On the Success Probability of Solving Unique SVP via BKZ

Eamonn W. Postlethwaite[♯♭], Fernando Virdia[♯♭]

Information Security Group,
Royal Holloway, University of London,
Egham, UK
eamonn.postlethwaite.2016@rhul.ac.uk
fernando.virdia.2016@rhul.ac.uk

Abstract. As lattice-based key encapsulation, digital signature, and fully homomorphic encryption schemes near standardisation, ever more focus is being directed to the precise estimation of the security of these schemes. The primal attack reduces key recovery against such schemes to instances of the unique Shortest Vector Problem (uSVP). Dachman-Soled *et al.* (Crypto 2020) recently proposed a new approach for fine-grained estimation of the cost of the primal attack when using Progressive BKZ for the lattice reduction step. In this paper we review and extend their technique to BKZ 2.0 and provide extensive experimental evidence of its accuracy. Using this technique we also explain results from previous primal attack experiments by Albrecht *et al.* (Asiacrypt 2017) where attacks succeeded with smaller than expected block sizes. Finally, we use our simulators to reestimate the cost of attacking the three lattice finalists of the NIST Post Quantum Standardisation Process.

Keywords: cryptanalysis, lattice-based cryptography, lattice reduction

1 Introduction

In recent years, the popularity of lattice-based cryptography has greatly increased. Lattices have been used to design traditional cryptographic primitives such as one way functions, public key encryption, key exchange, digital signatures, as well as more advanced constructions such as identity and attribute based encryption, and fully homomorphic encryption.

One reason for this popularity is that lattice problems, e.g. the Shortest Vector Problem (SVP) and Bounded Distance Decoding (BDD), are believed to be hard for both classical and quantum computers. Hence, schemes based on such

[♯] This work was supported by the EPSRC and the UK government as part of the Centre for Doctoral Training in Cyber Security at Royal Holloway, University of London (EP/P009301/1).

[♭] This work was carried out in part while the authors were visiting the Lattices: Algorithms, Complexity, and Cryptography program at the Simons Institute for the Theory of Computing.

problems are good candidates for providing quantum-safe public key cryptography. Indeed, 23 of the original 69 *complete and proper* schemes submitted to the National Institute of Standards and Technology (NIST) as part of the Post Quantum Standardisation Process [NIS16] are based on various lattice problems with varying amounts of structure. Given the long shelf life of cryptographic standards and the high stakes of standardising primitives, the security of these schemes, and thus the concrete hardness of lattice problems, should be understood in detail.

Two popular problems chosen to design lattice-based schemes are the Learning With Errors (LWE) problem (with its ring and module variants) and the NTRU problem. A variety of attack strategies against these problems exist. Asymptotically, the best option is the approach of Arora–Ge [AG11], while, again asymptotically, in the case of binary secrets, BKW variants [KF15,GJS15] perform well. In practice however, the best attacks seem to be the *primal*, *dual* and *hybrid* attacks. All three rely on lattice reduction algorithms, such as BKZ [SE91,SE94], BKZ 2.0 [CN11], Progressive BKZ [AWHT16], Self-Dual BKZ [MW16], G6K [ADH+19] and Slide Reduction [GN08a], to find either a unique (up to sign) embedded shortest vector, or more generally a good lattice basis. In particular, the primal attack is often estimated as the cheapest option [ACD+18].

The primal attack against LWE and NTRU consists of using lattice reduction to solve an instance of the unique Shortest Vector Problem (uSVP). The most popular lattice reduction algorithm is BKZ. Current complexity estimates for solving uSVP directly depend on estimating the smallest block size β such that BKZ-β successfully recovers the unique shortest vector. This β is commonly found by following the methodology introduced in [ADPS16, §6.3], and experimentally investigated in [AGVW17].

In their experiments, Albrecht *et al.* [AGVW17], and later Bai *et al.* [BMW19], report that smaller than expected block sizes can result in a non-negligible probability of solving uSVP instances arising from the primal attack, when using BKZ. Some concerns were raised [BCLv19] that this could indicate an overestimate of the complexity of the primal attack for cryptographically sized instances. Furthermore, the experiments carried out in 2017 [AGVW17] only focused on recovering a unique shortest vector sampled coefficientwise from a discrete Gaussian distribution. While [AGVW17] claims that the [ADPS16] methodology would also hold for binary and ternary distributions, the authors do not provide experimental evidence. Recent work [CCLS20] revisited the binary and ternary case in the small block size regime $\beta \leq 45$ and observed higher success probabilities than predicted.

Dachman-Soled *et al.* [DSDGR20] recently proposed an approach for estimating the complexity of the primal attack that makes use of probability distributions for the norms of particular projections of the unique shortest vector, rather than only expected values. This results in a new approach that allows one to better predict the behaviour of the attack when considering block sizes smaller than those expected to be successful by the [ADPS16] methodology. The

authors of [DSDGR20] use this approach to develop a simulator that predicts the expected block size by which Progressive BKZ will solve an isotropic uSVP instance. In this work, we call such a simulator a *uSVP simulator*. They use this in the setting of solving LWE instances with extra hints about the secret, and verify the accuracy of their predictions as the number of hints varies.

Our contributions. Our first contribution is the implementation of a variant of the uSVP simulator for Progressive BKZ, and the development of a new uSVP simulator for BKZ 2.0. Rather than only returning the expected successful block size, we extract full probability mass functions for successful block sizes, which allow for a more direct comparision to experimental results. Our simulators are also faster than that in [DSDGR20], allowing for potentially easier inclusion in parameter selection scripts, such as the LWE estimator [APS15].

Our second contribution is extensive experiments on the success probability of different block sizes for BKZ 2.0 and Progressive BKZ, on uSVP lattices generated from LWE instances with discrete Gaussian, binary or ternary secret and error distributions. Our experiments show that the uSVP simulators accurately predict the block sizes needed to solve uSVP instances via lattice reduction, for all distributions tested.

As a final contribution, we reestimate the security of the three lattice based NIST PQC finalists using our uSVP simulators. We compare the expected block sizes they suggest to those predicted by the original methodology of [ADPS16]. We note that our uSVP simulators estimate that a slightly larger average block size than predicted is required, meaning that [ADPS16] likely resulted in an underestimate of their security.[1] We also observe that this phenomenon can, in large part, be attributed to the original [ADPS16] methodology using the Geometric Series Assumption. Replacing this assumption with the output of the [CN11] BKZ simulator reduces the predictive gap between the [ADPS16] methodology and our uSVP simulators.

All our code and data can be found at `github.com/fvirdia/usvp-simulation`.

Related work. The Geometric Series Assumption (GSA), used to predict the output quality of lattice reduction, was introduced in [Sch03]. A simulator, specifically for the output quality of BKZ, was introduced in [CN11]. This simulator more accurately predicts the final, or *tail*, region of the basis profile of a BKZ reduced lattice, improving over the GSA. A refined BKZ simulator was presented in [BSW18], which improves over the [CN11] simulator in the first region, or *head*, of the basis profile. Alkim *et al.* [ADPS16] introduced a BKZ specific method for estimating the block size required to solve uSVP instances arising from the primal attack, the experimental accuracy of which was investigated in [AGVW17]. This method, combined with basis profile simulation after BKZ reduction and arguments about distributions describing the lengths of projections of the unique

[1] A similar phenomenon had also been observed in [DSDGR20] for NTRU-HPS.

short vector, is extended in [DSDGR20] to predict the expected block size by which Progressive BKZ will solve isotropic uSVP instances.

Paper structure. In Section 2 we introduce the necessary preliminaries and notation regarding linear algebra, lattice computational problems, and lattice reduction. In Section 3 we review the original [ADPS16] methodology for predicting the expected required block sizes for solving uSVP instances. In Section 4 we review the approach of [DSDGR20] and use it to propose uSVP simulators for BKZ 2.0 and Progressive BKZ. In Section 5 we describe our experiments and results. In Section 6 we use our uSVP simulators to provide preliminary estimates of the block sizes required to successfully perform key recovery attacks on the three NIST PQC lattice finalists, and compare this to predictions using the [ADPS16] methodology.

2 Preliminaries

Linear algebra. The set $\{1, \ldots, n\}$ is denoted by $[n]$. We denote vectors by bold lowercase letters such as $\boldsymbol{v}$, and matrices by bold uppercase letters such as $\boldsymbol{M}$. We denote the $n \times n$ identity matrix as $\boldsymbol{I}_n$. Throughout, we use row or column vectors as more convenient, and count indices from 1. We represent a basis $\{\boldsymbol{b}_1, \ldots, \boldsymbol{b}_d\}$ of $\mathbb{R}^d$ as the matrix $\boldsymbol{B}$ having the basis vectors as rows. Given a basis $\boldsymbol{B}$, we can derive an orthogonal basis $\boldsymbol{B}^*$ via the Gram–Schmidt process. The rows of $\boldsymbol{B}^*$ are

$$\boldsymbol{b}_i^* = \boldsymbol{b}_i - \sum_{j<i} \mu_{i,j} \boldsymbol{b}_j^* \quad \text{for} \quad i \in [d], \quad \text{where} \quad \mu_{i,j} = \langle \boldsymbol{b}_i, \boldsymbol{b}_j^* \rangle / \|\boldsymbol{b}_j^*\|^2 \quad \text{for} \quad i > j.$$

We denote by $\text{span}_{\mathbb{R}}(\{\boldsymbol{v}_i\}_i) = \{\sum_i \lambda_i \boldsymbol{v}_i \colon \lambda_i \in \mathbb{R}\}$ the real span of a set of real vectors $\{\boldsymbol{v}_i\}_i$. Given a basis $\boldsymbol{B}$ of $\mathbb{R}^d$ we denote by $\pi_{\boldsymbol{B},k} \colon \mathbb{R}^d \to \mathbb{R}^d$ the linear operator projecting vectors orthogonally to the subspace $\text{span}_{\mathbb{R}}(\{\boldsymbol{b}_1, \ldots, \boldsymbol{b}_{k-1}\})$. Note $\pi_{\boldsymbol{B},1}$ is the identity on $\mathbb{R}^d$. We write π_i when the basis is clear from context. Given a vector space $V = \text{span}_{\mathbb{R}}(\boldsymbol{B})$, its projective subspace $\pi_k(V)$ of dimension $d - k + 1$ has a basis $\{\pi_k(\boldsymbol{b}_k), \ldots, \pi_k(\boldsymbol{b}_d)\}$, where

$$\pi_k(\boldsymbol{b}_i) = \boldsymbol{b}_i - \sum_{j<k} \mu_{i,j} \boldsymbol{b}_j^* = \boldsymbol{b}_i^* + \sum_{k \le j < i} \mu_{i,j} \boldsymbol{b}_j^* \quad \text{for} \quad i \ge k.$$

By definition, this implies that $\pi_k(\boldsymbol{b}_k) = \boldsymbol{b}_k^*$, and that $\pi_j(\pi_k(\boldsymbol{v})) = \pi_k(\boldsymbol{v})$ for any $j \le k$. Given an orthogonal basis $\boldsymbol{B}^*$ and a vector $\boldsymbol{t} = t_1^* \boldsymbol{b}_1^* + \cdots + t_d^* \boldsymbol{b}_d^*$, its projections are given by $\pi_k(\boldsymbol{t}) = t_k^* \boldsymbol{b}_k^* + \cdots + t_d^* \boldsymbol{b}_d^*$. We abuse notation and write $\pi_i(\boldsymbol{B}[j:k])$ to mean the matrix with rows $\pi_i(\boldsymbol{b}_j), \ldots, \pi_i(\boldsymbol{b}_k)$.

Probability. Given a probability distribution D with support $S \subset \mathbb{R}$, we denote sampling an element $s \in S$ according to D as $s \leftarrow D$. For a finite support S, we denote the uniform distribution over S as $\mathcal{U}(S)$. We denote the mean and

variance of D as $\mathbb{E}(s)$ or $\mathbb{E}(D)$, and $\mathbb{V}(s)$ or $\mathbb{V}(D)$, respectively. Sometimes we use $\sqrt{\mathbb{V}}$ to denote the standard deviation. Given a discrete (resp. continuous) probability distribution D, we denote its probability mass function (resp. probability density function) as f_D and its cumulative mass function (resp. cumulative density function) as F_D. Given $s \leftarrow D$, by definition $P[s \leq x] = F_D(x)$. We recall the conditional probability chain rule. If E_1, ..., E_n are events, then

$$P[E_1 \cap \cdots \cap E_n] = P[E_1 | E_2 \cap \cdots \cap E_n] P[E_2 \cap \cdots \cap E_n].$$

We denote by Γ the gamma function

$$\Gamma(x) = \int_0^\infty t^{x-1} e^{-t} dt \quad \text{for} \quad x > 0.$$

The Gaussian distribution. We recall some properties of the continuous Gaussian distribution. We denote by $N(\mu, \sigma^2)$ the probability distribution over $\mathbb{R}$ of mean μ and standard deviation σ (i.e. variance σ^2) with probability density function

$$f_{N(\mu,\sigma^2)}(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}.$$

Given a random variable $X \sim N(\mu_X, \sigma_X^2)$ and a scalar $\lambda > 0$, the random variable $Y = \lambda \cdot X$ follows a distribution $N(\lambda\mu_X, \lambda^2\sigma_X^2)$. Given n independent and identically distributed random variables $X_i \sim N(0, 1)$, the random variable $X_1^2 + \cdots + X_n^2$ follows a chi-squared distribution χ_n^2 over $\mathbb{R}_{\geq 0}$ of mean n and variance $2n$, with probability density function

$$f_{\chi_n^2}(x) = \frac{1}{2^{n/2}\Gamma(n/2)}\, x^{n/2-1} e^{-x/2}.$$

Given n independent and identically distributed random variables $Y_i \sim N(0, \sigma^2)$, the random variable $Y_1^2 + \cdots + Y_n^2$ follows a distribution $\sigma^2 \cdot \chi_n^2$ of mean $n\sigma^2$ and variance $2n\sigma^4$, that is, a chi-squared distribution where every sample is scaled by a factor σ^2. We call this a *scaled* chi-squared distribution.

Discrete Gaussians. We denote by $D_{\mu,\sigma}$ the discrete Gaussian distribution over $\mathbb{Z}$ with mean $\mu \in \mathbb{R}$ and standard deviation $\sigma \in \mathbb{R}^+$. It has probability mass function $f_{D_{\mu,\sigma}} \colon \mathbb{Z} \to [0, 1], x \mapsto f_{N(\mu,\sigma^2)}(x)/f_{N(\mu,\sigma^2)}(\mathbb{Z})$, where $f_{N(\mu,\sigma^2)}(\mathbb{Z}) = \sum_{x\in\mathbb{Z}} f_{N(\mu,\sigma^2)}(x)$. Discrete Gaussian distributions with $\mu = 0$, or the distributions these imply over $\mathbb{Z}_q$ for some modulus q, are widely used in lattice cryptography to sample entries of error and secret vectors from. In our analyses below, we work with vectors $\boldsymbol{t}$ sampled coefficientwise from a discrete Gaussian, and with their projections $\pi_i(\boldsymbol{t})$. We model the the squared norms $\|\pi_i(\boldsymbol{t})\|^2$ as random variables following a scaled chi-squared distribution with the appropriate degrees of freedom. For example, for some vector $\boldsymbol{v} = (v_1, \ldots, v_d)$ with each $v_i \leftarrow D_{0,\sigma}$ sampled independently, we model $\|\pi_{\boldsymbol{B},i}(\boldsymbol{v})\|^2 \sim \sigma^2 \cdot \chi_{d-i+1}^2$, where $\boldsymbol{B}$ is a lattice basis obtained during lattice reduction run as part of the primal attack.

Bounded uniform distributions. Given a finite subset $S \subset \mathbb{Z}$, we call the uniform distribution $\mathcal{U}(S)$ a *bounded uniform* distribution. Of particular interest in this work are the binary and ternary distributions, where $S = \{0, 1\}$ and $S = \{-1, 0, 1\}$. Similarly to the case of the discrete Gaussian, works using the [ADPS16] methodology for estimating the complexity of lattice reduction, such as the "LWE estimator" [APS15], implicitly model $\|\pi_{\boldsymbol{B},i}(\boldsymbol{v})\|^2 \sim \sigma^2 \cdot \chi^2_{d-i+1}$ for vectors $\boldsymbol{v}$ sampled coefficientwise from a bounded uniform distribution having $\mathbb{E}(\mathcal{U}(S)) = 0$ and $\mathbb{V}(\mathcal{U}(S)) = \sigma^2$, and $\boldsymbol{B}$ a lattice basis being reduced.

Lattices. A real lattice of rank n and dimension d is the integer span of n linearly independent vectors $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n \in \mathbb{R}^d$, which we collect into a basis $\boldsymbol{B}$. The lattice generated by $\boldsymbol{B}$ is

$$\Lambda = \Lambda(\boldsymbol{B}) = \{x_1\boldsymbol{b}_1 + \cdots + x_n\boldsymbol{b}_n \colon x_i \in \mathbb{Z}\},$$

and is a discrete subgroup of $(\mathbb{R}^d, +)$. For $n \geq 2$ and $\Lambda = \Lambda(\boldsymbol{B})$, we have also $\Lambda = \Lambda(\boldsymbol{UB})$ for any $\boldsymbol{U} \in \mathrm{GL}_n(\mathbb{Z})$, meaning that Λ has multiple bases related by multiplication of unimodular matrices. An invariant of a lattice is its volume.

Definition 1 (Lattice volume). *Given any basis $\boldsymbol{B}$ for a lattice Λ,*

$$\mathrm{vol}(\Lambda) = \sqrt{\det(\boldsymbol{B}^{\boldsymbol{t}}\boldsymbol{B})} = \prod_{i=1}^{n} \|\boldsymbol{b}_i^*\|.$$

This quantity is exactly the volume of a fundamental parallelepiped of Λ, given by the set $\{\boldsymbol{xB} \colon \boldsymbol{x} \in [0, 1)^n\}$. Other properties of interest in lattices are their minima.

Definition 2 (Lattice minima). *Let $B_d(r)$ be the closed ball of radius r in $\mathbb{R}^d$ and $i \in [n]$. Define $\lambda_i(\Lambda)$, the i^{th} minima of Λ,*

$$\lambda_i(\Lambda) = \min\left\{r \in \mathbb{R}^+ \colon \Lambda \cap B_d(r) \text{ contains } i \text{ linearly independent vectors}\right\}.$$

A lattice can be tessellated by centring a copy of the fundamental domain on each lattice point. This fact can be used to approximate the number of lattice points in some 'nice enough' measurable set. The Gaussian heuristic says that the number of lattice points in a measurable set S is approximately $\mathrm{vol}(S)/\mathrm{vol}(\Lambda)$. The Gaussian heuristic can be used to approximate the first minimum $\lambda_1(\Lambda)$.

Definition 3 (Gaussian heuristic for the shortest vector). *Given a rank n lattice Λ, the Gaussian heuristic approximates the smallest radius containing a lattice point as*

$$\mathrm{gh}(\Lambda) = \sqrt{\frac{n}{2\pi e}}\mathrm{vol}(\Lambda)^{1/n}.$$

Various computational problems can be defined using lattices. We focus on the following.

Definition 4 (Shortest Vector Problem (SVP)). *Given a lattice Λ find a vector $\boldsymbol{v} \in \Lambda$ of length $\lambda_1(\Lambda)$.*

Definition 5 (γ-unique Shortest Vector Problem (uSVP$_\gamma$)). *Given a lattice Λ such that $\lambda_2(\Lambda) > \gamma\lambda_1(\Lambda)$, find the unique (up to sign) $\boldsymbol{v} \in \Lambda$ of length $\lambda_1(\Lambda)$. Unless specified, $\gamma = 1$.*

Definition 6 (Learning With Errors (LWE) [Reg09]). *Let n, q be positive integers, χ be a probability distribution on $\mathbb{Z}_q$ and $\boldsymbol{s}$ be a secret vector in $\mathbb{Z}_q^n$. We denote by $L_{\boldsymbol{s},\chi}$ the probability distribution on $\mathbb{Z}_q^n \times \mathbb{Z}_q$ obtained by sampling $\boldsymbol{a} \leftarrow \mathcal{U}(\mathbb{Z}_q^n)$, $e \leftarrow \chi$, and returning $(\boldsymbol{a}, c) = (\mathbf{a}, \langle \mathbf{a}, \mathbf{s} \rangle + e) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$.*
Decision LWE *is the problem of deciding whether pairs $(\mathbf{a}, c) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$ are sampled according to $L_{\mathbf{s},\chi}$ or $\mathcal{U}(\mathbb{Z}_q^n \times \mathbb{Z}_q)$.*
Search LWE *is the problem of recovering $\boldsymbol{s}$ from pairs sampled according to $L_{\mathbf{s},\chi}$. For a given distribution $L_{\mathbf{s},\chi}$ and power-of-prime modulus q,* Decision LWE *and* Search LWE *are polynomial time equivalent [Reg09].*

We note that the distribution χ from which the error is drawn tends to encode some notion of *smallness*, which is usually required for functionality. Throughout this work, we assume m LWE samples $\{(\boldsymbol{a}_i, c_i) \leftarrow L_{\mathbf{s},\chi}\}_{i=1}^m$ are available. These can be written in matrix form as $(\boldsymbol{A}, \boldsymbol{c}) = (\boldsymbol{A}, \boldsymbol{A}\boldsymbol{s} + \boldsymbol{e}) \in \mathbb{Z}_q^{m \times n} \times \mathbb{Z}_q^m$. In the original formulation, the LWE secret vector is sampled uniformly from $\mathbb{Z}_q^n$. A standard transformation [MR09,ACPS09] maps m samples from an LWE distribution $L_{\mathbf{s},\chi}$ with $\boldsymbol{s} \leftarrow \mathcal{U}(\mathbb{Z}_q^n)$ to $m - n$ samples from an LWE distribution $L_{\boldsymbol{s}',\chi}$ where the secret vector $\boldsymbol{s}'$ is sampled coefficientwise from χ. Such a distribution is said to be in *normal form*. In general, more efficient key exchange can be built from LWE distributions where the secret is sampled from a narrow distribution such as χ (*small secret* LWE) or from a distribution imposing or implying few non zero entries in $\boldsymbol{s}$ (*sparse secret* LWE). In this work χ_s (resp. χ_e) represents the distribution from which coefficients of $\boldsymbol{s}$ (resp. $\boldsymbol{e}$) are sampled. Note that with high probability any n samples $(\boldsymbol{A}, \boldsymbol{c})$ from an LWE distribution with $\boldsymbol{s} \leftarrow \chi_s^n$ and $\boldsymbol{e} \leftarrow \chi_e^n$ can be turned into n LWE samples $(\boldsymbol{A}^{-1}, \boldsymbol{A}^{-1}\boldsymbol{c})$ where the roles of χ_e and χ_s are swapped. This can be useful when creating embedding lattices (see below) while choosing to use $m \leq n$ samples.

Embedding lattices. The primal attack transforms the Search LWE problem into a uSVP instance. This can always be achieved using Kannan's embedding [Kan87]. In the case of small secret LWE, the Bai–Galbraith embedding variant [BG14] exists, which can also exploit differences in χ_s and χ_e, whenever the former is small or sparse. In particular, given LWE samples $(\boldsymbol{A}, \boldsymbol{c})$ in such an instance, the primal attack starts by constructing the following embedding lattice basis

$$\boldsymbol{B} = \begin{pmatrix} \mathbf{0} & q\mathbf{I}_m & \mathbf{0} \\ \nu\mathbf{I}_n & -\mathbf{A}^\top & \mathbf{0} \\ \mathbf{0} & \mathbf{c} & c \end{pmatrix} \tag{1}$$

and performs lattice reduction to recover the unique shortest vector $\boldsymbol{t} = (* \mid \boldsymbol{s} \mid 1) \cdot \boldsymbol{B} = (\nu\,\boldsymbol{s} \mid \boldsymbol{e} \mid c)$ for suitable values of $*$ and c, and a scalar ν that balances

the contributions of $\boldsymbol{s}$ and $\boldsymbol{e}$ to the norm of $\boldsymbol{t}$. An alternative approach is to first reduce the $(n+m) \times (n+m)$ top left minor of $\boldsymbol{B}$ as a form of preprocessing (e.g. if $\boldsymbol{A}$ is a common reference string for multiple LWE distributions), and later append the last row to finish the search for a specific target vector [LN13]. While lattice reduction software that takes $\boldsymbol{B}$ as input will require that $\nu \in \mathbb{Z}$, in Appendix A we discuss a standard way to construct variants of this embedding that allow us to use in practice any $\nu \in \mathbb{R}$, as well as allowing us to centre the χ_s and χ_e distributions. For example, applying these techniques to an LWE instance with a binary secret distribution results in an embedding where the first n coordinates of $\boldsymbol{t}$ are distributed uniformly in $\{-1, 1\}$.

Lattice reduction. In general, lattice reduction is any algorithmic technique that takes as input a basis of a lattice and finds a basis of better quality. Many different notions of reduced basis exist, most of which can be intuitively captured by a basis being formed of short and close to orthogonal vectors. The celebrated LLL algorithm [LLL82] achieves the following.

Definition 7 (LLL reduced). *For $\delta \in (1/4, 1)$ a basis $\boldsymbol{B}$ is δ-LLL reduced if $|\mu_{i,j}| \leq 1/2$ for all $1 \leq j < i \leq d$ and $(\delta - \mu_{i,i-1}^2) \left\|\boldsymbol{b}_{i-1}^*\right\|^2 \leq \|\boldsymbol{b}_i^*\|^2$ for $i \in \{2, \ldots, d\}$.*

In this work we consider the performance of the BKZ algorithm [SE91,SE94], which achieves the following.

Definition 8 (BKZ-β reduced). *A basis $\boldsymbol{B}$ is BKZ-β reduced if it is LLL reduced and for all $i \in [d-1]$, $\|\boldsymbol{b}_i^*\| = \lambda_1\left(\pi_i(\boldsymbol{B}[i\colon \min(i+\beta-1, d)])\right)$.*

In order to do this, an oracle O_{SVP} is used, that, given a lattice, finds its shortest vector. BKZ repeatedly calls O_{SVP} on the projected sublattices, or *blocks*, $\pi_i(\boldsymbol{B}[i\colon \min(i+\beta-1, d)])$. If the output vector $\boldsymbol{v}$ is shorter than the current first vector in the block, it is inserted into the basis at the beginning of the block. LLL is then run on the basis to remove linear dependencies introduced by this insertion. Throughout, we make use of the BKZ implementation in the FPLLL [dt16a] library, which sets $\delta = 0.99$ in its underlying calls to LLL.

In Algorithm 1, we present a description of the BKZ algorithm. In its original description, BKZ terminates after a full tour is executed without inserting. We follow algorithmic improvements and do not necessarily run tours until this point. In particular, the notion of *early abort* (called *auto-abort* in some implementations [dt16a]) was introduced as part of the BKZ 2.0 algorithm [CN11]. The idea is that the majority of improvement occurs in a few early tours, whereas many tours are required before convergence. Following experimental analysis of BKZ [Che13, Figure 4.6] [Alb17, §2.5], Albrecht [Alb17] identifies $\tau = 16$ as the number of tours after which little improvement is made to the basis quality. Furthermore, BKZ 2.0 integrates local block rerandomisation and preprocessing into the originally proposed O_{SVP} oracle, enumeration. We note that while recent advances in lattice sieving mean that enumeration O_{SVP} oracles are no longer the fastest in practice [ADH+19] for large SVP instances, our heuristic analysis

Data: LLL reduced lattice basis $\boldsymbol{B}$
Data: block size β

repeat `/* tour */`
  for $i \leftarrow 1$ **to** d **do**
    $\ell \leftarrow \|\boldsymbol{b}_i^*\|$
    $j \leftarrow \min(i + \beta - 1, d)$
    $\boldsymbol{v} \leftarrow O_{\mathrm{SVP}}(\pi_i(\boldsymbol{B}[i:j]))$
    if $\|\boldsymbol{v}\| \leq \ell$ **then**
      $\boldsymbol{v}' \leftarrow x_i\boldsymbol{b}_i + \cdots + x_j\boldsymbol{b}_j$ where $\boldsymbol{v} = x_i\pi_i(\boldsymbol{b}_i) + \cdots + x_j\pi_i(\boldsymbol{b}_j)$
      extend $\boldsymbol{B}$ by inserting $\boldsymbol{v}'$ into $\boldsymbol{B}$ at index i
      LLL on $\boldsymbol{B}$ to remove linear dependencies
      drop row with all zero entries
    if *if no insertion was made* **then yield** $\top$ **else yield** $\bot$
  if $\top$ *for all* i **then return**

Algorithm 1: Simplified view of the BKZ Algorithm. The instructions inside the `repeat` context are called a BKZ *tour*.

is independent of the underlying O_{SVP} oracle, and for the block sizes we consider the enumeration of FPLLL is slightly faster than the sieves of [ADH+19].

In [AWHT16], Aono *et al.* introduce another variant of BKZ that they name Progressive BKZ. Here, the basis is reduced using increasingly larger block sizes β, running tours of BKZ-β each time. For the purposes of this paper, we define Progressive BKZ as in Algorithm 2, allowing an arbitrary number τ of tours to be run for each block size.

Data: LLL reduced lattice basis $\boldsymbol{B}$ of rank d
Data: $\tau \in \mathbb{Z}^+$

$\beta \leftarrow 3$
while $\beta \leq d$ **do**
  run τ tours of BKZ-β on basis $\boldsymbol{B}$
  $\beta \leftarrow \beta + 1$

Algorithm 2: Progressive BKZ Algorithm, as used in this work.

One consequence of lattice reduction is that it controls how quickly the lengths of the Gram–Schmidt vectors $\boldsymbol{b}_i^*$ (for an output basis $\boldsymbol{B}$) decay. In particular, the larger β is chosen in BKZ, the slower these lengths decay and the closer to orthogonal the basis vectors are. We call the lengths of the Gram–Schmidt vectors, the *basis profile*.

Definition 9 (Basis profile). *Given a basis $\boldsymbol{B}$ of a lattice of rank n, we define the profile of $\boldsymbol{B}$ as the set of squared norms of the orthogonal vectors $\{\|\boldsymbol{b}_i^*\|^2\}_{i=1}^n$.*

Remark 1. In our algorithms, we refer to exact or estimated values $\|\boldsymbol{b}_i^*\|^2$ for a basis as `profile`$[i]$.

Theoretical results exist about the output quality of BKZ-β [HPS11,ALNSD20], as well as heuristic assumptions, which better model average case performance when reducing random q-ary lattices.

Definition 10 (Geometric Series Assumption (GSA) [Sch03]). *Given a basis $\boldsymbol{B}$, the norms of the Gram-Schmidt vectors $\boldsymbol{b}_i^*$ after lattice reduction satisfy*

$$\|\boldsymbol{b}_i^*\| = \alpha^{i-1} \cdot \|\boldsymbol{b}_1\|, \text{ for some } 0 < \alpha < 1.$$

In the case of BKZ-β, α can be derived as a function of β, by combining an estimate for $\|\boldsymbol{b}_1\|$ returned by BKZ [Che13] and the (constancy of the) lattice volume. The GSA can be seen as a global view of a lattice basis, using only the constant volume of the full lattice Λ and an estimate for the length of the first basis vector to calculate α. However, the volume of local *blocks* is not constant as LLL or BKZ is run on a basis. Chen and Nguyen propose a *BKZ simulator* [CN11] that takes this intuition into account to improve on the GSA in the case of BKZ. It takes as *input* a profile $\{\|\boldsymbol{b}_i^*\|^2\}_i$ and simulates a tour of BKZ-β by calculating, block by block, the Gaussian heuristic of the current β dimensional block, "inserting" a vector of that length at the beginning of said block, and redistributing the necessary length to the subsequent Gram–Schmidt vectors to keep $\mathrm{vol}(\Lambda)$ constant. Since projected sublattices of small rank, e.g. $n \leq 45$, do not behave as random,[2] to simulate the profile for the final indices of the basis the BKZ simulator stops using the Gaussian heuristic and instead uses experimental averages over unit volume lattices (scaled appropriately). This design also allows for one to simulate a fixed number of tours, rather than assuming convergence, as in the GSA. The process can be made probabilistic by "inserting" a vector with length drawn from a probability distribution centred on the length suggested by the Gaussian heuristic [BSW18]. The latter approach better captures a phenomenon of lattice reduction called the *head concavity*.

Throughout our work we make use of the Chen–Nguyen simulator as implemented in FPyLLL [dt16b]. In Algorithm 3 we define a BKZSim subroutine that returns a [CN11] simulation for an input basis profile. Here $\mathrm{LWE}_{n,q,\chi,m}$ is a basis produced as in (1) with $c = 1$, assuming normal form so that $\nu = 1$ and $\chi = \chi_s = \chi_e$. To produce the profile of an LLL reduced LWE basis, we considered three options. In the case of the instances used in our experiments, which are described in Section 5, such a profile can be easily obtained by performing LLL on any particular embedding basis. However, this is not the case for cryptographically sized embeddings, where FPLLL's implementation of LLL can only run with high enough floating point precision by using MPFR [FHL+07], which becomes impractically slow. An alternative is to use a GSA slope corresponding to LLL reduction. This correctly predicts the slope of the main section of the profile, but does not account for the role played by the q-vectors in the embedding basis, which are short enough to not be affected by LLL [How07]. The third option is to use a specific basis profile simulator for LLL that captures the effect of the q-vectors. We opt for the third option; a description of the Z-shape phenomenon and its simulation can be found in Appendix C.

[2] See e.g. [Che13, §2.3.2] for a formal introduction.

Input: (n, q, χ, m) or profile $\{\|\boldsymbol{b}_i^*\|^2\}_i$
Input: β, τ
if $\{\|\boldsymbol{b}_i^*\|^2\}_i$ *not provided as input* **then**
  $\{\|\boldsymbol{b}_i^*\|^2\}_i \leftarrow$ simulated profile of LLL reduced $\mathrm{LWE}_{n,q,\chi,m}$ instance
$\{\|\boldsymbol{b}_i^*\|^2\}_i \leftarrow$ [CN11] simulation of τ tours of BKZ-β on $\{\|\boldsymbol{b}_i^*\|^2\}_i$
return $\{\|\boldsymbol{b}_i^*\|^2\}_i$

Algorithm 3: BKZSim subroutine.

3 Choosing BKZ block sizes and the "2016 estimate"

In this section we motivate and explain the approach introduced in [ADPS16] to predict the block size required to solve uSVP using lattice reduction.

The runtime of BKZ-β is dominated by that of the O_{SVP} subroutine. The latter is often implemented using lattice point enumeration with preprocessing, which has time complexity $\beta^{\Theta(\beta)}$ or lattice sieving, which has time and memory complexity $2^{\Theta(\beta)}$. Therefore, to estimate the complexity of solving uSVP using lattice reduction, it is crucial to estimate the smallest block size sufficient to recover the unique shortest vector $\boldsymbol{t} \in \Lambda$.

The most successful approach for making such estimates was introduced in [ADPS16, §6.3] and is sometimes referred to in the literature as the "2016 estimate". The idea is to estimate a block size β such that at some point during lattice reduction, O_{SVP} will return a projection of the uSVP solution as the shortest vector in a local projected sublattice. If the rank of this projected sublattice is large enough, subsequent cheap lattice reduction operations (usually, a single call to LLL [AGVW17]) will recover the full uSVP solution. Concretely, this approach consists of finding the smallest β such that in the final full sized block starting at index $d - \beta + 1$,

$$\|\pi_{d-\beta+1}(\boldsymbol{t})\| \leq \left\|\boldsymbol{b}_{d-\beta+1}^*\right\|, \tag{2}$$

resulting in O_{SVP} recovering the projection of $\boldsymbol{t}$ at index $d - \beta + 1$.

In [ADPS16], the authors consider normal form LWE, and assume the secret distribution χ to be centred around 0. The uSVP solution will be an embedded vector for which each entry is drawn i.i.d. from a distribution of standard deviation σ and mean $\mu = 0$, with the addition of one final, constant, entry c.[3] Using the Bai–Galbraith embedding, our target vector is $\boldsymbol{t} = (\boldsymbol{s} \mid \boldsymbol{e} \mid c)$, of dimension $d = n + m + 1$. The squared norm $\|\boldsymbol{t}\|^2$ may be modelled as a random variable following a scaled chi-squared distribution $\sigma^2 \cdot \chi^2_{d-1}$ with $d-1$ degrees of freedom, plus a fixed contribution from c, resulting in $\mathbb{E}(\|\boldsymbol{t}\|^2) = (d-1)\sigma^2 + c^2$.

In [ADPS16], the authors approximate the left hand side of (2) as $\|\pi_{d-\beta+1}(\boldsymbol{t})\| \approx \mathbb{E}(\|\boldsymbol{t}\|)\sqrt{\beta/d} \approx \sigma\sqrt{\beta}$, where they approximate $\mathbb{E}(\|\boldsymbol{t}\|) \approx \sigma\sqrt{d}$. The approximation $\mathbb{E}(\|\boldsymbol{t}\|) \approx \sigma\sqrt{d}$ replaces $(d-1)\sigma^2 + c^2$ with $d\sigma^2$, which for large d or for

[3] This constant c is often chosen as 1, which gives better attacks in practice [AFG14,BG14,AGVW17], though formally it should be chosen as σ [LM09].

$c \approx \sigma$ introduces little error, and assumes that $\mathbb{E}(\|\boldsymbol{t}\|) = \mathbb{E}(\|\boldsymbol{t}\|^2)^{1/2}$. The error in this assumption tends to 0 as $d \to \infty$, so we ignore it. An exact derivation can be found in Appendix B. This assumption can also be avoided altogether by working with squared lengths, as we do in our analysis.

To approximate the right hand side of (2), in [ADPS16, §6.3] the authors make use of the GSA. Assuming that BKZ-β returns a first basis vector of length $\ell_1(\beta)$ when called with the basis of a random q-ary lattice as input, this results in the following win condition that β must satisfy for solving uSVP using BKZ-β,

$$\sqrt{\beta}\sigma \approx \|\pi_{d-\beta+1}(\boldsymbol{t})\| \leq \left\|\boldsymbol{b}^*_{d-\beta+1}\right\| \approx \alpha(\beta)^{d-\beta} \cdot \ell_1(\beta). \tag{3}$$

At first glance the careful reader may notice an apparent contradiction in the methodology. Indeed, the GSA describes the basis profile produced by BKZ for a random lattice, and in [ADPS16] ℓ_1 is determined assuming this is the case. However, we are reducing a uSVP embedding lattice. While the embedding basis looks like that of a random q-ary lattice, the shortest vector will be shorter than $\ell_1(\beta)$. Yet, this shortest vector is hard to find. What (3) aims to capture is exactly the moment where BKZ is able to find this shortest vector, and hence distinguish our uSVP embedding lattice from a random q-ary lattice. The GSA and ℓ_1 are used to describe the status of the basis up until this moment, while it still looks like the basis of a random q-ary lattice.

In this model, (3) provides a clear cut answer to what is the smallest viable block size to solve uSVP. In practice, BKZ 2.0 is a randomised algorithm, working on a random uSVP instance. In [AGVW17], the authors verify the validity of this win condition, resulting in a success probability of approximately 90% when using β chosen by following (3). However, they also measure that somewhat smaller block sizes also present some relatively high success probabilities of solving uSVP.

4 Simulating solving uSVP

In this section, we review and extend recent work on capturing the probabilistic nature of a uSVP win condition.

In [DSDGR20], Dachman-Soled *et al.* revisit the [ADPS16] heuristic methodology described in Section 3. The authors are concerned with accurately predicting the effects that introducing side channel information to their lattice embedding has on the success probability of solving uSVP using Progressive BKZ, while also maintaining accuracy in the small block size regime, $\beta \leq 45$. The authors describe a *uSVP simulator* (not to be confused with the BKZ simulator of [CN11]), designed to predict the success probability of Progressive BKZ solving an *isotropic* uSVP instance by a specific block size.[4] Using their uSVP simulator, they predict the expected successful block size for a series of experiments they run, and verify the accuracy of their predictions.

[4] Any uSVP instance used in the primal attack can be made isotropic, where $\sigma = 1$.

In this section, we start by simplifying the [DSDGR20] uSVP simulator for Progressive BKZ, and develop a similar uSVP simulator for BKZ 2.0. We describe the rationale of these new uSVP simulators, and in Section 5 we verify their accuracy with various sets of experiments.

4.1 Progressive BKZ

The approach proposed in [DSDGR20] to estimate the required block size to solve a uSVP instance is to simulate the status of a lattice basis as it is being reduced, and with it the probability for each step of the lattice reduction algorithm that the target vector is recovered.

Input: d
1 $p_{\mathrm{tot}} \leftarrow 0$, $\bar{\beta} \leftarrow 0$
2 `profile` $\leftarrow$ GSA profile of an LLL reduced, rank d, isotropic uSVP instance basis
3 **for** $\beta \leftarrow 3$ **to** d **do** `/* round */`
4 `profile` $\leftarrow \mathrm{BKZSim}(\texttt{profile}, \beta, 1)$
5 $p_{\mathrm{lift}} \leftarrow P[\boldsymbol{t}$ recovered in $\lfloor d/\beta \rfloor$ rounds $\mid \pi_{d-\beta+1}(\boldsymbol{t})$ recovered this round$]$
6 $p_{\mathrm{rec}} \leftarrow P[x \leftarrow \chi^2_\beta : x \leq \texttt{profile}[d-\beta+1]]$
7 $p_{\mathrm{new}} \leftarrow (1-p_{\mathrm{tot}}) \cdot p_{\mathrm{rec}} \cdot p_{\mathrm{lift}}$
8 $\bar{\beta} \leftarrow \bar{\beta} + \beta \cdot p_{\mathrm{new}}$
9 $p_{\mathrm{tot}} \leftarrow p_{\mathrm{tot}} + p_{\mathrm{new}}$
10 **if** $p_{tot} \geq 0.999$ **then break**
11 **return** $\bar{\beta}$

Algorithm 4: Isotropic uSVP simulator for Progressive BKZ with $\tau = 1$, as proposed in [DSDGR20]. We omit the details of computing p_{lift} for simplicity and note that p_{rec} represents $P[\pi_{d-\beta+1}(\boldsymbol{t})$ recovered this round$]$. Returns the expected block size $\bar{\beta}$ required to solve uSVP.

Let W be the event of solving uSVP during the run of Progressive SVP, W_β the probability of being able to solve uSVP during the round with block size β, and $F_\beta = \neg W_\beta$. Following the notation in Algorithm 2, we assume $\tau = 1$, meaning that for each block size β exactly one tour of BKZ-β is run. They implicitly partition W as follows

$$P[W] = P[W_3] + P[W_4 \wedge F_3] + P[W_5 \wedge F_4 \wedge F_3] + \cdots = \sum_{\beta=3}^{d} P\left[W_\beta \wedge \bigwedge_{j=3}^{\beta-1} F_j\right].$$

Their computation of the expected winning block size $\bar{\beta}$ amounts to implicitly defining a probability mass function for the random variable B representing the *first viable* block size to solve the uSVP instance, and computing its expected value. In the case of Progressive BKZ, a block size β being the first viable means that it is the round of BKZ run with block size β (i.e. the tour of Line 3 of

Algorithm 2 with block size β) and not any earlier round using a smaller block size, that will solve the uSVP instance. The resulting probability mass function for the distribution of B can be modelled as

$$P[B = \beta] = P\left[W_\beta \wedge \bigwedge_{j=3}^{\beta-1} F_j\right].$$

The probability $P[W_\beta]$ is itself modelled as the product of the probability of successfully recovering $\pi_{d-\beta+1}(\boldsymbol{t})$ by calling O_{SVP} on the last full size block of the basis,

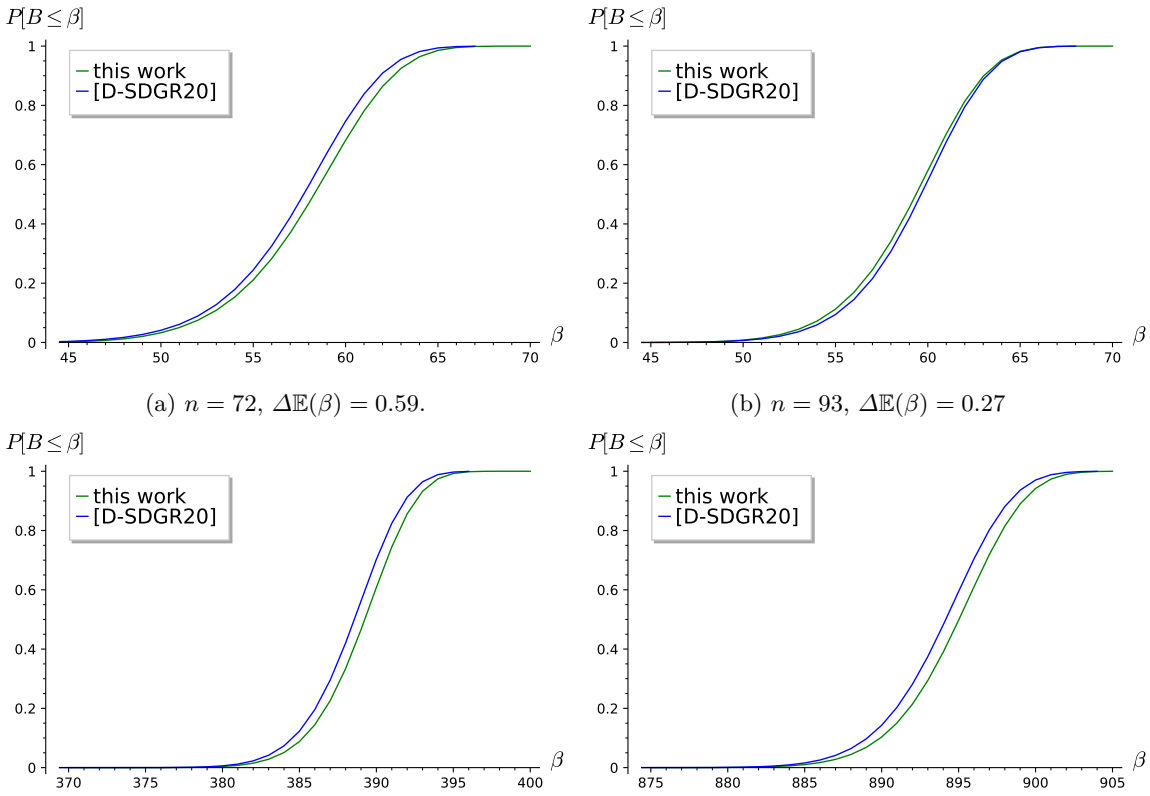
$$P[\pi_{d-\beta+1}(\boldsymbol{t}) \text{ recovered using block size } \beta] \approx P[x \leftarrow \chi_\beta^2 \colon x \leq \texttt{profile}[d-\beta+1]],$$

and the probability of successfully lifting the projection over subsequent rounds, p_{lift}. In their implementation of Algorithm 4, Dachman-Soled *et al.* use a chain of conditional probabilities to compute p_{lift}. This and all other events, such as that of recovering $\pi_{d-\beta+1}(\boldsymbol{t})$, as well as W_i and F_j for $i \neq j$, are considered to be independent.

We introduce two simplifications to the above uSVP simulator. Firstly, we noticed experimentally that running BKZ with block sizes smaller than 40 will not solve instances for which the [ADPS16] approach predicts a winning block size of $\beta \gtrsim 60$. The expected winning block size for our chosen experimental instances is approximately 60. Furthermore, values of p_{lift} approach 1 quickly as β increases, such that one can simply assign $p_{\mathrm{lift}} = 1$ for $\beta \geq 40$; a similar phenomenon is noted in [AGVW17]. Therefore, we skip probability computations for any block sizes smaller than 40. Finally, by allowing multiple tours per block size, we define a uSVP simulator, Algorithm 5, for Progressive BKZ as described in Algorithm 2 where τ may be greater than 1. A comparison between the output of Algorithms 4 and 5 can be found in Figure 1 for four isotropic LWE instances, where $\tau = 1$. To produce the plots, we tweaked the original [DSDGR20] code in order to extract the implicit probability mass function $P[B = \beta]$. We can see that the output probabilities $P[B \leq \beta]$ and the expected successful block sizes differ only slightly, while our tweaks significantly speed up the simulation by avoiding the expensive computation of p_{lift}. It should be noted that the slight difference in output for the cryptographically sized instances is due to the [DSDGR20] uSVP simulator using the GSA to simulate their input LLL bases, instead of an LLL specific simulator, such as the one we use. When tweaking our code to use the GSA, the outputs coincide for the two Kyber instances in Figure 1.

4.2 BKZ

Using the same approach as for Algorithm 4 and Algorithm 5, we implemented a uSVP simulator for BKZ, described in Algorithm 6. In this case, the basis profile after a number of tours of BKZ-β is simulated in one shot using the [CN11] simulator. Given that the block size is fixed, the probabilities are only accumulated over tours. It should be noted that the event of β being the first viable block size



(a) $n = 72$, $\Delta\mathbb{E}(\beta) = 0.59$.

(b) $n = 93$, $\Delta\mathbb{E}(\beta) = 0.27$

(c) Kyber 512, $\Delta\mathbb{E}(\beta) = 0.68$

(d) Kyber 1024, $\Delta\mathbb{E}(\beta) = 0.86$

Fig. 1: Comparison between the output of Algorithm 4 [DSDGR20] and Algorithm 5 (this work) for isotropic parameters ($\sigma = 1$) from Table 1, and on Kyber 512 and 1024 [SAB+19]. The difference in predicted mean first viable block size between the two simulators is reported as $\Delta\mathbb{E}(\beta)$.

changes in the case of BKZ. In this case, no unsuccessful tours with a smaller block size are run by the algorithm. Instead, we consider β being first viable if running BKZ-$(\beta - 1)$ would not result in a solution to the uSVP instance but running BKZ-β would.

Algorithm 6 returns the probability that τ tours of BKZ-β will solve uSVP, but does not exclude the possibility of winning with a smaller block size. We assume in our model that if τ tours of BKZ-β solve a given uSVP instance, then τ tours of BKZ-β', for $\beta' > \beta$, also will. The values output by Algorithm 6 for a given instance can therefore be interpreted as a cumulative mass function for the first viable block size, i.e. $P[B \leq \beta]$. By running the simulator for increasing block sizes until it outputs probability 1, one may recover the probability mass

Input: (n, q, χ, m), τ

$p_{\text{tot}} \leftarrow 0$, $P \leftarrow \{\}$, $\beta \leftarrow 3$
$d \leftarrow n + m + 1, \sigma^2 \leftarrow \mathbb{V}(\chi)$
`profile` $\leftarrow$ simulated profile of LLL reduced $\text{LWE}_{n,q,\chi,m}$ instance
while $\beta < 40$ **do**
    `profile` $\leftarrow \text{BKZSim}(\texttt{profile}, \beta, \tau)$
    $\beta \leftarrow \beta + 1$
while $\beta \leq d$ **do** `/* rounds */`
    for *tour* $\leftarrow 1$ **to** τ **do** `/* tours */`
        `profile` $\leftarrow \text{BKZSim}(\texttt{profile}, \beta, 1)$
        $p_{\text{new}} \leftarrow P[x \leftarrow \sigma^2 \chi_\beta^2 \colon x \leq \texttt{profile}[d - \beta + 1]]$
        $P[\beta] \leftarrow (1 - p_{\text{tot}}) \cdot p_{\text{new}}$
        $p_{\text{tot}} \leftarrow p_{\text{tot}} + P[\beta]$
        if $p_{tot} \geq 0.999$ **then break**
    $\beta \leftarrow \beta + 1$
return P

Algorithm 5: Unique-SVP success probability simulator running Progressive BKZ, running τ tours for each block size, then increasing the block size by 1. Returns the probability mass function $P[B = \beta]$ of solving uSVP in the round using block size β.

function $P[B = \beta]$ as

$$P[B = \beta] = P[B \leq \beta] - P[B \leq \beta - 1].$$

Input: (n, q, χ, m), β, τ

$p_{\text{tot}} \leftarrow 0, \sigma^2 \leftarrow \mathbb{V}(\chi)$
$d \leftarrow n + m + 1$
for *tour* $\leftarrow 1$ **to** τ **do**
    `profile` $\leftarrow \text{BKZSim}((n, q, \chi, m), \beta, \text{tour})$
    $p_{\text{new}} \leftarrow P[x \leftarrow \sigma^2 \chi_\beta^2 \colon x \leq \texttt{profile}[d - \beta + 1]]$
    $p_{\text{tot}} \leftarrow p_{\text{tot}} + (1 - p_{\text{tot}}) \cdot p_{\text{new}}$
return p_{tot}

Algorithm 6: Unique-SVP success probability estimator when running τ tours of BKZ-β. Returns the probability of solving the uSVP instance.

5 Experiments

In this section, we describe the experiments we run to check the accuracy of Algorithms 5 and 6, and discuss the results. We start by describing our original batch of experiments in Section 5.1. In Section 5.2 we make some observations

about our experimental results, and describe further tweaked experiments that we run to verify our understanding of the results.

5.1 Initial experiments

Our aim in this section is threefold: first, we want to provide experimental evidence for the accuracy of our BKZ and Progressive BKZ uSVP simulators when predicting the success probability of the primal attack against LWE with discrete Gaussian secret and error for different block sizes; second, we want to compare previous experiments [AGVW17] to our uSVP simulations; and finally, we want to explore the effect that binary or ternary distributions have on the primal attack. Throughout our experiments, we use BKZ 2.0 as implemented in FPyLLL [dt16b] version 0.5.1dev, writing our own Progressive BKZ script by using FPyLLL's BKZ 2.0 as a subroutine.

For our first goal, we choose three different parameterisations of the LWE problem, for which the [ADPS16] approach predicts an expected successful block size of either 60 or 61. We give the parameters in Table 1. All parameter sets in these batches use discrete Gaussian secret and error with $\mathbb{V}(\chi_s) = \mathbb{V}(\chi_e) = \sigma^2$. The number of LWE samples used, m, is determined by what the LWE estimator [APS15] predicts to be optimal, using (3). For each parameter set we generate 100 instances, and reduce them using either BKZ or Progressive BKZ. We then check whether lattice reduction positioned the embedded shortest target vector in the first index of the reduced basis.

In the case of BKZ, for each basis we run a number of tours of BKZ with block size $\beta = 45, \ldots, 65$. The number of tours, τ, takes the values $5, 10, 15, 20, 30$. This results in a total of 100 bases, reduced independently 21×5 times each, once for every combination of β and τ. For every set of 100 reductions, we record the success rate by counting the number of solved instances. We run a similar set of experiments using Progressive BKZ, allowing $\tau \geq 1$ tours per block size, in order to see at what point running extra tours per block size becomes redundant. For this reason, we reduce each basis 5 times, once per value of τ in $1, 5, 10, 15, 20$. After every call to the BKZ subroutine, we check whether the instance is solved. If not, we increase the block size by 1 and run a further tour of BKZ.

The resulting success rates for BKZ and Progressive BKZ (with $\tau = 1$) are plotted in Figure 2, together with the output of our uSVP simulators, interpolated as a curve. Figure 3 contains similar plots for Progressive BKZ with $\tau \geq 1$. In Figure 5 we plot the measured difference between the average mean and standard deviation for the simulated and experimental probability distributions, for both Progressive BKZ and BKZ.

For our second goal, we take the success probabilities reported in [AGVW17] for their experiments. In Figure 4 we report their measured success rates at optimal and smaller than optimal block sizes, and we superimpose our BKZ success probability simulations.

Finally, for our third goal, we run Progressive BKZ experiments for τ in $\{1, 5, 10, 15, 20\}$ on a set of three parameter sets with bounded uniform secrets. In particular, we pick the $n = 72$ and $n = 93$ parameters from Table 1 but sample

n	q	σ	m_{2016}	β_{2016}
72	97	1	87	61
93	257	1	105	61
100	257	$\sqrt{2/3}$	104	60

Table 1: List of LWE parameters used for testing our uSVP simulators. The instances are in normal form. We use the Bai–Galbraith embedding and the number of samples used, m_{2016}, is given by the LWE estimator (commit `428d6ea`).

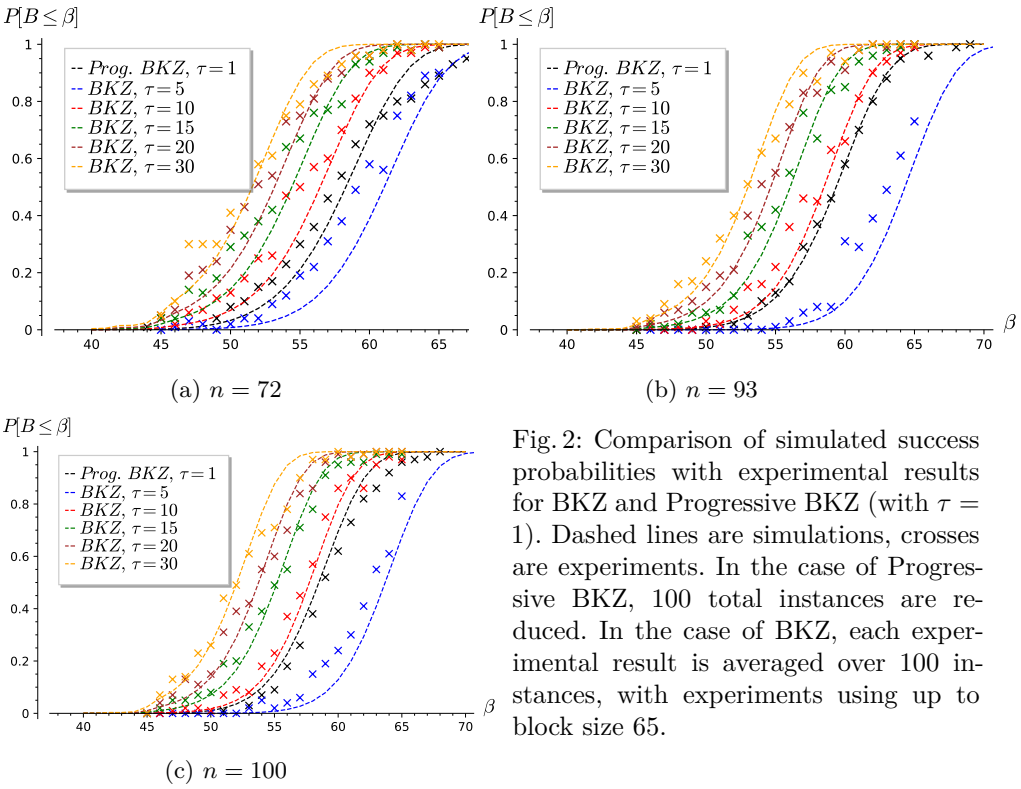
secret $\boldsymbol{s}$ and error $\boldsymbol{e}$ coefficients uniformly from the set $\{-1, 1\}$, and the $n = 100$ parameters with secret and error coefficients sampled uniformly from $\{-1, 0, 1\}$. This preserves the same standard deviations as in Table 1, while adding more structure to the target vector. In the first case, the $\boldsymbol{s}$ and $\boldsymbol{e}$ are equivalent to those of a scaled and centred LWE instance with binary secret and error (see Appendix A), while in the second case, the problem is LWE with ternary $\boldsymbol{s}$ and $\boldsymbol{e}$. The resulting success probability plots can be found in Figure 6.

5.2 Observations

Experimental success rates for both BKZ and Progressive BKZ are in line with the output of the simulators described in Section 4. Below, we look in detail at the results.

Progressive BKZ. In the case of Progressive BKZ, simulations seem to predict accurately the success probabilities up to $\tau \leq 10$ for all secret and error distributions used. Throughout our experiments reported in Figure 3, we observe two ways in which experiments slightly deviate from predictions.

Firstly, the success probability appears to stop significantly increasing as $\tau > 10$, even when the simulation does predict some improvement. We believe this to be a consequence of the large amount of lattice reduction being performed. Indeed, whenever the BKZ-β subroutine is called, the basis has already been reduced with τ tours of BKZ-$(\beta - j)$ for $j = 1, \ldots, \beta - 3$. This suggests that only little progress on the basis profile can be made with each new tour of BKZ-β. In our experiments, we use FPyLLL's BKZ 2.0 implementation with auto-abort. If it is the case that little progress can be made, then auto-abort will trigger and fewer than τ tours will be run. To verify this, we rerun experiments while measuring the number of tours run by the BKZ subroutine. The data for the $n = 100$ experiments can be found in Figure 7, and seems to confirm that auto-abort for $\beta > 20$ is much more frequently triggered for $\tau > 10$. This problem does not affect Progressive BKZ with $\tau = 1$ since even with auto-abort, one tour is always run, and only slightly affects $\tau = 5$ and $\tau = 10$. Predictions match experiments well in those cases. We note that, even if we were to force all τ tours to be performed, such little improvement or alteration would likely be made to



(a) $n = 72$

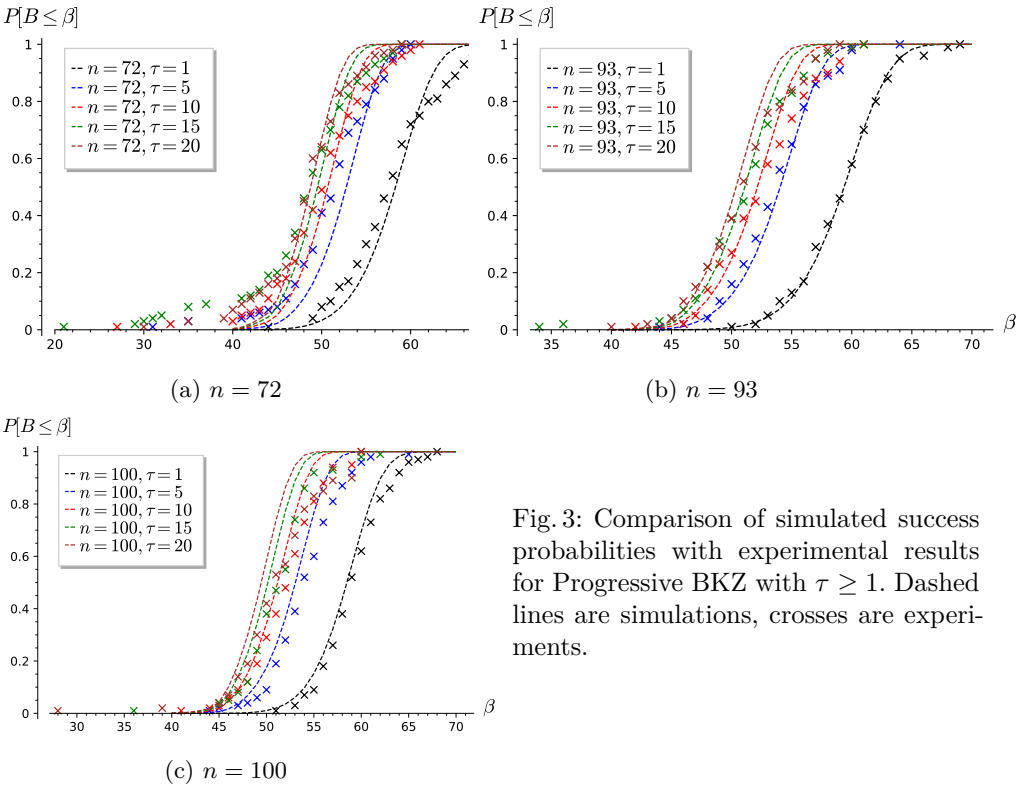
(b) $n = 93$

(c) $n = 100$

Fig. 2: Comparison of simulated success probabilities with experimental results for BKZ and Progressive BKZ (with $\tau = 1$). Dashed lines are simulations, crosses are experiments. In the case of Progressive BKZ, 100 total instances are reduced. In the case of BKZ, each experimental result is averaged over 100 instances, with experiments using up to block size 65.

the basis that the forced extra tours would not rerandomise it enough for the events of winning after two consecutive tours to be independent, as our model assumes.

The other phenomenon is the presence of a slight plateau in the probability plots as $P[B \leq \beta] \geq 0.8$. In the case of $n = 72$ we also see that smaller than predicted block sizes accumulate a significant success probability. Interestingly, this effect does not appear to be present in the case of binary secret and error LWE, see Figures 6a and 6b. We believe this phenomenon to be caused by the slight variation in *sample variance* throughout our experiments. Indeed, if we think of our target vector $\boldsymbol{t} = (t_1, \ldots, t_d)$ as sampled coefficientwise from some distribution χ with variance σ^2, in practice the resulting sample variance for each particular LWE instance $s^2 := \frac{1}{d}\sum_{i=1}^{d}(t_i - \bar{t})^2$, with $\bar{t} := \frac{1}{d}\sum t_i$ the *sample mean*, will likely slightly deviate from σ^2. This will cause the $\|\pi_i(\boldsymbol{t})\|$ norms to differ slightly from their expected value. However, in the case of $\chi = \mathcal{U}(\{-1, 1\})$, i.e. the distribution resulting from scaled and centred binary LWE embeddings,



(a) $n = 72$

(b) $n = 93$

(c) $n = 100$

Fig. 3: Comparison of simulated success probabilities with experimental results for Progressive BKZ with $\tau \geq 1$. Dashed lines are simulations, crosses are experiments.

this distribution has a very small expected variance of s^2 (i.e. $\mathbb{E}(\mathbb{V}(s^2))$),[5] meaning that most sampled target vectors will have sample variance almost exactly $\mathbb{V}(\chi) = 1$. To verify this hypothesis, we run a set of $n = 72$ and $n = 100$ discrete Gaussian experiments from Table 1, where we resample each LWE instance until the target vector's sample variance is within a 2% error of σ^2, and then run Progressive BKZ with τ in $1, 5, 10$. The resulting experimental probability distributions, shown in Figure 8, do not present plateaus (and in the case of $n = 72$, they also do not present the high success probability for small block sizes), supporting our hypothesis. In practice, this effect should not significantly affect cryptographic parameters, as $\mathbb{E}(\mathbb{V}(s^2)) \in O(\frac{1}{d})$, keeping the effect of fluctuations in $\|\pi_{d-\beta+1}(\boldsymbol{t})\|^2$ small as the embedding dimension d increases.

The fact that simulations seem similarly consistent in the case of scaled and centred binary, and ternary secret and errors, as seen in Figure 6, seems to be in line with the folklore notion that the hardness of solving uSVP via lattice reduction depends on the standard deviation of the target vector's coefficients

[5] Following [KK51,SR02], we compute $\mathbb{E}(\mathbb{V}(s^2))$ as approximately 0.00995, 0.00112, and 0.00005 for a discrete Gaussian with $\sigma^2 = 1$, $\mathcal{U}(\{-1, 0, 1\})$ and $\mathcal{U}(\{-1, 1\})$ respectively, for sets of 200 samples.

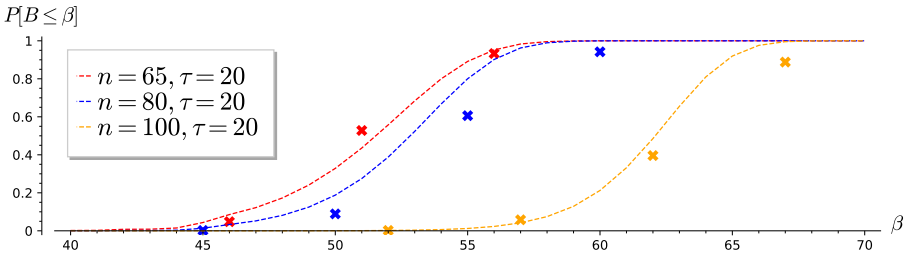
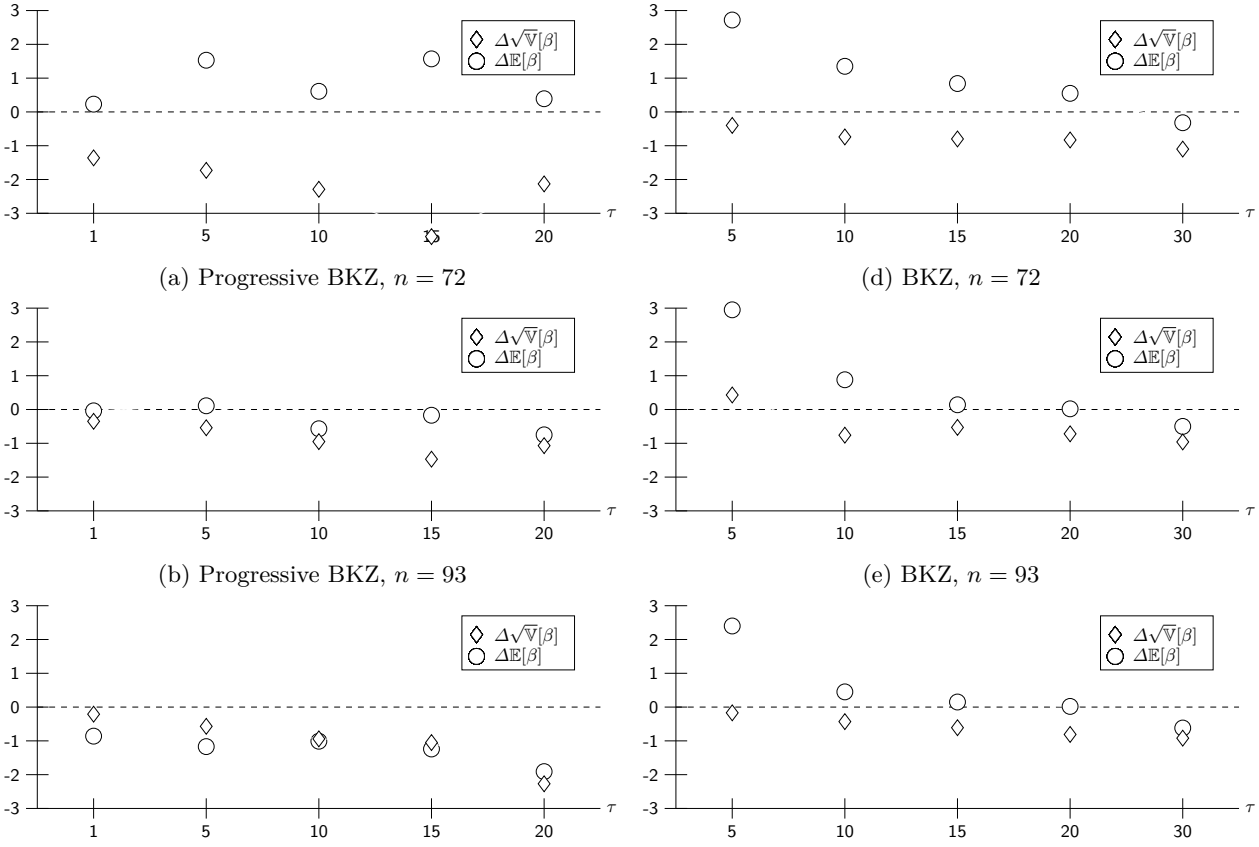


Fig. 4: Comparison of simulated BKZ success probabilities with experimental results reported in Table 1 of [AGVW17].



(a) Progressive BKZ, $n = 72$

(d) BKZ, $n = 72$

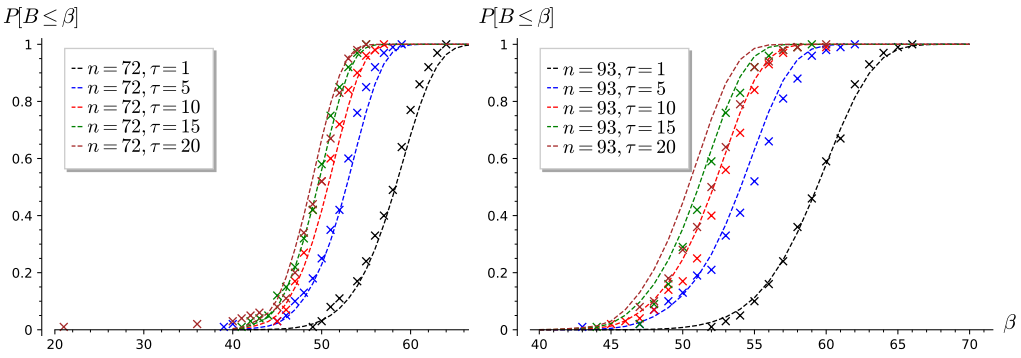
(b) Progressive BKZ, $n = 93$

(e) BKZ, $n = 93$

(c) Progressive BKZ, $n = 100$

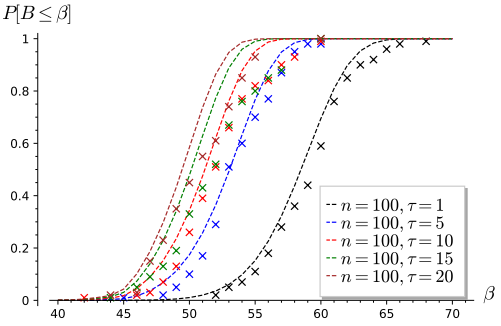
(f) BKZ, $n = 100$

Fig. 5: The measured difference $\Delta\mathbb{E}[\beta]$ (resp. $\Delta\sqrt{\mathbb{V}}[\beta]$) between the simulated and experimental successful block size mean (resp. standard deviation), as τ grows.



(a) $n = 72$

(b) $n = 93$



(c) $n = 100$

Fig. 6: Comparison of simulated success probabilities with experimental results for Progressive BKZ on LWE instances with scaled and centred binary secret and error (Figures 6a and 6b), and ternary secret and error (Figure 6c). Dashed lines are simulations, crosses are experiments. Each experimental result is averaged over 100 instances. No changes were made to the uSVP simulators.

rather than their exact distribution. We do not observe the higher than expected success probability reported in [CCLS20]. Possible reasons for this could be their experiments targeting the small block size regime, where some of the heuristics behind BKZ simulation are known not to hold [GN08b,CN11], and their analysis potentially not accounting for the contribution of the secret distribution width to the norms of $\pi_i(\boldsymbol{t})$.

BKZ. In the case of BKZ, simulations seem to stay similarly accurate across all secret dimensions n, as reported in Figure 2. It should be noted that, even though a larger gap than for Progressive BKZ can be seen between predictions and experiments in the case of $\tau = 5$, this predictive gap in expected block size of less than 3 corresponds to about 1 bit in a core-sieve cost model [ADPS16]. Furthermore, this gap narrows as τ increases. We also see that the phenomenon of success probabilities not increasing when $\tau \geq 10$, as in the Progressive BKZ case, does not occur here. This is compatible with our understanding of this phenomenon in the case of Progressive BKZ. Indeed, BKZ-β will not auto-abort as often due to the input basis not having already been reduced, for example, with BKZ-$(\beta-1)$. Following experimental results from [Che13, Figure 4.6] and [Alb17], designers often [ACD+18] consider it sufficient to reduce a basis using $\tau = 16$

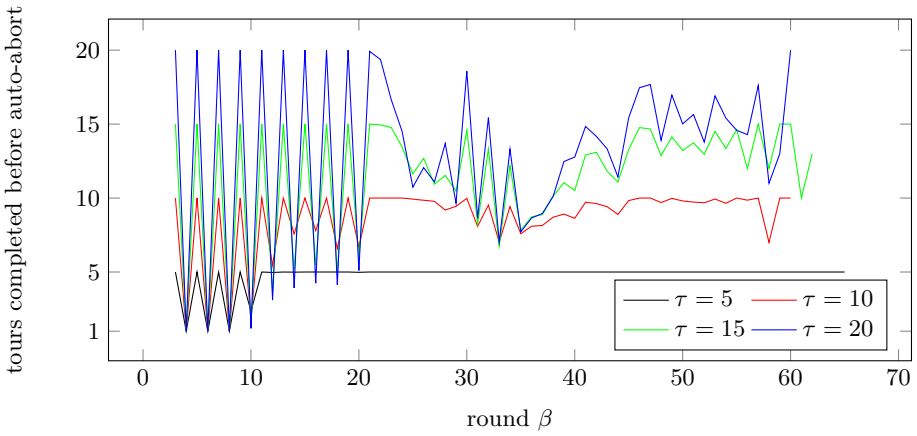


Fig. 7: Measured number of tours run by the BKZ 2.0 subroutine of Progressive BKZ with $\tau \geq 5$ for each round of reduction with block size β. Numbers are from experiments using the $n = 100$ parameters from Table 1, with discrete Gaussian secret and error. Values are averaged over 100 instances.

tours of BKZ when specifiying BKZ cost models, due to the basis quality not improving significantly after 16 tours. Our simulators seem accurate for values of τ in such a regime.

However, a different interesting phenomenon can be observed. Sometimes, as the block size is increased, the experimental success probability of BKZ lowers. Originally we believed this to be caused by the preprocessing strategies used in FPyLLL. Indeed, at the time of writing, preprocessing strategies for block size β (resp. $\beta+1$), could include running BKZ-β' (resp. BKZ-β''), with $\beta' > \beta''$, resulting in inferior quality preprocessing for BKZ-($\beta+1$) than for BKZ-β. We replaced the default preprocessing strategies with a custom one such that preprocessing block sizes are non decreasing as a function of β, however this did not remove the effect. A possible cause for this phenomenon could be that basis profiles output by the [CN11] simulator do not capture the possibility that Gram–Schmidt vector norms can be non decreasing as a function of their index. This means that one could have a BKZ-β reduced basis such that $\left\|\boldsymbol{b}^*_{d-\beta}\right\| < \left\|\boldsymbol{b}^*_{d-\beta+1}\right\|$. This event happening across instances or block sizes could be a potential cause for the phenomenon. The probabilistic BKZ simulator developed in [BSW18] seems to better capture this phenomenon, when run with a fixed PRNG seed. An example of the output of our uSVP simulator for BKZ, when replacing the [CN11] simulator with the [BSW18] simulator, can be found in Figure 9. However, our experimental measurements are averaged over 100 runs. Running our uSVP simulator with the [BSW18] simulator, and averaging its output, results in a simulation with strictly increasing probabilities, unlike our measurements. In any case, the overall success probability predictions stay reasonably accurate.

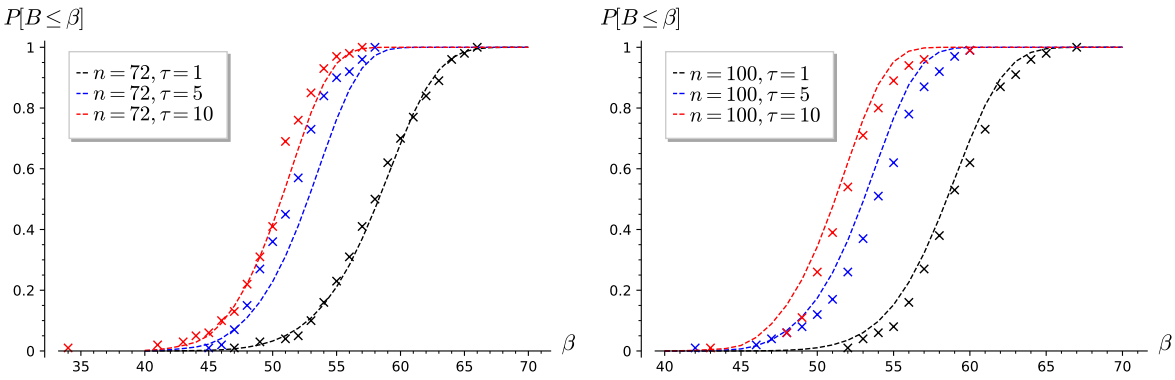


Fig. 8: Progressive BKZ success probability against LWE instances with discrete Gaussian secret and error and $(n, \sigma^2) \in \{(72, 1), (100, 2/3)\}$, such that their sample variance is within 2% of σ^2.

Finally, looking at Figure 4, it seems that our simulations are consistent with the measurements originally reported in [AGVW17, Table 1]. The simulators therefore seem to explain the reported success probabilities of lower than expected block sizes in that paper. It should be noted that while most other experiments described in this section could be similarly predicted by passing as input to our simulators the LLL basis profile predicted by the GSA rather than the LLL simulator (cf. Appendix C), in this particular case they cannot. Inputting a GSA basis profile results in a noticeable error. This may be an indirect consequence of [AGVW17] using a larger than optimal number of LWE samples. Using the LLL basis profile simulator solves this problem.

6 Simulations of cryptographically sized LWE instances

In previous sections we developed simulators for the sucess probability of solving uSVP instances, and tested them against uSVP embedding lattices generated from small LWE instances that could be solved in practice. An immediate application would be to use such simulators to estimate the behaviour of lattice reduction when used against cryptographically sized instances.

Here we use the simulator to compute the expected first viable block sizes required to solve LWE and NTRU instances proposed for the NIST PQC standardisation process. In particular we look at the second round versions of the three lattice finalists; Kyber [SAB+19], NTRU [ZCH+19], and Saber [DKRV19]. An interesting option would be to use the simulators to predict what block size is required to solve an instance with a target low success probability. However, as we have dicussed in Section 5.2, the simulations are not necessarily fully accurate for smaller or larger block sizes, due to the fluctuations in sample variance that an instance can have. While the effect should be minor for cryptographically

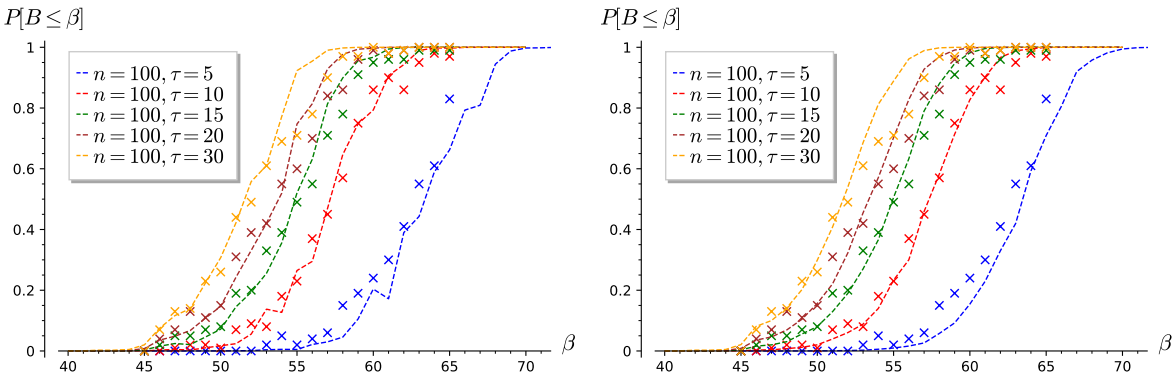


Fig. 9: Both figures show BKZ experiments and uSVP simulations for $n = 100$ instances with Gaussian secret and error case, where the calls to the [CN11] simulator made in Algorithm 6 are replaced. The left plot shows simulations where the [BSW18] simulator is used, with a fixed PRNG seed. The right plot shows the same experimental data with with simulations obtained by averaging the output of the [BSW18] simulator over 10 different seeds.

sized instances, low probability attacks may also include other combinatorial techniques not captured by our simulators. Therefore, extracting block sizes for low probability attacks from the simulated probabilities may not capture all the necessary subtleties. Furthermore, we will see that the window of block sizes predicted to be first viable is relatively narrow, so that lower success probability attacks without combinatorial tricks should not be significantly cheaper than higher success probability attacks.

In Table 2, we look at parameter sets from the three lattice-based finalists in the third round of the NIST PQC standardisation process [NIS16], as specified during the second round. We provide expected first viable block sizes $\mathbb{E}(\text{succ. } \beta)$ (and their standard deviation $\sqrt{\mathbb{V}}(\text{succ. } \beta)$) when using 15 tours of BKZ, Progressive BKZ, and Progressive BKZ using 5 tours of BKZ as subroutine (see Algorithm 2). We choose $\tau = 15$ for BKZ due to our experiments confirming the accuracy of our estimator for this value and its closeness to 16, which is commonly found in BKZ cost models. We choose $\tau = 1$ and $\tau = 5$ in the case of Progressive BKZ since our experiments suggest both cases are accurately predicted by the uSVP simulator; this allows us to see if running more tours in the BKZ subroutine has any effect on the complexity of cryptographically sized parameters.

Two clear disclamers should be made. First, in Table 2 we list the expected block size required to solve uSVP instances for the primal attack. While in an aggressive cost model for these algorithms, such as core-SVP [ADPS16], one could be tempted to make direct cost comparisons between algorithms based only on β, in the case of BKZ we assume that τ tours of BKZ-β are run, while in

the case of Progressive BKZ about $\tau\beta$ tours of varying block size are run. Second, for both algorithms we fixed the same number of samples m, chosen with the aid of the LWE estimator, as the optimal number of samples when using the "2016 estimate" (except in the case of NTRU, where we assume $m = n$ samples). This is not necessarily the optimal number of samples for each specific block size when computed using a uSVP simulator. We therefore avoid making claims and comparisons regarding the exact cost of soving uSVP using the two algorithms, and propose our results as an intermediate step between using the current LWE estimator, and finding a theoretically cheapest attack using our simulators.

6.1 Observations

In almost all cases the mean required block size $\mathbb{E}(\text{succ. } \beta)$ is predicted to be larger than the LWE estimator currently suggests. Our results for using Progressive BKZ with $\tau = 1$ against NTRU-HPS are in line with what Dachman-Soled *et al.* [DSDGR20, Table 5] predict (NTRU-HPS being the only examined scheme in common). The increase in $\mathbb{E}(\text{succ. } \beta)$ may seem counterintuitive. Indeed, the Alkim *et al.* [ADPS16] methodology already aims to recover $\mathbb{E}(\text{succ. } \beta)$, with the simulators described in Section 4 capturing the success probability of smaller block sizes, possibly reducing the value of $\mathbb{E}(\text{succ. } \beta)$. Indeed, the increase seems to be mainly due to the use of the [CN11] simulator rather than the GSA for predicting the profile of a BKZ reduced basis (i.e. the right hand side of (3)). An illustrative example of this happening in the case of Kyber 512 can be see in Figure 10. Indeed, patching the LWE estimator to partially[6] use the [CN11] simulator, we obtain $\mathbb{E}(\text{succ. } \beta)$ of Kyber 512 (resp. Kyber 768, Kyber 1024) of 390 (resp. 636, 890), narrowing the gap with the predictions obtained in Table 2 by using our uSVP simulators. The small standard deviations reported in Table 2 suggest that the success probability of block sizes below $\mathbb{E}(\text{succ. } \beta)$ decrease quickly.

7 Conclusion

Overall, we believe that the experiments of Section 5 show that the techniques of Section 4 help to more accurately predict lattice reduction success probabilities for solving uSVP. We also believe the experiments show that in the case of short vectors sampled coefficientwise from bounded uniform distributions, it is the variance of the distribution, and not the exact probability mass function, that determines the hardness of the LWE instance. The uSVP simulators also seem to explain the success probability for smaller than expected block sizes reported in [AGVW17].

[6] For simplicity, our patch uses the GSA to predict the required block size to perform lattice reduction and the optimal number of samples, as before. It uses the [CN11] simulator for the basis profile output by BKZ, and to predict the block size required to win by running O_{SVP} on the last basis block.

					BKZ 2.0, $\tau = 15$				Progressive BKZ, $\tau = 1$		Progressive BKZ, $\tau = 5$	
scheme	n	q	σ_s	σ_e	β_{2016}	m	$\mathbb{E}(\text{succ. } \beta)$	$\sqrt{\mathbb{V}}(\text{succ. } \beta)$	$\mathbb{E}(\text{succ. } \beta)$	$\sqrt{\mathbb{V}}(\text{succ. } \beta)$	$\mathbb{E}(\text{succ. } \beta)$	$\sqrt{\mathbb{V}}(\text{succ. } \beta)$
Kyber 512	512	3329	1	1	381	484	386.06	2.56	389.53	2.88	385.70	2.32
Kyber 768	768	3329	1	1	623	681	634.41	2.96	638.23	3.30	634.00	2.66
Kyber 1024	1024	3329	1	1	873	860	891.13	3.31	895.24	3.66	890.63	2.96
LightSaber	512	8192	$\sqrt{5/2}$	$\sqrt{21}/2$	404	509	409.59	2.65	413.02	2.96	409.13	2.39
Saber	768	8192	$\sqrt{2}$	$\sqrt{21}/2$	649	709	659.97	3.04	663.74	3.36	659.45	2.72
FireSaber	1024	8192	$\sqrt{3/2}$	$\sqrt{21}/2$	890	920	908.22	3.31	912.23	3.64	907.64	2.94
ntruhps2048509	508	2048	$\sqrt{2/3}$	$\sqrt{1/2}$	374	508	375.93	2.58	379.56	2.92	375.71	2.36
ntruhps2048677	676	2048	$\sqrt{2/3}$	$\sqrt{\frac{127}{338}}$	521	676	522.78	2.82	526.77	3.18	522.67	2.57
ntruhps4096821	820	4096	$\sqrt{2/3}$	$\sqrt{\frac{51}{82}}$	621	820	628.78	2.83	632.54	3.17	628.43	2.55
ntruhrss701	700	8192	$\sqrt{2/3}$	$\sqrt{2/3}$	471	700	477.20	2.48	480.51	2.77	476.72	2.23

Table 2: Security estimates for some lattice schemes. The number of samples m used in the embedding for Kyber (LWE) and Saber (LWR) is chosen using the LWE estimator, to optimise the cost of the attack following the 2016 estimate for BKZ [ADPS16]. In the case of NTRU, the number of samples m is chosen equal to n. β_{2016} is the block size suggested by the LWE estimator. For BKZ and Progressive BKZ, $\mathbb{E}(\text{succ. } \beta)$ and $\sqrt{\mathbb{V}}(\text{succ. } \beta)$ are the mean and standard deviation of the distribution of first viable block sizes.

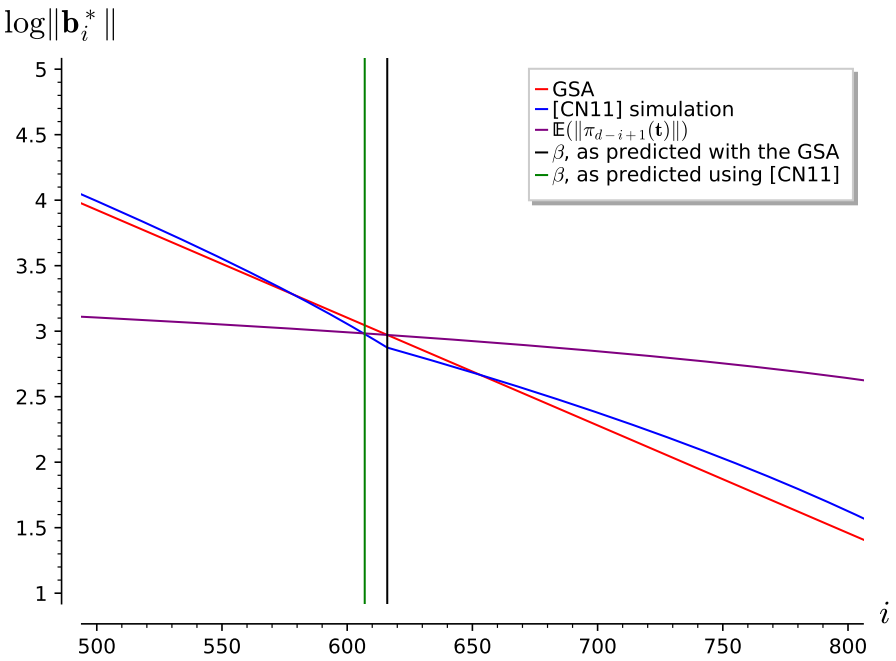


Fig. 10: Example plot showing the effect on the [ADPS16] methodology of using the [CN11] simulator rather than the GSA, in the case of Kyber 512. Due to the resulting higher basis profile, the GSA leads to picking a smaller block size. The required winning block size in the [ADPS16] methodology is the distance from the vertical line indicating the intersection to the final basis index d. Note that this plot is zoomed in ($d > 800$).

As part of our experiments, we also tested whether using Progressive BKZ with $\tau > 1$ could be beneficial for an attacker. This seems to be useful to some small degree from the point of view the of success probabilities, although BKZ seems to perform comparatively well. However, Progressive BKZ could be of interest to an attacker that wants to start performing lattice reduction as part of a long term attack, but initially has access to fewer resources[7] than necessary to run BKZ with the expected first viable block size. Progressive BKZ would then allow them to increase their resources as the attack progresses, with $\tau > 1$ allowing them to stop at an overall slightly smaller final block size.

We also notice that our preliminary estimates for the success probabilities of lattice reduction on cryptographically sized instances result in higher block sizes than output by the LWE estimator [APS15]. This seems to be mostly due to the use of a BKZ simulator rather than the GSA. A patch substituting the GSA with a BKZ simulator in the LWE estimator could mitigate this effect.

[7] Say, memory if using lattice sieving.

Acknowledgements. We would like to thank Martin Albrecht and Léo Ducas for useful conversations, and for their help simulating the LLL output profile.

References

ACD+18. Martin R. Albrecht, Benjamin R. Curtis, Amit Deo, Alex Davidson, Rachel Player, Eamonn W. Postlethwaite, Fernando Virdia, and Thomas Wunderer. Estimate all the LWE, NTRU schemes! In Dario Catalano and Roberto De Prisco, editors, *SCN 18*, volume 11035 of *LNCS*, pages 351–367. Springer, Heidelberg, September 2018.

ACPS09. Benny Applebaum, David Cash, Chris Peikert, and Amit Sahai. Fast cryptographic primitives and circular-secure encryption based on hard learning problems. In Halevi [Hal09], pages 595–618.

ADH+19. Martin R. Albrecht, Léo Ducas, Gottfried Herold, Elena Kirshanova, Eamonn W. Postlethwaite, and Marc Stevens. The general sieve kernel and new records in lattice reduction. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part II*, volume 11477 of *LNCS*, pages 717–746. Springer, Heidelberg, May 2019.

ADPS16. Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. Post-quantum key exchange - A new hope. In Thorsten Holz and Stefan Savage, editors, *USENIX Security 2016*, pages 327–343. USENIX Association, August 2016.

AFG14. Martin R. Albrecht, Robert Fitzpatrick, and Florian Göpfert. On the efficacy of solving LWE by reduction to unique-SVP. In Hyang-Sook Lee and Dong-Guk Han, editors, *ICISC 13*, volume 8565 of *LNCS*, pages 293–310. Springer, Heidelberg, November 2014.

AG11. Sanjeev Arora and Rong Ge. New algorithms for learning in presence of errors. In Luca Aceto, Monika Henzinger, and Jiri Sgall, editors, *ICALP 2011, Part I*, volume 6755 of *LNCS*, pages 403–415. Springer, Heidelberg, July 2011.

AGVW17. Martin R. Albrecht, Florian Göpfert, Fernando Virdia, and Thomas Wunderer. Revisiting the expected cost of solving uSVP and applications to LWE. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part I*, volume 10624 of *LNCS*, pages 297–322. Springer, Heidelberg, December 2017.

Alb17. Martin R. Albrecht. On dual lattice attacks against small-secret LWE and parameter choices in HElib and SEAL. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part II*, volume 10211 of *LNCS*, pages 103–129. Springer, Heidelberg, April / May 2017.

ALNSD20. Divesh Aggarwal, Jianwei Li, Phong Q. Nguyen, and Noah Stephens-Davidowitz. Slide reduction, revisited—filling the gaps in svp approximation. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020*, pages 274–295, Cham, 2020. Springer International Publishing.

APS15. Martin R Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. *JMC*, 2015.

AWHT16. Yoshinori Aono, Yuntao Wang, Takuya Hayashi, and Tsuyoshi Takagi. Improved progressive BKZ algorithms and their precise cost estimation by sharp simulator. In Fischlin and Coron [FC16], pages 789–819.

BCLv19. Daniel J. Bernstein, Chitchanok Chuengsatiansup, Tanja Lange, and Christine van Vredendaal. NTRU Prime. Technical report, National Institute of Standards and Technology, 2019. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions`.

BG14. Shi Bai and Steven D. Galbraith. Lattice decoding attacks on binary LWE. In Willy Susilo and Yi Mu, editors, *ACISP 14*, volume 8544 of *LNCS*, pages 322–337. Springer, Heidelberg, July 2014.

BMW19. Shi Bai, Shaun Miller, and Weiqiang Wen. A refined analysis of the cost for solving LWE via uSVP. In Johannes Buchmann, Abderrahmane Nitaj, and Tajje eddine Rachidi, editors, *AFRICACRYPT 19*, volume 11627 of *LNCS*, pages 181–205. Springer, Heidelberg, July 2019.

BSW18. Shi Bai, Damien Stehlé, and Weiqiang Wen. Measuring, simulating and exploiting the head concavity phenomenon in BKZ. In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018, Part I*, volume 11272 of *LNCS*, pages 369–404. Springer, Heidelberg, December 2018.

CCLS20. Hao Chen, Lynn Chua, Kristin Lauter, and Yongsoo Song. On the concrete security of lwe with small secret. Cryptology ePrint Archive, Report 2020/539, 2020. `https://eprint.iacr.org/2020/539`.

Che13. Yuanmi Chen. *Réduction de réseau et sécurité concrète du chiffrement complètement homomorphe*. PhD thesis, Université Paris Diderot, 2013.

CN11. Yuanmi Chen and Phong Q. Nguyen. BKZ 2.0: Better lattice security estimates. In Dong Hoon Lee and Xiaoyun Wang, editors, *ASIACRYPT 2011*, volume 7073 of *LNCS*, pages 1–20. Springer, Heidelberg, December 2011.

DKRV19. Jan-Pieter D'Anvers, Angshuman Karmakar, Sujoy Sinha Roy, and Frederik Vercauteren. SABER. Technical report, National Institute of Standards and Technology, 2019. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions`.

DSDGR20. Dana Dachman-Soled, Léo Ducas, Huijing Gong, and Mélissa Rossi. Lwe with side information: Attacks and concrete security estimation. In *CRYPTO*, 2020. `https://eprint.iacr.org/2020/292`.

dt16a. The FPLLL development team. fplll, a lattice reduction library. Available at `https://github.com/fplll/fplll`, 2016.

dt16b. The FPyLLL development team. fpylll, a python interface for fplll. Available at `https://github.com/fplll/fpylll`, 2016.

FC16. Marc Fischlin and Jean-Sébastien Coron, editors. *EUROCRYPT 2016, Part I*, volume 9665 of *LNCS*. Springer, Heidelberg, May 2016.

FHL+07. Laurent Fousse, Guillaume Hanrot, Vincent Lefèvre, Patrick Pélissier, and Paul Zimmermann. Mpfr: A multiple-precision binary floating-point library with correct rounding. *ACM Trans. Math. Softw.*, 33(2):13–es, June 2007.

GJS15. Qian Guo, Thomas Johansson, and Paul Stankovski. Coded-BKW: Solving LWE using lattice codes. In Gennaro and Robshaw [GR15], pages 23–42.

GN08a. Nicolas Gama and Phong Q. Nguyen. Finding short lattice vectors within Mordell's inequality. In Richard E. Ladner and Cynthia Dwork, editors, *40th ACM STOC*, pages 207–216. ACM Press, May 2008.

GN08b. Nicolas Gama and Phong Q. Nguyen. Predicting lattice reduction. In Nigel P. Smart, editor, *EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 31–51. Springer, Heidelberg, April 2008.

GR15. Rosario Gennaro and Matthew J. B. Robshaw, editors. *CRYPTO 2015, Part I*, volume 9215 of *LNCS*. Springer, Heidelberg, August 2015.

Hal09. Shai Halevi, editor. *CRYPTO 2009*, volume 5677 of *LNCS*. Springer, Heidelberg, August 2009.

How07. Nick Howgrave-Graham. A hybrid lattice-reduction and meet-in-the-middle attack against NTRU. In Alfred Menezes, editor, *CRYPTO 2007*, volume 4622 of *LNCS*, pages 150–169. Springer, Heidelberg, August 2007.

HPS11. Guillaume Hanrot, Xavier Pujol, and Damien Stehlé. Analyzing blockwise lattice algorithms using dynamical systems. In Phillip Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 447–464. Springer, Heidelberg, August 2011.

Kan87. Ravi Kannan. Minkowski's convex body theorem and integer programming. *Mathematics of Operations Research*, 12(3):415–440, Aug 1987.

KF15. Paul Kirchner and Pierre-Alain Fouque. An improved BKW algorithm for LWE with applications to cryptography and lattices. In Gennaro and Robshaw [GR15], pages 43–62.

KK51. J.F. Kenney and E.S. Keeping. *Mathematics of Statistics*, volume 2. Van Nostrand, 1951.

LLL82. H.W .Jr. Lenstra, A.K. Lenstra, and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261:515–534, 1982.

LM09. Vadim Lyubashevsky and Daniele Micciancio. On bounded distance decoding, unique shortest vectors, and the minimum distance problem. In Halevi [Hal09], pages 577–594.

LN13. Mingjie Liu and Phong Q. Nguyen. Solving BDD by enumeration: An update. In Ed Dawson, editor, *CT-RSA 2013*, volume 7779 of *LNCS*, pages 293–309. Springer, Heidelberg, February / March 2013.

MR09. Daniele Micciancio and Oded Regev. Lattice-based cryptography. In Daniel J. Bernstein, Johannes Buchmann, and Erik Dahmen, editors, *Post-Quantum Cryptography*, pages 147–191. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

MW16. Daniele Micciancio and Michael Walter. Practical, predictable lattice basis reduction. In Fischlin and Coron [FC16], pages 820–849.

NIS16. NIST. Submission requirements and evaluation criteria for the Post-Quantum Cryptography standardization process, 2016.

Reg09. Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM*, 56(6):1–40, Sep 2009.

SAB+19. Peter Schwabe, Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Gregor Seiler, and Damien Stehlé. CRYSTALS-KYBER. Technical report, National Institute of Standards and Technology, 2019. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions`.

Sch03. Claus Peter Schnorr. Lattice reduction by random sampling and birthday methods. In *Annual Symposium on Theoretical Aspects of Computer Science*, pages 145–156. Springer, 2003.

SE91. Claus-Peter Schnorr and M Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. In *International Symposium on Fundamentals of Computation Theory*, pages 68–85. Springer, 1991.

SE94. Claus-Peter Schnorr and Martin Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. *Mathematical programming*, 66(1-3):181–199, 1994.

SR02. Murray D Smith and C Rose. *Mathematical Statistics with Mathematica®*, page 264. Springer Berlin, 2002.

ZCH+19. Zhenfei Zhang, Cong Chen, Jeffrey Hoffstein, William Whyte, John M. Schanck, Andreas Hulsing, Joost Rijneveld, Peter Schwabe, and Oussama Danba. NTRUEncrypt. Technical report, National Institute of Standards and Technology, 2019. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions`.

A Scaling lattices in practice

As mentioned in Section 2, given LWE samples $(\boldsymbol{A}, \boldsymbol{c} = \boldsymbol{A}\boldsymbol{s} + \boldsymbol{e} \bmod q)$, it is possible to construct a lattice basis that embeds a shortest vector containing scaled and/or balanced components of $\boldsymbol{s}$ or $\boldsymbol{e}$. In the case of simply scaling the secret by a factor ν, one approach is to use the [BG14] embedding (1),

$$\mathbf{B} = \begin{pmatrix} \mathbf{0} & q\mathbf{I}_m & \mathbf{0} \\ \nu\mathbf{I}_n & -\mathbf{A}^\top & \mathbf{0} \\ \mathbf{0} & \mathbf{c} & c \end{pmatrix},$$

which contains in its integer span the vector $\boldsymbol{t} = (* \mid \boldsymbol{s} \mid 1) \cdot \boldsymbol{B} = (\nu\,\boldsymbol{s} \mid \boldsymbol{e} \mid c)$ for suitable values of $*$. In theory, the optimal value of ν could be any real not smaller than 1. In practice however, lattice reduction libraries such as FpLLL [dt16a] require input bases to have integer coefficients. In order to run experiments, this issue can be avoided by using the standard approach of clearing denominators. The idea is to use a rational approximation $\nu \approx x/y$, with $x, y \in \mathbb{Z}$ and $y \neq 0$. Then, one can define a basis $\boldsymbol{B}_1$ obtained by clearing the denominator

$$\mathbf{B}_1 = \begin{pmatrix} \mathbf{0} & yq\mathbf{I}_m & \mathbf{0} \\ x\mathbf{I}_n & -y\mathbf{A}^\top & \mathbf{0} \\ \mathbf{0} & y\mathbf{c} & yc \end{pmatrix} \approx y \cdot \boldsymbol{B}.$$

This has the effect of scaling every vector in the real span of $\boldsymbol{B}$. Assuming for simplicity the win condition from the [ADPS16] methodology, it is an immediate computation that the success condition for the scaled problem is equivalent to that of the original problem using a rational approximation of ν,

$$\|\pi_{d-\beta+1}(y \cdot \boldsymbol{t})\| \leq \left\|(y \cdot \boldsymbol{b})^*_{d-\beta+1}\right\| \iff \|\pi_{d-\beta+1}(\boldsymbol{t})\| \leq \left\|\boldsymbol{b}^*_{d-\beta+1}\right\|.$$

In the case of secret distributions with non-zero mean μ, two simple approaches can be used to generate an embedding with a target vector containing a balanced version of $\boldsymbol{s}$. This can be useful since it allows for a more aggressive choice of ν. For example, this is what we assume would be done by an attacker when we investigate the cost of solving uSVP with binary secrets in Section 5. The first approach is to map any LWE samples $(\boldsymbol{A}, \boldsymbol{c})$ into samples $(\boldsymbol{A}, \boldsymbol{c} - \boldsymbol{A}\boldsymbol{\mu})$, where $\boldsymbol{\mu} = (\mu, \ldots, \mu)$. This works since

$$(* \mid \boldsymbol{s} - \boldsymbol{\mu} \mid 1) \cdot \begin{pmatrix} \mathbf{0} & q\mathbf{I}_m & \mathbf{0} \\ \nu\mathbf{I}_n & -\mathbf{A}^\top & \mathbf{0} \\ \mathbf{0} & \mathbf{c} - \boldsymbol{A}\boldsymbol{\mu} & c \end{pmatrix} = (\nu\,(\boldsymbol{s} - \boldsymbol{\mu}) \mid \boldsymbol{e} \mid c).$$

Recovering the target vector on the right hand side results in solving LWE. Yet, the first n coefficients in the target vector are now distributed symmetrically around 0, rather than around μ. For example, if the secret distribution were to be binary, $\mathcal{U}(\{0, 1\})$, using $\nu = 2$ the first n coefficients of the target vector would be distributed uniformly in the set $\{-1, 1\}$.

An alternative basis for centring the secret distribution is

$$(* \mid \boldsymbol{s} \mid 1) \cdot \begin{pmatrix} \mathbf{0} & q\mathbf{I}_m & \mathbf{0} \\ \nu\mathbf{I}_n & -\mathbf{A}^\top & \mathbf{0} \\ -\nu\boldsymbol{\mu} & \mathbf{c} & c \end{pmatrix} = (\nu\,(\boldsymbol{s} - \boldsymbol{\mu}) \mid \boldsymbol{e} \mid c).$$

In the case that the error distribution were to have mean $\mu \neq 0$, mapping samples $(\boldsymbol{A}, \boldsymbol{c}) \mapsto (\boldsymbol{A}, \boldsymbol{c} - \boldsymbol{\mu})$ would have the same effect. In all cases, an integer basis can be obtained by appropriately clearing the denominators of any rational approximations of ν and μ.

B Exact square root expectation of the χ^2_d distribution

We note that although $\mathbb{E}(\sigma^2\chi^2_d) = \sigma^2 d$, it is not the case that $\mathbb{E}\left(\sqrt{\sigma^2\chi^2_d}\right) = \sigma\sqrt{d}$. By direct computation, if $x \leftarrow \chi^2_d$, then

$$\begin{aligned} \mathbb{E}\left(\sqrt{\sigma^2 \cdot x}\right) = \sigma\mathbb{E}(\sqrt{x}) &= \frac{\sigma}{2^{d/2}\Gamma\left(\frac{d}{2}\right)} \int_0^\infty x^{1/2} x^{d/2-1} e^{-x/2}\, dx \\ &= \frac{\sqrt{2}\sigma\Gamma(\frac{d+1}{2})}{\Gamma\left(\frac{d}{2}\right)} \xrightarrow{d\to\infty} \sigma\sqrt{d}. \end{aligned}$$

C LLL "Z-shape" simulation

As part of our uSVP simulations, we use an LLL simulator. This allows one to predict the characteristic *Z-shape* phenomenon [How07] that occurs when reducing bases of q-ary lattices.

The Z-shape nickname refers to the basis profile output by LLL when reducing a q-ary lattice basis in Hermite normal form (HNF) with the q-vectors set as the first basis vectors, e.g. as in (1). In such cases, most of the q-vectors will not be altered by LLL, since they are orthogonal and short. This results in the basis profile having a flat *head* corresponding to the first Gram–Schmidt vectors being q-vectors. Depending on the lattice volume and rank, the final Gram–Schmidt vectors will be unit vectors, obtained from the identity matrix minor in the HNF basis, resulting in a flat *tail* in the profile. The middle indices of the log-plot of the basis profile will be located along a straight line with the slope predicted by the GSA for LLL, $\log\alpha = -2\log\delta$, where δ is the root Hermite factor for LLL, $\delta \approx 1.02$. An example of a Z-shape basis profile and the output of the LLL simulator for a basis of the same rank and volume can be found in Figure 11.

The LLL simulator essentially predicts the Z-shape by first computing the GSA slope section of the profile. This is achieved by noticing that the longest vector in this section will have log-norm $\log q$. We model each subsequent vector as having log-norm shorter by $\log\alpha$, until the next log-norm would be negative. The number of head q-vectors is computed so that the total volume of the basis

is equal to that of the lattice, with any adjustments to volume (due to requiring an integer number of head vectors) distributed through the GSA slope. The tail of unit length vectors is then added, until the basis has the correct lattice rank. An implementation of this simulator can be found in our codebase.

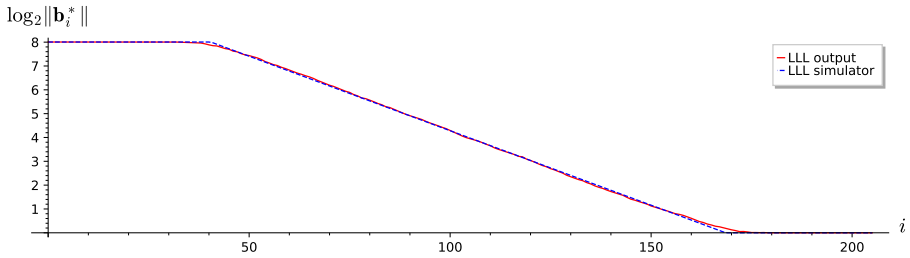


Fig. 11: Comparison between the output profile of LLL on a $n = 100$ instance parametrised as in Table 1, averaged over 25 instances, and the output of the LLL simulator used for our estimates.