情報処理学会論文誌 Vol.65 No.12 1892–1902 (Dec. 2024)

推薦論文

効率的な暗号処理に向けた FHE暗号方式・ライブラリの比較

辻 有紗[1,a)] 圓戸 辰郎[2] 小口 正人[1,b)]

受付日 2023年12月5日, 採録日 2024年9月9日

概要：クラウドサーバ上のデータ保護のために暗号化が重要であり，暗号化された状態で任意の演算が可能な完全準同型暗号（以下 FHE: Fully Homomorphic Encryption）はその目的に多いに期待されている．FHE の実用化に向けて，複数の暗号方式が提案されており，それぞれの暗号方式を実行可能なライブラリが複数存在する．本稿では，実行環境や処理内容に基づき，適切な暗号方式・ライブラリを選択するための比較を行った．具体的には，初めに OpenFHE，Lattigo，TFHEpp ライブラリにそれぞれ実装された BFV，BGV，CKKS，TFHE 亜種方式の時間空間計算量や得意な演算を整理した．実験から，128 bit security を満たすパラメータでは BGV，BFV，CKKS の順に高速であることを示した．また，乗算の深さが大きい場合は TFHE 亜種方式が適切であり，乗算の深さが小さい場合は CKKS 方式が適切であることを示した．ここで，クラウドでは使用可能な DRAM 容量が制限される場合が多い．そこで 2 番目に，TFHE 亜種方式が実装された OpenFHE と TFHEpp ライブラリを用いて，DRAM 容量制限時の実行時間や Solid State Drive（SSD）の帯域幅を比較した．実験から，DRAM 容量制限下では OpenFHE に比べて TFHEpp が高速であることを示した．

キーワード：完全準同型暗号，FHE 暗号方式，FHE 暗号ライブラリ

Comparison of FHE Schemes and Libraries for Efficient Cryptographic Processing

Arisa Tsuji[1,a)] Tatsuro Endo[2] Masato Oguchi[1,b)]

Received: December 5, 2023, Accepted: September 9, 2024

Abstract: Cryptographic processing is imperative for protecting data on cloud servers, and Fully Homomorphic Encryption (FHE), which can perform any calculations in an encrypted state, is highly expected for this purpose. In the pursuit of practical applications of FHE, multiple encryption schemes have been proposed, and several libraries are available for executing these schemes. In this study, we conducted a comparison to help select the appropriate FHE encryption scheme and library based on the execution environment and processing requirements of the application. Specifically, we first organize the time-space complexity and compatible operations for BFV, BGV, CKKS, and Zama's variant of TFHE schemes implemented in OpenFHE, Lattigo, and TFHEpp libraries. For achieving 128-bit security, it was found that BGV, BFV, and CKKS, in that order, are the fastest. Furthermore, Zama's variant of TFHE was more compatible when the depth of multiplication is large, but CKKS was more compatible when the depth of multiplication is small. Here, the available DRAM capacity is often limited in the cloud. Therefore, as a second consideration, we compared the execution times and Solid State Drive (SSD) bandwidths between OpenFHE and TFHEpp for Zama's variant of TFHE in environments with limited DRAM. It was found that TFHEpp is faster when DRAM is limited.

Keywords: (Torus) Fully Homomorphic Encryption, FHE schemes, FHE libraries

[1] お茶の水女子大学
Ochanomizu University, Bunkyo, Tokyo 112–8610, Japan
[2] マツダ株式会社
Mazda Motor Corporation, Aki-gun, Hiroshima 730–8670, Japan
[a)] arisa-t@ogl.is.ocha.ac.jp
[b)] oguchi@is.ocha.ac.jp

本稿の内容は 2022 年 7 月のマルチメディア，分散，協調とモバイル（DICOMO2022）シンポジウムにて報告され，コンピュータセキュリティ研究会主査により情報処理学会論文誌ジャーナルへの掲載が推薦された論文である．

1. はじめに

近年，クラウド上で大規模なシステムを運用し，ビッグデータを収集・分析する機会が増加している．従来の暗号技術を用いてクラウドを運用する場合，分析中にデータを復号する必要があり，個人情報のクラウドサーバからの漏洩や不正利用のリスクが高い．そこで，暗号化されたデータの任意の演算が可能な完全準同型暗号（以下 FHE: Fully Homomorphic Encryption）[1] が注目を集めている．FHE では，BFV 方式 [2]，BGV 方式 [3]，CKKS 方式 [4]，TFHE 亜種方式 [5] など複数の暗号方式が提案されており，それぞれ暗号処理の方法が異なる．また，これらの暗号方式を実行可能なライブラリが複数存在し，各ライブラリは細かい実装が異なる．実行環境や暗号化された状態で行う処理に応じて適切な暗号方式とライブラリを選択する必要があるが，暗号の研究者以外が FHE ライブラリ内で適切なパラメータを設定し，暗号方式やライブラリの比較を行うことは容易でない．

本稿では，複数の暗号方式とライブラリの計測を行い，FHE を使用してシステムを構築する際の参考情報を提供することを目的とする．具体的には，初めに OpenFHE [6]，Lattigo [7]，TFHEpp ライブラリ [8] でそれぞれ実装された BFV，BGV，CKKS，TFHE 亜種方式において，乗算実行時の時間空間計算量や効率的に実行可能な演算を整理した．実験から，整数の暗号処理を行う場合，Lattigo ライブラリの BGV 方式が最速であることが分かった．浮動小数点の暗号処理が可能な CKKS 方式では，連続した乗算回数が小さい場合は Lattigo が高速であり，多い場合は OpenFHE が高速であった．CKKS 方式と TFHE 亜種方式の比較では，任意の複数の値の乗算を行う際は TFHE 亜種方式の時間空間計算量が小さいが，ベクトル内積では CKKS 方式が効率的であることを示した．

実用的なアプリケーションを考慮した際のすべての FHE 暗号方式に共通の課題は，時間空間計算量が大きいことである．特に，クラウド環境では VM やコンテナの利用によって 1 台のリソースを分け合うため，DRAM の使用効率が低く，ストレージを活用する必要がある [9]．したがって 2 番目に，DRAM が制限された状態での TFHE 亜種方式の実行に焦点を当て，TFHE 亜種方式が実装された OpenFHE と TFHEpp の実行時間，SSD 帯域幅，および実装効率を比較した．利用可能な DRAM 容量に基づいて適切なライブラリを選択することが目的である．その結果，DRAM 容量制限により OpenFHE の gate key の生成時間が増加するため，TFHEpp が OpenFHE より高速であることを示した．

また，予備実験では，FHE 暗号方式・ライブラリを実アプリケーション内で評価し，FHE を用いた暗号処理の負荷が大きいことを示した．具体的には，HElib ライブラリの BGV 方式を用いたゲノム秘匿情報検索の計測を行った．ゲノム秘匿検索では，ある 1 つのゲノム塩基配列を複数のゲノム塩基配列が格納されたデータベースと照合することで，病気の原因であるか判定を行う．実験から，bootstrapping と暗号文どうしの値の比較を行う際の負荷が大きいことが分かった．

本稿の貢献を以下に示す．

- FHE 暗号スキーム（BFV, BGV, CKKS, TFHE 亜種方式）およびライブラリ（OpenFHE, Lattigo, TFHEpp）の時間空間計算量を測定し，実行環境と FHE を用いて行う操作に応じて適切なものを選択するための比較を行った．
- DRAM 容量の制約下で，TFHE 亜種方式が実装された OpenFHE と TFHEpp の実行時間，SSD 帯域幅，実装効率を比較した．この評価は，利用可能な DRAM 容量に基づいて適切なライブラリを選択することを目的とする．
- HElib ライブラリに実装された BGV 方式を用いたゲノム秘匿検索の計測を行い，FHE 暗号処理の負荷が大きいことが示した．

2. 準備

2.1 完全準同型暗号

FHE とは，任意の演算について，データを暗号化した状態で計算し，完了後に復号することで，暗号化せずに計算した場合と同じ結果が得られる暗号方式のことである．$\oplus$ を暗号化した状態で行われる加算，$\otimes$ を暗号化した状態で行われる乗算，m，n を暗号化される値とすると，FHE はつねに式 (1)，(2) を満たす．

$$Decrypt(Encrypt(m) \oplus Encrypt(n)) = Decrypt(Encrypt(m+n)) \quad (1)$$

$$Decrypt(Encrypt(m) \otimes Encrypt(n)) = Decrypt(Encrypt(m \times n)) \quad (2)$$

図 1 に FHE の暗号処理の流れを示す．各処理は以下のように行われる．

- **エンコーディング**：メッセージ m を平文 $\mathcal{M}$ に変換する．BFV，BGV，および CKKS 方式（2.2.1 項を参照）では，平文 $\mathcal{M}$ は $N-1$ 次の多項式であり，

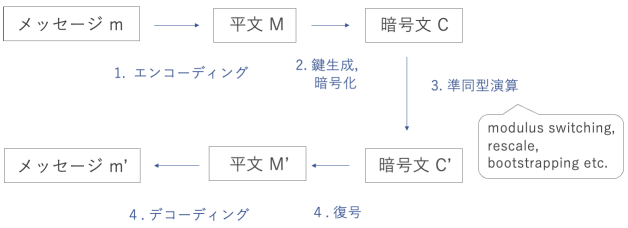


図 1 FHE の暗号処理の流れ
Fig. 1 Processing flow of FHE.

$R_t \in Z_t[x]/(x^N+1)$ と表現される．複数のメッセージ $m_s(s=0,\ldots,N-1)$ を 1 つの平文 $\mathcal{M}$ にパッキングすることができる．TFHE 亜種方式（2.2.1 項を参照）では，平文 $\mathcal{M}$ はトーラス上で定義され，1 つのメッセージ m が 1 つの平文 $\mathcal{M}$ に変換される．

- **鍵生成**：平文 $\mathcal{M}$ から暗号文 $\mathcal{C}$ に変換するための鍵を生成する．FHE は公開鍵暗号方式を用いており，秘密鍵と公開鍵の組を作成する．一般的に，公開鍵で暗号化，秘密鍵で復号を行う．
- **暗号化**：平文 $\mathcal{M}$ から暗号文 $\mathcal{C}$ を生成する．BFV，BGV，および CKKS 方式では，暗号文 $\mathcal{C}$ は多項式 $R_q \in Z_q[x]/(x^N+1)$ で表現される．平文 $\mathcal{M}$ が属する多項式 R_t は，ノイズ，スケーリングパラメータ，および公開鍵を使用して暗号文 $\mathcal{C}$ を表す多項式 R_q にマッピングされる．TFHE 亜種方式では暗号文は各要素がトーラスである $(n+1)$ 次のベクトルで表現される．
- **演算**：BFV，BGV，CKKS 方式では準同型加算・準同型乗算，TFHE（亜種）方式では準同型論理演算を行う．
- **Modulus switching または Rescale**：BFV，BGV，および CKKS 方式では，平文 $\mathcal{M}$ にノイズを加えて暗号化されるため，計算中に暗号文 $\mathcal{C}$ にノイズが蓄積する．ノイズは暗号文のモジュラスを q から q' $(q', q \in \mathbb{N}, q' < q)$ に変換することで削減される．この操作は，BFV，BGV 方式では modulus switching，CKKS 方式では rescale と呼ばれる．それぞれの暗号方式で，FHE 処理を開始する前に modulus switching または rescale の実行回数を決定する必要がある．
- **Bootstrapping**：暗号文 $\mathcal{C}$ に蓄積したノイズを削減する．ノイズが閾値を超えた暗号文には bootstrapping を適用できない．適切なタイミングで bootstrapping を行うことにより，任意回乗算することが可能である．特に，TFHE 亜種方式に実装された table lookup による単変数関数の実行時のノイズの削減は programmable bootstrapping と呼ばれる．
- **復号**：秘密鍵を使用して暗号文 $\mathcal{C}$ を復号し，平文 $\mathcal{M}'$ を出力する．
- **デコーディング**：エンコーディングの逆の手順を行い，平文 $\mathcal{M}'$ をメッセージ m' に変換する．

2.2 FHE 暗号方式とライブラリ

FHE には BFV，BGV，CKKS，TFHE [11]，TFHE 亜種方式など，複数の暗号方式が提案されており，それぞれ暗号処理の仕組みが異なる．また，各暗号方式が実行可能な複数のライブラリが公開されている．

2.2.1 暗号方式

表 1 に BFV，BGV，CKKS，TFHE，TFHE 亜種方式の特徴を示す．BFV，BGV，および CKKS 方式は，bootstrapping の理論など，基本的な暗号処理の流れは同じである．BFV，BGV，CKKS 方式では，暗号化された状態で復号回路を評価することで bootstrapping を実現する．また，準同型加算・準同型乗算を行う．ここで，BFV および BGV 方式は RLWE 問題 [12] を使用して整数を評価し，1 つの平文に N 個のメッセージを格納可能である．一方で，CKKS 方式は Fourier 変換を使用して浮動小数点を評価し，1 つの平文に $N/2$ 個のメッセージを格納可能である．

表 1 FHE 暗号方式の比較
Table 1 Features of FHE schemes.

FHE 暗号方式	特徴			
	算術演算	論理演算	ドメイン	パッキングサイズ
BFV	✓		integer	N
BGV	✓		integer	N
CKKS	✓		floating-point	$N/2$
TFHE		✓	binary	1
TFHE 亜種	✓	✓	floating-point	1

TFHE 方式は，BFV，BGV，CKKS 方式と基本的な暗号処理が異なる．TFHE 方式では，平文 $\mathcal{M}$ はバイナリで定義され，準同型演算により論理回路を評価する．準同型加算や準同型乗算を行う場合は，準同型論理演算を組み合わせる必要がある．bootstrapping は，ノイズが一定量に削減された暗号文が格納された LUT（Lookup Table）と呼ばれるベクトルの適切な位置を参照することで行われる．bootstrapping は回路を構成する各論理ゲートで実行される．

TFHE 亜種方式は TFHE 方式を拡張したもので，programmable bootstrapping と呼ばれる実装により，準同型論理演算だけでなく単変数関数の準同型演算も可能である．また，エンコーダの実装により，浮動小数点の評価が可能である．将来の課題の 1 つに計算中のドリフトエラーを削減することがあげられる [5]．本研究では，BFV，BGV，CKKS，TFHE 亜種方式の比較を行った．

2.2.2 ライブラリ

FHE の実装を行っているライブラリは複数存在する [13]．**表 2** に各暗号方式が実装されているライブラリの一部を示す．BFV，BGV，CKKS 方式は OpenFHE，Lattigo などのライブラリで実装されている．OpenFHE はオープンソースのライブラリであり，2021 年に Google によって発表された，C++のプログラムを FHE 暗号を用いた処理に変換するトランスパイラ [14] 内で使用されている．Lattigo は Go 言語のライブラリであり，秘密分散処理も実装されている．

TFHE，TFHE 亜種方式は OpenFHE，TFHEpp などのライブラリで実装されている．TFHEpp はオープンソースの C++のライブラリで，オリジナルの TFHE の実装 [15] より約 10%ほど高速である．FHE 暗号方式・ライブラリの比較では，OpenFHE，Lattigo，TFHEpp を使用し，FHE 暗号を用いた実アプリケーションの評価では，広く用いら

表 2 FHE 暗号方式が実装されているライブラリ
Table 2 Libraries in which the FHE encryption schemes are implemented.

FHE 暗号方式	ライブラリ		
	OpenFHE	Lattigo	TFHEpp
BFV	✓	✓	
BGV	✓	✓	
CKKS	✓	✓	
TFHE	✓		✓
TFHE 亜種	✓		✓

れている HElib を用いた.

2.3 ストレージ

クラウド環境では VM やコンテナを使用して 1 台のリソースが共有されるため，DRAM の使用効率が低く，ストレージを利用する必要がある．Solid State Drive（SSD）は高速な読み書き速度と低い消費電力を特徴としたストレージで，3DNAND フラッシュメモリ [16] や 3D XPoint [17] が提案されるなど，現在研究が勧められている [18]．CPU，DRAM，および SSD で構成されるシステムでは，初めに CPU が命令のフェッチ，デコード，実行，結果の出力を行う．実行される命令やデータは，DRAM への load/store 命令によりアクセスされる．SSD 内のデータは，DRAM ページキャッシュまたはバッファキャッシュに転送することで使用可能となる．DRAM 容量が不足している場合，DRAM の容量を広げるためキャッシュが解放され，SSD へのアクセス頻度が上昇する．また，未使用の DRAM 内のデータを一時的に SSD に退避するスワップアウトや，SSD の内容をメモリに転送するスワップインが実行される．スワップ操作により，実際の DRAM 容量より多くのメモリがあるように振る舞う.

3. 関連研究

Sathya ら [19] は複数の FHE 暗号方式を分類し，Editor ら [20] は各 FHE ライブラリで実行できる操作を比較している．しかしながら，これらの研究は実際の実行時の評価を含んでいない．本研究では，実行時の複数の FHE 暗号方式とライブラリの時間空間計算量を比較した．Gouert ら [21] は Terminator 2 Benchmark Suite と呼ばれるベンチマークを作成し，複数のライブラリ間で高速に処理できる計算ドメインとアプリケーションを比較している．しかしながら，異なるスキームを使用してライブラリ間の比較を行っている．本研究では，使用されるライブラリと暗号方式による影響を別々に検討している．Fawaz ら [22] は Microsoft SEAL ライブラリを使用して，BFV，BGV，および CKKS 方式の各暗号処理の実行時間を比較している．しかしながら，高速な暗号方式はライブラリによって異なる．本研究では，複数のライブラリを使用して暗号方式を比較した．Acar ら [23] は SHE [23]，LHE [24]，および FHE の理論の説明と，一般に公開されていない暗号方式の実行時間を評価している．本研究では，最新のオープンソースライブラリの実行時間を比較した．Doan ら [25] は複数の暗号方式とライブラリの実行時の評価を行っている．この評価では，BFV，BGV，CKKS 方式について暗号文多項式の次数 N を変化させた際のライブラリ間の実行時間を比較しているが，本研究では bootstrapping を行わずに連続して乗算することが可能な最大回数を変化させた際の時間空間計算量を比較した.

FHE の時間計算量が大きい課題に対して，ハードウェアアクセラレーションも検討されている．Jung ら [27] は GPU に適した FHE アルゴリズムを検討し，Riazi ら [28] と Beirendonck ら [30] は FPGA，Kim ら [29] は ASIC を使用した新しいアーキテクチャを提案している．これらの研究は，暗号処理のボトルネックとメモリ帯域幅を分析している．本研究では，TFHE 亜種方式における利用可能な DRAM 容量を変化させた際の実行時間や実装効率，SSD 帯域幅を評価し，TFHE 亜種方式の高速な実行に必要な DRAM 容量の検討の参考になる結果を示した.

4. FHE 暗号方式とライブラリの評価

OpenFHE，Lattigo，および TFHEpp ライブラリでそれぞれ実装された BFV，BGV，CKKS，TFHE 亜種方式を比較した．評価は以下の 3 つのステップで行った.

- 暗号文に対して準同型算術演算を行い，複数のメッセージを 1 つの平文にパッキングすることが可能という特徴を持つ BFV，BGV，CKKS 方式の比較を行った．Lattigo および OpenFHE ライブラリを用いて複数の暗号文間の乗算を行う際の時間空間計算量を計測した.
- BFV，BGV，CKKS 方式とは異なり，平文 $\mathcal{M}$ がバイナリで定義され，table lookup により単変数関数の準同型演算を行う TFHE 亜種方式について評価した．OpenFHE と TFHEpp ライブラリを用いて，sk（shared key）の作成，gk（gate key）の作成，8 bit の固定長の実数の暗号化，bootstrapping，復号を行う際の時間空間計算量を比較した.
- 実数の評価が可能な CKKS 方式と TFHE 亜種方式の比較を行った．なお，4.2.1 項で行った BFV，BGV，CKKS 方式を比較した結果と，4.2.3 項で行った CKKS 方式と TFHE 亜種方式を比較した結果を参照することで，BFV 方式と TFHE 亜種方式の比較，および BGV 方式と TFHE 亜種方式の比較が可能である．CKKS 方式では OpenFHE ライブラリ，TFHE 亜種方式では TFHEpp ライブラリを用いた．暗号処理の仕組みが異なる CKKS 方式と TFHE 亜種方式で公平な比較を

表 3 BFV，BGV，CKKS 方式の比較で使用するパラメータ
Table 3 Parameters used to compare BFV, BGV, and CKKS schemes.

(a) Parameters

暗号方式	t	N	batch
BFV	65,537	32,768	32,768
BGV	65,537	32,768	32,768
CKKS	65,537	65,536	32,768

(b) Ciphertext modulus q

暗号方式	ライブラリ	乗算深度				
		2	4	6	8	10
BFV	Lattigo	118	177	235	294	352
	OpenFHE	118	177	236	295	354
BGV	Lattigo	118	235	352	410	583
	OpenFHE	118	236	354	472	590
CKKS	Lattigo	146	236	326	416	506
	OpenFHE	145	235	325	415	505

表 4 TFHE 亜種方式の評価で使用するパラメータ
Table 4 Parameters used to evaluate Zama's variant of TFHE scheme.

ライブラリ	N	n	q	gadgetBase	baseSK
OpenFHE	2,048	512	2^{27}	2^7	2^7
TFHEpp	2,048	500	2^{32}	2^6	2^4

行うため，任意の値の乗算を行う場合とベクトル内積を計算する場合について，実行時間を比較した．

4.1 実行環境

表 3 に 4.2.1 項で BFV，BGV，CKKS 方式を比較する際に使用したパラメータを示す．batch は，エンコーディングの際に 1 つの平文にパッキングされるメッセージの数を表す．乗算深度は，bootstrapping を行わずに連続して乗算することが可能な最大回数を表す．また，**表 4** に 4.2.2 項で TFHE 亜種方式を評価する際に使用したパラメータを示す．TFHE 亜種方式では，暗号文は LWE 問題 [26] に基づくベクトルと RLWE 問題に基づく多項式環の 2 つの形式を用いる．GadgetBase は，RLWE に基づく暗号文間の乗算のための decomposition と呼ばれる操作において，暗号文のノイズを削減するために使用されるパラメータである．BaseSK は，RLWE に基づく暗号文を LWE 問題に基づく暗号文に変換するための key switching と呼ばれる操作で使用する．**表 5** に 4.2.3 項における CKKS 方式と TFHE 亜種方式間で (3) 任意の値の乗算および (4) ベクトル内積の実行時間を比較した際に用いたパラメータを示す．各評価において，128 ビットのセキュリティを満たす最速のパラメータを使用した．CPU は AMD Ryzen 7 5700G@3.80 GHz を使用し，8 つのコアと 16 の論理 CPU を備えている．DRAM は 16 GB の DDR4 を使用し，ストレージは容量が 238 GB の SK hynix PC711 SSD を用いた．

表 5 CKKS 方式の式 (3)，式 (4) の評価で使用するパラメータ
Table 5 Parameters used for the evaluation of CKKS scheme in Eq. (3) and Eq. (4).

n	t	N	batch	q	乗算深度
式 (3)					
2	65,537	8,192	4,096	183	2
20	65,537	65,536	32,768	955	20
40	65,537	65,536	32,768	1,475	26
60	65,537	65,536	32,768	1,640	29
80	65,537	65,536	32,768	1,750	31
100	65,537	65,536	32,768	1,805	32
式 (4)					
2	65,537	4,096	2	78	1
20	65,537	4,096	32	78	1
40	65,537	4,096	64	78	1
60	65,537	4,096	64	78	1
80	65,537	4,096	128	78	1
100	65,537	4,096	128	78	1

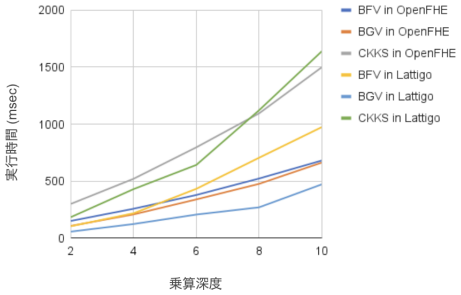


図 2 OpenFHE と Lattigo に実装された BFV，BGV，および CKKS 方式の乗算深度を変化させた際の実行時間の比較
Fig. 2 Comparison of the execution time of BFV, BGV, and CKKS schemes using openFHE and Lattigo libraries.

4.2 評価結果

4.2.1 BFV，BGV，CKKS 方式の比較

図 2 に OpenFHE と Lattigo を使用して BFV，BGV，および CKKS 方式の乗算深度を変化させた際の実行時間の比較を示す．暗号方式の比較では，すべての乗算深度で BGV，BFV，CKKS 方式の順に高速である．すべての暗号方式とライブラリの組合せの中で，Lattigo の BGV 方式が最速である．BFV と CKKS 方式では，乗算深度が小さい場合は Lattigo が高速であり，大きい場合は OpenFHE が高速である．**図 3** に OpenFHE と Lattigo を使用して BFV，BGV，および CKKS 方式の乗算深度を変化させた際のメモリ使用量を示す．CKKS 方式は BFV と BGV 方式に比べてメモリ使用量が多い．すべての暗号方式および乗算深度に対して，OpenFHE はより少ないメモリ容量で実行可能である．

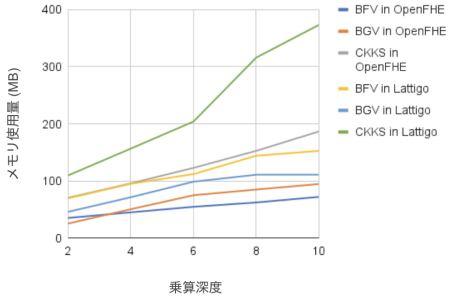
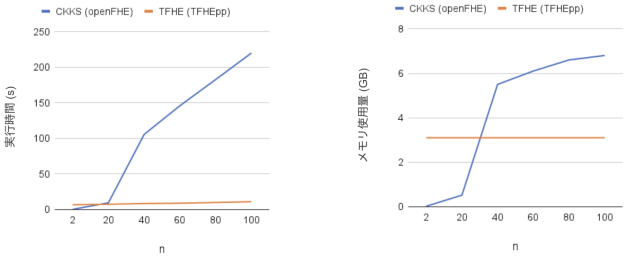


図 3 OpenFHE と Lattigo に実装された BFV，BGV，および CKKS 方式の乗算深度を変化させた際のメモリ使用量の比較

Fig. 3 Comparison of memory usage of BFV, BGV, and CKKS schemes using OpenFHE and Lattigo libraries.



(a) 実行時間　(b) メモリ使用量

図 4 CKKS 方式と TFHE 亜種方式における式 (3) の時間空間計算量の比較

Fig. 4 Comparison of the execution status of Eq. (3) between CKKS and Zama's variant of TFHE.

表 6 TFHE 亜種方式が実装された OpenFHE と TFHEpp の実行時間の比較

Table 6 Comparison of the execution time of Zama's variant between OpenFHE and TFHEpp.

ライブラリ	sk の生成 (ms)	gk の生成 (ms)	暗号化 (ms)	bootstrapping (ms)	復号 (ms)
OpenFHE	0.801	3,652.380	0.040	463.729	0.007
TFHEpp	0.041	6,442.310	0.018	26.865	0.003

4.2.2 TFHE 亜種方式の時間空間計算量

表 6 に OpenFHE と TFHEpp ライブラリを用いて TFHE 亜種方式を実行した際の実行時間を示す．Gate Key の生成と bootstrapping の実行時間が長いことが分かる．gate key の生成において，OpenFHE は TFHEpp よりも 1.764 倍高速である．一方で，bootstrapping では TFHEpp は OpenFHE よりも 17.261 倍高速である．したがって，bootstrapping が複数回実行される場合は TFHEpp が高速である．両ライブラリとも，gate key の生成中にメモリ使用量が増加し，完了時には約 3.3 GB に達する．その後実行される bootstrapping では，メモリ使用量は約 3.3 GB で推移する．5 章で詳細な評価を行った．

4.2.3 CKKS 方式と TFHE 亜種方式の比較

この実験では，式 (3) で表される任意の複数の値の乗算，式 (4) で表されるベクトル内積について，OpenFHE ライブラリに実装された CKKS 方式と TFHEpp ライブラリに実装された TFHE 亜種方式の時間空間計算量を比較した．$x_i, a_i, b_i \in \mathbb{R}$ とすると，式 (3) と式 (4) は $\boldsymbol{x} = (x_0, x_1, \ldots, x_{n-1})$，$\boldsymbol{a} = (a_0, a_1, \ldots, a_{n-1})$，$\boldsymbol{b} = (b_0, b_1, \ldots, b_{n-1})$ を用いて以下のように定義される．

$$f(\boldsymbol{x}) = \prod_{i=0}^{n-1} x_i \tag{3}$$

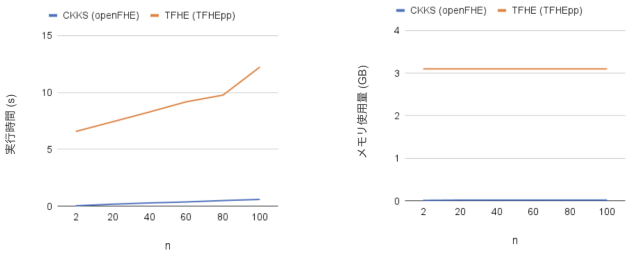
$$\boldsymbol{a} \cdot \boldsymbol{b} = \sum_{i=0}^{n-1} a_i b_i \tag{4}$$

本実験では，x_i, a_i, b_i は 8 bit の固定長で 0 を指定した．

式 (3) の評価　式 (3) を実行する際，CKKS 方式では，n 個のメッセージがパッキングされた暗号文多項式に対して Rotation と呼ばれる操作を行い，平文ベクトルが 1 つずつシフトされた新しい暗号文を作成する．それらの暗号文を掛け合わせることで，暗号化された n 個のメッセージの乗算が完了する．乗算が連続して行われるため，一定の乗算深度に達するたびに bootstrapping を実行する必要がある．一方で，TFHE 亜種方式では，各メッセージに対応する n 個の暗号文を作成し，それらを乗算する．毎回の乗算後に programmable bootstrapping を実行する．

図 4 に CKKS 方式および TFHE 亜種方式を使用して式 (3) を実行する際のデータ数 n の変化にともなう時間空間計算量の比較を示す．データ数 n が 20 未満の場合，CKKS 方式の時間空間計算量が小さく，40 以上の場合，TFHE 亜種方式の時間空間計算量が小さい．CKKS 方式ではパッキングされた暗号文に対して n 回，Rotation と乗算を行う必要がある．乗算回数が増加すると，必要な bootstrapping の回数も増加する．データ数 n が 2，20，40，60，80，100 の場合，それぞれ 0 回，0 回，2 回，3 回，4 回，5 回の bootstrapping が必要である．1 回の bootstrapping に必要な時間空間計算量が大きいため，必要な bootstrapping 回数が増加すると，TFHE 亜種方式と比較して効率が著しく低下する．TFHE 亜種方式では，n 回 bootstrapping が必要であり，時間空間計算量はデータ数 n に比例して増加する．ただし，programmble bootstrapping に必要な時間空間計算量が小さいため，著しい増加は見られない．

式 (4) の評価　式 (4) を実行する際，CKKS 方式では，n 個のメッセージをそれぞれパッキングした 2 つの暗号文を乗算する．その後，乗算結果の暗号文にパッキングされている要素間の加算を行う．一方で，TFHE 亜種方式では，各ベクトルの要素のセットごとに乗算と programmable bootstrapping を実行する必要がある．その後，n 個の乗算結果の加算を行う．**図 5** に CKKS および TFHE 亜種方式を使用して式 (4) を実行する際のデータ数 n の変化にともなう時間空間計算量の比較を示す．すべてのデータ数 n において，CKKS 方式は TFHE 亜種方式より小さい時間空間計算量で演算が可能である．CKKS 方式では，n 個の 2 要素間の乗算結果は 2 つの暗号文間の 1 回の乗算のみで得



(a) 実行時間　(b) メモリ使用量

図 5 CKKS 方式と TFHE 亜種方式における式 (4) の時間空間計算量の比較

Fig. 5 Comparison of the execution status of Eq. (4) between CKKS and Zama's variant of TFHE.

られ，bootstrapping は不要である．TFHE 亜種方式では，式 (3) と同様に，各ベクトル要素のセットごとに n 回の乗算と programmable bootstrapping を行う必要がある．

5. DRAM 容量制限時の TFHE 亜種方式の評価

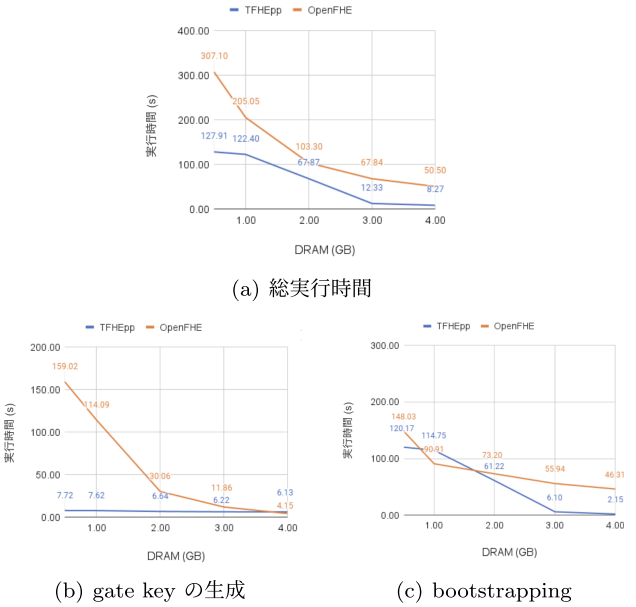
FHE 暗号は時間空間計算量が大きいが，実際はクラウド上で多く使用され，十分な DRAM が割り当てられていない状態で実行されることが多い．一方で，TFHE 亜種方式では，メッセージを 1 つずつ逐次的に処理するため，並列処理により高速化が可能だが，その際にメモリ使用量が多くなるといった特徴を持つ．したがって，本章では DRAM 容量が制限されている状態で，TFHE 亜種方式が実装された OpenFHE と TFHEpp ライブラリの実行時間，SSD 帯域幅，および実装効率を比較する．本実験では 100 のメッセージに対する一連の暗号処理（鍵生成，暗号化，bootstrapping，復号）の評価を行った．FHE がクラウド上で使用される環境をスケールダウンした実験を行い，DRAM を 0.5 GB，1 GB，2 GB，3 GB，および制限なしに設定した場合の結果を考察した．

5.1 実行環境

CPU は AMD Ryzen 7 5700G @ 3.8 GHz を使用し，8 つのコアと 16 の論理コアを備える．実験では DRAM 容量を制限するために Docker コンテナを使用した．スワップ先の SSD として，最大 238 GB まで使用可能な SK hynix PC711 を使用した．表 4 に OpenFHE および TFHEpp で使用したパラメータを示す．

5.2 実験結果

実行時間と SSD 帯域幅　図 6 に DRAM 容量が制限された際の TFHEpp および OpenFHE ライブラリにおける TFHE 亜種方式の総実行時間，gate key の生成時間，および bootstrapping の実行時間の変化を示す．メモリが制限されていない場合，gate key の生成では OpenFHE が TFHEpp より高速である．しかしながら，DRAM 容量の



(a) 総実行時間

(b) gate key の生成　(c) bootstrapping

図 6 DRAM 容量が制限された際の TFHEpp および OpenFHE ライブラリにおける TFHE 亜種方式の実行時間

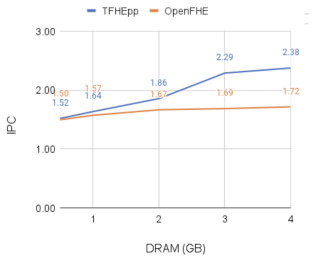
Fig. 6 Comparison of the execution time when the DRAM is limited in Zama's variant of TFHE with TFHEpp and OpenFHE.

表 7 TFHE 亜種方式が実装された TFHEpp および OpenFHE における DRAM 容量が制限された場合の SSD 帯域幅の変化

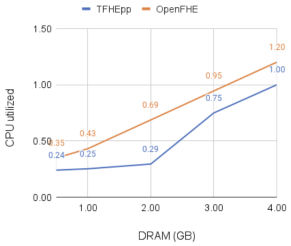
Table 7 Comparison of SSD bandwidth changes when the DRAM is limited in Zama's variant of TFHE between TFHEpp and OpenFHE.

暗号方式	ライブラリ	DRAM 容量制限値（GB）				
		0.5	1	2	3	制限なし
gate key の生成						
OpenFHE	read	41.902	47.347	64.349	31.816	0.166
	write	39.723	45.412	99.876	41.086	0.179
TFHEpp	read	0.635	0.103	0.138	0.053	0.034
	write	317.097	254.194	160.888	0.179	0.104
bootstrapping						
OpenFHE	read	31.358	19.946	17.116	7.985	0.005
	write	2.811	10.077	14.553	8.008	0.006
TFHEpp	read	33.020	28.433	30.074	38.378	0.724
	write	6.318	10.995	33.147	44.998	0.077

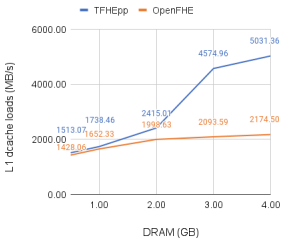
制限により OpenFHE における gate key の生成時間が著しく増加する．**表 7** に DRAM 容量が制限された場合の TFHE 亜種方式が実装された TFHEpp と OpenFHE における SSD 帯域幅の変化を示す．OpenFHE の gate key の生成では，DRAM 容量の制限による帯域幅の増加は見られない．OpenFHE の実装では gate key の生成中に大量のデータにアクセスしない傾向があり，DRAM 容量の制限により SSD へのアクセスの頻度が増加しないことが原因である．一方で，DRAM 容量の制限により演算処理に使用するメモリが不足し，実行時間が増加する．TFHEpp で gate key を生成する場合，DRAM 容量の制限により SSD 書き込み帯域幅が大幅に増加する．TFHEpp は gate key の生



(a) IPC



(b) CPU utilized



(c) L1 Dcache load

図 7 DRAM 容量が制限された際の TFHEpp および OpenFHE ライブラリにおける TFHE 亜種方式の実装効率

Fig. 7 Comparison of changes in implementation efficiency when the DRAM is limited in Zama's variant of TFHE between TFHEpp and OpenFHE.

成中に大量のデータにアクセスし，DRAM 容量の制限によりデータアクセス用のメモリが不足すると考えられる．

Bootstrapping では，メモリ制限がない場合，TFHEpp は OpenFHE より高速に実行可能である．しかしながら，TFHEpp は OpenFHE よりもメモリ制限の影響をうけ，3 GB 以下で実行時間が急速に増加する．OpenFHE，TFHEpp ともに，DRAM 容量制限により bootstrapping 時の SSD 帯域幅が増加しないことから，bootstrapping の演算処理の実行に必要なメモリが不足していることが分かる．総実行時間では，OpenFHE の gate key の生成時間の増加により，TFHEpp がすべての DRAM 容量の制限値で高速であった．

実装効率 図 7 に TFHE 亜種方式が実装された OpenFHE と TFHEpp における DRAM 容量が制限された場合の実装効率の変化を示す．DRAM 容量の制限がない場合，TFHEpp は OpenFHE より IPC が高く，計算効率が良いことが分かる．しかしながら，DRAM 容量の制限により TFHEpp の IPC は 1.5 に低下し，OpenFHE を用いて DRAM を 0.5 GB に制限した場合と同程度の値となる．また，CPU utilized の最大値は 16 だが，TFHEpp，OpenFHE ともに低い値であり，処理の並列性が非常に低いことが分かる．デフォルトでは，TFHEpp は全プロセスを単一スレッドで実行し，OpenFHE は鍵生成に 16 スレッド，bootstrapping に 1 スレッドを使用している．

L1D キャッシュのロード数は TFHEpp が 2.5 倍多い．DRAM 容量の制約により，TFHEpp の L1D キャッシュのロード数は OpenFHE と同程度の 1,513（MB/秒）まで減少する．CPU の演算性能の低下により，L1D キャッシュの利用率が低下していることが分かる．

6. 予備実験

4 章では，FHE 暗号方式・ライブラリのマイクロベンチマークを作成し，FHE の基本処理を評価した．本予備実験では，FHE 暗号方式・ライブラリを実アプリケーション内で評価し，FHE 暗号処理の負荷が大きいことを確認することを目的とする．具体的には，実社会で汎用的に使われている HElib ライブラリを用いたゲノム秘匿情報検索アプリケーションを使用した．また，暗号方式は，ゲノム秘匿検索の処理内で負荷が大きい，整数を暗号化した状態で比較することを高速に行うことが可能な BGV 方式を用いた．5 章で行った DRAM 容量制限時の TFHE 亜種暗号方式の SSD 帯域幅の評価では，TFHE 亜種方式の基本的な処理ではストレージアクセスに比べて，CPU と DRAM 間の load/store 処理または CPU の演算処理がボトルネックであることを示した．本予備実験では，ゲノム秘匿検索アプリケーションにおける CPU と DRAM 間の load/store 処理と CPU の演算処理の負荷を比較し，ボトルネックを特定することを目的とする．

6.1 ゲノム秘匿情報検索

アプリケーションについて述べる．ゲノム秘匿検索では，複数の塩基配列データを照合することで，特定の塩基配列が病気の原因であるか判定を行う．クライアントサーバ型通信で，クライアントがサーバのゲノムデータベース（PBWT [32]）に対して特定の塩基配列が最長マッチを持つか問合せを行う．具体的な手順を以下に示す．

(1) クライアントは検索したいクエリを全文暗号化し，その他のパラメータと一緒にサーバに送信する．
(2) サーバは FHE で暗号化された状態でクエリと PBWT の照合処理を行い，結果をクライアントに送信する．
(3) クライアントはサーバから送られてきたデータを復号し，結果を得る．

このとき，お互いに持っているゲノム塩基配列や調べたい内容を秘匿しながら行うことが FHE により実現され，個人情報の漏洩が防がれている．ここで，クエリ長の増加にともない，乗算深度が大きくなるため，bootstrapping の導入が必要である．

6.2 評価環境

CPU は Xeon E5-2643 v3 を使用し，6 つのコアと 12 の論理コアを備える．DRAM は 512 GB の DDR4 を使用し，ストレージは容量が 118 GB の Intel Optane SSD 800P を用いた．すべての評価は Docker のコンテナ上で行った．

ゲノムデータベースについては 1 サンプルあたり 10,000 文字のデータを 2,184 サンプル用意した．FHE ライブラリ

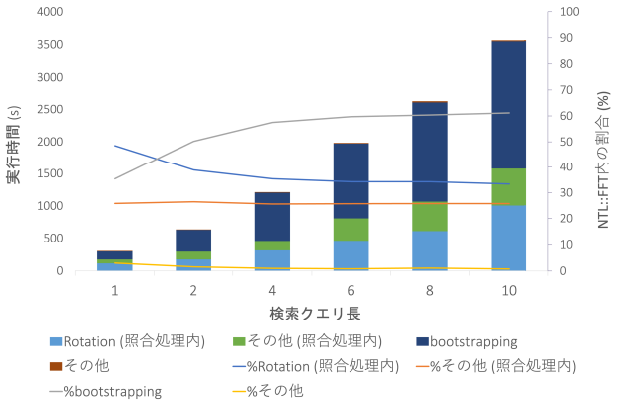


図 8 処理全体における関数ごとの NTL::FFT に与える負荷の内訳
Fig. 8 Breakdown of the load imposed on NTL::FFT for each function during execution.

は HElib [10] を用いた.

6.3 評価結果

6.3.1 処理全体の負荷

HElib ライブラリの BGV 方式を用いた場合，暗号文は多項式であるため，係数の小さい高次の多項式どうしの演算を表す NTL::FFT の負荷が処理全体の約 50%を占める．そこで，NTL::FFT の負荷が HElib のどの関数により引き起こされているのか分析を行った．**図 8** に検索クエリ長を変化させた場合の NTL::FFT を呼び出す関数の内訳の変化を示す．検索クエリ長が 1 伸びるにつれて，PBWT と検索クエリの照合回数は 2 回ずつ増加するが，1 回の照合にかかる負荷は変わらない．bootstrapping も 2 回増加するが，1 回ごとの処理の重さはクエリ長に依存せず一定である．クエリ長が 1 のとき，NTL::FFT の処理全体に対して，Rotation が 48.41%，bootstrapping が 35.43%を占める．Rotation は，暗号化された状態で適切な検索位置を特定するために実行される．その後，クエリ長の増加にともない，Rotation の割合は緩やかに減少し，bootstrapping の割合が増加する．準同型論理演算が可能な TFHE 暗号を用いることで，効率的に複数の暗号文間で元のメッセージが一致しているか判定することが可能である．

6.3.2 処理のボトルネック

ゲノム秘匿検索の CPU 使用率は，bootstrapping の前半では約 20%，後半では約 50%，値の照合処理では約 90%である．CPU 使用率が高い箇所について，CPU の演算処理と CPU への load/store 命令の処理の重さを比較するため，hyperthreading 機能の有効/無効にともなう実行時間の変化を測定した．hyperthreading のような仮想的な並列化では計算性能は約 2 倍となるが，メモリへの load/store は逐次処理になる．OpenMP を用いて独立性の高い計算部分をマルチスレッド化し，計算処理性能を最大限に引き出すと予想される，物理コア数の倍にあたる並列数 24 まで計測を行った．**表 8** に無効時並列数 12，有効時並列数 12，有効時並列数 24 の場合の実行時間を示す．実行時間は照合処理，bootstrapping ともに，無効時（並列数 12）< 有効時（並列数 12）< 有効時（並列数 24）となる．hyperthreading 有効時も，実行時間は並列数の増加にともない物理コア数と同じ並列数 12 まで減少し，その後は増加する．12 以上の並列化にともなう CPU コストの大きな変化が見られないことや，IPC が 2.49 から 2.19 に低下することから，メモリから CPU への load/store 命令の処理がボトルネックとなり，2 倍になった CPU の演算処理性能を活用できていないことが分かる．以上から，HElib ライブラリの BGV 方式を用いた場合，CPU 使用率が高い箇所について，CPU への load/store 命令の処理がボトルネックと考えられる．

表 8 hyperthreading 有効時・無効時の比較
Table 8 Comparison between when hyperthreading is enabled and disabled.

	無効時（並列数 12）	有効時（並列数 12）	有効時（並列数 24）
総実行時間（s）	477.33	605.57	623.67
照合処理（s）	144.17	155.43	166.41
bootstrapping（s）	164.96	223.12	230.30
IPC	2.56	2.49	2.19

7. まとめ

本稿では初めに，複数の暗号ライブラリ（OpenFHE, Lattigo, TFHEpp）および暗号方式（BFV, BGV, CKKS, TFHE 亜種）の時間空間計算量を比較した．整数を暗号処理する場合，Lattigo の BGV がすべての暗号方式，ライブラリの組合せの中で最速であった．浮動小数点を評価できる CKKS 方式では，乗算深度が小さいときは Lattigo が高速であり，乗算深度が大きいときは OpenFHE が高速であることを示した．また，CKKS 方式と TFHE 亜種方式の比較では，任意の値を 40 回以上乗算する場合，TFHE 亜種方式の時間空間計算量が小さいが，ベクトル内積を計算する場合は CKKS 方式が効率的であった．

2 番目に，TFHE 亜種方式が実装された OpenFHE と TFHEpp ライブラリを使用して，DRAM 容量を複数の値で制限した場合の実行時間，SSD 帯域幅，および実装効率を比較した．DRAM 容量が制限されている場合，TFHEpp は OpenFHE よりも高速である．メモリ制限により OpenFHE において gate key 生成時の演算処理に必要なメモリが不足し，実行時間が増加することが原因である．

また，予備実験では，HElib ライブラリの BGV 方式を用いて実装されたアプリケーションであるゲノム秘匿検索を使用して，暗号処理全体の負荷を計測した．実験から，bootstrapping と暗号文どうしの値の比較の負荷が大きいことが分かった．

謝辞 本研究の一部は，キオクシア株式会社の支援を受けて実施したものである．

参考文献

[1] Gentry, C.: A FULLY HOMOMORPHIC ENCRYPTION SCHEME, PhD Thesis, Stanford University (2009).

[2] Fan, J. and Vercauteren, F.: Somewhat practical fully homomorphic encryption, Cryptology ePrint Archive, Report 2012/144 (2012).

[3] Brakerski, Z., Gentry, C. and Vaikuntanathan, V.: (leveled) fully homomorphic encryption without bootstrapping, *Proc. 3rd Innovations in Theoretical Computer Science Conference* (*ITCS '12*), pp.309–325, Association for Computing Machinery (2012).

[4] Cheon, H.J., Kim, A., Kim, M. and Song, Y.: Homomorphic encryption for arithmetic of approximate numbers, *International Conference on the Theory and Application of Cryptology and Information Security* (*Asiacrypt '17*), pp.409–437, Cryptology ePrint Archive, Report 2017/421 (2017).

[5] Chillotti, I., Joye, M. and Paillier, P.: Programmable Bootstrapping Enables Efficient Homomorphic Inference of Deep Neural Networks, *Cyber Security, Cryptology, and Machine Learning* (*CSCML '21*), pp.1–19, Cryptology ePrint Archive, Report 2021/091 (2021).

[6] OpenFHE, available from ⟨https://github.com/openfheorg/openfhe-development⟩.

[7] Lattigo, available from ⟨https://github.com/tuneinsight/lattigo⟩.

[8] TFHEpp, available from ⟨https://github.com/kenmaro3/TFHEpp⟩.

[9] Tirmazi, M., Barker A., Deng, N., et al.: Borg: the Next Generation, *5th European Conference on Computer Systems* (*EuroSys '20*), pp.1–4, Association for Computing Machinery (2020).

[10] HElib, available from ⟨https://github.com/homenc/HElib⟩.

[11] Chillotti, I., Gama, N., Georgieva, M. and Izabachène, M.: TFHE: fast fully homomorphic encryption over the torus, *Journal of Cryptology*, Vol.33, No.1, pp.34–91 (2020).

[12] Brakerski, Z., Gentry, C. and Vaikuntanathan, V.: (Leveled) fully homomorphic encryption without bootstrapping, *Proc. 3rd Innovations in Theoretical Computer Science Conference* (*ITCS '12*), pp.309–325, Association for Computing Machinery (2012).

[13] awesome-he, available from ⟨https://github.com/jonaschn/awe-some-he⟩.

[14] Google, available from ⟨https://github.com/google/fully-hom-omorphic-encryption⟩.

[15] TFHE, available from ⟨https://github.com/tfhe/tfhe.git⟩.

[16] Seiichi Aritome: NAND Flash Memory Technologies, ISBN: 978-1-119-13260-8 (2015).

[17] Hady, T.F., Foong, A., Veal, B., et al.: Platform Storage Performance With 3D XPoint Technology, *Platform Storage Performance With 3D XPoint Technology*, Vol.105, No.9, pp.1822–1833 (2017).

[18] Do, J., Sengupta, S. and Swanson, S.: Programmable solid state storage in future cloud datacenters, *Comm. ACM*, Vol.62, No.6, pp.54–62 (2019).

[19] Sathya, S.S., Vepakomma, P., Raskar, R., Ramachandra, R. and Bhattacharya, S.: A Review of Homomorphic Encryption Libraries for Secure Computation, arXiv preprint arXiv:1812.02428 (2018).

[20] Editor, I. and El-Yahyaoui, A.: Fully Homomorphic Encryption: State of Art and Comparison, *International Journal of Computer Science and Information Security* (*IJCSIS '16*), Vol.14, No.4 (2016).

[21] Gouert, C., Mouris, D. and Tsoutsos, G.N.: SoK: New Insights into Fully Homomorphic Encryption Libraries via Standardized Benchmarks, *Proc. Privacy Enhancing Technologies* (*PoPETs '23*), No.3, pp.154–172, Cryptology ePrint Archive, Report 2022/425 (2022).

[22] Fawaz, M.S., Belal, N., Elrefaey, A. and Fakhr, W.M.: A Comparative Study of Homomorphic Encryption Schemes Using Microsoft SEAL, *Journal of Physics Conference Series*, Vol.2128, 012021 (2021).

[23] Acar, A., Aksu, H., Uluagac, S.A. and Conti, M.: A Survey on Homomorphic Encryption Schemes: Theory and Implementation, *ACM Computing Surveys*, Vol.51, No.4, Article No.79, pp.1–35 (2018).

[24] Armknecht, F., Boyd, C., Carr, C., et al.: A Guide to Fully Homomorphic Encryption, *International Association for Cryptologic Research* (*IACR*), Report 1192, pp.1–35 (2015).

[25] Doan, T.V.T., Messai, M., Gavin, G. and Darmont, J.: A survey on implementations of homomorphic encryption schemes, *The Journal of Supercomputing*, Vol.79, pp.15098–15139 (2023).

[26] Regev, O.: On lattices, learning with errors, random linear codes, and cryptography, *Journal of the ACM* (*JACM*), Vol.56, No.6, pp.1–40 (2009).

[27] Jung, W., Kim, S., Ahn, H.J., Cheon, H.J. and Lee, Y.: Over 100x Faster Bootstrapping in Fully Homomorphic Encryption through Memory-centric Optimization with GPUs, *IACR Trans. Cryptographic Hardware and Embedded Systems*, No.4, pp.114–148 (2021).

[28] Riazi, S.M., Laine, K., Pelton, B. and Dai, W.: HEAX: An Architecture for Computing on Encrypted Data, *Proc. 25th International Conference on Architectural Support for Programming Languages and Operating Systems* (*ASPLOS '20*), pp.1295–1309, arXiv preprint arXiv:1909.09731 (2020).

[29] Kim, S., Kim, J., Kim, J.M., et al.: BTS: An accelerator for bootstrappable fully homomorphic encryption, *Proc. 49th Annual International Symposium on Computer Architecture*, pp.711–725, arXiv preprint arXiv:2112.15479 (2021).

[30] Beirendonck, V.M., D'Anvers, J., Turan, F., Verbauwhede, I.: FPT: A Fixed-Point Accelerator for Torus Fully Homomorphic Encryption, *Proc. 2023 ACM SIGSAC Conference on Computer and Communications Security* (*CCS '23*), pp.741–755, Cryptology ePrint Archive, Paper 2022/1635 (2022).

[31] Ishimaki, Y., Imabayashi, H., Shimizu, K. and Yanama, H.: Privacy-preserving string search for genome sequences with FHE bootstrapping optimization, *2016 IEEE International Conference on Big Data* (*Big Data*), pp.3989–3991 (2016).

[32] Durbin, R.: Efficient haplotype matching and storage using the Positional Burrows-Wheeler Transform (PBWT), *Bioinformatics*, Vol.30, No.9, pp.1266–1272 (2014).

推薦文

時間空間計算量が一般に大きい完全準同型暗号（FHE）とゲノム情報検索の組合せを例に，FHE の代表的な実装ライブラリ HElib を用いて HW 性能も考慮した調査分析を詳細に行っており，FHE やその応用の研究者およびエンジニアなどにとって非常に興味深い研究結果と考えられ

る．特に CPU，DRAM，SSD などの HW およびサーバやストレージのプロトコル・インタフェースの性能をふまえたボトルネックや並列化可能性の調査分析結果は，今後の FHE の研究開発の進展に寄与する有益な結果と思われる．FHE の理論研究やライブラリ開発は今もさかんに行われているが，FHE の実用化には本研究のように実行環境の分析も含めた融合領域のアプローチも重要であり，論文の完成度も高いことから，本稿を推薦論文として推薦する．

（コンピュータセキュリティ研究会主査　千田浩司）

辻 有紗　（学生会員）

2022 年お茶の水女子大学理学部情報科学科卒業．2024 年同大学大学院人間文化創成科学研究科博士前期課程修了．現在，同大学大学院人間文化創成科学研究科博士後期課程在学中．セキュリティ・プライバシに関する研究に従事．

圓戸 辰郎

1976 年生．2002 年早稲田大学大学院理工学研究科機械工学専攻修士課程修了．2006 年東京大学大学院情報理工学系研究科知能機械情報学専攻博士課程単位取得退学．2015 年株式会社東芝入社後，2017 年から事業承継により株式会社キオクシア（旧東芝メモリ株式会社）へ異動，SSD の応用研究に従事．2023 年マツダ入社，DX および AI プラットフォームの研究開発に従事．

小口 正人　（正会員）

1995 年東京大学大学院博士課程了．博士（工学）．学術情報センター中核的研究機関研究員，東京大学生産技術研究所特別研究員，中央大学研究開発機構助教授，お茶の水女子大学助教授を経て，現在，同教授．ネットワークコンピューティング・ミドルウェアに関する研究に従事．IEEE，ACM，電子情報通信学会，日本データベース学会各会員．