効率的な暗号処理に向けた完全準同型暗号方式・ライブラリの比較

理学専攻 情報科学コース 2240657 辻 有紗　(指導教員：小口 正人)

1　はじめに

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)[1] 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．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．そこで本研究では，クラウドサーバで行う演算処理や，クライアントで行うデータのエンコーディングや暗号化の分析を行い，適切な暗号方式・ライブラリを選択する際の参考情報を提供することを目的とした．

2　FHE 暗号方式・ライブラリ

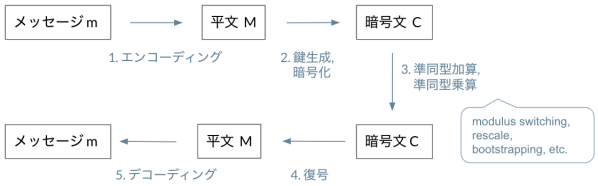


図 1: FHE 暗号の処理の流れ

表 1: BFV，BGV，CKKS 方式の比較で使用するパラメータ

(a) Parameters

暗号方式	t	N	batch
BFV	65537	32768	-
BGV	65537	32768	-
CKKS	65537	65536	32768

(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 2: TFHE 亜種方式の評価で使用するパラメータ

ライブラリ	N	n	q	gadgetBase	baseSK
OpenFHE	2048	512	2^{27}	2^7	2^7
TFHEpp	2048	500	2^{32}	2^6	2^4

図 1 に FHE 暗号の処理の流れを示す．FHE では，暗号化したいメッセージから平文に変換するエンコーディング，暗号化・復号時に使用する鍵の生成，(TFHE 方式では bootstrapping で使用する gate key の生成)，平文の暗号化，暗号文同士の演算，bootstrapping，復号という流れで処理が行われる．

表 3: CKKS 方式の式 (1)，式 (2) の評価で使用するパラメータ

n	t	N	batch	q	乗算深度
式 (1)					
2	65537	8192	4096	183	2
20	65537	65536	32768	955	20
40	65537	65536	32768	1475	26
60	65537	65536	32768	1640	29
80	65537	65536	32768	1750	31
100	65537	65536	32768	1805	32
式 (2)					
2	65537	4096	2	78	1
20	65537	4096	32	78	1
40	65537	4096	64	78	1
60	65537	4096	64	78	1
80	65537	4096	128	78	1
100	65537	4096	128	78	1

一方で，FHE では，BFV 方式，BGV 方式，CKKS 方式，TFHE 亜種方式など複数の暗号方式が提案されており，暗号処理の方法がそれぞれ異なる．BFV，BGV，CKKS 方式では，算術演算を使用して準同型演算を行う．BFV，BGV 方式では整数，CKKS 方式では実数の評価を行うことができる．また，1 つの暗号文に複数のメッセージを格納し，SIMD 演算を行うことが可能である．暗号文に蓄積したノイズを削減するための bootstrapping は，安全な状態で暗号文を復号し，復号結果を再び暗号化することで実現する．事前に設定する連続乗算可能回数 (乗算深度) を超える度に bootstrapping を実行する．

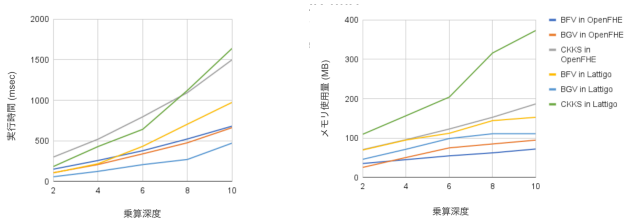
TFHE 方式では，暗号文を構成する各ビットが論理回路を使用して評価される．bootstrapping は，ノイズが一定量に削減された暗号文が格納された LUT(Lookup Table) と呼ばれるベクトルの適切な位置を参照することで行われる．bootstrapping は回路を構成する各論理ゲートで実行される．

本研究では，OpenFHE ライブラリと Lattigo ライブラリに実装された BFV，BGV，CKKS 方式と，OpenFHE ライブラリと TFHEpp ライブラリに実装された TFHE 亜種方式について評価を行った．表 1，表 2，表 3 に評価で使用したパラメータを示す．128bit security を満たす最速のパラメータを使用した．パラメータ t は平文を表す多項式の剰余，N，n は暗号文を表す多項式の次数，batch は 1 つの暗号文にパッキングされる平文の数，q は暗号文を表す多項式の剰余を表す．gadgetBase・baseSK は gate key の生成においてノイズを削減する際に使用するパラメータである．

3　評価

3.1　BFV，BGV，CKKS 方式の比較

図 2 に乗算時の計算量の比較を示す．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．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少ない場合は Lattigo が高速であり，乗算回数が多い場合は OpenFHE が高速である．また，全ての暗号方式において，OpenFHE は Lattigo に比べて少ないメモリで実行可能である．



(a) 実行時間　　(b) メモリ使用量

図 2: BFV，BGV，CKKS 方式の乗算時の比較

3.2　TFHE 亜種方式の評価

表 4: TFHE 亜種方式の実行時間の比較

ライブラリ	secret key の生成 (ms)	gate key の生成 (ms)	暗号化 (ms)	bootstrapping (ms)	復号 (ms)
OpenFHE	0.801	3652.380	0.040	463.729	0.007
TFHEpp	0.041	6442.310	0.018	26.865	0.003

表 4 に TFHE 亜種方式の実行時間の比較を示す．bootstrapping では TFHEpp は OpenFHE に比べて 17.261 倍高速であり，bootstrapping が複数回実行される場合は TFHEpp が高速である．

3.3　CKKS 方式と TFHE 亜種方式の比較

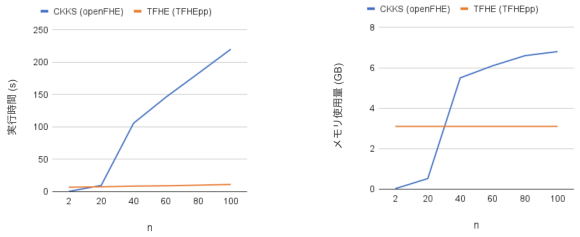
(1) 任意の複数の値の乗算, (2) ベクトル内積について，OpenFHE ライブラリの CKKS 方式と TFHEpp ライブラリの TFHE 亜種方式の計算量を比較した．$x \in \mathbb{R}$, $\boldsymbol{a} = (a_0, a_1, ..., a_{n-1})$, $\boldsymbol{b} = (b_0, b_1, ..., b_{n-1})$ とすると，(1) と (2) は以下のように定義される．

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

$$\boldsymbol{a} \cdot \boldsymbol{b} = \sum_{i=0}^{n-1} a_i b_i \tag{2}$$

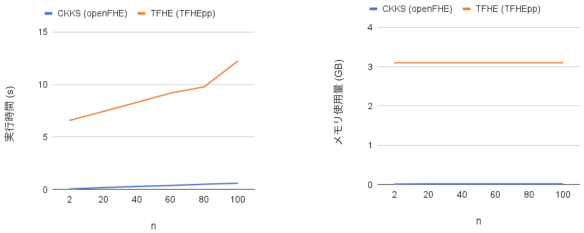
(1) の評価　CKKS 方式では，パッキングされた暗号文に対して n 回，Rotation と乗算を行う必要がある．1 回の bootstrapping に必要な計算量が大きいため，必要な bootstrapping 回数の増加に伴い，効率が著しく低下する．一方で，TFHE 亜種方式では，n 回 bootstrapping が必要だが，1 回の実行に必要な計算量が小さいため，著しい増加は見られない．

(2) の評価　CKKS 方式では，メッセージののパッキングを行うことで，n 個の 2 要素間の乗算結果は 2 つの暗号文間の 1 回の乗算のみで得られ，bootstrapping は不要である．TFHE 亜種方式では，任意の値の乗算と同様に，0 ～ n-1 番目の要素について，それぞれ乗算と bootstrapping を行う必要がある．



(a) 実行時間　　(b) メモリ使用量

図 3: CKKS 方式と TFHE 亜種方式における式 (1) の時間空間計算量の比較



(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 (2) の時間空間計算量の比較

3.4　クライアントの負荷の比較

表 5: クライアント側で行う処理の実行時間 (鍵生成，エンコーディング，暗号化の合計) の比較

暗号方式	ライブラリ	クライアント側の処理を行う機器		
		hpc5000 -xsl (ms)	Raspberry Pi 4 (ms)	Raspberry Pi Zero 2 W (ms)
TFHE 亜種	OpenFHE	3295.63	5904.02	295000.38
	TFHEpp	8242.40	50349.97	271984.86

表 5 にクライアント側の処理時間の比較を示す．hpc5000-xsl は 2 つの Intel(R) Xeon(R) Gold 5115 CPU@2.40GHz と 20 個の論理コアを持ち，DRAM 容量が 1920GB のサーバである．IoT クライアント端末との比較のために hpc5000-xsl で同じ処理を実行した．1 つの Cortex-A53@1GHz と 4 つの論理コアを持ち，DRAM 容量が 512MB である Raspberry Pi Zero 2 W など，性能が低い端末では，TFHE 亜種方式は CKKS 方式に比べて実行時間が増加する．特に，gate key の作成時間が長く，Raspberry Pi 4 から Rapsberry Pi Zero 2 W に変更することで，OpenFHE では約 74.29 倍, TFHEpp では約 4.53 倍増加する．実行時の DRAM 容量が 8GB から 512MB に減少することで，ストレージへの IO 待ち時間が増加することが原因である．

4　まとめと今後の課題

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．サーバで行う演算内容やクライアント側の処理を行う機器の性能に応じて，適切な暗号方式・ライブラリを選択する必要がある．今後は，IoT クライアント端末における TFHE 亜種方式の高速化に向けた検討を行う．

参考文献

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