SCIS 2025 2025 Symposium on
Cryptography and Information Security
Kokura, Japan, Jan. 28 – 31, 2025
The Institute of Electronics,
Information and Communication Engineers

準同型暗号 CKKS 方式の行列積実装の高速化
Accelerate the implementation for the matrix multiplication by using homomorphic encryption CKKS scheme

眞鍋 拓朗 * 若杉 飛鳥 † 服部 大地 † 小寺 雄太 *
Takuro Manabe Asuka Wakasugi Daichi Hattori Yuta Kodera
野上 保之 *
Yasuyuki Nogami

あらまし 準同型暗号は，データを暗号化した状態で直接計算を行うことを可能にする技術であり，プライバシーを保護しながら安全な計算環境を提供する．しかしながら，複数回の加算と乗算を要する行列積計算アルゴリズムでは，計算コストや精度低下が課題とされている．本稿では，ACM SIGSAC 2018 の研究で紹介された，単一の暗号文に単一および複数の行列を暗号化する行列積の，2 つの高速化手法に着目する．また，先行研究の手法に対して，詳細な実装設計を与える．加えて，行列積 1 回あたりの実行時間を測定し，及び，当該論文では行われていなかった，cleartext との誤差の検証を行う．具体的には，要素ごとに暗号化を行なった行列積と，先行研究の 2 つの高速化アルゴリズムの実行時間を比較し，先行研究の高速化の効果について確認する．また，上記全てにおいて cleartext との誤差の最大値を求め，精度を確認する．

キーワード 行列積，準同型暗号，CKKS，秘密計算，プライバシー保護

1 はじめに

インターネットの普及に伴い，大規模データに対する処理の需要が急速に増大する一方で，個人情報や機密データの漏洩が重大な懸念事項となっている．特に医療や金融といった分野では，データの安全性を確保しつつ，その有用性を損なわない技術が求められている．従来の暗号方式では，データを利用する際に復号が必要であり，外部組織へのデータ共有が困難であるという課題があった．

このような課題を解決する革新的な技術として，準同型暗号が注目されている．準同型暗号は，暗号化された入力データに対して直接演算を行い，その結果を復号することで cleartext に対応する演算結果と一致させる性質を持つ．この特性により，データを暗号化したまま操作が可能となり，プライバシーを保ちながら外部組織とのデータ共有を実現する．

しかし，準同型暗号を実際のアプリケーションで広く活用するためには，暗号化されたデータに対する演算処理の効率化が不可欠である．特に，行列演算は多くの応用で中心的な役割を果たしており，医療データ解析や金融リスク評価など高度な計算を伴う場面で頻繁に使用される．一般に，準同型暗号による暗号文の乗算は，cleartext 同士の乗算と比較して，かなりの実行時間を要する．また，$d \times d$ サイズの行列同士の行列積アルゴリズムでは，d^3 回の乗算が必要とされる．したがって，準同型暗号における行列積の高速化は，実用的な処理時間内での運用を可能にするための重要な課題である．

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 らによって暗号化された 2 つの行列間の行列積計算手法 [8, 11] が実現された．しかし，それらは $d \times d$ 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 [2] 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

* 岡山大学大学院,700-0082 岡山県岡山市北区津島中 3-1,Okayama University, 3-1 Tsushima-naka Kita-ku Okayama City Okayama 700-0082 Japan
† EAGLYS 株式会社，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 Japan

表 1: メソッドごとの計算コスト

Method	CMult	Rot	Mult	Depth
[8, 11]	$O(d^2)$	$O(d^2 \log d)$	$O(d^2)$	1 CM+1 M
[2]	$O(d)$	$O(d \log d)$	$O(d)$	1 CM+1 M
[7]	$O(d)$	$O(d)$	$O(d)$	2 CM+1 M
[6, 9]	$O(d)$	$O(d)$	$O(d)$	1 CM+1 M

つの暗号化された行列の積を計算する．同年，Jiang ら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 [7] を導入した．更に 2022 年には，Jang らや Rizomiliotis らによって Jiang らのアルゴリズムを改良するアプローチ [6, 9] が提案されている．

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．各メソッドの詳細については [12] を参照されたい．**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 の回数であり，CMult を CM，Mult を M で表している．

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，実数や複素数をエンコードすることができ，他 2 方式よりも cleartext 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，CKKS 方式ではその処理が不要なため，一般に，CKKS 方式の方が TFHE 方式よりも乗算の実行時間が速いとされる．

更には，CKKS 方式を用いた本稿での実装対象として Jiang らの手法 [7] に着目する．その理由として，後続の研究 [6, 9] の基礎となる手法であることに加え，ニューラルネットワークへの応用といった幅広い活用が成されているが，計算結果の精度検証が行われていない．CKKS を用いた暗号方式では，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，最終的な結果の精度に影響を及ぼす可能性がある．したがって，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．また，Jiang らは複数の行列を単一の暗号文にエンコードすることで，行列ごとに対する演算の性能を向上させる方法も示している．しかしながら，詳しい実装内容が与えられておらず，実験環境も明確にされていない．そこで本稿では詳細な実装設計を行うとともに，その性能および精度について詳細に評価を行うことを目的とする．

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する．更には，本稿での実装対象である Jiang らの高速化手法 [7] と当該手法に着目した背景を述べる．第 2 章では，準同型暗号方式の一つである CKKS 方式に関する詳細と，Microsoft SEAL[10] を用いた暗号化および復号の方法を解説する．第 3 章では，実装対象となるアルゴリズムの概要を示し，その漸近計算量について議論する．第 4 章では，誤差分析および実行時間の計測結果を示し，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，[7] における 3.3.3 章の “Further Improvments” と 4.3 章の “Parallel Matrix Computation” を中心に実装し，その効率化性能と精度を検証した．実装結果として，要素ごとに暗号化を行う schoolbook の行列積と比較して，$d \times d$ 行列における d が小さい値をとる場合には schoolbook の行列積の方が速い結果となった．しかし，d が大きくなると Jiang らの手法の方が高速であり，実行時間の削減率は d が大きくなるに伴いより顕著であった．これは，行列積 1 回に必要な schoolbook の行列積の漸近計算量が $O(d^3)$ であるのに対し，Jiang らの手法では $O(d)$ であるためである．

また，誤差に関しては，schoolbook の行列積と比較して Jiang らの手法では約 50 倍大きくなったが，これは CKKS 方式による要素ごとの乗算回数が，Jiang らの手法では schoolbook の行列積と比べて 3 倍多いことが原因であると考えられる．

さらに，入力する cleartext が複数個の行列の場合に，それらの行列積に対して，複数個の行列積を 1 度の演算で行う実験を行い，“Parallel Matrix Computation” の効果を検証した．その結果，cleartext が 1 つの行列の場合の行列積と比べて，実行時間および誤差に顕著な変化が見られなかった．

以上より，本稿で示した実装内容が正確であり，高速化手法の性能と精度に関する詳細な実装と評価を提供している．

2 CKKS 方式

本節では，CKKS 方式の概要を説明する．分析対象データを cleartext という．また，暗号化対象データを plaintext（平文）と呼び，cleartext から plaintext への変換アルゴリズムを encode，その逆変換アルゴリズムを decode という．CKKS 方式で用いられる encode 方式には，Coefficients（以下 Coeff）方式と Slot 形式の 2 種類が知られている．Coeff 方式とは，Cheon らによる encode 方式 [1] である．Slot 方式とは，cleartext に対して，フーリエ変換アルゴリズムを適応させてから，Coeff 方式を用いる encode 方式である．本稿では，Slot 形式の encode/decode アルゴリズムを $\mathrm{Encode}_{\mathrm{Slot}}, \mathrm{Decode}_{\mathrm{Slot}}$

とし，その平文を p_{Slot} と表す．平文空間は，整数係数多項式環の円分多項式による剰余環によって定められ，本稿では，その多項式の長さを N とする．cleartext が 1 次元配列のデータ型であるとき，Coeff 方式及び Slot 方式いずれの場合でも，平文は長さ N の多項式で表現される．本稿では，cleartext の配列の長さを n で表し，$n \leq N$ とする．また，以降では Slot 方式のみ扱う．

秘密鍵を sk, 公開鍵を pk とし，$\mathrm{Encrypt}_{\mathrm{pk}}(p)$ で定まる暗号文を c とする．c は長さ N の整数係数多項式である．このとき，sk による復号関数を用いると，$\mathrm{Decrypt}_{\mathrm{sk}}(c) = p$ が成り立つ．c に対して，次の 5 つのアルゴリズムの組 (Add, Sub, Mult, CMult, Rot) が定義される．以下では，c が定まる暗号文空間内での演算を考える．Add/Sub/Mult では，それぞれ，Slot 形式による暗号文を多項式加算/減算/乗算する．CMult では，Slot 形式による暗号文と cleartext を多項式乗算する．Rot では，正整数 r を用いて，Slot 形式による暗号文に対して，X^r を掛ける．

3 実装対象アルゴリズム

本章では，cleartext が複数個の行列の場合の encode 及び暗号化，実装にあたって必要なパッキング，行列積の高速化手法及び，そのサブルーチンについて説明する．また，本稿では Jiang らの論文における 3.3.3 章の Further Improvements と 4.3 章における Parallel Matrix Computation を中心に実装を行う．

3.1 パッキング

Jiang らの高速化手法では複数個の cleartext を単一の plaintext に変換する．このようにすることで，暗号文状態での演算回数を削減することができる．本稿では，$d \times d$ 行列の cleartext が g 個あるものとし，$0 < gd^2 \leq N$ を満たすものとする．これに従い，$0 \leq i \leq g-1$ に対して，$d \times d$ の行列 $M_i = (a_{i(j,j')})_{0 \leq j,j' \leq d-1}$ を用意する．更に，上記のパッキングアルゴリズムを**アルゴリズム 1** に示す．このように得られたベクトルを encode することで，cleartext が複数の行列の場合の暗号化を可能とする．V を新しい cleartext とみなすことで，パッキング及び暗号化して得られるベクトルに対して，前章で定義した演算全てが成り立つ．

Algorithm 1 packing

Input: $M_0, \ldots, M_{g-1} \in \mathbb{R}^{d \times d}$, N
Output: V

$V \leftarrow []$
for $i = 0$ **to** $g-1$ **do**
  for $j = 0$ **to** d^2-1 **do**
    $V[g \cdot j + i] \leftarrow M_i[(j - (j \bmod d))/d][j \bmod d]$
return V

Algorithm 2 encode_diag

Input: $A \in \mathbb{R}^{d \times d}$
Output: $A' \in \mathbb{R}^{d \times d}$

$A' \leftarrow []$
for $i = 0$ **to** $d-1$ **do**
  for $j = 0$ **to** $d-1$ **do**
    $A'[i][j] \leftarrow A[j][(i+j) \bmod d]$
return A'

3.2 サブルーチン

本節では今後用いるサブルーチンである，tile，roll，及び，対角変換アルゴリズムを説明する．

本稿でのアルゴリズムでは暗号化状態での行列積を効率化するために，幾つかのスパースベクトルを作成し，入力暗号ベクトルとの乗算を行うサブルーチンを用いる．暗号文では，$gd^2 < N$ の場合でも，暗号化によってそのサイズは N まで拡張される．そのため，スパースベクトルのサイズが N よりも小さい場合は，乗算後に行うシフト操作の結果が正しく得られない．よって，暗号文とスパースベクトルの乗算を行う前に，スパースベクトルのサイズが N になるまで要素を複製する処理を行う．スパースベクトルを V_sparce と定義したとき，この要素の複製処理を $\mathrm{tile}(V\_\mathrm{sparce}, N/(gd^2))$ とする．

幾つかのアルゴリズムにおいて，ベクトルの要素をシフトする操作がある．これに対してシフト対象のベクトルを V としたとき，右に r だけ移動するという操作を次のように定義する．対象ベクトルが plaintext の場合，$\mathrm{roll}(V, r)$，ciphertext の場合は前章の $\mathrm{Rot}(V, r)$ を用いる．

行列演算を効率的に行うために，暗号化されたデータを特定の形式に変換する必要がある．行列のエントリを特定の規則に従って再配置し，暗号化状態の行列で効率的な計算を可能にするための対角変換アルゴリズムを**アルゴリズム 2** に示す．このアルゴリズムでは，行列 A と $1 \leq i \leq d$ に対して，A の i 行目が i 番目の対角ベクトルとなるように再構成する．ここで，i 番目の対角ベクトルとは，$(A_{0,i}, A_{1,i+1} \ldots, A_{d-i-1,d-1}, A_{d-i,0}, \ldots, A_{d-1,i-1})$ なる d 次元のベクトルのことである．

3.3 Step1-1

Step1-1 では，暗号化ベクトル C_1 に対して，次の条件を満たす暗号化ベクトル C_1' を得ることを目標とする．C_1 の cleartext $M_0, \ldots, M_{g-1}$ 及び $0 \leq j \leq g-1$ に対して，M_j の各 i 行目を左方向に i だけシフトさせてから，パッキング及び暗号化することで得られるベクトルが C_1' である．暗号文状態の行列に対して，行方向のシ

Algorithm 3 construct_U^σ

Input: d

Output: $U^\sigma \in \mathbb{R}^{d^2 \times d^2}$

$U^\sigma \leftarrow []$
for $i = 0$ **to** $d^2 - 1$ **do**
  for $j = 0$ **to** $d - 1$ **do**
    for $k = 0$ **to** $d - 1$ **do**
      if $i = d \cdot j + (j + k) \bmod d$ **then**
        $U^\sigma[d \cdot j + k][i] \leftarrow 1$
      else
        $U^\sigma[d \cdot j + k][i] \leftarrow 0$
return U^σ

フト操作を行列とベクトルの積として表現するために，線形変換行列 U^σ を構築する．

アルゴリズム 3 では，サイズ $d^2 \times d^2$ の行列 U^σ を初期化し，各行および列のインデックスに基づいて行列要素を設定する．具体的には，行のインデックス j と列のインデックス k に基づいて，条件を満たす場合に対応する行列要素 $U[d \times j + k][i]$ に値 1 を割り当てる．これにより，行シフト操作が効率的に行える線形変換行列が得られる．以上より，線形変換行列 U^σ の作成方法を**アルゴリズム 3** に示す．

Step 1-1 では，**アルゴリズム 2** に示した対角変換 encode_diag と**アルゴリズム 3** に示した線形変換 U^σ によってスパースな変換行列を作成する．そして，入力の暗号文ベクトル C_1 と作成した変換行列に対して，Rot や tile 処理を行いながら CKKS での乗算処理 CMult を行う．これにより，Step1-1 の計算回数は Add:$2d$, CMult:$2d$, Rot:$3\sqrt{d}$ である．Step1-1 の詳細を**アルゴリズム 4** に示す．

3.4 Step1-2

Step1-2 では，暗号化ベクトル C_2 に対して，次の条件を満たす暗号化ベクトル C_2' を得ることを目標とする．C_2 の cleartext $M_0, ..., M_{g-1}$ 及び $0 \le j \le g - 1$ に対して，M_j の各 i 列目を上方向に i だけシフトさせてから，パッキング及び暗号化することで得られるベクトルが C_2' である．暗号文状態の行列に対して，列方向のシフト操作を行列とベクトルの積として表現するために，線形変換行列 U^τ を構築する．

アルゴリズム 5 の内容は基本的には **アルゴリズム 3** と同様であるが，U^σ は行シフト操作を目的としていたのに対して，U^τ は列シフト操作を目的としている．詳細な線形変換行列 U^τ の作成方法は **アルゴリズム 5** に示す．

Step 1-2 でも，Step1-1 と同様に encode_diag と**アルゴリズム 5** に示した線形変換 U^τ によってスパースな変換行列を作成し，入力された暗号文ベクトル C_2 に対して，Rot や tile 処理を行いながら CKKS での乗算処理 CMult を行う．なお，6 行目における i の定義域や各 Rot 及び roll の引数などが Step1-1 と異なる点であるが，詳細は Jiang らの論文を参照されたい．計算回数は Add:d, CMult:d, Rot:$2\sqrt{d}$ である．Step1-1 の詳細を**アルゴリズム 6** に示す．

Algorithm 4 Step 1-1

Input: C_1, d, g, N

Output: C_1'

$U \leftarrow$ encode_diag(construct_$U^\sigma(d)$)
rot_c $\leftarrow []$
for $j = 0$ **to** $\lceil\sqrt{d}\rceil - 1$ **do**
  rot_c$[j] \leftarrow \mathrm{Rot}(C_1, -jg)$
$C_1' \leftarrow []$
for $i = -\lceil\sqrt{d}\rceil$ **to** $\lceil\sqrt{d}\rceil - 1$ **do**
  $c' \leftarrow []$
  for $j = 0$ **to** $\lceil\sqrt{d}\rceil - 1$ **do**
    if $|\lceil\sqrt{d}\rceil \cdot i + j| < d$ **then**
      ex_$U \leftarrow []$
      for $k = 0$ **to** $d^2 - 1$ **do**
        for $\ell = 0$ **to** $g - 1$ **do**
          ex_$U[kg + \ell] \leftarrow U[\lceil\sqrt{d}\rceil i + j][k]$
      rot_ex_$U \leftarrow \mathrm{roll}(\mathrm{ex}_U, \lceil\sqrt{d}\rceil ig)$
      tile_rot_ex_$U \leftarrow \mathrm{tile}(\mathrm{rot_ex}_U, N/(gd^2))$
      $c\_U \leftarrow \mathrm{CMult}(\mathrm{rot}\_c[j], \mathrm{tile\_rot\_ex}\_U)$
      $c' \leftarrow \mathrm{Add}(c', c\_U)$
  $C_1' \leftarrow \mathrm{Add}(C_1', \mathrm{Rot}(c', -\lceil\sqrt{d}\rceil ig))$
return C_1'

Algorithm 5 construct_U^τ

Input: d

Output: $U^\tau \in \mathbb{R}^{d^2 \times d^2}$

$U^\tau \leftarrow []$
for $i = 0$ **to** $d^2 - 1$ **do**
  for $j = 0$ **to** $d - 1$ **do**
    for $k = 0$ **to** $d - 1$ **do**
      if $i = d \cdot ((j + k) \bmod d) + k$ **then**
        $U^\tau[d \cdot j + k][i] \leftarrow 1$
      else
        $U^\tau[d \cdot j + k][i] \leftarrow 0$
return U^τ

3.5 Step2

Step2 では，暗号化ベクトル C_1, C_2 に対して，次の条件を満たす暗号化ベクトルを要素とする長さ d のベクトル $C_{1,v}, C_{2,w}$ を得ることを目標とする．各 $0 \le i \le d-1$

Algorithm 6 Step 1-2

Input: C_2, d, g, N

Output: C_2'

$U \leftarrow \text{encode\_diag}(\text{construct\_}U^{\tau}(d))$
rot_c $\leftarrow []$
for $j = 0$ **to** $\lceil\sqrt{d}\rceil - 1$ **do**
  $\text{rot\_}c[j] \leftarrow \text{Rot}(C_2, -djg)$
$C_2' \leftarrow []$
for $i = 0$ **to** $\lceil\sqrt{d}\rceil - 1$ **do**
  $c' \leftarrow []$
  for $j = 0$ **to** $\lceil\sqrt{d}\rceil - 1$ **do**
    if $\lceil\sqrt{d}\rceil i + j < d$ **then**
      $\text{ex\_}U \leftarrow []$
      for $k = 0$ **to** $d^2 - 1$ **do**
        for $\ell = 0$ **to** $g - 1$ **do**
          $\text{ex\_}U[kg + \ell] \leftarrow U[d(\lceil\sqrt{d}\rceil i + j)][k]$
      $\text{rot\_ex\_}U \leftarrow \text{roll}(\text{ex\_}U, d\lceil\sqrt{d}\rceil ig)$
      $\text{tile\_rot\_ex\_}U \leftarrow \text{tile}(\text{rot\_ex\_}U, N/(gd^2))$
      $c\_U \leftarrow \text{CMult}(\text{rot\_}c[j], \text{tile\_rot\_ex\_}U)$
      $c' \leftarrow \text{Add}(c', c\_U)$
  $C_2' \leftarrow \text{Add}(C_2', \text{Rot}(c', -d\lceil\sqrt{d}\rceil ig))$
return C_2'

Algorithm 7 construct_v_k

Input: d, k

Output: v_k

$v_k \leftarrow []$
for $i = 0$ **to** $d^2 - 1$ **do**
  if $0 \le (i \bmod d) < (d - k)$ **then**
    $v_k[i] \leftarrow 1$
  else
    $v_k[i] \leftarrow 0$
return v_k

に対して，$C_{1,v}, C_{2,w}$ の i 番目の要素を $C_{1,v,i}, C_{2,w,i}$ とする．C_1 の cleartext $M_{1,0}, \ldots, M_{1,g-1}$ に対して，各行列の各行を左方向に i だけシフトさせてから，パッキング及び暗号化することで得られるベクトルが $C_{1,v,i}$ である．また，C_2 の cleartext $M_{2,0}, \ldots, M_{2,g-1}$ に対して，各行列の各列を上方向に i だけシフトさせてから，パッキング及び暗号化することで得られるベクトルが $C_{2,w,i}$ である．この操作により，C_1, C_2 のそれぞれに対して d 個のシフト操作されたベクトルを得る．

C_1 に対するシフト操作を可能にするために，$0 \le k \le d^2 - 1$ に対して，行シフトベクトル v_k を構築する．v_k は，0 番目から $(d - 1 - k)$ 番目までが 1，それ以外が 0 のベクトルとする．v_k の生成アルゴリズムを**アルゴリズム 7** に示す．

Algorithm 8 Step 2

Input: C_1, C_2, d, g, N

Output: $C_{1,v}$, $C_{2,w}$

$C_{1,v}, C_{2,w} \leftarrow [], []$
for $k = 0$ **to** $d - 1$ **do**
  if $k = 0$ **then**
    $C_{1,v}[0] \leftarrow C_1$
    $C_{2,w}[0] \leftarrow C_2$
  else
    $v_k \leftarrow \text{construct\_}v_k(d, k)$
    $\text{ex\_}v\_k \leftarrow []$
    for $\ell = 0$ **to** $d^2 - 1$ **do**
      for $m = 0$ **to** $g - 1$ **do**
        $\text{ex\_}v\_k[\ell g + m] \leftarrow v\_k[\ell]$
    $\text{rot\_ex\_}v\_k \leftarrow \text{roll}(\text{ex\_}v\_k, kg)$
    $\text{tiled\_rot\_ex\_}v\_k \leftarrow \text{tile}(\text{rot\_ex\_}v\_k, N/(gd^2))$
    $C_1' \leftarrow \text{CMult}(C_1, \text{tiled\_rot\_ex\_}v\_k)$
    $\text{rot\_}C_1' \leftarrow \text{Rot}(C_1', -kg)$
    $C_1'' \leftarrow \text{Sub}(C_1, C_1')$
    $\text{rot\_}C_1'' \leftarrow \text{Rot}(C_1'', (d - k)g)$
    $C_{1,v}[k] \leftarrow \text{Add}(\text{rot\_}C_1', \text{rot\_}C_1'')$
    $C_{2,w}[k] \leftarrow \text{Rot}(C_2, -dkg)$
return $C_{1,v}$, $C_{2,w}$

Step2 では C_1 と C_2 及び，d, g, N を入力として受け取る．入力暗号ベクトル C_1 に対して，**アルゴリズム 7** を用いて前述で示したシフト操作を行う．C_2 に対しては，dkg だけ左シフトを行うことで，前述の行列状態での上方向シフトを実現する．以上の計算回数は Add:$2d$, CMult:d, Rot:$3d$ である．Step2 の詳細な実装内容を**アルゴリズム 8** に示す．

3.6 メインアルゴリズム

メインの **アルゴリズム 9** は，暗号化ベクトル C_1, C_2 に対して，次の条件を満たす暗号化ベクトル C を得ることを目標とする．C_1 の cleartext $M_{1,0}, \ldots, M_{1,g-1}$ と C_2 の cleartext $M_{2,0}, \ldots, M_{2,g-1}$ 及び $0 \le i \le g - 1$ に対して，$M_{1,i}$ と $M_{2,i}$ の行列積 M_i を求めてから，$[M_0, \ldots, M_{g-1}]$ をパッキング及び暗号化することで得られるベクトルが C である．C_1 と C_2 及び，d, g, N を入力として受け取り， 3.2 節から 3.5 節を用いて入力暗号ベクトルそれぞれに対して変換処理を行い，Step3 でそれらの乗算処理を行う．本アルゴリズムの各ステップにおける計算回数を**表 2** に示す．高速化された行列積アルゴリズムの詳細を**アルゴリズム 9** に示す．

最も基本的な行列積の場合，Mult が d^3 回で，Add が $(d - 1)d^2$ 回必要である．更に，それらが g 組あるので，g 組全体での計算回数は Mult が gd^3 で，Add が

表 2: ステップごとの計算回数

Step	Add	CMult	Rot	Mult
1-1	$2d$	$2d$	$3\sqrt{d}$	-
1-2	d	d	$2\sqrt{d}$	-
2	$2d$	d	$3d$	-
3	d	-	-	d

Algorithm 9 行列積の高速化アルゴリズム

Input: C_1, C_2, d, g, N

Output: C

Step 1

$C'_1 \leftarrow \mathrm{Step1\_1}(C_1, d, g, N)$

$C'_2 \leftarrow \mathrm{Step1\_2}(C_2, d, g, N)$

Step 2

$C_{1,v}, C_{2,w} \leftarrow \mathrm{Step2}(C'_1, C'_2, d, g, N)$

Step 3

$C \leftarrow []$

for $k = 0$ **to** $d-1$ **do**

  $C \leftarrow \mathrm{Add}(C, \mathrm{Mult}(C_{1,v}[k], C_{2,w}[k]))$

return C

$g(d-1)d^2$ 回必要である．上記に対して，**アルゴリズム 9** では g がいくつであっても**表 2** に示す計算回数になる．これが本アルゴリズムによる高速化の効果である．

4 実験

本章では，要素ごとに暗号化を行う基本的な行列積 (以後 schoolbook の行列積とする) と前章に示した行列積の両方を実装し，実行時間の比較と cleartext での行列積との誤差を測定し確認する．具体的には，$1 \leq i \leq 6$ に対して $d = 2^i$ とし，各 d に対して 100 回ずつ行列積を行い実行時間の平均を求める．実行時間の測定対象は暗号化・復号を除き，行列積のみとする．誤差に関しては，1 つの cleartext の行列積と暗号文の行列積に対して，各要素の誤差の最大値と最小値をそれぞれ取得する．以上の操作を 100 回繰り返し，誤差の最大値及び最小値の平均値を求める．

実験環境は**表 3** に示す．本実験では，cleartext の生成に加えて 3.2 節で説明した tile と roll に対し，Numpy[5] モジュールに実装されている関数 random.rand(), tile(), roll() を用いる．また，random.rand() より，本実験で使用する行列の定義域は $[0, 1)$ である．更に，cleartext の行列積に対し，ndarray クラスに実装されている _matmul_ メソッドを用いる．

表 3: 実験環境

OS	Ubuntu 22.04.5 LTS
CPU	Intel(R) Core(TM) Ultra 7 155H
クロック周波数	3.80 GHz
RAM	32 GB
SEAL	4.0.0
Python	3.12.3
NumPy	1.26.4

表 4: 行列積 1 回あたりの計算回数の理論値

	schoolbook		**アルゴリズム 9**			
d	Add	Mult	Add	CMult	Rot	Mult
2	4	8	12	4	16	2
4	48	64	24	16	22	4
8	448	512	48	32	39	8
16	3840	4096	96	64	68	16
32	31744	32768	192	128	126	32
64	258048	262144	384	256	232	64

4.1 実験 1

本節では，入力する cleartext として $d \times d$ サイズの M_1 と M_2 の 2 つの行列を用意し，schoolbook の行列積と $g = 1$ の**アルゴリズム 9** を用いた暗号文状態での行列積 M_1M_2 のそれぞれの場合に対して，行列積 1 回にかかる実行時間を測定し，誤差を確認する．まずは，各 d に対して，**表 4** では，**表 2** などから算出される，各行列積における各演算回数を示す．また，実際に**アルゴリズム 9** を実装した際に，例えば，**アルゴリズム ??**の 9 行目の **if** 文内の演算が実行されないケースがある．**表 5** では，各 d, g に対して，各演算が実際に実行される計算回数および実行時間を示す．**表 5** より，$d \geq 8$ の場合，**アルゴリズム 9** を用いることで実行時間を大幅に削減できることが確認できる．一方で，$d = 2, 4$ の場合には schoolbook の行列積の方が実行時間が短い結果となっている．この要因として，**表 5** に示すように $d = 2, 4$ では各計算回数にそれほど大きな差はないため，計算手順が単純な schoolbook の行列積の方が早くなっていると考えられる．しかし，$d \geq 8$ の場合，実行時間の削減率は d の増加に伴い顕著になる．これは，行列サイズ d に対して，schoolbook の行列積における計算回数が $O(d^3)$ である一方で，**アルゴリズム 9** では効率化により $O(d)$ であるためである．このことから，**アルゴリズム 9** は特に高次元での計算においてその優位性が発揮されることが分かる．また，**表 5** の**アルゴリズム 9** における計算回数が，**表 2** と比較してわずかに少なくなっている．これは，**表 2** に示す計算回数がアルゴリズム内の条件判定

表 5: 行列積 1 回あたりの実行時間と計算回数の比較

	schoolbook			**アルゴリズム 9**				
d	Add	Mult	実行時間 [s]	Add	CMult	Rot	Mult	実行時間 [s]
2	4	8	1.28×10^{-2}	6	6	10	2	5.14×10^{-2}
4	48	64	9.73×10^{-2}	18	14	19	4	1.19×10^{-1}
8	448	512	8.12×10^{-1}	42	30	36	8	2.90×10^{-1}
16	3840	4096	6.45	90	62	65	16	6.32×10^{-1}
32	31744	32768	52.5	186	126	123	32	2.22
64	258048	262144	431	378	254	229	64	18.7

表 6: 誤差の最大値と最小値の比較

	schoolbook		**アルゴリズム 9**($g = 1$)	
d	最大値	最小値	最大値	最小値
2	1.09×10^{-7}	3.98×10^{-8}	5.55×10^{-6}	1.60×10^{-6}
4	2.33×10^{-7}	5.74×10^{-8}	1.18×10^{-5}	1.79×10^{-6}
8	4.54×10^{-7}	1.12×10^{-7}	2.41×10^{-5}	3.67×10^{-6}
16	8.63×10^{-7}	2.63×10^{-7}	4.73×10^{-5}	9.52×10^{-6}
32	1.62×10^{-6}	6.05×10^{-7}	9.86×10^{-5}	2.42×10^{-5}
64	3.00×10^{-6}	1.40×10^{-6}	1.71×10^{-4}	5.76×10^{-5}

を考慮していないためである．

表 6 に d 毎の各行列積の結果から，要素ごとの誤差の最大値と最小値を示す．**表 6** より，schoolbook の行列積に対して高速化手法の方が 50 倍程度大きい結果となっている．この差異の原因として考えられるのは，schoolbook の行列積では各要素に対する CKKS 方式での乗算回数が 1 回であるのに対し，**アルゴリズム 9** では 3 回行われる点である．CKKS 方式を利用した演算の復号結果は，実数を整数に近似することに起因する誤差が生じる．つまり，乗算回数の増加に伴って誤差が累積するため，**アルゴリズム 9** の方が schoolbook の行列積よりも大きな誤差を生じる結果となったと考えられる．また，d の増加に伴い誤差も大きくなっているのは，**表 5** から確認できるとおり，各計算回数も増加しているためだと考えられる．

4.2 実験 2

本節では，$j \geq 2$ とし，入力する cleartext が $M_{1,i}, M_{2,i}$ $(0 \leq i \leq j-1)$ の場合に，それらの行列積 $M_{1,i}M_{2,i}$ に対して，$g = j$ とする**アルゴリズム 9** を用いた演算を行い，実験 1 における $g = 1$ との比較を行う．本実験では $g = 8$ を主に使用しているが，N の制約上，$d = 64$ の場合には $g = 8$ ではなく $g = 2$ を使用している．各 d において，$g > 1$ での**アルゴリズム 9** を 100 回実行し，その

表 7: **アルゴリズム 9** で $g > 1$ の場合における実行時間

d	g	実行時間 [s]
2	8	5.23×10^{-2}
4	8	1.19×10^{-1}
8	8	2.94×10^{-1}
16	8	6.38×10^{-1}
32	8	2.22
64	2	18.5

表 8: **アルゴリズム 9** で $g > 1$ の場合の誤差の最大値と最小値

d	g	最大値	最小値
2	8	5.08×10^{-6}	1.55×10^{-6}
4	8	1.21×10^{-5}	1.96×10^{-6}
8	8	2.40×10^{-5}	3.72×10^{-6}
16	8	4.51×10^{-5}	9.59×10^{-6}
32	8	8.40×10^{-5}	2.39×10^{-5}
64	2	1.54×10^{-4}	3.54×10^{-5}

平均実行時間を**表 7** に示し，要素ごとの誤差の最大値と最小値を**表 8** に示す．

$g > 1$ の場合の**表 7** 及び**表 8** に対して，それぞれ，$g = 1$ での**表 5** 及び**表 6** と比較すると，各 d での値にほとんど変化は見られない．このことから，複数回分の行列積を 1 回分の時間で 1 度に行えることから，前節で述べた以上に効率化の効果は大きいと言える．この結果は，Jiang らの高速化アルゴリズムの効率化と，本稿での実装内容の正確性を示している．

5 まとめ

本稿では，Jiang らの提案した行列積の高速化アルゴリズムを実装し，実行時間の測定と誤差の検証を行った．

この研究の動機として，Jiang らの手法は多くの論文で引用されており，更なる効率化に向けた研究が行われている一方で，誤差の検証が行われている例はこれまで見受けられなかったためである．本研究では，高速化アルゴリズムの効果を検証するために，2 種類の実験を行った．1 つ目の実験では，要素ごとに暗号化を行った schoolbook の行列積と高速化手法を比較し，各 $d = 2^i (1 \leq i \leq 6)$ に対して 100 回ずつ $d \times d$ 行列の行列積を行い，実行時間の平均値と誤差を算出した．結果として，$d \leq 4$ の場合では schoolbook の行列積の方が高速であったが，$d \geq 8$ の場合では d の増加に伴い実行時間の削減率はより顕著になった．よって Jiang らの手法は特に高次元における計算において効果を発揮するといえる．誤差に関しては，高速化手法の方が schoolbook の行列積と比較して約 50 倍大きい結果となったが，これは CKKS 方式における各要素の演算回数が，高速化手法ではナイーブな行列積の 3 倍であることに起因すると考えられる．2 つ目の実験では，入力する cleartext が $M_{1,i}, M_{2,i} (0 \leq i \leq g-1)$ の場合に，それらの行列積 $M_{1,i} M_{2,i}$ に対して，g 個の行列積を 1 度の演算で行う実験を行った．結果として，g の値が増加しても $g = 1$ の場合と同程度の実行時間，及び，誤差で実現できることが確認された．これらの結果を踏まえ，本稿の実装が正確であることを確認し，高速化手法は行列積の効率化において有効であり，特に大規模な計算でその効果が顕著であることが示された．本研究の成果は，CKKS 方式を用いた行列積における高速化と誤差特性の理解に寄与すると考えられる．

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. “Multi-dimensional 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] 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.

[6] 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.

[7] 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.

[8] 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.

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

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

[11] 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.

[12] Xiaopeng Zheng, Hongbo Li, and Dingkang Wang. “A New Framework for Fast Homomorphic Matrix Multiplication”. In: *Cryptology ePrint Archive* (2023).