The Shortest Vector Problem and Applications to Cryptography

Aaron Hall and Sixian Liu

May 18, 2018

Abstract

In this paper, we present an overview of the shortest vector problem (and other lattice problems) and its applications to post-quantum cryptography. We begin with an overview of lattices and the difficult optimization problems that result from their special structure. We then present a fairly brief summary of the three main methods for solving SVP and other lattice problems: enumeration, Voronoi cells, and sieving. To supplement this, we also present a more detailed study of the list sieve, one of the algorithms based on sieving. Finally, we list a few cryptosystems that are based on lattice problems and discuss their basic functionality, security, and practicality.

1 Introduction

With the knowledge that Shor's algorithm can solve both the factoring problem and the discrete logarithm problem in polynomial time, one might wonder what problems are hard enough that they can be used for public-key cryptography. One promising answer is lattice problems, which can be very difficult to solve due to the unique nature of lattices. These problems include finding the shortest (nonzero) vector of a lattice (SVP), finding the closest lattice vector to some target vector (CVP), and finding a set of short independent vectors in a lattice (SIVP). Informally, the difficulty comes from the fact that only integral linear combinations of the basis vectors are allowed, so if all we know about a lattice is a "low-quality" basis (e.g., one with long, non-orthogonal vectors), it is very hard to see what the small vectors in the lattice are. More formally, many of these problems are known to be NP-hard, so it is commonly believed that there are no polynomial-time algorithms for these problems. NP-hardness is also a good indicator that the algorithm is resistant to polynomial-time attacks by a quantum computer.

The goal of this paper is to present the main findings about one of these problems: the shortest vector problem, or SVP. We will present an overview of the problem, including a discussion of its NP-hardness and the relation to other lattice problems. Then, we will discuss the three main techniques for solving it: enumeration, Voronoi cell computation, and sieving. For each of these we will give an overview of the technique, discuss the advantages and disadvantages compared to the other two techniques (including time/space complexity), and present examples of specific algorithms.

2 Introduction to SVP

2.1 Preliminaries

Let $\mathbb{R}$ denote the set of real numbers. For an integer n, $\mathcal{L} = \{\sum_{i=1}^{n} a_i b_i \mid a_i \in \mathbb{Z}\}$ is the n-dimensional lattice in $\mathbb{R}^n$ generated by the basis vectors $b_1, \ldots, b_n$. Note that while $\mathcal{L}$ is an additive subgroup of $\mathbb{R}^n$, it is *not* a subspace because only integral linear combinations are allowed. As such, $\mathcal{L}$ loses many of the linear-algebraic properties that $\mathbb{R}^n$ has. For a vector $x = (x_1, \ldots, x_n) \in \mathcal{L}$, the L_2 norm is defined by $\|x\| = \sqrt{\sum_{i=1}^{n} |x_i|}$. The goal of the shortest vector problem is to find the $x \in \mathcal{L}$ such that $\|x\| = sh(\mathcal{L})$[1], where $sh(\mathcal{L})$ denotes the length of the shortest non-zero vector in $\mathcal{L}$.

2.2 NP-Hardness and Relationship to Other Lattice Problems

mAs discussed in [10] and other sources, SVP is NP-hard. This comes from the fact that SVP can be fairly easily reduced to the closest vector problem (CVP), and CVP has been known to be NP-hard for quite some time. Furthermore, other NP-hardness proofs show that there is no α-approximation algorithm for SVP (with $\alpha \in [1, \sqrt{2}]$) unless P = NP. One should note that there are approximation algorithms that can solve SVP in less time but with a worse approximation ratio; for example, the Lenstra-Lenstra-Lovász (LLL) algorithm for basis reduction can be used solve SVP in polynomial time but with an approximation ratio of $2^{O(n)}$. Because SVP can be polynomial-time reduced to many other lattice problems (including CVP and SIVP), these NP-hardness results are useful in showing that all of these algorithms are very difficult to solve. The following table describes the time and space complexity of existing algorithms for SVP.

Algorithm Name	Classical		Quantum	
	log_2 (Time)	log_2 (Space)	log_2 (Time)	log_2 (Space)
Enumeration	$\Omega(nlogn)$	$O(logn)$	$\Omega(nlogn)$	$O(logn)$
Voronoi Cell	2.000n	1.000n	2.000n	1.000n
List Sieve-Birthday	0.802n	0.401n	0.602n	0.401n

Table 1. A comparison of time and space complexities of SVP algorithms [2]

3 Overview of Methods for Solving SVP

In this section, we present overviews of the three main techniques for solving SVP: enumeration, Voronoi cell computation, and sieving.[3]

3.1 Enumeration

The technique of enumeration is exactly what it sounds like: given a bounded region of $\mathbb{R}^n$ (e.g., a rectangle or an ellipsoid), we list all lattice points in that region. Thus, if we perform enumeration

[1] This notation is used in Ajtai, Kumar, and Sivakumar's paper "A sieve algorithm for the shortest lattice vector problem" (see [3]).

[2] This table is a part of the table in "Finding shortest lattice vectors faster using quantum search" by Thijs Laarhoven, Michele Mosca, Joop van de Pol.

[3] Many of these descriptions come from Daniele Micciancio's website `http://cseweb.ucsd.edu/~daniele/LatticeLinks/`, which has proven very convenient for us because it provides both high-level descriptions of the methods and links to papers discussing specific algorithms for them.

for some sufficiently large region containing the zero vector, the shortest vector is guaranteed to appear in the list, so it will be easy to search the list for the vector we want. Unlike the Voronoi cell and sieving methods discussed below, enumeration runs in polynomial space. However, this comes at the cost of a worst-case runtime of $n^{O(n)}$, which is actually the theoretical lower bound of deterministic polynomial-space algorithms for solving SVP (and related problems). This runtime can be reduced by preprocessing the lattice (often by reducing the basis), or by using heuristics (such as different enumeration strategies or pruning methods) that show good practical performance but aren't known to lead to a provable asymptotic speedup.

3.2 Voronoi Cell Computation

For a nonzero vector $v \in \mathcal{L}$, define the *half-set* H_v to be the set of all $u \in \mathbb{R}^n$ such that u is closer to 0 than to v, i.e., $\|u\| \leq \|u - v\|$. Then, the *Voronoi cell* $\mathcal{V}$ of $\mathcal{L}$ is a convex polytope defined by by

$$\mathcal{V} = \bigcap_{v \in \mathcal{L} \setminus \{0\}} H_v.$$

We typically represent the Voronoi cell as a collection of "Voronoi-relevant" vectors v, i.e., v such that $v/2$ lies on the center of a facet of $\mathcal{V}$; it turns out that the half-sets H_v for Voronoi-relevant vectors v are sufficient for representing $\mathcal{V}$ (see [12]). It is then easy to see how to reduce SVP to the problem of computing the Voronoi cell of $\mathcal{L}$: once we have $\mathcal{V}$, any vector in $\mathcal{L} \cap \bar{\mathcal{V}}$ (where $\bar{\mathcal{V}}$ is the topological closure of $\mathcal{V}$, or the union of $\mathcal{V}$ with its boundary) is a shortest vector in $\mathcal{L}$.

With this approach in mind, we have now reduced SVP to two main sub-problems, which can unfortunately be very hard. First, we need a way to compute the Voronoi cell of a lattice, and this problem is often considered harder than SVP itself. Second, given the Voronoi cell $\mathcal{V}$, usually described by half-sets for Voronoi-relevant vectors, we need a way to find a vector in $\mathcal{L} \cap \bar{\mathcal{V}}$. It turns out that a Voronoi cell can have at most $2(2^n - 1)$ facets, and so it can potentially require exponential space to represent. Because of these problems, Voronoi methods for solving SVP require exponential time and space to execute, but they are still noticeable improvements over the runtimes of enumeration algorithms.

Micciancio and Voulgaris present in [12] a relatively simple deterministic algorithm for computing the Voronoi cell of a lattice in single exponential time, which they then use in algorithms for solving SVP, CVP, and other lattice problems. The algorithm first reduces the lattice basis using the LLL algorithm. Then, using the reduced basis $B = \{b_1, \ldots, b_n\}$, the algorithm goes through the sublattices generated by $B_k = \{b_1, \ldots, b_k\}$ (for $k = 2, 3, \ldots, n$) of the lattice and computes their Voronoi cells. These Voronoi cells are computed by finding short vectors using a CVPP (closest vector problem with preprocessing) algorithm, then filtering out the vectors that are not Voronoi-relevant. This algorithm runs in time $O(2^{3.5n})$ and uses $O(2^n)$ space. Other algorithms based on this one have been developed, including a Las Vegas algorithm presented in [6], which runs faster but at the cost of being randomized.

3.3 Sieving

The intuitive way of understanding sieve algorithms is by imaging this kind of algorithms as a sieve. The algorithm begins with a pool of exponentially many random lattice points within a large but bounded region in $\mathbb{R}^n$. Then we let those points go through finer and finer sieves so that shorter and shorter vectors are selected from the pool. The earliest idea of a sieve algorithm (known as *AKS*

sieve) is presented by [3]. More specifically, we generate $2^{O(n)}$ random points within a sufficiently large parallelepiped. We perform the sieving procedure by producing a new pool of lattice vectors with the longest vector having the half of its original length. We perform this procedure iteratively until all of the vectors in the pool lies in the ball of a certain radius. Then, by examining this pool, we can gain the shortest vector by taking the difference of a short vector and the nearest neighbor. This earliest version of sieve algorithm can compute the shortest nonzero vector with time complexity 2^{cn} and probability close to 1, where c is an absolute constant.

However, the major problem of this algorithm is that it takes exponential space. As the dimension increases, sieve algorithm uses incredibly large computational resources while enumeration method only takes polynomial space. Besides AKS sieve, there are several heuristic variants. The best known variant, called *list sieve*, is proposed in the paper [11].

Unlike AKS sieve, list sieve starts with a list of points, and then adds new lattice points to it. We iteratively add new point to the list in order to produce shorter and shorter vectors until two vectors within a certain distance are found. The upper bound of the time and space complexity of list sieve can be estimated from the upper bound of the length of this list. This variant is also a probabilistic algorithm but improves the worst-case time and space complexity bounds.

A more practical variant based on list sieve called *Gauss sieve*, which is also proposed in [11], gives a new technique without errors (perturbations). The space complexity of this variant is bounded by $2^{0.402n}$ in theory and $2^{0.21n}$ in practice. However, there is no bounded running time since we cannot show this algorithm will not produce nonzero vectors due to the no perturbation. In practice, the running time of the algorithm seems to be between quadratic and cubic in the list size, but there is no evidence on worst-case upper bound. Since we cannot give the number of samples we need for the success of the algorithm, the algorithm usually terminates after a certain number of collisions, i.e. the number of lattice points near one point in the list. The estimation of collision varies with different lattices, but we usually set the number $\approx 10\%$ of the list size.

List Sieve

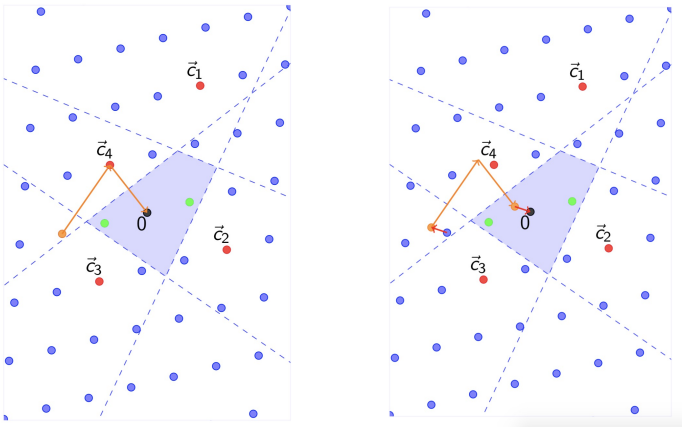
In the following paragraphs, we will take a closer look at *List Sieve*, including the techniques are used in this algorithm and the idea of proving time and space complexity. There are quite a lot of parameters setting in the original paper [11], but we will just examine some important parameters and see how they affect the algorithm.

Procedure

Overview: The algorithm iteratively builds the list L of lattice points by adding a point to the list. The points already in the list are never changed or removed. At every iteration, we randomly pick a vector v and subtract from it by the lattice vectors already in the list L until the length of v cannot be reduced any more. Then, v is added to the list L. The algorithm will generate shorter and shorter until two lattice points within distance μ are produced.

Sampling: We sample 2^{cn} points with $\|p\| \leq R_0$ (the sphere of radius R_0). Then, we cover the samples with spheres of radius $R_1 < R_0$ centered at the samples, and subtract the center to gain the shorter and shorter vectors. We introduce a standard perturbation technique. Instead of generating a random lattice point v,we generate a perturbed lattice point $p = v + e$ (Figure.1), where e is a small error vector with $\|e\| \leq \xi\mu$. Here, ξ is a parameter bounded by $0.5 < \xi < 0.7$. In

the figure below, instead of sampling v (Figure 1(a)), we sample p such that $v = p - e \in$ L (Figure 1(b)). [4]



(a) Sample the vector v (b) Sample the vector $p = v + e$

Figure 1: Sample with perturbation

List reduction: The length of p is reduced by the points already in the list, and the vector added to the list is $v = p - e$. Note that vectors from L can be chosen in any order.

Termination: As we mentioned, the algorithm terminates when the two lattice points within distance μ are produced. However, the algorithm also terminates after reducing K samples if no lattice vector of norm at most mu is found. Here, K is the maximum number of samples the algorithm will achieve this time.

Analysis of the Complexity

Theorem 3.1. Let ξ be a real number with $0.5 < \xi < 0.7$ and $c_1(\xi) = log(\xi + \sqrt{\xi^2 + 1} + 0.401)$, $c_2(\xi) = 0.5log(\xi^2/(\xi^2 - 0.25))$. List Sieve can solve SVP for the worst case in space $O(2^{c_1 n})$ and in time $O(2^{(2c_1+c_2)n})$.

We set the parameter $\xi \approx 0.685$ which gives the best running time. In this case, the space complexity is $< 2^{1.325n}$, and time complexity $< 2^{3.199n}$.

Space Complexity

Although the perturbation technique increases the space complexity of the algorithm, what is nice about it is that we will never get two lattice points very close to each other. Due to the perturbation, we can prove a lower bound of the angles between points of similar norm. Also, the distance of the points in the list L can be bounded by μ. This gives a lower bound on the angles between shorter points. The space complexity of the algorithm are determined by both of bounds above, which gives a overview bound on the angle between two lattice points of similar norm.

As we mentioned above, the perturbation error $\|e\| = \xi\mu$. If we set a certain μ in the algorithm (how short the vector we want to find), the parameter ξ is connected to the size of the perturbation.

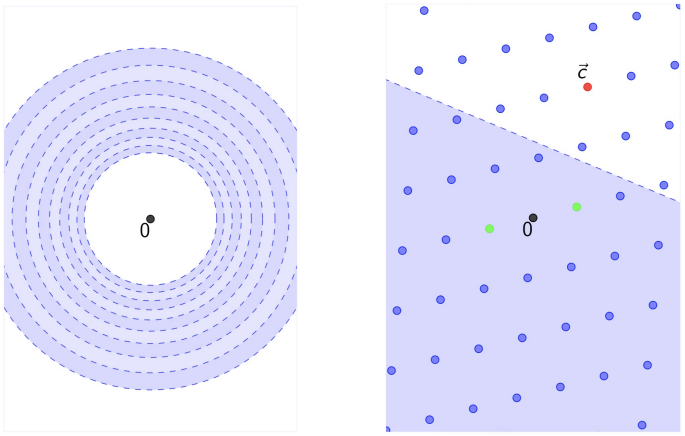
[4]The figures here and below are from the original slides by Panagiotis Voulgaris Daniele Micciancio in 2010, which can be found in this website $https://cseweb.ucsd.edu/\ pvoulgar/files/pres_sieve.pdf$.

The small ξ will cause smaller perturbation; thus lower space complexity. The lower bound of ξ, which is 0.5, makes sure that $B(p, \xi\mu)$ contains at least one lattice point. The basis idea of the proof (I am going to write a brief summery here) the space complexity is based on the slides and paper by Micciancio and Voulgaris in 2010 [11].

Theorem 3.2. Kabatiansky, Levenshtein 1978
Let S be the set defined as S:= $\{c_i, c_j : \phi_{c_i,c_j} > \phi_0\}$, where ϕ_{c_i,c_j} is the angle between c_i, c_j, then $|S| \leq 2^{k(\phi_0)n+o(n)}$.

Theorem 3.2 shows that the lower bounds between the angle in the list determines the upper bound on the size of the list L. We now divide the list into groups and bound the size of each group. First, we chose a ball of radius $\mu/2$, i.e. $S_0 = B(\mu/2)$, and then divide the space in to spherical thin shells S_i (Figure 2(a)). Since the algorithm terminates when two lattice points within μ are found, $|S_0| = 1$. Now let C be the space covered by $poly(n)$ shells.

Definition 3.1. Halfspaces (Figure 2(b))
c-halfspace is defined to be $\|p - c\| < \|p\|$;
0-halfspace is defined to be $\|p - c\| \geq \|p\|$.



(a) Sphere and thin shells (b) Halfspaces

Figure 2: Divide the space

Consider only one thin shell S_i, and a point c_i in $C \cap S_i$. Then, a new point produced is in 0-halfspace of c_i due to the perturbation p. We choose $c_j \in$ 0-halfspace $\cap S_i$. The upper bound of $\cos(\phi_{c_i,c_j})$ is given by

$$\cos(\phi_{c_i,c_j}) \leq 1 - \frac{1}{2(\xi + \sqrt{\xi^2 + 1})^2} + o(1),$$

which is greater than 0.5; or equivalently, $\phi \geq 60°$. The proof of such bound on $\cos(\phi_{c_i,c_j})$ is omitted due to the length and complexity of math (actually, half page in the original paper). The parameters used in the proof includes p, e, ξ, μ and new parameters R, γ(the factor bounds the norm of the vector w.r.t. R), δ (the shorter factor). The basic idea is using $cos(\phi_{c_i,c_j}) = v_i v_j / \|c_i\| \|c_j\|$, and then do a bunch of substitutions of the bounds we discussed above. Finally, by Theorem 3.2, we are able to compute the cardinality of the set S and therefore the space complexity of the algorithm.

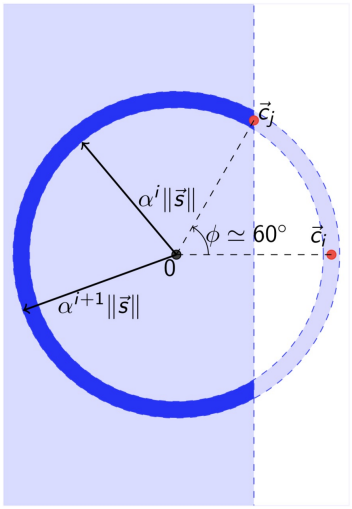


Figure 3: The lower bound on the angle

Time Complexity

Define *event* S as "$v_i = p_i - e_i$ is a solution of SVP (i.e. $0 < dist(L, v_i) \leq \mu$)". The main idea of the proof is that after a certain number of samples the *event* S will happen with high probability. We will just state the results without proof. If $0.5 < \xi < 0.7$, the algorithm will output a lattice point with norm $\leq \xi$ with probability close to 1 once $K \geq O(2^{(c_1+c_2)}n)$, where K is the number of samples. Then, the running time of the algorithm is given in Theorem 3.1. Note that since we use the perturbation technique in list Sieve and set μ at the beginning, we are able to prove that the algorithm will not produce nonzero vector. Therefore, unlike Gauss Sieve, we can prove the worst case time complexity.

4 Overview of Lattice-Based Cryptosystems

As mentioned before, the primary reason we care about lattice problems is that they can be used as a foundation for post-quantum cryptography. Most of the common cryptosystems in use today rely on the classical difficulty of factoring the product of two large primes or finding the discrete logarithm of an element of a cyclic group, but because of Shor's algorithm, both of these problems can be solved in polynomial time. Even though we are far away from being capable of building a quantum computer that can factor a 2048-bit RSA modulus, this is still enough of a concern that we want to find a sufficiently strong cryptosystem that can eventually replace RSA, ElGamal, and other cryptosystems threatened by Shor's algorithm. Lattice problems such as SVP and CVP are promising candidates for the basis of new cryptosystems, because we are not aware of any efficient algorithm (classical or quantum) that can solve them. Indeed, several lattice-based cryptosystems have already been proposed, and we will look into a few of them in the next few sections.

4.1 Ajtai and Dwork's SVP-Based Cryptosystem

The first lattice-based cryptosystem we will look into is that of Ajtai and Dwork, presented in [2]. This is a public-key encryption scheme whose security is based on the difficulty of SVP. In short,

the private key is a collection of hidden hyperplanes and the public key is a method of generating a point near one of the hyperplanes. Encryption is done bit-by-bit: to encrypt a 0, the sender uses the public key to generate a point close to one of the hyperplanes, and to encrypt a 1, the sender generates a point uniformly at random (within some large cube). Then, to decrypt a ciphertext (which will be a series of vectors), the receiver looks at each vector: if it is sufficiently close to the hyperplanes the decryption algorithm outputs 0, and otherwise it outputs 1.

This scheme is interesting for a couple of reasons. First, this scheme is based directly on the difficulty of SVP, since one who can find the hyperplanes from the public key has demonstrated the ability to solve worst-case SVP. Second, and more importantly, Ajtai and Dwork proved that there is an equivalence between the difficulties of average-case and worst-case SVP, which is important because average-case difficulty is the primary interest in cryptography (in contrast to, say, complexity theory, where worst-case hardness is the most interesting). However, as a practical cryptosystem, this scheme is not particular useful. This is because there is a small chance that the encryption of a 1 will output a vector close to the hyperplanes, so upon decryption a 0 will be outputted instead of 1. Thus, while Ajtai and Dwork's cryptosystem is interesting on a theoretical level, we will want to look at other lattice-based cryptosystems.

4.2 The GGH Encryption and Digital Signature Schemes

In [7], Goldreich, Goldwasser, and Halevi present a cryptosystem that is based on the difficulty of CVP. The private key is a "good" lattice basis B (i.e., one whose vectors are short and nearly orthogonal), and the public key is a "bad" basis $B' = UB$ (where U is some unimodular matrix). To encrypt a message, which is represented as a row vector m, the sender computes $mB' + e$, where e is some error, and to decrypt the ciphertext, the receiver uses B and the Babai rounding technique to recover m. To sign a message, the sender uses B to generate a lattice point near the message vector, and the receiver verifies that the point is in the lattice using B' and that it is sufficiently close to the message.

Unlike the previous scheme, the GGH scheme does not have a probability of decrypting incorrectly, assuming a sufficiently good basis B is chosen (which ensures that the Babai rounding technique will work). However, Nguyen in [14] discovered that the encryption scheme has major flaws. First, each ciphertext leaks information about the corresponding message. In particular, reducing the ciphertexts modulo a particular value yields a system of equations that, when solved, yields the message modulo this value. Second, the problem of decrypting ciphertexts can be reduced to a significantly easier version of CVP, so the scheme is not as secure as claimed. Finally, even Goldreich, Goldwasser, and Halevi noted in [7] that two different messages may have the same signature, if they are sufficiently close to each other, and this makes existential forgeries relatively trivial. Thus, the raw GGH schemes are far from being practically secure, but like Ajtai and Dwork's scheme, it is still interesting to study as a potential starting point for other schemes.

4.3 Other Lattice-Based Cryptosystems

In this section we briefly talk about two lattice-based cryptosystems that are generally beyond the scope of this paper, but are nevertheless worth mentioning. The first scheme is NTRUEncrypt and NTRUSign. Developed by Hoffstein, Pipher, and Silverman in [8], this scheme makes use of a polynomial mixing system and reduction modulo two values, and its difficulty is based on the hardness of SVP. It is believed that the NTRU schemes are secure for appropriate parameter

choices. The second scheme, which is actually a general class of schemes, is based on the difficulty of two more lattice problems: learning with errors (LWE) and ring learning with errors (RLWE). These are difficult problems in machine learning which relate to learning a secret vector $\mathbf{s}$ from polynomially many values $\langle \mathbf{a}, \mathbf{s} \rangle + e$, where $\mathbf{a}$ is a random vector and e is a random error (and for RLWE, we specifically work in rings of polynomials over finite fields). There are a number of cryptosystems based on (R)LWE, including key exchange protocols and public key encryption schemes, and like NTRU, these show a relatively large amount of promise for use as real-world cryptosystems.

References

[1] Aggarwal, D., Dadush, D., Regev, O., and Stephens-Davidowitz, N. Solving the shortest vector problem in 2^n time via discrete gaussian sampling. *CoRR abs/1412.7994* (2014).

[2] Ajtai, M., and Dwork, C. A public-key cryptosystem with worst-case/average-case equivalence. In *Proceedings of the Twenty-ninth Annual ACM Symposium on Theory of Computing* (New York, NY, USA, 1997), STOC '97, ACM, pp. 284–293.

[3] Ajtai, M., Kumar, R., and Sivakumar, D. A sieve algorithm for the shortest lattice vector problem. In *Proceedings of the Thirty-third Annual ACM Symposium on Theory of Computing* (New York, NY, USA, 2001), STOC '01, ACM, pp. 601–610.

[4] Ajtai, M., Kumar, R., and Sivakumar, D. Sampling short lattice vectors and the closest lattice vector problem. In *Proceedings 17th IEEE Annual Conference on Computational Complexity* (2002), pp. 41–45.

[5] Chen, Y., Chung, K., and Lai, C. Space-efficient classical and quantum algorithms for the shortest vector problem. *CoRR abs/1709.00378* (2017).

[6] Dadush, D., and Bonifas, N. *Short Paths on the Voronoi Graph and Closest Vector Problem with Preprocessing.* pp. 295–314.

[7] Goldreich, O., Goldwasser, S., and Halevi, S. Public-key cryptosystems from lattice reduction problems. Cryptology ePrint Archive, Report 1996/016, 1996. `https://eprint.iacr.org/1996/016`.

[8] Hoffstein, J., Pipher, J., and Silverman, J. H. Ntru: A ring-based public key cryptosystem. In *Lecture Notes in Computer Science* (1998), Springer-Verlag, pp. 267–288.

[9] Laarhoven, T., Mosca, M., and van de Pol, J. Finding shortest lattice vectors faster using quantum search. *Designs, Codes and Cryptography 77*, 2 (Dec 2015), 375–400.

[10] Micciancio, D. The shortest vector in a lattice is hard to approximate to within some constant. *SIAM Journal on Computing 30*, 6 (2001), 2008–2035.

[11] Micciancio, D., and Voulgaris, P. Faster exponential time algorithms for the shortest vector problem. In *Proceedings of the Twenty-first Annual ACM-SIAM Symposium on Discrete Algorithms* (Philadelphia, PA, USA, 2010), SODA '10, Society for Industrial and Applied Mathematics, pp. 1468–1480.

[12] Micciancio, D., and Voulgaris, P. A deterministic single exponential time algorithm for most lattice problems based on voronoi cell computations. *SIAM Journal on Computing 42*, 3 (2013), 1364–1391.

[13] Micciancio, D., and Walter, M. Fast lattice point enumeration with minimal overhead. Cryptology ePrint Archive, Report 2014/569, 2014. `https://eprint.iacr.org/2014/569`.

[14] Nguyen, P. Cryptanalysis of the goldreich-goldwasser-halevi cryptosystem from crypto '97. In *Advances in Cryptology — CRYPTO' 99* (Berlin, Heidelberg, 1999), M. Wiener, Ed., Springer Berlin Heidelberg, pp. 288–304.

[15] Nguyen, P. Q., and Vidick, T. Sieve algorithms for the shortest vector problem are practical. *J. of Mathematical Cryptology 2*, 2 (2008).

[16] Pujol, X., and Stehle, D. Solving the shortest lattice vector problem in time $2^{2.465n}$. Cryptology ePrint Archive, Report 2009/605, 2009. `https://eprint.iacr.org/2009/605`.

[17] Sommer, N., Feder, M., and Shalvi, O. Finding the closest lattice point by iterative slicing. *SIAM Journal on Discrete Mathematics 23*, 2 (2009), 715–731.