log p_k \end{pmatrix}.$$

We want to find the vector in this lattice closest to $(0, 0, \cdots, N^b \log N)$ (findable if the CVP can be solved, which is solvable if the SVP is). Suppose that the scalars for each basis vector are $x_1, x_2, \cdots, x_k$. Then the shortest vector is $(x_1\sqrt{\log p_1}, x_2\sqrt{\log p_2}, \cdots, N^b(x_1 \log p_1 + x_2 \log p_2 + \cdots + x_n \log p_n))$. We are most interested in the final component, which can be rewritten as $N^b \log(p_1^{x_1} p_2^{x_2} \cdots p_n^{x_n})$. This component must be very close to $N^b \log N$, so $N^b \log(p_1^{x_1} p_2^{x_2} \cdots p_n^{x_n}) - N^b \log N = N^b(\log(\frac{p_1^{x_1} p_2^{x_2} \cdots p_n^{x_n}}{N})$ is very close to 0. We simplify and exponentiate to find that $\frac{p_1^{x_1} p_2^{x_2} \cdots p_n^{x_n}}{N}$ is close to $e^{N^b 0} = 1$. Hence, for certain $x_1, x_2, \cdot, x_n$, $p_1^{x_1} p_2^{x_2} \cdots p_n^{x_n}$ is close to N. So there exists a $(\log N)^a$-smooth number fairly close to N.

Repeating this argument for other a's (and therefore other smallest primes $p_1, p_2, \cdots, p_k$), we find multiple "supersmooth" numbers very close to N. Of course, these results seem useful for factoring N, but the fastest algorithm that uses supersmooth numbers (the number field sieve) does not run in polynomial time. Instead, we must prove other results on the supersmooth numbers generated by this process, which eventually guarantees us a polynomial time reduction from the factoring problem to the SVP. For a rigorous proof, see Claus-Peter Schnorr's paper [6]. He even reduces the discrete logarithm problem to the SVP!

5 Lattice-Based Cryptosystem

Since the SVP (and therefore the CVP and SIVP) are NP-hard, we would expect the problem to be a good foundation for cryptosystems. Indeed, there are a multitude of different lattice-based cryptosystems, like NTRUEncrypt or GGH, and also many designed for key exchanges or sign-and-encrypt protocols. We present here one of the first such cryptosystems, one of three published by the same Ajtai with Cynthia Dwork in a 1997 paper [2].

Select a sufficiently large dimension n in which this cryptosystem will be constructed. Given positive M, d, we define an n-dimensional "(d, M)-lattice" L, by which we mean a lattice L satisfying the following conditions: First, there exists an $(n-1)$-dimensional sublattice $L' \subset L$ that has a basis $\{\mathbf{b}_1, \mathbf{b}_2, \cdots, \mathbf{b}_{n-1}\}$ such that $\|\mathbf{b}_i\| \leq M$. Second, if H is the $(n-1)$-dimensional subspace of $\mathbb{R}^n$ that contains all points in L' (in other words, H is the vector space — not lattice — generated by the same basis as L'), then for the unit vector $\mathbf{h}$ orthogonal to all vectors in H, the minimal nonzero distance d_L for which $\{\mathbf{u} + d_L\mathbf{h} \mid \mathbf{u} \in H\} \cap L \neq \emptyset$ must be at least d. Let $\mathcal{L}$ be the set of all (d, M)-lattices for certain M and $d > n^c M$ where $c > 5$. Note that d can be any positive number, but $d > M$ guarantees that there is only one (d, M)-lattice for L, and $d > n^c M$ allows us to use the *hidden hyperplane assumption*: If an attacker knows the basis for a randomly selected (d, M)-lattice $L \in \mathcal{L}$, and even the values of M and d, it is computationally difficult for him or her to find the corresponding $(n-1)$-dimensional sublattice L' (which can be called a "hyperplane," the namesake of our assumption).

We now define our cryptosystem that builds upon this hidden hyperplane assumption. First, select some M, which we make public. We randomly generate the corresponding sublattice $L' \subset L$ to some currently undefined L, with the L' basis $\{\mathbf{b}_1, \mathbf{b}_2, \cdots, \mathbf{b}_{n-1}\}$ being our private key. Again, we need $\|\mathbf{b}_i\| \leq M$. Let H be the $(n-1)$-dimensional subspace of $\mathbb{R}^n$ that contains all points in L' again. Next, select a positive $d > n^c M$ where $c > 5$ — and d is also public. Then, select a private number d_L between d and $2d$, inclusive. With $\mathbf{h}$ being the unit vector orthogonal to the vectors in H again, we define an n-dimensional lattice L generated by the basis of L' and the vector $d_L\mathbf{h}$. This L is now an (d, M)-lattice by construction. We now construct a random basis for L and let it be, in conjunction with M, our public key. An attacker cannot derive L' by only knowing the basis for L and the value of M, so we can safely use L' to encrypt messages. Here, the hidden hyperplane assumption corresponds to the approximate SIVP, as our basis vectors must either be bounded by M (for the $\mathbf{b}_i$'s) or $2d$ (for $d_L\mathbf{h}$). If we could solve the approximate SIVP, we could exhaust all possibilities for the basis of L' and eventually break the cryptosystem.

Now we define a simple encryption method if we want to send binary strings. We first define a positive real number $R = n^3 M$ and a positive integer $m \geq 4n$. We use $\mathbf{x} + \sum_{i=1}^{m} \mathbf{v}_i$ if we want to encrypt 0, where $\mathbf{x}$ is a lattice vector with each component within $[0, K]$ and the $\mathbf{v}_i$'s are random vectors such that $\|\mathbf{v}_i\| \leq R$ (imagine a random walk starting at $\mathbf{x}$ with m steps of length at most R). Here, K must be quite large and also greater than $2\sqrt[n]{d}$, and this lower bound, through Minkowski's Theorem in n dimensions, guarantees that a lattice vector $\mathbf{x}$ satisfying the desired conditions exists. If we want to encrypt 1, select a random vector $\mathbf{x}$ (not necessarily in the lattice) with each component within $[0, K]$.

Decryption is now very simple: Since the decrypter knows the basis of L' (and therefore the unit vector $\mathbf{h}$ orthogonal to the subspace H from before) and also the value of d_L, he or

she can compute $\frac{\mathbf{h} \bullet \mathbf{v}}{d_L}$, where $\mathbf{v}$ is the encrypted vector that was sent. If the result is within $\frac{mR}{d_L}$ of an integer, then a 0 is decrypted. Otherwise, a 1 was certainly encrypted.

Why does this decryption method work? The expression $\mathbf{h} \bullet \mathbf{v}$ measures the length of the projection of $\mathbf{v}$ onto $\mathbf{h}$. The projection of any lattice vector in L onto $\mathbf{h}$ must have a length that is a multiple of d_L, by the construction of L (the basis vector $d_L\mathbf{h}$ is orthogonal to all other basis vectors). And if 0 was encrypted, the projection can only be increased by at most mR if we happened to only move in the direction of $\mathbf{h}$ for each random step. Thus, the projection $\mathbf{h} \bullet \mathbf{v}$ must be within mR of a multiple of d_L, or in other words, $\frac{\mathbf{h} \bullet \mathbf{v}}{d_L}$ is within $\frac{mR}{d_L}$ of an integer. In short, we want to verify whether the sent vector is close to a lattice point (which is similar to the approximate CVP). Projecting onto a random vector other than $\mathbf{h}$ will tend to result in the projected lattice points being too close together, making the protocol above less effective.

Do note that a vector that is supposed to represent a 1 might actually be decrypted as a 0 (for instance, if the encryption vector is sufficiently close to a lattice point). For the sufficiently large K selected before, the probability of such errors should be quite low. Nevertheless, the encrypter can use some coding theory (such as repeating a message multiple times, where most of the copies of the message would probably have no errors) in order to prevent loss of information for our decrypter.

Cryptosystems based on lattice problems are particularly unique in the grand scope of cryptography because of the argument used to show that they are secure. In most systems, including Diffie-Hellman or RSA, we might not be able to simply select random public and private keys because an "average-case" instance of the discrete logarithm or factoring problems might be vulnerable to a certain attack. Indeed, attacks like Pohlig-Hellman (for the discrete logarithm problem) or the quadratic sieve (for factoring) work best when certain conditions — such as whether a prime factor of a key is close to smooth numbers — happen to be satisfied.

However, lattice-based cryptosystems (especially those that exploit the SVP) are not as vulnerable to attacks on "average-case" examples of a lattice problem. In order for an attack to solve the SVP in an average case, it must also be able to solve the SVP in a worst-case scenario, where, for instance, the basis used is particularly difficult to work with. Ajtai and Dwork prove in the same paper that there exists a worst-case average-case equivalence for the SVP: The time complexity for solving an average-case instance of the SVP is equal to the time complexity for solving a worst-case SVP. So, people using a lattice-based cryptosystem must be less selective about what public or private keys they choose — the underlying example of the SVP will still be difficult to solve (although, of course, stupid instances of the SVP should be avoided at all costs!). The authors use this result to prove that their cryptosystems are secure.

6 Conclusion

Currently, lattice-based cryptosystems seems to be the future of quantum-secure cryptography — for many lattice problems, including the SVP, there are no known polynomial-time algorithms that can solve them. Even the most efficient attacks, like the LLL algorithm, only have exponential time-complexity, and the SVP is even an NP-hard problem. Yet, the

permanence of lattices in cryptography is all but certain. Indeed, there was some anxiety when in 2024, Yilei Chen published a paper [3] that claimed to have cracked a form of the SVP in polynomial time on a quantum computer. Fortunately, an issue in one of the steps for the algorithm made the result (as of yet) false, but since lattice-based cryptography and quantum computing are still relatively recent, there may still be some vulnerability in SVP and other currently difficult lattice problems. Perhaps they might be proven even quantum-secure eventually, or valid attacks may be discovered, but only time will tell.

References

[1] Miklós Ajtai. The shortest vector problem in L_2 is NP-hard for randomized reductions. *Electronic Colloquium on Computational Complexity*, 1997.

[2] Miklós Ajtai and Cynthia Dwork. A public-key cryptosystem with worst-case/average-case equivalence. *STOC: Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, 1997.

[3] Yilei Chen. Quantum algorithms for lattice problems. Technical Report 555, Cryptography ePrint Archive, 2024.

[4] Daniele Micciancio. Efficient reductions among lattice problems. *Proceedings of SODA*, pages 84–93, 2008.

[5] Phong Q. Nguyen and Brigitte Vallée, editors. *The LLL Algorithm*. Springer, New York, New York, 2010.

[6] Claus-Peter Schnorr. Factoring integers and computing discrete logarithms via diophantine approximation. *DIMACS Series In Discrete Mathematics and Theoretical Computer Science*, 13, 1993.