Efficient Lattice Reduction Algorithms and their Applications to Lattice-Based Cryptography

王, 贇弢

KYUSHU UNIVERSITY

Efficient Lattice Reduction Algorithms and their Applications to Lattice-Based Cryptography

by

Yuntao Wang

A thesis submitted in fulfillment for the degree of
Doctor of Philosophy in Functional Mathematics

in the
Graduate School of Mathematics

Advisor: Tsuyoshi Takagi

July 2018

Abstract

In recent decades, the classical public key cryptosystems like RSA and ECC are mainly holding accountable for the security of information and communication. However, Shor's quantum algorithm proposed in 1994 is proved that the hard mathematical problems underlying RSA or ECC can be solved in polynomial time on a quantum computer, so the existent cryptosystems will be broken completely in future. Nowadays, with the increasing threats from recent process of quantum computing, post-quantum cryptography (PQC) has become one of the main focuses in cryptography research and real world applications. Under this situation, NSA showed their intention to move to PQC in the near future in 2015. Then in 2016, NIST started to call for post-quantum cryptosystems officially to nip in the bud. Among all of the PQC candidates, the lattice-based cryptography is one of the most compelling candidates in PQC standardization, for its impressive efficiency derived from the simple and light constructions, and its promising quantum-resistant features.

It is vital to set the proper parameters for a cryptosystem before putting it into service. For the cryptanalysis on lattice-based cryptography, one should estimate the concrete hardness of the underlying hard mathematical problems, such as the shortest vector problem (SVP), the closest vector problem (CVP), the learning with errors problem (LWE) and their variants. In our work, we firstly study the BKZ reduction algorithm, which is one of the most important tools to analyze these hard problems. And then, we propose an improved progressive BKZ (pBKZ) algorithm and compare its efficiency with some previous works. By using our pBKZ algorithm, we solved some instances in the Darmstadt SVP Challenge, the Darmstadt Approximate Ideal Lattice Challenge, and the Darmstadt LWE Challenge. Simultaneously, we also propose an accurate simulator to estimate the computational cost of pBKZ algorithm on solving the (approximate) SVP problem.

Then we apply our pBKZ algorithm and its simulator on solving LWE problem. By invoking Kannan's embedding technique, LWE can be reduced to unique-SVP, which is another hard problem in lattice theory. Based on our experimental results in the Darmstadt LWE Challenge, we decide the critical parameters in Kannan's embedding technique: the number of necessary samples m and the size of the embedding factor M.

Using the proper parameter settings and pBKZ simulator, we show how to estimate the practical hardness of solving the LWE problem.

In the end, we adopt the pBKZ simulator and the M.A. simulator to evaluate the secure parameter sets for Ding key exchange. We denote by RLWE the ring version of LWE problem. Ding key exchange is RLWE-based protocol, which is a proposal to NIST PQC standardization. To guarantee the error rate of key exchange protocol under 2^{-60}, we gave two proper parameter sets covering the security of AES-128/192/256 respectively, which satisfy NIST's security category I, III and V respectively.

Our work makes contribution to NIST's PQC standardization project and provides useful reference for the cryptanalysis in lattice-based cryptography.

Acknowledgements

I would like to express my gratitude to my supervisor Professor Tsuyoshi Takagi for his useful comments, remarks and engagement through the learning process of this PhD thesis. Especially, I am wholeheartedly grateful to his devotion in the guidance to me during my doctoral course. Professor Takagi's learning is profound and immense, which inspire me to keep studying and challenging all my life.

I would like to thank the excellent researchers in many joint works. Particularly the co-authors Dr. Yoshinori Aono, Dr. Takuya Hayashi, Professor Jintai Ding, Dr. Xinwei Gao, the useful discussing and collaborations gave me great help for the works in this doctoral thesis.

I would like to thank Professor Yoshihiro Mizoguchi, Professor Masaya Yasuda and Dr. Koichiro Akiyama, who have willingly shared their precious time during the process of interviewing.

Furthermore, thanks to the support of "Kyushu University Leading PhD Program in Mathematics for Key Technologies" on the way. I got many precious chances going abroad to communicate with foreign researchers and widen my horizon. Especially the coordinators of Professor Osamu Saeki and Professor Fumio Hiroshima are always encouraging me to overcome difficulties in my research.

Finally, I would like to thank my loved ones, my parents and my wife, who have supported me throughout entire process, both by keeping me harmonious and helping me putting pieces together. I will be grateful forever for your love.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Post-Quantum Cryptography

Post-quantum cryptography (PQC), also known as “quantum-resistant” or “quantum-safe” cryptography, is considered to be the next generational cryptographic algorithms which can withstand the outstanding quantum attacks.

Nowadays, the most widely applied public cryptographic schemes, such as RSA, ECC, DSA, ElGamal, Diffie-Hellman key exchange, etc., are constructed based on integer factorization problem (IFP), discrete logarithm problem (DLP) and their elliptic curve variants in certain groups, etc. With elaborately chosen parameters and implementations, the above cryptographic schemes are still secure against current computing resources. However, in 1994, Peter Shor proposed a quantum algorithm which can solve IFP and DLP on a quantum computer in polynomial time [78]. Therefore, once the sufficiently powerful quantum computer is developed, all of the above existences are vulnerable to such quantum computing with Shor’s algorithm. Recently, a prototype quantum computer with 50 quantum bits was built [41]. Since the above cryptosystems are widely deployed in real world applications (e.g. HTTPS, online banking, cryptocurrency, software and updates etc.), it is vital to develop secure and practical post-quantum alternatives for the upcoming post-quantum world.

Currently there are several approaches to build post-quantum cryptography primitives: lattice-based cryptography, multivariate-based cryptography, code-based cryptography, hash-based cryptography, etc. All these cryptosystems have advantages and limitations, therefore it is important to find the best trade-off between security and efficiency. Among them, lattice-based cryptography is considered to have robust security and desirable efficiency. Its versatility in practice is also remarkable, e.g. encryption, signature, key exchange, hash functions, homomorphic encryption etc. In 2015, NSA announced that it is planning the transition to quantum-resistant cryptography suites in near future. NSA wrote: "Unfortunately, the growth of elliptic curve use has bumped up against the fact of continued progress in the research on quantum computing, necessitating a re-evaluation of our cryptographic strategy." In early 2016, NIST formally published the calls for new post-quantum cryptographic algorithms [67]. This reinforces the importance and urgency to develop post-quantum alternatives for the near future. NIST focused on three primitives: public key encryption, digital signature and key establishment. In the round 1 submission, there are 69 schemes based on hard problems from lattices, multivariate polynomial, coding theory, hash functions etc. Lattice-based encryption and key establishment constructions are the majority of these submissions.

In order to apply the lattice-based cryptography in a practical way, we must precisely estimate the secure parameters in theory and practice by analyzing the previously known efficient algorithms for solving the underlying hard mathematical problems.

1.2 Lattice-based Cryptography

Lattices are discrete subgroups of $\mathbb{R}^m$. A lattice L is the set of all integer combinations of linearly independent vectors $\mathbf{b}_1, \ldots, \mathbf{b}_n$ in $\mathbb{R}^m$, which can be expressed as $L(\mathbf{b}_1, \ldots, \mathbf{b}_n) = \{\sum_{i=1}^{n} x_i \mathbf{b}_i, x_i \in \mathbb{Z}\}$. These vectors are known as a *basis* of the lattice and the integer n is its *dimension*. One of the most important problem in lattice is the *shortest vector problem* (SVP), which asks to find a nonzero lattice vector of the smallest norm, by a given lattice basis as input. Another famous problem is called the *closest vector problem* (CVP). For a given arbitrary vector in addition to the lattice, CVP asks to find the closest lattice point to that vector. Since Regev introduced the learning with errors (LWE) problem in 2005, LWE and its variants are also widely studied to build key exchange and encryption schemes [69]. The LWE problem comes from "learning parity with noise" by lifting the modulus value, and concreting the probability distribution of the “error” term [69]. The search LWE says when given a perturbation-merged LWE instance, one should find the solution. Simultaneously, in the decision LWE problem, one needs to distinguish if the given instance is holding LWE construction or just sampled from a uniform distribution.

Lattices in cryptography have been actively used as the foundation for constructing efficient or high-functional cryptosystems such as public-key encryptions [34, 45, 69], fully homomorphic encryptions [39, 20], multi-linear pairing [38], and so on [55, 81, 31]. The security of lattice-based cryptography is based on the hardness of solving the above problems in the underlying lattice theory [26, 55, 62, 63]. Simultaneously, the precise theoretical and the practical analysis of the relevant security parameter settings are required before these cryptosystems are adequate to the reality. Hence we can do cryptanalysis for the lattice-based cryptographic schemes such as NTRU[26], LWE[55] and GGH[63] ect., by using the algorithms for solving these problems [53, 75, 2, 24, 10]. In order to evaluate the hardness of lattice problems due to different lattice structures, Germany TU Darmstadt group initiated several challenges for SVP, approximate SVP, LWE and so on [28, 29, 30, 27]. The platform is public for researchers and some of the challenges will be comprised in this thesis as well.

1.3 Motivation and Contribution

It is known that cryptanalysis is one of the most essential parts before a cryptographic scheme is applied to the reality. Generally the hard problems underlying the lattice-based cryptographic schemes can be reduced to SVP or its variants by some certain coefficients. Therefore, to determine the proper parameters for the cryptographic schemes, it is a critical topic to study the hardness of SVP or its variants. Typically, there are two main practical ways being used to solve these problems. One category consists of approximation algorithms such as some versions of the LLL algorithm [53] and the BKZ algorithm [75]. Such algorithms find relatively short vectors, but usually not the shortest one in high dimension. Another category includes some exact searching algorithms to output the shortest or nearly shortest vectors, e.g. the Kannan strategy [48], the enumeration algorithm [75], the sieving algorithm [2], etc. Usually these algorithms cost exponential runtime, and especially exponential space are required for sieving series. In some cases, the searching algorithms are used as subroutines of reduction algorithms.

In this thesis, at first we investigate a variant of the BKZ algorithm, called progressive BKZ (pBKZ), which performs BKZ reductions by starting with a small blocksize and gradually switching to larger blocks as the process continues. We discuss techniques to accelerate the speed of the progressive BKZ algorithm by optimizing the following parameters: blocksize, searching radius, the success probability for pruning of the local enumeration algorithm, and the constant in the geometric series assumption (GSA). We then propose a simulator for predicting the length of the Gram-Schmidt basis obtained from the BKZ reduction. We also present a model for estimating the computational cost of the proposed progressive BKZ by considering the efficient implementation of the local enumeration algorithm and the LLL algorithm. Finally, we compare the cost of the proposed progressive BKZ with that of other algorithms using instances from the Darmstadt SVP Challenge. The proposed algorithm is approximately 50 times faster than BKZ 2.0 (proposed by Chen-Nguyen [24]) for solving the SVP Challenge up to 160 dimensions.

Then, using our proposed progressive BKZ algorithm and its simulator, we evaluate the practical hardness of solving the LWE problem. At first, invoking Kannan's embedding technique, we apply the pBKZ algorithm on the Darmstadt LWE Challenge instances of $\sigma/q = 0.005$. We observe that intuitively the embedding technique is more efficient with the embedding factor M closer to 1. Then we analyze the optimal number of samples m for a successful attack on LWE case with secret length of n. Thirdly, based on this analysis, we show the practical cost estimations using the precise pBKZ simulator. Simultaneously, our experimental results show that for $n \geq 55$ and the fixed $\sigma/q = 0.005$, the embedding technique with pBKZ is more efficient than Xu et al.'s implementation of the adapted enumeration algorithm in [84][27]. Moreover, by our parameter settings, we succeed in solving the LWE Challenge over $(n, \sigma/q) = (70, 0.005)$ using $2^{16.8}$ seconds (32.73 single core hours).

Finally, we introduce a proper approach to estimate the hardness of the Ring LWE problem (RLWE). More precisely, we apply the cryptanalysis to the "Ding Key Exchange" scheme (DKE), which is a new provably secure ephemeral-only RLWE+Rounding-based key exchange protocol [31]. Note that the main author of this thesis is last named in the DKE proposal to NIST PQC standardization. Since DKE is an ephemeral-only key exchange, it generates only one RLWE sample from protocol execution. In this part we call it the ONE-sample RLWE problem. Our approach is different from existing approaches that are based on estimation with multiple RLWE samples. In our security analysis, at first we reduce the RLWE problem to the short integer solution problem (SIS). We call it "SIS attack". Then, we adopt the progressive BKZ simulator as a practical reference and use the sieving-BKZ (sieving algorithm being used as SVP oracle of BKZ) estimator as a lower bound. The latter one is also called "M.A. simulator" in this thesis. We also analyze the exponential memory impact for sieving algorithm of large dimensions in our parameter settings. Though our analysis is based on some recently developed techniques in Darmstadt, our type of practical security estimate was never done before and it produces security estimates substantially different from the estimates before based on multiple RLWE samples. We present two parameter choices ensuring 2^{-60} key exchange failure probability, which cover security of AES-128/192/256 with concrete security analysis and implementation.

1.4 Organization

Chapter 2 recalls the notations and background on lattices, including some lattice problems and reduction algorithms. We introduce our proposed progressive BKZ algorithm (pBKZ) and its simulator in Chapter 3. In Chapter 4, using this pBKZ algorithm and its simulator, we estimate the practical hardness of the LWE problem. The security analysis for the Ding Key Exchange is shown in Chapter 5. Finally we give conclusion in Chapter 6.

Chapter 2

Background

In this chapter, we introduce some basic knowledge in lattice theory which will be used in this thesis, including some definitions, notations, lattice problems and Darmstadt lattice challenges.

2.1 Mathematical Background of Lattices

Definition 2.1. A *lattice* L is generated by a *basis* B which is a set of linearly independent vectors $\mathbf{b}_1, \ldots, \mathbf{b}_n$ in $\mathbb{R}^m$: $L(\mathbf{b}_1, \ldots, \mathbf{b}_n) = \{\sum_{i=1}^{n} x_i \mathbf{b}_i, x_i \in \mathbb{Z}\}$. Note that in this thesis we use integral lattices for convenience and we write the basis in a matrix form as $B = (\mathbf{b}_1, \ldots, \mathbf{b}_n) \in \mathbb{Z}^{m \times n}$. n is the *rank* of the lattice,which equals to the *dimension* of the vector space over a field spanned by L, i.e. $n = \dim(\mathrm{span}(L))$. It is called *full-rank* lattice when $m = n$.

Definition 2.2. The *Euclidean norm* of a lattice vector $\mathbf{v} \in \mathbb{R}^m$, also known as l_2-norm, is $\|\mathbf{v}\| := \sqrt{\mathbf{v} \cdot \mathbf{v}}$, where the dot product of any two vectors $\mathbf{v} = (v_1, \ldots, v_m)$ and $\mathbf{w} = (w_1, \ldots, w_m)$ is defined as $\mathbf{v} \cdot \mathbf{w} = \sum_{i=1}^{m} v_i w_i$. In some cases, we also denote $\|\mathbf{v}\|$ by $\|\mathbf{v}\|_2$ as well. Besides, let $\|\cdot\|_1$ be the l_1-norm, $\|\cdot\|_\infty$ be the l_∞-norm.

Definition 2.3. The *fundamental domain* for a lattice $L(B)$ corresponding to the basis B is the set $\mathcal{F}(\mathbf{b}_1, \ldots, \mathbf{b}_n) = \{t_1\mathbf{b}_1 + t_2\mathbf{b}_2 + \cdots + t_n\mathbf{b}_n : 0 \le t_i < 1\}$. The *volume* of an n-dimensional lattice $L(B)$ is the volume of $\mathcal{F}$, which can be denoted by $\mathrm{vol}(L)$.

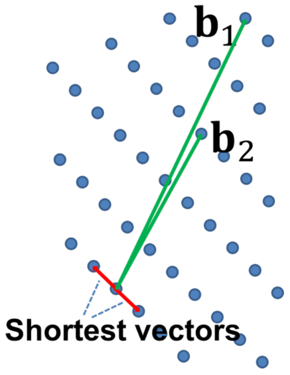

FIGURE 2.1: Modified Gaussian heuristic constant τ_i

Particularly for a full-rank lattice $L(B)$, the volume can be computed by the absolute value of the *determinant* of the basis B, i.e. $\mathrm{vol}(L) = |\det(B)|$.

Any lattice of dimension $n \geq 2$ has infinitely many bases and all the bases of L have the same number n of elements. By multiplying any unimodular matrix in the general linear group $GL_n(\mathbb{Z})$, an n-dimensional lattice basis can be transposed to another basis of the same lattice. However, despite of the varied bases, the same volume is determined by the same lattice space.

Moreover, we denote by $\mathrm{Ball}_n(R)$ the n-dimensional Euclidean ball of radius $R > 0$, and its volume is given by $V_n(R) = R^n \cdot \frac{\pi^{n/2}}{\Gamma(n/2+1)}$. Here the *gamma function* $\Gamma(s)$ is defined by $\Gamma(s) = \int_0^\infty t^{s-1} \cdot e^{-t} dt$ with $s > 0$. For large positive integer n, Stirling's approximation yields $\Gamma(n/2+1) \approx (n/2e)^{n/2}$ and $V_n(1)^{-1/n} \approx \sqrt{n/(2\pi e)} \approx \sqrt{n/17}$. The *beta function* defined by $\mathrm{B}(x,y) = \int_0^1 t^{x-1}(1-t)^{y-1} dt$ is also used in this thesis. Figure 2.1 is an example of 2-dimensional lattice. $\mathbf{b}_1$ and $\mathbf{b}_2$ are two basis vectors which span the lattice space and the "shortest vectors" is introduced as follows.

Definition 2.4 (Shortest Vectors and Successive Minimum)**.** There are at least two non-zero vectors with same minimal Euclidean norm but contrary sign in a lattice L with basis $B = (\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n)$: this norm is called the *first minimum* $\lambda_1(L)$ of $L(B)$. A *shortest vector* of L is of norm $\lambda_1(L)$. We also define the ith successive minimum as

$$\lambda_i(L) = \max\{\min\{\|\mathbf{b}_j\|,\ j = 1, \ldots, i \mid \mathbf{b}_j \text{ are independent}\}\}.$$

The norm of an *approximate shortest vector* $\mathbf{v}$ is slightly relaxed by $\|\mathbf{v}\| \leq \gamma(n) \cdot \lambda_1(L)$ for a function $\gamma(n) > 1$ of a lattice dimension n.

Definition 2.5 (Gram-Schmidt Orthogonalization)**.** We denote by $B^* = (\mathbf{b}_1^*, \ldots, \mathbf{b}_n^*)$ the associated *Gram-Schmidt orthogonal basis* of a given lattice basis $B = (\mathbf{b}_1, \ldots, \mathbf{b}_n)$. Here $\mathbf{b}_1^* = \mathbf{b}_1$ and $\mathbf{b}_i^* = \mathbf{b}_i - \sum_{j=1}^{i-1} \mu_{ij} \mathbf{b}_j^*$ for $1 \leq j < i \leq n$, where $\mu_{ij} = (\mathbf{b}_i \cdot \mathbf{b}_j^*)/\|\mathbf{b}_j^*\|^2$ are the Gram-Schmidt coefficients (abbreviated as *GS-coefficients*). We sometimes refer to $\|\mathbf{b}_i^*\|$ as the *Gram-Schmidt lengths* (abbreviated as *GS-lengths*). We also use the Gram-Schmidt variables (abbreviated as *GS-variables*) to denote the set of both GS-coefficients μ_{ij} and GS-lengths $\|\mathbf{b}_i^*\|$. The *lattice determinant* is defined as $\det(L) := \prod_{i=1}^{n} \|\mathbf{b}_i^*\|$ and it is equal to the volume $\mathrm{vol}(L)$ of the fundamental domain.

$B \in \mathbb{R}^{n \times m}$ can be decomposed uniquely as $B = \mu \cdot D \cdot Q$. Here $\mu = (\mu_{ij})$ is the $n \times n$ lower-triangular matrix with unit diagonal, D is the diagonal matrix formed by the GS-lengths $\|\mathbf{b}_i^*\|$'s, and Q is an $n \times m$ matrix with orthonormal row vectors.

Definition 2.6 (Projective Sublattice)**.** We denote the *orthogonal projection* by $\pi_i : \mathbb{R}^m \mapsto span(\mathbf{b}_1, \ldots, \mathbf{b}_{i-1})^\perp$ for $1 < i \leq n$. In particular, π_1 is used as the identity map. For natural numbers i and j with $i < j$, $[i : j]$ is the set of integers $\{i, i+1, \ldots, j\}$. Particularly, $[1 : j]$ is denoted by $[j]$. We denote the *local block* by the *projective sublattice*

$$L_{[i:j]} := L(\pi_i(\mathbf{b}_i), \pi_i(\mathbf{b}_{i+1}), \ldots, \pi_i(\mathbf{b}_j))$$

for $j \in \{i, i+1, \ldots, n\}$, whose volume is $\mathrm{vol}(L_{[i:j]} = \prod_{j=i}^{n} \|\mathbf{b}_j^*\|$. We sometimes use B_i instead of $B_{[i:j]} = (\pi_i(\mathbf{b}_i), \ldots, \pi_i(\mathbf{b}_j))$, where $j = i + \beta - 1$, to denote the basis of a β-sized sublattice $L_{[i:j]}$. That is, we omit the notation of blocksize $\beta = j - i + 1$ or j, if it is clear by context. We also denote $L_{[i:j]}$ by $\pi_i(L)$ in some cases. The notion of the first minimum is also defined for a projective sublattice as $\lambda_1(L_{[i:j]})$ (we occasionally refer to this as $\lambda_1(B_i)$ in this chapter).

Definition 2.7 (q-ary Lattice)**.** For a positive integer q, a lattice $L \subset \mathbb{Z}^m$ is called a q-ary lattice if $q\mathbb{Z}^m \subset L$. Let $\mathbf{A} \in \mathbb{Z}_q^{m \times n}$ $(m > n)$ be a matrix with column vectors. We define the following two m-dimensional q-ary lattices.

$$L_{(\mathbf{A},q)} = \Lambda_q(\mathbf{A}) = \{\mathbf{y} \in \mathbb{Z}_q^m | \mathbf{y} \equiv \mathbf{A}\mathbf{x} \pmod{q} \text{ for some } \mathbf{x} \in \mathbb{Z}^n\}$$

and

$$L^{\perp}_{(\mathbf{A},q)} = \Lambda_q^{\perp}(\mathbf{A}^T) = \{\mathbf{y} \in \mathbb{Z}_q^m | \mathbf{A}^T\mathbf{y} \equiv \mathbf{0} \pmod q\}.$$

The first lattice is analogous to the linear code generated by the columns of $\mathbf{A}$ mod q; the second lattice corresponds to the linear code with parity check matrix equal to $\mathbf{A}^T$ mod q. The two q-ary lattices hold $\mathrm{vol}(\Lambda_q(\mathbf{A})) \geq q^{m-n}$ and $\mathrm{vol}(\Lambda_q^{\perp}(\mathbf{A}^T)) \leq q^n$ respectively, while the condition is equivalent when the columns of $\mathbf{A}$ are linearly independent over $\mathbb{Z}_q$. We can construct the basis B of $L_{(\mathbf{A},q)}$, via eliminating the linearly dependent vectors in the following matrix by an elementary transformation.

$$\left(\frac{\mathbf{A}^T}{q\mathbf{I}_m}\right) \in \mathbb{Z}^{(m+n)\times m}.$$

In this thesis, we reduce this basis to a square matrix with Hermite Normal Form.

Definition 2.8 (Hermite Normal Form)**.** The Hermite Normal Form (HNF) of a basis B satisfies: (1) B is lower triangular; (2) the diagonal entries are positive; (3) any entry below the diagonal is a non-negative number strictly less than the diagonal entry in same column.

In this work, we use the HNF module in Victor Shoup's NTL library [79], which uses the Domich et al.'s algorithm [32]. Particularly, a basis of q-ary lattice $L_{(\mathbf{A},q)}$ has this form with some matrix $A'_{n\times(m-n)} \in \mathbb{Z}_q^{n\times(m-n)}$:

$$B_{\mathrm{HNF}} = \begin{pmatrix} q\mathbf{I}_{m-n} & \mathbf{0} \\ A'_{n\times(m-n)} & \mathbf{I}_n \end{pmatrix} \in \mathbb{Z}^{m\times m}.$$

Definition 2.9 (Gaussian Heuristic)**.** Given an n-dimensional lattice L and a continuous (usually convex and symmetric) set $S \subset \mathbb{R}^m$, the *Gaussian heuristic* says that the number of points in $S \cap L$ is approximately $\mathrm{vol}(S)/\mathrm{vol}(L)$.

In particular, assume S is the origin-centered ball of radius R. The number of lattice points is approximately $V_n(R)/\mathrm{vol}(L)$, which derives the approximate length of shortest vector λ_1 by R so that the volume of the ball is equal to that of the lattice:

$$\lambda_1(L) \approx \det(L)^{1/n}/V_n(1)^{1/n} = \frac{(\Gamma(n/2+1)\det(L))^{1/n}}{\sqrt{\pi}}$$

This is usually called the *Gaussian heuristic of a lattice*, and we denote it by $\mathrm{GH}(L) = \det(L)^{1/n}/V_n(1)^{1/n}$.

2.2 Lattice Problems

Definition 2.10 (Shortest Vector Problem)**.** Given a basis $B = (\mathbf{b}_1, \ldots, \mathbf{b}_n)$ of a lattice L, the *Shortest Vector Problem* (SVP) asks to find a non-zero shortest vector in L. Similarly, we define the approximate version of SVP as follows.

Definition 2.11 (Approximate Shortest Vector Problem)**.** The *approximate Shortest Vector Problem* (approx-SVP) gives a relaxed bound in comparison with SVP: Input a basis B of an n-dimensional lattice L and set a parameter $\gamma(n) > 1$, to find a non-zero short vector $\mathbf{v}$ in L, s.t. $\|\mathbf{v}\| \leq \gamma(n)\lambda_1(L)$.

Definition 2.12 (Unique Shortest Vector Problem)**.** The *Unique SVP Problem* (uSVP) is for a given lattice L which satisfies $\lambda_1(L) \ll \lambda_2(L)$, to find the shortest nonzero vector in L. It is called γ *Unique SVP problem* (γ-uSVP) if the gap of $\lambda_2(L)/\lambda_1(L) = \gamma$ is known.

Definition 2.13 (Short Integer Solution Problem)**.** Given an integer q and a matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$, *Short Integer Solution Problem* (SIS) is to compute a short vector $\mathbf{y} \in \mathcal{B}$ s.t. $\mathbf{A}\mathbf{y} \equiv \mathbf{0} \mod q$, where $\mathcal{B}$ is a set of short vectors with some Euclidean norm bound.

Definition 2.14 (Closest Vector Problem)**.** Given an input basis $B = (\mathbf{b}_1, \ldots, \mathbf{b}_n)$ of a lattice L and a vector $\mathbf{v} \in \mathbb{R}^m$ which is not in L, the *Closest Vector Problem* (CVP) is to find a vector $\mathbf{b} \in L$ that minimizes the Euclidean norm $\|\mathbf{v} - \mathbf{b}\|$.

The following BDD problem is a variant of CVP.

Definition 2.15 (Bounded Distance Decoding)**.** In a Euclidean space spanned by a n-dimensional lattice L, there is a target vector $\mathbf{w} \in \mathbb{R}^m$ which is guaranteed to be within a distance $r \leq \alpha\lambda_1(L)$ to some lattice point, where $\alpha > 0$. The *Bounded Distance Decoding* (BDD) outputs a vector $\mathbf{b} \in L$ s.t. $\|\mathbf{w} - \mathbf{b}\| \leq r$.

Let Λ be a discrete subset of $\mathbb{Z}^n$. For any vector $\mathbf{c} \in \mathbb{R}^n$ and any positive parameter $\sigma > 0$, let $\rho_{\sigma,\mathbf{c}}(\mathbf{x}) = e^{-\pi\|\mathbf{x}-\mathbf{c}\|^2/\sigma^2}$ be the Gaussian function on $\mathbb{R}^n$ with the center $\mathbf{c}$ and the parameter σ. Denote $\rho_{\sigma,\mathbf{c}}(\Lambda) = \sum_{x\in\Lambda} \rho_{\sigma,\mathbf{c}(x)}$ be the discrete integral of $\rho_{\sigma,\mathbf{c}}$ over Λ, and $D_{\Lambda,\sigma,\mathbf{c}}$ be the discrete Gaussian distribution over Λ with the center $\mathbf{c}$ and the parameter σ. For all $\mathbf{y} \in \Lambda$, we have $D_{\Lambda,\sigma,\mathbf{c}}(\mathbf{y}) = \frac{\rho_{\sigma,\mathbf{c}}(\mathbf{y})}{\rho_{\sigma,\mathbf{c}}(\Lambda)}$. In this chapter, we fix Λ to be $\mathbb{Z}^n$ and $\mathbf{c}$ to be zero vector. For ease of notation, we denote $D_{\mathbb{Z}^n,\sigma,0}$ as $D_{\mathbb{Z}^n,\sigma}$.

Let $\overset{\$}{\leftarrow} \chi$ denote a random sampling according to the distribution χ. Here we represent $\mathbb{Z}_q$ as $\{0, \cdots, q-1\}$ for convenience. However, on occasion, we treat elements in $\mathbb{Z}_q$ as elements in $\{-\frac{q-1}{2}, \cdots, \frac{q-1}{2}\}$, but we will remark the switch clearly.

Now we introduce the learning with errors problem (LWE problem) and its ring version (RLWE problem) as follows.

Definition 2.16 (Learning With Errors Problem [69])**.** There are four parameters in LWE problem: the number of samples $m \in \mathbb{Z}$, the length $n \in \mathbb{Z}$ of secret vector, modulo $q \in \mathbb{Z}$ and the standard deviation $\sigma \in \mathbb{R}_{>0}$ for the discrete Gaussian distribution $D_{\mathbb{Z}^n,\sigma}$. Sample a matrix $\mathbf{A} \in \mathbb{Z}_q^{m\times n}$ and a secret vector $\mathbf{s} \in \mathbb{Z}_q^n$ uniformly at random, and randomly sample a relatively small perturbation vector $\mathbf{e} \in \mathbb{Z}_q^m$ from Gaussian distribution $D_{\mathbb{Z}^n,\sigma}$, i.e. $\mathbf{e} \overset{\$}{\leftarrow} D_{\mathbb{Z}^n,\sigma}$. The LWE distribution Ψ is constructed by pairs $(\mathbf{A}, \mathbf{b} \equiv \mathbf{A}\mathbf{s} + \mathbf{e} \pmod{q}) \in (\mathbb{Z}_q^{m\times n}, \mathbb{Z}_q^m)$ sampled as above. The search *learning with errors problem* (LWE problem) is for a given pair $(\mathbf{A}, \mathbf{b})$ sampled from LWE distribution Ψ, to compute the pair $(\mathbf{s}, \mathbf{e})$. The decision version of LWE problem asks to distinguish that if the given pair $(\mathbf{A}, \mathbf{b})$ is sampled from LWE or from uniform distribution. The proof of equivalent hardness between these two versions is given in the original LWE paper [69].

Definition 2.17 (Ring Learning With Errors Problem [59])**.** Let $m \geq 1$ be a power of 2 and $q \geq 2$ be an integer. Let $R_q = \mathbb{Z}_q[x]/\Phi_m(x)$, where $\Phi_m(x) = x^n + 1$ is the m-th cyclotomic polynomial with $n = m/2$. Let χ be a β-bounded distribution. For secret polynomial $\mathbf{s} \overset{\$}{\leftarrow} \chi$ and error polynomial $\mathbf{e} \overset{\$}{\leftarrow} \chi$, choosing $\mathbf{a} \in R_q$ uniformly at random, output $(\mathbf{a}, \mathbf{b} = \mathbf{a}\cdot\mathbf{s} + \mathbf{e}) \in (R_q, R_q)$. The search version of *ring learning with errors problem* (RLWE problem) is: for $\mathbf{s} \overset{\$}{\leftarrow} \chi$, given $poly(n)$ number of samples of $(\mathbf{a}, \mathbf{b} = \mathbf{a}\cdot\mathbf{s} + \mathbf{e}) \in (R_q, R_q)$, find $\mathbf{s}$ (and $\mathbf{e}$ simultaneously). Decision version of RLWE problem is: for $\mathbf{a} \in R_q$, distinguish $\mathbf{b} = \mathbf{a}\cdot\mathbf{s} + \mathbf{e} \in R_q$ from uniformly sampled polynomials in R_q.

There is no straight-forward reduction between these two RLWE problems for general rings. However, normally people reduce the RLWE to the corresponding normal LWE problem, to evaluate the hardness of RLWE problems.

Definition 2.18 ((Root) Hermite Factor)**.** To evaluate the performance of lattice algorithms for solving SVP and its variants, we use the *Hermite Factor* defined in [36] as:

$$\mathrm{HF}(\mathbf{b}_1, \ldots, \mathbf{b}_n) = \|\mathbf{b}_1\| / \mathrm{Vol}(L)^{1/n}.$$

Also we usually use *root Hermite Factor* (rHF) in analysis, which is denoted as:

$$\delta = \mathrm{rHF}(\mathbf{b}_1, \ldots, \mathbf{b}_n) = (\|\mathbf{b}_1\| / \mathrm{vol}(L)^{1/n})^{1/n}. \tag{2.1}$$

Intuitively in a n-dimensional lattice, if the (root) Hermite factor is smaller, it means that the algorithm performs better due to the shorter vector. The Hermite factor depends on n, while the experimental results in [36] show that the root Hermite factor converges to a constant which is verified in [61] for large$n > 140$. Thus, informally we can say the root Hermite factor depends on the capability of reduction algorithms and their parameter settings.

Definition 2.19 (Geometric Series Assumption)**.** The *Geometric Series Assumption* (GSA) defined in [73] indicates the quality of an LLL-type reduced basis. It says that the l_2 norms of Gram-Schmidt vectors $\|\mathbf{b}_i^*\|$ in the reduced basis decrease geometrically with a constant r such as $\|\mathbf{b}_i^*\|^2 / \|\mathbf{b}_1\|^2 = r^{i-1}$, for $i = 1, \ldots, n$ and $r \in [3/4, 1)$. The size of r depends on the reduction algorithm and the corresponding parameter settings.

Proposition 2.20. *For a LLL-type reduced basis of large dimension n, the relation of the GSA constant r and the rHF δ is of $r \approx \delta^{-4}$.*

Proof. Assume the reduced basis of the given lattice L is $B = (\mathbf{b}_1, \ldots, \mathbf{b}_n)$, and its corresponding Gram-Schmidt basis is $B^* = (\mathbf{b}_1^*, \ldots, \mathbf{b}_n^*)$. Due to the definition of GSA: $\|\mathbf{b}_i^*\|^2 / \|\mathbf{b}_1\|^2 = r^{i-1}$, and the volume of L: $\mathrm{vol}(L) = \prod_{i=1}^{n} \|\mathbf{b}_i^*\|$, we get $\mathrm{vol}(L) = r^{n(n-1)/4} \cdot \|\mathbf{b}_1\|^n$. Substitute $\mathrm{vol}(L)$ to the root Hermite Factor (2.1), we get $r = \delta^{-4n/(n-1)}$ and we can get the approximate result for $n \to \infty$. $\square$

2.3 Darmstadt Lattice Challenge Series.

To estimate the concrete practical security of lattice bases cryptography, several types of challenges on lattice problems are published in Technical University of Darmstadt [28, 30, 29, 27]. We summarize the challenges in Table 2.1. Additionally, we give more details about the LWE Challenge, since it plays the main role in the experiments in Chapter 4.

TABLE 2.1: Type of Challenges

	Type of instances	Target	Parameters
Lattice Challenge [30]	$q - ary$ Lattices	$\|\mathbf{b}\| < q$	q, n
SVP Challenge [28]	Hermite Normal Form of Random Lattices	$\|\mathbf{b}\| < 1.05\mathrm{GH}$	n
Ideal Lattice Challenge [29]	Hermite Normal Form of Random Ideal Lattices	$\|\mathbf{b}\| < 1.05\mathrm{GH}$	n
Approx-Ideal Lattice Challenge [29]	Hermite Normal Form of Random Ideal Lattices	$\|\mathbf{b}\| < n \cdot \det^{1/n}$	n
LWE Challenge [27]	LWE Instances of $(\mathbf{A}, \mathbf{b} \equiv \mathbf{A}\mathbf{s} + \mathbf{e} \,(\mathrm{mod}\, q)) \in (\mathbb{Z}_q^{m \times n}, \mathbb{Z}_q^m)$	s	m, n, q, σ

Darmstadt LWE Challenge In 2016, TU Darmstadt, in alliance with UC San Diego and TU Eindhoven, published a platform for the concrete parameter analysis on the LWE problem [21][27]. In the LWE Challenge, the organizers merge the two parameters σ and q into the *relative error size* α such that $\alpha = \sigma/q$. n is the length of secret vector and q is the minimum prime number larger than n^2. For each case of length n, they offer the sampled n column vectors in basis $\mathbf{A}' \in \mathbb{Z}_q^{n^2 \times n}$, and one column target vector $\mathbf{b}' \in \mathbb{Z}_q^{n^2}$. The length n and the relative error size α are arithmetic sequences from 40 and 0.005, with common differences of 5 and 0.005 respectively. To adapt the current lattice algorithms used in the attack algorithm, one usually randomly sample $m \ll n^2$ entries of the column vectors in the original basis $\mathbf{A}' \in \mathbb{Z}_q^{n^2 \times n}$ and sample from the target vector $\mathbf{b}' \in \mathbb{Z}_q^{n^2}$ respectively, i.e. randomly sample from $(\mathbf{A}', \mathbf{b}') \in (\mathbb{Z}_q^{n^2 \times n}, \mathbb{Z}_q^{n^2})$ in Darmstadt LWE Challenge instance to the one of $(\mathbf{A}, \mathbf{b}) \in (\mathbb{Z}_q^{m \times n}, \mathbb{Z}_q^m)$. This is called *sub-lattice attack*, and we discuss how to choose a suitable m in Section 4.3.4.

Chapter 3

The Improved Progressive BKZ Algorithm and its Cost Simulator

In this chapter, we introduce our improved progressive BKZ and its cost simulator. At first we revisit progressive BKZ algorithms, which have been mentioned in several studies [36, 24, 74, 77, 43]. The main idea of progressive BKZ is starting with a small blocksize, and performing BKZ iteratively by increasing the blocksize β, which is practically faster than the direct execution of BKZ with a larger blocksize. The method used to increase the blocksize β strongly affects the overall computational cost of progressive BKZ. The research goal here is to find an optimal method of increasing the blocksize β according to the other parameters in the BKZ algorithms.

One major difference between BKZ 2.0 [24] and our algorithm is the usage of randomized enumeration in local blocks. To find a very short vector in each local block efficiently, BKZ 2.0 uses the randomizing technique in [37]. Then, it reduces each block to decrease the cost of lattice enumeration. Although it is significantly faster than the enumeration without pruning, it causes overhead because the bases are not good in practice after they have been randomized. To avoid this overhead, we adopted the algorithm with a single enumeration with a low probability.

Moreover, BKZ of a large blocksize with large pruning (i.e., a low probability) is generally better in both speed and quality of basis than that of a small blocksize with few pruning (i.e., a high probability), as a rule of thumb. We pursue this idea and add the freedom to choose the radius $\alpha \cdot \mathrm{GH}(L)$ of the enumeration of the local block; this value is fixed in BKZ 2.0 as $\sqrt{1.1} \cdot \mathrm{GH}(L)$.

To optimize the algorithm, we first discuss techniques for optimizing the BKZ parameters of enumeration subroutine, including the blocksize β, success probability p of enumeration, and α to set the searching radius of enumeration as $\alpha \cdot \mathrm{GH}(B_i)$. We then show the parameters' relation that minimizes the computational cost for enumeration of a BKZ-β-reduced basis. Next, we introduce the new usage of *full enumeration cost* (FEC), derived from Gama-Nguyen-Regev's cost estimation [37] with a Gaussian heuristic radius and without pruning, to define the quality of the basis and to predict the cost after BKZ-β is performed. Using this metric, we can determine the timing for increasing blocksize β that provides an optimized strategy; in previous works, the timing was often heuristic.

Furthermore, we propose a new BKZ simulator to predict the Gram-Schmidt lengths $\|\mathbf{b}_i^*\|$ after BKZ-β. Some previous works aimed to find a short vector as fast as possible, and did not consider other quantities. However, additional information is needed to analyze the security of lattice-based cryptosystems. In literature, a series of works on lattice basis reduction [73, 36, 24, 25] have attempted to predict the Gram-Schmidt lengths $\|\mathbf{b}_i^*\|$ after lattice reduction. In particular, Schnorr's GSA is the first simulator of Gram-Schmidt lengths and the information it provides is used to analyze the random sampling algorithm. We follow this idea, i.e., predicting Gram-Schmidt lengths to analyze other algorithms.

Our simulator is based on the Gaussian heuristic with some modifications, and is computable directly from the lattice dimension and the blocksize. On the other hand, Chen-Nguyen's simulator must compute the values sequentially; it has an inherent problem of accumulative error, if we use the strategy that changes blocksize many times. We also investigate the computational cost of our implementation of the new progressive BKZ, and show our estimation for solving challenge problems in the Darmstadt SVP Challenge [28] and Ideal Lattice Challenge [29]. Our cost estimation is derived by setting the computation model and by curve fitting based on results from computer experiments. Using our improved progressive BKZ, we solved Ideal Lattice Challenge of 600 and 652 dimensions in the exact expected times of $2^{20.7}$ and $2^{24.0}$ seconds, respectively, on a standard PC.

Finally, we compare our algorithm with several previous algorithms. In particular, compared with Chen-Nguyen's BKZ 2.0 algorithm [24, 25] and Schnorr's blocksize doubling strategy [77], our algorithm is significantly faster. For example, to find a vector shorter than $1.05 \cdot \mathrm{GH}(L)$, which is required by the SVP Challenge [28], our algorithm is approximately 50 times faster than BKZ 2.0 in a simulator-based comparison up to 160 dimensions. Furthermore, we also published our progressive BKZ source code in [11]. The work in this chapter was presented in Eurocrypt 2016 [10].

3.1 Introduction

Currently the most efficient algorithms for solving the SVP are perhaps a series of BKZ algorithms [75, 76, 24, 25]. Numerous efforts have been made to estimate the security of lattice-based cryptography by analyzing the BKZ algorithms. Lindner and Peikert [55] gave an estimation of secure key sizes by connecting the computational cost of BKZ algorithm with the root Hermite factor from their experiment using the NTL-BKZ [79]. Furthermore, van de Pol and Smart [81] estimated the key sizes of fully homomorphic encryptions using a simulator based on Chen-Nguyen's BKZ 2.0 [24]. Lepoint and Naehrig [54] gave a more precise estimation using the parameters of the full-version of BKZ 2.0 paper [25]. On the other hand, Liu and Nguyen [56] estimated the secure key sizes of some LWE-based cryptosystems by considering the BDD in the associated q-ary lattice. Aono et al. [9] gave another security estimation for LWE-based cryptosystems by considering the challenge data from the Darmstadt Lattice Challenge [30]. Recently, Albrecht et al. presented a comprehensive survey on the state-of-the-art of hardness estimation for the LWE problem [6].

The above analyzing algorithms are usually called "lattice-based attacks", which have a generic framework consisting of two parts:

(1)Lattice reduction: This step aims to decrease the norm of vectors in the basis by performing a lattice reduction algorithm such as the LLL or BKZ algorithm.

(2)Point search: This step finds a short vector in the lattice with the reduced basis by performing the enumeration algorithm.

In order to obtain concrete and practical security parameters for lattice-based cryptosystems, it is necessary to investigate the trade-offs between the computational cost of a lattice reduction and that of a lattice point search.

For our total cost estimation, we further limit the lattice-based attack model by (1) using our improved progressive BKZ algorithm for lattice reduction, and (2) using the standard (sometimes randomized) lattice vector enumeration algorithm with sound pruning [37]. To predict the computational cost under this model, we propose a simulation method to generate the computing time of lattice reduction and the lengths of the Gram-Schmidt vectors of the basis to be computed.

BKZ Algorithms:

Let $B = (\mathbf{b}_1, \ldots, \mathbf{b}_n)$ be the basis of the lattice. The BKZ algorithms perform the following local point search and update process from index $i = 1$ to $n-1$. The local point search algorithm, which is essentially the same as the algorithm used in the second part of the lattice-based attacks, finds a short vector in the local block $B_i = \pi_i(\mathbf{b}_i, \ldots, \mathbf{b}_{i+\beta-1})$ of the fixed blocksize β (the blocksize shrinks to $n - i + 1$ for large $i \geq n - \beta + 1$). Here, the lengths of vectors are measured under the projection π_i which is defined in Section 2.5. Then, the update process applies lattice reduction for the degenerated basis $(\mathbf{b}_1, \ldots, \mathbf{b}_{i-1}, \mathbf{v}, \mathbf{b}_i, \ldots, \mathbf{b}_n)$ after inserting vector $\mathbf{v}$ at ith index.

The point search subroutine finds a short vector in some searching radius $\alpha \cdot \mathrm{GH}(B_i)$ with some probability which is defined over random local blocks of the fixed dimension. Here, $\mathrm{GH}(B_i)$ is an approximation of the length of the shortest vector in the sublattice generated by B_i.

In the classical BKZ algorithms [76, 75], the local point search calls a single execution of a lattice vector enumeration algorithm with a reasonable pruning for searching tree. The BKZ 2.0 algorithm proposed by Chen and Nguyen [24] uses the extreme pruning technique [37], which performs the lattice enumeration with success probability p for $\lfloor 1/p \rceil$ different bases $G_1, \ldots, G_{\lfloor 1/p \rceil}$ obtained by randomizing the local basis B_i. They use the fixed searching radius as $\sqrt{1.1} \cdot \mathrm{GH}(B_i)$. We stress that BKZ 2.0 is practically the fastest algorithm for solving the approximate SVP of large dimensions. Indeed, many top-records in the Darmstadt Lattice Challenge [30] have been solved by BKZ 2.0.

TABLE 3.1: Technical comparison from BKZ 2.0

Technique	BKZ 2.0 [24]	Our algorithm
Enumeration setting		
randomizing basis [37]	yes	no
optimal pruning [37]	yes	yes
blocksize β	fixed	iteratively increasing (Sec. 3.6.1)
search radius $\alpha \cdot \mathrm{GH}(B_i)$	$\sqrt{1.1} \cdot \mathrm{GH}(B_i)$	} optimized by GSA (Sec. 3.4)
probability p	optimized by simulator	
Preprocessing local block	optimal BKZ strategy	progressive BKZ
Terminating BKZ strategy	simulator based (fixed)	FEC based (adaptive, Sec. 3.5)
Predicting $\|\mathbf{b}_i^*\|$	simulator based	simulator based (Sec. 3.5.1)

Roadmap: In Section 3.2 we introduce the basic facts on lattices. In Section 3.3 we give an overview of BKZ algorithms, including Chen-Nguyen's BKZ 2.0 [24] and its cost estimation; we also state some heuristic assumptions. In Section 3.4, we propose the optimized BKZ parameters under the Schnorr's geometric series assumption (GSA). In Section 3.5, we explain the basic variant of the proposed progressive BKZ algorithm and its simulator for the cost estimation. In Section 3.6, we discuss the optimized block strategy that improved the speed of the proposed progressive BKZ algorithm. In Section 3.7, we describe the details of our implementation and the cost estimation for processing local blocks. We then discuss an extended strategy using many random reduced bases [37] besides our progressive BKZ in Section 3.8. Finally, Section 3.9 gives the results of our simulation to solve the SVP Challenge problems and compares these results with previous works.

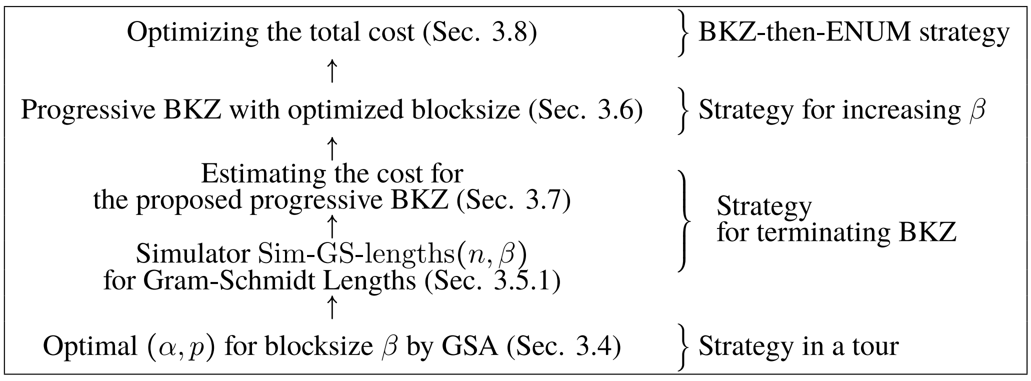
Optimizing the total cost (Sec. 3.8)
BKZ-then-ENUM strategy
Progressive BKZ with optimized blocksize (Sec. 3.6)
Strategy for increasing β
Estimating the cost for the proposed progressive BKZ (Sec. 3.7)
Simulator Sim-GS-lengths(n, β) for Gram-Schmidt Lengths (Sec. 3.5.1)
Strategy for terminating BKZ
Optimal (α, p) for blocksize β by GSA (Sec. 3.4)
Strategy in a tour

FIGURE 3.1: Roadmap of this paper: optimizing parameters from local to global

3.2 Preliminaries

For our analysis, we use the following lemma on the randomly generated points.

Lemma 1. Let $x_1, \ldots, x_K$ be K points uniformly sampled from the n-dimensional unit ball. Then, the expected value of the shortest length of vectors from origin to these points is

$$\mathbf{E}\Big[\min_{i\in[K]} \|x_i\|\Big] = K \cdot B\Big(K, \frac{n+1}{n}\Big) := K \cdot \int_0^1 t^{1/n}(1-t)^{K-1}dt.$$

In particular, letting $K = 1$, the expected value is $n/(n+1)$.

Proof. Since the cumulative distribution function of each $\|x_i\|$ is $F_i(r) = r^n$, the cumulative function of the shortest length of the vectors is $F_{\min}(r) = 1-(1-F_i(r))^K = 1-(1-r^n)^K$. Its probability density function is $P_{\min}(r) = \frac{dF}{dr} = Kn \cdot r^{n-1}(1-r^n)^{K-1}$. Therefore, the expected value of the shortest length of the vectors is

$$\int_0^1 rP_{\min}(r)dr = K \cdot \int_0^1 t^{1/n}(1-t)^{K-1}dt.$$

□

Throughout this thesis, function $\log$ denotes the natural logarithm, $\log_2$ denotes logarithm with base 2.

3.2.1 Enumeration Algorithm [48, 75, 37]

We explain the enumeration algorithm for finding a short vector in the lattice. The pseudo code of the enumeration algorithm is given in [75, 37]. For given lattice basis $(\mathbf{b}_1, \ldots, \mathbf{b}_n)$, and its Gram-Schmidt basis $(\mathbf{b}_1^*, \ldots, \mathbf{b}_n^*)$, the enumeration algorithm considers a search tree whose nodes are labeled by vectors. The root of the search tree is the zero vector; for each node labeled by $\mathbf{v} \in L$ at depth $k \in [n]$, its children have labels $\mathbf{v} + a_{n-k} \cdot \mathbf{b}_{n-k}$ $(a_{n-k} \in \mathbb{Z})$ whose projective length $\|\pi_{n-k}(\sum_{i=n-k}^{n} a_i \cdot \mathbf{b}_i)\|$ is smaller than a bounding value $R_{k+1} \in (0, \|\mathbf{b}_1\|]$. After searching all possible nodes, the enumeration algorithm finds a lattice vector shorter than R_n at a leaf of depth n, or its projective length is somehow short at a node of depth $k < n$. It is clear that by taking $R_k = \|\mathbf{b}_1\|$ for all $k \in [n]$, the enumeration algorithm always finds the shortest vector $\mathbf{v}_1$ in the lattice, namely $\|\mathbf{v}_1\| = \lambda_1(L)$.

Because $\|\mathbf{b}_1\|$ is often larger than $\lambda_1(L)$, we can set a better searching radius $R_n = \mathrm{GH}(L)$ to decrease the computational cost. We call this *the full enumeration algorithm* and define the full enumeration cost $\mathrm{FEC}(B)$ as the cost of the algorithm for this basis. With the same argument in [37], we can evaluate $\mathrm{FEC}(B)$ using the following equation.

$$\mathrm{FEC}(B) = \sum_{k=1}^{n} \frac{V_k(\mathrm{GH}(L))}{\prod_{i=n-k+1}^{n} \|\mathbf{b}_i^*\|}.$$

Because full enumeration is a cost-intensive algorithm, several improvements have been proposed by considering the trade-offs between running time, searching radius, and success probability [76, 37]. Gama-Nguyen-Regev [37] proposed a cost estimation model of the lattice enumeration algorithm to optimize the bounding functions of $R_1, \ldots, R_n$, which were mentioned above. The success probability p of finding a single vector within a radius c is given by

$$p = \Pr_{(x_1,\ldots,x_n) \leftarrow c\cdot S_n} \left[\sum_{i=1}^{\ell} x_i^2 < R_\ell^2 \text{ for } \forall\ \ell \in [n] \right],$$

where S_n is the surface of the n-dimensional unit ball. Then, the cost of the enumeration algorithm can be estimated by the number of processed nodes, i.e.,

$$N = \frac{1}{2} \sum_{k=1}^{n} \frac{\mathrm{vol}\{(x_1,\ldots,x_k) \in \mathbb{R}^k : \sum_{i=1}^{\ell} x_i^2 < R_\ell^2 \text{ for } \forall\ \ell \in [k]\}}{\prod_{i=n-k+1}^{n} \|\mathbf{b}_i^*\|}. \tag{3.1}$$

Note that the factor $1/2$ is based on the symmetry. Using the methodology in [37], Chen-Nguyen proposed a method to find the optimal bounding functions of $R_1, \ldots, R_n$ that minimizes N subject to p.

In this paper, we use the *lattice enumeration cost*, abbreviated as *ENUM cost*, to denote the number N in equation (3.1). For a lattice L defined by a basis B and parameters $\alpha > 0$ and $p \in [0,1]$, we use $\mathrm{ENUMCost}(B; \alpha, p)$ to denote the minimized cost N of lattice enumeration with radius $c = \alpha \cdot \mathrm{GH}(L)$ subject to the success probability p. This notion is also defined for a projective sublattice.

3.3 Lattice Reduction Algorithms

Lattice reduction algorithms transform a given lattice basis $(\mathbf{b}_1, \ldots, \mathbf{b}_n)$ to another basis whose Gram-Schmidt lengths are relatively shorter.

LLL algorithm [53]: The LLL algorithm transforms the basis $(\mathbf{b}_1, \ldots, \mathbf{b}_n)$ using the following two operations: size reduction $\mathbf{b}_i \leftarrow \mathbf{b}_i - \lfloor \mu_{ij} \rceil \mathbf{b}_j$ for $j \in [i-1]$, and neighborhood swaps between $\mathbf{b}_i$ and $\mathbf{b}_{i+1}$ if $\|\mathbf{b}_{i+1}^*\|^2 \leq 1/2 \|\mathbf{b}_i^*\|^2$ until no update occurs.

BKZ algorithms [75, 76]. For a given lattice basis and a fixed blocksize β, the BKZ algorithm processes the following operation in the local block B_i, i.e., the projected sublattice $L_{[i,i+\beta-1]}$ of blocksize β, starting from the first index $i = 1$ to $i = n-1$. Note that the blocksize β reluctantly shrinks to $n - i + 1$ for large $i > n - \beta + 1$, and thus we sometimes use β' to denote the dimension of B_i, i.e. $\beta' = \min(\beta, n - i + 1)$.

At index i, the standard implementation of the BKZ algorithm calls the enumeration algorithm for the local block B_i. Let $\mathbf{v}$ be a shorter vector found by the enumeration algorithm. Then the BKZ algorithm inserts $\mathbf{v}$ into $\mathbf{b}_{i-1}$ and $\mathbf{b}_i$, and constructs the degenerated basis $(\mathbf{b}_1, \ldots, \mathbf{b}_{i-1}, \mathbf{v}, \mathbf{b}_i, \ldots, \mathbf{b}_{\min(i+\beta-1,n)})$. For this basis, we apply the LLL algorithm (or BKZ with a smaller blocksize) so that the basis of shorter independent vectors can be obtained. One set of these procedures from $i = 1$ to $n-1$ is usually called a *tour*. The original version of the BKZ algorithm stops when no update occurs during $n - 1$ iterations. In this paper, we refer to the BKZ algorithm with blocksize β as the BKZ-β.

HKZ reduced basis: The lattice basis $(\mathbf{b}_1, \ldots, \mathbf{b}_n)$ is called Hermite-Korkine-Zolotarev (HKZ) reduced [65, Chapter 2] if it is size-reduced $|\mu_{ij}| \leq 1/2$ for all i and j, and $\pi_i(\mathbf{b}_i)$ is the shortest vector in the projective sublattice $L_{[i:n]}$ for all i. We can estimate the Gram-Schmidt length of the HKZ-reduced basis by using the Gaussian heuristic as $\|\mathbf{b}_i^*\| = \mathrm{GH}(L_{[i:n]})$. Since the HKZ-reduced basis is completely reduced in this sense, we will use this to discuss the lower bound of computing time in Section 3.8.2.

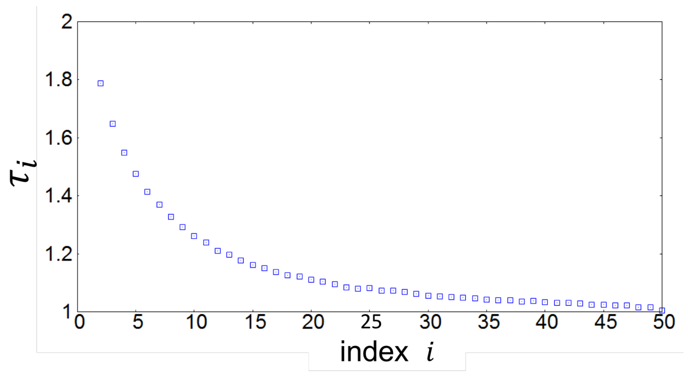

FIGURE 3.2: Modified Gaussian heuristic constant τ_i

3.3.1 Some Heuristic Assumptions in BKZ

Gaussian Heuristic in Small Dimensions: Chen and Nguyen observed that the length $\lambda_1(B_i)$ of the shortest vector in the local block B_i is usually larger than $\mathrm{GH}(B_i)$ in small dimensions i.e., small β' [24]. They gave the averaged values of $\|\mathbf{b}_i^*\|/\det(L)^{1/n}$ for the last indexes of highly reduced bases to modify their BKZ simulator, see [24, Appendix C]. For their 50 simulated values for $\|\mathbf{b}_{n-49}^*\|, \ldots, \|\mathbf{b}_n^*\|$, we define the modified Gaussian heuristic constant by

$$\tau_i := \frac{\lambda_1(\pi_{n-i+1}(L))}{\mathrm{GH}(\pi_{n-i+1}(L))} = \frac{\|\mathbf{b}_{n-i+1}^*\|}{V_i(1)^{-1/i} \cdot \prod_{j=n-i+1}^{n} \|\mathbf{b}_j^*\|^{1/i}}. \tag{3.2}$$

We show the graph of τ_i in Figure 3.2. We will use τ_i for $i \leq 50$ to denote these modifying constants; for $i > 50$ we define $\tau_i = 1$ following Chen-Nguyen's simulator [24].

In the rest of this paper, we assume that the shortest vector lengths of β-dimensional local blocks B_i of reduced bases satisfies

$$\lambda_1(B_i) \approx \begin{cases} \tau_\beta \cdot \mathrm{GH}(B_i) & (\beta \leq 50) \\ \mathrm{GH}(B_i) & (\beta > 50) \end{cases}$$

on average.

We note that there exists a mathematical theory to guarantee $\tau_i \to 1$ for random lattices when the dimension goes to infinity [71]. Though it does not give the theoretical guarantee $\tau_i = 1$ for BKZ local blocks, they are very close in our preliminary experiments.

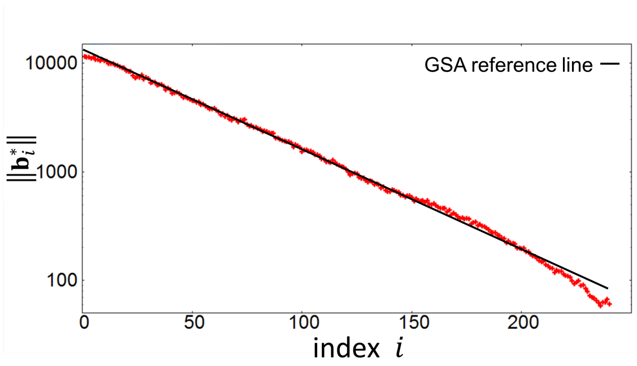

FIGURE 3.3: Semi-log graph of $\|\mathbf{b}_i^*\|$ of a 240-dimensional highly reduced basis

Geometric Series Assumption (GSA): Schnorr [73] introduced *geometric series assumption* (GSA), which says that the Gram-Schmidt lengths $\|\mathbf{b}_i^*\|$ in the BKZ-reduced basis decay geometrically with quotient r for $i = 1, \ldots, n$, namely, $\|\mathbf{b}_i^*\|^2/\|\mathbf{b}_1\|^2 = r^{i-1}$, for some $r \in [3/4, 1)$. Here r is called *the GSA constant*. Figure 3.3 shows the Gram-Schmidt lengths of a 240-dimensional reduced basis after processing BKZ-100 using our algorithm and parameters.

It is known that GSA does not hold exactly in the first and last indexes [22]. Several previous works [1, 22, 73] aimed to modify the reduction algorithm that outputs the reduced basis satisfying GSA. However, it seems difficult to obtain such a reduced basis in practice. In this paper, we aim to modify the parameters in the first and last indexes so that the proposed simulator performs with optimal efficiency (See section 3.5.1).

3.3.2 Chen-Nguyen's BKZ 2.0 Algorithm [24]

We recall Chen-Nguyen's BKZ 2.0 Algorithm in this section. The outline of the BKZ 2.0 algorithm is described in Algorithm 1.

Speed-up Techniques for BKZ 2.0: BKZ 2.0 employs four major speed-up techniques that differentiate it from the original BKZ:

1. BKZ 2.0 employs the extreme pruning technique [37], which attempts to find shorter vectors in the local blocks B_i with low probability p by randomizing basis B_i to more blocks $G_1, \ldots, G_M$ where $M = \lfloor 1/p \rceil$.

Algorithm 1 An outline of BKZ 2.0

```
Input: A lattice basis $B$ of $n$ dimensions, blocksize $\beta$, and some terminating condition.
Output: A reduced basis $B$.
1: $B \leftarrow \mathrm{LLL}(B)$;
2: for $i = 1$ to $n - 1$
3:     Set probability $p$ for local block $B_i$ of fixed blocksize $\beta_i' = \min(\beta, n - i + 1)$ and
       let $M = \lfloor 1/p \rceil$;
4:     Generate randomized local blocks $G_1, \dots, G_M$ from local block $B_i$, and
       preprocess $G_1, \dots, G_M$ (reduction by LLL and small blocksize BKZ);
5:     Find a vector $\mathbf{v}$ by lattice enumeration with radius $c = \min\{\|\mathbf{b}_i^*\|, \sqrt{1.1} \cdot \mathrm{GH}(B_i)\}$
       for $G_1, \dots, G_M$ with probability $p$;
6:     if $\mathbf{v}$ satisfies $\|\mathbf{v}\| < \|\mathbf{b}_i^*\|$ then update basis $B$ by $\mathbf{v}$;
7: end-for
8: if terminating condition is satisfied then return $B$;
9: else goto step 2.
```

2. For the search radius $\min\{\|\mathbf{b}_i^*\|, \alpha \cdot \mathrm{GH}(B_i)\}$ in the enumeration algorithm of the local block B_i, Chen and Nguyen set the value as $\alpha = \sqrt{1.1}$ from their experiments, while the previous works set the radius as $\|\mathbf{b}_i^*\|$.

3. In order to reduce the cost of the enumeration algorithm, BKZ 2.0 preprocesses the local blocks by executing the sequence of BKZ algorithm, e.g., 3 tours of BKZ-50 and then 5 tours of BKZ-60, and so on. The parameters blocksize, number of rounds and number of randomized bases, are precomputed to minimize the total enumeration cost.

4. BKZ 2.0 uses the terminating condition introduced in [42], which aborts BKZ within small number of tours. It can find a short vector faster than the full execution of BKZ.

Chen-Nguyen's BKZ 2.0 Simulator: In order to predict the computational cost and the quality of the output basis, they also propose the simulating procedure of the BKZ 2.0 algorithm. Let $(\ell_1, \dots, \ell_n)$ be the simulated values of the GS-lengths $\|\mathbf{b}_i^*\|$ for $i = 1, \dots, n$. Then, the simulated values of the determinant and the Gaussian heuristic are represented by $\prod_{j=1}^{n} \ell_j$ and $\mathrm{GH}(B_i) = V_{\beta'}(1)^{-1/\beta'} \prod_{j=i}^{i+\beta'-1} \ell_i$ where $\beta' = \min\{\beta, n-i+1\}$, respectively.

They simulate a BKZ tour of blocksize β assuming that each enumeration procedure finds a vector of projective length $\mathrm{GH}(B_i)$. Roughly speaking, their simulator updates (ℓ_i, ℓ_{i+1}) to (ℓ_i', ℓ_{i+1}') for $i = 1, \dots, n-1$, where $\ell_i' = \mathrm{GH}(B_i)$ and $\ell_{i+1}' = \ell_{i+1} \cdot (\ell_i / \ell_i')$.

Here, the last 50 GS-lengths are modified using an HKZ reduced basis. The details of their simulator are given in [24, Algorithm 3].

They also present the upper and lower bounds for the number of processed nodes during the lattice enumeration of blocksize β. From [25, Table 4], we extrapolate the costs as

$$\log_2(\mathrm{Cost}_\beta) = 0.000784314\beta^2 + 0.366078\beta - 6.125 \tag{3.3}$$

Then, the total enumeration cost of performing the BKZ 2.0 algorithm using blocksize β and t tours is given by

$$t \cdot \sum_{i=1}^{n-1} \mathrm{Cost}_{\min\{\beta, n-i+1\}}. \tag{3.4}$$

To convert the number of nodes into single-threaded time in seconds, we use the rational constant $4 \cdot 10^9/200 = 2 \cdot 10^7$, because they assumed that processing one node requires 200 clock cycles in a standard CPU, and we assume it can work at 4.0GHz.

We note that there are several models to extrapolate $\log_2(\mathrm{Cost}_\beta)$. Indeed, Lepoint and Naehrig [54] consider two models by a quadratic interpolation and a linear interpolation from the table. Albrecht et al. [6] showed another BKZ 2.0 cost estimation that uses an interpolation using the cost model $\log_2(\mathrm{Cost}_\beta) = O(n \log n)$. It is a highly non-trivial task to find a proper interpolation that estimates a precise cost of the BKZ 2.0 algorithm.

We further mention that the upper bound of the simulator is somewhat debatable, because they use the enumeration radius $c = \min\{\sqrt{1.1} \cdot \mathrm{GH}(B_i), \|\mathbf{b}_i^*\|\}$ for $i < n - 30$ in their experiment whereas they assume $c = \mathrm{GH}(B_i)$ for the cost estimation in their upper bound simulation. Thus, the actual cost of BKZ 2.0 could differ by a factor of $1.1^{O(\beta)}$.

3.4 Optimizing Parameters in Plain BKZ

In this section we consider the plain BKZ algorithm described in Algorithm 2, and roughly predict the GS-lengths of the output basis, which were computed by the GSA constant r. Using this analysis, we can obtain the optimal settings for parameters (α, p) in Step 4 of the plain BKZ algorithm of blocksize β.

Algorithm 2 Plain BKZ algorithm

```
Input: A lattice basis $B$ of $n$ dimensions, blocksize $\beta$.
Output: A reduced basis $B$.
1: $B \leftarrow \mathrm{LLL}(B)$;
2: $flag = 1$  // set $flag = 0$ when the basis is updated;
3: for $i = 1$ to $n - 1$
4:     Set $(\alpha, p)$ for local block $B_i$ of fixed blocksize $\beta_i' = \min(\beta, n - i + 1)$;
5:     Execute lattice enumeration with probability $p$ and radius $\alpha \cdot \mathrm{GH}(B_i)$;
6:     if $\mathbf{v}$ satisfies $\|\mathbf{v}\| < \alpha \cdot \mathrm{GH}(B_i)$, then update basis $B$ by $\mathbf{v}$ and $flag = 0$;
7: end-for
8: if $flag = 1$ then return $B$ else goto Step 2;
```

3.4.1 Relationship of Parameters α, β, p, r

We fix the values of parameters (α, β) and assume that the lattice dimension n is sufficiently large.

Suppose that we found a vector $\mathbf{v}$ of $\|\mathbf{v}\| < \alpha \cdot \mathrm{GH}(B_i)$ in the local block B_i. We update the basis B_i by inserting $\mathbf{v}$ at ith index, and perform LLL or small blocksize BKZ on the updated basis.

Theorem 3.1 (Rogers' theorem [71]). *Let S be a convex symmetric Borel set with measure V. Let L be a lattice of determinant 1 and be of large dimension n. The number of vector pairs $(\mathbf{v}, -\mathbf{v})$ of L in S is asymptotic to the Poisson distribution with mean $1/2 \cdot V$.*

Proof. Please refer to the original paper [71] (the Theorem (3) and its proof). □

Due to Rogers' theorem, approximately $\alpha^n/2$ vector pairs $(\mathbf{v}, -\mathbf{v})$ exist within the ball of radius $c = \alpha \cdot \mathrm{GH}(L)$.

Corollary 3.2. *Since the pruning probability is defined for a single vector pair, we expect the actual probability that the enumeration algorithm finds at least one vector shorter than c is roughly*

$$1 - (1 - p)^{\alpha^n/2} \approx p \cdot \frac{\alpha^n}{2}. \tag{3.5}$$

The right-hand side of relation (3.5) is roughly computed by the binomial theorem, under the assumption that $p < 1$ is very small and $n > 1$ is not too big. Thus, there may exist one lattice vector in the searching space by setting parameter p as

$$p = \frac{2}{\alpha^\beta}. \tag{3.6}$$

Remark 3.3. The probability setting of equation (3.6) is an optimal choice under our assumption. If p is smaller, the enumeration algorithm finds no short vector with high probability and basis updating at ith index does not occur, which is a waste of time. On the other hand, if we take a larger p so that there exist $p \cdot \alpha^\beta/2 > 1$ vector pairs, the computational time of the enumeration algorithm increases more than $p \cdot \alpha^\beta/2$ times [37]. Although it can find shorter vectors, this is also a waste of time from the viewpoint of basis updating.

Assume that one vector is found using the enumeration, and also assume that the distribution of it is the same as the random point in the β-dimensional ball of radius $\alpha \cdot \mathrm{GH}(B_i)$. Then, the expected value of $\|\mathbf{v}\|$ is $\frac{\beta}{\beta+1}\alpha \cdot \mathrm{GH}(B_i)$ by letting $K = 1$ in Lemma 1. Thus, we can expect that this is the value $\|\mathbf{b}_i^*\|$ after update.

Therefore, after executing a sufficient number of BKZ tours, we can expect that all the lengths $\|\mathbf{b}_i^*\|$ of the Gram-Schmidt basis satisfy

$$\|\mathbf{b}_i^*\| = \frac{\beta}{\beta+1}\alpha \cdot \mathrm{GH}(B_i) \tag{3.7}$$

on average. Hence, under Schnorr's GSA, we have the relation

$$\|\mathbf{b}_i^*\| = \frac{\alpha\beta}{\beta+1} \cdot V_\beta(1)^{-1/\beta} \|\mathbf{b}_i^*\| \prod_{j=1}^{\beta} r^{(j-1)/2\beta}, \tag{3.8}$$

and the GSA constant is

$$r = \left(\frac{\beta+1}{\alpha\beta}\right)^{\frac{4}{\beta-1}} \cdot V_\beta(1)^{\frac{4}{\beta(\beta-1)}}. \tag{3.9}$$

Therefore, by fixing (α, β), we can set the probability p and obtain r as a rough prediction of the quality of the output lattice basis. We will use the relations (3.6) and (3.9) to set our parameters. Note that any two of α, β, p and r are determined from the other two values.

Remark 3.4. Our estimation is somehow underestimate, i.e., in our experiments, the found vectors during BKZ algorithm are often shorter than the estimation in equation

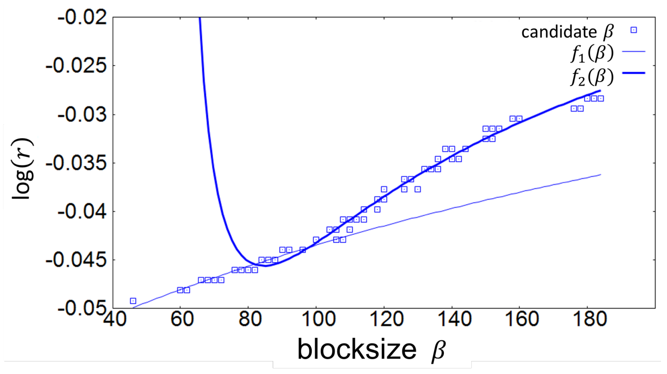

FIGURE 3.4: Relation between β and r that minimizes the computational cost

(3.7). This gap is mainly from the estimation in (3.5), which can be explained as follows. Let $(R_1, \ldots, R_\beta)$ be a bounding function of probability p for a vector of length $\|v\|$. Then, the probability p' for a vector of length $\|v'\|$ of a shorter vector is the same as the scaled bounding function $(R'_1, \ldots, R'_\beta)$ where $R'_i = \min\{1, R_i \cdot \|v\|/\|v'\|\}$. Here, p' is clearly larger than p due to $R'_i \geq R_i$ for $i \in [\beta]$. Therefore, when the above parameters are used, the quality of the output basis is better than that derived from equation (3.9) if we perform a sufficient number of tours. Hence, within a few tours, our algorithm can output a basis which has a good quality predicted by our estimation in this section.

3.4.2 Optimizing Parameters

Now for a fixed parameter pair (β, r), the cost $\mathrm{ENUMCost}(B_i; \alpha, p)$ of the enumeration algorithm in local block B_i satisfying GSA is computable. Concretely, we compute α using the relation (3.9), fix p by (3.6), and simulate the Gram-Schmidt lengths of B_i using $\|\mathbf{b}^*_i\| = r^{(i-1)/2}$. Using the computation technique in [37], for several GSA constants r, we search for the optimal blocksize β that minimizes the enumeration cost $\mathrm{ENUMCost}(B_i; \alpha, p)$. The small squares in Figure 3.4 show the results. From these points, we find the functions $f_1(\beta)$ and $f_2(\beta)$, whose graphs are also in the figure.

We explain how to find these functions $f_1(\beta)$ and $f_2(\beta)$. Suppose lattice dimension n is sufficiently large, and suppose the cost of the enumeration algorithm is roughly dominated by the probability p times the factor at $k = n/2$ in the summation (3.1). Then

$\mathrm{ENUMCost}(B_i; \alpha, p)$ is approximately

$$D = p \cdot \frac{V_{\beta/2}(\alpha \cdot \mathrm{GH}(B_r))}{\prod_{i=\beta/2+1}^{\beta} \|\mathbf{b}_i^*\|} = 2\alpha^{-\beta/2} \frac{V_{\beta/2}(1) V_\beta(1)^{-1/2}}{r^{\beta^2/16}},$$

where from equation (3.9) we have obtained

$$D \approx Const. \times r^{(\beta^2-2\beta)/16} \cdot \left(\frac{\beta}{e\pi}\right)^{\beta/4}, \text{ and } \frac{\partial \log D}{\partial \beta} \approx \frac{\beta-1}{8} \log r + \frac{1}{4} + \frac{1}{4} \log \frac{\beta}{e\pi}.$$

In order to minimize D, we roughly need the above derivative to be zero; thus, we use the following function of β for our cost estimation with constants c_i

$$\log(r) = 2 \cdot (\log \beta + 1 - \log(e\pi))/(1-\beta) = \frac{\log c_1 \beta}{c_2 \beta + c_3}.$$

From this observation, we fix the fitting function model as $f(\beta) = \frac{\log(c_1\beta + c_2)}{c_3\beta + c_4}$.

By using the least squares method implemented in `gnuplot`, we find the coefficients c_i so that $f(\beta)$ is a good approximation of the pairs $(\beta_i, \log(r_i))$. In our curve fitting, we separate the range of β into the interval $[40, 100]$, and the larger one. This is needed for converging to $\log(r) = 0$ when β is sufficiently large; however, our curve fitting using a single natural function did not achieve it. Curves $f_1(\beta)$ and $f_2(\beta)$ in Figure 3.4 are the results of our curve fitting for the range $[40, 100]$ and the larger one, respectively.

$$\log(r) = \begin{cases} -18.2139/(\beta + 318.978) & (\beta \le 100) \\ (-1.06889/(\beta - 31.0345)) \cdot \log(0.417419\beta - 25.4889) & (\beta > 100) \end{cases} \tag{3.10}$$

Moreover, we obtain pairs of β and minimize $\mathrm{ENUMCost}(B_i; \alpha, p)$, in accordance with the above experiments. Using the curve fitting that minimizes $\sum_\beta |f(\beta) - \log_2 \mathrm{ENUMCost}(B_i; \alpha, p)|^2$ using `gnuplot`, we find the extrapolating formula

$$\log_2 \mathrm{MINCost}(\beta) := \begin{cases} 0.1375\beta + 7.153 & (\beta \in [60, 105]) \\ 0.000898\beta^2 + 0.270\beta - 16.97 & (\beta > 105) \end{cases} \tag{3.11}$$

to $\log_2 \mathrm{ENUMCost}(B_i; \alpha, p)$. We will use this as the standard of the enumeration cost of blocksize β.

Remark 3.5. Our estimation from the real experiments is $0.25\beta \cdot \mathrm{ENUMCost}(B_i; \alpha, p)$ (See, Section 3.7.1), which crosses over the estimation of BKZ 2.0 simulator (3.3) at $\beta = 873$. Thus, the performance of BKZ 2.0 might be better in some extremely high block sizes, while our algorithm has a better performance in the realizable block sizes < 200.

3.4.3 Parameter Settings in Step 4 in Algorithm 2

Using the above arguments, we can fix the optimized pair (α, p) for each blocksize β. That is, to process a local block of blocksize β in Step 4 of the plain BKZ algorithm in Algorithm 2, we compute the corresponding r by equations (3.10), and additionally obtain the parameters α by equation (3.9) and p by equation (3.6). These are our basic parameter settings.

Modifying blocksize at first indexes: We sometimes encounter the phenomenon in which the actual $\mathrm{ENUMCost}(B_i; \alpha, p)$ in small indexes is much smaller than that in middle indexes. This is because $\|\mathbf{b}_i^*\|$ is smaller than $\mathrm{GH}(B_i)$ in small indexes. In other words, $\mathbf{b}_i$ is hard to update using the enumeration of blocksize β. To speed up the lattice reduction, we use a heuristic method that enlarges the blocksizes as follows.

From the discussion in the above subsection, we know the theoretical value of the enumeration cost at blocksize β. On the other hand, in the actual computing of BKZ algorithms, the enumeration cost is increased because the sequence $(\|\mathbf{b}_i^*\|, \ldots, \|\mathbf{b}_{i+\beta-1}^*\|)$, which mainly affects the computing cost, does not follow the GSA of slope r exactly. In some experiments, we will verify that the enumeration cost is approximately β times the theoretical value (See, Figure 3.6 in Section 3.7.1). Thus, we define the expected enumeration cost in blocksize β as $\beta \cdot \mathrm{MINCost}(\beta)$. With this expectation, we reset the blocksize as the minimum β satisfying $\mathrm{ENUMCost}(B_{[i:i+\beta-1]}; \alpha, p) > \beta \cdot \mathrm{MINCost}(\beta)$.

Modifying (α, p) at last indexes: For large indexes such as $i > n - \beta$, the blocksize of a local block shrinks to $\beta' = \min(\beta, n - i + 1)$. In our implementation, we increase the success probability to a new p', while $\mathrm{ENUMCost}(B_i; \alpha', p')$ is smaller than $\beta \cdot \mathrm{MINCost}(\beta)$. We also reset the radius as $\alpha' = (2/p')^{1/\beta}$ from equation (3.6).

3.5 Our Proposed Progressive BKZ: Basic Variant

Algorithm 3 Our progressive BKZ algorithm (basic variant)

```
Input: A lattice basis $B$ of $n$ dimensions, starting blocksize $\beta_{start}$, and ending blocksize $\beta_{end}$.
Output: A reduced basis $B$.
1: $B \leftarrow \mathrm{LLL}(B)$;
2: for $\beta = \beta_{start}$ to $\beta_{end}$ do
3:   while $\mathrm{FEC}(B) > \text{Sim-FEC}(n, \beta)$ do
4:     for $i = 1$ to $n - 1$
5:       Set $(\alpha, p)$ for local block $B_i$ of blocksize $\beta' = \min(\beta, n - i + 1)$
         using the setting in Section 3.4.3;
6:       Preprocess the basis by the progressive BKZ;
7:       Execute lattice enumeration with probability $p$ and radius $\alpha \cdot \mathrm{GH}(B_i)$;
8:       if $\mathbf{v}$ satisfies $\|\mathbf{v}\| < \alpha \cdot \mathrm{GH}(B_i)$ then update basis $B$ by $\mathbf{v}$;
9:     end-for
10:  end-while
11: end-for
```

In this section, we explain the basic variant of our proposed progressive BKZ algorithm.

In general, if the blocksize of the BKZ algorithm increases, a shorter vector $\mathbf{b}_1$ can be computed; however, the running cost will eventually increase. The progressive BKZ algorithm starts a BKZ algorithm with a relatively small blocksize β_{start} and increases the blocksize to β_{end} by some criteria. The idea of the progressive BKZ algorithm has been mentioned in several literatures, for example, [24, 74, 77, 43]. The research challenge in the progressive BKZ algorithm is to find an effective criteria for increasing blocksizes that minimizes the total running time.

In this paper we employ the *full enumeration cost* (FEC) in Section 3.2.1, in order to evaluate the quality of the basis for finding the increasing criteria. Recall that the FEC of basis $B = (\mathbf{b}_1, \ldots, \mathbf{b}_n)$ of n-dimensional lattice L is defined by $\mathrm{FEC}(B) = \sum_{k=1}^{n} \frac{V_k(\mathrm{GH}(L))}{\prod_{i=n-k+1}^{n} \|\mathbf{b}_i^*\|}$, where $\|\mathbf{b}_i^*\|$ represents the GS-lengths. Note that $\mathrm{FEC}(B)$ eventually decreases after performing several tours of the BKZ algorithm using the fixed blocksize β.

Moreover, we construct a simulator that evaluates the GS-lengths by the optimized parameters α, β, p, r for the BKZ algorithm described in the local block discussion in Section 3.4.3. The simulator for an n-dimensional lattice only depends on the blocksize

β of the local block; we denote by $\mathrm{Sim\text{-}GS\text{-}lengths}(n,\beta)$ the simulated GS-lengths $(\ell_1,\ldots,\ell_n)$. The construction of simulator will be presented in Section 3.5.1.

For this purpose, we define some functions defined on the simulated GS-lengths $(\ell_1,\ldots,\ell_n)$. $\mathrm{Sim\text{-}GH}(\ell_1,\ldots,\ell_n) = V_n(1)^{-1/n}\prod_{j=1}^{n}\ell_j^{1/n}$ is the simulated Gaussian heuristic. The simulated value of full enumeration cost is

$$\mathrm{Sim\text{-}FEC}(\ell_1,\ldots,\ell_n) := \sum_{k=1}^{n}\frac{V_k(\mathrm{Sim\text{-}GH}(\ell_1,\ldots,\ell_n))}{\prod_{i=n-k+1}^{n}\ell_i}.$$

Further, for $(\ell_1,\ldots,\ell_n) = \mathrm{Sim\text{-}GS\text{-}lengths}(n,\beta)$, we use the notation $\mathrm{Sim\text{-}FEC}(n,\beta) := \mathrm{Sim\text{-}FEC}(\ell_1,\ldots,\ell_n)$ in particular. The simulated enumeration cost $\mathrm{Sim\text{-}ENUMCost}(\ell_1,\ldots,\ell_\beta;\alpha,p)$ is defined by $\mathrm{ENUMCost}(B;\alpha,p)$ for a lattice basis B that has GS-lengths $\|\mathbf{b}_i^*\| = \ell_i$ for $i \in [\beta]$.

The key point of our proposed progressive BKZ algorithm is to increase the blocksize β if $\mathrm{FEC}(B)$ becomes smaller than $\mathrm{Sim\text{-}FEC}(n,\beta)$. In other words, we perform the BKZ tours of blocksize β while $\mathrm{FEC}(B) > \mathrm{Sim\text{-}FEC}(n,\beta)$. We describe the proposed progressive BKZ in Algorithm 3.

Remark 3.6. In the basic variant of our progressive BKZ described in Section 3.6.1, we increase the blocksize β in increments of one in Step 2. However, we will present an optimal strategy for increasing the blocksize in Section 3.6.

3.5.1 $\mathrm{Sim\text{-}GS\text{-}lengths}(n,\beta)$: Predicting Gram-Schmidt Lengths

In the following, we construct a simulator for predicting the Gram-Schmidt lengths $\|\mathbf{b}_i^*\|$ obtained from the plain BKZ algorithm of blocksize β.

Our simulator consists of two phases. First, we generate approximated GS-lengths using Gaussian heuristics; we then modify it for the first and last indexes of GSA in Section 3.3.1. We will explain how to compute $(\ell_1,\ldots,\ell_n)$ as the output of $\mathrm{Sim\text{-}GS\text{-}lengths}(n,\beta)$.

First phase: Our simulator computes the initial value of $(\ell_1,\ldots,\ell_n)$.

We start from the last index by setting $\ell_n = 1$, and compute ℓ_i backwards. From equations (3.2) and (3.7) we are able to simulate the GS-lengths ℓ_i by solving the following equation of ℓ_i:

$$\ell_i = \max\left\{\frac{\beta'}{\beta'+1}\alpha, \tau_{\beta'}\right\} \cdot \mathrm{GH}(\ell_i, \dots, \ell_{i+\beta'-1}), \text{ where } \beta' = \min(\beta, n-i+1). \quad (3.12)$$

Here, α is the optimized radius parameter in Section 3.4.3 and $\tau_{\beta'}$ is the coefficient of the modified Gaussian heuristic.

This simple simulation in the first phase is sufficient for smaller blocksizes ($\beta < 30$). However, for simulating larger blocksizes, we must modify the GS-lengths of the first and last indexes in Section 3.3.1.

Second phase: To modify the results of the simple simulation, we consider our two modifying methods described in Section 3.4.3. We recall that $\mathrm{MINCost}(\beta)$ is the standard value of the enumeration cost of blocksize β.

We first consider the modification for the last indexes $i > n - \beta + 1$, i.e., a situation in which the blocksize is smaller than β. We select the modified probability p_i at index i so that $\text{Sim-ENUMCost}(\ell_i, \dots, \ell_n; \alpha_i, p_i) = \beta \cdot \mathrm{MINCost}(\beta)$, where $\ell_i, \dots, \ell_n$ is the result of the first simulation, and we use $\alpha_i = (2/p_i)^{n-i+1}$. After all (α_i, p_i) for $n - \beta + 1 \le i \le n$ are fixed, we modify the GS-lengths by solving the following equation of ℓ_i again:

$$\ell_i = \max\left\{\frac{\beta'}{\beta'+1}\alpha_i, \tau_{\beta'}\right\} \cdot \mathrm{GH}(\ell_i, \dots, \ell_n) \text{ where } \beta' = n - i + 1.$$

Next, using the modified $(\ell_1, \dots, \ell_n)$, we again modify the first indexes as follows. We determine the integer parameter $b > 0$ for the size of enlargement. For $b = 1, 2, \dots$, we reset the blocksize at index i as $\beta_i := \beta + \max\{(b-i+1)/2, b-2(i-1)\}$ for $i \in \{1, \dots, b\}$. Using these blocksizes, we recompute the GS-lengths by solving equation (3.12) from $i = \beta_i$ to 1. Then, we compute $\text{Sim-ENUMCost}(\ell_1, \dots, \ell_{\beta+b}; \alpha, p)$. We select the maximum b such that this simulated enumeration cost is smaller than $2 \cdot \mathrm{MINCost}(\beta)$.

Experimental result of our GS-lengths simulator: We performed some experiments on the GS-lengths for some random lattices from the Darmstadt SVP Challenge [28]. We computed the GS-lengths for 120, 150 and 200 dimensions using the proposed progressive BKZ algorithm, with ending blocksizes of 40, 60, and 100, respectively (Note that the starting blocksize is irrelevant to the quality of the GS-lengths). The simulated result is shown in Figure 3.5. Almost all small squares of the computed

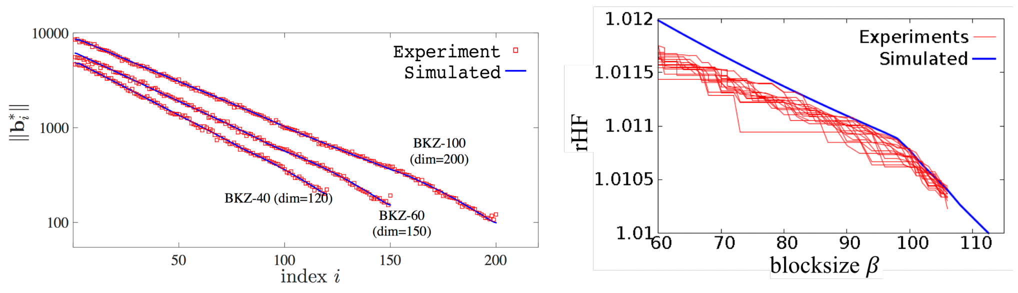

FIGURE 3.5: **Left figure**: Semi-log graph of $\|\mathbf{b}_i^*\|$ of reduced random lattices from the SVP Challenge problem generator: Simulation (bold lines) vs. Experiment (small squares). **Right figure**: The root Hermite factor of reduced random 300-dimensional bases after BKZ-β. Simulation (bold red lines) vs. Experiment (thin blue lines).

GS-lengths are plotted on the bold line obtained by our above simulation. Our simulator can precisely predict the GS-lengths of these lattices. The progress of the first vector, which uses 300-dimensional lattices, is also shown in the figure.

3.5.2 Expected Number of BKZ Tours at Step 3

At Step 3 in the proposed algorithm (Algorithm 3) we iterate the BKZ tour with blocksize β as long as the full enumeration cost $\mathrm{FEC}(B)$ is larger than the simulated cost $\text{Sim-FEC}(n,\beta)$. In the following we estimate the expected number of BKZ tours (we denote it as $\sharp\, tours$) at blocksize β.

In order to estimate $\sharp\, tours$, we first compute $(\ell_1,\ldots,\ell_n)$ and the output of $\text{Sim-GS-lengths}(n,\beta-1)$, and update it by using the modified Chen-Nguyen's BKZ 2.0 simulator described in Section 3.3.2, until $\text{Sim-FEC}(\ell_1,\ldots,\ell_n)$ is smaller than $\text{Sim-FEC}(n,\beta)$. We simulate a BKZ tour by updating the pair (ℓ_i,ℓ_{i+1}) to (ℓ'_i,ℓ'_{i+1}) for $i=1,\ldots,n-1$ according to the following rule:

$$\begin{aligned} \ell'_i &= \max\left\{\tfrac{\beta}{\beta+1}\alpha, \tau_\beta\right\}\cdot \mathrm{GH}(\ell_i,\ldots,\ell_{\min(n,i+\beta-1)}) \\ \text{and}\quad \ell'_{i+1} &= \ell_{i+1}\cdot(\ell_i/\ell'_i). \end{aligned}$$

Note that the former one is more important than the latter, since the latter heuristic is to make sure the accuracy of $\mathrm{GH}(\cdot)$ in the former one. At the simulation of tth BKZ tour, write the input GS-lengths $(\ell'_1,\ldots,\ell'_n)$; i.e., the output of the $(t-1)$th BKZ tour. We further denote the output of tth BKZ tour as $(\ell_1, \dots, \ell_n)$. Suppose they satisfy

$$\text{Sim-FEC}(\ell'_1, \dots, \ell'_n) > \text{Sim-FEC}(n, \beta) > \text{Sim-FEC}(\ell_1, \dots, \ell_n).$$

Then, our estimation of $\sharp tours$ is the interpolated value:

$$\sharp tours = (t-1) + \frac{\text{Sim-FEC}(\ell'_1, \dots, \ell'_n) - \text{Sim-FEC}(n, \beta)}{\text{Sim-FEC}(\ell'_1, \dots, \ell'_n) - \text{Sim-FEC}(\ell_1, \dots, \ell_n)}. \tag{3.13}$$

Note that we can use this estimation for other BKZ strategies, although we estimate the number of BKZ tours from BKZ-$(\beta-1)$ basis to BKZ-β basis, using BKZ-β algorithm. We will estimate the tours for other combinations of starting and ending blocksizes, and use them in the algorithm.

3.6 Optimizing Blocksize Strategy in Progressive BKZ

We propose how to optimally increase the blocksize β in the proposed progressive BKZ algorithm. Several heuristic strategies for increasing the blocksizes have been proposed. The following sequences of blocksizes after LLL-reduction have been used in the previous literatures:

20	$\to 21$	$\to 22$	$\to 23$	$\to 24$	$\to \cdots$	Gama and Nguyen [36]
2	$\to 4$	$\to 8$	$\to 16$	$\to 32$	$\to \cdots$	Schnorr and Shevchenko [77],
2	$\to 4$	$\to 6$	$\to 8$	$\to 10$	$\to \cdots$	Haque, Rahman, and Pieprzyk [43],
50	$\to 60$	$\to 70$	$\to 80$	$\to 90$	$\to \cdots$	Chen and Nguyen [24, 25]

The timings for changing to the next blocksize were not explicitly given. They sometimes continue the BKZ tour until no update occurs as the original BKZ. In this section we try to find the sequence of the blocksizes that minimizes the total cost of the progressive BKZ to find a BKZ-β reduced basis. To find this strategy, we consider all the possible combinations of blocksizes used in our BKZ algorithm and the timing to increase the blocksizes.

Notations on blocksize strategy: We say a lattice basis B of dimension n is β-reduced when $\text{FEC}(B)$ is smaller than $\text{Sim-FEC}(n, \beta)$. For a tuple of blocksizes

$(\beta^{alg}, \beta^{start}, \beta^{goal})$ satisfying $2 \le \beta^{start} < \beta^{goal} \le \beta^{alg}$, the notation

$$\beta^{start} \overset{\beta^{alg}}{\to} \beta^{goal}$$

is the process of the BKZ following algorithm. The input is a β^{start}-reduced basis B, and the algorithm updates B using the tours of BKZ-β^{alg} algorithm with parameters in Section 3.4.3. It stops when $\mathrm{FEC}(B) < \text{Sim-FEC}(n, \beta^{goal})$.

$\mathrm{TimeBKZ}(n, \beta^{start} \overset{\beta^{alg}}{\to} \beta^{goal})$ is the computing time in seconds of this algorithm. We provide a concrete simulating procedure in this and the next sections. We assume that $\mathrm{TimeBKZ}$ is a function of $n, \beta^{alg}, \beta^{start}$ and β^{goal}.

To obtain a BKZ-β reduced basis from an LLL reduced basis, many blocksize strategies are considered as follows:

$$\beta_0^{goal} = \mathrm{LLL} \overset{\beta_1^{alg}}{\to} \beta_1^{goal} \overset{\beta_2^{alg}}{\to} \beta_2^{goal} \overset{\beta_3^{alg}}{\to} \cdots \overset{\beta_D^{alg}}{\to} \beta_D^{goal} (= \beta). \qquad (3.14)$$

We denote this sequence as $\{(\beta_j^{alg}, \beta_j^{goal})\}_{j=1,\ldots,D}$, and regard it as the progressive BKZ given in Algorithm 4.

Algorithm 4 Our progressive BKZ algorithm with blocksize strategy

```
Input: A lattice basis $B$ of $n$ dimensions, Blocksize strategy $\{(\beta_j^{alg}, \beta_j^{goal})\}_{j=1,\ldots,D}$.
Output: A $\beta_D^{goal}$-reduced basis $B$.
1: $B \leftarrow \mathrm{LLL}(B)$;
2: for $j = 1$ to $D$ do
3:   while $\mathrm{FEC}(B) > \text{Sim-FEC}(n, \beta_j^{goal})$ do
4:     The same as Step 4-9 in Algorithm 3 with blocksize $\beta_j^{alg}$
5:   end-while
6: end-for
```

3.6.1 Optimizing Blocksize Strategies

Our goal in this section is to find the optimal sequence that minimizes the total computing time

$$\sum_{i=1}^{D} \mathrm{TimeBKZ}(n, \beta_{i-1}^{goal} \overset{\beta_i^{alg}}{\to} \beta_i^{goal}) \qquad (3.15)$$

of the progressive BKZ algorithm to find a BKZ-β_D^{goal} basis.

Based on our experimental results, which are given in Section 3.7, we can estimate the computing time of the BKZ algorithm:

$$\begin{aligned}&\mathrm{TimeBKZ}(n, \beta^{start} \overset{\beta^{alg}}{\to} \beta^{goal}) \text{ [sec.]}\\ &= \sum_{t=1}^{\sharp tours} \left[10^{-10} \cdot (\beta^{alg})^2 n^3 + 1.5 \cdot 10^{-8} \cdot \beta^{alg} \sum_{i=1}^{n-1} \mathrm{ENUMCost}(B_i; \alpha, p)\right]\end{aligned} \tag{3.16}$$

when dimension n is small ($n < 400$), and

$$\begin{aligned}&\mathrm{TimeBKZ}(n, \beta^{start} \overset{\beta^{alg}}{\to} \beta^{goal}) \text{ [sec.]}\\ &= \sum_{t=1}^{\sharp tours} \left[2.5 \cdot 10^{-4} \frac{n - \beta^{alg}}{250 - \beta^{alg}} \cdot n^2. + 1.5 \cdot 10^{-8} \cdot \beta^{alg} \sum_{i=1}^{n-1} \mathrm{ENUMCost}(B_i; \alpha, p)\right]\end{aligned} \tag{3.17}$$

when dimension n is large ($n \geq 400$). The difference is caused by the difference in the types to compute Gram-Schmidt variables in implementation. The former and latter implementation employ `quad_float` and `RR` (320 bits) respectively, where `RR` is the arbitrary precision floating point type in the NTL library [79]. To compute $\sharp tours$ we use the procedure in Section 3.5.2. The input of the $\mathrm{ENUMCost}$ function is from $\text{Sim-GS-lengths}(n, \beta^{start})$ at the first tour. From the second tour, we use the updated GS-lengths by the Chen-Nguyen's simulator with blocksize β^{alg}.

Using these computing time estimations, we discuss how to find the optimal blocksize strategy (3.14) that minimizes the total computing time. In this optimizing procedure, the input consists of n and β, the lattice dimension and the goal blocksize. We denote $\mathrm{TimeBKZ}(n, \beta^{goal})$ to be the minimized time in seconds to find a β-reduced basis from an LLL reduced basis, that is, the minimum of (3.15) from among the possible blocksize strategies. By definition, we have

$$\mathrm{TimeBKZ}(n, \beta^{goal}) = \min_{\beta', \beta^{alg}} \left\{\mathrm{TimeBKZ}(n, \beta') + \mathrm{TimeBKZ}(n, \beta' \overset{\beta^{alg}}{\to} \beta^{goal})\right\}$$

where we take the minimum over the pair of blocksizes (β', β^{alg}) satisfying $\beta' < \beta^{goal} \leq \beta^{alg}$.

For the given (n, β), our optimizing algorithm computes $\mathrm{TimeBKZ}(n, \bar{\beta})$ from small $\bar{\beta}$ to the target $\bar{\beta} = \beta$. As the base case, we define that $\mathrm{TimeBKZ}(n, 20)$ represents the time to compute a BKZ-20 reduced basis using a fixed blocksize, starting from an LLL reduced basis:

$$\mathrm{TimeBKZ}(n,20) := \min_{\beta^{alg}} \left\{ \mathrm{TimeBKZ}(n, \mathrm{LLL} \overset{\beta^{alg}}{\to} 20) \right\}.$$

3.6.2 Simulating Time to Find Short Vectors in Random Lattices

In this section, we give our simulating result of finding short vectors for random lattices. For the given lattice dimension n and the target length, we simulate the necessary BKZ blocksize β so that ℓ_1 of $\mathrm{Sim\text{-}GS\text{-}lengths}(n,\beta)$ is smaller than the target length. Then, we simulate $\mathrm{TimeBKZ}(n,\beta)$ by using the method in Section 3.6.1.

As an example, in Table 3.2, we show the optimized blocksize strategy and computing time to find a 102-reduced basis in $n = 600$ dimension. We estimate blocksize 102 is necessary to find a vector shorter than $n \cdot \det(L)^{1/n}$, which is the condition to enter the Hall of Fame in the Approximate Ideal Lattice Challenge [29].

TABLE 3.2: The optimized blocksize strategy and computational time in seconds in 600-dimensional lattice.

$\overset{\beta^{alg}}{\to} \beta^{goal}$	LLL	$\overset{32}{\to} 21$	$\overset{50}{\to} 36$	$\overset{58}{\to} 46$	$\overset{65}{\to} 55$	$\overset{71}{\to} 61$	$\overset{75}{\to} 70$	$\overset{81}{\to} 76$	$\overset{85}{\to} 84$
$\log_2$(Time [sec.])		15.61	15.86	16.04	16.21	16.31	16.51	16.70	17.07
$\overset{\beta^{alg}}{\to} \beta^{goal}$		$\overset{89}{\to} 88$	$\overset{91}{\to} 90$	$\overset{93}{\to} 92$	$\overset{99}{\to} 98$	$\overset{101}{\to} 100$	$\overset{103}{\to} 102$		
$\log_2$(Time [sec.])		17.42	17.67	17.97	18.89	19.49	20.09		

Table 3.3 shows the blocksize and predicted total computing time in seconds to find a vector shorter than $n \cdot \mathrm{GH}(L)$ (this corresponds to the n-approximate SVP from the learning with errors problem [69].), $n \cdot \det(L)^{1/n}$ (from the Approximate Ideal Lattice Challenge published in Darmstadt [29]), and $\sqrt{n} \cdot \mathrm{GH}(L)$. For comparison, the simulating result of BKZ 2.0 is given to find $n \cdot \det(L)^{1/n}$. Recall that their estimated cost in seconds is given by $\sharp\,\mathrm{ENUM}/2 \cdot 10^7$. From Table 3.3, our algorithm is asymptotically faster than BKZ 2.0. Moreover, it is remarkable that we solved Ideal Lattice Challenge of 600 and 652 dimensions in the exact expected times of $2^{20.7}$ and $2^{24.0}$ seconds respectively by our improved progressive BKZ.

TABLE 3.3: Simulated $\log_2(\text{Time [sec.]})$ of our algorithm and BKZ 2.0 for large dimensions to find short vectors. The time is after LLL-reduced basis. Because the estimate for BKZ 2.0 is only the cost for enumeration, our algorithm appears to be slow in small blocksizes.

Goal	$n \cdot \mathrm{GH}(L)$		$n \cdot \det(L)^{1/n}$			$\sqrt{n} \cdot \mathrm{GH}(L)$	
n	β	$\log_2(\text{Ours})$	β	$\log_2(\text{Ours})$	$\log_2(\text{BKZ 2.0})$	β	$\log_2(\text{Ours})$
600	35	15.8	102	20.1	16.0	145	38.4
650	45	16.6	114	24.3	21.9	157	51.0
700	59	17.3	124	28.3	28.2	169	60.4
800	100	20.8	144	38.6	41.3	193	82.1

3.6.3 Comparing with Other Heuristic Blocksize Strategies

In this section, we compare the blocksize strategy of our progressive BKZ in Algorithm 4. Using a random 256-dimensional basis, we experimented and simulated the progressive BKZ to find a BKZ-128 reduced basis with the three following strategies:

$$2 \overset{4}{\to} 4 \overset{8}{\to} 8 \overset{16}{\to} 16 \overset{32}{\to} 32 \overset{64}{\to} 64 \overset{128}{\to} 128$$

(Schnorr-Shevchenko's doubling strategy [77])

$$2 \overset{20}{\to} 20 \overset{21}{\to} 21 \overset{22}{\to} 22 \overset{23}{\to} 23 \overset{24}{\to} 24 \overset{25}{\to} \cdots \overset{128}{\to} 128$$

(Simplest step-by-step in Algorithm 3)

$$2 \overset{30}{\to} 20 \overset{35}{\to} 25 \overset{39}{\to} 29 \overset{43}{\to} 33 \overset{47}{\to} 37 \overset{48}{\to} \cdots \overset{128}{\to} 128$$

(Optimized blocksize strategy in Algorithm 4)

In experiment, our simple and optimized strategy takes about 27.1 minutes and about 11.5 minutes respectively to achieve BKZ-64 basis after the LLL reduction. On the other hand, Schnorr-Schevchenko's doubling strategy takes about 21 minutes.

After then, the doubling strategy switches to BKZ-128 and takes about 14 single-core days to process the first one index, while our strategies comfortably continues the execution of progressive BKZ.

Our simulator predicts that it takes about $2^{25.3}$, $2^{25.1}$ and $2^{37.3}$ seconds to finish BKZ-128 by our simple, optimized, and Schnorr-Schevchenko's doubling strategy, respectively. Our strategy is about 5000 times faster than the doubling strategy.

Interestingly, we find that the computing time of simple blocksize strategy is close to that of optimized strategy in many simulations when the blocksize is larger than about

100. Hence, the simple blocksize strategy would be better than the optimizing blocksize strategy in practice, because the latter needs a heavy precomputing as in Section 3.6.1.

3.7 Our Implementation and Cost Estimation for Processing Local Blocks

In this section we describe how to derive the estimation of the computing times of equations (3.16) and (3.17) of Step 3-10 in Algorithm 3. The total computing time is the sum of times to process local blocks (corresponds to Step 5-8 in Algorithm 3):

$$\begin{aligned}&\mathrm{TimeBKZ}(n, \beta^{start} \overset{\beta^{alg}}{\to} \beta^{goal}) = \\ &\sum_{t=1}^{\sharp tours} \sum_{i=1}^{n-1} \Big[\text{Time of processing local block } B_i \text{ with parameters } (\alpha, p) \Big].\end{aligned} \tag{3.18}$$

Because $\sharp tours$ is already given in Section 3.5.2, we consider the factor of time of processing local block B_i.

Algorithm 5 One BKZ tour of our implementation to process the local block. These lines correspond to Step 5-8 in Algorithm 3.

```
5-1: Compute the Gram-Schmidt lengths $\|\mathbf{b}_i^*\|$ and coefficients $\mu_{ij}$
  corresponding to the local block $B_i$ of blocksize $\beta' = \min(\beta, n-i+1)$
5-2: Set $(\alpha, p)$ for $B_i$ using the setting in Section 3.4.3;
6-1: Set near optimized pruning coefficients $(R_1, \ldots, R_\beta)$ for $(B_i, \alpha, p)$;
6-2: Preprocess $B_i$ by the simple version of progressive BKZ in Section 3.7.1;
6-3: if enumeration cost for $B_i$ computed using $(\alpha_p, R_1, \ldots, R_\beta)$ is large
   then optimize the bounding function;
7:   $\{\mathbf{v}_1, \ldots, \mathbf{v}_h\} \leftarrow$ (lattice enumeration for $B_i$ using $(\alpha_p, R_1, \ldots, R_\beta)$);
8-1: Construct the degenerated basis $(\mathbf{b}_1, \ldots, \mathbf{b}_{i-1}, \mathbf{v}_{i_1}, \ldots, \mathbf{v}_{i_g}, \mathbf{b}_i, \ldots, \mathbf{b}_{i+\beta'-1})$;
8-2: Apply the LLL algorithm to the basis $(\mathbf{b}_1, \ldots, \mathbf{b}_{i-1}, \mathbf{v}_{i_1}, \ldots, \mathbf{v}_{i_g}, \mathbf{b}_i, \ldots, \mathbf{b}_{i+\beta'-1})$
   and erase the zero vectors.
```

For the details of analysis, we introduce a pseudo-code of our implementation in Algorithm 5. We decompose the running time over the internal summation as follows.

$$\begin{aligned}&\sum_{i=1}^{n-1}(\text{Time of processing local block } B_i \text{ with parameters } (\alpha,p)) = \\ &Time_{GS} + Time_{Optimize} + Time_{Preprocess} + Time_{Enum} + Time_{LLL} + misc,\end{aligned} \tag{3.19}$$

where $Time_{GS}$ is for Step 5-1, $Time_{Preprocess}$ is for Step 6-2, $Time_{Optimize}$ is for 6-3, $Time_{Enum}$ is for Step 7, and $Time_{LLL}$ is the time for Step 8-2. Note that the miscellaneous part is the time for all the other steps including the memory allocation and vector insertion which are negligible.

In the rest of this section, we introduce our implementation[1] and give a rough estimating formula of computing times with some order notations. (Section 3.7.1 and 3.7.2.) Then, we fix the rational coefficients by using the experimental results. Although some of implementing techniques are folklore or trivial, we give them for the completeness of the paper.

How to construct the degenerated basis in Step 8-1: Suppose we have a set of vectors $\mathbf{v}_1,\dots,\mathbf{v}_h$ found in Step 7. First, compute the projections of $\mathbf{v}_i$ onto $\mathbf{b}_1,\dots,\mathbf{b}_{i-1}$ and let $\mathbf{v}_{i_1}$ be the vector with shortest projection. After choosing the gth vector, $\mathbf{v}_{i_g}$, the next vector $\mathbf{v}_{i_{g+1}}$ is selected as follows. Compute the projections $\|\pi'(\mathbf{v}_i)\|$ of $\mathbf{v}_i$ onto $\mathbf{b}_1,\dots,\mathbf{b}_{i-1},\mathbf{v}_{i_1},\dots,\mathbf{v}_{i_g}$. If there exists i such that $0 < \|\pi'(\mathbf{v}_i)\| < \|\mathbf{b}^*_{i+g-1}\|$, then $\mathbf{v}_{i_{g+1}}$ is $\mathbf{v}_i$ that minimizes $\|\pi'(\mathbf{v}_i)\|$; otherwise, stop this process and output the degenerated basis $(\mathbf{b}_1,\dots,\mathbf{b}_{i-1},\mathbf{v}_{i_1},\dots,\mathbf{v}_{i_g},\mathbf{b}_i,\dots,\mathbf{b}_{i+\beta'-1})$.

[1] We published our implementation of our BKZ on the NICT webpage. Please go to `pbkz-info@ml.nict.go.jp` for the details.

3.7.1 Implementation and Time Estimation of Step 6 and 7: $Time_{Optimize} + Time_{Preprocess} + Time_{Enum}$

We give the details of our implementation from Step 6 to 7 in Algorithm 5. The goal of this section is to justify

$$\begin{aligned} &Time_{Optimize} + Time_{Preprocess} + Time_{Enum} \\ &= W \cdot \beta \cdot \sum_{i=1}^{n-1} \text{Sim-ENUMCost}(\ell'_i, \ldots, \ell'_{i+\beta-1}; \alpha, p) \end{aligned} \tag{3.20}$$

by using a constant W which we will determine in Section 3.7.4. Here, $\ell'_i, \ldots, \ell'_{i+\beta-1}$ is a part of output of $\text{Sim-GS-lengths}(n, \beta)$ or its updated values by the simulator in Section 3.5.1.

Computing bounding coefficients: In Step 6-1, we use Aono's precomputing technique [8] to generate the bounding coefficients $R_1, \ldots, R_\beta$ for pruning in the enumeration algorithm in Section 3.2.1. We fix this bounding function to predict the enumeration cost in the preprocessing step 6-2 (see the next paragraph). After preprocessing, in Step 6-3, we search better bounding coefficients if the expected number of enumeration searching nodes is larger than 10^8, which corresponds to a few seconds in a single thread. The procedure for finding a better bounding coefficients is the simple algorithm that considers random perturbations of $(R_1, \ldots, R_\beta)$ as the strategy in [25].

$Time_{Optimize}$ is the sum of the computing time in Step 6-1 and 6-3. It is significantly smaller than the cost of lattice vector enumeration. In small blocksizes, Step 6-1 can be done in about 100 milliseconds and Step 6-3 is skipped. Thus, $Time_{Optimize} \approx 0$. Moreover, since the precomputing technique outputs $R_1, \ldots, R_\beta$ as a function of the dimension β, target probability and target GSA constant, we can reuse them in implementation. Thus, the computing times of these steps are very small. Note that the target GSA constant is computed from the line fitting by the least square method to the points $(i, \log_2 \|\mathbf{b}_i^*\|)$ when the GS-lengths $(\|\mathbf{b}_i^*\|, \ldots, \|\mathbf{b}_{i+\beta-1}^*\|)$ is given.

On the other hand, for large blocksizes (larger than about 80), the time for lattice vector enumeration is much larger than that of optimization in Step 6-3. Therefore, we can assume $Time_{Optimize} \ll Time_{Enum}$ in the both situations.

Implementation of preprocess step: In Step 6-2, we use our progressive BKZ algorithm proposed in this paper with small blocksizes. It starts with the blocksize $\bar{\beta} = 15$ and increases $\bar{\beta}$ one by one when $\mathrm{FEC}(B_i) < \text{Sim-FEC}(\beta, \bar{\beta})$.

At the end of each tour, we compute $\mathrm{ENUMCost}(B_i; \alpha, p)/$`persec` as the estimation for the time of main enumeration. Here, $\mathrm{ENUMCost}(B_i; \alpha, p)$ is estimated by using near optimized coefficients generated in Step 6-1. We note that if the bounding coefficients $R_1, \ldots, R_\beta$ are fixed, the volume factors in the equation (3.1) are also fixed. Thus, computing the table of volume factors V_k, we can easily compute the expected number of processed nodes as

$$N = \frac{1}{2} \sum_{k=1}^{\beta} \frac{V_k}{\prod_{j=\beta-k+1}^{\beta} \|\mathbf{b}_{i+j-1}^*\|}.$$

during the preprocessing.

`persec` is the number of processed nodes in lattice vector enumeration in one second, which can be determined from our preliminary benchmark. It is about $6.0 \cdot 10^7$ at the maximum in a single threaded implementation. On the other hand, it slows down to about $\sharp threads \times 3.0 \cdot 10^7$ when we use the multi-threaded programming. In our implementation, we use 12 threads which can process about $3.0 \cdot 10^8$ nodes in one second. During the preprocessing, we obtain several lattice bases in the ends of tours. We keep the basis that takes minimum $\mathrm{ENUMCost}(B_i; \alpha, p)$ among them and also keep its minimized cost. The preprocessing subroutine terminates when the elapsed (wallclock) time exceeds the kept minimum cost in seconds scaled by the benchmark result. We use the minimum pair $(B, cost)$ for the preprocessing output.

Because our preprocessing subroutine works in a single thread, $Time_{Preprocess}$ is proportional to the recorded minimum cost. While the actual time for the enumeration decreases by the optimizing bounding coefficients, we do not consider it and assume that $Time_{Preprocess} = A_{Preprocess} \cdot Time_{Enum}$ by a constant $A_{Preprocess}$.

Implementation of enumeration step: In Step 7, we implement our modified version of the lattice vector enumeration subroutine in the NTL library. In order to speed up, we use `double` type to keep the Gram-Schmidt coefficients and lengths during the enumeration while the original version uses `quad_float`. In addition, we use assembly codes optimized for a latest CPU.

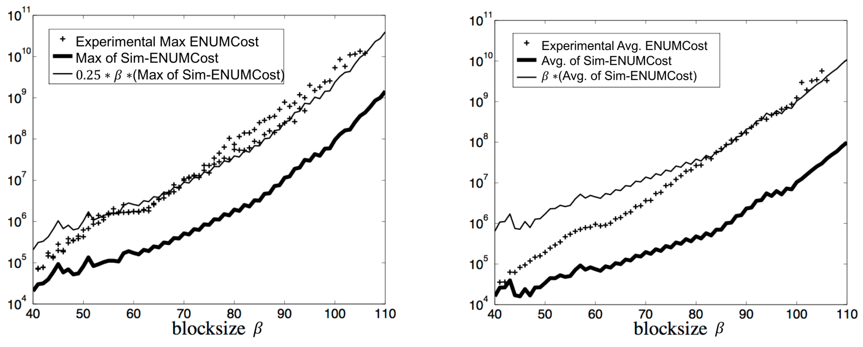

FIGURE 3.6: Maximum (left)/Average (right) of the number of nodes during the first tour at blocksize β

In the lattice enumeration subroutine, we find many vectors whose projective lengths are small, although their non-projective lengths are not small enough. In our implementation, we store the first $h = 16$ vectors ordered in the lexicographic order of the pair $(\beta - k, \|\pi_{\beta-k}(\mathbf{v})\|)$ satisfying $\|\pi_{\beta-k}(\mathbf{v})\| \leq R_{k+1}$. After the enumeration, the output is the stored vectors $\mathbf{v}_1, \ldots, \mathbf{v}_h$.

Our model and cost estimation of the enumeration step: To estimate $Time_{Enum}$, we performed experiments to compare the numbers of processed nodes and the simulated values.

Figure 3.6 shows the maximum and average of the number of processed nodes during the first tour of our progressive BKZ of blocksize β using random lattices of 300 dimensions. The symbol "+" indicates the actual number of nodes during the enumeration subroutine, while the bold curve is the maximum and average of $\text{Sim-ENUMCost}(\ell_i, \ldots, \ell_{i+\beta-1}; \alpha, p)$ for $i = 1, \ldots, n-\beta+1$, where $(\ell_1, \ldots, \ell_n)$ is the output of simulator $\text{Sim-GS-lengths}(n, \beta-1)$ in Section 3.5.1 and (α, p) is from Section 3.4.3.

As we can see in Figure 3.6, the numbers of processed nodes in our experiment are larger than the simulated numbers. This phenomenon is mainly caused by the basis updating, i.e., the vector inserting process in Step 8 in Algorithm 5. By inserting found vectors, the GS-lengths are changed and the corresponding enumeration cost is increased.

From the above experiments, we find the maximum of actual number of nodes is about 0.25β times the maximum of $\text{Sim-ENUMCost}(\ell_i, \ldots, \ell_{i+\beta-1}; \alpha, p)$ (See the left-hand side of Figure 3.6). A similar proportional relation is found in the average number of nodes (the right-hand side of the figure). Therefore, we assume the actual number of processed nodes during one BKZ tour for the basis whose GS-lengths are $(\ell_1, \ldots, \ell_n)$ is

$$A_{Enum} \cdot \beta \cdot \sum_{i=1}^{n-1} \text{Sim-ENUMCost}(\ell_i, \ldots, \ell_{i+\beta'-1}; \alpha, p),$$

where A_{Enum} is a rational constant. Thus, $Time_{Enum}$ in seconds is this value divided by `persec` (the number of processed nodes in one second in a single thread).

Total computing time in seconds in Step 6 and 7: Summarizing the above argument, we have

$$\begin{aligned} & Time_{Optimize} + Time_{Preprocess} + Time_{Enum} \ \ \text{[sec.]} \\ = & \frac{(A_{Preprocess} + 1) \cdot A_{Enum} \cdot \beta \cdot \sum_{i=1}^{n-1} \text{Sim-ENUMCost}(\ell'_i, \ldots, \ell'_{i+\beta'-1}; \alpha, p)}{\texttt{persec}}. \end{aligned}$$

Hence, letting $W = (A_{Preprocess} + 1) \cdot A_{Enum} \cdot /\texttt{persec}$, we get the equation (3.20). Note that we regard $A_{Optimize} = 0$.

3.7.2 Estimating Time of Step 5-1 and 8-2: $Time_{GS} + Time_{LLL}$ when the lattice dimension is small

In this section, we introduce our implementation to compute the GS-variables (i.e., the Gram-Schmidt lengths $\|\mathbf{b}_i^*\|$ and the coefficients μ_{ij}) used from Step 5-2 to 6-3. Then we show how to update these values in Step 8-2. In our implementation, we use the different precision types to treat GS-variables. If the lattice dimension n is smaller than 400, we compute and keep the GS-variables by using the `quad_float` type variables. Otherwise, we compute and keep them by using `RR` types which will be explained the detail in the next subsection.

In small dimensions $n < 400$, we keep GS-variables in the `quad_float` type and compute them directly. In short, Step 5-1 merely consists of copying the necessary parts $\|\mathbf{b}_{i'}^*\|$ and $\mu_{i'j'}$ for $i' \in \{i, \ldots, i + \beta' - 1\}$ and $j' \in [i']$ that corresponds to the local block B_i.

In Step 8-2, we apply the LLL algorithm to the degenerated basis consisting of the $i + \beta' - 1 + g$ vectors. Since we can assume that the first part of basis $(\mathbf{b}_1, \ldots, \mathbf{b}_{i-1})$ is

LLL reduced, the number of swaps can be approximated by that of the LLL algorithm for the basis consists of $\beta' + g$ vectors $(\pi_i(\mathbf{v}_{i_1}, \ldots, \mathbf{v}_{i_g}, \mathbf{b}_i, \ldots, \mathbf{b}_{i+\beta'-1}))$.

Hence, from the standard analysis of LLL [64], the number of swaps is $\Theta((\beta' + g)^2) = \Theta(\beta^2)$, and each swap requires the procedure for updating the Gram-Schmidt coefficients that takes $\Theta(n^2)$ floating point operations when the basis vectors are given by $m = O(n)$ dimensional vectors [2].

Thus, the required number of floating point operations in Step 8-2 is $\Theta(\beta^2 n^2)$, and the total computational time in seconds in one BKZ tour is

$$Time_{GS} + Time_{LLL} = \sum_{i=1}^{n-1} \Theta(n^2 \cdot \beta^2) = A_1 \cdot \beta^2 n^3 [sec.] \qquad (3.21)$$

with a rational constant A_1.

In this standard implementation using `quad_float` type, i.e., `LLL_QP` subroutine in NTL, about 400 dimension is the limit of stable computing in practice and the loss-of-precision error occurs in larger n. If we simply replace `quad_float` with another high precision type such as `RR` to avoid this error, the computing time must increase significantly. Several implementing techniques to reduce the cost of LLL have been proposed. One idea that we employ is to extract small local block by using small precision variables as in the next subsection.

3.7.3 Estimating Time of Step 5-1 and 8-2: $Time_{GS} + Time_{LLL}$ when the lattice dimension is large

To compute the LLL algorithm correctly, we need to compute the GS-variables in a sufficient accuracy. A naive method using a high precision floating point variables takes a large cost for large dimensional lattices, e.g., `LLL_RR` function in the NTL library. To decrease such costs, we introduce a heuristic algorithm using two types of floating point numbers that is used after the vector insertion in Step 8. Although it is a folklore among the programmers, we precisely define it to analyze the computational cost.

[2] If we treat a larger dimension $m \gg n$, we need to consider an additional term $O(mn)$ of computational cost.

We use the notations ($\mathtt{mu}_{i,j}^{\mathtt{type}}$,$\mathtt{c}_i^{\mathtt{type}}$) to denote the variables μ_{ij} and $\|\mathbf{b}_i^*\|$ using $\mathtt{type} \in \{\mathtt{qf}, \mathtt{RR}\}$ by which we can treat floating point numbers. Here, `qf` is the shortened form of `quad_float`.

In our implementation, we use ($\mathtt{mu}_{i,j}^{\mathtt{RR}}$,$\mathtt{c}_i^{\mathtt{RR}}$) to store the all GS-variables, and we also use ($\mathtt{mu}_{i,j}^{\mathtt{qf}}$,$\mathtt{c}_i^{\mathtt{qf}}$) as a "temporary cache" that stores the GS-variables of a projective sublattice to process local blocks as shown in Algorithm 5. Hence, a significant amount of basis updating (Step 8-2 in Algorithm 5) can be done efficiently within the cache variables with small precision, and the update of ($\mathtt{mu}_{i,j}^{\mathtt{RR}}$,$\mathtt{c}_i^{\mathtt{RR}}$) does not occur frequently.

We can assume the GS-variables ($\mathtt{mu}_{i,j}^{\mathtt{RR}}$,$\mathtt{c}_i^{\mathtt{RR}}$) of the basis are computed at the start point of Step 5-1, since they are computed in the LLL algorithm in Step 1 in Algorithm 3, or computed in the end of the previous loop.

Implementation: In our experiments, we use `RR` as the `RR` type with the 320 bit precision of the NTL library. The temporary cache is stored as a $H \times H$ matrix where we set $H = 250$.

Our implementation is described as follows. In the first step of our BKZ algorithm (Step 1 in Algorithm 4), we compute the GS-variables of the LLL-reduced basis and store them to ($\mathtt{mu}_{i,j}^{\mathtt{RR}}$,$\mathtt{c}_i^{\mathtt{RR}}$). From these high precision variables, we compute the cached values ($\mathtt{mu}_{i,j}^{\mathtt{qf}}$,$\mathtt{c}_i^{\mathtt{qf}}$) of size H starting from index I by constructing a new basis defined by the following matrix

$$B_{cache} = \begin{bmatrix} \|\mathbf{b}_I^*\| & & & \\ \|\mathbf{b}_{I+1}^*\| \cdot \mu_{I+1,I} & \|\mathbf{b}_{I+1}^*\| & & \\ \vdots & & \ddots & \\ \|\mathbf{b}_{I+H-1}^*\| \cdot \mu_{I+H-1,I} & \cdots & \cdots & \|\mathbf{b}_{I+H-1}^*\| \end{bmatrix}.$$

Here, the corresponding GS-variables are merely the copies of ($\mathtt{mu}_{i,j}^{\mathtt{RR}}$,$\mathtt{c}_i^{\mathtt{RR}}$). This temporary cache contains the GS-variables of the index from I to $I + H - 1$. We also keep the information on how the basis matrix is changed by processing the local block (Step 6-2 to 8-2 in Algorithm 5). The information is a unimodular matrix U that generates the basis B_{cache2} after processing and the basis B_{cache} before processing: $B_{cache2} = U \cdot B_{cache}$. (We can directly obtain U at Step 8-2, before shifting to the next cache.)

In Step 5-1, we check whether the indexes $[i : i + \beta - 1]$ of local block are a subset of the cache indexes $[I : I + H - 1]$. Here we recall that the notation $[I : J]$ is the set of integers between I and J. If these are not contained, we update the entire basis by multiplying the unimodular matrix U to the part of the basis $(\mathbf{b}_I, \dots, \mathbf{b}_{I+H-1})$, and then apply the LLL algorithm to $(\mathbf{b}_1, \dots, \mathbf{b}_n)$. After applying the LLL algorithm, the updated ($\mathtt{mu}^{\mathtt{RR}}_{i,j}$,$\mathtt{c}^{\mathtt{RR}}_i$) is computed. To process the local block, we once again compute the cache ($\mathtt{mu}^{\mathtt{qf}}_{i,j}$,$\mathtt{c}^{\mathtt{qf}}_i$) with the starting index $I = i$ and size $H = \min(250, n - I + 1)$. By this process, we can assume $[i : i + \beta - 1] \subset [I : I + H - 1]$ at Step 5-2 and the remaining part can be computed by using the cached variables.

Computational time: We estimate the number of floating point operations for `quad_float` and `RR` variables. First, to estimate the cost of computation in `RR` variables, we consider when the entire basis is updated. Since the index starts at $i = 1$ in Algorithm 5, the index range of the cache at this time is $[1 : H]$. By the fact that the blocksize is β, the updating process at Step 5-1 is occurred at the index $i = 2 + H - \beta$ and the new cache is from the indexes $[2 + H - \beta : 2H - \beta + 1]$. By the same argument, the jth updating is in the index $j + (H - \beta)(j - 1)$, and the corresponding indexes are $[j + (H - \beta)(j - 1) : \min(jH - (j - 1)(\beta - 1), n)]$. Hence, the number T of the entire updating in one BKZ tour is the minimum integer T so that $TH - (T - 1)(\beta - 1) > n$ and we have $T \approx (n - \beta)/(H - \beta)$.

In Step 5-1, we need to compute the multiplication by the unimodular matrix U and the LLL reduction in the large precision. We assume that the computational cost of the LLL is approximated by the time for computing the GS-variables in the caches because the updated basis is nearly LLL reduced. It is $\Theta(n^2 \cdot H)$ because $\Theta(n^2)$ floating point operations are required to compute one $\mathbf{b}^*_i$. The number of multiplication operations by U is $O(n \cdot H^2)$. Thus, the cost at Step 5-1 is $\Theta(n^2 \cdot H) + O(n \cdot H^2) = \Theta(n^2 \cdot H)$, and in total $\Theta(T \cdot n^2 \cdot H)$ throughout one BKZ tour.

On the other hand, the cost to update the local temporary cache in `quad_float` variables is $\Theta(\beta^2 H^2)$. We neglect this in our model for the large dimensions because the operations in `RR` are significantly heavier than that in `quad_float` from the preliminary experiments.

Therefore the total cost throughout one BKZ tour is as follows:

$$Time_{GS} + Time_{LLL} = \Theta(T \cdot n^2 \cdot H) = A_2 \cdot \frac{n-\beta}{H-\beta} \cdot n^2 H \text{ [sec.]}. \tag{3.22}$$

The rational constant A_2 will be fixed by the computer experiments.

3.7.4 Experimental Coefficient Fitting

Substituting equations (3.20), (3.21) and (3.22) to equation (3.19), we obtain our formulas for estimating the computational time of the progressive BKZ algorithm. For small dimensions (< 400) using only `quad_float` type of computing the GS-variables, the estimated computational time for finding a BKZ-β^{goal} reduced basis is as follows:

$$\begin{aligned} &Time_{\textit{Sim-small}}(dim, \beta, A_1, W_1) = \\ &\sum_{\beta^{start}}^{\beta^{goal}} \sum_{t=1}^{\sharp tours} \left[A_1 \cdot \beta^2 n^3 + W_1 \cdot \beta \sum_{i=1}^{n-1} \text{ENUMCost}(B_i; \alpha, p) \right] \text{[sec.]}. \end{aligned} \tag{3.23}$$

The computational time for the large dimensions is as follows:

$$\begin{aligned} &Time_{\textit{Sim-large}}(dim, \beta, A_2, W_2) = \\ &\sum_{\beta^{start}}^{\beta_{goal}} \sum_{t=1}^{\sharp tours} \left[A_2 \cdot \frac{n-\beta}{H-\beta} \cdot Hn^2 + W_2 \cdot \beta \sum_{i=1}^{n-1} \text{ENUMCost}(B_i; \alpha, p) \right] \text{[sec.]}. \end{aligned} \tag{3.24}$$

In this section, we conduct the computer experiments with the simple blocksize strategy:

$$2 \overset{20}{\to} 20 \overset{21}{\to} 21 \overset{22}{\to} 22 \overset{23}{\to} 23 \overset{24}{\to} 24 \overset{25}{\to} \cdots$$

and then we estimate the undefined variables W_1, W_2, A_1 and A_2 by the experimental computing time after BKZ-55, i.e., $\beta^{start} = 55$.

Generating method of test lattice bases: The input bases are the Goldstein-Mayer type random lattices generated by the SVP Challenge problem generator. The instance of SVP Challenge problem has three parameters: lattice dimension n, random seed s, and the bit-length parameter b. The determinant of the generated lattice is equal to a prime $p \approx 2^{bn}$, where the default value of b is 10 to generate the published challenge problems. However, if we use $b = 10$, then $\|\mathbf{b}_i^*\| = 1$ holds for some last indexes i after the LLL and BKZ with a small blocksize. When the basis has basis vectors of $\|\mathbf{b}_i^*\| = 1$ in last indexes, the FEC of basis does not indicate the reduction level correctly and fails to increase the blocksize in sharp timings. Indeed, this basis yields no sharp estimation from our preliminary experiments.

To prevent this, we take $b = \max(10, \lceil n/20 \rceil)$ for basis generation so that $\|\mathbf{b}_n^*\| > 1$ holds after the LLL reduction.

Finding the coefficients: The experiments are conducted using a server with two Intel Xeon CPU E5-2697@2.70GHz processors. The small squares in Figure 3.7 indicate the single-core seconds of finishing the BKZ-β algorithm. We let the CPU times of lattice reductions for computing a BKZ-β reduced basis of n dimensional basis be $Time_{Exp}(n, \beta)$. In other words, we exclude the time of memory allocation, generating pruning coefficients, computing $\text{Sim-GS-lengths}(n, \beta)$ and corresponding FEC. We also exclude the computing time of LLL reduced basis of an input lattice.

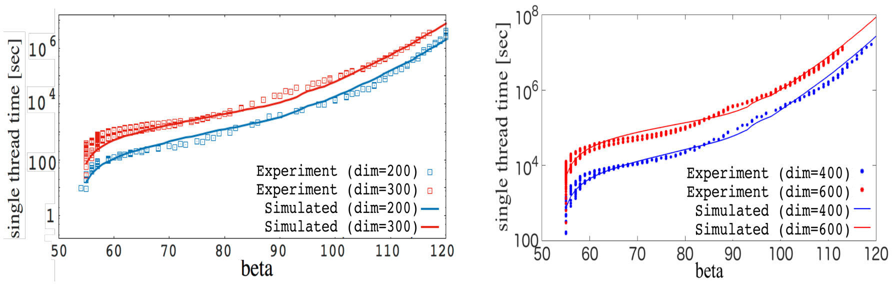

FIGURE 3.7: Result of our parameter fitting for cost estimation. Left Figure: implementation described in Section 3.7.2. Right Figure: implementation described in Section 3.7.3. In both graphs, experimental results are plotted by small squares and the simulating results are drawn in bold lines.

We find the suitable coefficients (A_1, W_1) by using the standard curve fitting method in semi-log scale, which minimize

$$\sum_{dim \in \{200,300\}} \sum_{\beta=55} \left| \log\left(T(dim, \beta, A_1, W_1)\right) - \log\left(Time_{Exp}(dim, \beta)\right) \right|^2 ,$$

where $T(dim, \beta, A_1, W_1) = Time_{Sim\text{-}large}(dim, \beta, A_1, W_1)$ in the small dimensional situation. For the large dimensional situation, we use the result of $dim \in \{600, 800\}$ to fix A_2 and W_2.

We find suitable coefficients

$$\begin{aligned} A_1 &= 10^{-10} \quad \text{and } W_1 = 1.5 \cdot 10^{-8} \\ A_2 &= 10^{-6} \quad \text{and } W_2 = 1.5 \cdot 10^{-8}. \end{aligned} \tag{3.25}$$

The fitting results are given in Figure 3.7. Using the equations (3.23) and (3.24) with the above coefficients (3.25), we can estimate the computing times of our progressive BKZ algorithm.

3.8 Pre/Post-Processing the Entire Basis

In this section, we consider an extended strategy that enhances the speed of our progressive BKZ by pre/post-processing the entire basis.

In preprocessing we first generate a number of randomized bases for input basis. Each basis is then reduced by using the proposed progressive BKZ algorithm. Finally we perform the enumeration algorithm for each reduced basis with some low probability in the post-processing. This strategy is essentially the same as the extreme pruning technique [37]. However, it is important to note that we do not generate a randomized basis inside the progressive BKZ. Our simulator for the proposed progressive BKZ is so precise that we can also estimate the speedup by the pre/post-processing using our simulator.

3.8.1 Algorithm for Finding Nearly Shortest Vectors

In the following, we construct an algorithm for finding a vector shorter than $\gamma \cdot \mathrm{GH}(L)$ with a reasonable probability using the strategy above, and we analyze the total computing time using our simulator for the BKZ algorithm.

Concretely, for given lattice basis B of dimension n, the preprocessing part generates M randomized bases $B_i = U_i B$ by multiplying unimodular matrices U_i for $i = 1, \ldots, M$. Next, we apply our progressive BKZ for finding the BKZ-β reduced basis. The cost to obtain the randomized reduced bases is estimated by $M \cdot (\mathrm{TimeRandomize}(n) +$

$\mathrm{TimeBKZ}(n,\beta))$. Here, $\mathrm{TimeRandomize}$ includes the cost of generating a random unimodular matrix and matrix multiplication, which is negligibly smaller than $\mathrm{TimeBKZ}$ in general. Thus we assume the computational cost for lattice reduction is $M \cdot \mathrm{TimeBKZ}(n,\beta)$.

Finally, in the post-processing part, we execute the standard enumeration algorithm with the searching radius parameter $\alpha = \gamma$ and probability parameter $p = 2 \cdot \gamma^{-n}/M$. As with the similar argument in Section 3.4.1, there exist about $\gamma^n/2$ short vector pairs in $\mathrm{Ball}_n(\gamma \cdot \mathrm{GH}(L))$. Therefore, the probability that one enumeration finds the desired vector is about $(\gamma^n/2) \cdot (2 \cdot \gamma^{-n}/M) = 1/M$ and the total probability of success is $1 - (1 - 1/M)^M \approx 0.632$.

Consequently, the total computing cost in our model is

$$M \cdot \left(\mathrm{TimeBKZ}(n,\beta) + \frac{\mathrm{ENUMCost}(B; \gamma, p = 2 \cdot \gamma^{-n}/M)}{6 \cdot 10^7} \right) \text{[sec.]}, \tag{3.26}$$

where $\mathrm{TimeBKZ}(n,\beta)$ and $\mathrm{ENUMCost}(B;\gamma,p)$ are defined by Section 3.6.1 and Section 3.2.1, respectively. We can optimize this total cost by finding the minimum of formula (3.26) over parameter (β, M). Here, note that the constant $6 \cdot 10^7$ comes from our best benchmark record of lattice enumeration. In Table 3.4, we provide the detailed simulating result with setting $\gamma = 1.05$ to analyze the hardness of the Darmstadt SVP Challenge in several dimensions. A comparison with previous works are given in Section 3.9 (See the line **C** in Figure 3.8).

3.8.2 Lower Bound of the Cost by an Idealized Algorithm

Here we discuss the lower bound of the total computing cost of the proposed progressive BKZ algorithm (or other reduction algorithm) with the pre/post-processing.

The total cost is estimated by the sum of the computational time for the randomization, the progressive BKZ algorithm, and the enumeration algorithm by the following extremely idealized situations. Note that we believe that they are beyond the most powerful cryptanalysis which we can achieve in the future, and thus we say that this is the lower bound in our model.

(a)The cost for the randomization becomes negligibly small. The algorithm for randomizing the basis would not only be the method of multiplying random unimodular bases, and we could find an ideal randomization at a negligibly small cost. Thus, $\mathrm{TimeRandomize}(n) = 0$.

(b)The cost for the progressive BKZ algorithm does not become lower than that of computing the Gram-Schmidt lengths. Even though the progressive BKZ algorithm ideally improved, we always need the Gram-Schmidt basis computation used for the enumeration algorithm or the LLL algorithm. The computation of the Gram-Schmidt basis (even though the computation is performed in an approximation using floating point operations with a sufficient precision) includes $\Theta(n^3)$ floating point arithmetic operations via the Cholesky factorization algorithm (See, for example [65, Chapter 5]). A modern CPU can perform a floating point operation in one clock cycle, and it can work at about 4.0GHz. Thus, we assume that the lower bound of the time in seconds is $(4.0 \cdot 10^9)^{-1} \cdot n^3$.

(c)The reduced basis obtained by the progressive BKZ (or other reduction algorithm) becomes ideally reduced. We define *the simulated γ-approximate HKZ basis $B_{\gamma\text{-}HKZ}$* by a basis satisfying

$$\|\mathbf{b}_i^*\| = \tau_{n-i+1}\mathrm{GH}(L_{[i:n]}) \text{ for } i = 2, \ldots, n \text{ and } \|\mathbf{b}_1\| = \gamma \cdot \mathrm{GH}(L).$$

For any fixed γ and p, we assume this basis minimizes the cost for enumeration over any basis satisfying $\|\mathbf{b}_1\| \geq \gamma \cdot \mathrm{GH}(L)$.

Therefore, the lower bound of the total cost of the idealized algorithm in seconds is given by

$$\min_{M \in \mathbb{N}} M \cdot \left((4.0 \cdot 10^9)^{-1} \cdot n^3 + \frac{ENUMCost(B_{\gamma\text{-}HKZ}; \alpha, p/M)}{6 \cdot 10^7} \right). \tag{3.27}$$

Setting $\gamma = 1.05$, we analyze the lower bound cost to enter the SVP Challenge. (See the line **D** in Figure 3.8).

TABLE 3.4: The cost of solving SVP Challenge using our optimal blocksize strategy

dim.	M	$\log_2$(Time [sec.])	optimal blocksize strategy in Section 3.6
100	5	10.8	$LLL \overset{23}{\to} 20 \overset{34}{\to} 32 \overset{40}{\to} 37 \overset{43}{\to} 41 \overset{48}{\to} 47 \overset{50}{\to} 49 \overset{59}{\to} 58 \overset{70}{\to} 69 \overset{80}{\to} 79$
110	9	15.5	$LLL \overset{23}{\to} 20 \overset{34}{\to} 31 \overset{40}{\to} 38 \overset{47}{\to} 43 \overset{48}{\to} 47 \overset{51}{\to} 49 \overset{61}{\to} 59 \overset{71}{\to} 70 \overset{80}{\to} 79 \overset{89}{\to} 89 \overset{96}{\to} 96$
120	9	20.3	$LLL \overset{23}{\to} 20 \overset{34}{\to} 29 \overset{40}{\to} 37 \overset{48}{\to} 47 \overset{49}{\to} 48 \overset{54}{\to} 52 \overset{62}{\to} 60 \overset{71}{\to} 70 \overset{80}{\to} 80 \overset{88}{\to} 88 \overset{95}{\to} 95 \overset{99}{\to} 99 \overset{103}{\to} 103 \overset{108}{\to} 107$
130	36	25.2	$LLL \overset{23}{\to} 20 \overset{34}{\to} 32 \overset{43}{\to} 40 \overset{48}{\to} 47 \overset{50}{\to} 48 \overset{54}{\to} 51 \overset{57}{\to} 55 \overset{64}{\to} 62 \overset{71}{\to} 70 \overset{77}{\to} 77 \overset{86}{\to} 86 \overset{93}{\to} 93 \overset{97}{\to} 97 \overset{101}{\to} 101 \overset{105}{\to} 105 \overset{110}{\to} 110 \overset{116}{\to} 115$
140	81	30.3	$LLL \overset{23}{\to} 20 \overset{34}{\to} 30 \overset{40}{\to} 36 \overset{48}{\to} 47 \overset{52}{\to} 50 \overset{57}{\to} 54 \overset{64}{\to} 61 \overset{70}{\to} 68 \overset{78}{\to} 77 \overset{86}{\to} 86 \overset{93}{\to} 93 \overset{97}{\to} 97 \overset{100}{\to} 100 \overset{103}{\to} 103 \overset{106}{\to} 106 \overset{110}{\to} 110 \overset{114}{\to} 114 \overset{118}{\to} 117 \overset{123}{\to} 122$
150	156	35.7	$LLL \overset{23}{\to} 20 \overset{34}{\to} 29 \overset{40}{\to} 37 \overset{47}{\to} 44 \overset{49}{\to} 48 \overset{56}{\to} 54 \overset{64}{\to} 60 \overset{67}{\to} 65 \overset{74}{\to} 72 \overset{80}{\to} 79 \overset{86}{\to} 86 \overset{93}{\to} 93 \overset{95}{\to} 95 \overset{99}{\to} 99 \overset{101}{\to} 101 \overset{103}{\to} 103 \overset{106}{\to} 106 \overset{109}{\to} 109 \overset{113}{\to} 113 \overset{117}{\to} 117 \overset{121}{\to} 121 \overset{124}{\to} 123 \overset{128}{\to} 127 \overset{132}{\to} 130$
160	327	41.2	$LLL \overset{23}{\to} 20 \overset{34}{\to} 29 \overset{40}{\to} 36 \overset{48}{\to} 47 \overset{52}{\to} 49 \overset{59}{\to} 57 \overset{64}{\to} 61 \overset{69}{\to} 66 \overset{73}{\to} 71 \overset{80}{\to} 78 \overset{86}{\to} 85 \overset{93}{\to} 93 \overset{95}{\to} 95 \overset{99}{\to} 99 \overset{101}{\to} 101 \overset{104}{\to} 104 \overset{107}{\to} 107 \overset{110}{\to} 110 \overset{113}{\to} 113 \overset{116}{\to} 116 \overset{119}{\to} 119 \overset{122}{\to} 122 \overset{126}{\to} 126 \overset{130}{\to} 129 \overset{133}{\to} 132 \overset{139}{\to} 137$
170	343	46.9	$LLL \overset{23}{\to} 20 \overset{34}{\to} 29 \overset{40}{\to} 37 \overset{48}{\to} 47 \overset{54}{\to} 52 \overset{62}{\to} 60 \overset{69}{\to} 65 \overset{73}{\to} 70 \overset{77}{\to} 75 \overset{83}{\to} 81 \overset{88}{\to} 87 \overset{93}{\to} 93 \overset{97}{\to} 97 \overset{100}{\to} 100 \overset{103}{\to} 103 \overset{106}{\to} 106 \overset{109}{\to} 109 \overset{112}{\to} 112 \overset{115}{\to} 115 \overset{118}{\to} 118 \overset{121}{\to} 121 \overset{124}{\to} 124 \overset{126}{\to} 126 \overset{129}{\to} 129 \overset{132}{\to} 131 \overset{134}{\to} 133 \overset{137}{\to} 136 \overset{139}{\to} 138 \overset{141}{\to} 139 \overset{145}{\to} 143 \overset{149}{\to} 146$
180	782	52.9	$LLL \overset{23}{\to} 20 \overset{34}{\to} 28 \overset{40}{\to} 36 \overset{48}{\to} 47 \overset{50}{\to} 48 \overset{59}{\to} 57 \overset{67}{\to} 65 \overset{74}{\to} 70 \overset{78}{\to} 75 \overset{82}{\to} 80 \overset{88}{\to} 86 \overset{93}{\to} 92 \overset{97}{\to} 97 \overset{101}{\to} 101 \overset{104}{\to} 104 \overset{107}{\to} 107 \overset{110}{\to} 110 \overset{113}{\to} 113 \overset{116}{\to} 116 \overset{119}{\to} 119 \overset{122}{\to} 122 \overset{124}{\to} 124 \overset{126}{\to} 126 \overset{129}{\to} 129 \overset{132}{\to} 132 \overset{134}{\to} 134 \overset{137}{\to} 136 \overset{139}{\to} 138 \overset{141}{\to} 140 \overset{144}{\to} 143 \overset{147}{\to} 145 \overset{152}{\to} 150 \overset{156}{\to} 153$
190	1431	59.0	$LLL \overset{23}{\to} 20 \overset{34}{\to} 28 \overset{40}{\to} 36 \overset{48}{\to} 47 \overset{54}{\to} 52 \overset{64}{\to} 62 \overset{72}{\to} 70 \overset{77}{\to} 74 \overset{81}{\to} 78 \overset{84}{\to} 82 \overset{89}{\to} 87 \overset{92}{\to} 91 \overset{93}{\to} 93 \overset{95}{\to} 95 \overset{99}{\to} 99 \overset{103}{\to} 103 \overset{107}{\to} 107 \overset{110}{\to} 110 \overset{113}{\to} 113 \overset{116}{\to} 116 \overset{119}{\to} 119 \overset{122}{\to} 122 \overset{124}{\to} 124 \overset{126}{\to} 126 \overset{129}{\to} 129 \overset{131}{\to} 131 \overset{133}{\to} 133 \overset{136}{\to} 136 \overset{138}{\to} 138 \overset{142}{\to} 141 \overset{145}{\to} 144 \overset{148}{\to} 147 \overset{150}{\to} 149 \overset{154}{\to} 152 \overset{157}{\to} 155 \overset{160}{\to} 157 \overset{163}{\to} 160$
200	3648	65.5	$LLL \overset{23}{\to} 20 \overset{34}{\to} 27 \overset{40}{\to} 35 \overset{48}{\to} 47 \overset{52}{\to} 50 \overset{60}{\to} 58 \overset{67}{\to} 66 \overset{76}{\to} 74 \overset{83}{\to} 82 \overset{88}{\to} 86 \overset{92}{\to} 90 \overset{95}{\to} 94 \overset{98}{\to} 98 \overset{101}{\to} 101 \overset{103}{\to} 102 \overset{104}{\to} 104 \overset{107}{\to} 107 \overset{110}{\to} 110 \overset{113}{\to} 113 \overset{116}{\to} 116 \overset{119}{\to} 119 \overset{122}{\to} 122 \overset{124}{\to} 124 \overset{126}{\to} 126 \overset{129}{\to} 129 \overset{131}{\to} 131 \overset{133}{\to} 133 \overset{136}{\to} 136 \overset{139}{\to} 139 \overset{142}{\to} 142 \overset{144}{\to} 144 \overset{148}{\to} 147 \overset{149}{\to} 148 \overset{152}{\to} 151 \overset{155}{\to} 154 \overset{156}{\to} 155 \overset{159}{\to} 157 \overset{163}{\to} 161 \overset{166}{\to} 163 \overset{169}{\to} 166$
210	11979	72.1	$LLL \overset{23}{\to} 20 \overset{34}{\to} 27 \overset{40}{\to} 35 \overset{48}{\to} 47 \overset{56}{\to} 54 \overset{65}{\to} 63 \overset{74}{\to} 73 \overset{81}{\to} 80 \overset{87}{\to} 84 \overset{89}{\to} 87 \overset{93}{\to} 93 \overset{98}{\to} 97 \overset{99}{\to} 99 \overset{101}{\to} 101 \overset{103}{\to} 103 \overset{105}{\to} 105 \overset{107}{\to} 107 \overset{110}{\to} 110 \overset{113}{\to} 113 \overset{116}{\to} 116 \overset{119}{\to} 119 \overset{122}{\to} 122 \overset{124}{\to} 124 \overset{126}{\to} 126 \overset{129}{\to} 129 \overset{131}{\to} 131 \overset{133}{\to} 133 \overset{136}{\to} 136 \overset{138}{\to} 138 \overset{141}{\to} 141 \overset{143}{\to} 143 \overset{146}{\to} 146 \overset{148}{\to} 148 \overset{150}{\to} 150 \overset{153}{\to} 152 \overset{156}{\to} 155 \overset{159}{\to} 158 \overset{161}{\to} 160 \overset{162}{\to} 161 \overset{165}{\to} 163 \overset{167}{\to} 165 \overset{170}{\to} 168 \overset{172}{\to} 169 \overset{174}{\to} 171 \overset{175}{\to} 172$
220	20770	78.9	$LLL \overset{23}{\to} 20 \overset{34}{\to} 27 \overset{40}{\to} 35 \overset{48}{\to} 47 \overset{52}{\to} 49 \overset{59}{\to} 57 \overset{68}{\to} 66 \overset{76}{\to} 75 \overset{82}{\to} 81 \overset{88}{\to} 87 \overset{93}{\to} 93 \overset{98}{\to} 97 \overset{101}{\to} 100 \overset{101}{\to} 101 \overset{103}{\to} 103 \overset{105}{\to} 105 \overset{107}{\to} 107 \overset{109}{\to} 109 \overset{111}{\to} 111 \overset{113}{\to} 113 \overset{116}{\to} 116 \overset{119}{\to} 119 \overset{122}{\to} 122 \overset{124}{\to} 124 \overset{126}{\to} 126 \overset{129}{\to} 129 \overset{131}{\to} 131 \overset{134}{\to} 134 \overset{136}{\to} 136 \overset{138}{\to} 138 \overset{141}{\to} 141 \overset{144}{\to} 144 \overset{148}{\to} 148 \overset{151}{\to} 151 \overset{153}{\to} 153 \overset{154}{\to} 154 \overset{155}{\to} 155 \overset{159}{\to} 158 \overset{164}{\to} 163 \overset{166}{\to} 165 \overset{167}{\to} 166 \overset{169}{\to} 167 \overset{171}{\to} 169 \overset{173}{\to} 171 \overset{175}{\to} 173 \overset{177}{\to} 174 \overset{181}{\to} 178$

3.9 Simulation Results for SVP Challenge and Comparison

In this section, we give our simulation results using our proposed progressive BKZ algorithm together with the pre/post-processing strategy in Section 3.8.1 for solving the Darmstadt SVP Challenge [28], which tries to find a vector shorter than $1.05 \cdot \mathrm{GH}(L)$ in the random lattice L of dimension n.

We also simulate the cost estimation of Lindner and Peikert [55] and that of Chen and Nguyen [24] in the same model. The summery of our simulation results and the latest records published in the SVP Challenge are given in Figure 3.8. The outlines of our estimations **A** to **D** in Figure 3.8 are given below.

From our simulation, the proposed progressive BKZ algorithm is about 50 times faster than BKZ 2.0 and about 100 times slower than the idealized algorithm that achieves the lower bound in our model of Section 3.8.2.

A: Lindner-Peikert's estimation [55]: From the experiments using the BKZ implementation in the NTL library [79], they estimated that the BKZ algorithm can find a short vector of length $\delta^n \det(L)^{1/n}$ in $2^{1.8/\log_2(\delta)-110}$ [sec.] in the n-dimensional lattice. The computing time of Lindner-Peikert's model becomes

$$Time_{\mathrm{LP}} = 2^{1.8/\log_2(\delta)-110} \text{ with } \delta = 1.05^{1/n} \cdot V_n(1)^{-1/n^2},$$

because this δ attains $1.05 \cdot \mathrm{GH}(L) = \delta^n \det(L)^{1/n}$.

B: Chen-Nguyen's BKZ 2.0 [24, 25]: We estimated the cost of BKZ 2.0 using the simulator in Section 3.3.2. Following the original paper [24], we assume that a blocksize is fixed and the estimation is the minimum of (3.4) over all possible pairs of the blocksize β and the number t of tours. Again we convert the number of nodes into the single-threaded time, we divide the number by $2 \cdot 10^7$.

C: Our estimation: We searched the minimum cost using the estimation (3.26) over M and β with setting $\gamma = 1.05$.

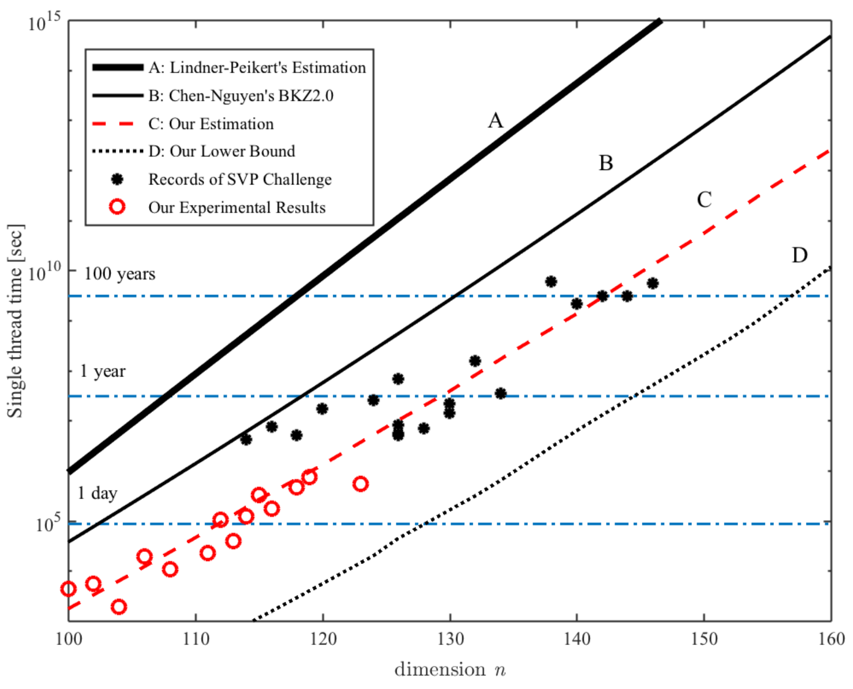

FIGURE 3.8: Semi-log graph of comparing cost in seconds. **A**: Lindner-Peikert estimation [55], **B**: Chen-Nguyen's BKZ 2.0 simulation (Sec. 3.3.2), **C**: Simulating estimation of our randomized BKZ-then-ENUM algorithm (Sec. 3.8.1), **D**: Lower bound in the randomized BKZ-then-ENUM strategy (Sec. 3.8.2). Records in the SVP Challenge [28] are indicated by the black circles "•", and our experimental results (Sec. 3.8.1) are indicated by the white circles "○".

D: Lower bound in our model: We searched the minimum cost using the estimation (3.27) over M with setting $\gamma = 1.05$.

Records of SVP Challenge: From the hall of fame in the SVP Challenge [28] and reporting paper [35], we listed up the records that contain the computing time with a single thread in Figure 3.8, as black circles "•". Moreover we performed experiments on our proposed progressive BKZ algorithm using the pre/post-processing strategy in Section 3.8.1 up to 123 dimensions which are also indicated by the white circles "○" in Figure 3.8.

3.10 Conclusions and Future Work

In this chapter, we proposed an improved progressive BKZ algorithm with optimized parameters and block-increasing strategy. We also gave a simulator that can precisely predict the Gram-Schmidt lengths computed using the proposed progressive BKZ. Thus, the total cost of the proposed progressive BKZ algorithm can be precisely evaluated by the sharp simulator. Simultaneously, we presented the efficient implementation of our proposed algorithm.

Moreover, we showed a comparison with other algorithms by simulating the cost of solving the instances from the Darmstadt SVP Challenge. Our progressive BKZ algorithm is about 50 times faster than the BKZ 2.0 proposed by Chen and Nguyen for solving the SVP Challenges up to 160 dimensions. Finally, we discussed a computational lower bound of the proposed progressive BKZ algorithm under certain ideal assumptions. These simulation results contribute to the estimation of the secure parameter sizes used in lattice based cryptography.

We outline some future works: (1) constructing a BKZ simulator without using our $\mathrm{ENUMCost}$, (2) adopting our simulator with other strategies such as BKZ-then-Sieve strategy for computing a short vector more efficiently, and (3) estimating the secure key length of lattice-based cryptosystems using the lower bound of the proposed progressive BKZ.

Chapter 4

Hardness Evaluation for Search LWE Problem using Progressive BKZ and its Simulator

In this chapter, we study the practical hardness of the LWE problem, by adapting Kannan's embedding technique, and using the improved progressive BKZ algorithm (pBKZ) introduced in Chapter 3. (In the rest of this thesis, we denote the progressive BKZ algorithm by *pBKZ* for short.) The LWE instances used in our experiments are sampled from Darmstadt LWE Challenge. From our experiments, we find that the algorithm can derive a better efficiency if the embedding factor M is closer to 1. We also give a preliminary analysis for the proper parameter settings as the number of LWE samples m should be used in the attack associate to the secret length n. Simultaneously we estimate the runtime of solving LWE cases using the BKZ simulator, whose preciseness is shown in Chapter 3. Moreover, our experimental results in LWE Challenge are close to the estimations. Especially for $n \geq 55$ and the fixed $\sigma/q = 0.005$, our implemented embedding technique with progressive BKZ is more efficient than Xu et al.'s implementation of the enumeration algorithm in [84][27]. Finally, we get the records of case $(70, 0.005)$ in Darmstadt LWE Challenge, using our extrapolated setting of m, which takes 32.73 single core hours. The work in this chapter was published in ICICS 2017 [82].

4.1 Introduction

Nowadays many post-quantum cryptographic schemes as fully homomorphic encryption and lattice-based signature schemes base their security on some lattice hard problems as the learning with errors (LWE) problem, short integer solution (SIS) and so on [69][60][40]. LWE problem was introduced by Regev in 2005 [69] and as an average-case lattice problem, LWE problem is proved as hard as certain worst-case lattice problems such as GapSVP and SIVP [69], which allows to build many provably secure lattice-based cryptographic schemes. However, for the practical application, it is indispensable to estimate the concrete parameters of LWE from sufficient experiments. In this work, we focus on the more practical lattice-based attack. At first, the LWE problem can be seen as a particular bounded distance decoding (BDD) instance on a q-ary lattice. For a given lattice and a target vector close to the lattice points in a reasonable bound, BDD is to find the closest lattice vector to the target.

There are two main methods to process the BDD instance. One is reducing the lattice basis first and searching the secret vector by Babai's NearestPlane [13] algorithm or its extensive variants [55][56]. Especially in [56], Liu and Nguyen intermingle the short error vector into an enumeration searching tree, which makes the attack more efficient. Xu et al.'s group solved some LWE Challenge instances using Liu-Nguyen's adapted enumeration technique [56] and they published this result at ACNS 2017 [84].

Another procedure is to reduce BDD to the unique-shortest vector problem (unique-SVP) by Kannan's embedding technique [49]. This procedure increases one more lattice dimension by adding the target vector and a so-called embedding factor M into the new basis. For parameters used in cryptography, with high probability, the error vector can be found in a component of the shortest vector in the constructed lattice. So there is a big gap between the shortest vector and the second shortest vector in the new lattice, which makes a lattice reduction algorithm or a searching algorithm find the shortest one more efficiently. Since both methods call the SVP oracle, their complexity grows exponentially with the dimension increasing.

Related works. Under the situation where one can get arbitrary number of samples, the hardness of LWE problem is related to three critical parameters: the length n of secret vector, the modulus q and the deviation σ of error vectors. Concerning with

Kannan's embedding technique [49], the theoretical discussion between the embedding factor t and the optimal number of required samples m was given in [4]. However they did not quantify the relevant runtime with these two critical parameters. [9] updated the parameters for LWE proposed in [55], using the improved computational time model $t(\delta) = 2^{1.8/\log(\delta)-130}$ where δ is the root Hermite factor. However, the time model is considered inaccurate and antiquated. [6] used several time models from fplll, enumeration and sieving, to evaluate the concrete hardness of LWE by invoking different algorithms such as BKW, SIS, decoding attack, etc. In Asiacrypt 2017, Albrecht et al. revisited the hardness of LWE using the so called "2016 estimate" and a BKZ time model adopting sieving as SVP oracle [5]. They also updated the parameters for some LWE-based cryptosystems using this computational lower bound. Some other theoretical analyses for the hardness of LWE are given as lattice-based attack [55][56], and BKW type attack [50], but rarely concrete parameters based on experiments were published. In order to assess the hardness of the LWE problem in practice, TU Darmstadt publishes a new platform "Darmstadt LWE Challenge" [21][27]. Indeed, we do the practical analysis of the parameter settings using LWE Challenge in this work.

Roadmap. Section 4.2 recalls the simulators for estimating the cost of BKZ algorithms. We introduce Kannan's embedding technique in Section 4.3. Our experimental results and preliminary analysis on the relevant parameters settings in Kannan's embedding technique are shown in Section 4.4. Finally we conclude in Section 4.5.

4.2 Cost Estimation of BKZ Algorithms Revisit

In this section, we will revisit two evaluating methods for BKZ cost: the pBKZ simulator (pBKZ simulator) we introduced in Chapter 3 and Martin Albrecht's estimator (M.A. estimator) [5]. The former one can give a practical estimation and we can get a lower bound of BKZ runtime using the latter one. Note that the only difference between "M.A. estimator" and "2016 estimate" is, the cost of sieving-BKZ is considered in the former one.

4.2.1 pBKZ Simulator

The **input** of the pBKZ simulator is (n, β), where n is the dimension and β is the target blocksize which means the pBKZ will terminate after the $\beta-$reduction was processed. The **output** is the total cost of the pBKZ. Note that the input parameter β can be replaced by the target GSA constant r, the enumeration subroutine search radius coefficient α or enumeration search success probability p, since we can compute the other three from one using the following the optimized equations (3.6) (3.9) and (3.10), in Section 3.4.

For $n \geq 100$ the cost (seconds, or "secs" for short in this thesis) of pBKZ is computed as

$$\begin{aligned} &\mathrm{TimeBKZ}(n, \beta_t) \\ &= \sum_{\beta=10}^{\beta_t} \sum_{t=1}^{\sharp tours} \Big[2.5 \cdot 10^{-4} \cdot \frac{n-\beta}{250-\beta} \cdot n^2 \\ &\qquad + 1.5 \cdot 10^{-8} \cdot \beta \cdot \sum_{i=1}^{n-1} \mathrm{ENUMCost}(B_i; \alpha, p) \Big]. \end{aligned} \tag{4.1}$$

Here $\mathrm{ENUMCost}(B_i; \alpha, p)$ is the number of enumeration search nodes in pBKZ simulator:

$$\begin{aligned} \mathrm{ENUMCost}(B_i; \alpha, p) &= p \cdot \frac{V_{\beta/2}(\alpha \cdot \mathrm{GH}(B_r))}{\prod_{i=\beta/2+1}^{\beta} \|\mathbf{b}_i^*\|_2} \\ &= 2\alpha^{-\beta/2} \cdot \frac{V_{\beta/2}(1) \cdot V_{\beta}(1)^{-1/2}}{r^{\beta^2/16}} \end{aligned}$$

Further details may be found in 3.4 and a reference implementation is freely available in [11].

4.2.2 M.A. Estimator

Before introducing M.A. estimator, we give simple introduction on the sieving algorithm. In 2001, Ajtai et al. proposed a sieving algorithm to solve SVP, which requires runtime of $2^{0.52n+o(n)}$ in n dimension lattice and simultaneously requires exponential storage of $2^{0.2n+o(n)}$ [2]. According to recent research results, for a n-dimensional lattice L and fixed blocksize β in BKZ, the runtime of sieving algorithm can be estimated in $2^{0.292\beta+o(\beta)}$ clock cycles for a β-dimensional subroutine [6], and totally BKZ-β costs $8n \cdot 2^{0.292\beta+16.4}$ operations [15]. We will also use this result to evaluate the security of our parameter choices in Chapter 5.

It is assumed that the BKZ-β reduced basis **B** of dimension n can give a short vector $\|\mathbf{b}_1\|_2 = \delta^n \cdot \det(B)$ in [23]. The root Hermite Factor (rHF) δ here is

$$\delta = (((\pi\beta)^{1/\beta}\beta)/(2\pi e))^{1/(2(\beta-1))}. \tag{4.2}$$

In Asiacrypt 2017 [5], Martin Albrecht et al. re-estimated the hardness of LWE problem using the so called “2016 estimate" from NewHope paper [7]: if the Gaussian Heuristic and the GSA hold for BKZ-β reduced basis and

$$\sqrt{\beta/n} \cdot \|(\mathbf{e}|1)\| \approx \sqrt{\beta}\sigma \leq \delta^{2\beta-n} \cdot \operatorname{vol}(L_{(\mathbf{A},q)})^{1/n}. \tag{4.3}$$

then error $\mathbf{e}$ can be found by BKZ-β with rHF δ. Here $(\mathbf{e}|1)$ means a vector composed by $\mathbf{e}$ and 1. In Martin Albrecht et al.'s estimator, they replace SVP oracle in BKZ by Gauss sieve algorithm whose cost is $(2^{0.292\beta+o(\beta)})$ [52]. They use the result of

$$\operatorname{Cost}(n, \beta) = 8n \cdot 2^{0.292\beta+16.4} \tag{4.4}$$

operations [15][3] as the cost of running BKZ-β in n-dimensional lattice. Note that they claim this runtime estimation can be adapted for $\beta > 90$. We call it "M.A. estimator" where the **input** is an LWE instance of parameter set (n, σ, q), the **output** is the target blocksize β and the runtime of BKZ algorithm to solve the given LWE instance, using formulas (5.2) (5.1) and (4.4).

4.3 Overview of Embedding Technique for Solving LWE Problem

In this section, we recall Kannan's embedding technique [49], and introduce the parameter settings in our experiments.

4.3.1 From LWE to BDD

The LWE problem can be reduced to BDD case as follows.

Input: a lattice $L = \{\mathbf{v} \in \mathbb{Z}_q^m | \mathbf{v} \equiv \mathbf{A}\mathbf{s} \pmod q\ ,\ \mathbf{s} \in \mathbb{Z}^n\}$ and a target vector $\mathbf{t}$ with bounded distance $\|\mathbf{e}\|$.

Output: a vector $\mathbf{v} \in L$ close to $\mathbf{t}$, and get $\mathbf{s}$ from $\mathbf{v} \equiv \mathbf{A}\mathbf{s}$ if succeeded.

In 2016, Xu et al. solved some instances of LWE Challenge by reducing LWE to BDD and using Liu-Nguyen's adapted enumeration algorithm, which can solve BDD directly with a considerable success probability. In this work we focus on solving BDD by embedding technique: further reduce BDD to unique-SVP [49]. The embedding attack is shown in Algorithm 6. We elaborate on the algorithm in the next section.

4.3.2 Solving LWE via the Embedding Technique

Algorithm 6 Kannan's embedding technique to solve LWE problem. [49]

```
Input: An LWE instance $(\mathbf{A}, \mathbf{b} \equiv \mathbf{A}\mathbf{s} + \mathbf{e} \pmod q) \in (\mathbb{Z}_q^{m\times n}, \mathbb{Z}_q^m)$.
Output: The secret vector $\mathbf{s} \in \mathbb{Z}_q^n$ and the short error vector $\mathbf{e} \in \mathbb{Z}_q^m$, s.t. $\mathbf{b} = \mathbf{A}\mathbf{s} + \mathbf{e}$.
1: Construct the basis $B$ of $q$-ary lattice
       $L_{(\mathbf{A},q)} = \{\mathbf{v} \in \mathbb{Z}_q^m \mid \mathbf{v} \equiv \mathbf{A}\mathbf{x} \pmod q, \mathbf{x} \in \mathbb{Z}^n\}$
   as $B = \begin{pmatrix} \mathbf{A}^T \\ \hline q\mathbf{I}_m \end{pmatrix} \in \mathbb{Z}^{(m+n)\times m}$; and compute the HNF of $B$ as
       $B_{\text{HNF}} = \begin{pmatrix} q\mathbf{I}_{m-n} & \mathbf{0} \\ \mathbf{A}'_{n\times(m-n)} & \mathbf{I}_n \end{pmatrix} \in \mathbb{Z}^{m\times m}$;
2: Reduce BDD to unique-SVP by rescaling $B_{\text{HNF}}$ to $B' = \begin{pmatrix} B_{\text{HNF}} & \mathbf{0} \\ \mathbf{b} & M \end{pmatrix} \in \mathbb{Z}^{(m+1)\times(m+1)}$;
3: Process $B'$ using lattice algorithm to derive a short vector $\mathbf{w}$ including the error vector $\mathbf{e}$;
4: Use $\mathbf{e}$ to compute the secret vector $\mathbf{s}$ by Gaussian elimination in $(\mathbf{b} - \mathbf{e}) = \mathbf{A}\mathbf{s}$.
```

Preprocessing. To solve a given LWE instance, necessary number of samples is considered. For instance, the Darmstadt LWE Challenge supplies the original basis $\mathbf{A}' \in \mathbb{Z}_q^{n^2 \times n}$ for each problem case, thus, a naive construction of matrices in Algorithm 6 requires a lattice reduction of a large number of matrices even for small LWE dimensions. Hence, we can choose m ($m \ll n^2$) vectors as a parameter to optimize the computational time, as from $(\mathbf{A}', \mathbf{b}') \in (\mathbb{Z}_q^{n^2 \times n}, \mathbb{Z}_q^{n^2})$ to $(\mathbf{A}, \mathbf{b}) \in (\mathbb{Z}_q^{m \times n}, \mathbb{Z}_q^m)$. We will discuss how to compute the optimal m in Section 4.3.4. Also during the random sampling, we should check the independence of vectors to make sure: (1) the correctness of the attack algorithm; (2) the volume of derived q-ary lattice is q^{m-n}, which will be used in section 4.3.4. We give explanations for each step in Algorithm 6.

Step 1. We follow the method in Section 2.7 to construct and compute the HNF basis B_{HNF} of q-ary lattice $L_{(\mathbf{A},q)} = \{\mathbf{v} \in \mathbb{Z}^m \mid \mathbf{v} \equiv \mathbf{A}\mathbf{x} \pmod q, \mathbf{x} \in \mathbb{Z}^n\}$.

Step 2. This step is the key point of embedding technique: expand the q-ary basis $B_{\mathrm{HNF}} \in \mathbb{Z}^{m \times n}$ by one dimension, and embed the target vector $\mathbf{b}$ and one embedding factor M into the new basis $B' \in \mathbb{Z}^{(m+1)\times(m+1)}$.

Step 3. At this step, we process the new basis B' by lattice algorithms. After the reduction, we get the error vector $\mathbf{e}$ from the output shortest vector $\mathbf{w}$, since $\mathbf{e} = \mathbf{b} - \mathbf{B}\mathbf{u}$ and $\mathbf{w} = B'\left(\frac{\mathbf{u}}{1}\right) = \left(\frac{\mathbf{e}}{M}\right)$ for some $\mathbf{u} \in \mathbb{Z}_q^m$. In our work, we use the progressive BKZ reduction algorithm [10].

Step 4. Simply get the secret vector $\mathbf{s}$ by Gauss elimination.

Theoretically, Algorithm 6 fails means no solution is given within a limited time T or the algorithm outputs a much shorter vector than the solution due to [4]. Here T is an input of our implementation. We set T large enough by preliminary experiments for each parameter set (n, q, α). We note that success rate of experiments are large enough, namely, $> 90\%$. Hence we claim that our definition of runtime is fair. However, the latter one occurs with an extremely low rate in general case (indeed not happened in our experiments). Hence, we define the "fail" in our work by the former one: Algorithm 6 fails means no solution is given within a limited time T. If Algorithm 6 fails, one should re-randomize the input samples or increase the number of samples m, then re-run the Algorithm 6. Now we explain four discussion points of the algorithm.

1) In the embedding procedure of **Step 3**, if the output vector $\mathbf{w}$ of the lattice algorithm satisfies

$$\|\mathbf{w}\| \leq \sqrt{\|\mathbf{e}\|^2 + M^2} \approx \left(\frac{\sqrt{2m}\sigma}{(Mq^{m-n})^{1/(m+1)}} \right)^{1/(m+1)} \tag{4.5}$$

with $\|\mathbf{e}\| \approx \sqrt{m}\sigma$, then the answer is correct with high probability.

2) There is a gap between the shortest vector and the linearly independent second shortest vector in $L'(B')$, namely we have to solve a unique-SVP in this lattice. The size of embedding factor M can affect the gap in some sense and we will discuss it in Section 4.3.3.

3) Since we do not know the exact value of $\|\mathbf{e}\|$, we can not terminate by condition (4.5). $\|\mathbf{w}\| \leq \sqrt{\|\mathbf{e}\|^2 + M^2}$ is the condition for a reduction or point searching algorithm to terminate in **Step 4**. However, during the update of lattice reduction of basis $(\mathbf{b}_1, \ldots, \mathbf{b}_n)$ in our progressive BKZ, we found that the root Hermite factor δ suddenly drops to a very small value from value around 1. We can set the algorithm to terminate when $\delta < 0.7$ for convenience.

4) There is a trade-off between the attack efficiency and success rate, depending on the dimension m of $L_{(\mathbf{A},q)}(\mathbf{A} \in \mathbb{Z}_q^{m\times n})$ and the embedding factor M of the sampled LWE instances in the embedding algorithm.

In this work, our goal is from experiments to get a preliminary analysis of the effect of m and M on the runtime for solving Darmstadt LWE Challenge instances.

4.3.3 How to Choose M at Step 2

The size of $\|\mathbf{e}\|$ and M intuitively affect the gap of the shortest and the second shortest vector in the unique-SVP of $L(B') \in \mathbb{Z}^{(m+1)\times(m+1)}$, since the reduction output is $\mathbf{w} = (\frac{\mathbf{e}}{M})$. For the entries of error vector $\mathbf{e}$ are randomly and linearly independently sampled from the discrete Gaussian distribution D_σ, we can assume that the distribution of $\|\mathbf{e}\|^2$ is very close to $\sigma^2 \times \chi^2$ when σ and m are sufficiently large. Here χ means chi distribution. So $\|\mathbf{e}\|^2$ has expectation of $m\sigma^2$ and we can estimate $\|\mathbf{e}\| \approx \sqrt{m}\sigma$. Lyubashevsky and

Micciancio [58] suggest that the choice for the embedding factor $M \in \mathbb{N}$ is $\|\mathbf{e}\|$. If M is bigger, then there is a lower chance to solve the given LWE instance, since the gap in unique-SVP will become smaller. On the other hand, if M is too small, with non-zero probability, there may exist a vector $\mathbf{v} \in L(B')$ such that $\|\mathbf{v} + c \cdot (\frac{\mathbf{b}}{M})\| < \|\mathbf{w}\| = \|(\frac{\mathbf{e}}{M})\|$ where $c \in \mathbb{Z}$, which causes the algorithm fails to extract the error vector according to [4]. We observe the runtime of attack by increasing M from 1 in Section 4.4.1. .

4.3.4 How to Choose m

In this part, we follow the analysis proposed by Micciancio and Regev [60]. With a small Gaussian standard deviation sigma, in many cases of instances, we may assume that the vector $(eM) \in L'$ is the shortest vector and it is much shorter than $\lambda_2(L')$. According to the Gaussian heuristic, $\lambda_2(L') = \lambda_1(L) \approx \sqrt{\frac{m}{2\pi e}} q^{(m-n)/m}$. In our experiments, we observed that the attack is more efficient if the embedding factor M is closer to 1 (see Section 4.4.1). So we want to enlarge the following gap in unique-SVP for an efficient attack:

$$\gamma(m) = \frac{\lambda_2(L')}{\lambda_1(L')} \approx \frac{\sqrt{\frac{m}{2\pi e}} q^{(m-n)/m}}{\sqrt{m}\sigma} \tag{4.6}$$

We need $\sigma \ll q^{\frac{m-n}{m}}$. Moreover, it is known that the gap $\gamma(m) > c \cdot \delta^m$ to solve the unique-SVP by using a lattice reduction algorithm with a root Hermite factor δ, with a high probability. The constant c is unknown, so we can maximize $q^{(m-n)/m}/\delta^m$, to get the optimal sub-dimension m of LWE sample instances is

$$m = \sqrt{n \log(q) / \log(\delta)}. \tag{4.7}$$

This can properly enlarge the gap in γ-unique SVP transformed from BDD, within a reduction algorithm's capability estimated by the root Hermite factor $\delta = \mathrm{rHF}(\mathbf{b}_1, \ldots, \mathbf{b}_n) = (\|\mathbf{b}_1\| / \mathrm{vol}(L)^{1/n})^{1/n}$.

4.4 Experimental Results and Analysis

In this section, we give the details in our experiments on solving LWE problems using embedding technique (Algorithm 6). All the cases are taken from Darmstadt LWE

Challenge [27]. In our experiments, we just observe the hardness of small dimensions from 40 to 60, with the same $\alpha = 0.005$. As a preparing work, we take δ in the range $[1.010, 1.011, \ldots, 1.025]$ and randomly sample $m = \sqrt{n \log(q)/\log(\delta)}$ vector entries for $\mathbf{A} \in \mathbb{Z}_q^{m \times n}$ in each LWE case. For each case with parameters (n, δ), we sample 20 different bases. The progressive BKZ algorithm and its open source code of version 1.1 are used in the **Step 3** of Algorithm 6. Our implementation using C language and NTL on Intel(R) Xeon(R) CPU E5-2697 v2 @ 2.70GHz with 24 cores. Xu et al. were using parallel implementation technique and the specifications of hardware are 3.60 GHz Intel Core i7 processor with eight cores and a cluster consisting of 20 c4.8xlarge instances, each equipped by 36 cores (hyper-threaded to 72 threads) [84]. The time unit in the following sections are all single thread seconds. In our work, the "runtime" exactly means the runtime on solving LWE successfully, namely, for an input LWE sample, we measure the runtime of the algorithm, until it outputs the corresponding solution within the limited time T (in this case we say the algorithm succeeds or successful case). On the other hand, if the algorithm does not halt in the limited time T, we do not measure the runtime of the algorithm (in this case we say the algorithm fails or failure cases, as we mentioned in Section 4.3.2). The average runtime in this paper is the average runtime of the successful cases without failure cases.

4.4.1 Efficiency by changing M.

As we discussed in Section 4.3.3, in the **Step 2** of Algorithm 6, the embedding factor M in basis $B' \in \mathbb{Z}^{(m+1)\times(m+1)}$ affects the size of gap in the unique-SVP of $L(B')$. In this section we will observe what size of M is better for an efficient embedding technique. The fixed dimension m of $L_{(\mathbf{A},q)}$ is referred to Section 4.4.2, and the embedding factor M is from 1 to around 55. For each case of parameters $n = 40, 45, 50, 55$ with fixed $\alpha = 0.005$, we sample a same basis $\mathbf{A} \in \mathbb{Z}^{m \times n}$ from Darmstadt LWE Challenge respectively. Note that since we want to observe the efficiency of Algorithm 6 for successful cases, the margin at runtime T is set by 12 hours, which is large enough to make sure the 100% success rate. Hence the "runtime" is of successful cases on solving LWE here. Figure 4.1 shows the runtime of Algorithm 6 for each case with increasing sequence of embedding factor M. We can observe that with growing M, the runtime of Algorithm 6 is gradually

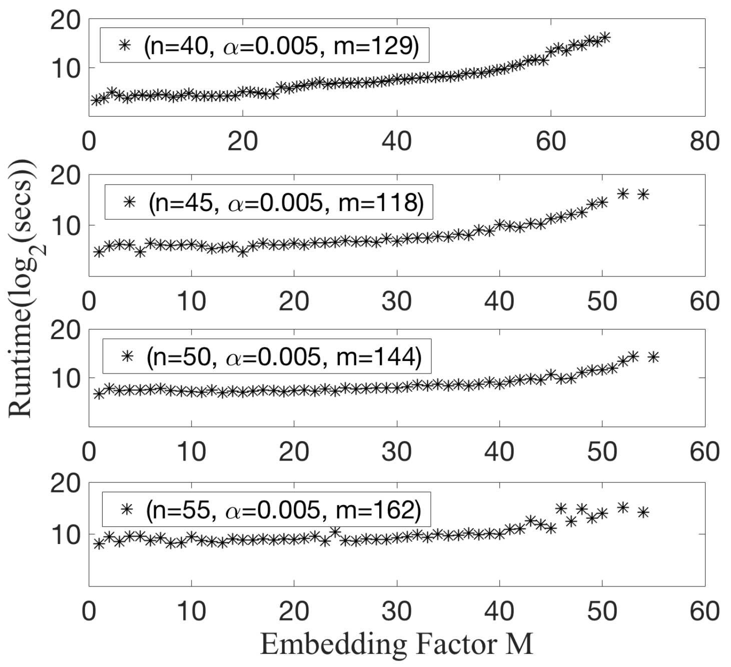

FIGURE 4.1: Runtime for cases (n, α) with fixed bases and increasing embedding factor M.

increasing. So it is more efficient to solve LWE problem with the embedding factor M closer to 1.

4.4.2 Optimal Choice of m for each (n, α)

Due to the equations (4.6) and (4.7) in Section 4.3.4, the dimension m of lattice $L_{(\mathbf{A},q)}$ in **Step 3** also affects the efficiency of Algorithm 6. A larger m will lead the root Hermite Factor smaller, which makes the lattice algorithm inefficient. While a smaller m will reduce the gap of unique-SVP and make the problem harder to solve. In this section, we observe the affect of size m on the efficiency of Algorithm 6.

At first for each case of $(n, \alpha = 0.005)$, we fix the embedding factor as $M = 1$. We take δ in the range $[1.010, 1.011, \ldots, 1.025]$ and for each δ calculate $m = \sqrt{n \log(q)/\log \delta}$. We did the experiments for $n = 40, 45, 50, 55, 60, 65$. Note that for case of $(n = 40, \alpha = 0.005)$, since the runtime are close to each other, we ignore it here. When Algorithm 6 fails, we randomly sample the input basis again and re-run Algorithm 6. When the random executions had been done up to 20 times (meaning 20 random samples of $\mathbf{A} \in \mathbb{Z}_q^{m \times n}$), we then increase the number of samples m (by decreasing delta from 1.025

TABLE 4.1: Experimental pBKZ runtime for each $(n, \alpha = 0.005)$ cases with parameter δ in range $[1.01, 1.011, \ldots, 1.025]$.

(n, α)	δ	m	Average pBKZ cost ($\log_2$(secs))	Minimum pBKZ cost ($\log_2$(secs))
(45, 0.005)	1.025	118	5.99	4.28
(50, 0.005)	1.019	144	7.51	6.49
(55, 0.005)	1.017	162	9.03	8.08
(60, 0.005)	1.013	195	13.13	10.62
(65, 0.005)	1.012	213	16.04	14.65

to 1.010) and re-run Algorithm 6. In the end, we calculated the runtime of successful cases. In Table 4.1 the "Average BKZ Runtime" shows the minimum of average runtime for each δ. Further, the "Minimum BKZ Runtime" is the minimum data for the relevant δ and m.

Lemma 4.1 ([6], Lamma 9)**.** *Given an (n, q, α) LWE instance, any lattice reduction algorithm achieving log root- Hermite factor:*

$$\log \delta = \frac{\log^2(\epsilon' \tau \alpha' \sqrt{2e})}{4n \log q}$$

solves LWE with success probability greater than

$$\epsilon_\tau \cdot \left(1 - \left(\epsilon' \cdot \exp\left(\frac{1 - \epsilon'^2}{2}\right)\right)^m\right)$$

for some constant $\epsilon' > 1$ and some fixed $\tau \leq 1$, and $0 < \epsilon_\tau < 1$ as a function of τ. Here $\alpha' = \alpha \cdot \sqrt{2\pi}$.

Experimentally $\tau \approx 0.3$ is derived when the embedding factor $M = 1$ due to [4]. The relevant parameter setting in our work is $(n, q, \alpha') = (n, n^2, 0.005 \cdot \sqrt{2\pi})$, hence by a certain success probability, we can get the approximate relation between m and n as follows.

$$
\begin{aligned}
m &= \sqrt{n \cdot \frac{\log q}{\log \delta}} \\
&\approx \sqrt{n \cdot 2 \cdot \log n \frac{8 \cdot n \log n}{\log^2(\epsilon' \tau \alpha' \sqrt{2e})}} \\
&\approx n \cdot \log n \cdot \frac{4}{\log(\epsilon' \tau \alpha' \sqrt{2e})} \\
&\approx n \cdot \log n \cdot const.
\end{aligned}
$$

So from experimental results shown in Table 4.1, we can get the following fitting function of optimal m and n.

$$
\text{Optimal}(m) = \ulcorner 0.9634 \cdot n \cdot \log n - 46.37 \lrcorner. \tag{4.8}
$$

Here the mark $\ulcorner \ldots \lrcorner$ means taking a rounding number. Note that our experimental results being used are of success rate $\geq 90\%$, while the experimental success rate is about 10% using the parameter settings in [6]. From the initial experiment of small scale, we set the limited time T as three days in this experiment. Therefore, we measure the (successful) runtime of solving LWE using the range of m computed from equation (4.7). Recall that the runtime in Table 4.1 is an average and minimum runtime of the successful cases. Namely, the runtime of the failure cases (less than 10% in our experiments) are not included in Table 4.1.

4.4.3 Estimating Runtime by pBKZ Simulator

Now we explain how to estimate the cost for solving the LWE cases using pBKZ simulator. At first, given (n, q, α)–LWE instance, we can get the optimal number of samples m corresponding to n by equation (4.8). Then we can compute the rHF δ required for the attack using the equation (4.7). From the Proposition 2.20, we get the relation between GSA constant r and rHF δ as $r = \delta^{-4m/(m-1)}$. So after computing r from δ we compute the target β from formulas (3.10). Finally we get the input parameter set (n, β) for pBKZ simulator and the runtime of pBKZ is derived. We plot the simulated pBKZ cost by a solid blue curve in Figure 4.2. The blue stars in Figure 4.2 are the

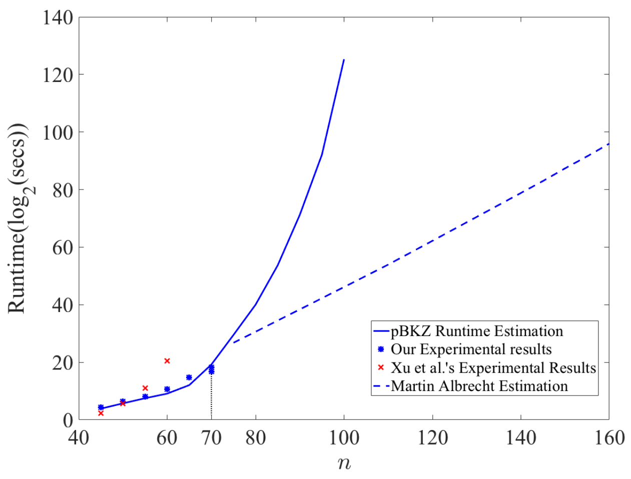

FIGURE 4.2: The runtime for embedding technique on Darmstadt LWE Challenge of $(n, \alpha = 0.005, q \approx n^2)$ cases: The solid line is the estimated cost from pBKZ simulator; The stars denotes our experimental results; The red crosses are Xu et al.'s records at the LWE Challenge website; The dot line is computed from M.A. estimator which can be seen as a lower bound heuristically.

minimum of Average Runtime in each (n, α) cases as showed in Table 4.1. It shows that our experimental results are close to the pBKZ simulating results.

More precisely, using the constructed function 4.8 from the experimental results of dimensions ≤ 65 (the stars in figure), we computed the optimal m for each dimension $n \in \{45, 50, \ldots, 100\}$ respectively. Simultaneously, we got the corresponding target root Hermite factor δ from equation 4.7 using the optimal number of samples m. Then we input m and δ into the pBKZ simulator and got the output of simulated runtime. Thus, we plotted the blue curve of simulating runtime. Moreover, from Figure 4.2 we can see that Xu et al.'s LWE Challenge records of $\alpha = 0.005$ stopped at $n = 65$ for the overwhelming runtime and low success probability [83]. Our implemented embedding technique with progressive BKZ can solve the LWE Challenge instances more efficiently than Xu et al.'s enumeration implementation for $n \geq 55$. We also show M.A. estimation as an asymptotic lower bound from dimension 75 (the target $\beta > 90$) in Figure 4.2.

Furthermore, in Table 4.2, we estimate the necessary dimension m and the relevant practical runtime by embedding technique on solving LWE Challenge cases $n \geq 70, \alpha = 0.005$. Our estimation is using pBKZ simulator.

Table 4.2: Estimation of effective m and runtime on solving$(n \geq 70, \sigma = 0.005)$ in the LWE Challenge when running the pBKZ algorithm.

(n, α)	q	δ	m	pBKZ cost ($\log_2$(secs))
(70, 0.005)	4903	1.0104	240	19.22
(75, 0.005)	5639	1.0092	266	29.51
(80, 0.005)	6421	1.0083	291	40.02
(85, 0.005)	7229	1.0075	317	53.69
(90, 0.005)	8101	1.0069	344	71.24
(95, 0.005)	9029	1.0063	370	92.12
(100, 0.005)	10007	1.0059	397	125.26

4.4.4 Record of $(n = 70, \alpha = 0.005)$ in LWE Challenge

For the cases of $(n = 70, \alpha = 0.005)$, we compute the extrapolated $m \approx 240$ from function (4.8). Then we increase the number of samples as m=214, 223, 233, 244 (relevant δ =1.013, 1.012, 1.011, 1.010). From the original matrix of size 4900×70, we randomly sample $m \times 70$ sized matrices by 5 times for each m, as input $\mathbf{A}$ in Algorithm 6. Simultaneously, we use 2^{21} seconds as the time limit T, which is slightly larger than the simulated $2^{19.22}$ seconds in Table 4.2. Consequently, there are two cases of $m = 233,\ 223$ ($\delta = 1.011, 1.012$) successfully solved by time $2^{16.8}$, $2^{18.2}$ seconds respectively. Hence totally we ran 20 times to solve the challenge and get the solution from two cases. Therefore, we can say the success probability is $2/20 = 10\%$ in this experiment. We plot it in Figure 4.2, which are close to the pBKZ estimation.

4.5 Conclusions

In this chapter, we studied Kannan's embedding technique on solving LWE problem. Especially we applied the progressive BKZ algorithm on some randomly sampled LWE instances from Darmstadt LWE Challenge. From our experiments of fixed relative error size $\alpha = \sigma/q = 0.005$, we observed that the algorithm has a more efficient trend if the embedding factor M is closer to 1. Furthermore, Xu et al.'s LWE Challenge records of $\alpha = 0.005$ stopped at $n = 55$ for the overwhelming runtime, while our experimental results show that for $n \geq 55$, the embedding technique with progressive BKZ can solve the LWE Challenge instances more efficiently than Xu et al.'s implementation of Liu-Nguyen's enumeration algorithm. Then we proposed a formula to decide the optimal number of LWE samples m, for given the length n of secret vector $\mathbf{s}$. Using this formula, we estimated the runtime of solving LWE cases using the pBKZ simulator and our experimental results are close to the estimations. Finally our LWE Challenge record of $(n, \alpha) = (70, 0.005)$ cases succeeded in $2^{16.8}$ seconds (32.73 single core hours), which also lies close to the pBKZ estimations.

Chapter 5

Evaluating Secure Parameters for Ding Key Exchange

At the beginning of 2016, NIST started the post-quantum cryptography competition to prepare for the standardization of the next-generation cryptography. Ding et al. proposed a “Ding Key Exchange" scheme, which is based on the Ring Learning With Errors Problem (RLWE). The number of samples in their scheme is just one, which is different from the case of normal integer LWE or other RLWE instances. In this chapter, we do the security analysis for Ding key exchange by primal attack which is reducing the RLWE to SIS. Hence we can expand the dimension of the attack basis to double. We adopt both the progressive BKZ simulator and the M.A. estimator introduced in the last two chapters 3 and 4. Guaranteeing the error rate of key exchange protocol within 2^{-60}, our parameter choices cover the security of AES-128/192/256 respectively, which satisfies NIST’s security category I, III and V respectively. This is one part work in the NIST PQC standardization proposal “Ding Key Exchange” [67].

5.1 Introduction

During recent years, various works are focusing on the lattice-based Ring Learning With Errors (RLWE) problem [59], which is the ring variant of Learning With Errors (LWE) problem [69]. They enjoy high efficiency as well as strong security, making them very promising towards the post-quantum world. Ever since LWE and RLWE problems were introduced in 2005 and 2010 respectively, people have tried to seek solutions to construct key exchange over LWE/RLWE problem. A major advantage for RLWE compared with LWE is that it has a much reduced key size, and this is more desirable for real world applications due to smaller communication and storage cost. Therefore, among all approaches to realize post-quantum cryptography, RLWE-based construction is one of the most promising and practical replacements for current public key cryptosystems.

At the first stage of NIST “call for PQC protocols", J. Ding, T. Takagi, X. Gao, Y. Wang (the author of this thesis) proposed a “Ding key exchange" [31], which is an ephemeral-only RLWE-based key exchange protocol with a new efficient rounding technique to reduce communication cost. The construction is a RLWE variant of the classic Diffie-Hellman key exchange protocol, which can be regarded as a direct drop-in replacement for current widely-deployed Diffie-Hellman key exchange protocol (and its variants, e.g. elliptic curve Diffie-Hellman) without significant modifications to current security protocols and applications. There are some schemes follow the same idea of sending additional information – signal value other than public key to construct LWE/RLWE/MLWE-based key exchange/Key Encapsulation Mechanism (KEM) protocols, as NewHope [7], Frodo [17], Kyber [19], etc.

Roadmap. Section 5.2 recalls the Ding key exchange, including some notations and functions defined in the protocol. Then we discuss the parameter settings in the specific ONE-sample RLWE case in Section 5.3. Our cryptanalysis and proposed secure parameter sets are shown in Section 5.4. Finally we give the conclusion in Section 5.5.

5.2 Ding Key Exchange Review

In this section, we simply recall the Ding key exchange, after give some notations and core functions being used in the protocol.

5.2.1 Core Functions in Ding Key Exchange.

We introduce several useful functions and lemmas that will be seen in Ding key exchange protocol in Section 5.2.2. For the proof of lemmas, please refer to the original proposal in [31]. In this section, we use the notations defined in Chapter 2 and following additional ones. Let $U[a, b]$ be the uniform distribution over discrete set $\{a, a+1, \cdots, b-1, b\}$ over integers. Let $\lfloor x \rfloor$ be the floor function which outputs the greatest integer that is less than or equal to x, $\lceil x \rceil$ be ceiling function which outputs the least integer that is greater than or equal to x, $\lfloor x \rceil$ be the rounding function which rounds x to nearest integer.

Lemma 5.1 ([80], lemma 2.5)**.** *For* $\sigma > 0$, $r \geq 1/\sqrt{2\pi}$, $\Pr[\|\mathbf{x}\|_2 > r\sigma\sqrt{n}; \mathbf{x} \xleftarrow{\$} D_{\mathbb{Z}^n,\sigma}] < (\sqrt{2\pi e r^2} \cdot e^{-\pi r^2})^n$.

Lemma 5.2. *For* $\mathbf{a}, \mathbf{b} \in R_q$, $\|\mathbf{a} \cdot \mathbf{b}\|_\infty \leq \|\mathbf{a}\|_2 \cdot \|\mathbf{b}\|_2$.

Rounding function. For $x \in \mathbb{Z}_q$, $q > p > 0$ be integers. x is a coefficient of polynomial in R_q, q, p are parameters of our protocol.

For the convenience of notation, we change the representation of $x \in \{-\frac{q-1}{2}, \cdots, \frac{q-1}{2}\}$ to $x \in \{0, \cdots, q-1\}$ before Round() runs. Function Round(x, p, q) is defined in algorithm 7.

Algorithm 7 Round(x, p, q)

```
Input: $x \in \mathbb{Z}_q, p, q$
Output: Rounded value $x'$ of $x$
1: $t \leftarrow \lfloor 2q/p \rfloor, k \leftarrow \lfloor x/t \rfloor$
2: if $x$ is odd number then
3:     $x' \leftarrow 2k + 1$
4: else $x' \leftarrow 2k$
5: if $x' = p$ then
6:     $rnd \xleftarrow{\$} U[0, 1]$
7:     if $rnd = 1$ then
8:         $x' \leftarrow x' - 2$
9:     else $x' \leftarrow (x' + 2) \mod (p + 1)$
```

Rounding function is defined for an integer $x \in \mathbb{Z}_q$. Rounding function for $\mathbf{a} \in R_q$ is computed by applying Round() for each coefficient $a_i \in \mathbb{Z}_q$ of $\mathbf{a} \in R_q$. In this document, we use the same notation Round() for both rounding functions over $\mathbb{Z}_q$ and R_q.

Recovering function. Recover() is a deterministic function. $q > p > 0$ be integers. x' is one coefficient of rounded polynomial, q, p are parameters of our protocol. Function Recover(x', p, q) is defined in algorithm 8.

Algorithm 8 Recover(x', p, q)

```
Input: $x', p, q$
Output: Recovered value $x''$ of $x'$
1: $t \leftarrow \lfloor q/p \rfloor$
2: if $x'$ is odd number then
3:     $x'' \leftarrow x' \cdot t + 1$
4: else $x'' \leftarrow (x' + 1) \cdot t$
```

In order to be consistent with theoretical analysis, we change representation of $x'' \in \{0, \cdots, q-1\}$ to $x'' \in \{-\frac{q-1}{2}, \cdots, \frac{q-1}{2}\}$ after Recover() runs.

Recovering function is defined for an integer x'. Recovering function for vector $\mathbf{a}$ is computed by applying Recover() for each coefficient a_i in vector $\mathbf{a}$. In this document, we use the same notation "Recover()" for both recovering functions over integer x' and vector $\mathbf{a}$.

Lemma 5.3. *For parameter p and q, let* $t = \lceil \log_2 q \rceil - \lceil \log_2 p \rceil$, $\boldsymbol{x} = (x_1, x_2, \cdots, x_n)$ *be a vector whose each coefficient is uniformly random sampled integer in* $\mathbb{Z}_q$, ***x'*** *be a vector whose each coefficient* x'_i *= Recover(Round(*x_i, p, q*),p,* q*). Let **d** = **x-x'** be a vector whose each coefficient* $d_i = x_i - x'_i$ *(*$i \in [1, n]$*). Then* d_i *is an even number with possible values in set* $\{-2^t, -2^t+2, \cdots, 2^t-2\}$. $\Pr[d_i = -2^t] = \Pr[d_i = -2^t+2] = \cdots = \Pr[d_i = 2^t - 2] = \frac{1}{2^t}$

With lemma 5.1 and 5.2, we have $4\|\mathbf{se}\|_\infty \le 4\|\mathbf{s}\|_2 \cdot \|\mathbf{e}\|_2 \le 4(r\sigma\sqrt{n})^2 = 4r^2\sigma^2 n$, where $r \ge 1/\sqrt{2\pi}$ is defined in lemma 5.1 and n is the degree of polynomial. With lemma 5.3, we have $2\|\mathbf{d}'\mathbf{s}\|_\infty \le 2\|\mathbf{d}'\|_2 \cdot \|\mathbf{s}\|_2 = 2\|\mathbf{d}'\|_2 \cdot r\sigma\sqrt{n}$. Recall that error tolerance $\delta = \frac{q}{4} - 2$. Therefore as long as $q \ge 4 \cdot [2 + (4r^2\sigma^2 n) + (2\|\mathbf{d}'\|_2 \cdot r\sigma\sqrt{n})]$, key exchange failure probability is estimated to be $(\sqrt{2\pi e r^2} \cdot e^{-\pi r^2})^n$.

Hint function. Hint functions $\sigma_0(x)$, $\sigma_1(x)$ from $\mathbb{Z}_q$ to $\{0,1\}$ are defined as:

$$\begin{aligned}
\sigma_0(x) &= \begin{cases} 0, x \in [-\lfloor \frac{q}{4} \rfloor, \lfloor \frac{q}{4} \rfloor] \\ 1, otherwise \end{cases}, \\
\sigma_1(x) &= \begin{cases} 0, x \in [-\lfloor \frac{q}{4} \rfloor + 1, \lfloor \frac{q}{4} \rfloor + 1] \\ 1, otherwise \end{cases}
\end{aligned}$$

Signal function. A signal function Sig() is defined as:

For any $y \in \mathbb{Z}_q$, Sig$(y) = \sigma_b(y)$, where $b \xleftarrow{\$} U[0,1]$. If Sig$(y) = 1$, we say y is in the outer region, otherwise y is in the inner region.

Signal function is defined for an integer $x \in \mathbb{Z}_q$. Signal function for $\mathbf{a} \in R_q$ is computed by applying Sig() for each coefficient $a_i \in \mathbb{Z}_q$ of $\mathbf{a} \in R_q$. In this document, we use the same notation "Sig()" for both signal functions over $\mathbb{Z}_q$ and R_q.

Reconciliation function. Mod$_2()$ is a deterministic function with error tolerance δ. Mod$_2()$ is defined as: for any x in $\mathbb{Z}_q$ and $w =$ Sig(x), Mod$_2(x, w) = (x + w \cdot \frac{q-1}{2} \mod q) \mod 2$. Here we treat elements in $\mathbb{Z}_q$ as elements in $\mathbb{Z}$ before we perform the modulo 2 operation.

There are some others being used in Ding key exchange as hint function, signal function, reconciliation funtion, etc. Please refer to [31] for more details.

5.2.2 Ding Key Exchange

The protocol is illustrated in Figure 5.1. Key exchange protocol is instantiated with following parameters:

- Modulus q
- Degree n of R_q
- σ of distribution $D_{\mathbb{Z}^n,\sigma}$ to sample s and e
- Rounding parameter p

$$\begin{array}{lcl}
\text{Party } i & & \text{Party } j \\
seed \xleftarrow{\$} \{0,1\}^{128} & & \\
a = \text{Derive\_a}() \in R_q & & a = \text{Derive\_a}() \in R_q \\
\text{Public key: } p_i = a \cdot s_i + 2e_i \in R_q & & \text{Public key: } p_j = a \cdot s_j + 2e_j \in R_q \\
\text{Private key: } s_i \in R_q & \xrightarrow{p'_i,\, seed} & \text{Private key: } s_j \in R_q \\
\text{where } s_i, e_i \xleftarrow{\$} D_{\mathbb{Z}^n,\sigma} & & \text{where } s_j, e_j \xleftarrow{\$} D_{\mathbb{Z}^n,\sigma} \\
p'_i = \text{Round}(p_i, p, q) & & p'_j = \text{Round}(p_j, p, q) \\
 & & \\
 & & p''_i = \text{Recover}(p'_i, p, q) \in R_q \\
p''_j = \text{Recover}(p'_j, p, q) \in R_q & & k_j = p''_i \cdot s_j \in R_q \\
k_i = p''_j \cdot s_i \in R_q & \xleftarrow{p'_j,\, w_j} & w_j = \text{Sig}(k_j) \in \{0,1\}^n \\
sk_i = \text{Mod}_2(k_i, w_j) \in \{0,1\}^n & & sk_j = \text{Mod}_2(k_j, w_j) \in \{0,1\}^n
\end{array}$$

FIGURE 5.1: The Ding key exchange protocol

Proposition 5.4. *From cryptanalysis perspective, the RLWE distribution with an even error polynomial in the Ding key exchange is no difference from the general RLWE instance.*

Proof. Let $\mathbf{z} = \text{Recover}(\text{Round}(\mathbf{a} \cdot \mathbf{s} + \mathbf{2e}, p, q), p, q) = \mathbf{as} + \mathbf{2e} + \mathbf{d} = \mathbf{as} + \mathbf{2f} \in R_q$, where $\mathbf{s}, \mathbf{e} \xleftarrow{\$} D_{\mathbb{Z}^n,\sigma}$ and $2\mathbf{f} = 2\mathbf{e} + \mathbf{d}$. Hence we can regard $\mathbf{f}$ as error term $\mathbf{e}$ in the definition of RLWE above. The attack on our protocol is given $\mathbf{z}$ and $\mathbf{a}$, output private key $\mathbf{s}$. This problem is equivalent to:

$$\begin{aligned}
& \mathbf{z} = \mathbf{a} \cdot \mathbf{s} + 2\mathbf{f} \mod q \\
\Leftrightarrow\ & 2^{-1}\mathbf{z} = 2^{-1}\mathbf{a} \cdot \mathbf{s} + \mathbf{f} \mod q \\
\Leftrightarrow\ & \mathbf{z}'' = \mathbf{a}'' \cdot \mathbf{s} + \mathbf{f} \mod q.
\end{aligned}$$

□

Standard deviation of term $\mathbf{f}$ is denoted as σ_f. Note that σ_f is different from σ notation as $\mathbf{f}$ no longer follows discrete Gaussian distribution (histogram shows similar shape as Gaussian distribution), therefore σ_f is computed as the square root of variance.

5.3 Parameter Settings for ONE-sample RLWE Case

There are some algorithms to solve the underlying LWE problem: the Blum-Kalai-Wassermann (BKW) algorithm [16], the decoding attacks [13][56], the embedding attacks [49] [14], the Arora-Ge attack [12], etc. Thereinto, the BKW attack and Arora-Ge attack are not practical since they require at least sub-exponential number of samples. In cases given fixed number of samples in the implementation, amplifying samples leads a higher noise.

Multiple samples are considered available for the adversary in some existing RLWE-based key exchange schemes, as BCNS [18], NewHope [7], etc. It is very clear that for an RLWE-based ephemeral key exchange, an attack can only get one sample, however existing security analysis mostly relies on techniques which requires multiple RLWE samples. Each RLWE sample can be expanded to n LWE samples by rotating elements in the convolution polynomial ring. For the case of only ONE-sample RLWE, when using the standard embedding method or the decoding attack, the polynomials are extracted to a lattice of $L_{(\mathbf{A},q)} = \{\mathbf{v} \in \mathbb{Z}_q^m \mid \mathbf{v} \equiv \mathbf{A}\mathbf{x} \pmod q, \mathbf{x} \in D_\sigma^n\}$ while the lattice is trivial when $m \leq n$. Recently a new paper [72] developed techniques in solving standard LWE instances with a restricted number of samples. However it is not adapted in practical security analysis of RLWE key exchanges directly. Especially we can not adopt the LWE-estimator [6] directly because of the perturbations from the rounding/recovering functions in our key exchange scheme. We developed the security analysis of the dual embedding attack (we call “SIS attack" in this work) on solving ONE-sample RLWE case.

Further, in Table 5.1 we show the complexity of solving the standard LWE instance using SIS attack given n and $2n$ samples. We use Regev's parameter settings $(n, \alpha = \frac{1}{\sqrt{2\pi n}\log^2 n}, q \approx n^2)$ in the original LWE paper [69]. We estimate the hardness of standard LWE for $n = \{128, 512, 1024, 2048\}$ using the LWE-estimator [6] and restrict the number of given samples to n and $2n$. From the table we can see that the gap of complexities is distinctly larger with n increasing. Note that the n and $2n$ samples here can be seen as extracted from ONE-sample RLWE case and TWO-samples RLWE case respectively. Hence for the security analysis of RLWE instance, the available number of samples may lead a big gap for high dimensions.

TABLE 5.1: Hardness estimation for restricted number of LWE samples with Regev's parameter settings from LWE-estimator.

n	128		512		1024		2048	
#{given samples}	128	512	512	1024	1024	2048	2048	4096
#{used samples}	128	228	512	919	1024	1853	2048	3821
logarithmic complexity (clock cycles)	66.8	57.7	241.4	201.6	497.3	410.2	1043.8	851.5

5.4 Estimating Security of One RLWE Sample

5.4.1 Algorithms for Solving RLWE

There are several algorithms for solving RLWE. In Figure 5.2, we show several possible attacks on RLWE problem with only one given instance.

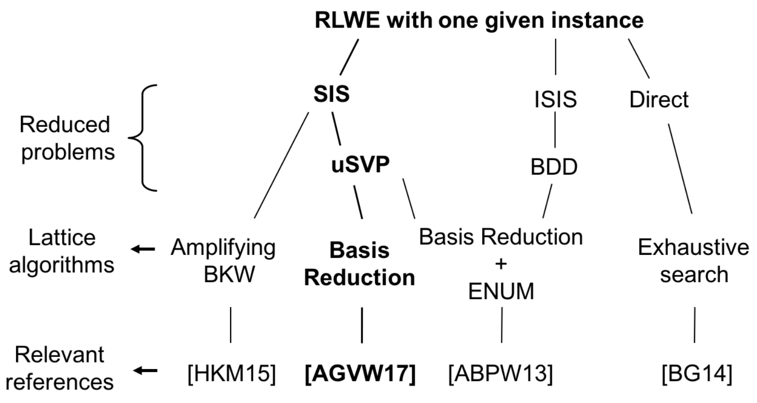

FIGURE 5.2: Possible attacks on search RLWE problem with only one instance. References [HKM15], [AGVW17], [ABPW13] and [BG14] are [44], [5], [9] and [14] respectively.

We explain how we choose appropriate attacks from available options:

- Firstly, exhaustive search is not efficient.
- Secondly, BKW algorithm can solve LPN problem with $2^{O(n/\log n)}$ samples and runtime. Since LWE is a descendant of LPN, BKW algorithm can also be adapted to solve LWE problem (both decision and search versions) with $2^{O(n)}$ complexity,

when modulus q has polynomial size of dimension n. Amplifying technique is used in BKW algorithm to solve LPN problem and LWE problem [57, 44]. However, the analysis on amplifying BKW until now are asymptotical and there is no precise analysis on RLWE problem, i.e. amplifying BKW requires $O(n\log_2(n))$ samples which will lead to much larger standard deviation of e.

- Thirdly, complexity analysis of "reduction+ENUM" method is not clear for large (> 1000) dimensional basis.
- Finally, for security analysis of our protocol, we adapt a conservative primal method: reduce RLWE problem to SIS problem, then reduce it to unique SVP problem. Then we process the basis using BKZ reduction algorithm with sieving algorithm as SVP oracle in BKZ subroutines.

We show the SIS attack on RLWE in Algorithm 9. This algorithm is an adaptation of the dual-embedding method mentioned in [72] and [14].

5.4.2 Phase Transition of Sieving Algorithm

Now we discuss the memory space consumption for sieving algorithms. If we convert the memory cost of sieving algorithm to the computational cost, the memory cost in the sieving overwhelms the computation one. In [15], the authors evaluate the Gauss Sieving and give the trade off between the runtime and memory space as equal as $2^{0.292\beta+o(\beta)}$, where β is the dimension of given basis or blocksize processed by sieving. Albrecht et al. replaced $o(\beta)$ by 16.4 based on the experiments in [51]. Note that the unit of space complexity here is the number of β-dimensional vectors sampled in sieving algorithm. We convert it to bits by multiplying β under the assumption that the elements in the vectors are only 1 bit.

Due to the report of [66] in 2011, the total amount of the data in the world is around 295 exabytes $\approx 2^{68}$ bits and it grows to 16.1 zettabytes $\approx 2^{77}$ bits in 2016 [70]. Furthermore, it is expected to be $10 = 2^{3.3}$ times larger in the next 10 years [70]. Totally, the number of atoms in the earth is around $10^{49} \approx 2^{162}$ bits [33]. Hence there is a phase transition at 2^{77} (whole data) and 2^{162} (atoms in the earth), namely we can not perform the sieving physically. The currently used parameters for AES128-equivalent security in almost all

Algorithm 9 The SIS Attack on Ring LWE Problem

```
Input: $m'$ instances from RLWE key exchange: $(\mathbf{a}, \mathbf{b} = \mathbf{a} \cdot \mathbf{s} + \mathbf{e}) \in (R_q, R_q)$. Here
    $m' = poly(n)$, $R_q = \mathbb{Z}_q[x]/(x^n + 1)$ with $q$ as a prime, where secret polynomial s
    and error polynomial $\mathbf{e}$ are sampled from Gaussian distribution $D_{\mathbb{Z}^n,\sigma}$ with standard
    deviation $\sigma$.
Output: The polynomial $\mathbf{e}$ and s s.t. $\mathbf{b} = \mathbf{a} \cdot \mathbf{s} + \mathbf{e} \in R_q$.
 1: Rewrite the RLWE instance to Short Integer Solution (SIS) instance by
    ① Write the polynomials $\mathbf{a_i} \in R_q$, $\mathbf{b_i} \in R_q$, $\mathbf{e_i} \overset{\$}{\leftarrow} D_{\mathbb{Z}^n,\sigma} \in R_q$ for $i \in [1, m']$ and the
    only one $\mathbf{s} \in R_q$ as follows:
        $\mathbf{a_i} = a_{i1} + a_{i2}x + \cdots + a_{in}x^{n-1}, \mathbf{b}_i = b_{i1} + b_{i2}x + \cdots + b_{in}x^{n-1},$
        $\mathbf{e_i} = e_{i1} + e_{i2}x + \cdots + e_{in}x^{n-1}$ and $\mathbf{s} = s_1 + s_2x + \cdots + s_nx^{n-1}$
    to vector form as
    $\mathbf{a_i} = (a_{i1}, a_{i2}, \cdots, a_{in}) \in \mathbb{Z}_q^{1\times n}, \mathbf{b}' = (b_{11}, b_{12}, \cdots, b_{1n}, \cdots, b_{m'1}, b_{m'2}, \cdots, b_{m'n})^T \in \mathbb{Z}_q^{m'n\times 1},$
        $\mathbf{e}' = (e_{11}, e_{12}, \cdots, e_{1n}, \cdots, e_{m'1}, e_{m'2}, \cdots, e_{m'n})^T \overset{\$}{\leftarrow} D_{\mathbb{Z}^{m'n},\sigma} \in \mathbb{Z}_q^{m'n\times 1},$
        $\mathbf{s} = (s_1, s_2, \cdots, s_n)^T \overset{\$}{\leftarrow} D_{\mathbb{Z}^n,\sigma} \in \mathbb{Z}_q^{n\times 1}.$
    ② Rotate polynomials $\mathbf{a}_i = a_{i1} + a_{i2}x + \cdots + a_{in}x^{n-1} \in R_q$ to get matrices
        $\mathbf{A}_i = \begin{pmatrix} a_{i1} & a_{i2} & \cdots & a_{i(n-1)} & a_{in} \\ -a_{in} & a_{i1} & a_{i2} & \cdots & a_{i(n-1)} \\ -a_{i(n-1)} & -a_{in} & a_{i1} & \cdots & a_{i(n-2)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ -a_{i2} & -a_{i3} & -a_{i4} & \cdots & a_{i1} \end{pmatrix}.$
    Then compose $\mathbf{A}' = [\mathbf{A}_1\mathbf{A}_2\cdots\mathbf{A}_{m'}]^T \in \mathbb{Z}_q^{m'n\times n}$.
    ③ Derive $\mathbf{A} \in \mathbb{Z}_q^{m\times n}$ by sampling $m$-row vectors from $\mathbf{A}' \in \mathbb{Z}_q^{m'n\times n}$. Similarly,
    derive $\mathbf{b} \in \mathbb{Z}_q^m$ from $\mathbf{b}' \in \mathbb{Z}_q^{m'n\times 1}$ and $\mathbf{e} \in \mathbb{Z}_q^m$ from $\mathbf{e}' \in \mathbb{Z}_q^{m'n\times 1}$.
    Then we get a randomly sampled normal LWE case as $(\mathbf{A}, \mathbf{b} \equiv \mathbf{A}\mathbf{s} + \mathbf{e} \pmod q) \in$
    $(\mathbb{Z}_q^{m\times n}, \mathbb{Z}_q^m)$.
    ④ Transform the randomly sampled LWE instance $(\mathbf{A}, \mathbf{b} \equiv \mathbf{A}\mathbf{s} + \mathbf{e} \mod q) \in$
    $(\mathbb{Z}_q^{m\times n}, \mathbb{Z}_q^m)$ from ③ to a SIS instance:
    Given $(\mathbf{A}, \mathbf{b}) \in (\mathbb{Z}_q^{m\times n}, \mathbb{Z}_q^m)$, find a short vector $(\mathbf{s} \mid \mathbf{e} \mid 1) \in \mathbb{Z}_q^{m+n+1}$ s.t.
        $(\mathbf{A}\ |I_m| {-\mathbf{b}})(\mathbf{s} \mid \mathbf{e} \mid 1)^T = \mathbf{0} \mod q.$
 2: Set $\mathbf{A}'' = (\mathbf{A}\ |I_m| {-\mathbf{b}}) \in \mathbb{Z}_q^{m\times(m+n+1)}$. Compute basis $B$ of $q$-ary lattice
        $L^{\perp}_{(\mathbf{A}'',q)} = \{\mathbf{x} \in \mathbb{Z}^{m+n+1} \mid \mathbf{A}''\mathbf{x} \equiv \mathbf{0} \pmod q\}.$
    Compute kernel $Ker(\mathbf{A}'')$ of $\mathbf{A}''$ over $\mathbb{Z}^{(m+n+1)\times(n+1)}$.
 3: For some matrix $\mathbf{A}^*$, construct basis $B = \begin{pmatrix} \frac{Ker(\mathbf{A}'')^T}{q\mathbf{I}_{m+n+1}} \end{pmatrix} \in \mathbb{Z}_q^{(m+2n+2)\times(m+n+1)}$
    and compute the HNF of $B$ as $B_{\text{HNF}} = \begin{pmatrix} q\mathbf{I}_m & \mathbf{0} \\ \mathbf{A}^*_{(n+1)\times m} & \mathbf{I}_{n+1} \end{pmatrix} \in \mathbb{Z}^{(m+n+1)\times(m+n+1)}$.
 4: Apply lattice reduction algorithm (LLL or BKZ) on basis $B_{\text{HNF}}$ and get a reduced
    basis $Red(B_{\text{HNF}})$. Inside of $Red(B_{\text{HNF}})$, a short vector $\mathbf{v} = (\mathbf{s} \mid \mathbf{e} \mid 1)$ if it suc-
    ceeded.
```

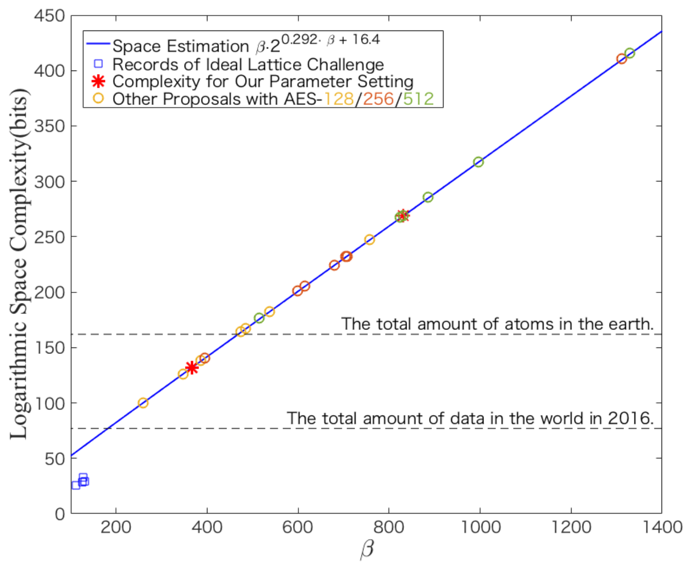

FIGURE 5.3: The space complexity of sieving algorithm.

NIST submissions are using the gigantic memory beyond the whole data in the world in 2016 due to the sieving algorithm [67]. The phase transition numbers are mentioned in NTRUEncrypt [46] and Lotus [68]. Thus NTRUEncrypt, Lotus and a few others are exceptionally using the enumeration-based method instead of the sieving algorithm.

In Figure 5.3 we give some references of required space to solve the SVP problem using sieving algorithm. We also show our estimation corresponding to dimensions of 366 and 831 in our parameter settings in Table 5.2. Simultaneously, we mark some records in ideal lattice challenge [29] which were solved by sieving algorithm: memory space of $2^{0.2\beta}$ for $\beta = 128$ [47] and memory $\{444341, 2759903, 4490083\}$ for $\beta = \{112, 126, 130\}$ in [85] respectively. Moreover, some lattice-based proposals to NIST, as uRound2, Lizard, Frodo, CRYSTALS, etc. [67] are also considered in Figure 5.3 for a reference.

5.4.3 Significance of Number of Samples in Practical Attack

At first we claim that because of the setting of our key exchange protocol: only one RLWE instance $(\mathbf{a}, \mathbf{b} = \mathbf{a} \cdot \mathbf{s} + \mathbf{e} \mod q) \in (R_q, R_q)$ is given, Kannan's embedding technique [49] and Liu-Nguyen's decoding attack [56] cannot be adopted since the lattice $L_{(\mathbf{A},q)} = \{\mathbf{v} \in \mathbb{Z}_q^m \mid \mathbf{v} \equiv \mathbf{A}\mathbf{x} \pmod q), \mathbf{x} \in D_\sigma^n\}$ is trivial when $m \leq n$. Therefore our estimator should be different from some other key exchange schemes as NewHope [7] and Albrecht's estimator [5] etc. which regard RLWE and normal LWE problem as having the same difficulty without considering the number of available RLWE samples. From the discussion in section 5.3 and the estimations in Table 5.1, we observe that there is a big gap of hardness estimations between ONE-sample RLWE and multiple-samples RLWE. Note that indeed the lowest number m of required LWE samples from the M.A. estimator (Section 5.4.4) are as follows: m is 576 for $(n, q, \sigma_f) = (512, 102833, 4.92)$ and m is 1097 for $(n, q, \sigma_f) = (1024, 120833, 4.72)$. Therefore, the optimal number in the M.A. estimator can be obtained only from "more than one RLWE samples" $(m = 576 > n = 512 \text{ and } m = 1097 > n = 1024)$. Hence in practical attack, we can get only one n-dimensional RLWE instance, which can be amplified to $2n + 1$ without changing the distribution of error vectors, see Algorithm 9. Therefore the lattice dimension of solving RLWE in our case is $d = 2n + 1$.

5.4.4 Our Simulator

For security analysis of our parameter choices, we refer to the approach in bold text in Figure 5.2. At AsiaCrypt 2017 [5], Albrecht et al. re-estimated the hardness of LWE problem using Kannan's embedding and Bai-Gal's embedding respectively under estimation in NewHope [7] (denoted as "2016 estimate"). 2016 estimate states that if the Gaussian Heuristic and the GSA [73] hold for BKZ-β reduced basis and

$$\sqrt{\beta/d} \cdot \|(\mathbf{e}|1)\|_2 \approx \sqrt{\beta}\sigma \leq \delta^{2\beta-d} \cdot \mathrm{Vol}(L_{(\mathbf{A},q)})^{1/d}. \tag{5.1}$$

Then error $\mathbf{e}$ can be found by BKZ-β with root Hermite Factor δ. In our case, we assume $\mathbf{f}$ is the Gaussian distributed error vector plus the uniformly distributed perturbation sampled from a bounded set due to Rounding-Recovering functions and $(\mathbf{s}|\mathbf{f}|1)$ is the target vector in our attack. So there is a gap between the distribution of $\mathbf{f}$ and the Gaussian distribution. However, given a same standard deviation σ_f, the expected length of vectors sampled from the hybrid distribution is bigger than the one sampled from Gaussian distribution on average, by a simple computation using the center limit theorem. Hence in our estimation we assume $\mathbf{f}$ is Gaussian distributed. We adapt the left side of the inequality (5.1) as $\sqrt{\beta/d} \cdot \|(\mathbf{s}|\mathbf{f}|1)\|_2 \approx \sqrt{\beta \cdot ({\sigma_e}^2 + {\sigma_f}^2)}$.

Moreover, the Bai-Galbraith's rescaling technique in [14] may be considered on the "unbalanced" target vector $(\mathbf{s}|\mathbf{f}|1)$. The idea is to make s the same standard deviation as σ_f, by multiplying a σ_f-decided parameter σ'_f on the former n vectors in the constructed dual lattice basis. Consequently the volume of the lattice is increased by $(\sigma'_f)^n$ s.t. the target root Hermite factor is raised, too. However, the rescaling technique is used in cryptanalysis when $\|\mathbf{e}\|_2 \ll \|\mathbf{f}\|_2$, i.e. the secret vectors are sparse or sampled from uniform binary or ternary distributions in random. In our case, the secret is Gaussian distributed and the expected lengths difference between $\mathbf{f}$ and $\mathbf{e}$ is $3 \sim 5$. Hence the Bai-Galbraith's rescaling technique will not improve the security level substantially and we do not involve this technique in our work.

Using SIS embedding approach to attack the protocol, we can get n samples by iterating only one given instance $z'' = a''s + f \in R_q$, therefore we need to evaluate the complexity of processing a $d = 2n + 1$ dimension basis. For BKZ reduction runtime estimation, we will give the result of progressive BKZ and Albrecht's BKZ with sieving estimator.

Step 1. A short vector $\|\mathbf{b}_1\|_2 = \delta^d \cdot \det(B)$ is assumed to be inside of the BKZ-β reduced basis B of dimension d [23], where the root Hermite Factor is

$$\delta = (((\pi\beta)^{1/\beta}\beta)/(2\pi e))^{\frac{1}{2(\beta-1)}}. \tag{5.2}$$

We pre-compute the expected root Hermite factor δ for $\beta = 10, \cdots, n$ and rewrite inequality (5.1) as

$$\sqrt{\beta \cdot ({\sigma_e}^2 + {\sigma_f}^2)} \leq \delta^{2\beta-d} \cdot \mathrm{Vol}(L_{(\mathbf{A},q)})^{1/d}. \tag{5.3}$$

To compute the target β in the progressive BKZ simulator, we use the correspondence between δ and the GSA constant r: Given a d-dimensional basis, in order to use progressive BKZ simulator, we need target β_t for our parameter choice. At this stage, from the Proposition 2.20 in Section 2.2, we can get the target GSA constant $r_t = \delta^{-4d/(d-1)}$. Therefore we can compute the terminating blocksize β_t in progressive BKZ corresponding to r_t by equation (3.10).

In our case, $d = 2n + 1$ is the dimension of lattice at step 2 in Algorithm 9 and also $\mathrm{Vol}(L_{(\mathbf{A},q)}) = q^n$. Therefore we can adapt inequality (5.3) to

$$\sqrt{\beta \cdot ({\sigma_e}^2 + {\sigma_f}^2)} \leq \delta^{2\beta-2n-1} \cdot q^{n/(2n+1)}. \tag{5.4}$$

Since σ_f can be experimentally derived from σ_e, we can compute lower bound of σ_f in RLWE(n, q, σ_f) which covers security of AES-128/192/256 using equations (5.7), (5.2) and (5.4). Note that f no longer follows discrete Gaussian distribution (histogram shows similar shape as Gaussian distribution). Therefore we take a heuristic approach to estimate σ_f.

Step 2. We compute the complexity of BKZ-β with sieving SVP oracle estimated as $8d \cdot 2^{0.292\beta+16.4}$ double precision floating point operations [15, 3]. We can easily translate this to complexity of bit unit by

$$T_{sieving-BKZ} = 8d \cdot 2^{0.292\beta+16.4} \cdot 64 \ \ (\text{bits}). \tag{5.5}$$

Simultaneously, T_{BKZ} can also be replaced by progressive BKZ simulator explained in section 3.7. We run the progressive BKZ simulator for both $n = 512$ and $n = 1024$ cases. Considering the number of iterations for each fixed blocksize in BKZ, we get following two fitting functions to estimate the runtime of two cases respectively.

$$\log_2(Time_{pBKZ}(secs)) = \begin{cases} 0.003924 \cdot \beta^2 - 0.568 \cdot \beta + 41.93 & (n = 512) \\ 0.004212 \cdot \beta^2 - 0.6886 \cdot \beta + 55.49 & (n = 1024) \end{cases} \tag{5.6}$$

Then we compute the complexity of bit unit by:

$$T_{pBKZ} = Time_{pBKZ} \times 2.7 \times 10^9 \times 64 \ \ (\text{bits}). \tag{5.7}$$

on our Intel(R) Xeon(R) CPU E5-2697 v2 @ 2.70GHz server.

We generate 1,000 and 2,000 $as + 2e$ samples for parameter choice $(n, \sigma, q, p) = (1024, 2.6, 120833, 7552)$ and $(n, \sigma, q, p) = (512, 4.19, 120833, 7552)$. For each sample, we apply Round() and Recover() functions, giving us

$$\mathbf{z} = \text{Recover}(\text{Round}(\mathbf{a} \cdot \mathbf{s} + 2\mathbf{e}, p, q), p, q) = \mathbf{a} \cdot \mathbf{s} + 2\mathbf{f}.$$

With $\frac{\mathbf{z}-\mathbf{a}\mathbf{s}}{2} = \mathbf{f}$, we compute standard deviation σ_f. Results are given in Table 5.2, where the parameter settings can ensure 2^{-60} failure probability.

Table 5.2: Our simulation data and parameter settings covering security of AES-128/192/256

Security level (n, q, σ)	AES-128 (512,120833,4.19)		AES-192 and AES-256 (1024,120833,2.6)	
Method	pBKZ	2016 estimate	pBKZ	2016 estimate
Logarithmic computational complexity	319.14	142.27	1473.09	279.05
Blocksize	330	366	660	831
GSA Const.	0.983		0.991	
σ (for s and e) of our parameter choice	4.19		2.6	
σ_f	4.92		4.72	

Due to the uncertainty simulation for runtime with large dimension and large β (> 1000 and > 200 respectively), we are not sure about the simulation results for our key exchange protocol. We will leave it as future work. However, our parameter choices can cover results from pBKZ simulator. Therefore we show results from pBKZ simulator in Table 5.2 as well.

The 2016 estimate for AES-128 and AES-192/256 security gives 142.27 and 279.05 bit operations respectively. As we discussed in section 5.4.2, the sieving algorithm actually requires exponential large memory $(\beta \cdot 2^{0.292*\beta+o(\beta)})$. Practically such a size memory's access will increase the computation cost by at least one magnitude $(\text{x}10)$, therefore we conclude that our parameter choices with $n = 512$ can achieve at least 145.59 bits security, $n = 1024$ can achieve at least 282.37 bits security. With results given in Table 5.2, we claim that parameter choices given in Ding Key Exchange proposal to NIST cover security of AES-128/192/256.

5.4.5 Statistically Analyzing the Gaussian Distribution and the Hybrid Distribution in DKE

We call the error distribution in the above analysis as "hybrid", which includes the original Gaussian distribution from LWE and the uniform distribution from rounding function. In this section, we will show that for our parameter settings, the vectors sampled from hybrid distribution is no shorter than the one sampled from Gauss (which we used in our analysis). Namely, the RLWE problem with proposed parameters in practical attack is no easier than our analysis based on the pure Gaussian distribution thereof.

Assume a random variable $X' = X + i$, where $X \overset{\$}{\leftarrow} D_{\mathbb{Z}^n,\sigma_1}$ and $i \overset{\$}{\leftarrow} U[-8,7]$ are independent. Note that $\sigma_1 = \sigma/\sqrt{2\pi}$ with σ the parameter for s and e in Ding Key Exchange. Then we can get

$$\mathbb{E}(X^2) = \sigma_1^2 \text{ and } \mathbb{E}(i^2) = \frac{1}{16}((-8)^2 + (-7)^2 + \cdots + 7^2) = 21.5.$$

Due to the Law of Large Numbers (LLN) and Central Limit Theorem (CLT), the expected length of a $n-$dimensional vector $\mathbf{x}$ composed by elements X' is

$$exp(\|\mathbf{x}\|) = \sqrt{n \cdot \mathbb{E}(X'^2)} = \sqrt{n \cdot (\mathbb{E}(X^2) + \mathbb{E}(2iX) + \mathbb{E}(i^2))} = \sqrt{n \cdot (\sigma_1^2 + 21.5)}$$

TABLE 5.3: The expected length of vectors sampled from the hybrid distribution and the Gaussian distribution.

Dimension n	Hybrid	Gauss
512	$4.928 \cdot \sqrt{n}$	$4.92 \cdot \sqrt{n}$
1024	$4.75 \cdot \sqrt{n}$	$4.72 \cdot \sqrt{n}$

Simultaneously, we can also compute the theoretical value for $\sigma_f = \sqrt{\mathbb{V}(X')}$ by the following results.

$$\begin{aligned}
\mathbb{E}(X') &= \mathbb{E}(X) + \mathbb{E}(i) = \frac{1}{16}((-8-7-6\cdots+6+7) = -\frac{1}{2};\\
\mathbb{V}(X') &= \mathbb{E}(X'^2) - \mathbb{E}(X')^2\\
&= \mathbb{E}(X^2 + 2ix + i^2) - \frac{1}{4}\\
&= \sigma_1^2 + 0 + 21.5 - \frac{1}{4}\\
&= \sigma_1^2 + 21.25.
\end{aligned}$$

The theoretical value of σ_f for dimensions 512 and 1024 are 4.903 and 4.725, which are extremely close to the experimental results given in Table 5.2.

5.5 Conclusion

In this chapter, we did cryptanalysis for the Ding key exchange using the progressive BKZ simulator and the : give secure parameter settings under NIST security category I, III and V. We use the Short Integer Solution(SIS) attack and our analysis procedure depending on the progressive BKZ simulation [10] and M.A. estimator [5]. Moreover, the overwhelming storage cost for sieving algorithm is also considered, which is used as SVP oracle of BKZ algorithm in M.A. estimator.

Chapter 6

Conclusion

In this thesis, we focus on the BKZ reduction algorithms and their applications in lattice-based cryptography. Firstly, we proposed an improved progressive BKZ (pBKZ) and a relevant simulating algorithm to estimate its computational cost. The experimental results showed the efficiency of our algorithm and the accuracy of the simulator. Furthermore, we won several Darmstadt challenges, including the SVP Challenge, the Approximate Ideal Lattice Challenge and the LWE Challenge, which indicate the impressive capability of our progressive BKZ. We also estimated the practical hardness of the learning with errors problem (LWE) using this proposed progressive BKZ, and gave a method based on the experimental results to decide the proper number of necessary samples m and the size of the embedding factor M in the attack. Simultaneously, using our method and pBKZ simulator we estimated the computational cost of some rest LWE challenge instances. In the end, we analyzed the Ring LWE based Ding key exchange protocol, which is a proposal to NIST PQC standardization project. To insure the error rate of key exchange protocol under 2^{-60}, we also gave several proper parameter sets providing the security of AES-128/192/256 respectively, which satisfies NIST's security category I, III and V respectively.

Bibliography

[1] M. Ajtai. The worst-case behavior of schnorr's algorithm approximating the shortest nonzero vector in a lattice. In *Proceedings of the 35th Annual ACM Symposium on Theory of Computing, June 9-11, 2003, San Diego, CA, USA*, pages 396–406, 2003.

[2] M. Ajtai, R. Kumar, and D. Sivakumar. A sieve algorithm for the shortest lattice vector problem. In *Proceedings of the Thirty-third Annual ACM Symposium on Theory of Computing*, pages 601–610, 2001.

[3] M. R. Albrecht. On dual lattice attacks against small-secret LWE and parameter choices in helib and SEAL. In *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Proceedings, Part II*, pages 103–129, 2017.

[4] M. R. Albrecht, R. Fitzpatrick, and F. Göpfert. On the efficacy of solving LWE by reduction to unique-svp. In *Information Security and Cryptology - ICISC 2013 - 16th International Conference, Revised Selected Papers*, pages 293–310, 2013.

[5] M. R. Albrecht, F. Göpfert, F. Virdia, and T. Wunderer. Revisiting the expected cost of solving usvp and applications to LWE. In *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Proceedings, Part I*, pages 297–322, 2017.

[6] M. R. Albrecht, R. Player, and S. Scott. On the concrete hardness of learning with errors. *Journal of Mathematical Cryptology*, 9(3):169–203, 2015.

[7] E. Alkim, L. Ducas, T. Pöppelmann, and P. Schwabe. Post-quantum key exchange-a new hope. In *USENIX Security Symposium*, pages 327–343, 2016.

[8] Y. Aono. A faster method for computing gama-nguyen-regev's extreme pruning coefficients. *CoRR*, abs/1406.0342, 2014.

[9] Y. Aono, X. Boyen, L. T. Phong, and L. Wang. Key-private proxy re-encryption under LWE. In *Progress in Cryptology - INDOCRYPT 2013 - 14th International Conference on Cryptology in India, Proceedings*, pages 1–18, 2013.

[10] Y. Aono, Y. Wang, T. Hayashi, and T. Takagi. Improved progressive BKZ algorithms and their precise cost estimation by sharp simulator. In *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Proceedings, Part I*, pages 789–819, 2016.

[11] Y. Aono, Y. Wang, T. Hayashi, and T. Takagi. The progressive bkz code. Available at `http://www2.nict.go.jp/security/pbkzcode/`, 2017.

[12] S. Arora and R. Ge. New algorithms for learning in presence of errors. In *Automata, Languages and Programming - 38th International Colloquium, ICALP 2011, Proceedings, Part I*, pages 403–415, 2011.

[13] L. Babai. On lovász' lattice reduction and the nearest lattice point problem (shortened version). In *STACS 85, 2nd Symposium of Theoretical Aspects of Computer Science, Proceedings*, pages 13–20, 1985.

[14] S. Bai and S. D. Galbraith. Lattice decoding attacks on binary LWE. In *Information Security and Privacy - 19th Australasian Conference, ACISP 2014, Proceedings*, pages 322–337, 2014.

[15] A. Becker, L. Ducas, N. Gama, and T. Laarhoven. New directions in nearest neighbor searching with applications to lattice sieving. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016*, pages 10–24, 2016.

[16] A. Blum, A. Kalai, and H. Wasserman. Noise-tolerant learning, the parity problem, and the statistical query model. *J. ACM*, 50(4):506–519, 2003.

[17] J. Bos, C. Costello, L. Ducas, I. Mironov, M. Naehrig, V. Nikolaenko, A. Raghunathan, and D. Stebila. Frodo: Take off the ring! practical, quantum-secure key

exchange from lwe. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1006–1018. ACM, 2016.

[18] J. W. Bos, C. Costello, M. Naehrig, and D. Stebila. Post-quantum key exchange for the tls protocol from the ring learning with errors problem. In *Security and Privacy (SP), 2015 IEEE Symposium on*, pages 553–570. IEEE, 2015.

[19] J. W. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, and D. Stehlé. CRYSTALS - kyber: a cca-secure module-lattice-based KEM. *IACR Cryptology ePrint Archive*, 2017:634, 2017.

[20] Z. Brakerski and V. Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011*, pages 97–106, 2011.

[21] J. A. Buchmann, N. Büscher, F. Göpfert, S. Katzenbeisser, J. Krämer, D. Micciancio, S. Siim, C. van Vredendaal, and M. Walter. Creating cryptographic challenges using multi-party computation: The LWE challenge. In *Proceedings of the 3rd ACM International Workshop on ASIA Public-Key Cryptography, AsiaPKC@AsiaCCS*, pages 11–20, 2016.

[22] J. A. Buchmann and C. Ludwig. Practical lattice basis sampling reduction. In *Algorithmic Number Theory, 7th International Symposium, ANTS-VII, Berlin, Germany, July 23-28, 2006, Proceedings*, pages 222–237, 2006.

[23] Y. Chen. Lattice reduction and concrete security of fully homomorphic encryption. *Dept. Informatique, ENS, Paris, France, PhD thesis*, 2013.

[24] Y. Chen and P. Q. Nguyen. Bkz 2.0: Better lattice security estimates. In *Advances in Cryptology – ASIACRYPT 2011: 17th International Conference on the Theory and Application of Cryptology and Information Security, Proceedings*, pages 1–20, 2011.

[25] Y. Chen and P. Q. Nguyen. Bkz 2.0: Better lattice security estimates. the full version, available at `http://www.di.ens.fr/~ychen/research/Full_BKZ.pdf.`, 2012.

[26] D. Coppersmith and A. Shamir. Lattice attacks on NTRU. In *Advances in Cryptology - EUROCRYPT '97, International Conference on the Theory and Application of Cryptographic Techniques, Proceeding*, pages 52–61, 1997.

[27] T. Darmstadt. Learning with errors challenge. Available at `https://www.latticechallenge.org/lwe_challenge`, 2017.

[28] T. Darmstadt. Svp challenge. Available at `https://www.latticechallenge.org/svp-challenge`, 2017.

[29] T. Darmstadt. Ideal lattice challenge. Available at `https://latticechallenge.org/ideallattice-challenge`, 2018.

[30] T. Darmstadt. Lattice challenge. Available at `https://latticechallenge.org`, 2018.

[31] J. Ding, T. Takagi, X. Gao, and W. Yuntao. Ding key exchange – a proposal to nist pqc competition. SCIS 2018, 2018.

[32] P. D. Domich, R. Kannan, and L. E. T. Jr. Hermite normal form computation using modulo determinant arithmetic. *Math. Oper. Res.*, 12(1):50–59, 1987.

[33] D. FermiGuy. The number of atoms in the world. Available at `http://www.fnal.gov/pub/science/inquiring/questions/atoms.html`, 2014.

[34] R. Fischlin and J. Seifert. Tensor-based trapdoors for CVP and their application to public key cryptography. In *Cryptography and Coding, 7th IMA International Conference, Cirencester, UK, Proceedings*, pages 244–257, 1999.

[35] M. Fukase and K. Kashiwabara. An accelerated algorithm for solving SVP based on statistical analysis. *JIP*, 23(1):67–80, 2015.

[36] N. Gama and P. Q. Nguyen. Predicting lattice reduction. In *Advances in Cryptology - EUROCRYPT 2008, 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Proceedings*, pages 31–51, 2008.

[37] N. Gama, P. Q. Nguyen, and O. Regev. Lattice enumeration using extreme pruning. In *Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Proceedings*, pages 257–278, 2010.

[38] S. Garg, C. Gentry, and S. Halevi. Candidate multilinear maps from ideal lattices. In *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Proceedings*, pages 1–17, 2013.

[39] C. Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009*, pages 169–178, 2009.

[40] C. Gentry, C. Peikert, and V. Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing*, pages 197–206, 2008.

[41] D. Gil. The future is quantum. Available at `https://www.ibm.com/blogs/research/2017/11/the-future-is-quantum/`, 2017.

[42] G. Hanrot, X. Pujol, and D. Stehlé. Analyzing blockwise lattice algorithms using dynamical systems. In *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, Proceedings*, pages 447–464, 2011.

[43] M. Haque, M. O. Rahman, and J. Pieprzyk. Analysing progressive-bkz lattice reduction algorithm. In *1st National Conference on Intelligent Computing & Information Technology, NCICIT 2013, Proceedings*, pages 73–80, 2013.

[44] G. Herold, E. Kirshanova, and A. May. On the asymptotic complexity of solving LWE. *IACR Cryptology ePrint Archive*, 2015:1222, 2015.

[45] J. Hoffstein, J. Pipher, and J. H. Silverman. NTRU: A ring-based public key cryptosystem. In *Algorithmic Number Theory, Third International Symposium, ANTS-III, Proceedings*, pages 267–288, 1998.

[46] J. Hoffstein and J. H. Silverman. Random small hamming weight products with applications to cryptography. *Discrete Applied Mathematics*, 130(1):37–49, 2003.

[47] T. Ishiguro, S. Kiyomoto, Y. Miyake, and T. Takagi. Parallel gauss sieve algorithm: Solving the SVP challenge over a 128-dimensional ideal lattice. In *Public-Key Cryptography - PKC 2014 - 17th International Conference on Practice and Theory in Public-Key Cryptography, Proceedings*, pages 411–428, 2014.

[48] R. Kannan. Improved algorithms for integer programming and related lattice problems. In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing*, pages 193–206, 1983.

[49] R. Kannan. Minkowski's convex body theorem and integer programming. *Math. Oper. Res.*, 12(3):415–440, 1987.

[50] P. Kirchner and P. Fouque. An improved BKW algorithm for LWE with applications to cryptography and lattices. In *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Proceedings, Part I*, pages 43–62, 2015.

[51] T. Laarhoven. Sieving for shortest vectors in lattices using angular locality-sensitive hashing. In *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Proceedings, Part I*, pages 3–22, 2015.

[52] T. Laarhoven and B. de Weger. Faster sieving for shortest lattice vectors using spherical locality-sensitive hashing. In *Progress in Cryptology - LATINCRYPT 2015 - 4th International Conference on Cryptology and Information Security in Latin America, Proceedings*, pages 101–118, 2015.

[53] A. Lenstra, H. Lenstra, and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261:515–534, 1982.

[54] T. Lepoint and M. Naehrig. A comparison of the homomorphic encryption schemes FV and YASHE. In *Progress in Cryptology - AFRICACRYPT 2014 - 7th International Conference on Cryptology in Africa, Proceedings*, pages 318–335, 2014.

[55] R. Lindner and C. Peikert. Better key sizes (and attacks) for lwe-based encryption. In *Topics in Cryptology - CT-RSA 2011 - The Cryptographers' Track at the RSA Conference 2011, Proceedings*, pages 319–339, 2011.

[56] M. Liu and P. Q. Nguyen. Solving BDD by enumeration: An update. In *Topics in Cryptology - CT-RSA 2013 - The Cryptographers' Track at the RSA Conference 2013, Proceedings*, pages 293–309, 2013.

[57] V. Lyubashevsky. The parity problem in the presence of noise, decoding random linear codes, and the subset sum problem. In *Approximation, Randomization and*

Combinatorial Optimization, Algorithms and Techniques, 8th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems, APPROX 2005 and 9th InternationalWorkshop on Randomization and Computation, RANDOM 2005, Proceedings, pages 378–389, 2005.

[58] V. Lyubashevsky and D. Micciancio. On bounded distance decoding, unique shortest vectors, and the minimum distance problem. In *Advances in Cryptology - CRYPTO 2009, 29th Annual International Cryptology Conference, Proceedings*, pages 577–594, 2009.

[59] V. Lyubashevsky, C. Peikert, and O. Regev. On ideal lattices and learning with errors over rings. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 1–23. Springer, 2010.

[60] D. Micciancio and O. Regev. *Lattice-based Cryptography*, pages 147–191. Springer, 2009.

[61] D. Micciancio and M. Walter. Practical, predictable lattice basis reduction. In *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Proceedings, Part I*, pages 820–849, 2016.

[62] P. Q. Nguyen. Cryptanalysis of the goldreich-goldwasser-halevi cryptosystem from crypto ’97. In *Advances in Cryptology - CRYPTO ’99, 19th Annual International Cryptology Conference, Proceedings*, pages 288–304, 1999.

[63] P. Q. Nguyen and O. Regev. Learning a parallelepiped: Cryptanalysis of GGH and NTRU signatures. In *Advances in Cryptology - EUROCRYPT 2006, 25th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Proceedings*, pages 271–288, 2006.

[64] P. Q. Nguyen and D. Stehlé. LLL on the average. In *Algorithmic Number Theory, 7th International Symposium, ANTS-VII, Proceedings*, pages 238–256, 2006.

[65] P. Q. Nguyen and B. Vallée, editors. *The LLL Algorithm - Survey and Applications*. Information Security and Cryptography. Springer, 2010.

[66] T. Nguyen. What is the world's data storage capacity? Available at `http://www.zdnet.com/article/what-is-the-worlds-data-storage-capacity`, 2011.

[67] NIST. Post-Quantum Cryptography | CSRC. Available at `https://csrc.nist.gov/projects/post-quantum-cryptography`, 2017.

[68] L. T. Phong, T. Hayashi, Y. Aono, and S. Moriai. Lotus: Algorithm specifications and supporting documentation. Available at `https://www2.nict.go.jp/security/lotus/index.html`, 2017.

[69] O. Regev. On lattices, learning with errors, random linear codes, and cryptography. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing*, pages 84–93, 2005.

[70] D. Reinsel, J. Gantz, and J. Rydning. Data age 2025: The evolution of data to life-critical. Available at `http://www.seagate.com/files/www-content/our-story/trends/files/Seagate-WP-DataAge2025-March-2017.pdf`, 2017.

[71] C. A. Rogers. The number of lattice points in a set. In *Proceedings of the London Mathematical Society*, volume s3-6, 1956.

[72] M. Schmidt and N. Bindel. Estimation of the hardness of the learning with errors problem with a restricted number of samples. *IACR Cryptology ePrint Archive*, 2017:140, 2017.

[73] C. Schnorr. Lattice reduction by random sampling and birthday methods. In *STACS 2003, 20th Annual Symposium on Theoretical Aspects of Computer Science, Proceedings*, pages 145–156, 2003.

[74] C. Schnorr. Accelerated slide- and lll-reduction. *Electronic Colloquium on Computational Complexity (ECCC)*, 18:50, 2011.

[75] C. Schnorr and M. Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. *Math. Program.*, 66:181–199, 1994.

[76] C. Schnorr and H. H. Hörner. Attacking the chor-rivest cryptosystem by improved lattice reduction. In *Advances in Cryptology - EUROCRYPT '95, International*

Conference on the Theory and Application of Cryptographic Techniques, Proceeding, pages 1–12, 1995.

[77] C. Schnorr and T. Shevchenko. Solving subset sum problems of densioty close to 1 by "randomized" bkz-reduction. *IACR Cryptology ePrint Archive*, 2012:620, 2012.

[78] P. W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 124–134, 1994.

[79] V. Shoup. NTL, a library for doing number theory. Available at `http://www.shoup.net/ntl/`, 2017.

[80] N. Stephens-Davidowitz. Discrete gaussian sampling reduces to cvp and svp. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016*, pages 1748–1764, 2016.

[81] J. van de Pol and N. P. Smart. Estimating key sizes for high dimensional lattice-based systems. In *Cryptography and Coding - 14th IMA International Conference, IMACC 2013, Proceedings*, pages 290–303, 2013.

[82] Y. Wang, Y. Aono, and T. Takagi. An experimental study of kannan's embedding technique for the search LWE problem. In *Information and Communications Security - 19th International Conference, ICICS 2017, Proceedings*, pages 541–553, 2017.

[83] R. Xu. Private communication. 2017.

[84] R. Xu, S. L. Yeo, K. Fukushima, T. Takagi, H. Seo, S. Kiyomoto, and M. Henricksen. An experimental study of the BDD approach for the search LWE problem. In *Applied Cryptography and Network Security - 15th International Conference, ACNS 2017, Proceedings*, pages 253–272, 2017.

[85] S. Yang, P. Kuo, B. Yang, and C. Cheng. Gauss sieve algorithm on gpus. In *Topics in Cryptology - CT-RSA 2017 - The Cryptographers' Track at the RSA Conference 2017, Proceedings*, pages 39–57, 2017.