DEIM Forum 2021 G31-1

IoT デバイスにおけるデータ活用のための準同型暗号を用いた暗号化の高速化

松本　茉倫†　　小口　正人†

† お茶の水女子大学　〒 112–8610 東京都文京区大塚 2-1-1
E-mail: †marin@ogl.is.ocha.ac.jp, oguchi@is.ocha.ac.jp

あらまし　スマートフォンを始めとする，IoT デバイスで取得したセンサデータを活用するためにクラウドサービスを利用した統計分析が普及している．IoT デバイスで取得したセンサデータの中には，位置情報などの秘匿性が高いデータが存在しており，必ずしも安全とは言えないクラウドサービス上では情報漏洩に備えて，個人情報を保護する必要がある．そこで，暗号文同士の加算・乗算が何度でも可能な完全準同型暗号 (以下 FHE: Fully Homomorphic Encryption) とあらかじめ設定した回数の範囲内で加算と乗算が可能な Leveled 準同型暗号 (以下 LHE: Leveled Homomorphic Encryption) が注目されている．しかし，FHE と LHE は暗号化に時間がかかり，暗号文サイズが大きいため，計算能力の低い IoT デバイスで利用するには向いていない．本研究では IoT デバイスで取得したデータの暗号化を高速化し，安全にデータ活用することを目的として，クライアント側では FHE や LHE より軽量な Somewhat 準同型暗号 (以下 SHE: Somewhat Homomorphic Encryption) で平文の暗号化を行い，代わりにサーバ側で SHE の暗号文から LHE の暗号文へ変換する手法を提案する．実験の結果，単純に LHE で平文を暗号化する場合と比べて，提案手法では IoT デバイスへの負荷と通信量が大幅に減らせることが示された．

キーワード　完全準同型暗号，Somewhat 準同型暗号，IoT デバイス

1　はじめに

スマートフォン・スマートウォッチなどの IoT デバイスの普及によって，行動履歴・心拍数・位置情報など様々なセンサデータを取得可能になった．それに伴い，クラウドサービス上で IoT デバイスから収集したデータを統計分析した結果を活用することが期待されている．IoT デバイスで取得したセンサデータの中には，秘匿性が高いデータが存在しており，必ずしも安全とは言えないクラウドサービス上では個人情報を平文のままで処理するべきではない．すなわち，IoT デバイスとクラウドサービス間等の通信の盗聴を防止するためだけではなく，不正アクセスなどによって起こる可能性のあるクラウドサービスからの情報漏洩に備えて，統計分析の際も個人情報を暗号化したままで処理を行う必要がある．そのような需要を満たすために，本研究では図 1 に示すような，IoT デバイスで行動履歴・心拍数といった個人情報を暗号化し，クラウドサービスで暗号化したままで統計分析した後で，データ分析者が分析結果のみを得るシステムを想定する．

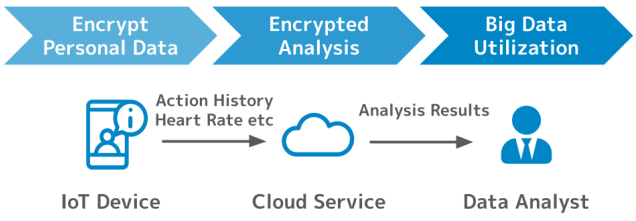

図 1: 想定するシステム

図 1 のような暗号化したままで統計分析するシステムを実現するためには，暗号文同士の加算・乗算が任意回数可能な性質を持つ完全準同型暗号 (以下 FHE: Fully Homomorphic Encryption) やあらかじめ設定した回数の範囲内で加算と乗算が可能な Leveled 準同型暗号 (以下 LHE: Leveled Homomorphic Encryption) が有用である．しかしながら，FHE と LHE は処理が重く，暗号文のサイズが大きいため通信量が多くなってしまうことが計算能力の低い IoT デバイス上での実装の課題となっている．

先行研究 [1] [2] では，AES を FHE に組み合わせることによってクライアント側の負荷を減らすことを提案し実装している．先行研究の手法では，クライアントは平文の暗号化に AES を使い，FHE を AES の鍵の暗号化のみに使用することで平文の暗号化を高速化し，サーバ側で AES の暗号文を FHE の暗号文に変換する．また，著者ら [3] [4] は AES に加えてストリーム暗号の TRIVIUM [5] や軽量ブロック暗号の KATAN [6] を LHE に組み合わせ，クライアントへの負荷・通信量・サーバへの負荷の比較をスマートフォンをクライアントとして行った．このような手法では，クライアント側では依然として負荷の大きい FHE または LHE を使用しなくてはならないため，さらなる IoT デバイスへの負荷削減を検討する必要がある．

本研究では，LHE を用いた統計分析を想定して，IoT デバイスでの暗号化の高速化・通信量削減が可能な手法を提案する．提案手法では，FHE や LHE より軽量な Somewhat 準同型暗号 (以下 SHE: Somewhat Homomorphic Encryption) を使った平文の暗号化をクライアント側で行い，代わりにサーバ側で SHE の暗号文から LHE の暗号文へ変換する．

表 1: 準同型暗号の分類

	加算	乗算	処理	暗号文	演算可能回数
SHE	可能	可能	高速	小さい	少ない
LHE	可能	可能	Level に依存	Level に依存	Level に依存
FHE	可能	可能	低速	大きい	多い

2 完全準同型暗号 (FHE)

2.1 特　　徴

FHE とは式 (1)，(2) のような暗号文同士の加法・乗法の演算が任意回数可能な性質をもつ公開鍵暗号の一つである．

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

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

FHE の概念自体は，1970 年代後半に公開鍵暗号が考案された当初より Rivest ら [7] によって提唱されていたが，2009 年に Gentry [8] が実現する手法を提案した．この実装は多項式環やイデアル格子を応用した暗号方式で，読解困難性を保つために，暗号文は平文を暗号化したものにランダムなノイズを加えた形式で表現される．加算や乗算を繰り返すたびにノイズが増加し，特に乗算では大きく増加する．そして，ノイズがある閾値を超えると正しく復号できなくなる．Gentry はこのような演算回数の上限を取り払うために bootstrapping と呼ばれる蓄積されたノイズを削減する手法を提案した．しかしながら，bootstrapping の処理は時間がかかるため実用的とは言い難い．

FHE の方式は近似 GCD 問題 [9]，LWE(Learning with Errors) 問題 [10]，RLWE(Ring Learning with Errors) 問題 [11]，NTRU [12] などを暗号ベースとしたものが提案されている．

2.2 準同型暗号の分類

暗号文同士の加法・乗法の演算がどちらかのみ可能であれば，それぞれを加法準同型暗号，乗法準同型暗号と呼ぶ．加法・乗法の演算がどちらも成立する暗号は暗号文同士の演算可能な回数によって，SHE，LHE，FHE の 3 種類に分類される [13] [14] [15]．

SHE は演算可能な回数が制限される代わりに，LHE や FHE よりも高速に動作し，暗号文サイズも小さい．本研究では Gentry らによって提案された，円分整数が作り出す RLWE 問題をベースとする SHE [11] [2] [16] を実装し実験に用いた．

LHE は演算可能な回数を決めるパラメータの Level を任意の値にあらかじめ設定する手法である．任意の回数の演算が可能なため LHE を含めて FHE と呼ぶ場合もある．Level を大きく設定すると暗号文同士で演算可能な回数が増加するが，計算時間と暗号文サイズも増加する．つまり，サーバ側の演算が複雑になるほど Level を高く設定しなくてはならなくなり，クライアントにおける暗号化時間・暗号文サイズが大きくなるという関係にある．本研究では，LHE で統計分析する場面を想定して実験に用いた．

FHE は，演算可能な回数に上限のない方式であるが，処理が重く暗号文サイズが大きいという欠点がある．既存手法では，SHE または LHE に対して bootstrapping を用いることが唯一の FHE を構成する方法である．

表 1 には SHE，LHE，FHE の違いをまとめた．

2.3 ライブラリ

FHE または LHE を実装しているライブラリには HElib [17]，SEAL [18]，PALISADE [19] などが挙げられる．HElib は IBM の研究者らによって初期に公開された広く知られているライブラリの一つで，bootstrapping をサポートしており，C++で実装されている．SEAL は Microsoft Research によって開発された C++で実装されたライブラリで，現時点では bootstrapping はサポートされていない．PALISADE はニュー・ジャージー工科大学によって開発されたライブラリで C++で実装されていて，現在 bootstrapping をサポートしている．

本研究では 2019 年 5 月 21 日にコミットされたバージョンの HElib を実装に用いた．

3 先 行 研 究

Lauter ら [1] [15] は FHE の暗号文サイズの削減を目的として図 2 に示すような共通鍵暗号の AES と FHE を組み合わせることを提案した．このシステムで行なっている AES の暗号文から FHE の暗号文への変換とは，FHE で暗号化したままで AES の復号処理を行うことで，AES で暗号化された平文を FHE で暗号化された平文に変えるということである．

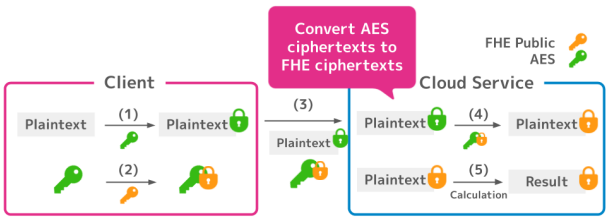

図 2: Lauter らの提案したシステム

(1) クライアントは AES で平文を暗号化．

(2) クライアントは FHE の公開鍵で AES の共通鍵を暗号化．

(3) AES の暗号文と FHE で暗号化した AES の共通鍵をクラウドサービスに送信．

(4) クラウドサービスは AES の暗号文を FHE の暗号文に変換.
(5) クラウドサービスは FHE の暗号文を使って何らかの演算を行う.

表 2: 先行研究マシン性能

Device	OS	CPU	Number of Cores	Processor Speed	RAM
Lenovo X230	Ubuntu 14.04	Intel Core i5-3320M	2	2.6 GHz	4GB

Gentry ら [2] [15] は Lauter らが提案した手法の (4) の処理にあたる，AES の暗号文を FHE の暗号文へ変換する処理の実装を行った．AES を使用した理由としては，広く使われている暗号であり，並列処理や最適化しやすい構造であるためとしている．FHE のライブラリである HElib を用いて，2880B の平文を AES で暗号化し，FHE で暗号化された AES の鍵を使い復号する設計になっている．表 2 に示すようなマシンを実験に使用して 2880B の AES 暗号文から FHE 暗号文への変換に 18 分，16B あたり 6 秒を要した．

AES 以外では， National Security Agency(NSA) によってリリースされた SIMON [20] [21] や低レイテンシで小面積実装が可能な PRINCE [22] [23] といった軽量ブロック暗号を使った実装もされている．しかしながら，そのような軽量ブロック暗号では実装を簡略化する代わりにラウンド数を多くして安全性とのバランスを取っているため，FHE または LHE の暗号文へ変換する際の演算回数が多くなりその結果，変換処理に多くの時間がかかる．Canteaut ら [24] は共通鍵暗号の暗号文を FHE の暗号文へ変換する処理の軽量化を目指し，欧州におけるストリーム暗号選定プロジェクト eSTREAM [25] で高い評価を得た TRIVIUM [5] を FHE と組み合わせることを提案し，実装した．また，TRIVIUM には 80bit セキュリティのものしかないため，Canteaut らは独自に TRIVIUM を 128bit セキュリティに発展させた Kreyvium を提案し，TRIVIUM と Kreyvium で FHE の暗号文への変換処理を比較した．

著者ら [3] [4] は Gentry や Canteaut らの実装を発展させ，AES・TRIVIUM に加えて，軽量ブロック暗号の KATAN を LHE に組み合わせ，スマートフォンをクライアントとして実装し，クライアントへの負荷・通信量・サーバへの負荷を比較した．

このように，どのような共通鍵暗号と FHE または LHE を組み合わせれば効率的なのかという議論が今までにされてきた．しかし，図 2 の Lauter らが提案した手法では，クライアントは依然として負荷の大きい FHE または LHE を使わなくてはならない．

4 システムデザイン

4.1 LHE only (既存手法)

4.2 節で提案する手法との比較のために，LHE のみを暗号化に使用するデザインの概要を図 3 に示し，LHE only とする．

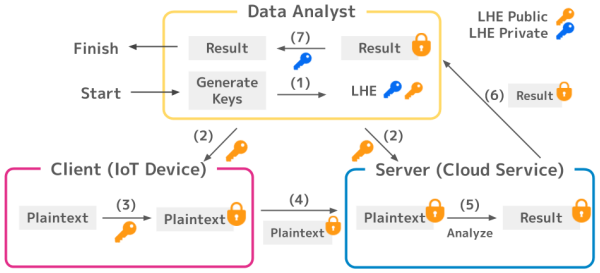

図 3: LHE only

(1) データ分析者は LHE の公開鍵と秘密鍵を生成.
(2) データ分析者はクライアント (IoT デバイスユーザ) とサーバ (クラウドサービス) に公開鍵を送信.
(3) クライアントは LHE で平文を暗号化.
(4) クライアントは LHE の暗号文をサーバに送信.
(5) サーバは LHE の暗号文を使って分析.
(6) サーバは分析結果をデータ分析者に送信.
(7) データ分析者は分析結果を復号.

4.1.1 既存手法の特徴

既存手法の LHE only はシンプルで実装が容易である．一方で，クライアントにおける LHE を使った暗号化に時間がかかること，暗号文サイズが大きくなることが問題点として挙げられる．また，2.2 節にあったように，サーバ側で複雑な演算をする場合，つまりパラメータの Level を高く設定するほどクライアントへの負荷が大きくなり通信量も多くなるという欠点もまたある．

4.2 SHE+LHE (提案手法)

LHE only ではシンプルに，クライアントが LHE で平文を暗号化するとクライアントに負荷がかかってしまっていた．そこで，クライアントには LHE よりも軽量な暗号を使わせ，サーバ側で LHE の暗号文に変換すればよいという発想から設計されたのが図 4 に示すような提案手法の SHE+LHE である．

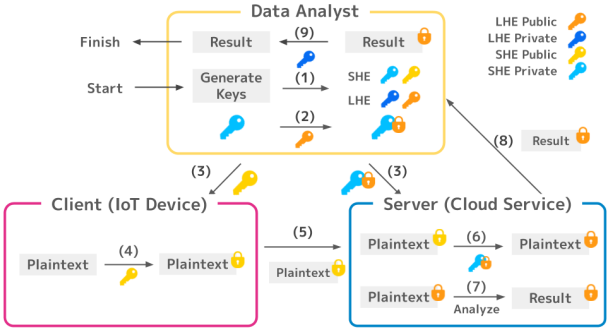

図 4: SHE+LHE

(1) データ分析者は LHE と SHE の公開鍵と秘密鍵を生成.
(2) データ分析者は SHE の秘密鍵を LHE の公開鍵で暗号化.

表 3: マシン性能

Device	OS	CPU	Number of Cores	Processor Speed	RAM
Google Pixel 3	Android 9.0	ARM Cortex-A75 ARM Cortex-A55	8	2.5 GHz 1.6 GHz	4 GB
MacBook Pro	OS X 10.15	Intel Core i5	4	1.4 GHz	8 GB

(3) データ分析者は SHE の公開鍵をクライアントに，LHE で暗号化された SHE の秘密鍵をサーバに送信．
(4) クライアントは SHE で平文を暗号化．
(5) クライアントはサーバに SHE の暗号文を送信．
(6) サーバは SHE の暗号文を LHE の暗号文へ変換．
(7) サーバは LHE の暗号文を使って分析．
(8) サーバは分析結果をデータ分析者に送信．
(9) データ分析者は分析結果を復号．

図 4(6) の「SHE の暗号文を LHE の暗号文へ変換」とは，LHE で暗号化した SHE の秘密鍵を使って SHE の暗号文を復号するということである．

4.2.1 提案手法の特徴

提案手法の SHE+LHE を利用する利点はクライアントが LHE を使わなくてよいことである．それによって，サーバ側での演算が複雑になりパラメータの Level を高く設定してもクライアントへは影響しない．一方でこのデザインは LHE only に比べて実装が複雑であり，サーバへの負荷は大きいという欠点がある．しかしながらクライアントの性能よりもサーバの性能が一般に高いことを活かすことができる．

5 実　　験

5.1 実験環境

実験に使用したマシンの性能を表 3 に示す．本研究ではクライアントとしてスマートフォンの Google Pixel 3，サーバとしてラップトップの MacBook Pro を実験に用いた．

5.2 予備実験

予備実験として，表 4 にはデバイスの性能の違いを比較した結果を示す．この予備実験では，Google Pixel 3 と MacBook Pro で SHE を用いて 16B の平文を暗号化した場合の実行時間を計測した．表 4 より，Google Pixel 3 では MacBook Pro の約 4 倍の時間が暗号化にかかることがわかる．

表 4: デバイスごとの SHE による暗号化時間

	SHE encryption time
Google Pixel 3	0.004 [s]
MacBook Pro	0.001 [s]

5.3 実験概要

クライアントは Google Pixel 3，サーバは MacBook Pro を使用し，平文サイズを 16B，32B，48B，64B，80B と変化させ，既存手法の LHE only，提案手法の SHE+LHE で以下の 3 つの比較を行った．

(1) クライアントへの負荷
(クライアントにおける実行時間)
(2) 通信量
(クライアント・サーバ間，データ分析者・クライアント間)
(3) サーバへの負荷
(サーバで LHE の暗号文へ変換する場合の実行時間)

「通信量」はクライアントに関わる部分のみを比較の対象とし，クライアント・サーバ間とデータ分析者・クライアント間の通信量を指す．クライアント・サーバ間の通信量とはクライアントがサーバに送信する暗号文のファイルサイズで，データ分析者・クライアント間の通信量とはデータ分析者がクライアントに共有する公開鍵のファイルサイズである．

5.4 パラメータ

実験に使用したパラメータを表 5，6 に示す．ここでパラメータ m は円分多項式 (3) の

$$\Phi_m(x) = \prod_{\substack{0 \le k < m \\ gcd(k,m)=1}} \left(x - e^{2\pi i k/m} \right) \tag{3}$$

m を意味する．この値を大きく設定するとセキュリティレベルが向上するが処理が重くなる関係にある．Level を大きく設定する場合は，セキュリティレベルを保つために m も大きく設定しなければならない．

LHE only ではサーバが LHE の暗号文を受け取った直後，SHE+LHE ではサーバが LHE の暗号文に変換した直後に加算・乗算が 3 回ずつ可能なパラメータを選択した．

また，セキュリティレベルについては，LHE には HElib を使用しているため HElib が提供しているセキュリティレベルを算出する関数で取得し，SHE のセキュリティレベルは LWE Estimator [26] を用いて算出した．

表 5: LHE only のパラメータ

LHE			SHE		
m	Level	Security	m	Level	Security
3307	59	80	N/A	N/A	N/A

表 6: SHE+LHE のパラメータ

LHE			SHE		
m	Level	Security	m	Level	Security
4409	60	80	256	1	80

5.5 実験結果

5.5.1 クライアントへの負荷

図 5 にはクライアントの実行時間を比較したグラフを示す．LHE only には LHE による平文の暗号化時間が含まれており，SHE+LHE には SHE による平文の暗号化時間が含まれている．SHE+LHE では LHE only に比べて非常に高速に暗号化できることがわかる．LHE only では 80 B の平文の暗号化に約 35 秒を要しており，実用的とは言い難い．

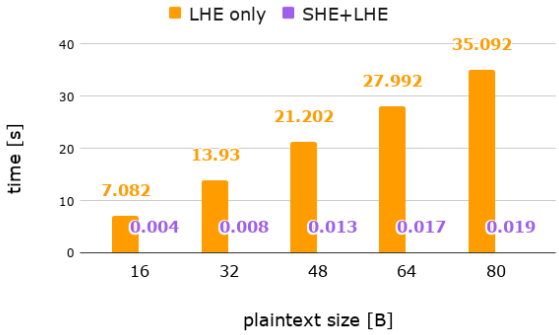

図 5: 実験結果：クライアントへの負荷

5.5.2 通信量

図 6 にはクライアント・サーバ間の通信量，つまりクライアントがサーバに送信する暗号文サイズを示す．LHE only では LHE の暗号文が含まれており，SHE+LHE には SHE の暗号文が含まれている．LHE only では 80B の平文を暗号化した場合に暗号文サイズが約 76MB にもなっており，SHE+LHE に比べて遥かに大きい．

図 7 にはデータ分析者・クライアント間の通信量，すなわちデータ分析者がクライアントに共有する公開鍵のサイズを示す．LHE only では LHE の公開鍵が含まれており，SHE+LHE には SHE の公開鍵が含まれている．LHE only では共有される鍵サイズが SHE+LHE の約 385 倍にもなる．

図 6，7 より，SHE の暗号文サイズと鍵サイズは LHE に比べてはるかに小さいため，提案手法の SHE+LHE は既存手法の LHE only よりも通信量の点で実用的であることがわかる．

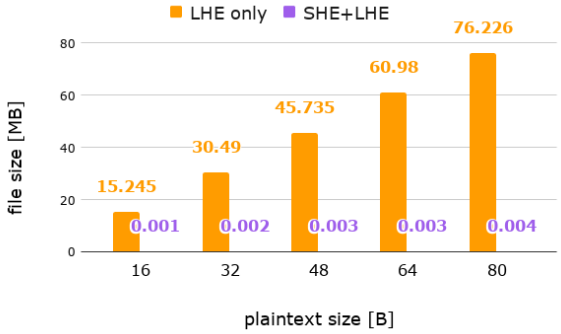

図 6: 実験結果：通信量 (クライアント・サーバ間)

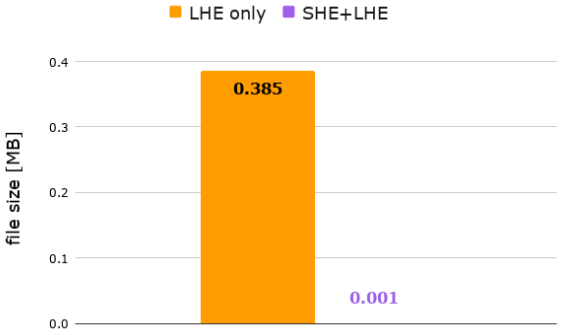

図 7: 実験結果：通信量 (データ分析者・クライアント間)

5.5.3 サーバへの負荷

図 8 にはサーバで LHE の暗号文へ変換する場合の実行時間を示す．この処理は LHE only では行わないため，0 秒である．SHE+LHE には SHE の暗号文から FHE の暗号文への変換処理時間を含んでいる．提案手法の SHE+LHE ではクライアントへの負荷を減らす代わりにサーバへの負荷が LHE only よりも大きい．

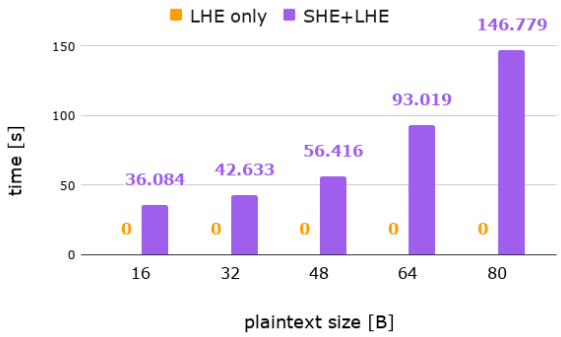

図 8: 実験結果：サーバへの負荷

5.6 実験結果まとめ

ここまでの実験結果のまとめを表 7 に示す．既存手法と提案手法でクライアントへの負荷・通信量・サーバへの負荷に加えて，実装の容易さも比較した．SHE+LHE では LHE only よりも生成する鍵が多く，FHE への変換処理も発生するため実装が難しいとした．

表 7: 実験結果のまとめ

	クライアントへの負荷	通信量	サーバへの負荷	実装の容易さ
LHE only (既存手法)	×	×	○	○
SHE+LHE (提案手法)	○	○	×	×

6 結　　論

表 7 より，**SHE+LHE が最適なシステムデザイン**である．

その理由は，IoT デバイスにとっては SHE+LHE が適したデザインであるためである．一般に IoT デバイス (クライアント) はクラウドサービス (サーバ) に比べて性能が低いため，IoT デバイスへの負荷を減らすことを優先する必要がある．また，IoT デバイスとクラウドサービス間の通信量が少ない方が IoT デバイスへの負荷を減らすことに繋がるため，通信量が少ないデザインを選択するべきである．

7 まとめと今後の課題

IoT デバイスで取得したセンサデータを安全に統計分析などに有効活用する際には，暗号文同士の演算が可能な準同型暗号が有用である．加算・乗算の両方可能な公開鍵暗号である準同型暗号は演算可能な回数によって SHE，LHE，FHE の 3 つに分けられる．SHE は数回程度なら演算が可能で比較的高速であり，LHE はあらかじめ設定した回数の範囲内で演算が可能であるが演算可能な回数を増やす設定をするほど遅くなる．FHE には演算回数の制限はないが，何度でも演算可能にするための bootstrapping という処理が重いという特徴がある．

本研究では LHE で統計分析する場面を想定した．計算能力が高いとは言い難い IoT デバイスにとっては，LHE で平文を暗号化するとなると大きな負荷がかかる．そのため，IoT デバイスへの負荷を減らす手法を提案した．提案手法では，IoT デバイス側では LHE よりも軽量な SHE によって平文を暗号化し，代わりにクラウドサービス側で SHE の暗号文から LHE の暗号文へ変換する．IoT デバイスへの負荷を減らす代わりにクラウドサービスへ負荷をかけるようになっており，一般にクライアント側 (IoT デバイス) よりもサーバ側 (クラウドサービス) の方が性能が良いという特徴を生かすことができる．

実験の結果，IoT デバイスで平文を LHE で暗号化する既存手法に比べて，提案手法では IoT デバイスへの負荷と通信量の観点から格段に優れているという結果が示された．このような実験結果から，提案手法は既存手法よりも優れたシステムデザインであると考えられる．

今後はサーバ側での具体的なアプリケーションを想定した実験を検討している．

謝　　辞

本研究は一部，JST CREST JPMJCR1503 の支援を受けたものである．

文　　献

[1] Kristin Lauter, Michael Naehrig, and Vinod Vaikuntanathan. Can homomorphic encryption be practical? In *CCSW '11: Proceedings of the 3rd ACM workshop on Cloud computing security workshop*, pp. 113–124, 2011.

[2] Craig Gentry, Shai Halevi, and Nigel P. Smart. Homomorphic evaluation of the aes circuit. In *Advances in Cryptology – CRYPTO 2012*, pp. 850–867, 2012.

[3] 松本茉倫, 小口正人. 完全準同型暗号と共通鍵暗号を組み合わせた IoT デバイスにおけるセンサデータ暗号化の高速化. マルチメディア、分散、協調とモバイル (DICOMO2020) シンポジウム論文集, pp. 712 – 719, 2020.

[4] Marin Matsumoto and Masato Oguchi. Speeding up sensor data encryption with a common key cryptosystem combined with fully homomorphic encryption on smartphones. In *Proceedings of the World Conference on Smart Trends in Systems, Security and Sustainability (WS4 2020)*, pp. 500 – 505, 2020.

[5] Christophe De Canni‘ere and Bart Preneel. Trivium specifications, estream, ecrypt stream cipher project, 2006.

[6] Christophe De Cannière, Orr Dunkelman, and Miroslav Knežević. Katan and ktantan — a family of small and efficient hardware-oriented block ciphers. In *Cryptographic Hardware and Embedded Systems - CHES 2009*, pp. 272–288, 2009.

[7] Ronald L. Rivest, Len Adleman, and Michael L. Dertouzos. On data banks and privacy homomorphisms. In *Foundations of Secure Computation*, pp. 169–180, 1978.

[8] Craig Gentry. *A FULLY HOMOMORPHIC ENCRYPTION SCHEME*. PhD thesis, Stanford University, 2009.

[9] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. In *Advances in Cryptology - EUROCRYPT 2010*, pp. 24–43, 2010.

[10] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *Advances in Cryptology – CRYPTO 2013*, pp. 75–92, 2013.

[11] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *ITCS '12: Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, pp. 309–325, 2012.

[12] Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In *STOC '12: Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, pp. 1219–1234, 2012.

[13] Frederik Armknecht, C. Boyd, Christopher Carr, Kris-

tian Gjøsteen, Angela Jäschke, Christian A. Reuter, and M. Strand. A guide to fully homomorphic encryption. *IACR Cryptol. ePrint Arch.*, Vol. 2015, p. 1192, 2015.

[14] Abbas Acar, Hidayet Aksu, A Selcuk Uluagac, and Mauro Conti. A survey on homomorphic encryption schemes: Theory and implementation. *ACM Computing Surveys (CSUR)*, Vol. 51, No. 4, pp. 1–35, 2018.

[15] 佐藤宏樹, 馬屋原昂, 石巻優, 今林広樹, 山名早人. 完全準同型暗号のデータマイニングへの利用に関する研究動向. 情報科学技術フォーラム講演論文集 (FIT), pp. 165–172, 2016.

[16] 有田正剛. イデアル格子暗号入門. 情報セキュリティ総合科学, Vol. 6, , nov 2014.

[17] HElib. `https://github.com/homenc/HElib` (2020 年 11 月 22 日閲覧).

[18] Microsoft SEAL. `https://github.com/Microsoft/SEAL` (2020 年 11 月 22 日閲覧).

[19] PALISADE. `https://palisade-crypto.org/software-library/` (2020 年 11 月 22 日閲覧).

[20] R. Beaulieu, S. Treatman-Clark, D. Shors, B. Weeks, J. Smith, and L. Wingers. The simon and speck lightweight block ciphers. In *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pp. 1–6, 2015.

[21] Tancrede Lepoint and Michael Naehrig. A comparison of the homomorphic encryption schemes fv and yashe. In *International Conference on Cryptology in Africa*, pp. 318–335, 2014.

[22] Julia Borghoff, Anne Canteaut, Tim Güneysu, Elif Bilge Kavun, Miroslav Knezevic, Lars R. Knudsen, Gregor Leander, Ventzislav Nikov, Christof Paar, Christian Rechberger, Peter Rombouts, Søren S. Thomsen, and Tolga Yalçın. Prince – a low-latency block cipher for pervasive computing applications. In *Advances in Cryptology – ASIACRYPT 2012*, pp. 208–225, 2012.

[23] Yarkın Doröz, Aria Shahverdi, Thomas Eisenbarth, and Berk Sunar. Toward practical homomorphic evaluation of block ciphers using prince. In *International Conference on Financial Cryptography and Data Security*, pp. 208–220, 2014.

[24] Anne Canteaut, Sergiu Carpov, Caroline Fontaine, Tancrède Lepoint, María Naya-Plasencia, Pascal Paillier, and Renaud Sirdey. Stream ciphers: A practical solution for efficient homomorphic-ciphertext compression. *Journal of Cryptology*, Vol. 31, No. 3, pp. 885–916, 2018.

[25] ECRYPT - European Network of Excellence in Cryptology. The estream streamcipher project. `https://www.ecrypt.eu.org/stream/project.html`.

[26] LWE Estimator. `https://lwe-estimator.readthedocs.io/en/latest/` (2020 年 12 月 25 日閲覧).