Parallel Implementation of BDD enumeration for LWE

Elena Kirshanova, Alexander May, Friedrich Wiemer

Horst Görtz Institute for IT-Security
Faculty of Mathematics
Ruhr University Bochum, Germany
{elena.kirshanova, alex.may, friedrich.wiemer}@rub.de

Abstract. One of the most attractive problems for post-quantum secure cryptographic schemes is the LWE problem. Beside combinatorial and algebraic attacks, LWE can be solved by a lattice-based Bounded Distance Decoding (BDD) approach. We provide the first parallel implementation of an enumeration-based BDD algorithm that employs the Lindner-Peikert and Linear Length pruning strategies. We ran our algorithm on a large variety of LWE parameters, from which we derive the following interesting results. First, our parallel enumeration achieves almost perfect speed-up, which allows us to provide for the first time practical cryptanalytic results on standard LWE parameters of meaningful size. Second, we conclude that lattice-based attacks perform better than recent advanced BKW-type algorithms even for small noise, while requiring way less samples. Third, we experimentally show weaknesses for a binary matrix LWE proposal of Galbraith.
Keywords. LWE security, Bounded Distance Decoding, Lattices

1 Introduction

Estimating the hardness of the *Learning with Errors Problem* (LWE) is of great importance in cryptography since its introduction by Regev [20]. Nowadays, the standard way to check concrete hardness of an LWE instance is by comparison with tables in LWE cryptanalysis papers (see [5, 16, 17] for lattice-based attacks, [1, 12, 15] for combinatorial attacks of BKW-type, [3] for an algebraic attack). Also, [2] provides a publicly available LWE-estimator that collects all known attacks and predicts their running-times on given LWE parameters. Due to the large memory- and sample-complexity of combinatorial algorithms, the lattice-based approach seems more practical. This belief was questioned by a recent result on BKW of Kirchner and Fouque [15], where an LWE instance of dimension 128 was solved in 13 hours. Currently, this is the record for combinatorial attacks on LWE. So it is reasonable to ask whether a similar result can be achieved by lattice-based attacks.

In this paper we present results on a parallel implementation of lattice-based attacks on LWE. We view the LWE problem as a BDD instance on a q-ary lattice. From here there are two approaches to go for: one can solve a BDD instance

either via Kannan's embedding [14], or via reducing a lattice basis first and then solving a CVP problem on a reduced basis (reduce-then-decode). While Kannan's embedding performs well for small dimensions [18], its complexity grows with the dimension since the algorithm calls an SVP solver as a subroutine.

We take the reduce-then-decode approach because the decoding part contains a tree-traversal algorithm that can be almost perfectly parallelized.

Our main contribution is a parallelization of BDD enumeration [16, 17]. From our experiments we conclude that:

1. BDD enumeration can be almost perfectly parallelized, i.e. with n processors the achieved speed-up is roughly n.
2. For standard LWE-settings (e.g. uniform secret) instances with dimension of order $n = 100$ can be broken in several hours (see Sect. 5)
3. Lattice-based techniques are more efficient than current combinatorial algorithms even for binary secret.
4. Small error rates in BDD (binary or ternary error-vectors) allow for a much more efficient decoding.
5. A concrete instance of a space-efficient LWE variant of Galbraith [9] is weaker than previously thought (see Sect. 4)

To the best of our knowledge, our implementation provides the first results for lattice-based enumeration attacks on *concrete* LWE instances. Our attack is carried out in combination with the BKZ algorithm implemented in the NTL library [21]. Further improvements of lattice reduction (like in [8]) would in combination with our parallel BDD implementation certainly speed-up the attacks even further. Our code will be made available online.[1]

The remainder of this paper is organized as follows. Section 2 covers notations and background. In Section 3 we describe Babai's enumeration algorithm and its generalization. Our main algorithm, the parallelized BDD enumeration, is described in Section 3. Section 4 discusses variants of LWE and differences to the standard BDD attack. Our implementation results are presented in Section 5.

2 Background

We use bold lower-case letters for vectors $\mathbf{b}$ and we let $\|\mathbf{b}\|$ denote their Euclidean norm. For vectors $(\mathbf{b}_1, \ldots, \mathbf{b}_k)$, we construct a basis matrix $\mathbf{B}$ consisting of rows $\mathbf{b}_i$. For linearly independent $(\mathbf{b}_1, \ldots, \mathbf{b}_k) \in \mathbb{R}^m$, the *fundamental domain* $\mathcal{P}_{1/2}(\mathbf{B})$ is $\left\{\sum_{i=1}^k c_i \mathbf{b}_i \colon c_i \in [-\frac{1}{2}, \frac{1}{2})\right\}$. The *Gram-Schmidt orthogonalization* $\widetilde{\mathbf{B}} = (\widetilde{\mathbf{b}}_1, \ldots, \widetilde{\mathbf{b}}_k)$ is obtained iteratively by setting $\widetilde{\mathbf{b}}_1 = \mathbf{b}_1$ and $\widetilde{\mathbf{b}}_i$ as the orthogonal projection of $\mathbf{b}_i$ on $(\mathbf{b}_1, \ldots, \mathbf{b}_{i-1})^{\perp}$ for $i = 2, \ldots, k$. This orthogonalization process can be described via matrix-decomposition $\mathbf{B} = \mu\widetilde{\mathbf{B}}$, where μ is a lower-triangular matrix with $\mu_{i,j} = \langle \mathbf{b}_i, \widetilde{\mathbf{b}}_j \rangle / \|\widetilde{\mathbf{b}}_j\|^2$ for $i \geq j$.

[1] https://github.com/pfasante/cvp-enum

We deal with a q-ary lattice with basis $\mathbf{B}$:

$$\Lambda_q(\mathbf{B}) = \Big\{\mathbf{y} \in \mathbb{Z}^m \colon \mathbf{y} = \sum_{i=1}^{k} z_i \cdot \mathbf{b}_i \mod q \colon z_i \in \mathbb{Z}\Big\}.$$

Vectors from this lattice are in $\mathrm{Im}(\mathbf{B})$. The kernel of matrix $\mathbf{B}$ forms another lattice $\Lambda_q^{\perp}(\mathbf{B}) = \{\mathbf{x} \in \mathbb{Z}^k \colon \mathbf{x}\mathbf{B} = 0 \mod q\}$. For a lattice $\Lambda(\mathbf{B})$, the first successive minimum $\lambda_1(\Lambda(\mathbf{B}))$ is the length of its shortest vector.

In this paper we describe an algorithm to solve the so-called *Bounded Distance Decoding Problem* (BDD) and the most cryptographically relevant instance of it, the *Learning with Errors Problem* (LWE). BDD asks to find a lattice point $\mathbf{v}$ closest to a given point $\mathbf{t} \in \mathbb{R}^m$ under the promise that $\|\mathbf{v}-\mathbf{t}\| = \|\mathbf{e}\| \leq R$, where R is usually much smaller than the lattice's packing radius. In the LWE case, we know in addition that the error-vector $\mathbf{e}$ is distributed as a *discrete Gaussian* i.e. its probability distribution, denoted D_s, is proportional to $\exp(-\pi\|\mathbf{e}\|^2/s^2)$. In LWE it suffices to consider the integer lattice $\mathbb{Z}$ as a support for the error distribution, so we used the Ziggurat Algorithm implemented in [6] for the sampling. A discrete Gaussian sampler over *any* lattice can be found in [11].

Apart from the scaled standard deviation s, the LWE problem is parametrized by a dimension $n \geq 1$, an integer modulus $q = \mathrm{poly}(n)$ and the number of LWE samples m. For secret $\mathbf{s} \in \mathbb{Z}_q^n$, an *LWE sample* is obtained by choosing a vector $\mathbf{a} \in \mathbb{Z}_q^n$ uniformly at random, an error $e \leftarrow D_s$, and outputting m pairs $(\mathbf{a}, t = \langle \mathbf{a}\,, \mathbf{s}\rangle + e \bmod q) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$. Typically a cryptosystem reveals $m = \Theta(n)$ samples (commonly as a public key) and for lattice-based attack we consider $m \leq 2n$.

We write the obtained m pairs as $(\mathbf{A}, \mathbf{t} = \mathbf{s}\mathbf{A} + \mathbf{e} \bmod q) \in \mathbb{Z}^{n\times m} \times \mathbb{Z}^m$ for $\mathbf{t} = (t_1, \ldots, t_m)$, $\mathbf{e} = (e_1, \ldots, e_m)$ and the columns of matrix $\mathbf{A}$ are composed of the $\mathbf{a}_i$. From this it is easy to see that (the search version of) the LWE problem is an average-case hard Bounded Distance Decoding problem for the q-ary lattice $\Lambda(\mathbf{A}) = \{\mathbf{z} \in \mathbb{Z}^m \colon \exists \mathbf{s} \in \mathbb{Z}_q^n \text{ s.t. } \mathbf{z} = \mathbf{s}\mathbf{A} \bmod q\}$, i.e. $\mathbf{t}$ is close to a linear combination of rows of $\mathbf{A}$. Assuming $\mathbf{A}$ is full-rank (which is the case w.h.p.), its determinant is $\det(\Lambda(\mathbf{A})) = q^{m-n}$ and the rows of the matrix below form its basis over $\mathbb{Z}^m$

$$\mathbf{B} = \begin{pmatrix} \mathbf{A}' & \mathbf{I}_{m-n} \\ q\mathbf{I}_{m-n} & \mathbf{0} \end{pmatrix} \in \mathbb{Z}^{m\times m}, \tag{1}$$

where $\mathbf{A} = (\mathbf{A}'|\mathbf{I}_{m-n})$ and $\mathbf{A}' \in \mathbb{Z}^{n\times n}$ is a row-reduced echelon form of $\mathbf{A}$.
Reduce-then-decode is our approach to solve LWE in practice. For the reduction step, we β-BKZ reduce the basis defined in Eq. (1). The reduction's running time is determined by m and the running time of an SVP-solver on a lattice of dimension β. Our decoding step is described in the subsequent section.

3 Enumeration Tree

Let us describe our implementation of the tree-traversal algorithm for the BDD enumeration. Recall that a BDD instance is given by a (BKZ-reduced) basis

$\mathbf{B} \in \mathbb{Z}^{m \times m}$ and a target $\mathbf{t} \in \mathbb{Z}^m$ that is close to a lattice point $\mathbf{v} = \sum_{k=1}^{m} v_k \mathbf{b}_k$. Our goal is to find the coordinates v_k. Knowing that $\mathbf{t} - \mathbf{v} = \mathbf{e}$ is short, we enumerate over all coefficient vectors $(v_m, \ldots, v_1)$ that result in a vector close to $\mathbf{t}$. A way to find the coordinates v_k via iterative projections is the Nearest Plane Algorithm of Babai [4]. In the k-th iteration ($k = m, \ldots, 1$), the target $\mathbf{t}$ is projected onto $\mathrm{Span}(\mathbf{b}_1, \ldots, \mathbf{b}_{k-1})^{\perp}$ choosing the closest translate of the sub-lattice $\Lambda(\mathbf{b}_1, \ldots, \mathbf{b}_{k-1})$ (line 4, Alg. 1) and the projected vector becomes a new target (line 5). The procedure results in a closest vector $\mathbf{v}$, s.t. $\|\mathbf{e}\| \leq 1/2\sqrt{\sum_{k=1}^{m} \|\widetilde{\mathbf{b}}_k\|^2}$. An iterative version of the Nearest Plane Algorithm is presented in Algorithm 1.

Algorithm 1 Babai's `NearestPlane` $(\mathbf{B}, \mathbf{t})$

Input: $\mathbf{B} = (\mathbf{b}_1, \ldots, \mathbf{b}_m) \in \mathbb{Z}^{m \times m}, \mathbf{t} \in \mathbb{Z}^m$
Output: $\mathbf{v} \in \mathcal{L}(\mathbf{B})$ close to $\mathbf{t}$ and $e = \|\mathbf{e}\| = \|\mathbf{t} - \mathbf{e}\|$

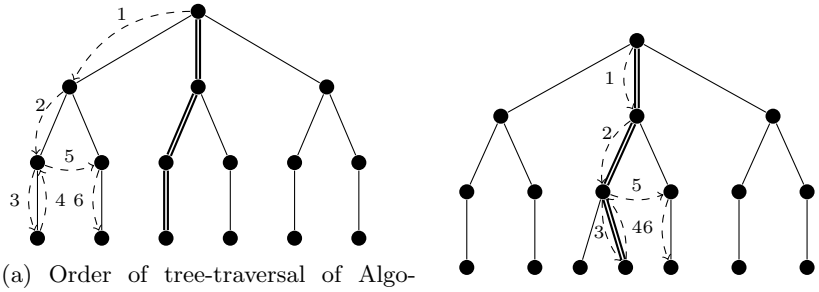
1: $\mathbf{t}^{(m)} \leftarrow \mathbf{t}, e^{(m)} \leftarrow 0, k \leftarrow m.$
2: Let $\widetilde{\mathbf{B}} \leftarrow \mathrm{GSO}(\mathbf{B})$
3: **while** $k > 0$ **do**
4: $\quad c^{(k)} \leftarrow \left\langle \mathbf{t}^{(k)}, \frac{\widetilde{\mathbf{b}}_k}{\|\widetilde{\mathbf{b}}_k\|^2} \right\rangle$ ▷ Compute the closest hyperplane $U^{(k)}$
5: $\quad \mathbf{t}^{(k-1)} \leftarrow \mathbf{t}^{(k)} - \lceil c^{(k)} \rfloor \mathbf{b}_k$ ▷ Project onto $U^{(k)} = c^{(k)}\widetilde{\mathbf{b}}_k + \mathrm{Span}(\mathbf{b}_1, \ldots, \mathbf{b}_{k-1})$
6: $\quad e^{(k-1)} \leftarrow e^{(k)} + (c^{(k)} - \lceil c^{(k)} \rfloor)^2 \|\widetilde{\mathbf{b}}_k\|^2$ ▷ Compute the squared error-length
7: $\quad k \leftarrow k - 1$
8: **return** $(\mathbf{t} - \mathbf{t}^{(0)}, e^0)$

While the above Nearest Plane procedure is very efficient even for large m, the output $\mathbf{t}^{(0)}$ is the correct one only if $\mathbf{e} \in \mathcal{P}_{1/2}(\mathbf{B})$. As a given basis $\mathbf{B}$ may be 'far away' from being orthogonal, the choice of the closest hyperplane (line 4, Alg. 1) may not lead to the actual closest vector. On each iteration, the additive factor to the squared error-length can be as large as $\frac{1}{2}\|\widetilde{\mathbf{b}}\|^2$.

To mitigate the non-orthogonality of the input basis, Lindner and Peikert [16] proposed to project on *several* close hyperplanes, i.e. in Step 5 of Algorithm 1, $c_i^{(k)}$, $1 \leq i \leq d_k$ are chosen, resulting in d_k new targets $t_i^{(k-1)}$. To guarantee a constant success probability, d_k must be chosen such that $d_k \cdot \|\widetilde{\mathbf{b}}_k\| > 2e_k$, i.e. the error-vector $\mathbf{e}$ must be contained in the stretched fundamental parallelepiped $\mathcal{P}_{1/2}(\mathbf{B} \cdot \mathrm{diag}(d_1, \ldots d_m))$. For the LWE-case the sequence $(d_i)_{i=1,\ldots,m}$ can be computed given $(\|\widetilde{\mathbf{b}}_i\|)_{i=1,\ldots,m}$ and the parameter s.

Our algorithm is implemented as a depth-first tree traversal where each level-k node ($k = m, \ldots, 1$), represents a partial assignment $(c^{(m)}, \ldots, c^{(k)})$ of the target $\mathbf{t}^{(k)} = \mathbf{t} - \sum_{i=k}^{m} c^{(i)}\mathbf{b}_i$. A children-set for this node is generated by projecting $\mathbf{t}^{(k)}$ onto d_{k-1} closest hyperplanes $U_i^{(k-1)} = c_i^{(k-1)}\widetilde{\mathbf{b}}_{k-1} + \mathrm{Span}(\mathbf{b}_1, \ldots, \mathbf{b}_{m-k})$, $i = 1, \ldots, d_{k-1}$. Each leaf is a candidate-solution $v = \sum_{i=1}^{m} c^{(i)}\mathbf{b}_i$, whose corresponding error is checked against the currently shortest. Figure 1a represents the case $m = 3$, $d_1 = 3$, $d_2 = 2$, $d_1 = 1$.

Note that the length of an error-vector is not explicitly bounded by the Lindner-Peikert enumeration tree. Instead, one imposes a restriction on its individual coordinates e_i. In Liu and Nguyen's Length Pruning Algorithm [17],



(a) Order of tree-traversal of Algorithm 2. The left-most children are visited first.

(b) Order of tree-traversal of the implemented best-first search.

Fig. 1: Orders of tree-traversal

the number of children for a node is determined only by the length of the error accumulated so far and hence, as opposed to the Lindner-Peikert strategy, might differ for two nodes on the same level. For Gaussian error, one would expect that on level k the value $e^{(k-1)}$ (line 6, Alg. 1) satisfies $e^{(k-1)} < R_k \approx s^2(m-k+1)$ resulting in $e^{(0)} = \|\mathbf{e}\| = s^2 m$. This strategy is called *Linear Pruning* and is used in our experiments. We do not consider the so-called *Extreme Pruning* strategy where the bounds satisfy $R_k \ll s^2(m-k+1)$ (i.e. the success probability is very low, but boosted via re-randomizing the basis and repeating). While Extreme Pruning proved to be more efficient in the SVP setting [10], in the BDD case re-randomizing an instance causes re-running the expensive BKZ reduction (as the re-randomization distorts the reducedness).

Both enumeration strategies, Lindner-Peikert and Length Pruning, can be generalized by considering a family of bounding functions $B^{(k)} : \mathbb{Q} \to \mathbb{Q}$, $1 \leq k \leq m$ that take a squared error-length as input and output the remaining allowed length depending on the chosen strategy. From the value $B^{(k)}(e^{(k)})$, one can compute the number of children for a node on level k (line 6, Alg. 2). The Lindner-Peikert bounding function ignores the error-length setting $B^{(k)} = (d_k \|\widetilde{\mathbf{b}}_k\|)^2$ by having d_k children for *all* k-level nodes. For the Length Pruning of [17], we set $B^{(k)} = R_k - e^{(k)}$. Our BDD Enumeration in Algorithm 2 describes the depth-first tree-traversal under this generalization.

Algorithm 2 constructs an enumeration tree with a k-level node storing a target-vector $\mathbf{t}^{(k-1)}$, a coefficient vector $c^{(k)}$ of a candidate-solution $\sum_{k=1}^{m} c^{(k)}\mathbf{b}_k$ and an accumulated error-length $e^{(k-1)}$ (lines 10–12). A path from a root ($k = m$) to a leaf ($k = 1$) gives one candidate-solution $\mathbf{v} = \sum_{k=1}^{m} c^{(k)}\mathbf{b}_k$ with error-length $e^{(0)} = \mathbf{t} - \mathbf{v}$. The path with the minimal error-value is the output of the algorithm.

Notice that different paths have different success probabilities: the path corresponding to Babai's solution $\sum_{k=1}^{m} c^{(k)}\mathbf{b}_k$ is the most promising one. So instead of choosing the left-most child and traversing its sub-tree, the implemented tree-traversal algorithm chooses Babai's path first, i.e. a 'middle' child of a node, and

Algorithm 2 BDD Enumeration $(\mathbf{B}, \mathbf{t}, B^{(k)})$

Input: $\mathbf{B} = (\mathbf{b}_1, \ldots, \mathbf{b}_m) \in \mathbb{Z}^{m \times m}, \mathbf{t} \in \mathbb{Z}^m$, a family of bounding functions $B^{(k)} : \mathbb{Q} \to \mathbb{Q}$

Output: $\mathbf{v} \in \mathcal{L}(\mathbf{B})$ close to $\mathbf{t}$ and $e = \|\mathbf{e}\| = \|\mathbf{t} - \mathbf{e}\|$

$\mathbf{t}^{(m)} \leftarrow \mathbf{t}, e^{(m)} \leftarrow 0, k \leftarrow m.$
Let $\widetilde{\mathbf{B}} \leftarrow \mathrm{GSO}(\mathbf{B})$
$(\mathbf{t}^{(0)}, \mathrm{minLen}) \leftarrow \texttt{NearestPlane}(\mathbf{B}, \mathbf{t})$
while (true) **do**
  if $(k > 0)$ **then**
    $Int \leftarrow \sqrt{B^{(k)}(e^{(k)})}/\|\widetilde{\mathbf{b}}_k\|$ ▷ Number of children
    $c^* \leftarrow \langle \mathbf{t}^{(k)}, \widetilde{\mathbf{b}}_k \rangle / \|\widetilde{\mathbf{b}}_k\|^2$
    $c_{\min} \leftarrow \lceil c^* - \frac{1}{2} Int \rceil$ ▷ Left-most child
    $c_{\max} \leftarrow \lfloor c^* + \frac{1}{2} Int \rfloor$ ▷ Right-most child
    $c^{(k)} \leftarrow c_{\min}$
    $\mathbf{t}^{(k-1)} \leftarrow \mathbf{t}^{(k)} - c^{(k)} \mathbf{b}_k$ ▷ Project onto $U^{(k)} = c^{(k)} \widetilde{\mathbf{b}}_k + \mathrm{Span}(\mathbf{b}_1, \ldots, \mathbf{b}_{k-1})$
    $e^{(k-1)} \leftarrow e^{(k)} + (c^{(k)} - c^*)^2 \|\widetilde{\mathbf{b}}_k\|^2$ ▷ Compute the squared error-length
    $k \leftarrow k - 1$ ▷ Go down the tree
  else ▷ On a leaf
    if $(e^{(k)} < \mathrm{minLen})$ **then**
      $\mathbf{v} \leftarrow \sum_{i=1}^{k} c^{(i)} \mathbf{b}_i$ ▷ Current best solution
      $\mathrm{minLen} = e^{(k)}$
    repeat ▷ Traverse up
      if $(k = 0$ AND $c^{(k)} > c_{\max})$ **then** ▷ On the root, no right siblings
        return $(\mathbf{v}, \mathrm{minLen})$
      $k \leftarrow k + 1$
    until $(c^{(k)} \geq c_{\max})$
    $c^{(k)} \leftarrow c^{(k)} + 1$ ▷ Traverse to the right sibling
    $\mathbf{t}^{(k-1)} \leftarrow \mathbf{t}^{(k)} - \lceil c^{(k)} \rfloor \mathbf{b}_k$
    $e^{(k-1)} \leftarrow e^{(k)} + (c^{(k)} - \lceil c^{(k)} \rfloor)^2 \|\widetilde{\mathbf{b}}_k\|^2$
return $(\mathbf{t}^{(0)}, e^{(0)})$

then examines all nearby paths. This strategy of ordering the paths by decreasing success probability is called *Length best first search* (see Fig. 1b).

3.1 Parallel implementation

In Algorithm 2, sub-tree traversals for two different nodes on the same level are independent, so we can parallelize the BDD Enumeration. Let #NThreads be the number of threads (processors) available. Our goal is to determine the upper-most level k having at least as many nodes $\#\mathrm{N}(k)$ as #NThreads. Then we can traverse the $\#\mathrm{N}(k)$ sub-trees in parallel by calling Algorithm 2 on each thread.

We start traversing the enumeration tree in a *breadth-first* manner using a queue. In a breadth-first traversal, once all the nodes of level k are visited, the queue contains all their children (i.e. all the nodes of level $k+1$), thus their

number $\#\mathrm{N}(k+1)$ can be computed. Once a level k with $\#\mathrm{N}(k) \geq c \cdot \#\mathrm{NThreads}$ for some small constant $c \geq 1$ is found, we stop the breadth-first traversal and start Algorithm 2 for each of the $\#\mathrm{N}(k)$ sub-trees in an own thread. The benefit of having $c > 1$ is that whenever one of the threads finishes quickly, it can be assigned to traverse another sub-tree. This strategy compensates for imbalanced sizes of sub-trees.

This breadth-first traversal is described in Algorithm 3. At the root we have $\#\mathrm{N}(m) = 1$. The associated data to each node are the target $\mathbf{t}^{(m-1)}$, the error-length $e^{(m-1)}$ and the partial solution $\mathbf{s}^{(m-1)}$. We store them in queues Q_t, Q_e, Q_s. Traversing the tree down is realized via dequeuing the first element from a queue (line 9) and enqueuing its children into the queue. When Algorithm 3 terminates, we spawn a thread that receives as input a target $\mathbf{t}^{(k)}$ from Q_t, an accumulated so far error-length $e^{(k)} \in Q_e$, a partial solution $\mathbf{s}^{(k-1)} \in Q_s$, GSO-lengths $(\|\widetilde{\mathbf{b}}_{k-1}\|, \ldots, \|\widetilde{\mathbf{b}}_1\|)$ and bounding functions $B^{(i)}$, $1 \leq i \leq k-1$. Since the number of possible threads is usually a small constant, there is no blow-up in memory usage in the breadth-first traversal.

Note that for a family of bounding functions $B^{(k)}$ that allows to compute the number of children per node without actually traversing the tree, e.g. the Lindner-Peikert bounding strategy, it is easy to find the level where we start parallelization. In case of Lindner-Peikert, $\#\mathrm{N}(k) = \prod_{i=m}^{m-k} d_i$ and hence, we simply compute the largest level k where $\#\mathrm{N}(k) \geq c \cdot \#\mathrm{NThreads}$.

Algorithm 3 Traverse Breadth-First $(\mathbf{B}, \mathbf{t}, B^{(k)})$

Input: $\mathbf{B} = (\mathbf{b}_1, \ldots, \mathbf{b}_m) \in \mathbb{Z}^{m \times m}, \mathbf{t} \in \mathbb{Z}^m$, a family of bounding functions $B^{(k)}$, $\#\mathrm{NThreads} \in \mathbb{Z}$, $c \in \mathbb{Z}$

Output: An array $(\mathbf{t}^{(k)})_i$ of size $\#\mathrm{N}(k)$, where $\#\mathrm{N}(k) \geq c \cdot \#\mathrm{NThreads}$, an array of associated error-length $(e^{(k)})_i$, an array of associated partial solutions $(\mathbf{s}^{(k)})_i$, $1 \leq i \leq \#\mathrm{N}(k)$.

1: Initialize queues Q_t, Q_e, Q_s
2: Q_t.Enqueue($\mathbf{t}$), Q_e.Enqueue(0), Q_s.Enqueue($\mathbf{0}$)
3: Let $\widetilde{\mathbf{B}} \leftarrow \mathrm{GSO}(\mathbf{B})$
4: $\#\mathrm{N}(m) \leftarrow 1$
5: $k \leftarrow m - 1$
6: **while** $(\#\mathrm{N}(k+1) < c \cdot \#\mathrm{NThreads})$ **do**
7: $\quad \#\mathrm{N}(k) \leftarrow 0$
8: $\quad$ **for** $j = 1 \ldots \#\mathrm{N}(k+1)$ **do**
9: $\quad\quad \mathbf{t} \leftarrow Q_t$.Dequeue(), $e \leftarrow Q_e$.Dequeue(), $\mathbf{s} \leftarrow Q_s$.Dequeue()
10: $\quad\quad \#\mathrm{N}(k) \leftarrow \#\mathrm{N}(k) + \lceil \sqrt{B^{(m)}(e)} / \|\widetilde{\mathbf{b}}_m\| \rceil$
11: $\quad\quad c^* \leftarrow \langle \mathbf{t}, \widetilde{\mathbf{b}}_m \rangle / \|\widetilde{\mathbf{b}}_m\|^2$
12: $\quad\quad$ **for** $i = 0 \ldots \lceil \sqrt{B^{(m)}(e)} / \|\widetilde{\mathbf{b}}_m\| \rceil - 1$ **do**
13: $\quad\quad\quad Q_t$.Enqueue($\mathbf{t} - \lceil c^* \pm i \rfloor \mathbf{b}_k$)
14: $\quad\quad\quad Q_e$.Enqueue($e + (c^* - \lceil c^* \pm i \rfloor)^2 \|\widetilde{\mathbf{b}}_k\|^2$)
15: $\quad\quad\quad Q_s$.Enqueue($\mathbf{s} + \lceil c^* \pm i \rfloor \mathbf{b}_k$)
16: $\quad k \leftarrow k - 1$
17: **return** (Q_t, Q_e, Q_s)

4 Variants of LWE

Binary secret LWE. Recent results on the BKW algorithm for LWE [12, 15] show that BKW's running time can be significantly sped up for small LWE secret vectors $\mathbf{s}$. For a binary secret, the complexity drops from fully exponential to $2^{\mathcal{O}(n/\log\log n)}$, and Kirchner and Fouque [15] report on a successful secret-recovery for $n = 128$ within 13 hours using 2^{28} LWE samples.

Lattice-based techniques in turn can also profit from the fact that the secret is small (smaller than the error). As described by Bai and Galbraith [5], one transforms a BDD instance $(\Lambda(\mathbf{A}), \mathbf{b} = \mathbf{sA} + \mathbf{e})$ with error $\mathbf{e}$ into a BDD instance

$$\left(\Lambda_q^\perp \begin{pmatrix}\mathbf{I}_m \\ \mathbf{A}\end{pmatrix}, (\mathbf{b}, \mathbf{0}^n)\right) \tag{2}$$

with error $(\mathbf{e}, \mathbf{s})$. The instance is correctly defined since

$$((\mathbf{e}, \mathbf{s}) - (\mathbf{b}, \mathbf{0}^n)) \begin{pmatrix}\mathbf{I}_m \\ \mathbf{A}\end{pmatrix} = 0 \mod q.$$

The lattice $\Lambda_q^\perp \begin{pmatrix}\mathbf{I}_m \\ \mathbf{A}\end{pmatrix} \in \mathbb{Z}^{n+m}$ is generated by the rows of $\mathbf{A}^\perp$, where

$$\mathbf{A}^\perp = \begin{pmatrix}-\mathbf{A}|\mathbf{I}_n \\ q\mathbf{I}_{n+m}\end{pmatrix}.$$

We run the BDD Enumeration of Algorithm 2 on instances defined by Eq. (2) (see Sect. 5, Tab. 1).

Binary matrix. To implement an LWE-based encryption on lightweight devices, Galbraith [9] proposed not to store the whole random matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$, but to generate the entries of a *binary* $\mathbf{A} \in \mathbb{Z}_2^{n\times m}$ via some PRNG. Galbraith's ciphertexts are of the form $(C_1, C_2) = (\mathbf{Au}, \langle \mathbf{u}, \mathbf{b}\rangle + m\lceil q/2\rceil \mod q)$ for a message $m \in \{0, 1\}$, some random $\mathbf{u} \in \{0, 1\}^m$ and a modulus $q \in \mathbb{Z}$. The task is to recover $\mathbf{u}$ given $(\mathbf{A}, \mathbf{Au})$.

Let us describe a simple lattice-attack on the instance $(\mathbf{A}, \mathbf{Au})$. Notice that $C_1 = \mathbf{Au}$ holds over $\mathbb{Z}$ and, hence, over $\mathbb{Z}_q$ for large enough modulus q since we expect to have $\mathbf{Au} \approx m/4$. First, we find any solution $\mathbf{w}$ for $\mathbf{Aw} = C_1 \mod q$. Note that

$$(\mathbf{w} - \mathbf{u}) \in \ker(\mathbf{A}).$$

So we have a BDD instance $(\Lambda_q^\perp(\mathbf{A}), \mathbf{w})$, with $\mathbf{u}$ as the error-vector of length $m/2$ and a lattice with $\det(\Lambda_q^\perp(\mathbf{A})) = q^n$. Since we can freely choose q to be as large as we want, we can guarantee that $\lambda_1(\Lambda_q^\perp(\mathbf{A})) \gg m/2$. Such an instance can be solved by first running β-BKZ for some small constant β and then Babai's CVP algorithm.

As a challenge, Galbraith proposes a parameter-set $(n = 256, m = 400)$ and estimates that computing $\mathbf{u}$ from $\mathbf{Au}$ should take around one day. We solve this instance using NTL's BKZ implementation with $\beta = 4$ and $q = 500009$ in 4.5 hours (see Tab. 1).

5 Implementation results

We implemented our BDD enumeration step with Lindner-Peikert's Nearest Planes and Liu-Nguyen's Linear Length Pruning. All programs are written in C++ and we used C++11 STL for implementing the threading. Our tests were performed on the Ruhr-University's "Crypto Crunching Cluster" (C3) which consists of one master node to schedule jobs and four computing nodes. Each computing node has four AMD Bulldozer Opteron 6276 CPUs, and thus 64 cores, running at 2.3 GHz and 256 GByte of RAM. The results of our experiments are presented in Table 1.

Our experiments are run on

1. standard LWE parameters (top part of Tab. 1),
2. LWE with binary- and ternary-error (middle part),
3. binary secret LWE,
4. the space-efficient proposal of Galbraith (bottom).

Let us describe the results of our experiments in more details.

1. For the standard LWE case and Gaussian error, the dimensions we successfully attacked in several hours are within the interval $n \in [70, 100]$. We achieve an almost perfect speed-up – the gained factor in the running times is roughly equal to the number of processors (#NThreads). This shows that our distribution of processors balances the workload. The largest successfully decoded parameters are $(n = 100, s = 4)$. For comparison, the instance $(n = 192, s = 9)$ achieves 2^{87}-security level as estimated in [16].
2. Not surprisingly, once the error is changed from Gaussian to binary or ternary, the decoding attack performs better, but balancing the BKZ-reduction and BDD steps becomes more subtle, since a smaller error is more favourable for the decoding. Hence, such an instance can be attacked with a less reduced basis than a similar LWE instance with Gaussian noise. To balance the reduction and enumeration steps, we first choose a smaller block-size β for the reduction and, second, choose fewer than $2n$ samples. Our choice for m additionally lowers the running time of BKZ-reduction, while it still guarantees successful decoding. The maximal dimension achieved in this regime is $n = 130$. Binary and ternary errors are especially interesting for cryptanalysis of NTRU [13] and for special variants of LWE considered by Buchmann et al. [7] and Micciancio and Peikert [19].
3. For binary secret we are able to attack dimensions $n \in [100, 140]$. In contrast to the BKW attack of Kirchner and Fouque [15], we choose as few samples as possible to aid the reduction step (while keeping a unique solution). More concretely, for $n = 130$, we used only $m = 150$ samples, as opposed to $m = 2^{28}$ samples required in the BKW attack. Our attack takes only 7.6h, which is faster than the reported 13h in [15]. Moreover, we are able to attack dimension $n = 140$ for which we benefit again from parallelization.
4. For the space-efficient binary-matrix case of [9], we choose $q = 50009$ and solve the instance $(n = 256, m = 400)$ in 4.5h with $\beta = 4$ and Babai's CVP algorithm.

Table 1: Running-times of the BDD-decoding attack on LWE . The superscript B indicates that Babai's Nearest Plane Algorithm already solved the instance. Uniform binary and ternary error distributions are denoted by $s = \{0, 1\}$ and $s = \{-1, 0, 1\}$.

LWE-parameters				BKZ-reduction		Lindner-Peikert		Length Pruning	
n	q	s	m	β	T	#NThreads	T	#NThreads	T
70	4093	6	140	15	41min	—	—	1	14h
70	4093	6	140	15	41min	10	9.6h	10	1.5h
70	4093	6	140	20	65min	—	—	1	44min
70	4093	6	140	20	65min	—	—	10	5min
80	4093	5	150	25	4.3h	1	55h	1	13h
80	4093	5	150	25	4.3h	10	45h	10	1.5h
80	4093	5	150	25	4.3h	20	12h	20	50min
90	4093	4	170	22	11.3h	—	—	1	35.5h
90	4093	4	170	22	11.3h	20	49.5h	10	3.6h
100	4093	4	200	20	6.9h	24	3.0h	24	2.7h
Binary error [7, 19]									
100	4093	$\{0, 1\}$	140	4	1h	—	—	1	2.5min
110	4093	$\{0, 1\}$	160	5	1.4h	1	5.7h	1	3.1h
120	4093	$\{0, 1\}$	170	10	1.6h	—	—	1	27min
130	4093	$\{0, 1\}$	190	18	4.5h	—	—	1	13.5h
130	4093	$\{0, 1\}$	190	18	4.5h	—	—	10	1.7h
130	4093	$\{0, 1\}$	200	10	3.1h	—	—	1	1h
130	4093	$\{0, 1\}$	260	20	30.3h	16	1.8min	16	75s
Ternary error [13]									
100	4093	$\{-1, 0, 1\}$	140	10	50min	1	5.6h	1	9min
100	4093	$\{-1, 0, 1\}$	200	7	2.2h	1	17s	1	17s
110	4093	$\{-1, 0, 1\}$	180	7	1.5h	1	10min	1	10min
120	4093	$\{-1, 0, 1\}$	240	7	4.5h	20	13h	20	14min
128	4093	$\{-1, 0, 1\}$	256	20	28.8h	30	92s	30	43s
Binary secret									
120	16411	7	150	10	2.3h	—	—	1	2h
130	16411	5	150	15	6.6h	—	—	1	1h
140	16411	5	170	15	12h	—	—	1	16.3h
140	16411	5	170	15	12h	—	—	10	1.7h
Binary matrix SIS [9]									
256	500009	–	400	4	4.5h	1	2min^B	—	—
280	500009	–	440	4	6.5h	1	3min^B	—	—

All our experiments confirm that Linear Length Pruning works much more efficient than Lindner-Peikert Decoding for most of the considered variants of LWE. Another observation is that lowering the number of samples significantly

speeds up the reduction in practice and slows down the decoding step. Since the latter can be parallelized, a proper choice of the number of samples leads to a better trade-off between the reduction and enumeration.

6 Acknowledgements

We thank Gottfried Herold and the anonymous reviews for their helpful feedback and valuable suggestions. Elena Kirshanova and Friedrich Wiemer were supported by UbiCrypt, the research training group 1817/1 funded by the DFG.

References

[1] M. R. Albrecht, C. Cid, J. Faugère, R. Fitzpatrick, and L. Perret. "On the complexity of the BKW algorithm on LWE". In: *Des. Codes Cryptography* 74.2 (2015), pp. 325–354.

[2] M. R. Albrecht, R. Player, and S. Scott. "On the concrete hardness of Learning with Errors". In: *J. Mathematical Cryptology* 9.3 (2015), pp. 169–203. URL: https://eprint.iacr.org/2015/046.

[3] S. Arora and R. Ge. "New Algorithms for Learning in Presence of Errors". In: *Automata, Languages and Programming - ICALP 2011*. Vol. 6755. Lecture Notes in Computer Science. Springer, 2011, pp. 403–415.

[4] L. Babai. "On Lovász' Lattice Reduction and the Nearest Lattice Point Problem (Shortened Version)". In: *STACS 1985*. Vol. 182. Lecture Notes in Computer Science. Springer, 1985, pp. 13–20.

[5] S. Bai and S. D. Galbraith. "Lattice Decoding Attacks on Binary LWE". In: *Information Security and Privacy - ACISP 2014*. Vol. 8544. Lecture Notes in Computer Science. Springer, 2014, pp. 322–337. URL: https://eprint.iacr.org/2013/839.

[6] J. A. Buchmann, D. Cabarcas, F. Göpfert, A. Hülsing, and P. Weiden. "Discrete Ziggurat: A Time-Memory Trade-Off for Sampling from a Gaussian Distribution over the Integers". In: *Selected Areas in Cryptography - SAC 2013*. Vol. 8282. Lecture Notes in Computer Science. Springer, 2013, pp. 402–417. URL: https://eprint.iacr.org/2013/510.

[7] J. Buchmann, F. Gpfert, R. Player, and T. Wunderer. "On the Hardness of LWE with Binary Error: Revisiting the Hybrid Lattice-Reduction and Meet-in-the-Middle Attack". In: *Progress in Cryptology - AFRICACRYPT 2016, To appear*. Lecture Notes in Computer Science. Springer, 2016. URL: https://eprint.iacr.org/2016/089.

[8] Y. Chen and P. Q. Nguyen. "BKZ 2.0: Better Lattice Security Estimates". In: *Advances in Cryptology - ASIACRYPT 2011*. Vol. 7073. Lecture Notes in Computer Science. Springer, 2011, pp. 1–20. URL: https://www.iacr.org/archive/asiacrypt2011/70730001/70730001.pdf.

[9] S. D. Galbraith. "Space-efficient variants of cryptosystems based on learning with errors". URL: https://www.math.auckland.ac.nz/~sgal018/compact-LWE.pdf.

[10] N. Gama, P. Q. Nguyen, and O. Regev. "Lattice Enumeration Using Extreme Pruning". In: *Advances in Cryptology - EUROCRYPT 2010*. Vol. 6110. Lecture Notes in Computer Science. Springer, 2010, pp. 257–278. URL: https://www.iacr.org/archive/eurocrypt2010/66320257/66320257.pdf.

[11] C. Gentry, C. Peikert, and V. Vaikuntanathan. "Trapdoors for hard lattices and new cryptographic constructions". In: *STOC 2008*. Ed. by C. Dwork. ACM, 2008, pp. 197–206.

[12] Q. Guo, T. Johansson, and P. Stankovski. "Coded-BKW: Solving LWE Using Lattice Codes". In: *Advances in Cryptology - CRYPTO 2015*. Vol. 9215. Lecture Notes in Computer Science. Springer, 2015, pp. 23–42.

[13] J. Hoffstein, J. Pipher, and J. H. Silverman. "NTRU: A ring-based public key cryptosystem". In: *Algorithmic Number Theory: Third International Symposiun*. 1998, pp. 267–288.

[14] R. Kannan. "Minkowski's Convex Body Theorem and Integer Programming". In: *Mathematics of Operations Research* 12.3 (1987), pp. 415–440. ISSN: 0364765X, 15265471.

[15] P. Kirchner and P. Fouque. "An Improved BKW Algorithm for LWE with Applications to Cryptography and Lattices". In: *Advances in Cryptology - CRYPTO 2015*. Vol. 9215. Lecture Notes in Computer Science. Springer, 2015, pp. 43–62. URL: https://eprint.iacr.org/2015/552.

[16] R. Lindner and C. Peikert. "Better Key Sizes (and Attacks) for LWE-Based Encryption". In: *Topics in Cryptology - CT-RSA 2011*. Vol. 6558. Lecture Notes in Computer Science. Springer, 2011, pp. 319–339. URL: https://eprint.iacr.org/2010/613.

[17] M. Liu and P. Q. Nguyen. "Solving BDD by Enumeration: An Update". In: *Topics in Cryptology - CT-RSA 2013*. Vol. 7779. Lecture Notes in Computer Science. Springer, 2013, pp. 293–309.

[18] L. Luzzi, D. Stehlé, and C. Ling. "Decoding by Embedding: Correct Decoding Radius and DMT Optimality". In: *IEEE Transactions on Information Theory* 59.5 (2013), pp. 2960–2973.

[19] D. Micciancio and C. Peikert. "Hardness of SIS and LWE with Small Parameters". In: *Advances in Cryptology - CRYPTO 2013*. Lecture Notes in Computer Science. 2013, pp. 21–39. URL: https://eprint.iacr.org/2013/069.

[20] O. Regev. "On Lattices, Learning with Errors, Random Linear Codes, and Cryptography". In: *STOC 2005*. ACM, 2005, pp. 84–93.

[21] V. Shoup. *Number Theory Library 9.6.2 (NTL) for C++*. URL: http://www.shoup.net/ntl/.