Fast Slicer for Batch-CVP: Making Lattice Hybrid Attacks Practical

Alexander Karenin[1], Elena Kirshanova[2], Alexander May[4], and Julian Nowakowski[3]

[1] Technology Innovation Institute, Abu Dhabi, United Arab Emirates
tremeloesalex@gmail.com
[2] Technology Innovation Institute, Abu Dhabi, United Arab Emirates
elenakirshanova@gmail.com
[3] Ruhr University Bochum, Bochum, Germany julian.nowakowski@rub.de
[4] Ruhr University Bochum, Bochum, Germany alex.may@rub.de

Abstract. We study the practicality of a primal hybrid lattice attack on LWE. Despite significant research efforts, the state-of-the-art in practical LWE record computations so far is the plain primal attack with Kannan's embedding.

Building on an idea by Espitau and Kirchner, Bernstein proposed in 2023 an LWE hybrid attack that asymptotically outperforms the primal attack. In a nutshell, Bernstein's attack enumerates some coordinates of an LWE key and uses the sophisticated Batch-CVP *(Randomized) Slicer* algorithm to solve LWE in a dimension-reduced lattice. The practical implications of this attack however remain widely unclear. One of the major obstacles for judging practicality is the lack of a fast, fully functional Slicer implementation. For the first time, we provide an efficient Slicer implementation that includes all required algorithmic ingredients like locality sensitive hashing.

Building on our Slicer implementation, we implement a generalization of Bernstein's algorithm. While Bernstein's attack works only for LWE, ours also applies to a more general BDD setting. Let $(\mathbf{B}, \mathbf{t})$ be a BDD instance, where the target $\mathbf{t}$ is off from the d-dimensional lattice $\mathcal{L}(\mathbf{B})$ by some error $\mathbf{e}$, sampled coordinate-wise from a distribution $\mathcal{D}$. We show that for hard BDD instances, our BDD hybrid asymptotically speeds up primal's complexity of $T = 2^{0.292d+o(d)}$ down to $T^{1-\mathcal{K}}$, where $\mathcal{K} \approx \big(1 + \frac{\mathrm{H}(\mathcal{D})}{0.058}\big)^{-1}$ with $\mathrm{H}(\cdot)$ the Shannon entropy. Depending on $\mathcal{D}$, the constant $\mathcal{K}$ can be small, making practical improvements difficult.

We test our Slicer-based implementation inside an implementation of our BDD hybrid lattice attack to tackle LWE instances. We choose two ternary LWE secrets with different entropies $\mathrm{H}(\mathcal{D})$ as used in NTRU, and the centered binomial distribution as used in Kyber. For all three distributions in all tested LWE dimensions $n \in [160, 210]$, our Slicer-based implementation practically demonstrates measurable speedups over the primal attack, up to a factor of 5. We also show that for parameters as originally suggested by Regev, the hybrid attack cannot improve over primal.

1 Introduction

Since the invention of lattice based cryptography [Ajt96, HPS98, Reg05] there have been significant cryptanalytic efforts to foster our understanding of the hardness of the underlying lattice problems. In a nutshell, lattice based encryption encodes a message as a lattice point $\mathbf{v}$, disturbed by a small error vector $\mathbf{e}$. Thus, the security of lattice based encryption relies on the hardness of solving the *closest vector problem* (CVP) of recovering $\mathbf{v}$ from the *target* $\mathbf{t} = \mathbf{v} + \mathbf{e}$. To ensure unique decoding of $\mathbf{v}$, the error $\mathbf{e}$ is chosen sufficiently small. Accordingly, the problem is also called a *bounded distance decoding* (BDD) problem, a promise version of CVP with small built-in distance to the lattice.

Not only lead BDD solutions to message recovery, but also to secret key recovery. This is easily seen for LWE instances with public key $(\mathbf{A}, \mathbf{b}) \in \mathbb{Z}_q^{m \times n} \times \mathbb{Z}_q^m$ that satisfy the key equation $\mathbf{A}\mathbf{s}_1 + \mathbf{s}_2 = \mathbf{b} \bmod q$ for some small secret $\mathbf{s}_1$ and small error $\mathbf{s}_2$. The matrix $\mathbf{A}$ defines an $(m+n)$-dimensional lattice

$$\mathcal{L} := \left\{ (\mathbf{x}, \mathbf{y})^T \in \mathbb{Z}^m \times \mathbb{Z}^n \mid \mathbf{x} \equiv \mathbf{A}\mathbf{y} \mod q \right\},$$

which contains the vector $\mathbf{v} := (\mathbf{b} - \mathbf{s}_2, \mathbf{s}_1)^T \equiv (\mathbf{A}\mathbf{s}_1, \mathbf{s}_1)^T \mod q$. Since $\mathbf{v} \in \mathcal{L}$ is close to $\mathbf{t} := (\mathbf{b}, 0^n)^T$, it follows that solving BDD on $\mathcal{L}$ with target $\mathbf{t}$ recovers the LWE secret $\mathbf{s}_1$.

Enumerate-then-reduce hybrid. A natural way to solve BDD is to *embed* the target $\mathbf{t}$ into the lattice. To this end, one works in essence with $\mathcal{L}' := \mathcal{L} + \mathbf{t} \cdot \mathbb{Z}$, thereby inducing a new shortest vector $\mathbf{e} = \mathbf{t} - \mathbf{v}$ in the enhanced lattice $\mathcal{L}'$. Thus, the search for a closest vector is reduced to the solution of a *shortest vector problem* (SVP), which is algorithmically solved via lattice reduction, as implemented in G6K [ADH+19]. This method of reducing BDD to SVP is called Kannan's Embedding [Kan83], and was successfully applied in many cases. As an example, see e.g. the lattice attacks of Coppersmith and Shamir [CS97] for NTRU message recovery [HPS98]. This pure lattice reduction approach was extended in [MS01] with a hybrid of key enumeration, followed by a dimension-reduced lattice reduction (also known as *drop-and-solve* [ACW19]). In more generality, one may enumerate keys [GMN23, BM24] and then use the *lattices with hints* framework [DDGR20, DGHK23, MN23] to speed up lattice reduction. Notice, though, that this enumerate-then-reduce hybrid approach suffers from a multiplication of run times for enumeration and lattice reduction, since one has to run lattice reduction for each key guess anew. As a result, unless the key is sampled from an *extremely* narrow distribution, this approach is inferior to Kannan's Embedding.

BDD decoding, reduce-then-enumerate. Besides Kannan's Embedding, another practical approach is to solve BDD directly via lattice reduction, followed by an application of Babai's Nearest Plane algorithm [Bab86]. This approach however only succeeds if the lattice is extremely well reduced, thereby making it inferior to Kannan's Embedding. Lindner and Peikert [LP11] proposed to enhance Babai's algorithm by enumerating all nearest planes within a certain radius. This method was further improved by Liu and Nguyen [LN13] via rerandomization of lattice reduction, which in turn allowed for larger success probabilities, and for a balance of the costs for lattice reduction and decoding to a nearest lattice point. As a consequence, the hybrid approach of Liu-Nguyen has as runtime the sum of the run times of lattice reduction and an a posteriori enumeration process.

Howgrave-Graham's hybrid. Howgrave-Graham [How07] proposed another BDD algorithm that combines lattice reduction with enumeration, where enumeration is done in a Meet-in-the-Middle fashion. However, the success probability of this Meet-in-the-Middle approach was left as an open problem. It was shown by Wunderer and Nguyen [Wun19, Ngu21] that the success probability is too small to make Howgrave-Graham's hybrid approach competitive in practice.

Practical records, and other approaches. The state-of-the-art for practical cryptanalysis lattice records is Kannan's Embedding [Dar13, Boc25]. While several other methods for attacking lattice based assumptions have been proposed like dual attacks [MR09, PS24, CST24], BKW type methods [BKW00, GJMS17, WBWL23, DEL25], Meet-in-the-Middle key enumeration [May21, GM23], or algebraic methods [ACFP14, Ste24], most of them lack practicality for tackling typical parameter sets, as used for today's standard lattice-based cryptography [DKL⁺18, BDK⁺18]. Even if theoretically applicable, none of the above methods currently offers a competitive implementation that is suited to achieve practical cryptanalysis records.

Batched-CVP, reduce-then-batch. In 2020, Espitau and Kirchner [EK20] sketched a BDD algorithm, that first reduces a lattice, then solves a CVP on the tail of the basis, and finishes BDD using Babai's Nearest Plane algorithm. At the same time, Doulgerakis, Laarhoven, de Weger [DLdW19] and Ducas, Laarhoven, van Woerden [DLvW20] introduced a sophisticated technique for CVP, called the *Randomized Slicer*. The Randomized Slicer allows to solve *large* batches of CVP instances at the cost of a single CVP instance. Proof-of-concept implementations of the Randomized Slicer were released by van Woerden [vW20], and Wang, Xia and Gu [WXG25]. As one of the potential applications of the Randomized Slicer, [DLvW20] mentions lattice hybrid attacks.

Bernstein [Ber23, Ber22] turned the sketch of Espitau-Kirchner in combination with the Randomized Slicer into a lattice hybrid algorithm that after lattice reduction solves a batch of CVP instances. In the LWE case, the batch of CVP instances correspond to an enumeration of a subkey of $\mathbf{s}_2$. This is especially interesting, since modern lattice standards like Kyber [BDK⁺18] and Dilithium [DKL⁺18] sample their secret keys $\mathbf{s}_1, \mathbf{s}_2$ from narrow distributions that make subkey enumeration easier than in Regev's original scheme [Reg05].

Bernstein showed that his lattice hybrid approach achieves asymptotic improvements over pure primal lattice attacks. However, the practical implications of Bernstein's lattice hybrid remained unclear so far.

Comparison of hybrid lattice algorithms. We compare the so far discussed lattice hybrids in Table 1. Here, we exclude Howgrave-Graham's hybrid because of its small success probability.

Table 1. Comparison of practical lattice hybrid algorithms.

	dim reduce	add costs
Enumerate-then-reduce [MS01, DDGR20]	✓	—
BDD decoding [LP11, LN13]	—	✓
Batch-CVP [Ber22, Ber23]	✓	✓

The first column of Table 1 indicates whether a method works in dimension reduced lattices, which speeds up lattice reduction. The second column indicates whether the two costs of enumeration and lattice reduction are additive. As discussed before, both BDD decoding and Batch-CVP achieve additive costs. Thus, Batch-CVP combines the best of the worlds, and therefore deserves practical investigation.

1.1 Our contributions

Tail-BDD. In [EK20], Espitau and Kirchner briefly mention the following BDD algorithm: "Once a highly reduced basis is found, it is enough to compute a CVP on the tail of the basis, and finish with Babai's algorithm." We work out the details of such an algorithm that we call *Tail-BDD*, and provide a full fledged analysis for it. Interestingly, our condition for successfully solving BDD with algorithm *Tail-BDD* is identical to the required condition for the primal attack with Kannan's embedding. This indicates that the starting point of using plain *Tail-BDD* is already competitive with the primal attack, but comes with the benefit of allowing for further improvement by using a hybrid of key enumeration in combination with Batch-CVP from the Randomized Slicer.

BDD hybrid attack. We reformulate the LWE algorithm of Bernstein [Ber22, Ber23] and generalize to a BDD setting, which might be of independent interest. We call the resulting algorithm *BDD hybrid attack*. Let $(\mathbf{B}, \mathbf{t})$ be a BDD instance with d-dimensional lattice basis $\mathbf{B} \in \mathbb{R}^{m \times d}$ and $\mathbf{t} = \mathbf{v} + \mathbf{e}$, where $\mathbf{v} \in \mathcal{L}(\mathbf{B})$ is the solution, and $\mathbf{e}$ the error.

The general idea of the *BDD hybrid attack* is to *enumerate* κ coordinates of $\mathbf{e}$. Every of the M guesses defines a target $\mathbf{t}^{(i)}$, $i = 1, \ldots, M$. These guesses lead to a multi-target BDD instance $(\mathbf{B}', \mathbf{t}^{(1)}, \ldots, \mathbf{t}^{(M)})$ with a basis $\mathbf{B}'$ of *reduced dimension* $d - \kappa$.

We enhance our *Tail-BDD* algorithm using the Randomized Slicer to find all solutions of the multi-target BDD instance $(\mathbf{B}', \mathbf{t}^{(1)}, \ldots, \mathbf{t}^{(M)})$. We call the resulting algorithm *Batch-Tail-BDD*. The BDD solution of the so-called *golden target* $\mathbf{t}^{(i)}$, corresponding to the correct guess of the κ coordinates of $\mathbf{e}$, eventually reveals the solution $\mathbf{v}$ of the original BDD instance $(\mathbf{B}, \mathbf{t})$.

Asymptotics of BDD hybrid attack. For LWE instances with d-dimensional lattice basis and secrets sampled from distribution $\mathcal{D}$, Bernstein [Ber23] showed for his LWE hybrid attack an asymptotic speedup from primal's complexity of $T = 2^{0.292d+o(d)}$ down to $T^{1-\mathcal{K}}$, where $\mathcal{K} \approx \left(1 + \frac{\log_2(|\operatorname{supp}(\mathcal{D})|)}{0.058}\right)^{-1}$ with $\operatorname{supp}(\cdot)$ the support and constant 0.058 coming from the Randomiced Slicer. Bernstein left it as an open problem, whether $|\operatorname{supp}(\mathcal{D})|$ can be improved by a more refined key enumeration process.

Using a recent result of Glaser, May and Nowakowski [GMN23] we answer this open problem in the affirmative. Namely, we show that both Bernstein's and our generalized BDD hybrid algorithm achieve $\mathcal{K} \approx \left(1 + \frac{\mathrm{H}(\mathcal{D})}{0.058}\right)^{-1}$ with $\mathrm{H}(\cdot)$ the Shannon entropy. We have $\mathrm{H}(\mathcal{D}) \leq \log_2 |\operatorname{supp}(\mathcal{D})|$ with equality iff $\mathcal{D}$ is the uniform distribution.

To provide an illustrative numerical example let $\mathcal{D} = \mathcal{B}(3)$ be the Binomial distribution with support $\{-3, \ldots, 3\}$ and $\mathrm{H}(\mathcal{B}(3)) = 2.333 < 2.807 = \log_2(|\operatorname{supp}(\mathcal{B}(3))|)$. Using Bernstein's formula, one achieves $\mathcal{K} = (1 + \frac{2.807}{0.058})^{-1} = 0.020$, whereas we improve to $\mathcal{K} = (1 + \frac{2.333}{0.058})^{-1} = 0.024$. However, even with our asymptotic improvement, $\mathcal{K}$ is so small that the realization of a practical speedup for BDD instances in dimensions d of cryptographic interest remains difficult.

We also show that for wide key distributions as originally proposed in [Reg05] our BDD hybrid attack cannot achieve asymptotical speedups. Hybrid attacks thus indicate that modern practical lattice schemes like Kyber, Dilithium or NTRU have to pay a (small) price for switching to rather narrow key distributions.

Slicer implementation. We provide for the first time an implementation of the Randomized Slicer that implements its full functionality.[5] The proof-of-concept implementations [vW20, WXG25] do not include any nearest neighbor techniques that are crucial to achieve the claimed speedups of the slicer technique. Our implementation is the first that is suited to attain practical speedups, and to demonstrate practical superiority of a BDD hybrid attack over a primal attack using Kannan's Embedding.

[5] There is concurrent work [XWG25] that also provides an implementation of randomized slicer with nearest neighbor search.

Comparison of our BDD hybrid with primal. To test the practical effectiveness of our Slicer implementation inside the BDD hybrid attack, we implemented scaled down versions of Kyber with the centered binomial distribution $\mathcal{B}(3)$ in LWE dimensions $n \in [140, 170]$, and of NTRU-type encryption [CDH+21] with ternary distributions of two different entropies in dimensions $n \in [160, 210]$. We ran both a pure primal Kannan's Embedding lattice attack, as well as a BDD hybrid attack with our Randomized Slicer implementation. For all instances, we improved over the primal attack. The achieved speedups in the considered dimensions for $\mathcal{B}(3)$ and for the larger entropy ternary distribution are up to a factor of 3.4, for the lower entropy ternary distribution we even achieve speedups up to a factor of 4.5.

Artifacts. The implementation of our slicer with scripts to reproduce the figures are available at https://github.com/ElenaKirshanova/g6k_hybrid/tree/ac_artifact

Organization of the paper. In Section 2.1 we introduce some useful notions and recap the Randomized Slicer. In Section 3 we present the Tail-BDD algorithm together with a detailed analysis of its success condition, followed by its batched version Batched-Tail-BDD and the generalization of Bernstein's algorithm to a BDD hybrid attack. In Section 4 we describe our full-fledged implementation of the Randomized Slicer and the practical speedups achieved using it inside BDD hybrid.

2 Preliminaries

2.1 Notations

For a positive integer n, we set $[n] := \{1, 2, \ldots, n\}$. Vectors are denoted by bold lower case letters, e.g., $\mathbf{v}$. Matrices are denoted by bold upper case letters, e.g., $\mathbf{B}$. We write $\mathbf{B} = (\mathbf{b}_0, \ldots, \mathbf{b}_{d-1})$ for a matrix with columns $\mathbf{b}_i$. The transpose of a vector $\mathbf{v}$ is $\mathbf{v}^T$. For a set $S \subseteq \mathbb{R}^m$, we denote by $\operatorname{span}(S)$ the linear span of S over $\mathbb{R}$, i.e., the smallest linear subspace $V \subseteq \mathbb{R}^m$ containing S. The n-dimensional identity matrix is denoted $\mathbf{I}_n$.

Probabilities. If $\mathcal{D}$ is a distribution, then $X \leftarrow \mathcal{D}$ denotes that X is a random variable, drawn from $\mathcal{D}$. Similarly, if S is a set, then $X \leftarrow S$ denotes that X is drawn uniformly at random from S. The support of a distribution $\mathcal{D}$ is denoted by $\operatorname{supp}(\mathcal{D})$. The Shannon entropy of $\mathcal{D}$ is denoted $\mathrm{H}(\mathcal{D})$, i.e.,

$$\mathrm{H}(\mathcal{D}) := -\sum_{i \in \operatorname{supp}(\mathcal{D})} p_i \log_2(p_i), \quad \text{where } p_i := \Pr_{X \leftarrow \mathcal{D}}[X = i].$$

Note that $\mathrm{H}(\mathcal{D}) \leq \log_2(|\operatorname{supp}(\mathcal{D})|)$ with equality if and only if $\mathcal{D}$ is the uniform distribution (see, e.g., [CT06, Theorem 2.6.4]).

The *centered binomial distribution* with parameter $\eta \in \mathbb{N}$, denoted $\mathcal{B}(\eta)$, is defined as

$$\Pr_{X \leftarrow \mathcal{B}(\eta)}[X = i] := \begin{cases} \binom{2\eta}{\eta+i} 2^{-2\eta}, & \text{if } i \in \{-\eta, -\eta+1, \dots \eta\} \\ 0, & \text{else} \end{cases} .$$

The *weighted ternary distribution* with parameter $w \in [0, 1]$, denoted $\mathcal{T}(w)$, is defined as

$$\Pr_{X \leftarrow \mathcal{T}(w)}[X = i] := \begin{cases} 1 - w, & \text{if } i = 0 \\ w/2, & \text{if } i = \pm 1 \\ 0, & \text{else} \end{cases} .$$

Note that for $\mathbf{v} \leftarrow \mathcal{T}(w)^d$, we have on expectation that $\|\mathbf{v}\|^2 = w^2 d$.

Euclidean Geometry. The Euclidean inner product and norm are denoted by $\langle \cdot, \cdot \rangle$ and $\| \cdot \|$, respectively. For a linear subspace $U \subset \mathbb{R}^m$, we denote the orthogonal complement by $U^\perp$. For a vector $\mathbf{v} \in \mathbb{R}^m$, we denote by $\pi_U(\mathbf{v})$ the orthogonal projection of $\mathbf{v}$ onto U. Equivalently, $\pi_U(\mathbf{v}) := \mathbf{U}(\mathbf{U}^T\mathbf{U})^{-1}\mathbf{U}^T\mathbf{v}$, where $\mathbf{U}$ is any matrix whose columns form a basis of U. For an ordered set of linearly independent vectors $(\mathbf{b}_0, \dots, \mathbf{b}_{d-1})$ and $1 \le i \le d$, we define the short hand notations $\pi_i(\cdot) := \pi_{\text{span}\{\mathbf{b}_0,\dots,\mathbf{b}_{i-1}\}}(\cdot)$ and $\pi_i^\perp(\cdot) := \pi_{\text{span}\{\mathbf{b}_0,\dots,\mathbf{b}_{i-1}\}^\perp}(\cdot)$. For $i = 0$, we extend the definitions of $\pi_i(\cdot)$ and $\pi_i^\perp(\cdot)$ as $\pi_0(\cdot) := \pi_{\{\mathbf{0}\}}(\cdot)$ and $\pi_0(\cdot) := \pi_{\{\mathbf{0}\}^\perp}(\cdot)$. Equivalently, $\pi_0(\mathbf{x}) := \mathbf{0}$ and $\pi_0^\perp(\mathbf{x}) := \mathbf{x}$, for every $\mathbf{x} \in \mathbb{R}^m$. The Gram-Schmidt orthogonalization of $(\mathbf{b}_0, \dots, \mathbf{b}_{d-1})$ is $\mathbf{b}_0^*, \dots, \mathbf{b}_{d-1}^*$, where $\mathbf{b}_i^* := \pi_i^\perp(\mathbf{b}_i)$.

2.2 Lattices

For a non-singular matrix $\mathbf{B} \in \mathbb{R}^{m\times d}$, we denote by $\mathcal{L}(\mathbf{B}) := \{\mathbf{B}\mathbf{x} \mid \mathbf{x} \in \mathbb{Z}^n\}$ the lattice generated by $\mathbf{B}$. We call $\mathbf{B} \in \mathbb{R}^{m\times d}$ a *basis* of $\mathcal{L}(\mathbf{B})$, and d the *dimension* of $\mathcal{L}(\mathbf{B})$. If $m = d$, then we call $\mathcal{L}(\mathbf{B})$ a *full-rank* lattice. Two bases $\mathbf{B}_1, \mathbf{B}_2 \in \mathbb{R}^{m\times d}$, generate the same lattice if and only if there exists a *unimodular* matrix $\mathbf{U} \in \mathbb{Z}^{d\times d}$ (i.e., $\det(\mathbf{U}) = \pm 1$) with $\mathbf{B}_1 = \mathbf{B}_2\mathbf{U}$. Every integer lattice $\mathcal{L} \subseteq \mathbb{Z}^m$ has a canonical basis, called the *Hermite Normal form (HNF)* basis. Given a basis $\mathbf{B}$ of some integer lattice $\mathcal{L}$, the HNF basis of $\mathcal{L}$ can be computed in polynomial time. If $\mathcal{L}$ has full-rank, then the HNF basis is an upper-triangular matrix with non-zero diagonal.

The *first successive minimum* of a lattice $\mathcal{L}$ is $\lambda_1(\mathcal{L}) := \min_{\mathbf{v}\in\mathcal{L}\setminus\{\mathbf{0}\}} \|\mathbf{v}\|$. For a lattice $\mathcal{L}$ with basis $\mathbf{B}$, its *determinant* is defined as $\det(\mathcal{L}) := \sqrt{\det(\mathbf{B}^T\mathbf{B})}$. A lattice $\mathcal{L}$ is called called *q-ary*, if $q\mathbb{Z}^m \subseteq \mathcal{L} \subseteq \mathbb{Z}^m$, for some $q \in \mathbb{N}$. For a lattice with basis $\mathbf{B} = (\mathbf{b}_0, \dots, \mathbf{b}_{d-1})$ and $i < j \le d$, we define $\mathbf{B}_{[i,j)} := (\pi_i^\perp(\mathbf{b}_i), \dots, \pi_i^\perp(\mathbf{b}_{j-1}))$. The lattice $\mathcal{L}(\mathbf{B}_{[i,j)})$ is called a *projected sublattice* of $\mathcal{L}(\mathbf{B})$. Any vector $\widetilde{\mathbf{v}} = \mathbf{B}_{[i,j)}\mathbf{x} \in \mathcal{L}(\mathbf{B}_{[i,j)})$ can be *lifted* to a vector $\mathbf{v} \in \mathcal{L}(\mathbf{B})$ by setting $\mathbf{v} := \mathbf{B}(\mathbf{u}, \mathbf{x}, 0^{d-i-j})^T$, where $\mathbf{u} \in \mathbb{Z}^i$ is an arbitrary integer vector, e.g., $\mathbf{u} = 0^i$. Note that, for any choice of $\mathbf{u}$, the lifted vector $\mathbf{v}$ satisfies $\pi_i^\perp(\mathbf{v}) = \widetilde{\mathbf{v}}$.

The Gaussian Heuristic. For a random[6] d-dimensional lattice $\mathcal{L}$ and $V \subseteq \mathrm{span}(\mathcal{L})$, the *Gaussian heuristic* predicts that

$$|\mathcal{L} \cap V| \approx \frac{\mathrm{vol}(V)}{\det(\mathcal{L})},$$

where $\mathrm{vol}(\cdot)$ denotes the Lebesgue measure on $\mathrm{span}(\mathcal{L}) \simeq \mathbb{R}^d$. Let $\mathcal{B}_d$ denote the open unit ball in $\mathrm{span}(\mathcal{L})$, and let[7]

$$\mathrm{GH}(\mathcal{L}) := \left(\frac{\det(\mathcal{L})}{\mathrm{vol}(\mathcal{B}_d)}\right)^{1/d} \approx \sqrt{\frac{d}{2\pi e}}\,\det(\mathcal{L})^{1/d}. \tag{1}$$

Then for any $\mathbf{t} \in \mathrm{span}(\mathcal{L})$, the ball $V := \mathbf{t} + \mathrm{GH}(\mathcal{L}) \cdot \mathcal{B}_d$ has volume $\det(\mathcal{L})$. Hence, under the Gausian heuristic, V contains only one lattice point from $\mathcal{L}$. This suggests, the following two heuristics.

Heuristic 2.1. *Let $\mathcal{L}$ be a random lattice. Then it holds that*

$$\lambda_1(\mathcal{L}) \approx \mathrm{GH}(\mathcal{L}). \tag{2}$$

Heuristic 2.2. *Let $\mathcal{L}$ be a random lattice, and let $\mathbf{t} = \mathbf{v} + \mathbf{e}$, where $\mathbf{v} \in \mathcal{L}$ and $\mathbf{e} \in \mathrm{span}(\mathcal{L})$, such that*

$$\|\mathbf{e}\| \leq \mathrm{GH}(\mathcal{L}). \tag{3}$$

Then $\mathbf{v}$ is the (unique) closest lattice vector to $\mathbf{t}$.

We like to point out that there also exist rigorous variants of Heuristics 2.1 and 2.2, that give slightly worse guarantees: When replacing Equation (2) in Heuristic 2.1 by $\frac{\lambda_1(\mathcal{L})}{2} \leq \mathrm{GH}(\mathcal{L})$, and Equation (3) in Heuristic 2.2 by $\|\mathbf{e}\| < \frac{\lambda_1(\mathcal{L})}{2}$, then both statements hold uncoditionally for *any* lattice $\mathcal{L}$. (The former is commonly known as *Minkowski's first theorem.*)

2.3 Lattice Problems

In this work, we mainly study three lattice problems: *BDD*, *LWE* and *Batch-CVP*. All of these can be seen as special variants of the *Closest Vector Problem (CVP)*. In CVP, one is given a basis of some lattice $\mathcal{L}$ along with some *target vector* $\mathbf{t} \in \mathrm{span}(\mathcal{L})$. The goal is to find a lattice vector $\mathbf{v} \in \mathcal{L}$ closest to $\mathbf{t}$.

In Batch-CVP, one is given *many* target vectors $\mathbf{t}^{(1)}, \mathbf{t}^{(2)}, \ldots \mathbf{t}^{(M)}$. The goal is to find for each target $\mathbf{t}^{(i)}$ one lattice vector $\mathbf{v}^{(i)} \in \mathcal{L}$ closest to $\mathbf{t}^{(i)}$. BDD is a promise-variant of CVP in which the solution $\mathbf{v}$ is guaranteed to be unique. LWE is a BDD variant in which the lattice $\mathcal{L}$ is a random q-ary lattice. More precisely, BDD and LWE are defined as follows.

[6] As usual in the lattice cryptanalysis literature, we do not formally define a probability distribution for random lattices. Instead, we heuristically assume that typical lattices encountered in cryptography behave sufficiently randomly.

[7] The approximation in Equation (1) is Stirling's formula.

Definition 2.3 (Bounded Distance Decoding (BDD)). *Let $\mathcal{L} \subset \mathbb{R}^m$ be a lattice with basis $\mathbf{B} \in \mathbb{R}^{m \times d}$. Let $\mathbf{t} = \mathbf{v} + \mathbf{e}$, where $\mathbf{v} \in \mathcal{L}$ and $\mathbf{e} \in \operatorname{span}(\mathcal{L})$ with $\|\mathbf{e}\| < \lambda_1(\mathcal{L})/2$. In the* Bounded Distance Decoding problem (BDD)*, the goal is to find $\mathbf{v}$, when given $\mathbf{B}$ and $\mathbf{t}$.*

For a BDD instance $\mathbf{t} = \mathbf{v} + \mathbf{e}$, we call $\mathbf{v}$ the *solution* and $\mathbf{e}$ the *error*. The constraint $\|\mathbf{e}\| < \lambda_1(\mathcal{L})/2$ on the error norm ensures uniqueness of the solution $\mathbf{v}$. In practice, $\mathcal{L}$ is typically a full-rank integer lattice, and $\mathbf{e}$ is drawn from some product distribution $\mathcal{D}^d$. As we will see later, the hardness of BDD mainly depends on the dimension d, and the *log-gap*, as defined below.

Definition 2.4 (Log-Gap). *Let $(\mathbf{B}, \mathbf{t})$ be a BDD instance with $\mathbf{B} \in \mathbb{R}^{m \times d}$ and $\mathbf{t} = \mathbf{v} + \mathbf{e}$, where $\mathbf{v}$ is the solution and $\mathbf{e}$ the error. The* log-gap *of $(\mathbf{B}, \mathbf{t})$ is defined as*

$$\gamma_{(\mathbf{B},\mathbf{t})} := \log_d \left(\frac{\sqrt{d} \det(\mathcal{L})^{1/d}}{\|\mathbf{e}\|} \right).$$

Geometrically, the log-gap $\gamma_{(\mathbf{B},\mathbf{t})}$ measures the distance between the BDD target $\mathbf{t}$ and the lattice $\mathcal{L} := \mathcal{L}(\mathbf{B})$ in relation to the density of $\mathcal{L}$. The closer $\mathbf{t}$ is to $\mathcal{L}$ (i.e., the smaller $\|\mathbf{e}\|$) and the sparser $\mathcal{L}$ (i.e., the bigger $\det(\mathcal{L})^{1/d}$), the bigger log-gap and the easier the BDD instance. Note, since $\|\mathbf{e}\| < \lambda_1(\mathcal{L}) \leq \sqrt{d} \det(\mathcal{L})^{1/d}$, we have $\gamma_{(\mathbf{B},\mathbf{t})} \geq \log_d(1) = 0$.

Definition 2.5 (Learning with Errors (LWE)). *Let $\mathbf{b} = \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2 \mod q$, where $q \in \mathbb{N}$, $\mathbf{A} \leftarrow \mathbb{Z}_q^{m \times n}$, $\mathbf{s}_1 \leftarrow \mathcal{D}^n$, and $\mathbf{s}_2 \leftarrow \mathcal{D}^m$, for some probability distribution $\mathcal{D}$ over $\mathbb{Z}$. In the* Learning with Errors problem (LWE)*, the goal is to find $\mathbf{s}_1$, when given $\mathbf{A}$, $\mathbf{b}$, q, and a description of $\mathcal{D}$.*

For an LWE instance $\mathbf{b} \equiv \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2 \mod q$, we call $\mathbf{s}_1$ the *LWE secret* and $\mathbf{s}_2$ the *LWE error*. In Regev's original definition of LWE [Reg05], only the error is drawn from $\mathcal{D}$, whereas the secret is drawn uniformly at random from $\mathbb{Z}_q^n$. However, most practical LWE-based schemes use the *short-secret variant* from Definition 2.5. Both variants are essentially equivalent (see [ACPS09]).

To see that LWE is indeed a special variant of BDD, note that from the public LWE parameters $\mathbf{A}$, $\mathbf{b}$ and q, we can efficiently compute

$$\mathbf{B} := \begin{pmatrix} q\mathbf{I}_m & \mathbf{A} \\ & \mathbf{I}_n \end{pmatrix}, \quad \mathbf{t} := \begin{pmatrix} \mathbf{b} \\ 0^n \end{pmatrix}. \tag{4}$$

Let $\mathbf{x} := 1/q\,(\mathbf{b} - \mathbf{A}\mathbf{s}_1 - \mathbf{s}_2) \in \mathbb{Z}^m$. Then for $\mathbf{v} := \mathbf{B}(\mathbf{x}, \mathbf{s}_1)^T = (\mathbf{b} - \mathbf{s}_2, \mathbf{s}_1)^T \in \mathcal{L}(\mathbf{B})$ and $\mathbf{e} := (\mathbf{s}_2, -\mathbf{s}_1)^T$ it holds that

$$\mathbf{t} = \mathbf{v} + \mathbf{e}. \tag{5}$$

Since $\mathbf{e} = (\mathbf{s}_2, -\mathbf{s}_1)^T$ typically has very small norm, this shows that $\mathbf{t}$ is close to the lattice vector $\mathbf{v}$. Hence, Equation (5) defines a BDD instance with solution $\mathbf{v}$ and error $\mathbf{e}$. If we solve this BDD instance, then we can easily read-off the LWE secret $\mathbf{s}_1$ from $\mathbf{v} = (\mathbf{b} - \mathbf{s}_2, \mathbf{s}_1)^T$ and thus solve the LWE instance $(\mathbf{A}, \mathbf{b}, q, \mathcal{D})$.

2.4 Lattice Algorithms

Sieving. Sieving is a method to find shortest vectors in a lattice [AKS01]. Its heuristic version [NV08, MV10] is the best known SVP solver in practice [ADH$^+$19]. The details of the algorithm are not relevant here, important is that it returns in time $(3/2)^{d/2+o(d)} \approx 2^{0.292d+o(d)}$ a list containing $(4/3)^{d/2+o(d)}$ short lattice vectors. This is crucial for the CVP solver we describe later, as this CVP solver relies on having a large list of short lattice vectors available.

BKZ. Let $\mathbf{B}$ be a lattice basis with columns $\mathbf{b}_0, \ldots, \mathbf{b}_{d-1}$. The *basis profile* of $\mathbf{B}$ is the tuple $(\|\mathbf{b}_0^*\|, \ldots, \|\mathbf{b}_{d-1}^*\|)$. In many applications, it is desirable to have a basis with *flat* basis profile. That is, a basis whose profile does not decrease *too fast*. To obtain such a basis, one typically resorts to the famous BKZ lattice reduction algorithm [Sch87]. To achieve a flat basis profile, BKZ computes shortest vectors in projected sublattices, and thereby obtains a basis with the following property:

Definition 2.6 (BKZ-reduced basis). *A lattice basis* $\mathbf{B} = (\mathbf{b}_0, \ldots, \mathbf{b}_{d-1})$ *is* BKZ-reduced with parameter β *(or* BKZ-β reduced*, for short), if*

$$\|\mathbf{b}_i^*\| = \lambda_1(\mathcal{L}_{[i,\min(i+\beta,d))}(\mathbf{B}))$$

for all $i = 0, 1, \ldots, d-1$.

The parameter β in BKZ, called the *blocksize*, controls the slope of the basis profile. The higher β, the flatter the profile. However, the larger β, the higher the runtime of BKZ. Indeed, the runtime of BKZ grows as $2^{0.292\beta+o(\beta)}$.

Besides computing bases with flat profiles, another important application of BKZ is computing short vectors: Suppose some lattice $\mathcal{L}$ contains an *unusually* short non-zero vector, i.e., a non-zero vector $\mathbf{v} \in \mathcal{L}$ with $\|\mathbf{v}\| \ll \mathrm{GH}(\mathcal{L})$. Then running BKZ (with sufficiently large β) on any basis of $\mathcal{L}$ will (heuristically) result in a basis containing the unusually short vector $\mathbf{v}$ (or $-\mathbf{v}$). Importantly, the larger $\frac{\mathrm{GH}(\mathcal{L})}{\|\mathbf{v}\|}$, the smaller β is necessary for BKZ to find $\mathbf{v}$.

Babai's Nearest Plane. Suppose we are given a BDD instance $(\mathbf{B}, \mathbf{t})$. If the basis profile of $\mathbf{B}$ is sufficiently flat, then Babai's famous *Nearest Plane* algorithm [Bab86] solves our BDD instance in polynomial time. In particular, we have the following well-known guarantee.

Lemma 2.7. *Let* $\mathbf{B} = (\mathbf{b}_0, \ldots, \mathbf{b}_{d-1}) \in \mathbb{Q}^{m \times d}$ *be a lattice basis, and let* $\mathbf{t} = \mathbf{v}+\mathbf{e}$ *with* $\mathbf{v} \in \mathcal{L}(\mathbf{B})$ *and* $\mathbf{e} = \sum_{i=0}^{d-1} \lambda_i \frac{1}{\|\mathbf{b}_i^*\|}\mathbf{b}_i^*$*, for some* $\lambda_0, \ldots, \lambda_{d-1} \in \mathbb{R}$*. If*

$$|\lambda_i| < \frac{1}{2}\|\mathbf{b}_i^*\|,$$

for every $i = 0, 1, \ldots, d-1$*, then on input* $(\mathbf{B}, \mathbf{t})$*, Babai's Nearest Plane algorithm outputs* $\mathbf{v}$ *in polynomial time.*

In the remainder of the paper, we write $\mathbf{w} \leftarrow \mathsf{BabaiNP}(\mathbf{B}, \mathbf{t})$ to denote that $\mathbf{w}$ is the output of Babai's Nearest Plane on input $(\mathbf{B}, \mathbf{t})$.

Randomized Slicer. The Iterative Slicer is a CVP solver proposed by Sommer, Feder and Shalvi [SFS09]. Given a CVP target $\mathbf{t}$ and a basis of some d-dimensional lattice $\mathcal{L}$, the algorithm starts by computing a long list $L \subset \mathcal{L}$ of short lattice vectors, for example, using a sieving algorithm. Next, the Iterative Slicer tries to reduce the norm of $\mathbf{t}$ by finding a $\mathbf{v} \in L$ that minimizes $\|\mathbf{t} - \mathbf{v}\|$. The algorithm iterates the process with the new shorter $\mathbf{t}' = \mathbf{t} - \mathbf{v}$ until there are no more vectors in L that can reduce the norm of the target. The difference $\mathbf{t} - \mathbf{t}' \in \mathcal{L}$ between the original target and the final $\mathbf{t}'$ then is a CVP solution. For the algorithm to provably succeed, we need L to be of size $2^{d+o(d)}$ [DB15], making it prohibitive in practice.

Heuristic solutions [Laa16, DLdW19, DLvW20] propose to reduce the size of the list L. This reduces the success probability exponentially. To rectify the success probability, [DLdW19] suggests to re-randomize the target vector exponentially many times, thereby leading to constant success probability under (standard) heuristic assumptions. More precisely, instead of running the iterative slicer on $\mathbf{t}$ only, we sample a vector from the coset $\mathbf{t} + \Lambda$ according to some discrete Gaussian distribution. The resulting algorithm is called the *Randomized Slicer*. Doulgerakis-Laarhoven-de Weger show in [DLdW19] that for $|L| = (4/3)^{d/2+o(d)}$, randomizing the target $(16/13)^{d/2+o(d)}$ times leads to a CVP algorithm with heuristic runtime

$$T_{|L|} + (18/13)^{d/2+o(d)} \approx T_{|L|} + 2^{0.234d+o(d)},$$

and constant success probability, where $T_{|L|}$ denotes the cost for computing L.

In [DLvW20], Ducas, Laarhoven and van Woerden show that the Randomized Slicer can be applied to many targets at once, thus solving Batch-CVP. This batched version of the Randomized Slicer algorithm for Batch-CVP is summarized in Algorithm 1.

Using near neighbor search, one can speed-up Algorithm 1's for-loop in the same way it is used in sieving algorithms [BDGL16]. We give more details on how this technique is implemented later, for now it is sufficient to state the resulting complexity of the Randomized Slicer that uses near neighbor search.

Heuristic Claim 2.8 (Randomized Slicer for Batch-CVP, [DLdW19, DLvW20]). *Given a basis of some d-dimensional lattice along with M Batch-CVP targets, Randomized Slicer (Algorithm 1) returns the M CVP solutions in time and space*

$$T_{\mathtt{Slicer}} = \max\{2^{0.292d+o(d)}, M \cdot 2^{0.234d+o(d)}\}.$$

3 BDD Algorithms

In this section, we discuss various algorithms for solving BDD. First, we recall the well established *primal attack* aka *Kannan's Embedding*. After that, we compare it to an approach sketched by Espitau and Kirchner in [EK20], which we call *Tail-BDD*. We give a new, fine-grained analysis for Tail-BDD and show that

Algorithm 1: Randomized Slicer [DLdW19, DLvW20]

```
Input: $\mathbf{B}$ – a basis of $\mathcal{L}(\mathbf{B})$, $\mathbf{t}^{(1)}, \ldots, \mathbf{t}^{(M)}$ – Batch-CVP targets
Output: $\mathbf{v}^{(1)}, \ldots, \mathbf{v}^{(M)} \in \mathcal{L}(\mathbf{B})$ – solutions for Batch-CVP
1  Procedure Randomize($\mathbf{t}, N$):
2      $L_\mathbf{t} \leftarrow \{\}$
3      while $|L_\mathbf{t}| < N$ do
4          $L_\mathbf{t} \leftarrow L_\mathbf{t} \cup \{\mathbf{t}' \leftarrow$ Discrete Gaussian over $\mathbf{t} + \mathcal{L}(\mathbf{B})\}$
5      end
6      Return $L_\mathbf{t}$
7  Compute a list of size $(4/3)^{d/2+o(d)}$ of shortest vectors from $\mathcal{L}(\mathbf{B})$ via sieving
8  $L' \leftarrow \{\}$
9  Compute $N$ as described in [DLvW20]
10 for $i = 1 \ldots M$ do
11     $L' \leftarrow L' \cup$ Randomize($\mathbf{t}^{(i)}, N$)
12 end
13 $L_{\text{out}} \leftarrow \{\}$
14 for $\mathbf{t} \in L'$ do
15     repeat
16         Find $\mathbf{v} \in L \cup \{\mathbf{0}\}$ minimizing $\|\mathbf{t} - \mathbf{v}\|$, set $\mathbf{t}' := \mathbf{t} - \mathbf{v}$
17     until $\|\mathbf{t}'\|$ no longer decreases;
18     $L_{\text{out}} \leftarrow L_{\text{out}} \cup \{\mathbf{t} - \mathbf{t}'\}$
19 end
20 Return $M$ vectors from $L_{\text{out}}$ corresponding to CVP solutions for
     $\mathbf{t}^{(1)}, \ldots, \mathbf{t}^{(M)}$.
```

the algorithm has the same performance as Kannan's Embedding. Building on top of that, we introduce our BDD hybrid attack, generalizing and improving Bernstein's hybrid attack on LWE [Ber23].

3.1 Kannan's Embedding

The best-known and most thoroughly studied approach for solving BDD is *Kannan's Embedding* [Kan83]. Given a BDD instance $(\mathbf{B}, \mathbf{t})$ with $\mathbf{B} \in \mathbb{R}^{m \times d}$ and $\mathbf{t} = \mathbf{v} + \mathbf{e}$ (where $\mathbf{v}$ is the solution and $\mathbf{e}$ the error), Kannan's Embedding constructs a lattice basis

$$\overline{\mathbf{B}} := \begin{pmatrix} \mathbf{B} & \mathbf{t} \\ & c \end{pmatrix} \in \mathbb{R}^{(m+1) \times (d+1)}, \tag{6}$$

where $c > 0$ is some scaling parameter. Optimal choices for c are discussed in [BSW16].

The lattice generated by $\overline{\mathbf{B}}$ contains the vector $\bar{\mathbf{e}} := (-\mathbf{v} + \mathbf{t}, c)^T = (\mathbf{e}, c)^T$. For typical parameters, $\bar{\mathbf{e}}$ is *unusually* short, i.e., it is likely (up to sign) the *unique* shortest non-zero vector in $\mathcal{L}(\overline{\mathbf{B}})$. To solve the BDD instance, one runs BKZ with sufficiently large blocksize β on $\overline{\mathbf{B}}$ to obtain $\pm\bar{\mathbf{e}}$. Given $\pm\bar{\mathbf{e}} = \pm(\mathbf{e}, c)^T$, one then easily obtains the BDD solution $\mathbf{v} = \mathbf{t} - \mathbf{e}$. A formal description of this approach is given in Algorithm 2.

```
Algorithm 2: Kannan's Embedding
   Input: BDD instance $(\mathbf{B}, \mathbf{t})$, BKZ blocksize $\beta$
   Output: Solution $\mathbf{v} \in \mathcal{L}(\mathbf{B})$ of BDD instance, or error symbol $\perp$
 1 Construct lattice basis
       $\overline{\mathbf{B}} := \begin{pmatrix} \mathbf{B} & \mathbf{t} \\ & c \end{pmatrix}.$
 2 Run BKZ-$\beta$ on $\overline{\mathbf{B}}$.
 3 Parse the shortest vector in the reduced basis as $(\mathbf{b}, u \cdot c)^T$, where $u \in \mathbb{Z}$.
 4 if $u = \pm 1$ then
 5     return $\mathbf{t} - u\mathbf{b}$
 6 else
 7     return $\perp$
```

Heuristically, Algorithm 2 succeeds under the following condition, which was first introduced in [ADPS16].

Heuristic Claim 3.1 (Kannan's Embedding Average Case). *Let the notation be as above, and let $\overline{\mathbf{b}}_0, \ldots, \overline{\mathbf{b}}_d$ denote the columns of $\overline{\mathbf{B}}$ after BKZ-reducing it with blocksize β. Algorithm 2 outputs the correct solution $\mathbf{v}$, provided that*

$$\left\| \pi^{\perp}_{d-\beta+1}(\overline{\mathbf{e}}) \right\| < \left\| \overline{\mathbf{b}}^{*}_{d-\beta+1} \right\| \tag{7}$$

for some sufficiently large β.

For $\beta \geq 50$, Heuristic Claim 3.1 is confirmed by extensive experiments, see, e.g. [AGVW17]. Under the *Geometric Series Assumption* [Sch03], one can estimate

$$\left\| \overline{\mathbf{b}}^{*}_{d-\beta+1} \right\| \approx \left(\frac{\beta}{2\pi e} (\pi\beta)^{\frac{1}{\beta}} \right)^{\frac{2\beta-d}{2(\beta-1)}} \left(|\det(\overline{\mathbf{B}})| \right)^{\frac{1}{d+1}} .$$

Making the additional (mild) assumption that $\overline{\mathbf{e}}$ is not too skewed in the direction of any of the vectors $\overline{\mathbf{b}}_0, \ldots, \overline{\mathbf{b}}_d$, one expects that $\left\| \pi^{\perp}_{d-\beta+1}(\overline{\mathbf{e}}) \right\| \approx \sqrt{\frac{\beta}{d+1}} \|\overline{\mathbf{e}}\|$. As a consequence, instead of using Equation (7) in Heuristic Claim 3.1, the literature often uses

$$\sqrt{\frac{\beta}{d+1}} \|\overline{\mathbf{e}}\| < \left(\frac{\beta}{2\pi e} (\pi\beta)^{\frac{1}{\beta}} \right)^{\frac{2\beta-d}{2(\beta-1)}} \left(|\det(\overline{\mathbf{B}})| \right)^{\frac{1}{d+1}} . \tag{8}$$

Moreover, asymptotically, Equation (8) can be rewritten as follows.

Heuristic Claim 3.2 (Kannan's Embedding Asymptotics). *Let $(\mathbf{B}, \mathbf{t})$ be a BDD instance on a d-dimensional lattice with bounded log-gap $\gamma_{(\mathbf{B},\mathbf{t})} = \mathcal{O}(1)$.*

If we run Algorithm 2 on input $(\mathbf{B}, \mathbf{t})$ *with blocksize* $\beta = \mathcal{B}d + o(d)$*, where*

$$\mathcal{B} = \frac{1}{1 + 2\gamma_{(\mathbf{B},\mathbf{t})}},$$

then the algorithm returns the correct solution $\mathbf{v}$*. The runtime of Algorithm 2 is* $2^{0.292\mathcal{B}d+o(d)}$.

Justification. Heuristically, the smallest β such that Algorithm 2 outputs $\mathbf{v}$ is the smallest β satisfying Equation (8), i.e.,

$$\sqrt{\frac{\beta}{d+1}}\|\overline{\mathbf{e}}\| < \left(\frac{\beta}{2\pi e}(\pi\beta)^{\frac{1}{\beta}}\right)^{\frac{2\beta-d}{2(\beta-1)}} \left(|\det(\overline{\mathbf{B}})|\right)^{\frac{1}{d+1}}.$$

Dividing $\|\overline{\mathbf{e}}\|$ from both sides, and applying $\log_d(\cdot)$ to the above, the inequality becomes

$$\log_d\left(\sqrt{\frac{\beta}{d+1}}\right) < \left(1 - \frac{d+2}{2(\beta-1)}\right)\cdot\log_d\left(\frac{\beta}{2\pi e}(\pi\beta)^{\frac{1}{\beta}}\right) + \log_d\left(\frac{\left|\det(\overline{\mathbf{B}})|\right)^{\frac{1}{d+1}}}{\|\overline{\mathbf{e}}\|}\right), \tag{9}$$

where we used the fact that $\frac{2\beta-d}{2(\beta-1)} = 1 - \frac{d+2}{2(\beta-1)}$. We show that our choice of $\beta = \mathcal{B}d + o(d)$ satisfies Equation (9).

We first note that, for our choice of β, we have

$$\begin{aligned}\left(1 - \frac{d+2}{2(\beta-1)}\right)\cdot\log_d\left(\frac{\beta}{2\pi e}(\pi\beta)^{\frac{1}{\beta}}\right) &= \left(1 - \frac{d+2}{2(\beta-1)}\right)\cdot\log_d\left(\Theta(d)\right)\\ &= 1 - \frac{d}{2\beta} + o(1),\end{aligned}$$

where the second equality follows from the facts that $\log_d(\Theta(d)) = 1 + o(1)$ and $\frac{d}{2\beta} = \Theta(1)$. Together with the approximations

$$\log_d(\|\overline{\mathbf{e}}\|) \approx \log_d(\|\mathbf{e}\|), \quad \text{and} \quad \log_d\left((|\det(\overline{\mathbf{B}})|)^{\frac{1}{d+1}}\right) \approx \log_d\left((|\det(\mathbf{B})|)^{\frac{1}{d}}\right),$$

and

$$\frac{d}{\beta} = \frac{1}{\mathcal{B} + o(1)} = \frac{1}{\mathcal{B}} + o(1) = 1 + 2\gamma_{(\mathbf{B},\mathbf{t})} + o(1),$$

it follows that, for our choice of β, the right hand side in Equation (9) becomes

$$\begin{aligned}1 - \frac{d}{2\beta} + \log_d\left(\frac{\det(\mathcal{L})^{1/d}}{\|\mathbf{e}\|}\right) + o(1) &= \frac{1}{2} - \frac{d}{2\beta} + \log_d\left(\frac{\sqrt{d}\det(\mathcal{L})^{1/d}}{\|\mathbf{e}\|}\right) + o(1)\\ &= \frac{1}{2} - \frac{d}{2\beta} + \gamma_{(\mathbf{B},\mathbf{t})} + o(1)\\ &= o(1).\end{aligned}$$

Since the left-hand side of Equation (9) also grows as $o(1)$, we can ensure that $\beta = \mathcal{B}d + o(d)$ satisfies Equation (9) by a suitable choice of the lower-order terms in $o(d)$. ◇

3.2 Tail-BDD

In [EK20], Espitau and Kirchner briefly mention the following alternative approach for solving BDD: "Once a highly reduced basis is found, it is enough to compute a CVP on the tail of the basis, and finish with Babai's algorithm." We give a more detailed description of this approach in Algorithm 3.

Algorithm 3: Tail-BDD

```
  Input: BDD instance $(\mathbf{B}, \mathbf{t})$, where $\mathbf{B} \in \mathbb{R}^{m \times d}$ is a BKZ-$\beta$ reduced basis with
         blocksize $\beta$
  Output: Solution $\mathbf{v} \in \mathcal{L}(\mathbf{B})$ of BDD instance
1 Compute a lattice vector $\widetilde{\mathbf{v}}_2 = \mathbf{B}_{[d-\beta,d)}\widetilde{\mathbf{x}}_2 \in \mathcal{L}(\mathbf{B}_{[d-\beta,d)})$, closest to $\pi^{\perp}_{d-\beta}(\mathbf{t})$.
2 Lift $\widetilde{\mathbf{v}}_2$ to $\mathcal{L}(\mathbf{B})$ by computing $\mathbf{v}_2 := \mathbf{B}(0^{d-\beta}, \widetilde{\mathbf{x}}_2)^T$.
3 $\mathbf{v}_1 \leftarrow \mathsf{BabaiNP}(\mathbf{B}_{[0,d-\beta)}, \pi_{d-\beta}(\mathbf{t} - \mathbf{v}_2))$
4 return $\mathbf{v}_1 + \mathbf{v}_2$
```

Since the original analysis by Espitau and Kirchner is somewhat implicit, we discuss the algorithm in detail below. We show that the algorithm has essentially the same performance as Kannan's Embedding. However, a major advantage over Kannan's Embedinng is that it allows to solve multiple instances at once.

Why Tail-BDD works. Suppose we run Algorithm 3 on input of our BDD instance $(\mathbf{B}, \mathbf{t})$ with $\mathbf{t} = \mathbf{v} + \mathbf{e}$. The main idea behind Algorithm 3 is to reduce our BDD instance over the d-dimensional lattice $\mathcal{L}(\mathbf{B})$ to a CVP instance over the β-dimensional lattice $\mathcal{L}(\mathbf{B}_{[d-\beta,d)})$. Specifically, Line 1 of Algorithm 3 solves a CVP instance on $\mathcal{L}(\mathbf{B}_{[d-\beta,d)})$, taking time exponential in β. After that, Lines 2 to 4 transform the resulting CVP solution into a solution of our initial BDD instance in polynomial time.

Let us write

$$\mathbf{v} = \mathbf{B}\mathbf{x}, \quad \text{for some } \mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2)^T \in \mathbb{Z}^{d-\beta} \times \mathbb{Z}^{\beta}. \tag{10}$$

Applying $\pi^{\perp}_{d-\beta}(\cdot)$ to $\mathbf{t}$, we obtain a CVP-like equation on the *tail* of our lattice, i.e., on the projected sublattice $\mathcal{L}(\mathbf{B}_{[d-\beta,d)})$:

$$\pi^{\perp}_{d-\beta}(\mathbf{t}) = \mathbf{B}_{[d-\beta,d)}\mathbf{x}_2 + \pi^{\perp}_{d-\beta}(\mathbf{e}). \tag{11}$$

Since $\mathbf{B} = (\mathbf{b}_0, \ldots, \mathbf{b}_{d-1})$ is BKZ-β reduced, we have $\lambda_1(\mathcal{L}(\mathbf{B}_{[d-\beta,d)})) = \|\mathbf{b}^*_{d-\beta}\|$, see Definition 2.6. Hence, if

$$\left\|\pi^{\perp}_{d-\beta}(\mathbf{e})\right\| < \frac{1}{2}\left\|\mathbf{b}^*_{d-\beta}\right\|, \tag{12}$$

then $\mathbf{B}_{[d-\beta,d)}\mathbf{x}_2$ is the *unique* closest lattice vector to $\pi^{\perp}_{d-\beta}(\mathbf{t})$.

If β is sufficiently large (and thereby the basis profile is sufficiently flat), then there is a decent chance that Equation (12) holds. Suppose this is the case, i.e., $\mathbf{B}_{[d-\beta,d)}\mathbf{x}_2$ is indeed the unique closest lattice vector to $\pi^{\perp}_{d-\beta}(\mathbf{t})$. Then Algorithm 3 recovers in Line 1 the vector $\widetilde{\mathbf{v}}_2 = \mathbf{B}_{[d-\beta,d)}\mathbf{x}_2$. Given $\widetilde{\mathbf{v}}_2$, Algorithm 3 computes in Line 2

$$\mathbf{v}_2 := \mathbf{B}(0^{d-\beta}, \widetilde{\mathbf{x}}_2)^T = \mathbf{B}(0^{d-\beta}, \mathbf{x}_2)^T.$$

From $\mathbf{t} = \mathbf{v} + \mathbf{e}$ and Equation (10) it then follows that

$$\mathbf{t} - \mathbf{v}_2 = \mathbf{B}(\mathbf{x}_1, \mathbf{x}_2)^T + \mathbf{e} - \mathbf{B}(0^{d-\beta}, \mathbf{x}_2)^T = \mathbf{B}_{[0,d-\beta)}\mathbf{x}_1 + \mathbf{e}.$$

The vector $\pi_{d-\beta}(\mathbf{t} - \mathbf{v}_2)$, computed in Line 3, is thus given by

$$\pi_{d-\beta}(\mathbf{t} - \mathbf{v}_2) = \mathbf{B}_{[0,d-\beta)}\mathbf{x}_1 + \pi_{d-\beta}(\mathbf{e}). \tag{13}$$

Equation (13) now defines a (hopefully) easy BDD-instance on $\mathcal{L}(\mathbf{B}_{[0,d-\beta)})$. Indeed, since $\mathbf{B}$ is BKZ-β reduced, the Gram Schmidt norms $\|\mathbf{b}^*_0\|, \ldots, \|\mathbf{b}^*_{d-\beta-1}\|$ do not decay too fast. Thus, Babai's algorithm should be able to solve the BDD-instance defined by Equation (13). More precisely, writing

$$\pi_{d-\beta}(\mathbf{e}) = \sum_{i=0}^{d-\beta-1} \frac{\lambda_i}{\|\mathbf{b}^*_i\|}\mathbf{b}^*_i, \tag{14}$$

with $\lambda_i \in \mathbb{R}$, Lemma 2.7 shows that Babai's algorithm successfully recovers $\mathbf{B}_{[0,d-\beta)}\mathbf{x}_1$ in Line 3, provided that

$$|\lambda_i| < \frac{1}{2}\|\mathbf{b}^*_i\|, \quad \text{for every } i = 0, 1, \ldots, d-\beta-1. \tag{15}$$

If this is the case, then Algorithm 3 returns

$$\mathbf{v}_1 + \mathbf{v}_2 = \mathbf{B}(\mathbf{x}_1, 0^{\beta}) + \mathbf{B}(0^{d-\beta}, \mathbf{x}_2) = \mathbf{B}\mathbf{x} = \mathbf{v},$$

and thereby solves the BDD instance $\mathbf{t} = \mathbf{v} + \mathbf{e}$.

Worst case analysis. Summarizing the above discussion, Algorithm 3 outputs the correct solution, provided that both Equations (12) and (15) hold. As we show below, a sufficient *worst case* condition for both equations to hold is

$$\|\mathbf{e}\| < \frac{1}{2}\left\|\mathbf{b}^*_{d-\beta}\right\|. \tag{16}$$

Indeed, if Equation (16) holds, then

$$\left\|\pi^{\perp}_{d-\beta}(\mathbf{e})\right\| \leq \|\mathbf{e}\| < \frac{1}{2}\left\|\mathbf{b}^*_{d-\beta}\right\|,$$

showing that Equation (12) holds. Making the mild assumption that the basis profile of $\mathbf{B}$ is non-increasing, i.e., $\|\mathbf{b}_0^*\| \geq \|\mathbf{b}_1^*\| \geq \ldots \geq \|\mathbf{b}_{d-1}^*\|$,[8] we obtain

$$|\lambda_i| \leq \sqrt{\sum_{i=0}^{d-\beta-1} \lambda_i^2} = \|\pi_{d-\beta}(\mathbf{e})\| \leq \|\mathbf{e}\| < \frac{1}{2}\left\|\mathbf{b}_{d-\beta}^*\right\| \leq \frac{1}{2}\left\|\mathbf{b}_i^*\right\| \tag{17}$$

for every $i = 0, 1, \ldots, d-\beta-1$. (Here the identity $\sqrt{\sum_{i=0}^{d-\beta-1} \lambda_i^2} = \|\pi_{d-\beta}(\mathbf{e})\|$ follows from Equation (14) and the fact that $\{\frac{1}{\|\mathbf{b}_i^*\|}\mathbf{b}_i^*\}_{i=0}^{d-\beta-1}$ is an orthonormal basis.) Since this shows that Equation (15) holds as well, we obtain the following theorem.

Theorem 3.3 (Tail-BDD Worst Case). *Let* $(\mathbf{B}, \mathbf{t})$ *be a BDD instance with* $\mathbf{t} = \mathbf{v} + \mathbf{e}$*, where* $\mathbf{v}$ *is the solution and* $\mathbf{e}$ *the error. If* $\mathbf{B} = (\mathbf{b}_0, \ldots, \mathbf{b}_{d-1})$ *is BKZ-*β *reduced, has non-increasing basis profile, and satisfies*

$$\|\mathbf{e}\| < \frac{1}{2}\left\|\mathbf{b}_{d-\beta}^*\right\|,$$

then on input $(\mathbf{B}, \mathbf{t})$*, Algorithm 3 outputs the correct solution* $\mathbf{v}$.

Using the Geometric Series Assumption to estimate $\|\mathbf{b}_{d-\beta}^*\|$, one may now use Theorem 3.3 to compute a β, for which Algorithm 3 succeeds. However, the β's obtained from Theorem 3.3 would be somewhat larger than the actual β's required in practice, since we used rather coarse upper bounds in the discussion above. Below, we give an heuristic average case analysis of Algorithm 3 that estimates the required β more accurately.

Average case analysis of Tail-BDD. As discussed above, Algorithm 3 outputs the correct solution, provided that both Equations (12) and (15) hold. Recall that we used Equation (12) to ensure that $\mathbf{B}_{[d-\beta,d)}\mathbf{x}_2$ is the unique closest lattice vector in $\mathcal{L}(\mathbf{B}_{[d-\beta,d)})$ to $\pi_{d-\beta}^{\perp}(\mathbf{t})$. Under the Gaussian heuristic (see Heuristics 2.1 and 2.2), we may replace Equation (12) by the slightly weaker condition

$$\left\|\pi_{d-\beta}^{\perp}(\mathbf{e})\right\| < \|\mathbf{b}_{d-\beta}^*\|, \tag{18}$$

i.e., we may get rid of the factor $\frac{1}{2}$ in Equation (12), and can still expect $\mathbf{B}_{[d-\beta,d)}\mathbf{x}_2$ to be the unique closest lattice vector in $\mathcal{L}(\mathbf{B}_{[d-\beta,d)})$ to $\pi_{d-\beta}^{\perp}(\mathbf{t}) = \mathbf{B}_{[d-\beta,d)}\mathbf{x}_2 + \pi_{d-\beta}^{\perp}(\mathbf{e})$. Hence, we expect Algorithm 3 to output the correct solution, whenever both Equations (15) and (18) hold.

As we show below, heuristically, we can expect Equation (18) to imply Equation (15). As a result, we obtain the following estimate.

Heuristic Claim 3.4 (Tail-BDD Average Case). *Let* $(\mathbf{B}, \mathbf{t})$ *be a BDD instance with* $\mathbf{t} = \mathbf{v} + \mathbf{e}$*, where* $\mathbf{v}$ *is the solution and* $\mathbf{e}$ *the error. If* $\mathbf{B} =$

[8] In practice, this is virtually always the case.

$(\mathbf{b}_0, \ldots, \mathbf{b}_{d-1})$ *is BKZ-β reduced and satisfies*

$$\left\|\pi^{\perp}_{d-\beta}(\mathbf{e})\right\| < \left\|\mathbf{b}^*_{d-\beta}\right\|,$$

for some sufficiently large β, then on input $(\mathbf{B}, \mathbf{t})$, Algorithm 3 outputs the correct solution $\mathbf{v}$.

Justification. We have to argue that the condition $\|\pi^{\perp}_{d-\beta}(\mathbf{e})\| < \|\mathbf{b}^*_{d-\beta}\|$ (i.e., Equation (18)) implies Equation (15). Heuristically, we may assume that $\mathbf{e}$ is not too skewed in any direction of the orthonormal basis $\{\frac{1}{\|\mathbf{b}^*_i\|}\mathbf{b}^*_i\}_{i=0}^{d-1}$. In other words, we may assume that all d coordinates λ_i in Equation (14) are of similar size. As a consequence, (unless β is extremely small) we may assume that for every i it holds that

$$\|\pi^{\perp}_{d-\beta}(\mathbf{e})\| = \sqrt{\sum_{i=d-\beta}^{d-1} \lambda_i^2} \approx \sqrt{\beta} \cdot |\lambda_i| \gg |\lambda_i|.$$

In particular, we may safely assume $|\lambda_i| < \frac{1}{2}\|\pi_{d-\beta}(\mathbf{e})\|$.

Now making again the mild assumption that the basis profile of $\mathbf{B}$ is non-increasing, we obtain

$$|\lambda_i| < \frac{1}{2}\|\pi^{\perp}_{d-\beta}(\mathbf{e})\| < \frac{1}{2}\|\mathbf{b}^*_{d-\beta+1}\| \leq \frac{1}{2}\|\mathbf{b}^*_i\|$$

for every $i = 0, 1, \ldots, d-\beta-1$, which shows that Equation (15) holds. $\diamond$

Remark 3.5. In the justification of Heuristic Claim 3.4, we essentially showed that when Equation (18) holds, and β is not extremely small, then Babai's Nearest Plane successfully lifts the projected vector $\pi_{d-\beta}(\mathbf{e})$ to $\mathbf{e}$. For Kannan's Embedding, [AGVW17] already showed the same thing.

Comparison with Kannan's Embedding. Notice that Heuristic Claim 3.4 requires essentially the same condition for Tail-BDD as Heuristic Claim 3.1 requires for Kannan's Embedding. In fact, the only difference is an index-shift by 1, which comes from the fact that Kannan's Embedding works with a $(d+1)$-dimensional lattice, whereas Tail-BDD works with a d-dimensional lattice. In practice, this does not make any substantial difference. Hence, Algorithm 3 has essentially the same performance as Kannan's Embedding. In particular, we get the exact same asymptotic behavior as for Kannan's Embedding.

Heuristic Claim 3.6 (Tail-BDD Asymptotics.). *Let $(\mathbf{B}, \mathbf{t})$ be a BDD instance on a d-dimensional lattice with bounded log-gap $\gamma_{(\mathbf{B},\mathbf{t})} = \mathcal{O}(1)$. If we run Algorithm 3 on input $(\mathbf{B}, \mathbf{t})$ with blocksize $\beta = \mathcal{B}d + o(d)$, where*

$$\mathcal{B} = \frac{1}{1 + 2\gamma_{(\mathbf{B},\mathbf{t})}},$$

then the algorithm returns the correct solution $\mathbf{v}$. The runtime of Algorithm 3 is $2^{0.292\mathcal{B}d+o(d)}$.

Batched Tail-BDD. Suppose we are given a basis $\mathbf{B}$ of some d dimensional lattice along with *many* targets $\mathbf{t}^{(1)}, \ldots, \mathbf{t}^{(M)}$ defining M BDD instances over $\mathcal{L}(\mathbf{B})$. If we want to solve all M BDD instances using Kannan's Embedding, then we have to construct M different lattice bases $\overline{\mathbf{B}}$, as in Equation (6), and BKZ-reduce each of these. Using the Tail-BDD algorithm, on the other hand, we have to BKZ-reduce $\mathbf{B}$ only once, and then solve M CVP instances on $\mathcal{L}(\mathbf{B}_{[d-\beta,d)})$, where β denotes some sufficiently large blocksize. In other words, Tail-BDD allows reducing M BDD instances on a d-dimensional lattice to *one* Batch-CVP instance on a β-dimensional lattice. Using the Randomized Slicer for solving this Batch-CVP instance allows to gain a significant speedup over Kannan's Embedding: For Kannan's Embedding, we (heuristically) obtain asymptotic runtime $M \cdot 2^{0.292\beta+o(\beta)}$, whereas the approach based on the Randomized Slicer (heuristically) runs in time

$$\max\{2^{0.292\beta+o(\beta)}, M \cdot 2^{0.234\beta+o(\beta)}\}.$$

In particular, solving $M \leq 2^{0.058\beta}$ instances has the same cost as solving a *single* instance with Kannan's Embedding.

Algorithm 4: Batched-Tail-BDD

```
  Input: BKZ-$\beta$ reduced basis $\mathbf{B} \in \mathbb{R}^{m \times d}$ and BDD targets $\mathbf{t}^{(1)}, \ldots, \mathbf{t}^{(M)}$,
         blocksize $\beta$
  Output: Solutions $\mathbf{v}^{(i)} \in \mathcal{L}(\mathbf{B})$ to the BDD instances defined by the $\mathbf{t}^{(i)}$'s
1 Use Randomized Slicer to compute a list of lattice vectors $(\widetilde{\mathbf{v}}_2^{(1)}, \ldots, \widetilde{\mathbf{v}}_2^{(M)})$ in
    $\mathcal{L}(\mathbf{B}_{[d-\beta,d)})$, where $\widetilde{\mathbf{v}}_2^{(i)} = \mathbf{B}_{[d-\beta,d)}\widetilde{\mathbf{x}}_2^{(i)}$ is closest to $\pi_{d-\beta}^{\perp}(\mathbf{t}^{(i)})$.
2 for $i = 1, \ldots, M$ do
3     Lift $\widetilde{\mathbf{v}}_2^{(i)}$ to $\mathcal{L}(\mathbf{B})$ by computing $\mathbf{v}_2^{(i)} := \mathbf{B}(0^{n-\beta}, \widetilde{\mathbf{x}}_2^{(i)})$.
4     $\mathbf{v}_1^{(i)} \leftarrow \mathsf{BabaiNP}(\mathbf{B}_{[0,d-\beta)}, \pi_{d-\beta}(\mathbf{t}^{(i)} - \mathbf{v}_2^{(i)}))$
5     $\mathbf{v}^{(i)} := \mathbf{v}_1^{(i)} + \mathbf{v}_2^{(i)}$ .
6 end
7 return $(\mathbf{v}^{(1)}, \ldots, \mathbf{v}^{(M)})$
```

The whole process is formally described in Algorithm 4. Analogous to Heuristic Claims 3.4 and 3.6, we have the following estimates.

Heuristic Claim 3.7 (Batched-Tail-BDD Average Case). *For $i = 1, \ldots, M$, let $(\mathbf{B}, \mathbf{t}^{(i)})$ be BDD instances with $\mathbf{t}^{(i)} = \mathbf{v}^{(i)} + \mathbf{e}^{(i)}$, where $\mathbf{v}^{(i)}$ is the solution and $\mathbf{e}^{(i)}$ the error. Suppose we run Algorithm 4 with blocksize β on input $(\mathbf{B}, \mathbf{t}^{(1)}, \ldots, \mathbf{t}^{(M)})$. If $\mathbf{B} = (\mathbf{b}_0, \ldots, \mathbf{b}_{d-1})$ is BKZ-β reduced and satisfies*

$$\left\|\pi_{d-\beta}^{\perp}(\mathbf{e})^{(i)}\right\| < \left\|\mathbf{b}_{d-\beta}^{*}\right\|,$$

for some sufficiently large β and some $i \in [M]$, then the i-th vector returned by Algorithm 4 is the correct solution of the i-th BDD instance $(\mathbf{B}, \mathbf{t}^{(i)})$.

Heuristic Claim 3.8 (Batched-Tail-BDD Asymptotitcs). *For $i = 1, \dots, M$, let $(\mathbf{B}, \mathbf{t}^{(i)})$ be BDD instances on a d-dimensional lattice with bounded log-gaps $\gamma_{(\mathbf{B},\mathbf{t}^{(i)})} = \mathcal{O}(1)$. If we run Algorithm 4 on input $(\mathbf{B}, \mathbf{t}^{(1)}, \dots, \mathbf{t}^{(M)})$ with blocksize $\beta = \mathcal{B}d + o(d)$, where*

$$\mathcal{B} = \frac{1}{1 + 2\gamma_{(\mathbf{B},\mathbf{t}^{(i)})}},$$

then the algorithm returns the correct solution $\mathbf{v}^{(i)}$ of $(\mathbf{B}, \mathbf{t}^{(i)})$. The runtime of Algorithm 4 is $\max\{2^{0.292\beta+o(\beta)}, M \cdot 2^{0.234\beta+o(\beta)}\}$.

3.3 Bernstein's Hybrid Attack Generalized

Bernstein [Ber23] uses Batched-Tail-BDD (Algorithm 4) as the central building block in his hybrid attack on LWE. We show now that his attack is not specific to LWE, but can be generalized to BDD. Additionally, we improve the runtime of his attack by using an improved routine for enumerating secret keys from [GMN23].

Splitting the BDD instance. The main idea behind the hybrid attack is to split our BDD instance $\mathbf{t} = \mathbf{v}+\mathbf{e}$ into two subproblems on the lattices $\mathcal{L}(\mathbf{B}_{[0,d-\kappa)})$ and $\mathcal{L}(\mathbf{B}_{[d-\kappa,d)})$, where $\kappa \in [d]$ is some parameter to be optimized later. In a nutshell, the attack consists of an enumeration step on the κ-dimensional lattice $\mathcal{L}(\mathbf{B}_{[d-\kappa,d)})$, and a lattice reduction step on the $(d-\kappa)$-dimensional lattice $\mathcal{L}(\mathbf{B}_{[0,d-\kappa)})$. The hope is that both steps together outperform lattice reduction on the d-dimensional lattice $\mathcal{L}(\mathbf{B})$ in Kannan's Embedding and (non-batched) Tail-BDD.

Let us write $\mathbf{v} = \mathbf{B}(\mathbf{u}_1, \mathbf{u}_2)^T$, where $(\mathbf{u}_1, \mathbf{u}_2)^T \in \mathbb{Z}^{d-\kappa} \times \mathbb{Z}^{\kappa}$. Applying $\pi_{d-\kappa}(\cdot)$ and $\pi^{\perp}_{d-\kappa}(\cdot)$ to $\mathbf{t} = \mathbf{v} + \mathbf{e}$, we obtain

$$\pi^{\perp}_{d-\kappa}(\mathbf{t}) = \mathbf{B}_{[d-\kappa,d)}\mathbf{u}_2 + \pi^{\perp}_{d-\kappa}(\mathbf{e}), \tag{19}$$

and

$$\pi_{d-\kappa}(\mathbf{t} - \mathbf{B}(0^{d-\kappa}, \mathbf{u}_2)^T) = \mathbf{B}_{[0,d-\kappa)}\mathbf{u}_1 + \pi_{d-\kappa}(\mathbf{e}). \tag{20}$$

Using Equations (19) and (20), the hybrid attack recovers $\mathbf{u}_1$ and $\mathbf{u}_2$ in a two-step approach, thereby solving our BDD instance $\mathbf{t} = \mathbf{v} + \mathbf{e}$.

At first glance, this may seem very similar to the (non-batched) Tail-BDD algorithm. Indeed, when replacing κ with β, then Equations (19) and (20) become *identical* to Equations (11) and (13) from the analysis of Tail-BDD. However, there is a crucial difference between the choice of β in Tail-BDD and the choice of κ in the hybrid attack: In Tail-BDD the choice of β ensures that the vector $\mathbf{B}_{[d-\beta,d)}\mathbf{x}_2$ in Equation (11) is *the closest* lattice vector to the projected target $\pi^{\perp}_{d-\beta}(\mathbf{t})$. In contrast, for our choice of κ, the vector $\mathbf{B}_{[d-\kappa,d)}\mathbf{u}_2$ in Equation (19) will generally only be *close* to $\pi^{\perp}_{d-\kappa}(\mathbf{t})$. As a result, we require different techniques for recovering $\mathbf{u}_2$ from $\pi^{\perp}_{d-\kappa}(\mathbf{t})$: Instead of simply applying a CVP-solver to $\pi^{\perp}_{d-\kappa}(\mathbf{t})$, we have to enumerate a list of candidate vectors $(\widetilde{\mathbf{w}}_2^{(1)}, \dots, \widetilde{\mathbf{w}}_2^{(M)}) \subset \mathcal{L}(\mathbf{B}_{[d-\kappa,d)})$ for $\mathbf{B}_{[d-\kappa,d)}\mathbf{u}_2$, that are close to $\pi^{\perp}_{d-\kappa}(\mathbf{t})$.

Efficient enumeration on $\mathcal{L}(\mathbf{B}_{[d-\kappa,d)})$. Before we can introduce the hybrid attack, we have to discuss how we enumerate our list of lattice vectors $(\widetilde{\mathbf{w}}_2^{(1)}, \dots, \widetilde{\mathbf{w}}_2^{(M)})$. Let us consider the typical setting, where $\mathcal{L}(\mathbf{B})$ is a full-rank d-dimensional integer lattice, and $\mathbf{e}$ is sampled from some (known) product distribution $\mathcal{D}^d$. Without loss of generality, we may assume that $\mathbf{B} \in \mathbb{Z}^{d\times d}$ is the HNF basis of $\mathcal{L}(\mathbf{B})$. Then, due to the upper-triangular shape of $\mathbf{B}$, we have $\operatorname{span}(\mathbf{B}_{[0,d-\kappa)}) = \mathbb{R}^{d-\kappa} \times \{0\}^\kappa$. In particular, for every $\mathbf{v} = (v_0, \dots, v_{d-1}) \in \mathbb{R}^d$, we have $\pi^\perp_{d-\kappa}(\mathbf{v}) = (0^{d-\kappa}, v_\kappa, \dots, v_{d-1})$. It follows that the projected BDD error $\pi^\perp_{d-\kappa}(\mathbf{e})$ is distributed as $\pi^\perp_{d-\kappa}(\mathbf{e}) \leftarrow \{0\}^{d-\kappa} \times \mathcal{D}^\kappa$. As a result, we can easily enumerate candidates $\widetilde{\mathbf{w}}_2^{(i)} \in \mathcal{L}(\mathbf{B}_{[d-\kappa,d)})$ for $\mathbf{B}_{[d-\kappa,d)}\mathbf{u}_2 = \pi^\perp_{d-\kappa}(\mathbf{t}) - \pi^\perp_{d-\kappa}(\mathbf{e})$ by simply iterating over all vectors $\mathbf{x}^{(i)} \in \operatorname{supp}(\mathcal{D})^\kappa$, and computing[9] $\widetilde{\mathbf{w}}_2^{(i)} := \pi^\perp_{d-\kappa}(\mathbf{t}) - (0^{d-\kappa}, \mathbf{x}^{(i)})$. Clearly, enumeration then runs in time $|\operatorname{supp}(\mathcal{D})|^\kappa$.

Importantly, if $\mathcal{D}$ is not the uniform distribution, then enumeration can be implemented significantly more efficiently: Suppose instead of enumerating *all* vectors $\mathbf{x}^{(i)}$ in the support of $\mathcal{D}^\kappa$, we randomly sample $2^{\mathrm{H}(\mathcal{D})\kappa+1}$ vectors from $\mathcal{D}^\kappa$, where $\mathrm{H}(\cdot)$ denotes the Shannon entropy of $\mathcal{D}$. As shown in [GMN23], the desired vector $\pi^\perp_{d-\kappa}(\mathbf{e})$ is then, with constant probability, among all sampled candidates $(0^{d-\kappa}, \mathbf{x}^{(i)})$.

If $\mathcal{D}$ is the uniform distribution, then $|\operatorname{supp}(\mathcal{D})| = 2^{\mathrm{H}(\mathcal{D})}$. Hence, for the uniform distribution, both approaches of enumerating all vectors from $\operatorname{supp}(\mathcal{D})^\kappa$ or sampling $2^{\mathrm{H}(\mathcal{D})\kappa+1}$ vectors are essentially equivalent. However, for any $\mathcal{D}$ different from the uniform distribution, we have $|\operatorname{supp}(\mathcal{D})| > 2^{\mathrm{H}(\mathcal{D})}$. Thus, sampling $2^{\mathrm{H}(\mathcal{D})\kappa+1}$ vectors improves over enumeration of $|\operatorname{supp}(\mathcal{D})|^\kappa$ by a factor exponential in κ, while the success probability decreases only by a small constant.

Our enumeration routine is described in Algorithm 5.

Enumeration + Batch-Tail-BDD. Having explained our enumeration routine, we can now give the hybrid attack in Algorithm 6.

Let $(\widetilde{\mathbf{w}}_2^{(1)}, \dots, \widetilde{\mathbf{w}}_2^{(M)})$ be the list produced by Algorithm 5's for-loop. As discussed above, this list likely contains $\mathbf{B}_{[d-\kappa,d)}\mathbf{u}_2$. To identify $\mathbf{B}_{[d-\kappa,d)}\mathbf{u}_2$ among the $\widetilde{\mathbf{w}}_2^{(i)}$'s, and to recover the remaining coordinates $\mathbf{u}_1$ of our BDD solution $\mathbf{v} = \mathbf{B}(\mathbf{u}_1, \mathbf{u}_2)^T$, Algorithm 6 uses Batched-Tail-BDD as follows.

For every $\widetilde{\mathbf{w}}_2^{(i)} = \mathbf{B}_{[d-\beta,d)}\mathbf{u}_2^{(i)} \in \mathcal{L}(\mathbf{B}_{[d-\beta,d)})$, Algorithm 5 lifts $\widetilde{\mathbf{w}}_2^{(i)}$ to $\mathcal{L}(\mathbf{B})$ as $\mathbf{w}_2^{(i)} := \mathbf{B}(0^{d-\kappa}, \mathbf{u}_2^{(i)})^T$, and then computes $\mathbf{t}^{(i)} := \pi_{d-\kappa}(\mathbf{t} - \mathbf{w}_2^{(i)})$. By construction, among all i's, there likely is one i^* with

$$\mathbf{w}_2^{(i^*)} = \mathbf{B}(0^{d-\kappa}, \mathbf{u}_2)^T. \tag{21}$$

Hence, by Equation (20), the corresponding $\mathbf{t}^{(i^*)} = \pi_{d-\kappa}(\mathbf{t} - \mathbf{w}_2^{(i^*)})$ is

$$\mathbf{t}^{(i^*)} = \mathbf{B}_{[0,d-\kappa]}\mathbf{u}_1 + \pi_{d-\kappa}(\mathbf{e}). \tag{22}$$

[9] If for some $\mathbf{x}^{(i)}$, the resulting vector $\widetilde{\mathbf{w}}_2^{(i)} = \pi^\perp_{d-\kappa}(\mathbf{t}) - (0^{d-\kappa}, \mathbf{x}^{(i)})$ does not belong to $\mathcal{L}(\mathbf{B}_{[d-\kappa,d)})$, then $\mathbf{x}^{(i)}$ can immediately be discarded.

```
Algorithm 5: Enumeration
  Input: BDD instance $(\mathbf{B},\mathbf{t})$,where $\mathbf{B}\in\mathbb{R}^{d\times d}$ is HNF basis, and the error is
         sampled from some distribution $\mathcal{D}^d$,
         parameters $\beta,\kappa$
  Output: List of targets $(\mathbf{t}^{(1)},\ldots,\mathbf{t}^{(M)})$, where $M=2^{\mathrm{H}(\mathcal{D})\kappa+1}$
1 $M := 2^{\mathrm{H}(\mathcal{D})\kappa+1}$
2 for $i=1,2,\ldots,M$ do
3     $\mathbf{x}^{(i)} \leftarrow \mathcal{D}^\kappa$
4     $\widetilde{\mathbf{w}}_2^{(i)} := \pi_{d-\kappa}^{\perp}(\mathbf{t}) - (0^{d-\kappa},\mathbf{x}^{(i)})$
5     if $\widetilde{\mathbf{w}}_2^{(i)} \in \mathcal{L}(\mathbf{B}_{[d-\beta,d)})$ then
6         Compute $\mathbf{u}_2^{(i)}\in\mathbb{Z}^\kappa$ with $\widetilde{\mathbf{w}}_2^{(i)} = \mathbf{B}_{[d-\beta,d)}\mathbf{u}_2^{(i)}$.
7         $\mathbf{w}_2^{(i)} := \mathbf{B}(0^{d-\kappa},\mathbf{u}_2^{(i)})^T$
8         $\mathbf{t}^{(i)} = \pi_{d-\kappa}(\mathbf{t}-\mathbf{w}_2^{(i)})^T$
9     end
10 end
11 return $(\mathbf{t}^{(1)},\ldots,\mathbf{t}^{(M)})$
```

```
Algorithm 6: BDD Hybrid Attack
  Input: BDD instance $(\mathbf{B},\mathbf{t})$,where $\mathbf{B}\in\mathbb{R}^{d\times d}$ is HNF basis, and the error is
         sampled from some distribution $\mathcal{D}^d$,
         parameters $\beta,\kappa$
  Output: Solution $\mathbf{v}\in\mathcal{L}(\mathbf{B})$ of BDD instance
1 Enumerate list of targets $(\mathbf{t}^{(1)},\ldots,\mathbf{t}^{(M)})$ using Algorithm 5.
2 $\widetilde{\mathbf{B}} \leftarrow \mathsf{BKZ}_\beta\left(\mathbf{B}_{[0,d-\kappa)}\right)$
3 Run Batched-Tail-BDD with blocksize $\beta$ on input $\widetilde{\mathbf{B}}$ and $(\mathbf{t}^{(1)},\ldots,\mathbf{t}^{(M)})$ to
    compute a list of lattice vectors $(\mathbf{w}_1^{(1)},\ldots,\mathbf{w}_1^{(M)})$ in $\mathcal{L}(\mathbf{B}_{[0,d-\kappa)})$
4 return $\mathbf{w}_1^{(i)}+\mathbf{w}_2^{(i)}$, for $i\in[M]$ minimizing $\|\mathbf{t}-\mathbf{w}_1^{(i)}-\mathbf{w}_2^{(i)}\|$.
```

Let us call $\mathbf{t}^{(i^*)}$ the *golden target*. Since $\mathcal{L}(\mathbf{B}_{[0,d-\kappa)})$ is a sublattice of $\mathcal{L}(\mathbf{B})$, we have $\lambda_1(\mathcal{L}(\mathbf{B})) \le \lambda_1(\mathcal{L}(\mathbf{B}_{[0,d-\kappa)}))$, and thus

$$\|\pi_{d-\kappa}(\mathbf{e})\| \le \|\mathbf{e}\| < \frac{1}{2}\lambda_1(\mathcal{L}(\mathbf{B})) \le \frac{1}{2}\lambda_1(\mathcal{L}(\mathbf{B}_{[0,d-\kappa)})).$$

Together with Equation (22) this shows that $\mathbf{B}_{[0,d-\kappa]}\mathbf{u}_1$ is *the* closest lattice vector in $\mathcal{L}(\mathbf{B}_{[0,d-\kappa)})$ to $\mathbf{t}^{(i^*)}$. In particular, our golden target $\mathbf{t}^{(i^*)}$ defines a BDD instance on $\mathcal{L}(\mathbf{B}_{[0,d-\kappa)})$ with solution $\mathbf{B}_{[0,d-\kappa)}\mathbf{u}_1$.

Hence, if we instantiate Algorithm 6 with sufficiently large blocksize β, then the i^*-th vector $\mathbf{w}_1^{(i^*)}$ returned by Batch-Tail-BDD in Line 3 is $\mathbf{w}_1^{(i^*)} = \mathbf{B}_{[0,d-\kappa)}\mathbf{u}_1$. Since by Equation (21) we have

$$\mathbf{w}_1^{(i^*)} + \mathbf{w}_2^{(i^*)} = \mathbf{B}(\mathbf{u}_1, 0^\kappa)^T + \mathbf{B}(0^{d-\kappa}, \mathbf{u}_2^{(i)})^T = \mathbf{B}(\mathbf{u}_1,\mathbf{u}_2)^T = \mathbf{v},$$

it follows that Algorithm 6 successfully solves our BDD instance $\mathbf{t} = \mathbf{v} + \mathbf{e}$ by simply computing the $i \in [M]$ that minimizes $\|\mathbf{t} - \mathbf{w}_1^{(i)} - \mathbf{w}_2^{(i)}\|$, and returning $\mathbf{w}_1^{(i)} + \mathbf{w}_2^{(i)}$.

Asymptotic analysis. The asymptotic runtime T of Algorithm 6 is the sum of the runtimes of BKZ and Batched-Tail-BDD with $M = 2^{\mathrm{H}(\mathcal{D})\kappa+1}$ targets. Hence, by Heuristic Claim 3.8, we have

$$T = \max\{2^{0.292\beta+o(\beta)}, M \cdot 2^{0.234\beta+o(\beta)}\}. \tag{23}$$

For Algorithm 6 to succeed, we have to chose κ and β for which Batched-Tail-BDD successfully solves the BDD instance defined by the golden target $\mathbf{t}^{(i^*)}$ in Equation (22). By Heuristic Claim 3.8, this requires us to take

$$\beta \geq \mathcal{B}_\kappa(d-\kappa) + o(d-\kappa), \quad \text{where } \mathcal{B}_\kappa = \frac{1}{2(1+\gamma_{(\mathbf{B}_{[0,n-k)},\mathbf{t}^{(i^*)})})}.$$

In any typical lattice $\mathcal{L}(\mathbf{B})$, we may safely assume that, for any κ, the sublattice $\mathcal{L}(\mathbf{B}_{[0,d-\kappa]})$ is *not* denser than $\mathcal{L}(\mathbf{B})$. Hence, we expect the log-gap $\gamma_{(\mathbf{B}_{[0,d-k)},\mathbf{t}^{(i^*)})}$ to be non-decreasing in κ. Conversely, we can safely assume $\mathcal{B}_\kappa$ to be non-increasing in κ. In particular, $\mathcal{B}_0 \geq \mathcal{B}_\kappa$ for all κ. Thus, taking

$$\beta = \mathcal{B}_0(d-\kappa) + o(d-\kappa) = \frac{d-\kappa}{2(1+\gamma_{(\mathbf{B},\mathbf{t})})} + o(d-\kappa) \tag{24}$$

suffices for Algorithm 6 to succeed. By Equation (23), the runtime T is minimized for

$$\mathrm{H}(\mathcal{D})\kappa + 0.234\beta = 0.292\beta. \tag{25}$$

Plugging $\beta = \mathcal{B}_0(d-\kappa)$ from Equation (24) into Equation (25) and solving for κ shows that the optimal κ is

$$\kappa = \frac{\mathcal{B}_0 d}{\mathcal{B}_0 + \frac{\mathrm{H}(\mathcal{D})}{0.058}}.$$

Together with Equation (24), this results in the following asymptotic runtime for the hybrid attack.

Heuristic Claim 3.9. *Let $(\mathbf{B}, \mathbf{t})$ be a BDD instance with HNF basis $\mathbf{B} \in \mathbb{R}^{d\times d}$, error drawn from some product distribution $\mathcal{D}^d$, and bounded log-gap $\gamma_{(\mathbf{B},\mathbf{t})} = \mathcal{O}(1)$. Algorithm 6 solves $(\mathbf{B}, \mathbf{t})$ with constant probability in time $2^{0.292(\mathcal{B}-\mathcal{K})d+o(d)}$, where*

$$\mathcal{B} = \frac{1}{1+2\gamma_{(\mathbf{B},\mathbf{t})}}, \quad \text{and} \quad \mathcal{K} = \frac{\mathcal{B}^2}{\mathcal{B} + \frac{\mathrm{H}(\mathcal{D})}{0.058}}.$$

Comparison with Bernstein. For the special case of BDD instances derived from LWE, Bernstein's attack has runtime $2^{0.292(\mathcal{B}-\mathcal{K}')d+o(d)}$, where

$$\mathcal{K}' = \frac{\mathcal{B}^2}{\mathcal{B}^2 + \frac{\log_2(|\operatorname{supp}(\mathcal{D})|)}{0.058}}.$$

In [Ber23, Section 4.1], Bernstein asked whether one can improve the runtime of his attack, when $\mathcal{D}$ is not the uniform distribution. Our algorithm answers this question in the positive: If $\mathcal{D}$ is the uniform distribution, we obtain the exact same runtime as Bernstein. However, for $\mathcal{D}$ different from the uniform distribution, our algorithm is always more efficient due to the improved enumeration routine from [GMN23].

Implications. Since Kannan's Embedding and Tail-BDD both run in time $2^{0.292\mathcal{B}d+o(d)}$, see Heuristic Claims 3.2 and 3.6, it follows that the hybrid attack with runtime $2^{0.292(\mathcal{B}-\mathcal{K})d+o(d)}$ asymptotically improves over the state-of-the-art by a factor $2^{0.292\mathcal{K}d}$. Let us discuss what this implies for the hardness of BDD.

BDD instances with $\|\mathbf{e}\| = \Theta(\sqrt{d}\det(\mathcal{L})^{1/d})$ are considered to be hardest. (That is, when $\|\mathbf{e}\|$ is within a constant factor of the Gaussian heuristic.) For such instances, we have

$$\gamma_{(\mathbf{B},\mathbf{t})} = \log_d\left(\frac{\sqrt{d}\det(\mathcal{L})^{1/d}}{\|\mathbf{e}\|}\right) = \log_d\left(\mathcal{O}(1)\right) = o(1),$$

and thus $\mathcal{B} = \frac{1}{1+o(1)} = 1 + o(1)$. In particular,

$$\mathcal{K} = \left(1 + \frac{\mathrm{H}(\mathcal{D})}{0.058}\right)^{-1} + o(1).$$

More generally, for our setting of bounded log-gap $\gamma_{(\mathbf{B},\mathbf{t})} = \mathcal{O}(1)$, we always have $\mathcal{B} = \Theta(1)$ and thus

$$\mathcal{K} = \Theta\left(\mathrm{H}(\mathcal{D})^{-1}\right).$$

It follows that the speed-up of the hybrid attack mainly depends on the entropy of the underlying error distribution $\mathcal{D}$: The smaller the entropy, the larger the speed-up. More precisely, for distributions with large entropy $\mathrm{H}(\mathcal{D}) = \omega(1)$, we improve only by a subexponential factor $2^{0.292\mathcal{K}d} = 2^{o(d)}$. However, for all distributions with small entropy $\mathrm{H}(\mathcal{D}) = \mathcal{O}(1)$, we improve by an exponential factor $2^{0.292\mathcal{K}d} = 2^{\Omega(d)}$. In particular, if $|\operatorname{supp}(\mathcal{D})| = \mathcal{O}(1)$, (e.g., if the error is ternary), then the hybrid attack improves over Kannan's Embedding and Tail-BDD by a factor exponential in the lattice dimension. Concrete numbers for various distributions are shown in Table 2.

Implications for LWE. Let us consider an LWE instance $(\mathbf{A}, \mathbf{b}) \in \mathbb{Z}_q^{m\times n} \times \mathbb{Z}_q^m$ with secret $\mathbf{s}_1 \leftarrow \mathcal{D}^n$ and error $\mathbf{s}_2 \leftarrow \mathcal{D}^m$. As in Equation (4), we can transform

Table 2. Asymptotic speed-up $2^{0.292\mathcal{K}d}$ of the hybrid attack for $\mathcal{K} = (1 + \frac{\mathrm{H}(\mathcal{D})}{0.058})^{-1}$.

$\mathcal{D}$	$\mathcal{B}(1)$	$\mathcal{B}(2)$	$\mathcal{B}(3)$	$\mathcal{T}(\frac{2}{3})$	$\mathcal{T}(\frac{1}{3})$	$\mathcal{T}(\frac{1}{6})$
$\mathrm{H}(\mathcal{D})$	1.500	2.030	2.333	1.584	1.251	0.816
$0.292\mathcal{K}$	0.010	0.008	0.007	0.010	0.012	0.019

the LWE instance into a BDD instance with solution $\mathbf{v} = (\mathbf{b} - \mathbf{s}_2, \mathbf{s}_1)$ and error $\mathbf{e} = (\mathbf{s}_2, -\mathbf{s}_1)$. Suppose our LWE instance has parameters as originally suggested by Regev [Reg05], i.e., the coordinates of $\mathbf{e}$ follow a discrete Gaussian distribution $\mathcal{D}$ with standard deviation $\sigma = \mathsf{poly}(n)$. Approximating $\mathrm{H}(\mathcal{D})$ with the entropy of the continous Gaussian distribution, we then have

$$\mathrm{H}(\mathcal{D}) \approx \frac{1}{2}\log_2(2\pi e\sigma^2) = \Omega(\log(n)) = \omega(1),$$

and, by the discussion above, $\mathcal{K} = o(1)$. Hence, the hybrid attack improves here only by a subexponential factor $2^{0.292\mathcal{K}d} = 2^{o(d)}$. In other words, LWE instances with parameters suggested by Regev are essentially immune to the hybrid attack.

However, for efficiency, many *practical* LWE schemes use parameters that are very far from the ones in Regev's reduction. Most crucially, many practical schemes use distributions with rather small entropy, for instance, by using ternary secrets. For such schemes, our analysis shows that the hybrid attack improves over Kannan's Embedding and Tail-BDD by an exponential factor. Hence, while choosing parameters far from Regev's can be beneficial for efficiency, the hybrid attack shows that this comes (at least asymptotically) at the cost of a significant reduction in hardness.

We like to stress that these asymptotic results, however, do not tell much about runtimes for concrete parameters. In particular, we warn against plugging concrete parameters of some practical LWE scheme into Heuristic Claim 3.9 and then (falsely) concluding that breaking that scheme costs time exactly $2^{0.292(\mathcal{B}-\mathcal{K})d}$. To obtain a more accurate picture of the algorithm's performance in practice, we implemented the attack and ran it on concrete instances. The results are discussed in the following chapter.

4 Practical Speed-ups

We provide the first open-source implementation of the Randomized Slicer that incorporates all necessary techniques to make the algorithm truly practical. Building on top of that, we implemented our BDD hybrid attack (Algorithm 6).

Throughout the Section we perform our experiments on 2 identical servers each equipped with 2 AMD EPYC 7742 processors and 2 TB of RAM. With multithreading that yields 128 physical and 256 virtual cores per machine.

4.1 Implementation Details

Our main practical contribution is an implementation of the Randomized Slicer algorithm [DLvW20] for solving Batch-CVP.

Our CPU-based implementation builds on and extends the General Sieve Kernel (G6K) library [ADH+19]. This is the fastest open source implementation of lattice sieving algorithms. G6K includes a CPU implementation due to [DSvW21] of the asymptotically fastest sieve algorithm from [BDGL16] (`BDGL`), which is central to our implementation of Batch-CVP. As input, our implementation receives a list L of short lattice vectors and a batch of CVP targets. It proceeds by rerandomizing the targets to create a list L'. For all $\mathbf{y} \in L'$ the slicer searches for $\mathbf{x} \in L$ that *reduces* $\mathbf{y}$. That is, the slicer searches for $\mathbf{y}$ such that $\|\mathbf{y} \pm \mathbf{x}\| < \|\mathbf{y}\|$. Given such a vector $\mathbf{x}$, the slicer replaces $\mathbf{y}$ with the shorter vector $\mathbf{y} \pm \mathbf{x}$. As explained in Section 2.4, we solve Batch-CVP once L' contains sufficiently many short vectors.

Bucketing. An asymptotically optimal way to search for *reducing* pairs $\mathbf{x}, \mathbf{y}$ is proposed in [BDGL16] and a (more practical) version of it is implemented in [DSvW21]. The idea is to assign the vectors from L, L' into buckets with the property that any two elements assigned to the same bucket are likely to be closer to each other than the elements from different buckets. This bucketing procedure is realised via decoding the vectors with respect to a specifically crafted code. Vectors are assigned to the same bucket if they decode to the same codeword.

The way this idea is implemented in our slicer is pictorially presented in Figure 1. The lattice vectors from L are assigned to buckets $B_1, \ldots, B_m$, and similarly, the target vectors from L' are assigned to their buckets $B'_1, \ldots, B'_m$ *using the same code*, i.e., $B_i \subset L$ and $B'_i \subset L'$ contain vectors that are decoded to the same codeword. Processing buckets consists in checking for each $\mathbf{y} \in B'_i$ whether there exists $\mathbf{x} \in B_i$ that reduces $\mathbf{y}$. If several reductions of $\mathbf{y}$ are possible, we choose the one that gives the shortest output. Once all buckets are processed, we sort the new $\mathbf{y}'$'s according to their Euclidean norm in increasing order.

This finishes one iteration of the slicer. A new iteration of the slicer starts by rerandomizing the code used for bucketing and proceeds with bucketing L and L'.

Rerandomization. In order to have constant success probability of our slicer, all input targets need to be rerandomized [DLvW20]. While in theory this randomization consists in choosing a Gaussian vector from the lattice shifted by the target, such Gaussian sampling is costly in practice. Instead, we choose two relatively close vectors from the list of lattice vectors L and add their sum/difference (whichever is the shortest) to our target. Before randomizing the targets, we reduce them with Babai's algorithm. This significantly reduces the amount of slicer's iterations and decreases the sizes of floating point numbers involved in the computations. Moreover, since we store vectors in L and L' with respect to the Gram-Schmidt basis, Babai-reducing the targets makes their coefficients small.

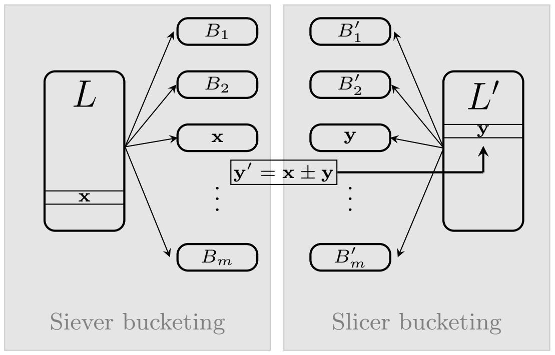

Fig. 1. One iteration of bucketing Randomized Slicer

Termination condition. We terminate the slicer once sufficiently many short vectors in L' are obtained. In our implementation, we allow the slicer to return more vectors than the number of (non-randomized) targets. This is due to the fact that for some targets more than one of their randomizations can be close to a lattice vector. In order to match the original non-reduced, non-randomized target with its closest lattice vector, we keep a database of the non-randomized targets and, for each vector in L' (with randomizations) we store its index in the non-randomized database.

Implementation details: C++/Cython/Python. As in G6K, we implement low level algorithms of the slicer in C++. This includes randomizing the input targets, running bucketing, and sorting the lists. The last two procedures are parallelized. Slicer's bucketing is built upon the implementation of the `bdgl` sieve from [DSvW21]. To provide a user-friendly Python interface to the low-level procedures of the slicer, we use Cython. This enables us to implement Algorithm 6 efficiently using only Python that internally calls efficient C++ functions.

Optimizations. Both targets and lattice vectors are stored in *fixed-size* arrays as coefficients w.r.t. the Gram-Schmidt basis. This makes the norm computations (which is the core operation) faster.

To facilitate these computations, analogously to what is done in G6K, we store for each vector its 256-bit SimHash value [Cha02, FBB+15, Duc18]. A 64-bit hash unique identifier for each vector is stored to keep track of duplicates in L'. We also keep track of a light-weight copy of L', where for each vector we store only its length, a SimHash, and its index in the main database, allowing for more time-optimized sorting. The overall memory requirement per vector in our slicer is slightly less than in the `bdgl` sieving in G6K due to the fact that we do not store the integral coefficient vectors of targets w.r.t. the original basis. Every integral coefficient vector is stored a fixed size array of dimension 128 (by default), thus saving $\approx 2 \cdot 128$ bytes per vector.

4.2 Output Quality of our Slicer for BDD

We compare the success probability of our slicer with the implementation of Babai's algorithm from FPyLLL [dt25]. Babai's algorithm [Bab86] is fast in practice, but it can only solve BDD instances where the target $\mathbf{t}$ is very close to the lattice $\mathcal{L} = \mathcal{L}(\mathbf{B})$, cf. Lemma 2.7. For approximation factors $\gamma = \frac{\mathrm{dist}(\mathbf{t},\mathcal{L})}{\mathrm{GH}(\mathbf{B})}$ close to 1, Babai most likely fails even on well-reduced bases. On the other extreme, there is an implementation of a CVP solver in FPyLLL that enumerates all lattice points around the target within a certain radius.[10] This enumeration almost always succeeds, but its runtime is prohibitive (the experiments on dimensions starting from 70 do not in a reasonable time).

To set up the experiments on the success probability of tail-BBD (Algorithm 3), we first BKZ-reduce 50 random n-dimensional q-ary lattices for $q \approx 12$ bits. We choose $\beta = n - 10$ as the BKZ blocksize. After the BKZ reduction, we run sieving on the whole lattice. For each lattice and for each approximation factor $\gamma \in \{0.9, 0.92, 0.94, 0.96, 0.98, 1\}$, we generate 50 targets by sampling a random error from a sphere and adding it to a lattice vector.

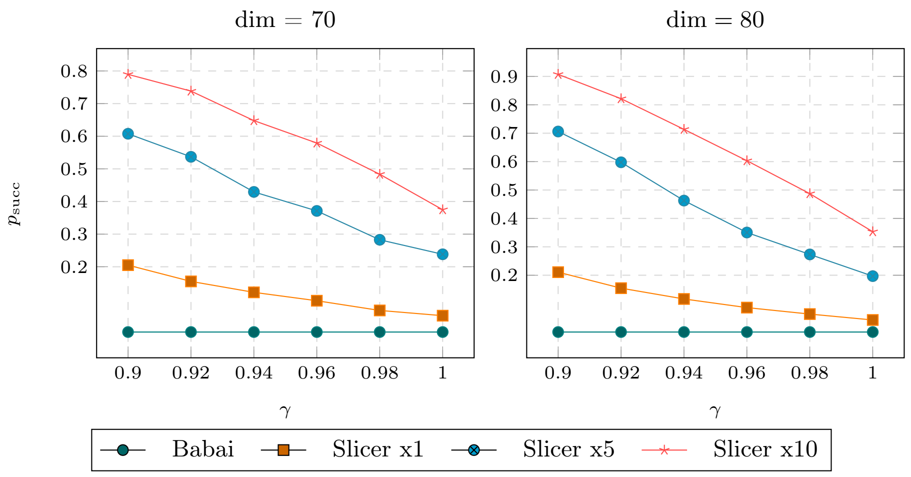

Fig. 2. Success probability of Babai's algorithm vs. our iterative slicer on a q-ary lattice of dimensions 70, 80 on 50 targets for each approximation factor γ. We increase the number of randomizations per target by factors of 1, 5, and 10 compared to asymptotical predictions.

The results of the experiments for dimensions 70 ans 80 are given in Figure 2. We run the slicer algorithm with various numbers of rerandomizations. Theory predicts [DLvW20] the number of randomizations only up to $\widetilde{\mathcal{O}}$-notation that may hide a $\mathrm{poly}(d)$ multiplicative term. Indeed, we observe that considering 1 as

[10] https://github.com/fplll/fpylll/blob/master/src/fpylll/fplll/svpcvp.pyx

the hidden term in $\widetilde{\mathcal{O}}()$ in the number of randomizations results in low success probability of the slicer (see "Slicer x1" orange lines). Increasing this number by a factor of 5 or 10 ("Slicer x5", resp. "Slicer x10") gives a visible improvement in success probability.

Experiments on Batch-Tail-BDD. In order to investigate the efficacy of our Algorithm 4 implementation we generated 5 dimension-120 bases of q-ary lattices with 5 Batch-Tail-BDD instances each containing 10 BDD instances. We run Algorithm 4 with $\beta = 55$ and observe the distributions of approximation factors γ of successfully solved BDD instances (green histogram) and the failed ones (red) in Figure 3. The failures occur due to the probabilistic nature of the slicer: we do not have sufficiently many vectors in L to solve all BDD instances provably. The rate of the fails increases as the approximation factor grows. The data suggests that even when the approximation factor of the BDD targets projected onto $\mathbf{B}_{[d-\beta)}$ is close to 1 we are able to solve most of the BDD instances. A stronger BKZ reduction may increase the success rate of our slicer. In addition, saturating L more will also reduce the number of failures.

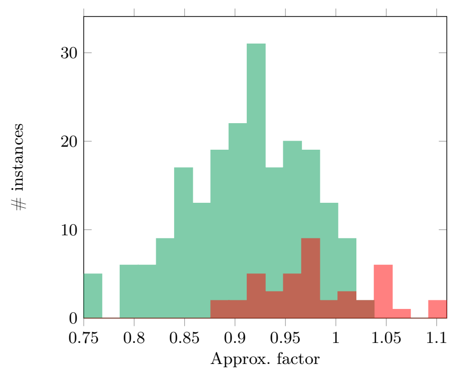

Fig. 3. Histograms of successes and failures at solving Tail-BDD instances combined into the Batch-Tail-BDD instances. Horizontal axis: approximation factor of the BDD targets $\pi_{d-\beta}(\mathbf{t})$ w.r.t. the Gaussian heuristic. Vertical axis: number of BDD instances. Lattice dimension 120, $\beta = 55$. 5 lattices with 5 batch Batch-Tail-BDD instances per each one. Each Batch-Tail-BDD instance consists of 10 BDD instances (250 BDD instances in total).

4.3 Hybrid Attack vs. Primal Attack

In this section we compare the runtimes of the primal attack to the hybrid attack approach for various dimensions and distributions.

We attack LWE instances with secret and error sampled from the centered binomial $\mathcal{B}(3)$ and weighted ternary distribution $\mathcal{T}(w)$ with $w = \frac{1}{3}$ and $\frac{1}{6}$. We conduct our experiments for $\mathcal{B}(3)$ in dimensions $n \in \{140, 150, \ldots, 170\}$. For $\mathcal{T}(1/3)$

and $\mathcal{T}(1/6)$, we run the experiments in dimensions $n \in \{160, 170, \ldots, 210\}$ and $n \in \{170, \ldots, 210\}$, respectively. For each dimension n we generate 10 matrices $\mathbf{A}_i$. For every $\mathbf{A}_i$, we generate 10 LWE instances $(\mathbf{A}_i, \mathbf{b}_j)_{0 \le j < 10}$, resulting in 100 LWE instances per dimension in total.

Primal two-step attack. Our experiments for the primal attack proceed as follows. For each LWE instance we construct a lattice using Kannan's embedding following Algorithm 2. We then run progressive BKZ algorithm on this lattice increasing the value β_{primal} of the BKZ blocksize. For each β_{primal} we compute (for a concrete reduced basis) the value d_s – the least dimension of the last projective lattice such that when calling the sieving on it and lifting the resulting vector yields a solution. This approach to run primal attack is known as two-step approach (bkz+sieving). Both runtimes of BKZ-β_{primal} and sieving in dimension d_s can be estimated as in [ADH+19, XWW+24].[11] When these predicted runtimes equalize, we perform the actual sieving and lift the found vector.

Table 3. Experiment parameters for hybrid attack.

$\mathcal{T}(1/3)$				$\mathcal{T}(1/6)$				$\mathcal{B}(3)$			
n	κ	β	β_s	n	κ	β	β_s	n	κ	β	β_s
160	5	50	53	170	8	50	52	140	1	61	63
170		56	61	180		54	56	150		71	73
180	6	64	66	190		61	63	160		81	83
190		74	76	200		68	70	170		91	93
200		83	85	210		77	79				

Parameter choice for the hybrid attack. To keep the number of variables in our hybrid attack manageable, we fix for each dimension n the number of coordinates κ to be enumerated. The blocksize β for BKZ inside the hybrid attack is manually fine-tuned to some value slightly smaller than β_{primal}, see Table 3.

Improving practical runtime and success probability. We applied a few tweaks to Algorithm 6 to make the algorithm more practical. Recall that for Algorithm 6 to succeed, we need the following:

(i) The correct $\mathbf{u}_2$ is among all M vectors $\mathbf{u}_2^{(i)}$ sampled by our enumeration routine (Algorithm 5).

[11] See also `https://github.com/fplll/g6k/blob/master/lwe_challenge.py` for the concrete implementation of the two-step primal attack.

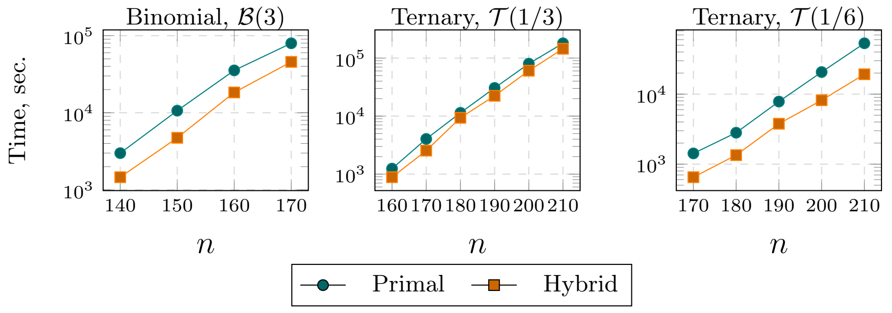

Fig. 4. Timings of the primal attack vs. timings of the hybrid attack adjusted by the inverse of the success rates.

(ii) Batched-Tail-BDD succeeds to solve the BDD instance defined by the golden target over $\mathcal{L}(\mathbf{B}_{[0,d-\kappa)})$.

In theory, [GMN23, Theorem 3.7] requires that we need to sample $M = 2^{\mathrm{H}(\mathcal{D})\kappa+1}$ vectors $\mathbf{u}_2$ for (i) to have a decent success probability. However, experimental evidence [GMN23, Section 5] suggests that $M = 2^{\mathrm{H}(\mathcal{D})\kappa}$ also works well in practice. To improve the runtime of our implementation by a factor 2, we thus work with $M = 2^{\mathrm{H}(\mathcal{D})\kappa}$.

In Algorithm 6 we use the same blocksize β for BKZ reduction in Line 2 and for Batched-Tail-BDD in Line 3. Note that the call to BKZ is thus a bit more costly than the cost to Batched-Tail-BDD, since BKZ runs sieving polynomially many times in dimension β, whereas Batched-Tail-BDD essentially has the same runtime as sieving *once*. Hence, if we use a slightly larger $\beta_s > \beta$ for Batched-Tail-BDD we may balance the costs of Lines 2 and 3, similar to two-step-primal attack. Since the blocksize for Batched-Tail-BDD increases the success probability of (ii), it follows that we may boost Algorithm 6's success probability at essentially no cost in runtime.

To illustrate the benefits of this approach we plot in Figure 5 two success rate histograms: one for $\beta_s = 91$ (left) and one for $\beta_s = 93$ (right). The data shows that increasing the slicer's dimension by 2 increases the success rate from 38% to 52% while not increasing the runtime substantially.

Computation of runtimes. Line 3 of Algorithm 6 feeds a list of M targets to the Randomized Slicer. As shown in Algorithm 1, the Randomized Slicer then creates a list L' of randomized targets and a list of short lattice vectors L. For our parameters, we have $|L'| \gg |L|$. To be able to efficiently search for near neighbours in L and L' we, thus, have to split L' into sublists $L'_1, L'_2, \ldots, L'_N$ with $|L'_i| \approx |L|$ and $N \approx |L'|/|L|$, and run Near Neighbour (NN) on all pairs $L \times L'_i$.

To simplify our experiments, we ran NN only on one pair $L \times L'_i$ with L'_i containing the golden target, and on one pair without the golden target. To

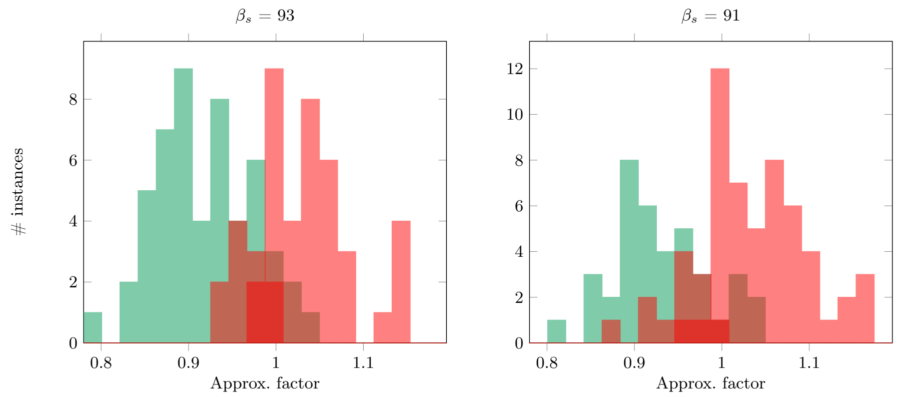

Fig. 5. Histograms of successful secret recoveries, and failures for solving Tail-BDD. Horizontal axis: approximation factor of the golden target $\pi_{d-\beta_s}(\mathbf{t})$ w.r.t. the Gaussian heuristic. Vertical axis: number of instances. Error distribution $\mathcal{B}(3)$, lattice dimension 339, slicer dimension $\beta_s = 93$ (left), 91 (right), $\beta = 91$, 10 lattices lattices with 10 LWE instance per each one.

accurately compute the runtime of the hybrid attack, we then multiply the measured runtime of NN on L'_i without golden target by $(N - 1)$.

Finally, in contrast to the primal attack, our hybrid attack succeeds only with a certain probability. For a fair comparison, we divide the measured runtimes by its success rate.

Runtime. The results of the experiments are presented in Figure 4. The data show that for every dimension and distribution of $\mathbf{s}$ and $\mathbf{e}$ we were able to outperform the primal attack. In some cases (dimensions 170, 180 for $\mathcal{T}(1/3)$ and $\mathcal{T}(1/6)$ respectively) the hybrid attack's runtimes ended up close to the ones of the primal attack. This can be explained by the fact that our BKZ algorithm switches between enumeration and sieving at blocksize $\beta = 55$, making the slope of the primal attack complexity change. This phenomenon disappears as dimensions grow.

In our experiments we observed speedups by factors up to 3.4 for distributions $\mathcal{B}(3)$ and $\mathcal{T}(1/3)$, and up to 4.5 for distribution $\mathcal{T}(1/3)$.

4.4 Directions for improvements

Even though our implementation of Randomized Slicer is the fastest available and includes various optimizations, there is still a lot of directions for potential improvements. Here we list some of them.

1. Sieving algorithms implemented in G6K terminate when "enough" short vectors are found. The exact quantity is by default $0.5 \cdot \sqrt{4/3}^d$, while asymptotically we expect to have $\sqrt{4/3}^d$ short vectors. Increasing the constant

0.5 to, say, 0.75 will increase the success probability of the slicer making sieving slower. The optimal trade-off point depend on the application of the Randomized Slicer.
2. On-the-fly lifting technique, similar to the one used in G6K implementation of sieving, can be used in Algorithms 3 and 4: instead of running BabaiNP after the Randomized Slicer, one can lift the projected solutions to the full lattice.
3. We expect significant speed-ups in GPU implementations of Randomized Slicer, comparable to speed-ups achieved for BDGL sieve in [DSvW21]. Similarly, it is interesting to adapt Randomized Slicer to the streamed memory access implementation of BGJ sieve from [ZDY25].

Acknowledgments. Alexander May and Julian Nowakowski are funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) grant 465120249.

References

ACFP14. Martin R. Albrecht, Carlos Cid, Jean-Charles Faugère, and Ludovic Perret. Algebraic algorithms for LWE. Cryptology ePrint Archive, Report 2014/1018, 2014. 3

ACPS09. Benny Applebaum, David Cash, Chris Peikert, and Amit Sahai. Fast cryptographic primitives and circular-secure encryption based on hard learning problems. In Shai Halevi, editor, *CRYPTO 2009*, volume 5677 of *LNCS*, pages 595–618. Springer, Berlin, Heidelberg, August 2009. 9

ACW19. Martin R. Albrecht, Benjamin R. Curtis, and Thomas Wunderer. Exploring trade-offs in batch bounded distance decoding. In Kenneth G. Paterson and Douglas Stebila, editors, *SAC 2019*, volume 11959 of *LNCS*, pages 467–491. Springer, Cham, August 2019. 2

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, Cham, May 2019. 2, 10, 26, 30

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. 13

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, Cham, December 2017. 13, 18

Ajt96. Miklós Ajtai. Generating hard instances of lattice problems (extended abstract). In *28th ACM STOC*, pages 99–108. ACM Press, May 1996. 2

AKS01. Miklós Ajtai, Ravi Kumar, and D. Sivakumar. A sieve algorithm for the shortest lattice vector problem. In *33rd ACM STOC*, pages 601–610. ACM Press, July 2001. 10

Bab86. László Babai. On lovász'lattice reduction and the nearest lattice point problem. *Combinatorica*, 6:1–13, 1986. 2, 10, 28

BDGL16. Anja Becker, Léo Ducas, Nicolas Gama, and Thijs Laarhoven. New directions in nearest neighbor searching with applications to lattice sieving. In Robert Krauthgamer, editor, *27th SODA*, pages 10–24. ACM-SIAM, January 2016. 11, 26

BDK$^+$18. Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. Crystals-kyber: a cca-secure module-lattice-based kem. In *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 353–367. IEEE, 2018. 3

Ber22. Daniel J. Bernstein. Multi-ciphertext security degradation for lattices. Cryptology ePrint Archive, Report 2022/1580, 2022. 3, 4

Ber23. Daniel J. Bernstein. Asymptotics of hybrid primal lattice attacks. Cryptology ePrint Archive, Report 2023/1892, 2023. 3, 4, 5, 12, 20, 24

BKW00. Avrim Blum, Adam Kalai, and Hal Wasserman. Noise-tolerant learning, the parity problem, and the statistical query model. In *32nd ACM STOC*, pages 435–440. ACM Press, May 2000. 3

BM24. Alessandro Budroni and Erik Mårtensson. Further improvements of the estimation of key enumeration with applications to solving lwe. *Cryptography and Communications*, 16(5):1163–1182, 2024. 2

Boc25. Bochum Challenges, September 2025. Available at `https://bochum-challeng.es`. 3

BSW16. Shi Bai, Damien Stehlé, and Weiqiang Wen. Improved reduction from the bounded distance decoding problem to the unique shortest vector problem in lattices. In Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi, editors, *ICALP 2016*, volume 55 of *LIPIcs*, pages 76:1–76:12. Schloss Dagstuhl, July 2016. 12

CDH$^+$21. C Chen, O Danba, J Hoffstein, A Hülsing, J Rijneveld, T Saito, JM Schanck, P Schwabe, W Whyte, K Xagawa, et al. Ntru: A submission to the nist post-quantum standardization effort, 2021. 6

Cha02. Moses Charikar. Similarity estimation techniques from rounding algorithms. In *34th ACM STOC*, pages 380–388. ACM Press, May 2002. 27

CS97. Don Coppersmith and Adi Shamir. Lattice attacks on NTRU. In Walter Fumy, editor, *EUROCRYPT'97*, volume 1233 of *LNCS*, pages 52–61. Springer, Berlin, Heidelberg, May 1997. 2

CST24. Kevin Carrier, Yixin Shen, and Jean-Pierre Tillich. Faster dual lattice attacks by using coding theory, 2024. 3

CT06. Thomas M. Cover and Joy A. Thomas. *Elements of information theory*. John Wiley & Sons, second edition, 2006. 6

Dar13. TU Darmstadt. LWE Challenges, 2013. 3

DB15. Daniel Dadush and Nicolas Bonifas. Short paths on the voronoi graph and closest vector problem with preprocessing. In Piotr Indyk, editor, *26th SODA*, pages 295–314. ACM-SIAM, January 2015. 11

DDGR20. Dana Dachman-Soled, Léo Ducas, Huijing Gong, and Mélissa Rossi. LWE with side information: Attacks and concrete security estimation. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part II*, volume 12171 of *LNCS*, pages 329–358. Springer, Cham, August 2020. 2, 4

DEL25. Léo Ducas, Lynn Engelberts, and Johanna Loyer. Wagner's algorithm provably runs in subexponential time for SIS$^\infty$. Cryptology ePrint Archive, Paper 2025/575, 2025. 3

DGHK23. Dana Dachman-Soled, Huijing Gong, Tom Hanson, and Hunter Kippen. Revisiting security estimation for LWE with hints from a geometric perspective. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part V*, volume 14085 of *LNCS*, pages 748–781. Springer, Cham, August 2023. 2

DKL$^+$18. Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS-Dilithium: A lattice-based digital signature scheme. *IACR TCHES*, 2018(1):238–268, 2018. 3

DLdW19. Emmanouil Doulgerakis, Thijs Laarhoven, and Benne de Weger. Finding closest lattice vectors using approximate voronoi cells. In Jintai Ding and Rainer Steinwandt, editors, *Post-Quantum Cryptography - 10th International Conference, PQCrypto 2019*, pages 3–22. Springer, Cham, 2019. 3, 11, 12

DLvW20. Léo Ducas, Thijs Laarhoven, and Wessel P. J. van Woerden. The randomized slicer for CVPP: Sharper, faster, smaller, batchier. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *PKC 2020, Part II*, volume 12111 of *LNCS*, pages 3–36. Springer, Cham, May 2020. 3, 11, 12, 26, 28

DSvW21. Léo Ducas, Marc Stevens, and Wessel P. J. van Woerden. Advanced lattice sieving on GPUs, with tensor cores. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part II*, volume 12697 of *LNCS*, pages 249–279. Springer, Cham, October 2021. 26, 27, 33

dt25. The FPLLL development team. fpylll, a Python wrapper for the fplll lattice reduction library, Version: 0.6.3. Available at `https://github.com/fplll/fpylll`, 2025. 28

Duc18. Léo Ducas. Shortest vector from lattice sieving: A few dimensions for free. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part I*, volume 10820 of *LNCS*, pages 125–145. Springer, Cham, April / May 2018. 27

EK20. Thomas Espitau and Paul Kirchner. The nearest-colattice algorithm: Time-approximation tradeoff for approx-cvp. *Open Book Series*, 4(1):251–266, 2020. 3, 4, 11, 15

FBB$^+$15. Robert Fitzpatrick, Christian H. Bischof, Johannes Buchmann, Özgür Dagdelen, Florian Göpfert, Artur Mariano, and Bo-Yin Yang. Tuning GaussSieve for speed. In Diego F. Aranha and Alfred Menezes, editors, *LATINCRYPT 2014*, volume 8895 of *LNCS*, pages 288–305. Springer, Cham, September 2015. 27

GJMS17. Qian Guo, Thomas Johansson, Erik Mårtensson, and Paul Stankovski. Coded-BKW with sieving. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part I*, volume 10624 of *LNCS*, pages 323–346. Springer, Cham, December 2017. 3

GM23. Timo Glaser and Alexander May. How to enumerate LWE keys as narrow as in Kyber/Dilithium. In Jing Deng, Vladimir Kolesnikov, and Alexander A. Schwarzmann, editors, *CANS 23*, volume 14342 of *LNCS*, pages 75–100. Springer, Singapore, October / November 2023. 3

GMN23. Timo Glaser, Alexander May, and Julian Nowakowski. Entropy suffices for guessing most keys. Cryptology ePrint Archive, Report 2023/797, 2023. Version 20240308:085956. 2, 5, 20, 21, 24, 31

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, Berlin, Heidelberg, August 2007. 3

HPS98. Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. NTRU: A ring-based public key cryptosystem. In *Third Algorithmic Number Theory Symposium (ANTS)*, volume 1423 of *LNCS*, pages 267–288. Springer, June 1998. 2

Kan83. Ravi Kannan. Improved algorithms for integer programming and related lattice problems. In *15th ACM STOC*, pages 193–206. ACM Press, April 1983. 2, 12

Laa16. Thijs Laarhoven. Sieving for closest lattice vectors (with preprocessing). In Roberto Avanzi and Howard M. Heys, editors, *SAC 2016*, volume 10532 of *LNCS*, pages 523–542. Springer, Cham, August 2016. 11

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, Berlin, Heidelberg, February / March 2013. 3, 4

LP11. Richard Lindner and Chris Peikert. Better key sizes (and attacks) for LWE-based encryption. In Aggelos Kiayias, editor, *CT-RSA 2011*, volume 6558 of *LNCS*, pages 319–339. Springer, Berlin, Heidelberg, February 2011. 3, 4

May21. Alexander May. How to meet ternary LWE keys. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part II*, volume 12826 of *LNCS*, pages 701–731, Virtual Event, August 2021. Springer, Cham. 3

MN23. Alexander May and Julian Nowakowski. Too many hints - when LLL breaks LWE. In Jian Guo and Ron Steinfeld, editors, *ASIACRYPT 2023, Part IV*, volume 14441 of *LNCS*, pages 106–137. Springer, Singapore, December 2023. 2

MR09. Daniele Micciancio and Oded Regev. Lattice-based cryptography. In *Post-quantum cryptography*, pages 147–191. Springer, 2009. 3

MS01. Alexander May and Joseph H Silverman. Dimension reduction methods for convolution modular lattices. In *International Cryptography and Lattices Conference*, pages 110–125. Springer, 2001. 2, 4

MV10. Daniele Micciancio and Panagiotis Voulgaris. Faster exponential time algorithms for the shortest vector problem. In Moses Charika, editor, *21st SODA*, pages 1468–1480. ACM-SIAM, January 2010. 10

Ngu21. Phong Q Nguyen. Boosting the hybrid attack on ntru: Torus lsh, permuted hnf and boxed sphere. In *NIST Third PQC Standardization Conference*, 2021. 3

NV08. Phong Q. Nguyen and Thomas Vidick. Sieve algorithms for the shortest vector problem are practical. *Journal of Mathematical Cryptology*, 2(2):181–207, 2008. 10

PS24. Amaury Pouly and Yixin Shen. Provable dual attacks on learning with errors. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 256–285. Springer, 2024. 3

Reg05. Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *37th ACM STOC*, pages 84–93. ACM Press, May 2005. 2, 3, 5, 9, 25

Sch87. Claus-Peter Schnorr. A hierarchy of polynomial time lattice basis reduction algorithms. *Theoretical computer science*, 53(2-3):201–224, 1987. 10

Sch03. Claus Peter Schnorr. Lattice reduction by random sampling and birthday methods. In *STACS 2003: 20th Annual Symposium on Theoretical Aspects of Computer Science Berlin, Germany, February 27–March 1, 2003 Proceedings 20*, pages 145–156. Springer, 2003. 13

SFS09. Naftali Sommer, Meir Feder, and Ofir Shalvi. Finding the closest lattice point by iterative slicing. *SIAM Journal on Discrete Mathematics*, 23(2):715–731, 2009. 11

Ste24. Matthias Johann Steiner. The complexity of algebraic algorithms for LWE. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part III*, volume 14653 of *LNCS*, pages 375–403. Springer, Cham, May 2024. 3

vW20. Wessel van Woerden. Randomized Slicer Implementation, 2020. 3, 5

WBWL23. Yu Wei, Lei Bi, Kunpeng Wang, and Xianhui Lu. An improved BKW algorithm for solving LWE with small secrets. In Elias Athanasopoulos and Bart Mennink, editors, *ISC 2023*, volume 14411 of *LNCS*, pages 578–595. Springer, Cham, November 2023. 3

Wun19. Thomas Wunderer. A detailed analysis of the hybrid lattice-reduction and meet-in-the-middle attack. *Journal of Mathematical Cryptology*, 13(1):1–26, 2019. 3

WXG25. Geng Wang, Wenwen Xia, and Dawu Gu. Heuristic algorithm for solving restricted svp and its applications. In *Post-Quantum Cryptography*, pages 119–152, 2025. 3, 5

XWG25. Wenwen Xia, Geng Wang, and Dawu Gu. Solve approximate cvp via variants of nearest-colattice. Cryptology ePrint Archive, Report 2025/1176, 2025. 5

XWW^{+}24. Wenwen Xia, Leizhang Wang, Geng Wang, Dawu Gu, and Baocang Wang. A refined hardness estimation of LWE in two-step mode. In Qiang Tang and Vanessa Teague, editors, *PKC 2024, Part II*, volume 14603 of *LNCS*, pages 3–35. Springer, Cham, April 2024. 30

ZDY25. Ziyu Zhao, Jintai Ding, and Bo-Yin Yang. Sieving with streaming memory access. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2025(2):362–384, Mar. 2025. 33