Solving BDD by Enumeration: An Update

Mingjie Liu and Phong Q. Nguyen

[1] Beijing International Center for Mathematical Research, Peking University and Tsinghua University, Institute for Advanced Study, China.
[2] INRIA, France and Tsinghua University, Institute for Advanced Study, China.
http://www.di.ens.fr/~pnguyen/

Abstract. Bounded Distance Decoding (BDD) is a basic lattice problem used in cryptanalysis: the security of most lattice-based encryption schemes relies on the hardness of some BDD, such as LWE. We study how to solve BDD using a classical method for finding shortest vectors in lattices: enumeration with pruning speedup, such as Gama-Nguyen-Regev extreme pruning from EUROCRYPT '10. We obtain significant improvements upon Lindner-Peikert's Search-LWE algorithm (from CT-RSA '11), and update experimental cryptanalytic results, such as attacks on DSA with partially known nonces and GGH encryption challenges. Our work shows that any security estimate of BDD-based cryptosystems must take into account enumeration attacks, and that BDD enumeration can be practical even in high dimension like 350.

1 Introduction

There is growing interest in cryptography based on hard lattice problems (see the survey [11]). The field started with the seminal work of Ajtai [1] back in 1996, and recently got a second wind with Gentry's breakthrough work [7] on fully-homomorphic encryption. It offers asymptotical efficiency, potential resistance to quantum computers and new functionalities. Most of the provably-secure lattice-based constructions are based on either of the following two average-case problems:

- the Small Integer Solutions (SIS) problem proposed by Ajtai [1], which allows to build one-way functions, collision-resistant hash functions, signature schemes and identification schemes: see [11].
- the Learning with Errors (LWE) problem introduced by Regev [17] (see the survey [18]), which allows to build public-key encryption [17,16], and more powerful primitives such as ID-based encryption [8] and even fully-homomorphic encryption [3].

Both SIS and LWE are provably as hard as certain worst-case lattice problems (see [1,11] for SIS, and [17,16] for LWE), which allows to design many cryptographic schemes with security related to the hardness of lattice problems, without actually dealing explicitly with lattices.

Due to the importance of LWE, it is very important to know what are the best attacks on LWE, especially if one is interested in selecting concrete parameters. And this issue is independent of LWE (quantum or not) reductions [17,16] to lattice problems. At CT-RSA '11, Lindner and Peikert [10] generalized Babai's Nearest Plane algorithm [2] to solve Bounded Distance Decoding (BDD), and claimed that this was the best attack known on Search-LWE. Given a lattice and a target vector unusually close to the lattice, BDD asks to find the closest lattice vector to the target: this basic lattice problem has many applications in cryptanalysis (see [14]), and LWE is simply a particular case of BDD. Despite its importance, it is not obvious at the moment what is the best algorithm for solving BDD in practice: several parameters impact the answer, *e.g.* the dimension, the size and shape of the error error. Until now, the largest BDD cryptanalytical instances ever solved in practice were tackled using the so-called embedding method that heuristically reduces BDD to the unique-shortest vector problem (USVP) (see *e.g.* [13]). And in the past few years, there has been significant process in practical lattice reduction [6,4].

Our Results. We present lattice attacks on Search-LWE which are significantly better than the Lindner-Peikert attack [10]: in practice, the speedup can be as big as 2^{32} for certain parameters considered in [10] (see Table 1). First, we revisit the Lindner-Peikert BDD algorithm (which turns out to be essentially Schnorr's random sampling [19]) by rephrasing it in the pruned-enumeration framework of Gama-Nguyen-Regev (GNR) [6]. This allows us to present a simple randomized variant which already performs much better than the original LP algorithm [10] in the case of LWE, independently of the efficiency of lattice reduction: our randomization is similar to GNR's extreme pruning [6], *i.e.* we use several random reduced bases.

Next, we consider GNR pruned-enumeration algorithms [6] to solve BDD: this provides even better attacks on Search-LWE, and seems to be the method of choice in practice for the general BDD case. We illustrate this point by reporting improved experimental results for attacks on DSA with partially known nonces [15] and the solution of GGH encryption challenges [13,9]. Though enumeration is a classical method to solve BDD, it was unknown how efficient in practice was GNR pruned-enumeration in BDD applications. In the DSA case, pruned enumeration can recover the DSA secret key in a few hours, given each 2 least significant bits of the nonces corresponding to 100 DSA signatures: previously, the best lattice experiment [15] required 3 bits. For GGH encryption challenges, we provide the first partial secret-key recovery in dimensions 200-300: this is a proof-of-concept, and the rest of the secret key could easily be recovered by simply repeating our experiments a few times. And we re-solved the 350-dimensional message-recovery challenge using much weaker lattice reduction: in this case, enumeration is clearly preferable to the embedding method, even in very high dimension like 350.

Our work shows that any security estimate on BDD must take into account enumeration attacks, and that the Lindner-Peikert BDD algorithm [10]

does not seem to offer any practical advantage over GNR pruning [6], despite having appeared later.

Road Map. The paper is organized as follows. In Section 2, we provide background. In Section 3, we revisit the Lindner-Peikert algorithm and present our randomized variant. Finally, in Section 4, we consider GNR pruned enumeration [6] to solve BDD, and apply it to LWE, GGH encryption challenges and attacks on DSA with partially known nonces. In Appendix, we provide pseudo-code for BDD enumeration.

2 Background

We use row notation for vectors. We denote by $\|\mathbf{v}\|$ the Euclidean norm of a vector $\mathbf{v}$, and by $\text{Ball}_m(R)$ the m-dimensional closed ball of radius R, whose volume is $V_m(R) \sim (\sqrt{\frac{2\pi e}{m}})^m R^m$. We use S^{m-1} to denote the m-dimensional unit sphere. The *fundamental parallelepiped* $\mathcal{P}_{1/2}(B)$ of a matrix $B = (\mathbf{b}_1, \ldots, \mathbf{b}_m)$ is $\{\sum_{i=1}^m x_i \mathbf{b}_i : -\frac{1}{2} \leq x_i < \frac{1}{2}\}$. The *volume* $\text{vol}(L)$ of a lattice L is the m-dimensional volume of $\mathcal{P}_{1/2}(B)$ for any basis B of L.

Orthogonalization. The Gram-Schmidt orthogonalization of B is denoted by $B^* = (\mathbf{b}_1^*, \mathbf{b}_2^*, \ldots, \mathbf{b}_m^*)$, where $\mathbf{b}_i^* = \pi_i(\mathbf{b}_i)$, with π_i being the orthogonal projection over $(\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_{i-1})^\perp$. Thus, $\pi_i(L)$ is an $(m+1-i)-$dimensional lattice generated by the basis $(\pi_i(\mathbf{b}_i), \ldots, \pi_i(\mathbf{b}_m))$ with volume $\text{vol}(\pi_i(L)) = \prod_{j=i}^m \left\|\mathbf{b}_j^*\right\|$.

Gaussian Heuristic. The Gaussian Heuristic provides a heuristic estimate on the number of lattice points inside a set as follows: Given a "nice" lattice L and a "nice" set S, the number of points in $S \cap L$ is heuristically $\approx \text{vol}(S)/\text{vol}(L)$.

Lattice reduction. Lattice reduction algorithms aim at finding bases with short and nearly orthogonal vectors. Their output quality is usually measured by the Hermite factor $\|\mathbf{b}_1\|/(\text{vol}(L))^{\frac{1}{m}}$, where $\mathbf{b}_1$ is the first vector of the output basis. The experiments of Gama and Nguyen [5] show that the Hermite factor of the best algorithms known is exponential δ^m in the dimension m in practice, and the recent work of Chen and Nguyen [4] provides a correspondence between the exponentiation base δ (the root Hermite factor) and the running time of the best state-of-the-art implementation (BKZ 2.0). This is related to the geometric series assumption (GSA) proposed by Schnorr [20]: for fixed parameters, the norms of the Gram-Schmidt vectors $\mathbf{b}_i^*$ decrease roughly geometrically with i, say $\left\|\mathbf{b}_i^*\right\| / \left\|\mathbf{b}_{i+1}^*\right\| \approx q$, in which case the root Hermite factor δ is $\approx \sqrt{q}$. In [10], Lindner and Peikert used different running-time estimates of lattice reduction than [4]:

– One is a numerical extrapolation based on their experiments with the BKZ implementation of the NTL library. However, NTL's implementation of BKZ dates back from 1997 and does not take into account recent progress, such as extreme pruning [6]. The state-of-the-art implementation of BKZ developed by Chen and Nguyen [4] achieves several exponential speedups compared to NTL's implementation.
– The other one is used in the tables of [10]. It is simply a conservative lower bound of the first one. They divide the running time by some arbitrary (large) constant, and change (conservatively) the slope of the curve.

We believe it is preferable to use the Chen-Nguyen estimates [4], but any comparison with the LWE algorithm of [10] must also take into account the lattice reduction estimates of [10] for completeness.

Discrete Gaussian Distribution. Let $s > 0$ be real. The discrete (centered) Gaussian distribution over L has density $D_{L,s}(\mathbf{x}) = \frac{\rho_s(\mathbf{x})}{\rho_s(L)}$ where $\mathbf{x} \in L$, $\rho_s(\mathbf{x}) = e^{-\pi\|\mathbf{x}/s\|^2}$ and $\rho_s(L) = \sum_{\mathbf{y}\in L}\rho_s(\mathbf{y})$. Over $\mathbb{Z}^n$, most of the mass is within the ball of radius $O(s\sqrt{n})$.

Bounded Distance Decoding (BDD). Given L and a target $\mathbf{t}$ very "close" to L, BDD asks to find $\mathbf{u} \in L$ minimizing $\|\mathbf{u} - \mathbf{t}\|$. There are many ways to formalize what is meant by very "close": here, we assume that there exists a unique $\mathbf{u} \in L$ such that $\|\mathbf{u} - \mathbf{t}\| \leq \gamma \mathrm{vol}(L)^{1/m}$ for some small given $\gamma > 0$: the smaller γ, the easier BDD. The vector $\mathbf{u} - \mathbf{t}$ is called the BDD error.

The Learning with Errors problem (LWE). The input of LWE is a pair $(A, \mathbf{t} = \mathbf{s}A + \mathbf{e})$ where $A \in \mathbb{Z}_q^{n\times m}$ and $\mathbf{s} \in \mathbb{Z}_q^n$ are chosen uniformly at random, $\mathbf{e} \in \mathbb{Z}_q^m$ is chosen according to some public distribution χ depending on a parameter $\alpha \in (0, 1)$. In the original LWE article [17], χ is the integral rounding of a continuous Gaussian distribution: namely, χ is the distribution of the random variable $\lfloor qX \rfloor \mod q$, where X is a (continuous) normal variable with mean 0 and standard deviation $\alpha/\sqrt{2\pi}$ reduced modulo 1. The Lindner-Peikert article [10] uses instead the discrete Gaussian distribution $\chi = D_{\mathbb{Z}^m,\alpha q}$, and mention (without proof) that LWE hardness results also hold for this LWE variant.

Given $(A, \mathbf{t})$, Search-LWE asks to recover $\mathbf{s}$, while Decision-LWE asks to distinguish $(A, \mathbf{t})$ from a uniformly random $(A, \mathbf{t})$. Regev [17] proved that if $\alpha q \geq 2\sqrt{n}$, Search-LWE is at least as hard as quantumly approximating (Decision)-SVP or SIVP to within $\tilde{O}(n/\alpha)$ in the worst case for dimension n. Under suitable constraints on LWE parameters, Search-LWE can be reduced to Decision-LWE [17,16].

Search-LWE can be viewed as a BDD-instance in the m-dimensional lattice $\Lambda_q(A) = \{\mathbf{y} \in \mathbb{Z}^m : \mathbf{y} = \mathbf{s}A \mod q \text{ for } \mathbf{s} \in \mathbb{Z}_q^n\}$, and with BDD error $\mathbf{e}$ (provided that α is sufficiently small). With overwhelming probability over A, $\mathrm{vol}(\Lambda_q(A)) = q^{m-n}$. This standard lattice attack can be improved using

the so-called sublattice attack (see [5,11]), which replaces m by some $m' \leq m$ to optimize the use of current reduction algorithms. Given a root Hermite factor δ, we expect to achieve $\left\|\mathbf{b}^*_{m'}\right\| \approx \delta^{-m'} q^{\frac{m'-n}{m'}}$, where $m' = \sqrt{n \log q / \log \delta}$ maximizes the norm of $\mathbf{b}^*_{m'}$. As long as $m' < m$, one applies a lattice attack to the sublattice with dimension m', otherwise one uses the full lattice.

Beta distribution. The density function of the Beta distribution of parameters $a, b > 0$ is $x^{a-1}(1-x)^{b-1}/B(a,b)$, where $B(a,b)$ is the beta function $\frac{\Gamma(a)\Gamma(b)}{\Gamma(a+b)}$. The corresponding cumulative distribution function is the regularized incomplete beta function $I_x(a,b) = \frac{1}{B(a,b)} \int_0^x u^{a-1}(1-u)^{b-1} du, x \in [0,1]$. If $(u_1, u_2, \ldots, u_m)$ is chosen uniformly at random from the unit sphere S^{m-1}, then $\sum_{i=1}^k u_i^2$ has distribution $Beta(k/2, (m-k)/2)$.

3 Lindner-Peikert's NearestPlanes Algorithm Revisited

In this section, we revisit the Lindner-Peikert NearestPlanes algorithm [10] (NP), which is a simple variant of Babai's algorithm [2] to solve BDD, and which turns out to be similar to Schnorr's random sampling [19]. We establish a connection with Gama-Nguyen-Regev's pruned enumeration [6], which allows us to randomize and generalize the NP algorithm.

3.1 Babai's Nearest Plane Algorithm

Since LWE $(\mathbf{A}, \mathbf{t} = \mathbf{sA} + \mathbf{e})$ is a BDD instance for the lattice $\Lambda_q(A)$, the simplest method is Babai's (deterministic polynomial-time) Nearest Plane algorithm, see Alg. 1.

Algorithm 1 Babai's Nearest Plane algorithm [2]

Input: A basis $B = (\mathbf{b}_1, \ldots, \mathbf{b}_m) \in \mathbb{Q}^m$ of a lattice L and a target point $\mathbf{t} \in \mathbb{Q}^m$.
Output: $\mathbf{v} \in L$ such that $\mathbf{v} - \mathbf{t} \in \mathcal{P}_{1/2}(B^*)$.

$\mathbf{v} \leftarrow \mathbf{0}$
For $i \leftarrow m, ..., 1$
  Compute the integer c closest to $\langle \mathbf{b}_i^*, \mathbf{t} \rangle / \langle \mathbf{b}_i^*, \mathbf{b}_i^* \rangle$
  $\mathbf{t} \leftarrow \mathbf{t} - c\mathbf{b}_i$,$\mathbf{v} \leftarrow \mathbf{v} + c\mathbf{b_i}$
Return $\mathbf{v}$

Babai's algorithm outputs a lattice vector $\mathbf{v}$ relatively close to the input target vector $\mathbf{t}$. More precisely, $\mathbf{v}$ is the unique lattice vector such that $\mathbf{v} - \mathbf{t} \in \mathcal{P}_{1/2}(B^*)$. If the input is a BDD instance such that the closest lattice point to $\mathbf{t}$ is $\mathbf{v} \in L$, then Babai's algorithm solves BDD if and only if $\mathbf{v} - \mathbf{t} \in \mathcal{P}_{1/2}(B^*)$, *i.e.*

$$\forall i \in \{1, \ldots, m\} \ \ |\langle \mathbf{e}, \mathbf{b}_i^* \rangle| < \langle \mathbf{b}_i^*, \mathbf{b}_i^* \rangle / 2. \tag{1}$$

In the LWE case, we can rigorously define a success probability, even though Babai's algorithm is deterministic: the probability of $\mathbf{v} - \mathbf{t} \in \mathcal{P}_{1/2}(B^*)$ is with respect to LWE parameter generation, *i.e.* the generation of $\mathbf{t}$. Unfortunately, one does not know how to compute this probability efficiently. Instead, Lindner and Peikert [10] compute the success probability in an idealized model (which we call CLWE for "continuous" LWE) where the LWE error distribution χ is replaced by a continuous Gaussian distribution with mean 0 and standard deviation $\alpha q/\sqrt{2\pi}$, and claim (without proof) that the actual probability is very close.

By definition of CLWE, the distribution of the error $\mathbf{e}$ is spherical, which implies because B^* has orthogonal rows:

$$\Pr[\mathbf{e} \in \mathcal{P}_{1/2}(B^*)] = \prod_{i=1}^{m} Pr[|\langle \mathbf{e}, \mathbf{b}_i^* \rangle| < \langle \mathbf{b}_i^*, \mathbf{b}_i^* \rangle / 2] = \prod_{i=1}^{m} \mathrm{erf}\left(\frac{\|\mathbf{b}_i^*\| \sqrt{\pi}}{2s} \right),$$

where $\mathrm{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-y^2} dy$. Typically, $\mathbf{b}_m^*$ is exponentially shorter than $\mathbf{b}_1$, which makes the probability of $\mathbf{e} \in \mathcal{P}_{1/2}(B^*)$ very small.

3.2 The NearestPlanes Algorithm

Lindner and Peikert [10] presented a simple generalization of Babai's nearest plane algorithm, by adding some exhaustive search to increase the success probability, at the expense of the running time. Instead of choosing the closest plane in every i-th level, the NearestPlanes algorithm (Alg. 2) enumerates d_i distinct planes.

Algorithm 2 NearestPlanes Algorithm [10]

Input: A lattice basis $B = (\mathbf{b}_1, \ldots, \mathbf{b}_m)$, a vector $\mathbf{d} = (d_1, d_2, \ldots, d_m) \in \mathbb{Z}^m$, a target point $\mathbf{t} \in \mathbb{Q}^m$.

Output: A set of $\prod_{i=1}^{m} d_i$ distinct lattice vectors in $\mathcal{L}(\mathbf{B})$ close to $\mathbf{t}$.

1: **if** $m = 0$ **then**
2: Return $\mathbf{0}$
3: **else**
4: Compute the d_m integers $c_1, c_2, \ldots, c_{d_m} \in \mathbb{Z}$ closest to $\langle \mathbf{b}_\mathbf{m}^*, \mathbf{t} \rangle / \langle \mathbf{b}_\mathbf{m}^*, \mathbf{b}_\mathbf{m}^* \rangle$
5: Return $\bigcup_{i \in [d_m]} (c_i \mathbf{b}_m + NearestPlanes(\{\mathbf{b}_1, \ldots, \mathbf{b}_{\mathbf{m-1}}, (d_1, \ldots, d_{m-1}), \mathbf{t} - c_i \mathbf{b}_m\})$
6: **end if**

Compared to Babai's nearest plane algorithm, the NearestPlanes algorithm is also deterministic, its running time is essentially multiplied by $\prod d_i$, and its CLWE success probability (*i.e.* under the assumption that the LWE distribution is continuous) increases to: $\prod_{i=1}^{m} \mathrm{erf}\left(\frac{d_i \|\mathbf{b}_i^*\| \sqrt{\pi}}{2s} \right)$. In fact, the algorithm

succeeds if and only if $\mathbf{e} \in \mathcal{P}_{1/2}(\mathrm{diag}(\mathbf{d}) \cdot B^*)$, where $\mathrm{diag}(\mathbf{d})$ is the $m \times m$ diagonal matrix formed by the d_i's.

Lindner and Peikert [10] briefly compared their algorithm to the distinguisher of Micciancio and Regev [11]. Their data suggests that their algorithm is better for most parameters and success probability, with larger improvements in the high-advantage regime.

3.3 Connection with Schnorr's Random Sampling

We note that the NP algorithm is very similar to Schnorr's random sampling [19] from STACS '03. Schnorr's method aims at finding short vectors, but it can easily be adapted to BDD: in the BDD setting, there is an integer parameter $u \in \{1, \ldots, m\}$, and one computes all lattice vectors $\mathbf{v} \in L$ such that $\mathbf{v} - \mathbf{t} \in \mathcal{P}(\Delta B^*))$ where Δ is the $m \times m$ diagonal matrix formed by $m - u$ coefficients equal to 1, followed by u coefficients equal to 2. In other words, Schnorr's method corresponds to the particular case of NP where $(d_1, \ldots, d_m) = (1, 1, \ldots, 1, 2, 2, \ldots, 2)$, the number of 2's being exactly u. However, Schnorr's analysis is very different from [10] and uses a more debatable model: it assumes that the 2^u vectors $\mathbf{v} - \mathbf{t}$ are uniformly distributed over $\mathcal{P}(\Delta B^*))$, which cannot actually hold.

3.4 Connection with Lattice Enumeration

We note that both Babai's algorithm and the NP algorithm can be viewed as a pruned enumeration but with a different kind of pruning rule than Gama *et al.* [6]. Let us recall what is lattice enumeration. Given a target $\mathbf{t} \in \mathbb{Q}^m$, a basis $B = (\mathbf{b}_1, \ldots, \mathbf{b}_m)$ of a lattice L and a BDD radius R, an enumeration algorithm enumerates all lattice vectors $\mathbf{v} \in L$ such that $\|\mathbf{v} - \mathbf{t}\| \leq R$, and selects the closest one to solve BDD. This is done by searching over a huge tree defined as follows: for each level $k \in \{0, \ldots, m\}$ where $k = 0$ corresponds to the root and $k = m$ corresponds to the leaves, the tree nodes of level k are all vectors $\mathbf{v} \in L$ such that $\|\pi_{m+1-k}(\mathbf{t} - \mathbf{v})\| \leq R$. This means that the number of nodes at level k is exactly the number of points in the projected lattice $\pi_{m+1-k}(L)$ which are within distance R of $\pi_{m+1-k}(\mathbf{t})$, which can be heuristically estimated as $H_k = V_k(R)/\mathrm{vol}(\pi_{m+1-k}(L))$ by the Gaussian Heuristic: such estimates seem to be fairly accurate in practice [6].

Pruned enumeration was first proposed by Schnorr and Euchner [21] to cut down some branches in the enumeration tree to decrease the time complexity, at the cost of potentially missing the solution vector. It was hoped that the overall cost (taking into account failure probability) would decrease. An algorithmic description of pruned enumeration for BDD is given in Appendix A. The first rigorous analysis of pruned enumeration was only recently given by Gama, Nguyen and Regev [6], where a framework generalizing [21] was proposed. For every tree level k, they used a variable enumeration radius $\|\pi_{m+1-k}(\mathbf{v} - \mathbf{t})\| \leq R_k$ such that $R_1 \leq R_2 \leq \cdots \leq R_m$. This means that the pruned tree is a subset of the enumeration tree.

Babai's algorithm and the NP algorithm also only consider a subset of the enumeration tree, but it is a different subset than [6]. More precisely, Babai's algorithm looks at a single branch of the enumeration tree with radius $R = \frac{1}{2}\sqrt{\sum_{i=1}^{m} \left\|\mathbf{b}_i^*\right\|^2}$. Since the NP algorithm enumerates all lattice points inside the (orthogonal) parallelepiped $\mathcal{P}_{1/2}(\mathrm{diag}(\mathbf{d}) \cdot B^*)$ centered at $\mathbf{t}$, it actually considers the radius $R = \frac{1}{2}\sqrt{\sum_{i=1}^{m} d_i^2 \left\|\mathbf{b}_i^*\right\|^2}$. Then, among all the nodes $\mathbf{v} \in L$ of the enumeration tree at level k, it only considers those such that $|\xi_i(\mathbf{v}-\mathbf{t})| \leq d_i\|\mathbf{b}_i^*\|/2$ for all $i \geq m+1-k$, where $\xi_i(\mathbf{x})$ denotes the i-th coordinate of $\mathbf{x}$ in the normalized Gram-Schmidt basis $(\mathbf{b}_1^*/\left\|\mathbf{b}_1^*\right\|, \cdots, \mathbf{b}_m^*/\left\|\mathbf{b}_m^*\right\|)$, *i.e.* $\xi_i(\mathbf{x}) = \langle \mathbf{x}, \mathbf{b}_i^* \rangle / \|\mathbf{b}_i^*\|$.

In other words, GNR pruning [6] only keeps the nodes with bounded projections $\|\pi_{m+1-k}(\mathbf{v}-\mathbf{t})\| \leq R_k$, whereas NP [10] only keeps the nodes with bounded coordinates $|\xi_i(\mathbf{v}-\mathbf{t})|$. In some sense, NP can be viewed as a secondary pruning of GNR: if the coordinates are all bounded, then so are the projections, which means that the NP tree is a subset of some GNR tree.

3.5 Randomizing the NearestPlanes Algorithm

This connection of NP with enumeration allows us to revisit and improve NP. First, NP can be generalized by selecting arbitrary bounds on coordinates, namely $|\xi_i(\mathbf{v}-\mathbf{t})| \leq R_i$ instead of $|\xi_i(\mathbf{v}-\mathbf{t})| \leq d_i\|\mathbf{b}_i^*\|/2$, where the bounds $R_1, \ldots, R_m$ are parameters which are not necessarily multiples of the $\|\mathbf{b}_i^*\|/2$'s.

Second, we perform a randomization similar to the one of extreme pruning [6] compared to basic pruning. More precisely, we randomize the algorithm by repeating several times the basic algorithm with different randomized reduced bases. If we use ℓ different bases, then the running time is roughly multiplied by ℓ (depending on the exact reduction time compared to the NP exhaustive search), but the success probability is also heuristically multiplied by ℓ. This gives rise to Alg. 3, and allows more optimization. In particular, the

Algorithm 3 Our Randomized NearestPlanes Algorithm

Input: A lattice basis B, a vector $\mathbf{d} = (d_1, d_2, \ldots, d_m)$, a target point $\mathbf{t} \in \mathbb{Q}^m$, and a number ℓ of iterations.

Output: A set of candidate lattice vectors in $\mathcal{L}(\mathbf{B})$ close to $\mathbf{t}$.

1: Repeat ℓ times
2: Randomize the input basis, and apply lattice reduction to obtain a (random) reduced basis B'
3: Run the NearestPlanes algorithm on $(B', \mathbf{d}, \mathbf{t})$

numerical data from [10, Table 3] is far from optimal, since the running times

of basis reduction and enumeration are not totally balanced. When the enumeration time is longer than the reduction time, we can decrease the total cost by decreasing the number of enumerations. We can obtain better trade-offs because the randomized algorithm has more freedom than the original one.

This can be proved by the simple analysis below. Since $\mathrm{erf}(a) < \frac{\mathrm{erf}(c \cdot a)}{c} (c < 1)$, and we use cd_i instead of d_i, the algorithm is $1/c$ times faster than the original one, while the success probability is higher than a c-fraction of the original one. Thus, by choosing different bases to repeat the algorithm several times, the total cost will actually decrease. In [10], no information was given on how to choose the d_i's. We performed an exhaustive search in the proper range to find an optimal $(d_1, d_2, \ldots, d_m)$ for the randomized algorithm, which yielded a much more efficient attack.

A numerical comparison between the NP algorithm [10] and our randomized variant is given in Table 1 below. This shows that randomization can provide significant speedups, as high as 2^{32} *e.g.* $n = 320$. Basis reduction time estimates are taken from [10], where it is estimated that $lg(T_{recd}) = \frac{1.8}{\lg(\delta)} - 110$. Columns marked with "NearestPlanes" are from [10], while columns with "Randomized-NP" correspond to our randomized variant. When we search for the best root Hermite constant, we increment by 0.0001 each time as in [10]. Because the estimates of [10] for lattice reduction are debatable, we use the estimates of Chen and Nguyen [4] to update the cost estimates of the attack in Table 2.

				NearestPlanes [10]				Randomized-NP						Log
n	q	s	Adv (log)	δ	Red [10]	Enum	Cost	δ	Red [10]	Enum	Pr	log(Nb of bases)	Cost	speedup
128	2053	6.7	0	1.0089	30.8	47	32	1.0104	10.6	27	-9.6	9.6	21.2	10.8
			-32	1.0116	< 0	13	< 0	1.0114	< 0	17	-25.5	0	< 0	
			-64	1.013	< 0	1	< 0							
192	4093	8.9	0	1.0067	76.8	87	78	1.0077	52.6	68.9	-12	12	65.6	12.4
			-32	1.0083	40.9	54	42							
			-64	1.0091	27.7	44	29							
256	4093	8.3	0	1.0052	130.5	131	132	1.006	98.6	115	-11.8	11.8	111.3	20.7
			-32	1.0063	88.7	87	90	1.0065	82.6	99	-34	2	85.6	4.4
			-64	1.0068	74.1	73	75	1.007	68.9	85	-67	3	72.9	2.1
320	4093	8	0	1.0042	187.7	163	189	1.005	140.2	156.9	-15.7	15.7	156.9	32.1
			-32	1.0052	130.6	138	132	1.0053	126	143	-34.7	2.7	129.7	2.3
			-64	1.0055	117.5	117	119	1.0056	113.4	127	-61.6	0	114.4	4.6

Table 1: NP vs. randomized NP. δ is the root Hermite factor. `Adv` is the target success probability. `log(Nb of bases)` is the number of bases needed to reach the target success probability in base-2 logarithm. `Red` and `Cost` indicate the lattice reduction time (using [10]) and total time in base-2 logarithm seconds respectively. `Enum` and `Pr` are respectively the number $\prod_{i=1}^{n} d_i$ of enumerations and success probability, in base-2 logarithm.

4 Solving BDD by (GNR) Pruned Enumeration

In this section, we study how BDD can be solved in practice using the pruned enumeration algorithm of Gama, Nguyen and Regev [6] (GNR). Timings given are for a single 3-Ghz Intel-Core2 core. First, we consider a theoretical application to LWE, to compare GNR pruning with our randomized NP algorithm. Then, we report improved experimental results on two well-known cryptanalytical applications of BDD: attacks on DSA with partially known nonces [15] and attacks on the GGH encryption challenges [13]. In these settings, the best method known in practice was to use the heuristic embedding method that transforms BDD into a Unique-SVP instance. Our experiments show that the embedding method is now outperformed by pruned enumeration, even without taking into account past improvements in lattice reduction algorithms [4]; and we did not find any application where Randomized-NP was better than pruned enumeration.

4.1 Further background on GNR Pruned Enumeration

We recall additional information on GNR pruned enumeration, to complete the description of Sect. 3.4. Given a target $\mathbf{t} \in \mathbb{Q}^m$, a basis $B = (\mathbf{b}_1, \ldots, \mathbf{b}_m)$ of a lattice L and a BDD radius R, GNR aims at finding the supposedly unique lattice vector $\mathbf{u} \in L$ such that $\|\mathbf{u} - \mathbf{t}\| \leq R$. To this aim, GNR [6] selects a bounding function determined by radius $R_1 \leq R_2 \leq \cdots \leq R_m = R$, and performs a depth-first search of the pruned-enumeration tree defined as follows: for each level $k \in \{0, \ldots, m\}$ where $k = 0$ corresponds to the root and $k = m$ corresponds to the leaves, the tree nodes of level k are all vectors $\mathbf{u} \in L$ such that $\|\pi_{m+1-i}(\mathbf{t} - \mathbf{u})\| \leq R_i$ for all $0 \leq i \leq k$. By assumption, there is at most a single leaf in the tree.

The complexity of the pruned enumeration is equal to the total number of nodes in the pruned-enumeration tree, up to some polynomial-time factor. To estimate this number of nodes, GNR [6] introduced the k-dimensional cylinder intersection defined by $R_1 \leq R_2 \leq \cdots \leq R_m$ as follows:

$$C_{R_1,R_2,\ldots,R_k} = \{(x_1, x_2, ..., x_k) \in \mathbb{R}^k, \forall j \leq k, \sum_{l=1}^{j} x_l^2 \leq R_j^2\}.$$

GNR apply the Gaussian heuristic to (heuristically) estimate the number of nodes at depth k by $H_k = \frac{\mathrm{vol}(C_{R_1,R_2,\ldots,R_k})}{\prod_{i=m+1-k}^{m} \|\mathbf{b}_i^*\|}$. GNR [6] provide efficient algorithms to estimate $\mathrm{vol}(C_{R_1,R_2,\ldots,R_k})$, and therefore H_k. This can be used to find good bounding functions by numerical optimization. In practice, one considers either the *linear bounding function* defined by $R_k = \sqrt{k/m} R_m$, or numerical functions found by numerical optimization.

GNR [6] rigorously defined a success probability, by assuming that the input basis is a random reduced basis. This probability p_{succ} is the probability that the closest lattice vector $\mathbf{u}$ to $\mathbf{t}$ remains in the enumeration tree.

They explained how to compute this probability in practice, if one assumes that the BDD error vector $\mathbf{t} - \mathbf{u}$ is a vector chosen uniformly at random in the sphere (or ball) of radius R. And they also give theoretical estimates for certain bounding functions: for instance, $p_{succ} = 1/m$ for linear bounding. The total cost of pruned-enumeration is roughly equal to:

$$\frac{T_{redu} + T_{node} \cdot N(R_1, \ldots, R_m, \mathbf{b}_1^*, \ldots, \mathbf{b}_m^*)}{p_{succ}(R_1, \ldots, R_m)},$$

where T_{redu} is the basis reduction time, T_{node} is the time for enumerating one node, and $N(R_1, \ldots, R_m, \mathbf{b}_1^*, \ldots, \mathbf{b}_m^*) = \sum_{k=1}^{m} H_k$ is the (approximate) number of nodes.

Though GNR [6] presented the first rigorous framework to analyzed pruned-enumeration algorithms, it must be stressed that the GNR analysis only holds for certain settings. For instance, when one wants to solve BDD in practice, it is often the case that we do not know the exact value of $\|\mathbf{t} - \mathbf{u}\|$, which makes a theoretical analysis difficult. In this case, we may take for R the expected value of $\|\mathbf{t} - \mathbf{u}\|$, or an upper bound satisfied with high probability; then pretending that $\mathbf{t} - \mathbf{u}$ is a vector chosen uniformly at random in the sphere (or ball) of radius R will only provide a rough estimate of p_{succ}. However, the distribution of $\mathbf{t} - \mathbf{u}$ is often known, which allows to compute experimentally (by sampling) the probability that $\mathbf{t} - \mathbf{u}$ satisfies a given bounding function, provided that the probability is not too small. This is the method we used in our experiments, and this is what we mean by success probability.

4.2 Application to LWE

In this subsection, we provide numerical evidence that pruned enumeration is better than NP for solving LWE. As mentioned previously, the analysis of [6] must be adapted, because the LWE noise distribution does not fit the GNR model: in particular, the exact norm of the noise is not known.

If we assume the CLWE model, the error vector $\mathbf{e}$ has continuous Gaussian distribution, then all the coordinates of $\mathbf{e}$ with respect to the normalized Gram-Schmidt basis $(\mathbf{b}_m^* / \|\mathbf{b}_m^*\|, \cdots, \mathbf{b}_1^* / \|\mathbf{b}_1^*\|)$, have Gaussian distribution. It follows that the success probability is:

$$p_{succ} = p_{succ}(R_1, \ldots, R_m) = Pr_{\mathbf{e} \sim \chi}(\forall j \in [1, m], \sum_{i=1}^{j} u_i^2 \leq R_j^2),$$

where χ is the (continuous) noise distribution of CLWE. If p_{succ} is high, we can use Monte Carlo sampling to compute it numerically. But for extreme pruning, where probabilities are voluntarily chosen very small, this is impractical. However, we notice that $\mathbf{e}/\|\mathbf{e}\|$ is uniformly distributed in the unit sphere, by definition. Hence, if we use a bound R_m such that $Pr(\|\mathbf{e}\| > R_m)$ is negligible, the probability $Pr_{\mathbf{u} \sim S^{m-1}}(\forall j \in [1, m], \sum_{i=1}^{j} u_i^2 \leq \frac{R_j^2}{R_m^2})$ can be used as a lower bound of the actual success probability.

Table 2 provides numerical comparisons between Randomized-NP and linear pruning (where the success probability is lower-bounded, with optimized enumeration radius around the expected length of error). As opposed to Table 1, Table 2 uses Chen-Nguyen estimates [4] for lattice reduction times. Independently of reduction time estimates, linear pruning is better in practice than NearestPlanes (even randomized).

				Red [4]	Adv	Randomized-NP				Linear Pruning				Log
n	q	s	δ	(log)	(log)	Enum (log) (log)	Pr (log)	Nb. of bases	Cost	Enum	Pr (log)	Radius factor	Cost	Speedup
128	2053	6.7	1.01	18.4	0	35.3	-7.1	7.1	26.6	35.4	-3.9	1.01	23.6	3
			1.012	8.2	-32	25.1	-34.5	2.5	11.7					
			1.013	< 0	-64	10.10	-63.1	0	< 0					
192	4093	8.9	1.007	61.8	0	78.7	-3.7	3.7	66.5	74.1	-0.9	1.5	62.8	3.7
			1.008	42	-32	48.6	-32	0	42					
			1.009	28	-64	44.9	-66.1	2.1	31					
256	4093	8.3	1.006	95.3	0	112.2	-15.1	15.1	111.4	113	-8.5	0.9	105.5	5.8
			1.006	95.3	-32	71.9	-31.6	0	95.3					
			1.007	62.2	-64	79.2	-77.7	13.7	76.9					

Table 2: Randomized-NP vs. Linear Pruning. δ is the root Hermite factor. `Red` indicates approximate lattice reduction times from Chen-Nguyen [4] in base-2 logarithm (in seconds). The rest of the data is organized as in Table 1, except that the enumeration radius chosen for linear pruning is the expected error length times he square root of `Radius factor`.

4.3 Application to GGH

In this subsection, we apply pruned-enumeration on the encryption challenges for the Goldreich-Goldwasser-Halevi cryptosystem [9] (GGH). The lattice dimension is either 200, 250, 300, 350 and 400. There are two types of challenges: key-recovery and message-recovery. Key-recovery can be viewed as solving m BDD-instances with error vector chosen uniformly at random from $[-4, \ldots, +3]^m$. To the best of our knowledge, none of these key-recovery challenges was ever solved. Each message-recovery challenge is a BDD instance where the error vector is chosen uniformly at random from $\{-3, +3\}^m$: Nguyen [13] solved the message-recovery challenges in dimensions 200,250,300 and 350 by showing how to reduce each such BDD instance to a small number of BDD instances with error in $\{-1/2, +1/2\}^m$, and solving these easier BDD instances using the embedding strategy and lattice reduction: BKZ-20 was enough for dimensions 200-300, but dimension 350 required pruned-BKZ with higher blocksize 60.

First, we re-solved the GGH-350 message-recovery challenge by using only a BKZ-20 reduced basis and high-probability pruning, which shows that pruning is better than the well-known embedding method. In this case, the BDD radius is exactly $R_m = \frac{\sqrt{m}}{2}$, after applying Nguyen's trick [13], and the BDD factor γ is ≈ 0.125. We used a high-probability bounding function defined as follows: $R_k^2 = \min\{E(X_k) + 3\sigma(X_k), 1\}R_m^2$ where E and σ denote respectively the

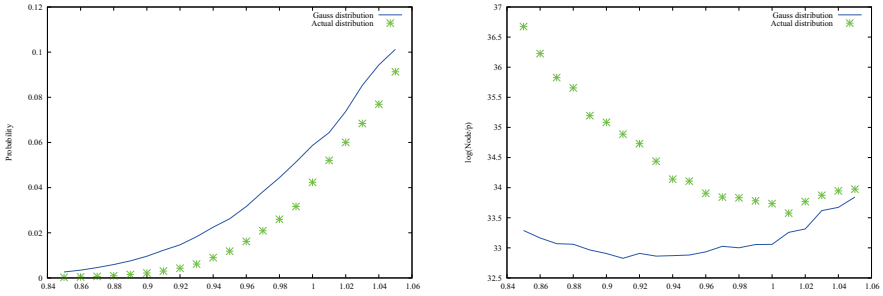
expectation and standard deviation of the distribution $Beta(k/2, (m-k)/2)$, namely $\frac{k}{m}$ and $\sqrt{\frac{k(m-k)}{m^2(\frac{m}{2}+1)}}$. The pruned tree contained 1.76×10^{11} nodes, very close to the Gaussian heuristic estimate 1.73×10^{11}. The error behaved as if it was uniformly distributed in the sphere of radius R_m: indeed, in both our experiments and in the uniform model, the success probability was ≈ 0.92. Note that the largest GNR experiments [6] used a lattice dimension of 110. Because enumeration (pruned or not) has a super-exponential running time, one may have thought that much larger dimensions would be unreachable. However, our experiments show that dimensions as high as 350 are reachable, provided that the BDD enumeration radius is sufficiently small.

Next, we recovered several secret-key vectors in dimensions 200, 250 and 300, using pruned-enumeration and BKZ 2.0 [4] as the reduction algorithm: the experiments are summarized in Table 3, where the Gaussian heuristic is used to estimate the number of nodes. In these BDD instances, the exact

Dimension	200	250	300	300
BKZ blocksize	60	60	90	90
Bounding function	linear	linear	linear	optimized
Estimated Nb of Nodes	800	5.84×10^8	7.58×10^{10}	1.33×10^9
Average Nb of Nodes	666	5.93×10^8	-	1.35×10^9
Success probability	0.0418	0.0409	0.0371	0.0185
Nb of Success	4	11	-	2

Table 3: Key-recovery for GGH Challenges

BDD radius is unknown, but the factor γ is approximately $\frac{\sqrt{\frac{11}{2}m}}{4\sqrt{m}} \approx 0.59$, which is more difficult than for the message-recovery challenges. For each dimension, we computed only one BKZ-reduced basis, and tried to recover as many secret-key vectors with the same basis: because the success probability is much lower than 1 for one pruned enumeration, we actually only recovered a fraction of all secret-key vectors, but of course, our experiments show that one could recover all secret-key vectors simply by repeating our experiments a small number of times. For dimension 300, linear bounding was not sufficient, so we tried another bounding function by optimization, using the GNR method [6]: we start with the linear bounding function, then randomly modify it by small perturbation successively. Using this bounding function, one pruned-enumeration only takes several minutes. In these experiments, the enumeration radius was chosen as the expected error length. However, there is clearly a trade-off if one wants to optimize the total running time: by selecting a smaller radius, one can decrease the running time of a single enumeration, at the expense of the success probability. Fig 1 shows the impact of varying the enumeration radius around the expected length, on the success



(a) Success probability for GGH-250 key-recovery

(b) Total cost for GGH-250 key-recovery

Fig. 1: Here, the x-coordinate is the ratio between the (squared) enumeration radius with the (squared) expected length of the error vector. In (b), the y-coordinate is $\log_2$(Number of enumeration nodes/Success probability)

probability and the total cost: here, one obtains slightly better results by increasing the radius. Fig 1 also compares experimental probabilities and costs with that of a Gaussian modelization where the error vector is chosen with Gaussian distribution of expectation $\sqrt{\frac{11}{2}m}$: we see that it is better to compute experimental probabilities (by sampling the error distribution).

We also tried to solve GGH challenges using Randomized-NP but the performances were worse than pruned enumeration.

4.4 Application to DSA

In this subsection, we apply pruned-enumeration to attack the Digital Signature Algorithm [12] (DSA) with partially known nonces. Each DSA signature generation require the use of a one-time key k modulo q, where q is usually a 160-bit prime number. It is well-known (and obvious) that disclosing the full one-time key k of a single (message,signature) pair allows to recover the DSA secret key in polynomial time. It is also well-known (but not obvious) that disclosing ℓ bits of each one-time key k for several (message,signature) pair allows to recover the DSA secret key, see *e.g.* [15]. More precisely, this cryptanalytical problem can be reduced to the so-called hidden number problem (HNP), which can itself be reduced to BDD. For any real z, let the symbol $|\cdot|_q$ be $|z|_q = \min_{b\in\mathbb{Z}} |z - bq|$. $APP_{\ell,q}(n)$ denotes any rational number r satisfying $|n - r|_q \leq \frac{q}{2^{\ell+1}}$. The HNP asks to recover $\alpha \in \mathbb{Z}_q$, given many approximations $u_i = APP_{\ell,q}(\alpha t_i)$ where each t_i is known and chosen uniformly at random, for $1 \leq i \leq d$. The reduction to BDD works as follows. One constructs the

$(d+1)$-dimensional lattice spanned by the following row matrix:

$$\begin{pmatrix} q & 0 & \cdots & 0 & 0 \\ 0 & q & \ddots & \vdots & \vdots \\ \vdots & \ddots & \ddots & 0 & \vdots \\ 0 & \cdots & 0 & q & 0 \\ t_1 & \cdots & \cdots & t_d & \frac{1}{2^{l+1}} \end{pmatrix} \tag{2}$$

The target vector is $\mathbf{u} = (u_1, u_2, \ldots, u_d, 0)$. There exists a lattice vector $\mathbf{h} = (\alpha t_1 + qh_1, \ldots, \alpha t_d + qh_d, \frac{\alpha}{2^{\ell+1}})$, such that $\| \mathbf{h} - \mathbf{u} \| \leq \sqrt{d+1}\frac{q}{2^{\ell+1}}$. And finding $\mathbf{h}$ discloses α.

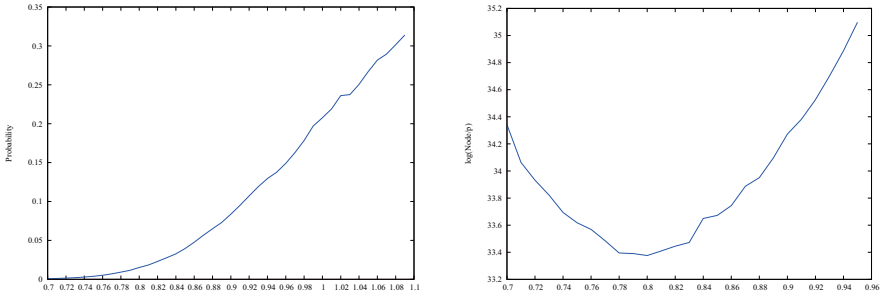
Nguyen and Shparlinski [15] used this attack to recover the DSA secret key in a few hours, given the $\ell = 3$ least significant bits of each one-time key for about 100 signatures, but the attack failed for $\ell = 2$. By using BKZ-90 reduction [4] and linear pruning, we were able to attack the $\ell = 2$ case given about 100 signatures, within a few hours, but the lattice needs to be slightly changed: indeed, for the GNR analysis to hold, one needs that the error vector looks like a random vector in the Gram-Schmidt basis; because the shape of reduced bases is special for these HNP lattices, one needs to modify the right-bottom coefficient of the row matrix by some scaling factor.

We constructed 100 instances to check the cost and success probability. The average number of nodes was 1.37×10^{10}, slightly smaller than the estimated number of nodes 1.5×10^{10}. The average actual running time is about 4185 seconds per enumeration. Our experiments solved 23 out of 100 instances, which means that the running time of a single enumeration needs to be multiplied by roughly 4, which is a few hours at most. Because the exact length of the error vector is not known, like in the GGH case, there is a trade-off for choosing the enumeration radius: Fig. 2 shows the impact when varying the enumeration radius around the expected length, like Fig. 1. Here, the optimal enumeration radius is about $\sqrt{0.8} \approx 0.89$ smaller than the expected length.

Acknowledgements. Part of this work is supported by the Commission of the European Communities through the ICT program under contract ICT-2007-216676 ECRYPT II, the European Research Council, and by China's 973 Program (Grants 2013CB834201 and 2013CB834205).

References

1. M. Ajtai. Generating hard instances of lattice problems (extended abstract). In *STOC*, pages 99–108, 1996.
2. L. Babai. On Lovász' lattice reduction and the nearest lattice point problem (shortened version). In *Proc. STACS'85*, volume 182 of *Lecture Notes in Computer Science*, pages 13–20. Springer, 1985.



(a) Success probability for DSA (b) Total cost for DSA

Fig. 2: Here, the x-coordinate is the ratio between the (squared) enumeration radius and the (squared) expected length the error vector. In (b), the y-coordinate is $\log_2$(Number of enumeration nodes/Success probability)

3. Z. Brakerski and V. Vaikuntanathan. Fully homomorphic encryption from ring-lwe and security for key dependent messages. In *Proc. CRYPTO'11*, volume 6841 of *Lecture Notes in Computer Science*, pages 505–524. Springer, 2011.
4. Y. Chen and P. Q. Nguyen. BKZ 2.0: Better lattice security estimates. In *Proc. ASIACRYPT'11*, volume 7073 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2011.
5. N. Gama and P. Q. Nguyen. Predicting lattice reduction. In *Proc. EUROCRYPT'08*, volume 4965 of *Lecture Notes in Computer Science*, pages 31–51. Springer, 2008.
6. N. Gama, P. Q. Nguyen, and O. Regev. Lattice enumeration using extreme pruning. In *Proc. EUROCRYPT'10*, volume 6110 of *Lecture Notes in Computer Science*, pages 257–278. Springer, 2010.
7. C. Gentry. Fully homomorphic encryption using ideal lattices. In *Proc. STOC '09*, pages 169–178. ACM, 2009.
8. C. Gentry, C. Peikert, and V. Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *Proc.STOC'08*, pages 197–206. ACM, 2008.
9. O. Goldreich, S. Goldwasser, and S. Halevi. Public-key cryptosystems from lattice reduction problems. In *Proc. CRYPTO 1997*, volume 1294 of *Lecture Notes in Computer Science*, pages 112–131. Springer, 1997.
10. R. Lindner and C. Peikert. Better key sizes (and attacks) for lwe-based encryption. In *CT-RSA*, volume 6558 of *Lecture Notes in Computer Science*, pages 319–339. Springer, 2011.
11. D. Micciancio and O. Regev. Lattice-based cryptography. In D. J. Bernstein, J. Buchmann, and E. Dahmen, editors, *Post-Quantum Cryptography*, pages 147–191. Springer, 2009.
12. National Institute of Standards and Technology (NIST). Fips publication 186:digital signature standard. 1994.
13. P. Q. Nguyen. Cryptanalysis of the Goldreich-Goldwasser-Halevi cryptosystem from Crypto '97. In *Proc. CRYPTO 1999*, volume 1666 of *Lecture Notes in Computer Science*, pages 288–304. Springer, 1999.
14. P. Q. Nguyen. Public-key cryptanalysis. In I. Luengo, editor, *Recent Trends in Cryptography*, volume 477 of *Contemporary Mathematics*. AMS–RSME, 2009.
15. P. Q. Nguyen and I. Shparlinski. The insecurity of the digital signature algorithm with partially known nonces. *J. Cryptology*, 15(3):151–176, 2002.

16. C. Peikert. Public-key cryptosystems from the worst-case shortest vector problem: extended abstract. In *Proc. STOC'09*, pages 333–342. ACM, 2009.
17. O. Regev. On lattices, learning with errors, random linear codes, and cryptography. In *Proc. STOC'05*, pages 84–93. ACM, 2005.
18. O. Regev. The learning with errors problem (invited survey). In *Proc. IEEE Conference on Computational Complexity*, pages 191–204, 2010.
19. C.-P. Schnorr. Lattice reduction by random sampling and birthday methods. In *Proc. STACS '03*, volume 2607 of *Lecture Notes in Computer Science*, pages 145–156. Springer, 2003.
20. C.-P. Schnorr. Lattice reduction by random sampling and birthday methods. In *Proc.STACS'03*, volume 2607 of *Lecture Notes in Computer Science*, pages 145–156. Springer, 2003.
21. C.-P. Schnorr and M. Euchner. Lattice basis reduction: improved practical algorithms and solving subset sum problems. *Math. Programming*, 66:181–199, 1994.

A Description of Pruned Enumeration for BDD

Algorithm 4 Pruned Enumeration for BDD (BDD version of [6])

Input: A basis $B = (\mathbf{b}_1, \ldots, \mathbf{b}_m)$, a target vector $\mathbf{t} = \sum_{i=1}^{m} t_i \mathbf{b}_i$, a bounding function $R_1^2 \leq \cdots \leq R_m^2$, the Gram-Schmidt matrix μ and the (squared) norms $\|\mathbf{b}_1^*\|^2, \ldots, \|\mathbf{b}_m^*\|^2$.

Output: Nothing or the coefficients of a lattice vector $\mathbf{v}$ such that the projections of $\mathbf{v} - \mathbf{t}$ have norms less than the R_i's, *i.e.* $\|\pi_{m+1-k}(\mathbf{v} - \mathbf{t})\| \leq R_k$ for all $1 \leq k \leq m$.

$\sigma \leftarrow (0)_{(m+1)\times m}$; $r_0 = 0; r_1 = 1; \cdots; r_m = m; \rho_{m+1} = 0$
for $k = m$ **downto** 1
  for $i = m$ **downto** $k+1$ **do** $\sigma_{i,k} \leftarrow \sigma_{i+1,k} + (t_i - v_i)\mu_{i,k}$ **endfor**
  $c_k \leftarrow t_k + \sigma_{k+1,k}$ // $c_k \leftarrow t_k + \sum_{i=k+1}^{m}(t_i - v_i)\mu_{i,k}$, centers
  $v_k \leftarrow \lfloor c_k \rceil$ // *current combination*;
  $w_k = 1$ // *jumps*;
  $\rho_k = \rho_{k+1} + (c_k - v_k)^2 \cdot \|\mathbf{b}_k^*\|^2$
endfor
$k = 1$;
while true **do**
  $\rho_k = \rho_{k+1} + (c_k - v_k)^2 \cdot \|\mathbf{b}_k^*\|^2$ // *compute norm squared of current node*
  if $\rho_k \leq R_{m+1-k}^2$ (*we are below the bound*) **then**
    if $k = 1$ **then**
      return $(v_1, \ldots, v_m)$; (*solution found; program ends*)
    else
      $k \leftarrow k - 1$ // *going down the tree*
      $r_{k-1} \leftarrow \max(r_{k-1}, r_k)$ // *to maintain the invariant for* $j < k$
      for $i = r_k$ **downto** $k+1$ **do** $\sigma_{i,k} \leftarrow \sigma_{i+1,k} + (t_i - v_i)\mu_{i,k}$ **endfor**
      $c_k \leftarrow t_k + \sigma_{k+1,k}$ // $c_k \leftarrow t_k + \sum_{i=k+1}^{m}(t_i - v_i)\mu_{i,k}$
      $v_k \leftarrow \lfloor c_k \rceil$; $w_k = 1$
    end if
  else
    $k \leftarrow k + 1$ // *going up the tree*
    if $k = m + 1$ **then**
      return $\emptyset$ (*there is no solution)*
    end if
    $r_{k-1} \leftarrow k$ // *since* v_k *is about to change, indicate that* (i, j) *for* $j < k$ *and* $i \leq k$ *are not synchronized*
    // *update* v_k
    if $v_k > c_k$ **then** $v_k \leftarrow v_k - w_k$ **else** $v_k \leftarrow v_k + w_k$
    $w_k \leftarrow w_k + 1$
  end if
end while