自己双対型 PotBKZ 基底簡約の提案と BKZ との比較

◎ 佐藤 新[1]　安田 雅哉[1]

[1] 立教大学

2025 年 1 月 29 日（水）

研究背景

格子暗号の安全性評価と基底簡約アルゴリズム

- **格子暗号は耐量子計算機暗号の一つ（ML-KEM, ML-DSA, Falcon）**
  - 安全性は SVP や CVP などの格子問題の求解困難性に基づく
  - 格子暗号の安全性評価には，格子問題の求解困難性評価が必要
- **基底簡約は格子問題の求解に必須の技術**
  - LLL [7] $\rightsquigarrow$ 格子次元に関して多項式時間計算量
  - BKZ [9] やその改良 $\rightsquigarrow$ ブロックサイズに関して指数時間計算量
    - ★ BKZ は強力な基底簡約アルゴリズムで，安全性解析における標準ツール
    - ★ LLL と異なり，BKZ の停止性は理論的には保証されてない
    - ★ 実用上，実行時間や繰り返し（ツアー）回数に上限を設けて強制停止 (`fplll`[2] 内の BKZ オプションとして `max_time` や `max_loops` がある)

本研究の目的

停止性を保証する BKZ の変種の提案と BKZ との実験比較

1. **停止性を保証する BKZ[9] の変種 PotBKZ を提案**
   - 基底に対して定まるポテンシャル量を利用
     - ★ ポテンシャル量は LLL[7] の多項式時間での停止性を保証する量
   - PotBKZ ではポテンシャル量を単調減少させる基底操作のみを行う

   $\Longrightarrow$ 多項式回のツアー回数で停止することが保証できる
2. **PotBKZ の様々な変種を開発＋実験比較**
   - 双対型や自己双対型の PotBKZ まで開発
   - BKZ と実験比較：BKZ とのツアー回数，出力基底の品質を比較

数学的準備：格子の基礎

定義 1 (格子の GSO ベクトルと体積)

- 一次独立な $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n$ の整数係数の一次結合全体

$$L = \mathcal{L}(\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n) := \left\{ \sum_{i=1}^{n} a_i \boldsymbol{b}_i \,\middle|\, a_1, \ldots, a_n \in \mathbb{Z} \right\}$$

を**格子**，$\{\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n\}$ を**基底**，n を**次元**と呼ぶ．

$$\boldsymbol{b}_1^\star := \boldsymbol{b}_1,\ \boldsymbol{b}_i^\star := \boldsymbol{b}_i - \sum_{j=1}^{i-1} \mu_{i,j} \boldsymbol{b}_j^\star,\ \mu_{i,j} := \langle \boldsymbol{b}_i, \boldsymbol{b}_j^\star \rangle / \|\boldsymbol{b}_j^\star\|^2$$

で定義される $\boldsymbol{b}_1^\star, \ldots, \boldsymbol{b}_n^\star$ を **GSO ベクトル**，$\mu = (\mu_{i,j})$ を **GSO 係数行列**と呼ぶ．

- 格子 L の**体積**を

$$\mathrm{vol}(L) = \prod_{i=1}^{n} \|\boldsymbol{b}_i^\star\|$$

と表し，<u>これは基底にとり方によらない</u>

数学的準備：基底簡約の概要

- **基底簡約**：長く平行に近い基底を短く直交に近い基底に変換する操作
  - 格子の基底は無数に存在
  - 良い基底の方が SVP や CVP などの格子問題が求解しやすい
  - 代表的なアルゴリズム：LLL[7]，BKZ[9]

$$\begin{bmatrix} 87349 & 0 & 0 & 0 \\ 83474 & 1 & 0 & 0 \\ 98247464 & 0 & 1 & 0 \\ 847984 & 0 & 0 & 1 \end{bmatrix} \xrightarrow[\text{基底簡約}]{} \begin{bmatrix} -8 & 8 & -1 & 3 \\ -7 & -9 & 14 & 4 \\ -2 & -3 & -11 & 16 \\ 8 & 6 & 12 & 17 \end{bmatrix}$$

悪い基底
基底のノルムが長くほぼ平行

良い基底
基底のノルムが短くほぼ直交

ポテンシャル量とLLLアルゴリズム

基底 $\{\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n\}$ のポテンシャル量

$$\mathrm{Pot}(\{\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n\}) := \prod_{k=1}^{n} \|\boldsymbol{b}_k^\star\|^{2(n-k+1)} > 0$$

LLL アルゴリズム [7]

- 次の条件を満たす基底を見つける
  1. $|\mu_{i,j}| \leq 1/2$
  2. $\delta \cdot \left\|\boldsymbol{b}_{k-1}^\star\right\|^2 \leq \|\pi_{k-1}(\boldsymbol{b}_k)\|^2$
- 特に，条件 2 を満たさない $(\boldsymbol{b}_k, \boldsymbol{b}_{k+1})$ を交換（Step 7）
  - ポテンシャルが δ 倍ずつ減少

    $\Longrightarrow$ **多項式時間で停止**

Algorithm LLL 基底簡約アルゴリズム [7]

Require: 格子 L の基底 $\{\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n\}$
 定数 $1/4 < \delta < 1$
Ensure: δ に関して LLL 簡約された基底

```
1: k ← 2
2: while k ≤ n do
3:     基底をサイズ簡約 [6]
4:     if 条件 2 を満たす then
5:         k ← k + 1
6:     else
7:         swap(b_k, b_{k+1})
8:         k ← max{k − 1, 2}
```

DeepLLL アルゴリズムと PotLLL アルゴリズム

基底に対する deep-inserion という操作

$$\sigma_{k,\ell}(\{\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n\}) = \{\ldots, \boldsymbol{b}_{k-1}, \boldsymbol{b}_\ell, \boldsymbol{b}_k, \ldots, \boldsymbol{b}_{\ell-1}, \boldsymbol{b}_{\ell+1} \ldots\}$$

DeepLLL アルゴリズム [9]

- 次の条件を満たす基底を見つける
  1. $|\mu_{i,j}| \leq 1/2$
  2. $\delta \cdot \|\boldsymbol{b}_i^\star\|^2 \leq \|\pi_i(\boldsymbol{b}_k)\|^2$
- 条件 2 を満たさない基底を deep-insertion で入れ替え
  - ポテンシャルは増えたり減ったり
    $\Longrightarrow$ **指数時間**

PotLLL アルゴリズム [3]

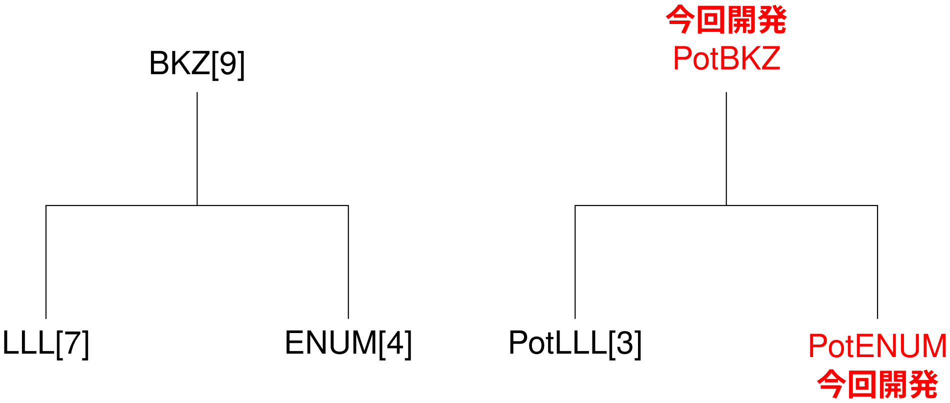
- 次の条件を満たす基底を見つける
  1. $|\mu_{i,j}| \leq 1/2$
  2. $\delta \cdot \mathrm{Pot}(\boldsymbol{B}) \leq \mathrm{Pot}(\sigma_{k,\ell}(\boldsymbol{B}))$
- 条件 2 を満たさない基底を deep-insertion で入れ替え
  - $\mathrm{Pot}(\boldsymbol{B})$ を下げる deep-insertion のみ
  - ポテンシャルが単調減少
    $\Longrightarrow$ **多項式時間で停止**

PotENUM：PotLLL の一般化

- $\mathrm{Pot}(\boldsymbol{B})$ を減少させる格子ベクトル $\boldsymbol{v} = \sum_{i=j}^{m} v_i \boldsymbol{b}_i$ $(v_m = 1)$ の数え上げ
  - $\boldsymbol{C} = \{\ldots, \boldsymbol{b}_{j-1}, \boldsymbol{v}, \boldsymbol{b}_j, \ldots, \breve{\boldsymbol{b}}_m, \ldots\}$：$j$ 番目に $\boldsymbol{v}$ を挿入+$\boldsymbol{b}_m$ を除去
  - $\mathrm{Pot}(\boldsymbol{C}) < \mathrm{Pot}(\boldsymbol{B})$
    （cf., PotLLL[3] では deep-insertion $\sigma_{j,m}$ で得られる基底に限定）
- 従来の ENUM[4] と PotENUM の比較（$D_i := \|\pi_i(\boldsymbol{v})\|$, $B_i := \|\boldsymbol{b}_i^\star\|$）

	ENUM[4]	PotENUM
$\boldsymbol{v}$ の条件	$D_j < B_j$	$D_j^j \prod_{i=j+1}^{m-1} D_i^2 < \delta \prod_{i=1}^{m-1} B_i$
探索方法	$D_j < R^2$	$D_j < R_j^2$, $R_j^2 = \left(\dfrac{\delta B_1 \cdots B_{m-1}}{D_{k+1} \cdots D_{m-1}}\right)^{1/j}$
挿入による効果	$\|\boldsymbol{b}_j^\star\|$ を減少	$\mathrm{Pot}(\boldsymbol{B})$ を減少

PotBKZ（1/2）：BKZ との構成の差異



- PotBKZ では BKZ の LLL を PotLLL に，ENUM を PotENUM に置き換え
- PotLLL と PotENUM はどちらもポテンシャルを単調減少
  - PotENUM が呼ばれる回数（ツアー回数）は**格子次元に関して多項式的**

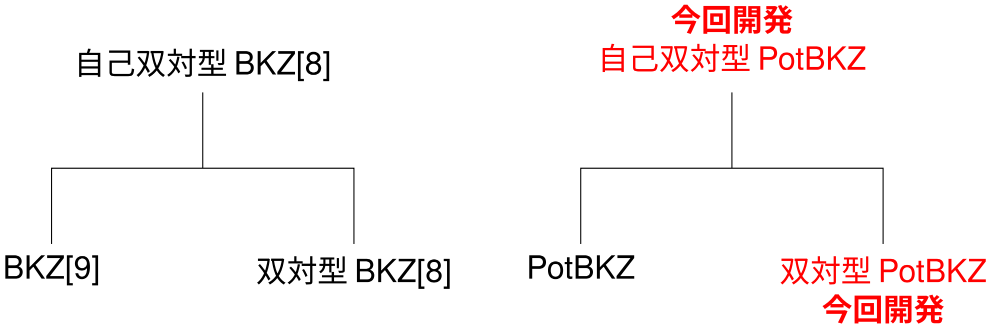
PotBKZ（2/2）：アルゴリズムの概要

Algorithm PotBKZ 基底簡約アルゴリズム

Require: n 次元格子の基底 $\{\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n\}$，簡約変数 $1/4 < \delta < 1$
Ensure: 簡約された基底 $\{\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n\}$

$\{\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n\} \leftarrow \texttt{PotLLL}(\{\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n\}, \delta)$ /* BKZ[9] では LLL[7] を利用*/
$z \leftarrow 0$
while $z < n - 1$ **do**
  $j \leftarrow (j \bmod n - 1) + 1; k \leftarrow \min(j + \beta - 1, n)$
  $\boldsymbol{v} \leftarrow \texttt{PotENUM}(\pi_j(\boldsymbol{b}_j), \ldots, \pi_j(\boldsymbol{b}_k))$ /* BKZ では ENUM[4] で最短ベクトルを探索*/
  if no solution **then**
    $z \leftarrow z + 1$
  else
    $\{\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n\} \leftarrow \texttt{MLLL}(\{\ldots, \boldsymbol{b}_j, \boldsymbol{v}, \boldsymbol{b}_{j+1}, \ldots\}, \delta)$ /* MLLL[9] で一次独立性を除く*/
    $\{\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n\} \leftarrow \texttt{PotLLL}(\{\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n\}, \delta)$
    $z \leftarrow 0$

自己双対型 PotBKZ：自己双対型 BKZ との構成の差異

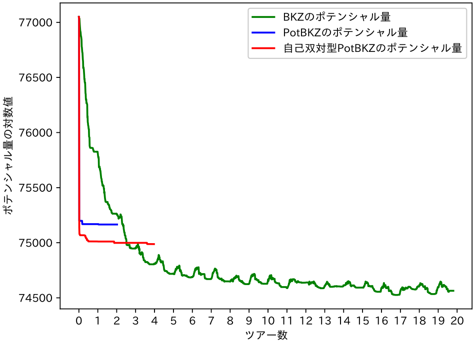


- 自己双対型 PotBKZ は PotBKZ とその双対型を**交互に呼び出す**
- 双対型 PotBKZ を開発
  - 双対基底を求めるには**逆行列計算**が必要
  - 双対型 PotENUM⇝<u>双対基底への挿入</u>でポテンシャルが減少するベクトルの数え上げ
  - **逆行列計算が不要**

実験結果（1/3）：実装方法と実験方法

- **実装方法**
  - 実装は総て `C++`言語で `NTL` ライブラリ [10] と `Eigen` ライブラリ [5] を使用
  - `g++`でコンパイル
  - コンパイルオプションは`-O3 -mtune=native -march=native -mfpmath=both`
  - 基底は `long` 型，GSO 情報は `long double` 型を使用
- **実験方法**
  - SVP-challenge[1] のシード 0〜9 で実行
  - **ツアー回数，ポテンシャル量の対数値，GSA スロープの傾き**の平均をとる
    - ★ GSA スロープの傾きは良い基底を判断する指標の一つ（小さい方が良い）

実験結果 (2/3)：100 次元のポテンシャル量の変化



- **PotBKZ** は 1 ツアー目で一気にポテンシャルが低くなり，その後 2 ツアーで**停止**
- **自己双対型 PotBKZ** も同様の挙動を取り 4 ツアーで**停止**
  - PotBKZ より小さいポテンシャル量をとる
- 一方，**BKZ**[9] は緩やかに減少し，その後 7 ツアー目頃から**停滞**を続ける

実験結果 (3/3)：

- **実験結果**
  - Intel Core i7-1355U @ 1.70 GHz 上で実験
  - ブロックサイズは $\beta = 40$ を選択
  - PotBKZ は約 2 回程度のツアー回数で停止
  - 自己双対型 PotBKZ も約 5 回程度のツアー回数で停止
    - ⋆ PotBKZ よりも**低い GSA スローブの傾きとポテンシャル量の対数値の比をとる**
  - 一方, BKZ[9] と比較すると高く，**十分に下げられていない**

格子階数	BKZ[9]			PotBKZ			自己双対型 PotBKZ		
	ツアー回数	$\log \mathrm{Pot}$ の比	$-\rho$	ツアー回数	$\log \mathrm{Pot}$ の比	$-\rho$	ツアー回数	$\log \mathrm{Pot}$ の比	$-\rho$
100	≥ 20	0.9668	0.0553	1.9	0.9735	0.0615	4.8	0.9714	0.0595
110	≥ 20	0.9658	0.0561	1.9	0.9722	0.0617	5.2	0.9697	0.0594
120	≥ 20	0.9616	0.0558	1.7	0.9685	0.0612	5.9	0.9665	0.0597

まとめ

- **PotBKZ と自己双対型 PotBKZ の開発と実験**
  - ポテンシャルの単調減少により停止性の保証された BKZ[9] の変種
  - 基底への挿入でポテンシャルが減少するベクトルを数え上げる PotENUM を開発
  - その双対版，自己双対版を開発
  - 120 次元格子での実験結果
    - ⇝ PotBKZ は **2 ツアー程度**，自己双対型 PotBKZ は **5 ツアー程度**で停止
    - ⇝ 一方 BKZ は **20 ツアー以上を要する**
- **今回の実験結果から得られた知見**
  - PotBKZ や自己双対型 PotBKZ は BKZ ほどポテンシャルを下げれない
    - ★ より大きなブロックサイズを試せばBKZ 程度の品質が出せると期待
    - ★ より大きなブロックサイズの利用のため，**枝切り**や**篩**の適用をしたい

PotBKZ とその変種の公開ソースコード

- 本研究の成果となるソースコードを GitHub にて公開
  https://github.com/satoshin-des/self-dual-PotBKZ
- ファイル構成

```
├── src
│    │
│    └── 各種ソース（SelfDualPotBKZ.h，PotBKZ.h など）
├── svp_challenge_list
│    │
│    └── テストケースとして svp challenge[1] の基底（LLL 済）
├── libSDPotBKZ.so
│     ⇝C++ファイルをコンパイルした共有ライブラリ
└── excample.py
      ⇝libSDPotBKZ.so を python で動かすテストコード
```

参考文献 I

[1] TU Darmstadt.
SVP challenge.
Available at https://www.latticechallenge.org/svp-challenge/.

[2] The FPLLL development team.
fpylll, a Python wrapper for the fplll lattice reduction library, Version: 0.6.1.
Available at https://github.com/fplll/fpylll, 2023.

[3] Felix Fontein, Michael Schneider, and Urs Wagner.
PotLLL: A polynomial time version of LLL with deep insertions.
Designs, Codes and Cryptography, 73:355–368, 2014.

参考文献 II

[4] Nicolas Gama, Phong Q Nguyen, and Oded Regev.
Lattice enumeration using extreme pruning.
In Advances in Cryptology–EUROCRYPT 2010, volume 6110 of Lecture Notes in Computer Science, pages 257–278. Springer, 2010.

[5] Gaël Guennebaud, Benoît Jacob, et al.
Eigen v3.
`http://eigen.tuxfamily.org`, 2010.

[6] C. Hermite.
Extraits de lettres de M. Ch. Hermite à M. Jacobi sur différents objects de la théorie des nombres.
Journal für die reine und angewandte Mathematik, 40:261–277, 1850.

参考文献 III

[7] Arjen Klaas Lenstra, Hendrik Willem Lenstra, and László Lovász.
Factoring polynomials with rational coefficients.
Mathematische Annalen, 261(4):515–534, 1982.

[8] Daniele Micciancio and Michael Walter.
Practical, predictable lattice basis reduction.
In Advances in Cryptology–EUROCRYPT 2016, volume 9665 of Lecture Notes in Computer Science, pages 820–849. Springer, 2016.

[9] Claus-Peter Schnorr and Martin Euchner.
Lattice basis reduction: Improved practical algorithms and solving subset sum problems.
Mathematical programming, 66:181–199, 1994.

参考文献 IV

[10] Victor Shoup.
NTL: A Library for doing Number Theory.
http://www.shoup.net/ntl/.