SCIS 2026 2026 Symposium on
Cryptography and Information Security
Hakodate, Japan, Jan. 26 – 30, 2026
The Institute of Electronics,
Information and Communication Engineers

準同型暗号 CKKS 方式における 3 次元配列表現を用いた行列積の構成
Matrix Multiplication Using 3-Dimensional Array Representation in CKKS Homomorphic Encryption Scheme

磯川 楽丸 * 若杉 飛鳥 † 服部 大地 † 小寺 雄太 *
Tanoshimaru Isokawa Asuka Wakasugi Daichi Hattori Yuta Kodera
野上 保之 *
Yasuyuki Nogami

あらまし 準同型暗号は，データの漏洩リスクを低減しプライバシーを保護しつつ機密情報を解析したいという要求に応える技術であり，暗号化されたデータに対して直接計算を施すことを可能とする．準同型暗号における演算は平文に比べて著しく高コストであるため，実際のアプリケーションで広く活用するのは困難である．特に行列積は AI モデルの推論や学習において中核的な演算である一方，$d \times d$ 行列の積には通常 d^3 回の乗算が必要となる．このため，暗号文状態における行列積の高速化が求められる．本稿では，Mahon らが提案した BFV 方式における 3 次元配列表現を用いる行列積の高速化手法を CKKS 方式へ拡張し，その実装と検証を行う．本手法は，1 つの暗号文で表現できる d の最大値が制限されるという課題はあるが，演算コストに関しては，**Add** や **Rot** を $O(\log(d))$，**cMult** や **Mult** を $O(1)$ に削減する．実験では，Jiang らの手法と実行時間と精度の比較を行い，評価したすべての d において本手法が高速であり，特に d の増加に伴ってその効果が顕著となることを確認した．誤差は Jiang らの手法とほぼ同等であり，本手法の有効性を実証した．

キーワード 準同型暗号，CKKS，行列積

1 はじめに

多くの企業が，データに基づいた意思決定を重視するようになっている．高度なデータ解析の専門知識や技術に乏しい企業や組織では，自身の会社や組織内のデータ解析を外部へ委託することで，新しい洞察を得ようとする場合がある．しかし，医療や金融などの分野では，機密性の高いデータも含まれるため，そのまま外部へ共有することは容易ではない．そこで，データの漏洩リスクを低減し，プライバシーを保護しつつ機密情報を解析する方法が求められている．準同型暗号は，このような要求に応える技術であり，暗号化されたデータに対して復号を行うことなく直接計算を施すことを可能とする．これにより，外部へ機密データを直接共有することなくデータを分析し，外部からの知見を得ることが可能になる．

しかし，準同型暗号における演算は平文に比べて著しく高コストであるため，準同型暗号を広く応用するには，暗号化されたデータに対する演算処理の高速化が不可欠である．特に，行列演算は，機械学習モデルの推論や学習などの場面において必要な演算であり，その高速化は重要な課題である．

本研究で着目した行列積は，行列演算の中でも特に計算量が大きく，通常の $d \times d$ 行列の行列積では，d^3 回の乗算が必要となる．加えて，一般に，準同型暗号における暗号文同士の乗算は，平文同士の乗算に比べて著しく高コストである．このような状況から，準同型暗号における行列積を高速化することは，実用的な処理時間で演算を可能にするために極めて重要な課題である．

1.1 先行研究

準同型暗号を用いた行列積の高速化に関する先行研究として，以下のような取り組みが報告されている．まず，2017 年に Lu ら，2019 年に Wang ら [9, 14] によって暗号化された 2 つの行列間の行列積計算手法が実現され

* 岡山大学大学院 環境生命自然科学研究科, 〒 700-8530 岡山県岡山市北区津島中 3-1-1. Graduate School of Environmental, Life, Natural Science and Technology, Okayama University, 3-1-1 Tshushima-naka, Kita-ku, Okayama-shi, Okayama, 700-8530, Japan.
† EAGLYS 株式会社, 〒 151-0051 東京都渋谷区千駄ヶ谷 5 丁目 27-3 やまとビル 7F. EAGLYS Inc., 7F Yamato building, 5-27-3 Sendagaya, Shibuya-ku, Tokyo, 151-0051, Japan.

表 1: 手法ごとの CKKS 暗号文演算の実行回数と深さの比較

Method	**Rot**	**cMult**	**Mult**	Depth
[9, 14]	$O(d^2 \log d)$	$O(d^2)$	$O(d^2)$	1cMult + 1Mult
[2]	$O(d \log d)$	$O(d)$	$O(d)$	1cMult + 1Mult
[8]	$O(d)$	$O(d)$	$O(d)$	2cMult + 1Mult
[7, 12]	$O(d)$	$O(d)$	$O(d)$	1cMult + 1Mult
[10]	$O(\log d)$	$O(\log d)$	1	2cMult + 1Mult

た．しかし，それらは $d \times d$ 暗号文行列を表現するために d 個の暗号文が必要であった．2018 年には，Cheon ら [2] によって行列全体を単一の暗号文で表現する効率的な行列積計算手法を提案した．この手法では，一方の行列から対角線成分を抽出し，他方の行列を回転させることで，2 つの暗号化された行列の積を計算する．同年，Jiang ら [8] も行列を単一の暗号文で表現する手法を採用し，新たな行列積計算アプローチを導入した．我々の研究グループ [15] は，Jiang らの手法に対して，詳細な実装設計および誤差評価を与えた．更に 2022 年には，Jang ら [7] や Rizomiliotis ら [12] によって Jiang らのアルゴリズムを改良するアプローチが提案されている．2025 年に Mahon ら [10] が，行列を 3 次元構造としてエンコードし，3 次元回転（row/col/square rotations）を用いて部分積を効率に配置・集約する手法を提案した．多項式の長さ N に対して，行列サイズ d が $d^3 \leq N/2$ を満たす必要があるため，Jiang らの手法と比較して，1 暗号文で表現できる行列サイズが制限される．しかし，従来 $O(d)$ であった **Add** 及び **Rot** は $O(\log d)$ に抑えられ，**cMult** を 3 回，**Mult** を 1 回に削減した．

以上の先行研究に関して，1 回の行列積に必要とされる演算回数を**表** 1 に示す．**表** 1 の Depth とは，行列積 1 回あたりの，1 つの暗号文に対して実行される **cMult** と **Mult** の回数である．また，上記の研究の流れとは別に，Park[11] は，大規模行列に対する暗号文行列積アルゴリズムを提案した．さらに，Gentry ら [5] は，暗号文行列演算に適した，CKKS ライクな新しい準同型暗号方式を提案した．

1.2 目的と構成

準同型暗号には，BFV [4] や TFHE[3]，CKKS[1] などのいくつかの方式が知られているが，本稿では，CKKS 方式による実装を行う．CKKS 方式では，実数や複素数を平文として扱えるため，他 2 方式よりも平文の定義域が広い特徴を持つ．また，TFHE 方式では，暗号文同士の乗算を行うのに，Bootstrap という演算が必要である．一方，BFV/CKKS 方式における暗号文乗算は同一のアルゴリズムであり，Bootstrap 処理は不要である．よって，一般に，BFV/CKKS 方式の方が TFHE 方式よりも乗算の実行時間が速い．

本稿では，Mahon らの行列積の高速化手法 [10] を CKKS 方式で実装し，その性能を評価する．Mahon らの論文では BFV 方式によって実装されているが，CKKS 方式で実装することで平文の定義域が広がり，より多様なデータに対応しつつ，暗号文に対する行列積を高速化することを目的とする．Mahon らの手法は Jiang らの手法に比べて演算コストが低いため，本稿は，我々の研究グループ [15] よりも高速であることを実験的にも示す．

また，本稿では実行時間に加えて，暗号文行列積による計算結果の誤差も測定する．一方で，いくつかの先行研究では，計算結果の精度検証が明示されていないことがある．CKKS 方式では，行列積などの複雑な演算を繰り返すことで誤差が累積し，最終的な計算結果へ影響を及ぼす可能性がある．よって，誤差検証を行うことは計算結果の信頼性を確保する上で非常に重要である．

本稿は次のように構成される．第 1 章では，準同型暗号の概要を説明し，行列積の高速化に関する先行研究を紹介し，Mahon らの高速化手法に着目した背景を述べた．第 2 章では，CKKS 方式に関する詳細と，本稿で用いる用語を解説する．第 3 章では，提案アルゴリズムの概要を示し，その演算コストについて議論する．第 4 章では，実行時間や精度を計測し得られた結果について考察する．最後に，第 5 章で本研究の結論を述べる．

1.3 貢献

本稿では，Mahon ら [10] が提案した BFV 方式における 3 次元配列表現を用いた行列積の高速化手法を CKKS 方式へ対応させたアルゴリズムを与える．そして，そのアルゴリズムの実装を Jiang ら [8] の手法と性能比較した結果を示す．なお，Jiang らの手法は，我々の研究グループ [15] が与えた実装設計を基に実装を行う．結果として，行列サイズ $d = 4, 8, 16$ において，本手法は，Jiang らの手法より誤差はわずかに大きいものの，実行時間はより高速であり，実行時間の削減率は d の増大に伴い顕著であった．これは，行列積 1 回に必要な CKKS 暗号文演算の実行回数が，Jiang らの手法では全演算が $O(d)$ であるのに対し，本手法では **Add** や **Rot** が $O(\log(d))$，**cMult** や **Mult** が $O(1)$ であるためだと考えられる．なお，本手法は $d \times d$ 行列の行列積に少なくとも d^3 のスロットを用いるため，1 つの暗号文で表現できる行列サイズの最大値が制限されるが，$d^3 \leq N/2$ を満たす範囲内においては顕著な高速化効果を発揮する．

以上より，Mahon らの手法を CKKS 方式へ対応させたアルゴリズムと，その性能に関する評価を提供している．

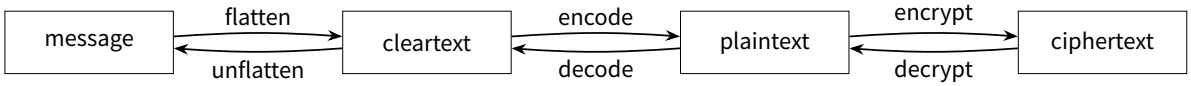

図 1: message, cleartext, plaintext, ciphertext 間の相互変換

2 CKKS 方式

本章では，CKKS 方式の概要と本稿で用いる用語について説明する．

本稿では，入力データ，暗号化アルゴリズムにおける平文，暗号化アルゴリズムにおける引数，暗号文をそれぞれ message, cleartext, plaintext, ciphertext と表記する．これらのデータ間の相互変換による対応関係を**図 1** にて示す．message は，複素数からなる 1 次元配列もしくは 2 次元配列とし，cleartext は長さ $N/2$ の固定長複素 1 次元配列として定義される．plaintext は，長さ N の整数係数多項式であり，ciphertext は，長さ N の整数係数多項式の組として定められる．

message は，長さ ℓ_1 の 1 次元配列を長さ ℓ_2 だけもつような 2 次元配列とし，全体の長さ $n = \ell_1\ell_2$ に対して，$n \leq N/2$ を満たすものとする．ここで，$\ell_2 = 1$ のときに限り，message を 1 次元配列とみなす．message から cleartext へ変換するアルゴリズムを flatten，その逆変換を unflatten と定める．flatten は，$1 \leq i_1 \leq \ell_1, 1 \leq i_2 \leq \ell_2$ に対して，message の (i_1, i_2) 成分を cleartext の $(i_2 - 1)\ell_1 + i_1$ 番目の要素へ移す操作である．$n < N/2$ の場合，cleartext の残りの要素は全て 0 とする．

CKKS 方式における plaintext は，次数 N の整数係数多項式環の要素として表現されるが，$N/2$ 個の複素数体の直積への順序を保存する同型写像が存在するため，この多項式は $N/2$ 次元の複素数ベクトルと同一視される．このとき，この同型対応により各複素数に対応付けられるような，多項式の係数の組をスロットと呼ぶ．encode は，逆離散フーリエ変換（IDFT）に相当する線形写像を用いて，cleartext の各要素を plaintext のスロットへ埋め込むアルゴリズムとして定義される [1]．本稿では，cleartext の構成から，常に $N/2$ 個のスロットを持つことに注意されたい．encode の逆変換である decode は，plaintext から cleartext へ変換する操作として定まる．

また，ciphertext は Ring-LWE 構造を持ち，法 q における整数係数多項式環 $R_q = \mathbb{Z}_q[X]/(X^N + 1)$ 上の 2 本の多項式からなるベクトルとして表される [1]．ここで N は 2 べきの整数である．encrypt は，公開鍵 pk を用いて plaintext を ciphertext へ変換するアルゴリズムとして定義される．decrypt は，秘密鍵 sk を用いて ciphertext を plaintext へ変換するような，encrypt の逆操作として定められる．

CKKS 暗号文空間上では，5 つの演算 **Add**, **Sub**, **Mult**, **cMult**, **Rot** が定義される．**Add** は，暗号文同士の各多項式を法 q で加算する演算である．**Sub** は，暗号文同士の各多項式を法 q で減算する演算である．**Mult** は，暗号文同士の各多項式を法 q で乗算し，その後 relinearize と rescale と呼ばれるサブルーチンを行う演算である．**cMult** は，暗号文の各多項式と plaintext を法 q で乗算した後，rescale を行う演算である．**Rot** は，暗号文に対応するスロットをシフト量 j だけ左方向へ循環移動させる演算である．具体的には，i 番目のスロットは，$(i - j) \mod N/2$ 番目に移動する．

3 提案手法

本章では，準同型暗号 CKKS 方式における 3 次元配列表現を用いた行列積のメインアルゴリズム及び，そのサブルーチンについて説明する．なお，本手法は，Mahon ら [10] が提案した BFV 方式における 3 次元配列表現を用いた行列積の高速化手法を，サブルーチンを一部変更することで CKKS 方式へ対応させたものである．

3.1 サブルーチン

本節では，今後用いるサブルーチンである，**ShiftCols**, **ShiftRows**, **ShiftSqrs** を説明する．これらのアルゴリズムは，Mahon ら [10] の論文で提案された 3 次元回転操作を実現するものである．Mahon らの論文では，**SwapCols**, **SwapRows**, **SwapSqrs** という名称で提案されているが，CKKS 方式に対応させるため，操作を一部変更している．また，混同を避けるため，名称も変更している．

一般に，$d \times d$ 行列の行列積では，全体で d^3 個の乗算結果が生じる．Mahon らの手法では，これら d^3 個の要素をそれぞれ CKKS 暗号文のスロットへ格納する．そのため，cleartext においても，少なくとも d^3 個の成分を格納できることが必要である．本稿では，d, N を共に 2 冪とし，$N/2 > d^3$ を満たすと仮定する．このとき，d, N は共に 2 冪であるため，$N/2 \geq 2d^3$ が成り立つ．

例えば，$d = 4$ において，暗号文に対応する cleartext を $\{1, 2, \ldots, d^3, 0, \ldots, 0\}$ とする．cleartext の初めの d^3 個の要素のみを図示すると，**図 2** の Original のようになる．ここで，**ShiftCols** と **ShiftRows** は，暗号文に対する操作であり，それぞれ，暗号文に対応する cleartext の全正方形ブロックに対して列や行を i だけシフトさせる．また，**ShiftSqrs** は，暗号文に対応する cleartext

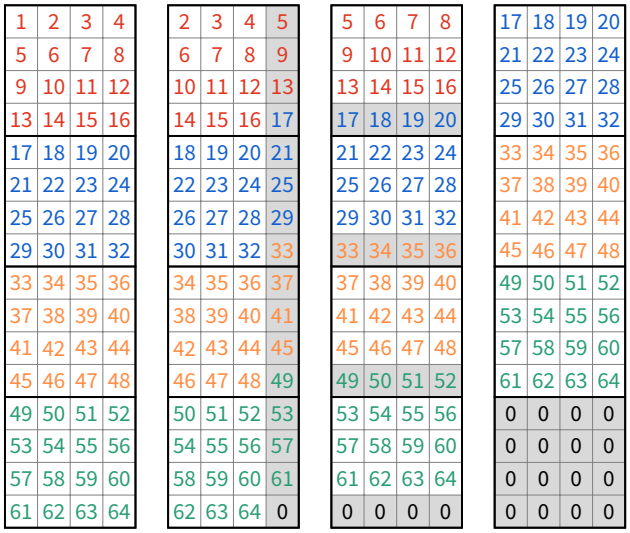

図 2: $d = 4, i = 1$ における **ShiftCols**，**ShiftRows**，**ShiftSqrs** の動作例

の正方形ブロックを i だけシフトさせる操作である．なお，正方形ブロックとは，太線で囲った $d \times d$ サイズの領域のことであり，1 つの暗号文に $N/2d^2$ 個存在する．

$d \times d$ 行列の暗号文をメインアルゴリズムに入力したとき，暗号文 c に対して，それぞれの操作は以下のように定められる：

$$\begin{aligned}\mathbf{ShiftCols}(c, i) &= \mathbf{Rot}(c, i)\\ \mathbf{ShiftRows}(c, i) &= \mathbf{Rot}(c, id)\\ \mathbf{ShiftSqrs}(c, i) &= \mathbf{Rot}(c, id^2)\end{aligned}$$

ただし，i は整数で，シフト量を表し，正の値の場合は左方向へのシフト，負の値の場合は右方向へのシフトを意味する．**図 2** に，$d = 4, i = 1$ の場合において，Original に対して各操作を適用した際の cleartext を示す．なお，図中の灰色部分は，のちにマスク処理が施される部分である．よって，例えば，図中の **ShiftCols** の場合では，Original の全ブロックの第 2–4 行目が，第 1–3 行目へシフトしたものとして解釈できる．

3.2 メインアルゴリズム

暗号文に対する行列積を実行するメインアルゴリズムを**アルゴリズム 1** に示す．このアルゴリズムは，任意の 2 つの $d \times d$ 行列 A', B' を message として順に flatten, encode, encrypt した暗号文 A, B に対して，行列積 $C = A \times B$ を計算する．つまり，C へ順に decrypt, decode, unflatten を適用して得られる行列 C' は，元の行列 A', B' の積の近似である．また，メインアルゴリズムのサブルーチンが，暗号文 A, B をそれぞれ中間的な暗号文 $\bar{A}, \bar{B}$ に変換し，それらから最終的な行列積を求める．

Algorithm 1 メインアルゴリズム

Input: A, B: 暗号文
Output: C: 暗号文
1: $\bar{A} \leftarrow$ **ComputeAbar**(A)
2: $\bar{B} \leftarrow$ **ComputeBbar**(B)
3: $C \leftarrow$ **ComputeC**$(\bar{A}, \bar{B})$
4: **return** C

3.3 中間暗号文 $\bar{A}, \bar{B}$ の構成

次に，暗号文 A を入力して，中間行列 $\bar{A}$ を出力するアルゴリズムの疑似コードは**アルゴリズム 2** に示す．2–4 行目では，$\bar{A}$ に対して **ShiftCols**，**ShiftSqrs** や列へのマスクを行う **cMult** を順に実行し，**図 3** に示すような変換を行う．ここで，マスクをかける列，行，正方形単位のインデックスを k とすると，マスク行列 $M_C(k)$，$M_R(k)$，$M_S(k)$ を以下のように定義する．

- $M_C(k)$: 各正方形の k 番目の列に対応する要素が 1，それ以外の要素が 0 である cleartext を encode して得られる plaintext
- $M_R(k)$: 各正方形の k 番目の行に対応する要素が 1，それ以外の要素が 0 である cleartext を encode して得られる plaintext
- $M_S(k)$: k 番目の正方形に対応する要素が 1，それ以外の要素が 0 である cleartext を encode して得られる plaintext

また，5–6 行目で **ShiftCols** を用いて $\bar{A}$ の列を複製することで，中間行列 $\bar{A}$ を生成する．これらの操作は，**図 4** にて示される．

同様に，暗号文 B を入力して，中間行列 $\bar{B}$ を出力するアルゴリズムの疑似コードは**アルゴリズム 3** で与えられる．$\bar{A}$ では A の列に関する操作に焦点を当てたが，$\bar{B}$ では B の行に関して操作を行う．2–4 行目では**図 5** に示す変換を行い，5–6 行目では**図 6** に示す操作が行われる．

3.4 最終暗号文 C の構成

最後に，$\bar{A}$ と $\bar{B}$ を用いて，C を構成する操作について説明する．このようなアルゴリズムの疑似コードは，**アルゴリズム 4** で与えられる．1 行目では，$\bar{A}$ と $\bar{B}$ の暗号文乗算を行う．2–4 行目では，C に対して，**ShiftSqrs** と **cMult** を行う．このような操作は，**図 7** にて示される．

Algorithm 2 ComputeAbar

Input: A: 暗号文
Output: $\bar{A}$: 暗号文

$\bar{A} \leftarrow A$
for $i = 0$ **to** $\log d - 1$ **do**
  $\bar{A} \leftarrow \mathbf{Add}(\bar{A}, \mathbf{ShiftSqrs}(\mathbf{ShiftCols}(\bar{A}, 2^i), -2^i))$
$\bar{A} \leftarrow \mathbf{cMult}(\bar{A}, M_C(0))$
for $i = 0$ **to** $\log d - 1$ **do**
  $\bar{A} \leftarrow \mathbf{Add}(\bar{A}, \mathbf{ShiftCols}(\bar{A}, -2^i))$
return $\bar{A}$

Algorithm 3 ComputeBbar

Input: B: 暗号文
Output: $\bar{B}$: 暗号文

$\bar{B} \leftarrow B$
for $i = 0$ **to** $\log d - 1$ **do**
  $\bar{B} \leftarrow \mathbf{Add}(\bar{B}, \mathbf{ShiftSqrs}(\mathbf{ShiftRows}(\bar{B}, 2^i), -2^i))$
$\bar{B} \leftarrow \mathbf{cMult}(\bar{B}, M_R(0))$
for $i = 0$ **to** $\log d - 1$ **do**
  $\bar{B} \leftarrow \mathbf{Add}(\bar{B}, \mathbf{ShiftRows}(\bar{B}, -2^i))$
return $\bar{B}$

$\bar{A}, \bar{B}$ および C を求めるのに必要な演算コストと乗算の深さを **表 2** に示す．

4 実験および結果

本章では，暗号文に対する行列積として，Jiang ら [8] の手法と**アルゴリズム 1** を実装し，実行時間を比較する．なお，Jiang らの手法は，我々の研究グループ [15] が与えた実装設計を基に実装を行う．また，出力暗号文の復号値と真の値である message に対する行列積との誤差を評価する．message に対する行列積とは，message を NumPy[6] の matmul 関数に入力して得られた出力値のことである．実験環境を**表 3**，CKKS パラメータを**表 4** に示す．秘密鍵の分布は ternary であり，パラメータは 128 bit security を満たす．なお，これらのパラメータは全て $N/2 \geq 2d^3$ を満たす．

まず，暗号文行列積の実行時間を測定する．具体的に

Algorithm 4 ComputeC

Input: $\bar{A}, \bar{B}$: 暗号文
Output: C: 暗号文

$C \leftarrow \mathbf{Mult}(\bar{A}, \bar{B})$
for $i = 0$ **to** $\log d - 1$ **do**
  $C \leftarrow \mathbf{Add}(C, \mathbf{ShiftSqrs}(C, 2^i))$
$C \leftarrow \mathbf{cMult}(C, M_S(0))$
return C

表 2: アルゴリズム毎の CKKS 暗号文演算の実行回数と乗算の深さ

	Add	**Rot**	**cMult**	**Mult**	深さ
$\bar{A}$	$2\log d$	$3\log d$	1	-	1 **cMult**
$\bar{B}$	$2\log d$	$3\log d$	1	-	1 **cMult**
C	$\log d$	$\log d$	1	1	1 **Mult** + 1 **cMult**
Total	$5\log d$	$7\log d$	3	1	1 **Mult** + 2 **cMult**

表 3: 実験環境

項目	詳細
機種名	MacBook Air (13-inch, 2025)
チップ	Apple M4
OS	macOS Tahoe 26.1
RAM	32 GB
SEAL[13]	4.1.2 (SEAL_USE_INTEL_HEXL=OFF)
Python	3.12.10

は，$2 \leq k \leq 4$ に対して $d = 2^k$ とし，$d \times d$ 行列に対して，Jiang らの手法と本手法それぞれで行列積を 1000 回ずつ行い実行時間の平均を求める．なお，実験において，message は 暗号文に変換されるが，その変換・逆変換時間は実行時間には含まれないものとする．測定した実行時間を CKKS 暗号文演算の実行回数 とともに**表 5** に示す．また，各 d に対して，本手法により得られる [15] からの実行時間の削減率を**表 6** に示す．

表 5 より，行列サイズ $d = 4, 8, 16$ いずれの場合においても，本手法は Jiang らの手法より高速であり，実行時間の削減率は d の増加に伴い顕著であった．これは，本手法における行列積 1 回に必要な CKKS 暗号文演算の実行回数が，Jiang らの手法よりも少ないためである．Jiang らの手法では，各演算の実行回数が $O(d)$ であるが，本手法では，表 2 より，**Add** と **Rot** が $O(\log(d))$ であり，**cMult** と **Mult** が $O(1)$ である．

次に，行列積の結果に対する誤差を確認する．定義域 $[-1, 1)$ の一様乱数を要素に持つ 2 つの $d \times d$ 行列 を message とする．message を暗号化した暗号文を入力とし，Jiang らの手法と本手法を用いた行列積のそれぞれの場合に対して，message に対する行列積との誤差の最大値および最小値を計測する．以上の操作を 1000 回繰

表 4: CKKS パラメータ

パラメータ	値
polynomial modulus degree N	16384
coefficients modulus	{ 60, 40, 40, 40, 60 }
scale	2^{40}
noise standard deviation	3.19

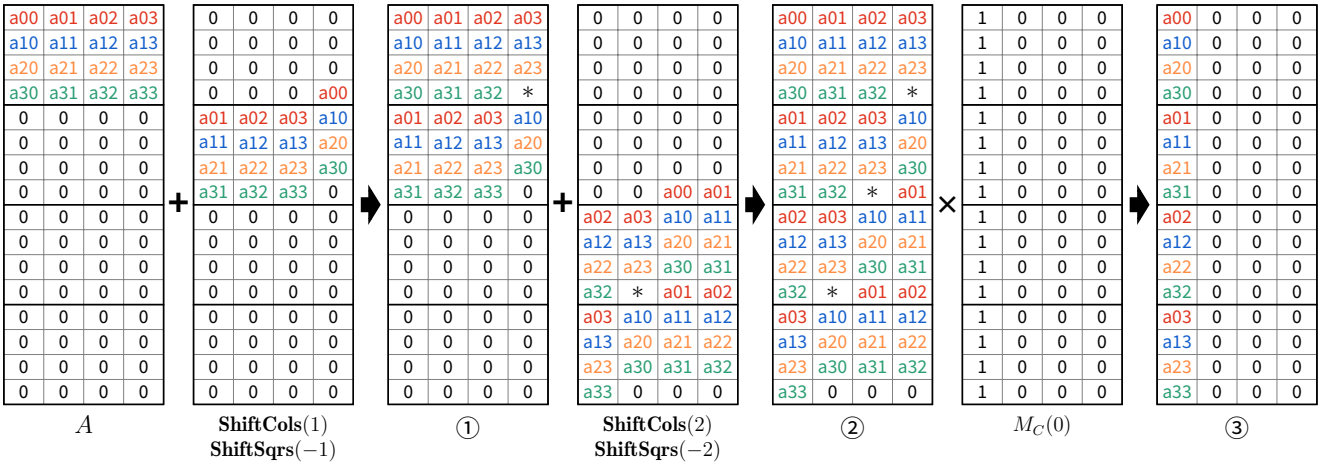

図 3: **ComputeAbar** (1)

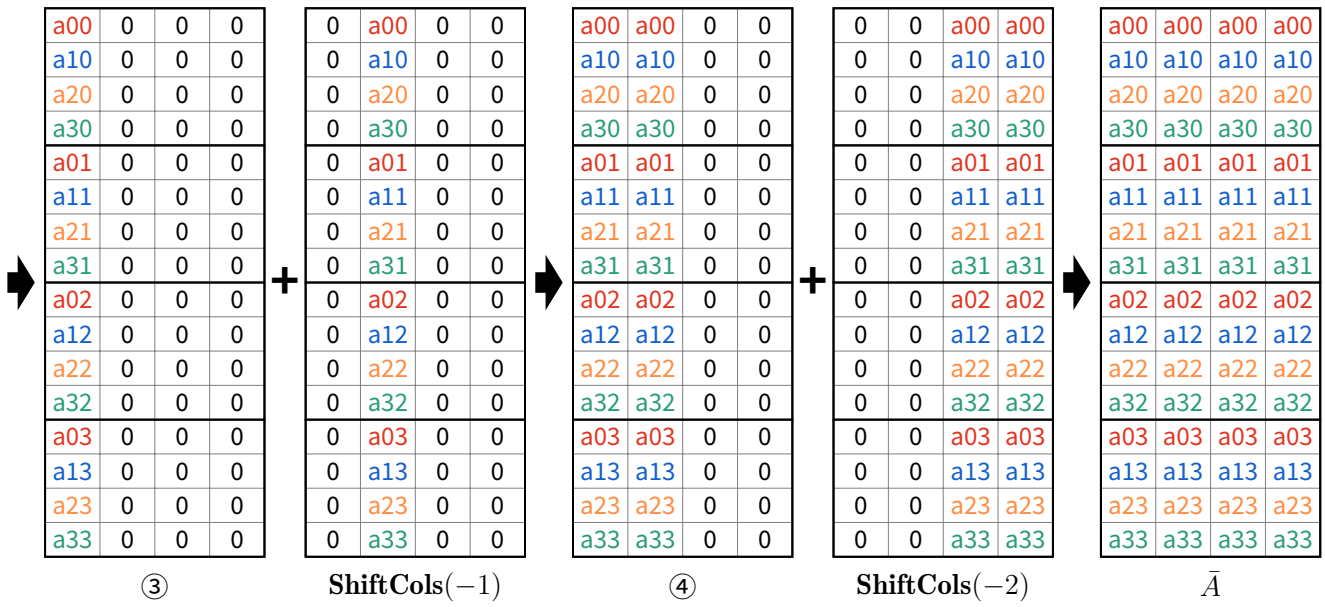

図 4: **ComputeAbar** (2)

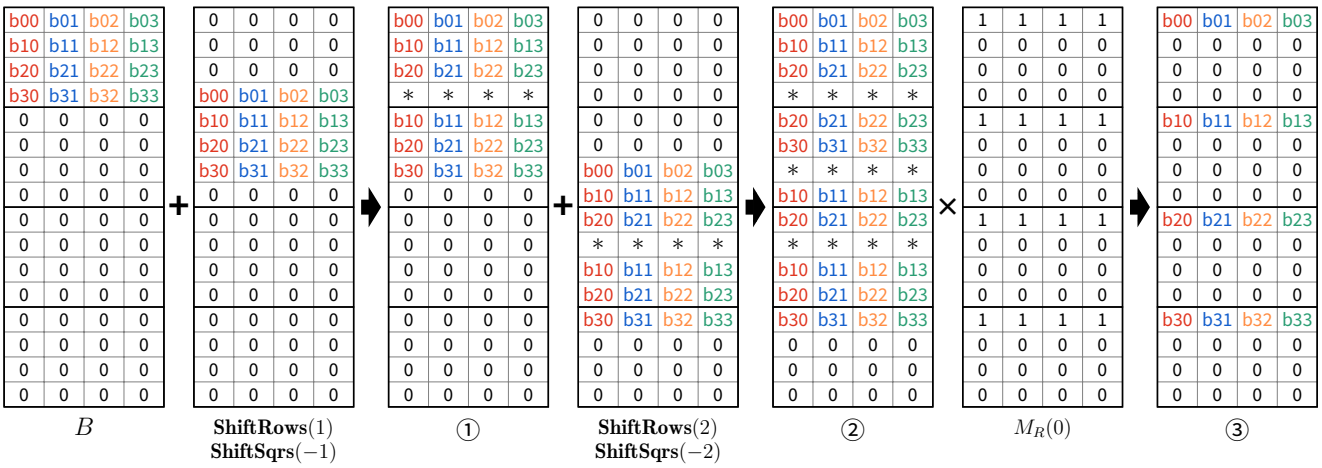

図 5: **ComputeBbar** (1)

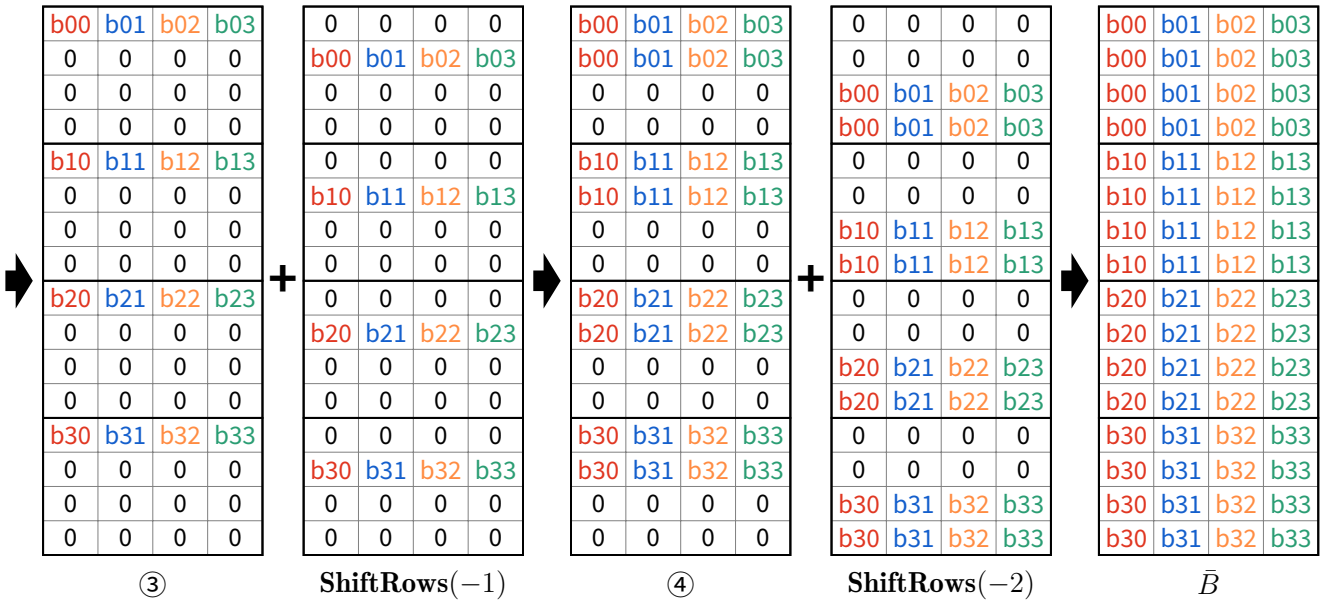

図 6: **ComputeBbar** (2)

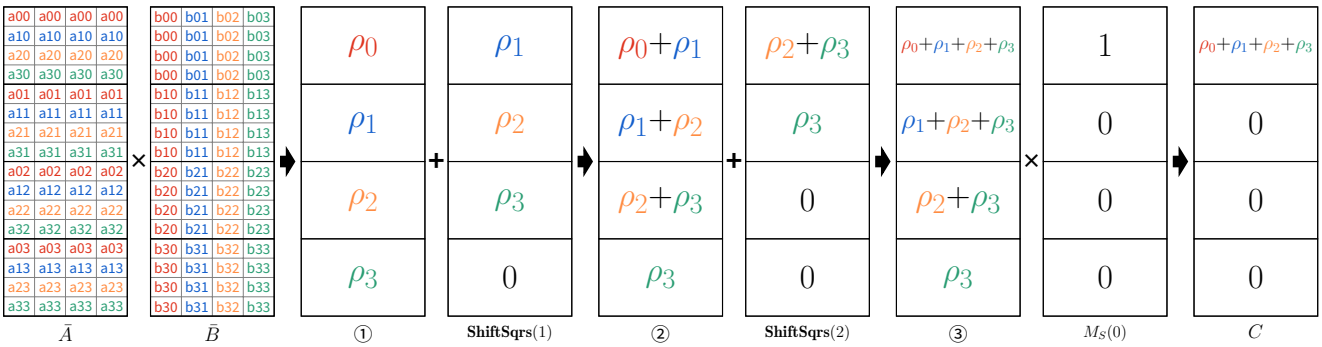

図 7: **ComputeC**

表 5: 暗号文行列積 1 回での CKKS 暗号文演算の実行回数と行列積全体の実行時間

手法	d	**Add**	**Rot**	**cMult**	**Mult**	実行時間 [ms]
[15]	4	18	14	19	4	56.1
本手法	4	10	14	3	1	40.9
[15]	8	42	30	36	8	140
本手法	8	15	21	3	1	58.1
[15]	16	90	62	65	16	282
本手法	16	20	28	3	1	75.1

表 6: 本手法による [15] との実行時間の削減率

d	実行時間削減率	削減時間 [ms]
4	27.1%	15.2
8	58.5%	81.9
16	73.4%	206.9

り返し，誤差の最大値および最小値の平均値を求め，**表 7** に示す．**表 7** より，全ての d に対して，本手法の誤差は Jiang らの手法より大きいことを確認したが，いずれの場合も同程度の値であった．

5 まとめ

本稿では，Mahon ら [10] が提案した BFV 方式における 3 次元配列表現を用いた行列積の高速化手法を，サブルーチンを一部変更することで CKKS 方式へ対応させた．そして，そのアルゴリズムの実装を Jiang ら [8] の手法と性能比較した．なお，Jiang らの手法は，我々の研究グループ [15] が与えた実装設計を基に実装を行った．

結果として，本手法は，Jiang らの手法と誤差はほぼ同等であり，実行時間はより高速であった．なお，実行時間の削減率は d の増大に伴い顕著であった．これは，行列積 1 回に必要な CKKS 暗号文演算の実行回数が，Jiang らの手法では全演算が $O(d)$ であるのに対し，本手法では **Add** や **Rot** が $O(\log(d))$，**cMult** や **Mult** が $O(1)$ であるためだと考えられる．一方で，本手法は，Jiang らの手法と比べ，1 つの暗号文で表現できる行列サイズの最大値が制限される．即ち，$d \times d$ 行列の行列積に少なくとも d^3 のスロットを用いるため，空間計算量の観点からは課題がある．

今後の課題として，例えば，$d = 1024$ などの大規模な行列に対して，本手法を適用しようとすると，長さが 2^{31} の多項式がパラメータとして必要となり，現実的な実行時間に収まらない可能性がある．そのため，1 つの暗号文ではなく，複数個の暗号文へ分割しても，Mahon らの手法が適用できるかを考察したい．

表 7: 誤差の最大値・最小値の平均値の比較

手法	d	最大値の平均値	最小値の平均値
[15]	4	1.13×10^{-5}	4.01×10^{-7}
本手法	4	1.34×10^{-5}	5.08×10^{-7}
[15]	8	2.10×10^{-5}	1.47×10^{-7}
本手法	8	2.36×10^{-5}	1.72×10^{-7}
[15]	16	3.56×10^{-5}	5.33×10^{-8}
本手法	16	3.91×10^{-5}	6.06×10^{-8}

参考文献

[1] Jung Cheon et al. “Homomorphic Encryption for Arithmetic of Approximate Numbers”. In: Nov. 2017, pp. 409–437. ISBN: 978-3-319-70693-1. DOI: 10.1007/978-3-319-70694-8_15.

[2] Jung Hee Cheon, Andrey Kim, and Donggeon Yhee. “Multidimensional Packing for HEAAN for Approximate Matrix Arithmetics”. In: *IACR Cryptol. ePrint Arch.* 2018 (2018), p. 1245. URL: https://api.semanticscholar.org/CorpusID:57760349.

[3] Ilaria Chillotti et al. “TFHE: Fast Fully Homomorphic Encryption Over the Torus”. In: *Journal of Cryptology* 33 (Apr. 2019). DOI: 10.1007/s00145-019-09319-x.

[4] Junfeng Fan and Frederik Vercauteren. *Somewhat Practical Fully Homomorphic Encryption*. Cryptology ePrint Archive, Paper 2012/144. 2012. URL: https://eprint.iacr.org/2012/144.

[5] Craig Gentry and Yongwoo Lee. *Fully Homomorphic Encryption for Matrix Arithmetic*. Cryptology ePrint Archive, Paper 2025/1935. 2025. URL: https://eprint.iacr.org/2025/1935.

[6] Charles R. Harris et al. “Array programming with NumPy”. In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2.

[7] Jaehee Jang et al. “Privacy-Preserving Deep Sequential Model with Matrix Homomorphic Encryption”. In: *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security*. ASIA CCS ’22. Nagasaki, Japan: Association for Computing Machinery, 2022, pp. 377–391. ISBN: 9781450391405. DOI: 10.1145/3488932.3523253. URL: https://doi.org/10.1145/3488932.3523253.

[8] Xiaoqian Jiang et al. “Secure Outsourced Matrix Computation and Application to Neural Networks”. In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. CCS ’18. Toronto, Canada: Association for Computing Machinery, 2018, pp. 1209–1222. ISBN: 9781450356930. DOI: 10.1145/3243734.3243837. URL: https://doi.org/10.1145/3243734.3243837.

[9] Wen-jie Lu, Shohei Kawasaki, and Jun Sakuma. “Using Fully Homomorphic Encryption for Statistical Analysis of Categorical, Ordinal and Numerical Data”. In: Jan. 2017. DOI: 10.14722/ndss.2017.23119.

[10] Hannah Mahon and Shane Kosieradzki. *Encrypted Matrix Multiplication Using 3-Dimensional Rotations*. Cryptology ePrint Archive, Paper 2025/1367. 2025. URL: https://eprint.iacr.org/2025/1367.

[11] Jai Hyun Park. *Ciphertext-Ciphertext Matrix Multiplication: Fast for Large Matrices*. Cryptology ePrint Archive, Paper 2025/448. 2025. URL: https://eprint.iacr.org/2025/448.

[12] Panagiotis Rizomiliotis and Aikaterini Triakosia. “On Matrix Multiplication with Homomorphic Encryption”. In: Nov. 2022, pp. 53–61. DOI: 10.1145/3560810.3564267.

[13] *Microsoft SEAL (release 4.1)*. https://github.com/Microsoft/SEAL. Microsoft Research, Redmond, WA. Jan. 2023.

[14] Shufang Wang and Hai Huang and. “Secure Outsourced Computation of Multiple Matrix Multiplication Based on Fully Homomorphic Encryption”. In: *KSII Transactions on Internet and Information Systems* 13.11 (Nov. 2019), pp. 5616–5630. DOI: 10.3837/tiis.2019.11.019.

[15] 眞鍋 拓朗 et al. “準同型暗号 CKKS 方式の行列積実装の高速化 (4C2-3)”. In: *2025 年 暗号と情報セキュリティシンポジウム (SCIS2025)*. 電子情報通信学会. 2025.