原文：BKZ 2.0: Better Lattice Security Estimates

作者：Yuanmi Chen and Phong Q. Nguyen

译者：随缘<su1yu4n@qq.com>

摘要

目前已知最佳的高维格基规约算法是 Schnorr-Euchner 的 BKZ 算法：所有格密码系统的安全性估计都是基于之前 NTL 中 BKZ 算法的实现。然而，格枚举算法研究的最新进展表明，当前 NTL 中的 BKZ 实现已不再是最好的实现。不过人们还不清楚此进展对格密码的安全性估计的确切影响。我们通过 BKZ 2.0 的大量实验对影响进行了评估。BKZ 2.0 是第一个使用了最新成果的 BKZ 实现。实现中使用了近期提出的算法改进，例如 Gama-Nguyen-Regev 剪枝。我们提出了一种高效的模拟算法来模拟 BKZ 在高维度格且分块大小(blocksize)大于 50 的情况下的行为。这个模拟算法可以近似地预测输出质量和运行时间，从而修正格的安全性估计。例如，我们的模拟表明最小的 NTRU 签名参数集（据称可以提供相当于 93 位密钥的安全性来抵御基于格的密钥恢复攻击）实际上最多只能提供 65 位密钥的安全性。

1 引言

格是$\mathbb{R}^m$的离散子群。格L由一组格基表示，即$\mathbb{R}^m$中一组线性无关向量$\mathbf{b}_1, ..., \mathbf{b}_n$。$L$为$\mathbf{b_i}$所有整数线性组合构成的集合$L(\mathbf{b}_1, ..., \mathbf{b}_n) = \{\sum_{i=1}^n x_i \mathbf{b}_i \, , x_i \in \mathbb{Z}\}$。我们称整数$n$是$L$的维数。格基约化的目的是找到格$L$一个由较短且近似正交的向量组成的基。格基规约算法有许多应用（见[35]），特别是在公钥密码分析中，它们被用于破解 RSA 和 DSA 等密码体制的特例（见[32]和其中的参考文献）。大体上，格基规约算法有两种类型：

- 近似算法，如著名的 LLL 算法[22,35]及其衍生出的分块格基规约算法[41,42,7,8]。这样的算法可以找到相对较短的向量，但在高维的格中通常找不到最短的向量。

- 精确算法，输出最短（或几乎最短）的向量。有空间复杂度能够接受的枚举算法[38,20,6,42,43,10]和空间复杂度为指数级别的算法[3,36,30,29]。它们的时间复杂度都

是$2^{O(n)}$，不过后者在实践中优于前者。

在高维格上我们只能调用近似算法，不过这两种算法是互补的：近似算法会调用精确算法，将其作为它的子算法；而精确算法也会调用近似算法作为预处理。理论上，最佳近似算法是 Gama-Nguyen 约化[8]。但实验（如[9]的实验，或 GGH 挑战[12]的密码分析[31,21]）表明，实践中已知的高维最佳近似算法是 BKZ 算法。这个算法由 Schnorr 和 Euchner 于 1994 年发表[42]，并在 NTL[44]中实现。与所有分块格基规约算法[41,7,8]一样，BKZ 与 LLL 算法相比有一个额外的输入参数——分块大小β，它会影响 BKZ 算法的运行时间和输出质量：BKZ 频繁调用格枚举子算法[38,20,6,42]，这个子算法会在维数小于等于β的投影格中查找几乎最短的向量。随着β的增加，输出基约化程度越来越高，但代价却显著增加：枚举子算法的开销在β中通常是超指数级别的，即进行$2^{O(\beta^2)}$次多项式时间的运算（见[10]）；实验[9]表明，调用次数随着β和格的维数n的增加而急剧增加：例如固定β大于等于 30，如果n不是指数的话，调用次数似乎是超多项式级别的。于是 BKZ 有两个常规用法：

1. 输入格的维度n为任意值时选取$\beta \approx 20$，或中等维度n时将β选取为 30-40 左右（最多 100 左右）。这种情况下，BKZ 会在可接受的时间内运行完，并且结果通常优于 LLL 约化。
2. 对于高维数n中令分块大小$\beta \geq 40$，寻找越来越短的格向量。这种情况下，BKZ 算法不会在合适的时间内完成。在实际应用中，如果输入基比较好的话，算法通常会在几个小时或几天后跑完并给出结果。我们注意到，Hanrot 等人[14]最近对使用了中止技术的 BKZ 算法（下称中止 BKZ）的输出格基质量最坏情况界限给出了证明，它只比不使用中止技术的 BKZ 稍差一点。一般通过剪枝技术来优化枚举子算法的运行时间[42,43,10]：例如，NTL 中 BKZ 的实现提出了 Schnorr-Horner 剪枝[43] （SH 剪枝），它增加了另一个输入参数p，其影响在[10]中才阐明。有人使用分块大小 $\beta=40$ 和 SH 因子 $p=14$ 的中止 BKZ 求解出了最高维数的 GGH 密码破解挑战[12]。

如何评估 BKZ 的输出质量是一个重要问题，因为格算法的实际性能表现往往比理论上的预期更好。格基规约算法输出基的质量由 Hermite 因子来衡量，这是 Gama 和 Nguyen 的建议[9]。实际上，所有已知的格算法的 Hermite 因子对于维数n通常是指数的，即c^n，其中c取决于算法的参数。文献[9]的实验表明，在实际应用中，BKZ 的 Hermite 因子通常是$c(\beta,n)^n$，其中$c(\beta,n)$在β一定的情况下随n趋于无穷迅速收敛。然而，只有在β较小的时候($\beta \leq 30$)，$c(\beta,n)$的极限值才已知，并且$c(\beta,n)$的理论上界[9,14]显著高于实验值。

格密码体制的所有安全性估计和参数建议（如近期文献[28,39,23]和 NTRU 文献[18]）都以 NTL 之前的 BKZ 实现为基准，但这些估计是否值得参考有待商榷。首先，这些基准都是按照上面的用法 1 计算的：文献[18]指出对于 NTRU 挑战，“从未发现剪枝能减少运行时间，因此推测剪枝过程基本没有被调用”，并使用$\beta \leq 25$，而[39,23]使用$\beta \leq 30$。这说明这些安全性估计要么假设 BKZ 不能在$\beta \geq 30$ 的情况下运行，要么从$\beta \leq 30$ 时的情况来推测β较大时的$c(\beta, n)$。此外格枚举算法的最新进展[10]表明，现在可以在比以前想象的更高的维度（例如$\beta \approx 110$）上进行枚举，但是对于$\beta \geq 50$ 的情况，没有已知的$c(\beta, n)$近似值。而 NTL 的实现并没有参考这些最近的改进，因此这个实现并不是目前最优的实现。

我们的成果：我们首次将β较大（$\beta \geq 40$）的 BKZ 用于高维的格，进行了拓展实验。这是通过实现 BKZ 2.0 算法（一种改进版 BKZ）做到的，该算法应用了最新的算法改进。主要的改进是将 Gama、Nguyen 和 Regev[10]在 EUROCRYPT'10 上发明的深度剪枝(sound pruning)技术结合起来。这些修改显著地降低了枚举子算法的运行时间，且在选取适当参数的情况下几乎不会降低输出质量，从而让我们得以使用很大的β。BKZ 2.0 的性能优于 NTL 对 BKZ 的实现（即使是与使用了 SH 剪枝[43]的那种 BKZ 相比），这一点我们通过打破诸如 Darmstadt 的格挑战[24]或 SVP 挑战[40]等问题的求解记录得以确认：例如，我们在以前的 214 维 NTRU 格[18]中用 242.62 个时钟周期计算出了最短向量，进行的运算次数至少比之前少 70 倍[25]。

更重要的是，通过实验我们提出一种有效的模拟算法来模拟 BKZ 在（任意）大的分块大小≥ 50 的情况下的执行，来猜测此时算法输出向量的近似长度和所需时间。值得提出的是，该算法首次对任意大的β提供了$c(\beta, n)$的预测($\beta > 50$)。对于给定的目标长度，模拟算法可以估算获得这种短向量所需的β，以及所需的枚举调用数量。一旦我们知道枚举算法开销的一个精确近似值，就能得到算法的一个近似运行时间。我们为目前最好的枚举子算法计算了这样的近似值。

我们的模拟改进了 Gama-Nguyen 关于格上困难问题的安全性估计[9]，原估计没有考虑剪枝，比如 NTRU[19,16]和[23,39]的安全性估计。我们通过修正安全性估计来说明我们模拟的价值。我们的模拟表明，最小的 NTRU 签名参数集（据称至少能提供 93 位的安全性来抵御密钥恢复格攻击）实际上最多能提供 65 位的安全性。我们利用模拟对 Gentry 和 Halevi 最近提出的全同态加密挑战[11]进行了第一次具体的安全评估。似乎所有这些挑战都没有提供非常高的安全级别，除了最大的一个，它似乎最多提供 100 位的安全级别。

论文组织结构(roadmap)：我们以第 2 节为起点，介绍格相关的数学背景和符号。在第 3 节中我们回顾了 BKZ 算法。在第 4 节中我们通过描述如何修改原

先 BKZ 算法，以实现 BKZ 2.0。在第 5 节中我们简要记录了获得的格算法的新记录。我们在第 6 节中提出了一种仿真算法来预测 BKZ 2.0 在（任意）分块大小较大情况下的性能，并用其在第 7 节中修正现有的安全性估计。更多信息可以在完整版本中找到。

2 前置背景知识

符号说明：我们用矩阵的行向量表示格基（与算法实现保持一致），并使用粗体字体表示向量：如果 $\mathrm{B} = (\mathbf{b}_1, ...\mathbf{b}_n)$ 是矩阵，则其行向量是$\mathbf{b}_i$。向量$\mathbf{v} \in \mathbb{R}^m$的欧氏范数记作$\|\mathbf{v}\|$。我们用$\mathrm{Ball}_n(R)$表示半径为$R$的$n$维欧氏球，用$V_n(R) = R^n \cdot \frac{\pi^{n/2}}{\Gamma(n/2+1)}$表示其体积。$n$维单位球由 S^{n-1} 表示。设L是$\mathbb{R}^m$中的n维格。它的体积$\mathrm{vol}(L)$指由L的任何基生成基本域的n维体积。

正交化：$n \times m$的基$B = (\mathbf{b}_1, ...\mathbf{b}_n)$可以唯一地写成$B = \mu \cdot D \cdot Q$，其中$\mu = (\mu_{i,j})$是对角线为 1，$n \times n$的下三角矩阵，$D$是$n \times n$正定对角矩阵，$Q$是行向量相互正交的$n \times m$矩阵。那么$\mu D$是$B$（相对于$Q$）的下三角表示，$B^* = DQ = (\mathbf{b}_1^*, \cdots, \mathbf{b}_n^*)$ 是对基进行施密特正交化后的结果。D是由 $\mathbf{b}_i^*$ 形成的对角矩阵。我们用$\pi_i (1 \le i \le n+1)$表示$(\mathbf{b}_1, ...\mathbf{b}_n)^\perp$上的正交投影。对于 $1 \le j \le k \le n$，我们用$B_{[j,k]}$表示局部投影块$\left(\pi_j(\mathbf{b}_j), \pi_j(\mathbf{b}_{j+1}), \cdots, \pi_j(\mathbf{b}_k)\right)$，用$L_{[j,k]}$表示由$B_{[j,k]}$生成的格，其维数为$k - j + 1$。

随机格：基于典型群的 Haar 测度（见[1]），给定格的体积有随机（实）格这一概念。最近的实验中使用了随机整格这一概念：对于任意整数V，一个体积为V的随机n维整格是指在体积V的有限多个n维整格中均匀随机选择的一个格。文献[13]表明，当V趋于无穷时，整格按$V^{1/n}$缩小时，体积V的整格上的均匀分布收敛到单位体积的随机（实）格上的分布。在随机格的实验中，我们说的n维整格是指在体积为V中的格等可能选取时，选出的某一个格，这里的V是一个比特长度为$10n$的素数：对于体积为素数的情况，使用 Hermite 标准型在均匀分布中采样是很简单的。理论上比特长度为$O(n^2)$更好（根据[13]的结果），但这样会显著增加算法运行时间，且对实验结果的影响并不明显。

Gaussian Heuristic：给定一个格L和一个“好”的集合S，Gaussian Heuristic 预测$S \cap L$中的点数约为 $\mathrm{vol}(S)/\mathrm{vol}(L)$。在某些情况下，可以确定这个启发函数是切合实际的[1]或偏差较大的[27]。

最短向量：L的非零最短向量的范数为$\lambda_1(L)=\min_{\mathbf{v}\in L,\mathbf{v}\neq 0}\|\mathbf{v}\|$。如果 Gaussian Heuristic 对任意球S成立，我们有$\lambda_1(L)\approx GH(L)$，其中$GH(L)=\mathrm{vol}(L)^{1/n}\cdot V_n(1)^{-1/n}$。Minkowski 定理表明，对于任意格$L$，$\lambda_1(L)\ \leq 2GH(L)$。对于随机实格，$\lambda_1(L)$以极大的概率近似于$GH(L)$（见[1]）。

约化基：我们回顾了一些经典的约化概念。一组格基$B=(\mathbf{b}_1,...\mathbf{b}_n)$是：

- 长度约化的(size reduced)，如果其 Gram-Schmidt 矩阵μ满足$|\mu_{i,j}|\leq 1/2(1\leq j<i\leq n)$。
- LLL-约化的(LLL-reduced)，如果该格基满足以下条件：给定因子$0<\epsilon<1$ [22]，基B是长度约化的且其 Gram-Schmidt 正交化满足$\|\mathbf{b}_{i+1}^*+\mu_{i+1,i}\mathbf{b}_i^*\|^2\geq(1-\epsilon)\|\mathbf{b}_i^*\|^2$，$1\leq \mathrm{i}<\mathrm{n}$。如果忽略因子$\epsilon$，则表示因子$\epsilon=0.01$，实践中经常将$\epsilon$选择为这个值。
- BKZ-约化的(BKZ-reduced)[41]，如果满足以下条件：给定分块大小$\beta\geq 2$和因子$0<\epsilon<1$，该格基对于ϵ是 LLL-约化的，并且对于$1\leq j\leq n$有$\|\mathbf{b}_j^*\|=\lambda_1(L_{[j,k]})$，其中$k=\min(j+\beta-1,n)$。

人们通常感兴趣的是将 Hermite 因子$\|\mathbf{b}_1\|/\mathrm{vol}(L)^{1/n}$降低（见[9]）（这个因子完全由数列$\|\mathbf{b}_1^*\|,...,\|\mathbf{b}_n^*\|$决定）。这是因为 Hermite 因子决定了在解决重要格难题时算法的表现：关于近似 SVP 和 SVP，参见[9]，关于 SIS 和 LWE，参见[28,39,23]。结果表明，在输入基是足够随机的条件下，一般归约算法（如 LLL 或 BKZ）产生的基的 Gram-Schmidt 系数具有某种“典型形状”[9,34]。简单来说，形状大致满足$\|\mathbf{b}_i^*\|/\|\mathbf{b}_{i+1}^*\|\approx q$，其中$q$取决于归约算法，除了第一个索引$i$。这意味着 Hermite 因子的形式通常是$c^n$，其中$c\approx\sqrt{q}$。

3 BKZ 算法

3.1 算法描述

BKZ 算法[42]由格L的输入基$B=(\mathbf{b}_1,...\mathbf{b}_n)$计算出分块大小$\beta\geq 2$且约化因

子$\epsilon > 0$ 的 BKZ 约化基。该算法首先对B进行 LLL 约化，然后对每个局部基$B_{[j, min\,(j+\beta-1,n)]}(1 \leq j \leq n)$进行约化以确保这局部基中第一个向量在投影格中是最短的。这就是算法 1 的流程。算法中每个局部块在被枚举之前先进行 LLL 约化，然后再这样做：将一个索引 j 初始化为 1。在每次迭代中，BKZ 对局部投影格$L_{[j,k]}$进行枚举以找到$v = (v_1, ..., v_n) \in \mathbb{Z}^n$，满足$\left\|\pi_j\left(\sum_{i=j}^{k} v_i \mathbf{b}_i\right)\right\| = \lambda_1(L_{[j,k]})$，其中$k = \min(j + \beta - 1, n)$。我们令$h = \min(k + 1, n)$为下一次迭代中新块的尾索引：

- 如果 $\|\mathbf{b}_j^*\| > \lambda_1(L_{[j,k]})$，则将$\mathbf{b}^{\text{new}} = \sum_{i=j}^{k} v_i \mathbf{b}_i$插入$\mathbf{b}_{j-1}$和$\mathbf{b}_j$之间。这时基已不再是 LLL-约化基，因此要对$(\mathbf{b}_1, ..., \mathbf{b}_{j-1}, \mathbf{b}^{\text{new}}, \mathbf{b}_j, ..., \mathbf{b}_h)$调用 LLL，以产生新的 LLL-约化基$(\mathbf{b}_1, ..., \mathbf{b}_h)$。
- 否则，对截断的基$(\mathbf{b}_1, ..., \mathbf{b}_h)$调用 LLL。

因此，在每次迭代结束时，基$B = (\mathbf{b}_1, ..., \mathbf{b}_h)$必然是 LLL-约化基。当 j 达到 n 时，除非枚举不成功，否则它会被重置为 1。在枚举失败的情况下，算法终止。算法 1 中，z 就是用来记录失败的的枚举数以检查算法是否需要终止的。

Algorithm 1. The Block Korkin-Zolotarev (BKZ) algorithm

Input: A basis $B = (\mathbf{b}_1, \ldots, \mathbf{b}_n)$, a blocksize $\beta \in \{2, \ldots, n\}$, the Gram-Schmidt triangular matrix μ and $\|\mathbf{b}_1^*\|^2, \ldots, \|\mathbf{b}_n^*\|^2$.

Output: The basis $(\mathbf{b}_1, \ldots, \mathbf{b}_n)$ is BKZ-β reduced

```
z ← 0; j ← 0; LLL(b_1, ..., b_n, μ);// LLL-reduce the basis, and update μ
while z < n − 1 do
  j ← (j mod (n − 1)) + 1; k ← min(j + β − 1, n); h ← min(k + 1, n); // define the local block
  v ←Enum(μ_[j,k], ||b_j*||^2, ..., ||b_k*||^2); // find v = (v_j, ..., v_k) ∈ Z^(k−j+1) − 0 s.t. ||π_j(Σ_{i=j}^k v_i b_i)|| = λ_1(L_[j,k])
  if v ≠ (1, 0, ..., 0) then
    z ← 0; LLL(b_1, ..., Σ_{i=j}^k v_i b_i, b_j, ..., b_h, μ) at stage j; //insert the new vector in the lattice at the start of the current block, then remove the dependency in the current block, update μ.
  else
    z ← z + 1; LLL(b_1, ..., b_h, μ) at stage h − 1; // LLL-reduce the next block before enumeration.
  end if
end while
```

算法 1　BKZ 算法

3.2 枚举子算法

BKZ 需要一个寻找局部投影格$L_{[j,k]}$中最短向量的子算法：给定两个整数 j 和 k 作为输入，其中$j \leq k \leq n$，输出满足 $\left\|\pi_j\left(\sum_{i=j}^{k} v_i \mathbf{b}_i\right)\right\| = \lambda_1(L_{[j,k]})$ 的 $v = (v_j, ..., v_k) \in \mathbb{Z}^{k-j+1}$。在实践中以及在 BKZ 原版论文[42]中，这子算法是通过枚举实现的。算法将$R = \|\mathbf{b}_j^*\|$赋值为$\lambda_1(L_{[j,k]})$的初始上界。枚举遍历由局部投影格$L_{[k,k]}, L_{[k-1,k]}, ..., L_{[j,k]}$的“一半”向量组成的枚举树的$L_{[j,k]}$范数。树的深度为$k - j + 1$，对于每个$d \in \{0, ..., k - j + 1\}$，深度为 d 的节点为 0。并且当满足$\mathbf{u} = \sum_{i=j}^{k'} u_i \mathbf{b}_i$（其中对于$j \leq k' \leq k$ ，$u_{k'} > 0$）且$\|\pi_{k-d+1}(\mathbf{u})\| \leq R$ 时，$\pi_{k-d+1}(\mathbf{u})$一定在格$L_{[k-d+1,k]}$中，即$\pi_{k-d+1}(\mathbf{u}) \in L_{[k-d+1,k]}$。深度为 d 的节点$\mathbf{u} \in L_{[k-d+1,k]}$的父结点是深度为 $d-1$ 的 $\pi_{k-d+2}(\mathbf{u})$。每个节点的子节点是按欧式范数递增排列的。Schnorr-Euchner 算法[42]对树进行深度优先搜索(DFS)，以输出最小范数的非零叶子节点，并进行以下修改:每次发现一个新的(非零的)叶，就将枚举半径R 赋值为叶的范数。基约化的程度越高，树中的节点就越少，枚举的时间成本就越低。枚举算法的运行时间为N个多项式时间操作，其中N为树节点总数。Hanrot 和 Stehlé[15]注意到,假设算法不更新R,深度d的节点数量可以根据 Gaussian Heuristic 估计为:

$$H_d(R) = \frac{1}{2} \cdot \frac{V_d(R)}{\prod_{i=k-d+1}^{k} |b_i^*|} = \frac{1}{2} \cdot \frac{R^d V_d(1)}{\prod_{i=k-d+1}^{k} |b_i^*|} \qquad (1)$$

Gama 等人的研究[10]表明,至少对于足够大的$k - j + 1$和典型的约化基来说，这个估计与实验所测非常吻合。因此，在实际的（会更新R 的）Schnorr-Euchner 算法中，我们可以通过对 R 赋值为$R = \lambda_1(L_{[j,k]})$并设等式(1)中$R = \|\mathbf{b}_j^*\|$ 来给出深度为 d 的节点个数的大致范围。由[10]可以看出，对于典型的约化基，$H_d(R)$在中深度 $d \approx (k-j)/2$ 附近最大，d 不在此范围时$H_d(R)$明显较小。

3.3 结果分析

BKZ 的时间复杂度目前还没有比较精确的上界。（对枚举子算法的）调用数量已知的最佳上界是指数级的（参见[14]）。在实践中（参见[9]），BKZ 在$\beta = 20$时是非常实用的，但是当 $\beta \geq 25$ 时，运行时间显著增加，使得选取$\beta \geq 40$在高维格开销过大。在实践中，BKZ 输出的基的质量优于理论的最佳最差情况边界：根据[9],高维格的 Hermite 因子通常是$c(\beta, n)^n$。$c(\beta, n)$似乎在n趋于正无穷时收敛速度很快，而理论上界是$c'(\beta)^n$，且$c'(\beta)$显著大于$c(\beta, n)$。例如,对于大n，$c(20, n) \approx 1.0128$。此外，近期[14]表明,如果在 BKZ 运行了合适的多项式次数后中止，推出的理论上界仍然只是略小于$c'(\beta)^n$。

4 BKZ 2.0

众所周知[9]，分块大小足够大时（≥ 30 时）BKZ 的总体运行时间基本由枚举子算法决定，这子算法能够找到一个m维局部投影格$L_{[j,k]}$的最短向量，枚举半径 R 初始化为$\|\mathbf{b}_j^*\|$,其中 $1 \leq j \leq k \leq n$ 且 $m = k - j + 1$。

在本节中，我们将描述 BKZ 2.0。这是 BKZ 的升级版，相比之前提到的 BKZ 有四个改进。我们通过修改 NTL[44]的 BKZ[42]实现了这些改进。第一个改进是简单的早期中止，这在密码分析中是很常见的做法，并且最近的成果[14] 给这种做法带来了一定的理论依据。早期中止的做法是这样的：我们添加一个参数来指定应该执行多少次迭代，即我们预测每次枚举的调用次数。这就已经给 BKZ 的复杂度带来了一个指数级优化，因为根据[9]的实验，调用的次数对于选定的$\beta \geq 30$似乎呈指数级增长。其他三个改进旨在减少枚举子算法的运行时间：深度剪枝[10]、局部基的预处理和更短的枚举半径。虽然这些改进可能是众所周知的，但我们要强调的是，目前 BKZ 的实现中并没有用到这些改进（除了 Schnorr 和 Hörner[43]设计的一种较弱的剪枝算法在 NTL[44]中实现了），而且实现它们绝非易事。

4.1 深度剪枝(Sound Pruning)

剪枝就是通过丢弃枚举树的某些分支从而加速枚举，但剪枝后可能不会返回任何向量，或者可能不会返回最短的向量。剪枝枚举的思想最早可以追溯到 Schnorr 和 Euchner 的研究[42]， Schnorr 和 Hörner[43]于 1995 年首次对其进行了分析。Gama 等人[10]最近重新研究了这个问题，他们注意到对[43]的分析是有缺陷的，因此剪枝不是最优的。他们证明了一个精心选择的高概率剪枝可以在完全枚举的基础上提高$2^{m/4}$的渐近速度，并引入了一种极限剪枝技术，可以在完全枚举的基础上提高$2^{m/2}$的渐近速度。我们使用随机化处理将这两种剪枝结合起来。形式上，剪枝以$\|\pi_{k+1-d}(\mathbf{u})\| \leq R_d \cdot R$取代了这 $k-j\ +1$ 个不等式：$\|\pi_{k+1-d}(\mathbf{u})\| \leq R,\ 1 \leq d \leq k-j+1$ 其中 $0 \leq R_1 \leq \cdots \leq R_{k-j+1} = 1$ 是由剪枝策略定义的 $k-j+1$ 个实数。对于任意边界函数$(R_1, \ldots, R_{k-j+1})$，[10]考虑N'和p_{succ}的定义如下：

- $N' = \sum_{d=1}^{k-j+1} H'_d$ 是对剪枝枚举树中节点数的启发式估计，其中$H'_d = \frac{1}{2}\frac{R^d V_{R_1,\ldots,R_d}}{\prod_{i=k+1-d}^{k} \|\mathbf{b}_i^*\|}$ 且 $V_{R_1,\ldots,R_d}$ 是 $C_{R_1,\ldots,R_d} = \{(x_1, \ldots, x_d) \in \mathbb{R}^d, \forall 1 \leq i \leq d,\ \sum_{l=1}^{i} x_l^2 \leq R_i^2\}$ 的体积。

- $p_{\text{succ}} = p_{\text{succ}}(R_1, \ldots, R_m) = \Pr_{\mathbf{u} \sim S^{m-1}}\left(\forall i \in [1, m], \sum_{l=1}^{i} u_l^2 \leq R_i^2\right)$。设$t \in L_{[j,k]}$为满足 $\|\pi_j(\mathbf{t})\| = R$ 的目标向量。如果局部基$B_{[j,k]}$是随机的，那么在（理想化的）假设下（即用对局部基$B_{[j,k]}$标准正交化后得到的 Gram-Schmidt 基$(\mathbf{b}_j^*/\|\mathbf{b}_j^*\|, \ldots, \mathbf{b}_k^*/\|\mathbf{b}_k^*\|)$表示出$\pi_j(\mathbf{t})$的坐标后，$\|\pi_j(\mathbf{t})\|$近似地服从均匀分布时），$p_{\text{succ}}$即为$\pi_j(\mathbf{t})$是剪枝后的枚举树叶子的概率。

我们强调上面所说的只是一种理想情况。在实践中，当m很小时，对于局部块$B_{[j,k]}$中不可忽略的部分，“$B_{[j,k]}$的一个向量是$L_{[j,k]}$的最短向量”这个事件的概率应该为零。考虑到 BKZ 的应用场景，针对不同的p_{succ}来设置不同的边界函数(bounding functions)是有意义的，比如p_{succ}取值为 1%到 95%，但是与此同时要注意尽可能让代价 N' 小一些。基于[10]的方法，我们进行了自动搜索来生成这样的边界函数，块大小β为 $35 \sim 90$，步长为 5, p_{succ}范围为 $1\% \sim 95\%$。

需要注意的是，BKZ 调用格$L_{[j,k]}$上的枚举子算法，而格$L_{[j,k]}$的维数$m = k - j + 1$ 并不一定等于β。当 $j \leq n - \beta + 1$ 时，分块大小m等于β，但当$j \geq n - \beta$ 时，分块大小m严格小于β。为了避免给每个维度都生成边界函数，我们决定在这种情况下根据为β找到的那些边界函数进行插值拟合，并检查这种拟合对p_{succ}的影响大不大（我们希望影响不大）。最后，为了提高p_{succ}，我们添加了一个可选参数ν，使得 BKZ 实际执行ν次剪枝枚举，每个都从同一个局部块的不同的随机基开始。这相当于[10]的极限剪枝。

4.2 局部块的预处理

枚举的开销与局部基的质量紧密相关，特别是当分块大小增加时：局部基越小，局部投影格$L_{[k-d+1,k]}$的体积就越大，因此在枚举树的最大填充深度中的节点越少。这一点是众所周知的。不过考虑到 BKZ 每一轮都会改进基的质量，有人可能会认为在枚举前没必要对局部基进行约化。然而：

- 对于每次枚举，虽然整个基的约化程度是有可能高于 LLL 约化的，但只能保证得到的局部基是 LLL 约化基。

- 在较大的分块大小中，大多数枚举都是成功的：能够得到比分块中第一个向量更短的向量。这意味着将执行局部 LLL 格基规约，以从生成集获得基：参见算法 1 的第 1 行。在下一次迭代中，枚举往往会在一般的 LLL 约化基上进行，而不是在约化程度更好的基上进行。

这表明，对于大多数枚举，得到的局部基一般仅仅是 LLL 约化基，即使在枚举过程中其他局部基可能会被更好地约化。这一点得到了实验的证实。

因此，我们实现了一个简单的加速：确保在每次枚举之前，局部基的约化程度比 LLL 约化高，但不会花费太多时间。在枚举之前，我们对局部基使用递归的中止 BKZ 预处理：我们根据β自动寻找一个合适的参数。

4.3 优化枚举半径

众所周知，枚举成本也会受到初始半径R的选择的影响，尽管该半径在枚举过程中会更新。最初，枚举半径是$R = \|\mathbf{b}_j^*\|$，但是如果我们事先知道输出向量有多短，我们会选择一个较小的初始半径R，从而减少枚举时间。实际上，枚举树深度d处的节点数与R^d成正比（不论剪枝与否）。不幸的是，除了一般的界限

外，我们对关于$\lambda_1(L_{[j,k]})$应该有多小的问题（从理论上看）知之甚少。因此，我们进行了实验，以查看在实践中通过枚举找到的最终范数是什么：图 1 比较了一轮 BKZ 的最终范数(通过枚举找到)与$GH(L_{[j,k]})$，这取决于局部块的起始索引 j。对于最低的指数 j，可以看到最终范数明显低于$GH(L_{[j,k]})$，而对于最大的指数，其最终范数明显大于$GH(L_{[j,k]})$。在占大多数枚举的中间，最终范数和 Gaussian Heuristic 预测之间的比率大多在 0.95 到 1.05 之间，而第一个局部基向量的范数和$GH(L_{[j,k]})$之间的比率通常略低于 1.1。因此，我们使用了以下优化：对于除最后 30 个索引之外的所有索引 j，我们设$R = min(\sqrt{\gamma}GH(L_{[j,k]}), \|\mathbf{b}_j^*\|)$。而不是$R = \|\mathbf{b}_j^*\|$，其中$\gamma$是半径参数。实际上，我们选择了$\sqrt{\gamma} = \sqrt{1.1} \approx 1.05$。

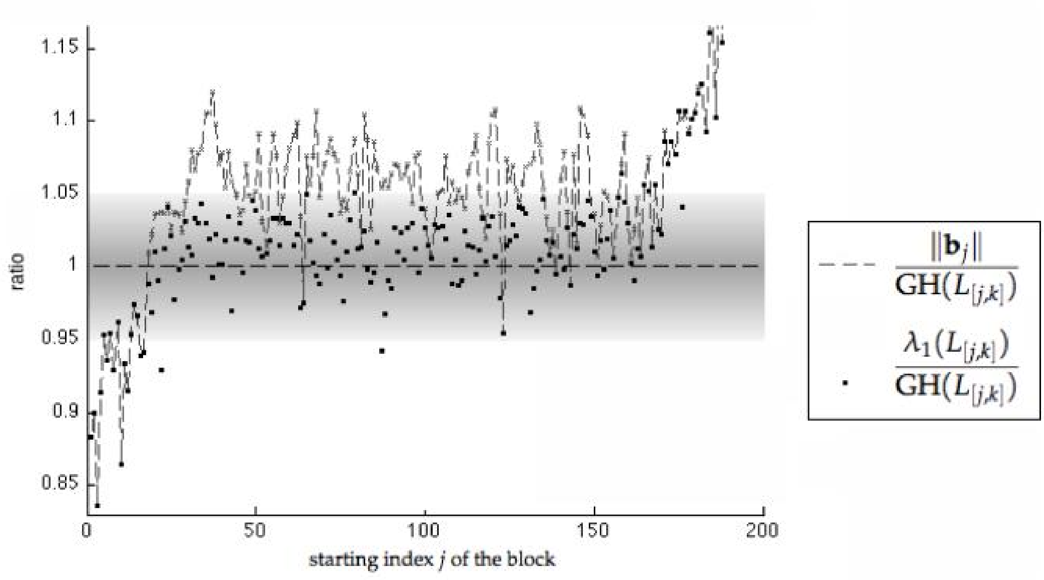

图 1 对于每个局部块$B_{[j,k]}$，比较$\|\mathbf{b}_j^*\|$，$\lambda_1(L_{[j,k]})$和$GH(L_{[j,k]})$的结果

5 格算法的新纪录

在这里，我们简要报告了使用 64 位 Xeon 处理器打破一些格算法记录的实验，这表明 BKZ 2.0 是目前实际应用中最好的格基规约算法。

5.1 Darmstadt 的格挑战问题

Darmstadt 格挑战问题于 2008 年提出。对于每个维度，完成挑战就是要在 Ajtai 格[2]中找到范数$< q$的向量，其中 q 取决于维度；并尽量使范数最小化。到目前为止，完成的最难挑战是 725 维：第一个解决 575 到 725 维挑战的方案是由 Gama 和 Nguyen 在 2008 年发现的，他们使用 NTL 的 BKZ 和 SH 剪枝实现了 BKZ。自那以后又找到了更短的解决方案（见完整列表[24]），但没有人完成更高维度的挑战。所有人的求解方法都是通过约化更小维度（通常在 150-200 左右）的合理选取的子格的基来找到的，这些子格的存在源于 Ajtai 格的结构。我们也使用了同样的策略。

Dim(lattice)	Dim(sublattice)	New norm	Previous norm	Ratio	Hermite factor
800	230	120.054	Unsolved		1.00978^{230}
775	230	112.539	Unsolved		1.00994^{230}
750	220	95.995	Unsolved		1.0976^{220}
725	210	85.726	100.90	0.85	1.00978^{210}
700	200	78.537	86.02	0.91	1.00993^{200}
675	190	72.243	74.78	0.97	1.00997^{190}
650	190	61.935	66.72	0.93	1.00993^{190}
625	180	53.953	59.41	0.91	1.00987^{180}
600	180	45.420	52.01	0.87	1.00976^{180}
575	180	39.153	42.71	0.92	1.00977^{180}
550	180	32.481	38.29	0.85	1.00955^{180}
525	180	29.866	30.74	0.97	1.00990^{180}

表 1 Darmstadt 格挑战的新纪录

分块大小为 90 的 BKZ 2.0（18 次成功概率为 5%的剪枝枚举）找到了针对挑战 750、775 和 800 的第一个解，并且在所有挑战 525 到 725 中的向量显著缩短，总共使用了大约 3 个核心年，如表 1 所总结的：第一列是挑战的维度，第二列是我们用来寻找解的子格的维度，第三列是 BKZ 2.0 找到的最佳范数，第四列是以前算法找到的最佳范数，第五列是我们用来寻找解的子格的维度，第六个是子格约化基的 Hermite 因子，它略低于1.01^{dim}。2008 年，Gama 和 Nguyen[9]认为1.01^{dim}中的系数是当时最先进的极限，这表明了确实有提升。

5.2 SVP 挑战

SVP 挑战赛[40]于 2010 年 5 月开始。挑战赛中的格L是大体积的随机整格，因此$\lambda_1(L) \approx GH(L)$的概率很大。这个挑战是要找到一个几乎最短的向量，即范数$\leq 1.05GH(L)$的非零格向量。我们使用分块大小为 75，成功概率为 20%的剪枝的 BKZ 2.0，成功解决了从 90 维到 112 维的所有挑战。

6 利用模拟算法预测 BKZ 2.0

现在，我们提出了一种有效的模拟算法来预测 BKZ 2.0 在维数高、分块大小较大($\beta \geq 50$)下的性能。从运行时间的角度和输出质量来看，我们的模拟与在 64 位 Xeon 处理器上使用几个核心年的随机格和 Darmstadt 格挑战上的实验相当一致。因此我们相信，我们提出的模拟可以用来大致预测使用（比我们在实验中所使用的设备计算能力强很多的）高算力设备能做到什么，从而得出更令人信服的安全性估计。

6.1 模拟算法描述

我们的模拟算法的目标是在执行 BKZ 的过程中，（更准确地说是在每轮开始时）对 Gram-Schmidt 序列 $(\mathbf{b}_1^*, \mathbf{b}_2^*, ..., \mathbf{b}_n^*)$ 进行预测。在算法 1 的步骤 1 中，每当 $j=0$ 时，BKZ 算法就会开始新的一轮。所以 BKZ 的每一轮实际上调用了 $n-1$ 次枚举。我们假设输入基是一个“随机”的约化基，没有特殊性质。

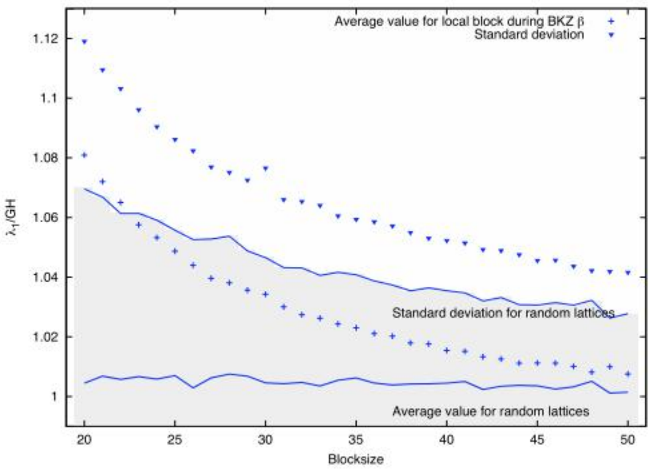

图 2 比较 $\frac{\lambda_1(L)}{GH(L)}$ 在 BKZ-β约化中不使用极限剪枝的局部块中，和在维数为β的随机格中的大小。图上给出了有标准差和无标准差的期望。

我们进行模拟的出发点是基于 4.3 节的直觉，即大多数局部块的第一个最小值看起来像随机格：这在分块大小≤ 30 时不成立（如 Gama 和 Nguyen[9]所指出的），但随着分块大小的增加，它变得越来越接近实际情况，如图 2 所示，其中我们看到$\frac{\lambda_1(L)}{GH(L)}$的预期和标准偏差似乎收敛到随机格的预期和标准偏差。直观地

讲，这可以用集中现象来解释：随着维数的增加，随机格在格的集合中占据主导地位。因此除非有充分的理由说明我们不能认为给定的格是随机的，我们就总可以就假定它的行为接近随机格。一旦我们能够预测每个局部块的$\lambda_1(L_{[j,k]})$的值，根据枚举子算法可知这将是$\|\mathbf{b}_j^*\|$的新值，我们就能预测下一局部块的体积，并由此迭代这一过程直到回合结束。这就有了我们的模拟算法（参见算法 2）。

Algorithm 2. Simulation of BKZ reduction

Input: The Gram-Schmidt norms, given as $\ell_i = \log(\|\mathbf{b}_i^*\|)$, for $i = 1, \ldots, n$, a blocksize $\beta \in \{45, \ldots, n\}$, and a number N of rounds.

Output: A prediction for the Gram-Schmidt norms $\ell'_i = \log(\|\mathbf{b}_i^*\|)$, $i = 1, \ldots, n$, after N rounds of BKZ reduction.

for $k = 1, \ldots, 45$ **do**
  $r_k \leftarrow$ average $\log(\|\mathbf{b}_k^*\|)$ of an HKZ-reduced random unit-volume 45-dim lattice
end for
for $d = 46, \ldots, \beta$, **do** $c_d \leftarrow \log(GH(\mathbb{Z}^d)) = \log(\frac{\Gamma(d/2+1)^{1/d}}{\pi^{1/2}})$ **end for**
for $j = 1, \ldots, N$ **do**
  $\phi \leftarrow$ true //flag to store whether $L_{[k,n]}$ has changed
  for $k = 1$ to $n - 45$ **do**
    $d \leftarrow \min(\beta, n-k+1)$ // Dimension of local block
    $f \leftarrow \min(k+\beta, n)$ //End index of local block
    $\log V \leftarrow \sum_{i=1}^{f} \ell_i - \sum_{i=1}^{k-1} \ell'_i$
    if $\phi =$ true **then**
      if $\log V/d + c_d < \ell_k$ **then**
        $\ell'_k \leftarrow \log V/d + c_d$;
        $\phi \leftarrow$ false
      end if
    else
      $\ell'_k \leftarrow \log V/d + c_d$
    end if
  end for
  $\log V \leftarrow \sum_{i=1}^{n} \ell_i - \sum_{i=1}^{n-45} \ell'_i$
  for $k = n-44$ to n **do**
    $\ell'_k \leftarrow \frac{\log V}{45} + r_{k+45-n}$
  end for
  $\ell_{1,\ldots,n} \leftarrow \ell'_{1,\ldots,n}$
end for

算法 2　BKZ 约化模拟算法

我们以如下方式预测$L_{[j,k]}$中最短非零向量的范数$\lambda_1(L_{[j,k]})$：

- 对于大多数指标j，除非$\|\mathbf{b}_j^*\|$比$GH(L_{[j,k]})$小，否则我们以$GH(L_{[j,k]})$作为其预测值。
- 然而，对于最后一轮的索引j，即最后一个β维块$L_{[n-\beta+1,n]}$内的索引，我

们做了一些不同的事情：由于这最后一个块将在回合结束时被 HKZ 约化，我们假设它的行为类似于与$L_{[n-\beta+1,n]}$等体积随机格的 HKZ-约化基。由于这些平均值对于较大的 β计算开销可能比较大，因此我们应用了一个简化的规则：我们用随机 45 维单位体积格的 HKZ 约化基的平均 Gram-Schmidt 范数（通过实验计算）来近似后 45 个 Gram-Schmidt 范数，并使用 Gaussian Heuristic 计算前$\beta-45$ 个 Gram-Schmidt 范数。但是这个模型可能不适用于某些特殊结构的基，例如 NTRU Hermite 标准型的部分约化，这也是我们只分析输入为随机约化基这种情况的原因。

此模拟算法允许我们在给定任意分块大小的情况下猜测 BKZ 2.0 实现的近似 Hermite 因子，如表 2 所示：对于给定的维度n，应运行模拟算法，因为实际的分块大小也取决于该维度。如第 2 节所述，Hermite 因子决定了解决与密码学相关的格问题的性能：近似 SVP 和唯一 SVP 见[9]，SIS 和 LWE 见[28，39，23]。显然，我们只能希望得到一个近似值，因为当输入基数被随机化时，Hermite 因子存在众所周知的变化。在我们知道枚举子算法的成本的情况下，模拟算法还使用轮数为我们提供了大约的运行时间：我们稍后将更精确地讨论这些点。

Target Hermite Factor	1.01^n	1.009^n	1.008^n	1.007^n	1.006^n	1.005^n
Approximate Blocksize	85	106	133	168	216	286

表 2 模拟预测的高维 BKZ 的近似所需分块大小

6.2 与实验的一致性

结果表明,我们的模拟与使用随机格和 Darmstadt 格挑战的实验结果很符合。首先，就随机约化基的 Gram-Schmidt 序列 $(\mathbf{b}_1^*, \mathbf{b}_2^*, ..., \mathbf{b}_n^*)$而言，我们的模拟算法是相当准确的，如图 3 所示。这意味着我们的模拟算法可以很好地预测 BKZ 在任意某轮时的 Hermite 因子，图 4 证实了这一点。此外，图 4 表明，多项式的调用次数似乎足以算出与完全约化差距不大的 Hermite 因子。主要的约化似乎发生在 BKZ 的前几轮，这说明使用中止 BKZ 是合理的。这是对目前 BKZ 研究的一点补充。图 4 表明，对于随机约化基，我们的模拟算法是相当准确的，这意味着我们的模拟算法可以很好地预测 BKZ 的 Hermite 因子，图 4 证实了这一点

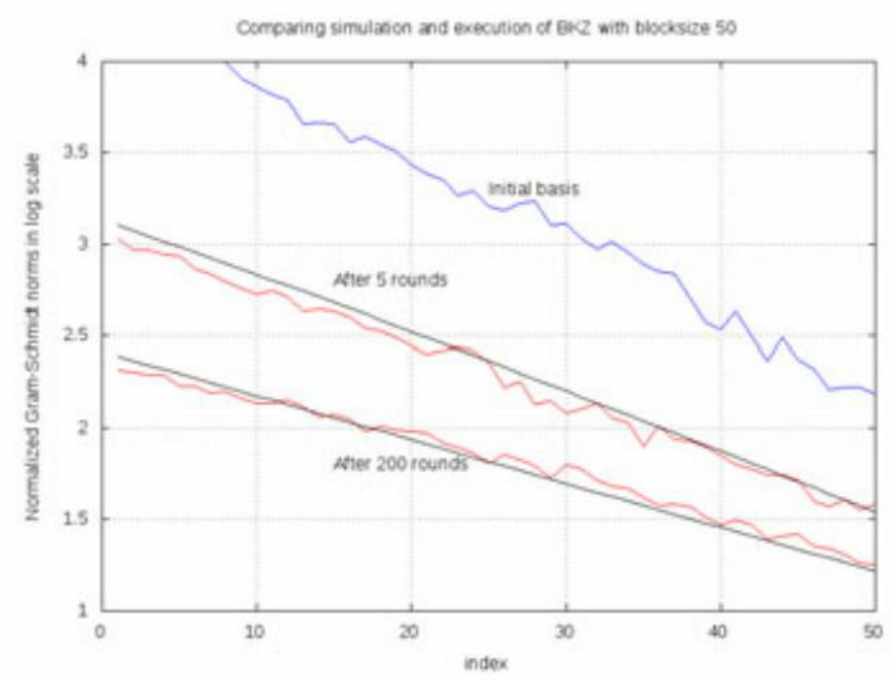

图 3　200 维随机格 BKZ-50 约化过程中 Gram-Schmidt 范数的预测值与实际值

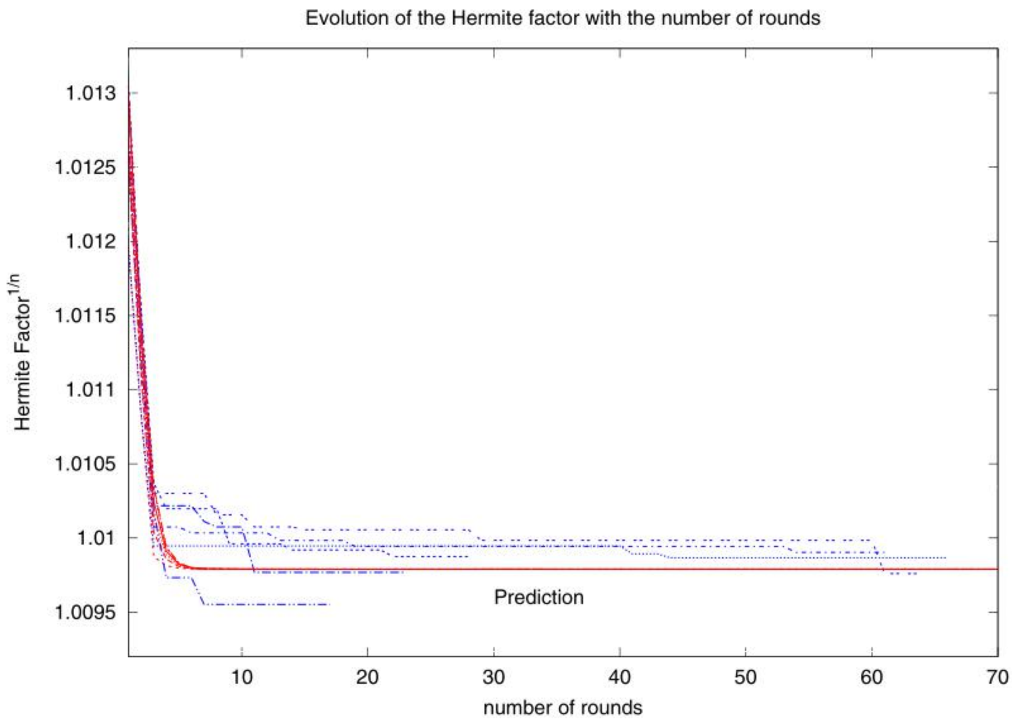

图 4 使用 BKZ-90 约化完成 180 维 Darmstadt 格挑战第 500 到 625 挑战时，$(\|\mathbf{b}_1\|/\mathrm{vol}(L)^{1/n})^{1/n}$ 的实际值和预测值

6.3 枚举子算法

我们还需要估计枚举半径为 Gaussian Heuristic 的枚举子算法的成本。首先，我们按照[10]的搜索方法，对使用了 BKZ 2.0 进行约化的基上应用极限剪枝来计算上限。表 3 给出了使用 BKZ-75-20%作为预处理，半径等于 Gaussian Heuristic 的分块大小为 100-250 的极限剪枝的近似成本（用节点数的对数来表示）。节点数可以近似地换算为时钟周期，方法如下：在[10]的实现中，一个节点需要大约 200 个时钟周期进行双精度枚举，但这个数字取决于维度，对于分块大小较大的情况，我们可能需要比双精度更高的精度。例如，表 3 显示，在分块大小为 120 的情况下应用极限剪枝最多需要大约 253 个节点，假设精度加倍，这在 1.86 GHz Xeon 上不到 30 个核心年。这对于确定深度攻击的参数非常有用。然而，这些上限并非是严格的，这是因为枚举技术的性能取决于预处理。通过更好的预处理（包括具有不同参数的 BKZ 2.0）很可能会获得更好的数据（与表 3 相比）。

Blocksize	100	110	120	130	140	150	160	170	180	190	200	250
BKZ-75-20%	41.4	47.1	53.1	59.8	66.8	75.2	84.7	94.7	105.8	117.6	129.4	204.1
Simulation of BKZ-90/100/110/120	40.8	45.3	50.3	56.3	63.3	69.4	79.9	89.1	99.1	103.3	111.1	175.2

表 3 枚举子算法开销的上限，使用带有中止的 BKZ 预处理的极限剪枝。成本以 $\log_2$(节点数)表示。

事实上表 3 还提供了一个更好的上界(这是基于我们对 BKZ 模拟的结果得到的，模拟中使用分块大小为 90-120 的 BKZ 约化作为预处理）。为了提供具有良好安全边际的安全估计，我们需要估计可以取得多大进展。有趣的是，枚举技术有其局限性。Nguyen[33]在假设 Gaussian Heuristic 很好地估计了节点数的前提下，对枚举树的每个深度的节点数建立了一个下界(这是分析枚举技术复杂性的常用手法）。下界基于格的 Rankin 不变量$\gamma_{n,m}(L)$

$$\gamma_{n,m}(L) = \min_{\substack{S \text{ sublattice of } L \\ \dim S = m}} \left(\frac{\mathrm{vol}(S)}{\mathrm{vol}(L)^{m/n}} \right)$$

特别地，[33]证明了半径为$GH(L)$的d维格L的完全计数的中间深度的节点数是$\geq V_{d/2}(1)\sqrt{\gamma_{d,d/2}(L)/V_d(1)}$的。对于典型的格$L$，其 Rankin 不变量$\gamma_{n,m}(L)$非常接近于 Rankin 常数$\gamma_{n,m}$的下界（见[7]）：

$$\gamma_{n,m} \geq \left(n \frac{\prod_{j=n-m+1}^{n} Z(j)}{\prod_{j=2}^{m} Z(j)} \right)^{\frac{2}{n}}$$

其中$Z(j) = \zeta(j)\Gamma\left(\frac{j}{2}\right)/\pi^{\frac{j}{2}}$，$\zeta$是黎曼 zeta 函数：$\zeta(j) = \sum_{p=1}^{\infty} p^{-j}$。这些下界适用于完全枚举，并且可以据这些下界和剪枝加速效果来得到剪枝枚举的节点数下界（如[10]中所分析的）。表 4 分别给出了实战中使用线性剪枝和理论上以渐进加速

倍率为$2^{n/2}$进行估计的极限剪枝得到的下界数据。与表 3 的上限相比，有一个很大的差距：线性剪枝的下限告诉了我们，如果为枚举算法找到更强大的预处理，可以有多大的提升。最后，我们注意到，Sieve 算法[3]的启发式变体[36,30,45]基本上比剪枝枚举更快。然而，目前还不清楚它对安全估计的意义有多大，因为这些变体需要指数空间，而且在实践中表现更好。而且需要比[36,30]更多的实验来精确评估它们的实际运行时间。但是我们的模型可以很容易地适应枚举算法中的新进展，这要归功于表 2。

Blocksize	100	120	140	160	180	200	220	240	280	380
Linear pruning	33.6	44.5	56.1	68.2	80.7	93.7	107.0	120.6	148.8	223.5
Extreme pruning	9	15	21.7	28.8	36.4	44.4	52.8	61.5	79.8	129.9

表 4 根据[33, 10]，使用线性剪枝或极限剪枝的枚举子算法的成本（在日志节点中）的下界

7 对安全性估计进行修正

在这里，我们将说明如何用我们的模拟算法获得更精确的安全性估计。

7.1 NTRU 格

在 NTRU 密码系统[18]中，用公钥计算私钥相当于在具有特殊结构的高维格中寻找最短向量。因为 NTRU 安全评估是基于 BKZ 的基准，所以看到这种方法的局限性是很有趣的。在原始文献[18]中，最小参数集 NTRU-107 对应于 214 维格，估计密钥恢复至少需要 250 次基本运算。通过直接使用格基约化算法（不使用像[25, 26, 9]这样利用 NTRU 格特殊结构的特别技术）恢复 NTRU-107 密钥的最佳实验结果是 1999 年 5 月的[25]，其中记载了一次使用 BKZ 结合 SH 剪枝的成功实验[43]，在 200 MHz 处理器上进行了 663 小时，即 248.76 个时钟周期。我们用 BKZ 2.0 在 10 个随机的 NTRU-107 格上进行了实验：我们应用了 LLL 和 BKZ-20，最多只需要几分钟；我们应用了 BKZ-65，进行了 5%的剪枝，并每 5 分钟检查第一个基向量是否是与密钥对应的最短向量，在这种情况下我们终止算法。BKZ 2.0 对于每个格都是成功的，在 2.83 MHz 的单核上，BKZ-65 的失败平均只用了不到 2000 年的时间。因此，总体运行时间不到 40 分钟，即 242.62 个时钟周期，与 5 月的实验相比，这至少加速了 70%，而且明显低于 250 次基本操作。因此，最初的安全预估2^{50}和实际安全级别（最多约为 40 比特）之间差了一个数量级。现在，我们再来回顾一下 NTRU 签名的最新参数。Hoffstein 等人在最近的一篇文章中[17]

总结了 NTRU 加密和签名的最新参数。特别地，NTRU 签名的最小参数是$(N, q) = (157,256)$，据声称有针对所有已知攻击的 80 位安全性，以及针对密钥恢复格攻击的 93 位安全性。类似于[9]，我们估计在 $2N = 314$ 维格中恢复范数 $< q$ 的向量实质上与找到密钥一样困难，体积q^N，对应于1.008862^N的 Hermite 因子。我们对这些参数运行了我们的模拟算法，以根据分块大小从 BKZ-20 约化基（其成本在这里可以忽略不计）开始猜测需要多少轮：大约 6 轮的 BKZ-110 应该足以破解 157 维的 NTRU 签名，这相当于大约2^{11}次枚举。根据表 3，分块大小为 110 的极限剪枝枚举可以通过搜索至多2^{47}个节点来完成，这大约对应于一般的处理器上的2^{54}个时钟周期。这表明最小的 NTRU 签名参数抵御最先进的格攻击的安全级别最多是 65 位而不是 93 位，这两个数据差距很大。

7.2 Gentry-Halevi 的全同态加密挑战

我们现在转向 Gentry-Halevi 的主要全同态加密挑战[11]，没有给出具体的安全性估计。解密密文相当于求解一个 BDD 问题实例，这可以使用 Babai 的最近平面算法来完成，最远距离为 $\min_i \|\mathbf{b}_i\|^*/2$。以给定的 $\min_i \|\mathbf{b}_i\|^*$为目标可以转化为对偶格中的目标 Hermite 因子。这允许我们基于 BDD 实例和格体积的近似距离来估计求解 BDD 实例所需的 Hermite 因子，如表 5 所示。

Dimension n	512	2048	8192	32768
Name	Toy	Small	Medium	Large
Target Hermite factor [9]	1.67^n	1.14^n	1.03^n	1.0081^n
Algorithm expected to decrypt a fresh ciphertext	LLL	LLL	LLL	BKZ with blocksize ≈ 130
Time estimate	30 core-days	$\leq$ 45 core-years	$\leq$ 68582 core-years	$\approx 2^{100}$ clock-cycles

表 5 Gentry-Halevi 面临的主要挑战的安全评估[11]

据此我们推测，就玩具模型、小挑战和中型挑战而言，其解密可以使用 LLL 约化来解决。但由于格的维度很大且格基比特长度巨大，这并不简单。（请注意，基矩阵中元素很大的 LLL 约化有新的理论进展[37]）我们使用 fplll[4]的修改版进行了实战约化，从而验证了玩具挑战确实是这种情况。对于中小型挑战，我们根据截断的挑战推断运行时间，利用我们对 fplll 的修改具有启发式运行时间$O(N^3d^2)$的事实，其中d是格体积的比特长度，O中的常数取决于浮点精度(随着维度的增加)。根据我们的模拟，完成大型挑战将需要分块大小的 LLL-130 和大约 60000 轮（从≈开始），即 231 个枚举调用。根据表 3，此枚举程序最多花费 260 个节点，因此大型挑战提供的安全性最多约为 100 位。另一方面，如果发现了更强大的枚举预处理，据表 4 的数据来预测，安全估计可能要除以2^{10}–2^{40}范围内的一个因子。

译者致谢

我要感谢首都师范大学的朱子阳。没有他的帮助，我无法准确翻译出第 2 节的一些数学名词。此外还要感谢南京邮电大学的王少辉老师为我提供了 BKZ 2.0 论文这一翻译选题，翻译这篇论文让我对格基规约算法了解许多。