DEIM Forum 2022 J41-2

IoT デバイス・クラウドサーバ連携時の準同型暗号化手法の比較

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　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，安全とは言えないクラウドサーバ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・乗算が任意回数可能な Leveled 準同型暗号 (以下 LHE: Leveled Homomorphic Encryption) が注目されている．LHE によって，IoT デバイスで個人情報を暗号化しクラウドサーバで暗号化したまま分析，データ分析者が分析結果を復号することが可能になる．しかし，LHE による暗号化は時間がかかり，暗号文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．本稿では，IoT デバイスで取得したデータのクラウドサーバを使ったデータ活用を想定したシステムデザインを既存手法と提案手法で 3 つ実装した．IoT デバイスとして Google Pixel 3 と Raspberry Pi を実装に用いて，IoT デバイスへの負荷・クラウドサーバへの負荷・通信量の 3 つの指標で比較した．その結果，単純に LHE で暗号化する場合に比べ，提案手法はスマートフォンにおいて最大で約 2670 倍高速に暗号化可能であることを示した．

キーワード　準同型暗号，IoT デバイス

1　はじめに

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の中には秘匿性が高いデータが存在しており，このデータを安全とは言えないクラウドサーバへ送信して蓄積・解析を行うためには，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・乗算が任意回数可能な Leveled 準同型暗号 (以下 LHE: Leveled Homomorphic Encryption) や演算回数に制限のない完全準同型暗号 (以下 FHE: Fully Homomorphic Encryption) が注目されている．LHE や FHE によって，図 1 のように IoT デバイスで個人情報を暗号化しクラウドサーバで暗号化したまま分析，データ分析者が分析結果を復号することが可能になる．本稿では，LHE を使ったシステムを想定した．

しかしながら，LHE・FHE による暗号化は時間がかかり，また暗号文サイズが大きく通信量が多くなってしまうため，計算能力や通信能力の低い IoT デバイスへの実装は実用面で現実的でないと考えられる．ただし，スマートフォンのように IoT デバイスの中でも比較的計算能力が高く，今後の能力向上が期待されているデバイスでは，クラウドサーバ側とうまく連携すれば高速な暗号化処理も現実的になると考えられる．

Lauter ら [1] は，クライアントの負荷削減を目的として，クライアント側において共通鍵暗号 AES で平文を暗号化し，AES 鍵を FHE で暗号化する手法を提案した．この手法ではクラウドサーバ側で AES の復号処理を暗号化した状態で行うことで AES の暗号文を FHE の暗号文へ変換し，FHE の演算を行うことが可能となる．本稿では，IoT デバイス側で LHE・FHE よりも軽量で暗号文同士の加算が可能な加法準同型暗号 (以下 AHE: Additive Homomorphic Encryption) のみを使って暗号化し，クラウドサーバ側で LHE の暗号文へ変換する手法を提案する．Baseline としてシンプルに LHE で暗号化した手法，既存手法である Lauter らの提案を基にした手法，提案手法の 3 つのシステムデザインを実装し比較した．

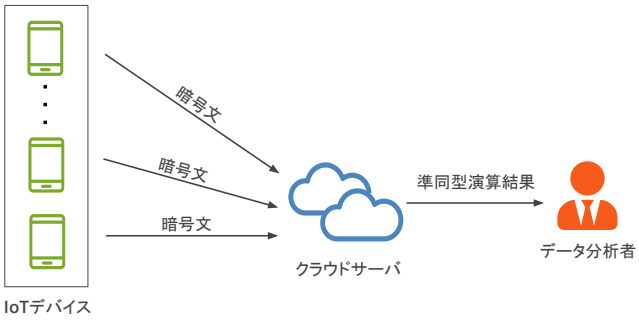


図 1: 本稿で想定する準同型暗号を用いたビッグデータ活用システム

本稿の貢献を以下に示す．

- LHE でデータを暗号化した際の IoT デバイスへの負荷を把握し，提案手法では IoT デバイス側が実用的な処理時間となる事を示すため，高性能な IoT デバイスであるスマートフォンと低性能な IoT デバイスである Raspberry Pi へ実装して実験を行った．
- シンプルに LHE で暗号化する手法・AES と LHE を組み合わせた既存手法に加えて，AHE と LHE を組み合わせた手法を新たに提案し実装した．
- 3 つの手法を IoT デバイスへの負荷・クラウドサーバへの負荷・IoT デバイスの送受信する通信量という指標で比較し，それぞれの手法の利点・欠点を示した．

2 準　　備

本節では，加算・乗算が可能な準同型暗号と RLWE 問題ベースの加法準同型暗号について述べる．

2.1 準同型暗号

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 [2] [3].

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．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 [4] によって提唱されていたが，2009 年に Gentry [5] 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．

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．

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．

2.2 RLWE 問題ベースの加法準同型暗号

本稿で実験に用いた RLWE 問題ベースの AHE [1] [6] の詳細について説明する．記号の一覧は表 1 にまとめた．

n を 2 の巾乗，モジュラス q を奇数から選び，ノイズレベルとして σ を選択する．高々$n-1$ 次で係数が $(-p/2, p/2] \cap \mathbb{Z}$ の以下のような多項式 R_p を定義する．$\mathbb{F}_p[x]/(x^n+1)$ は $\mathbb{F}_p[x]$ を x^n+1 で割った余りである．R_p 同士の積で増えた項は x^n+1 で割って n 項を超えないようにする．

$$R_p := \mathbb{F}_p[x]/(x^n+1)$$

- **鍵生成**

各係数が 0 か 1 である s と各係数が $(-q/2, q/2] \cap \mathbb{Z}$ の a を一様ランダムに選ぶ.

$$s \leftarrow R_2$$

$$a \leftarrow R_q$$

表 1: 記号の意味

記号	意味
$a := b$	b で a を定義する．
$(a, b]$	a より大きく b 以下の実数からなる区間.
$[a]_n$	a を n で割った余り．余りは区間 $(-n/2, n/2]$ にとる．
$\mathbb{F}_p$	p 個の元 $(-p/2, p/2] \cap \mathbb{Z}$ からなる有限体.
$\mathbb{F}_p[x]$	x を変数とする $\mathbb{F}_p$ 係数の多項式全体.
$\mathbb{F}_p[x]/(x^n+1)$	$\mathbb{F}_p[x]$ を x^n+1 で割った余り．
sk	秘密鍵．sk_{LHE} であれば LHE の秘密鍵の意味.
pk	公開鍵．pk_{LHE} であれば LHE の公開鍵の意味.
$ptxt$	平文.
$LHE(x)$	x を LHE で暗号化した結果.
$AES(x)$	x を AES で暗号化した結果.
$AHE(x)$	x を AHE で暗号化した結果.

平均 0，分散 σ^2 のガウス分布 $N(0, \sigma^2)$ から実数をサンプルし，もっとも近い整数に丸め込んだもの係数にもつノイズ e を生成する．

$$e \leftarrow N(0, \sigma^2)$$

a を s 倍したものに 2 倍したノイズ $2e$ を加えて b とする．

$$b := [as + 2e]_q$$

s が秘密鍵 sk，(a, b) が公開鍵 pk となる．

- **暗号化**

平文 $ptxt \in R_2$ に対して，乱数 v と 2 つのノイズ e_0, e_1 を生成する．

$$v \leftarrow R_2$$

$$e_0, e_1 \leftarrow N(0, \sigma^2)$$

pk に含まれる b を v 倍したものにノイズ e_0 の 2 倍と $ptxt$ を加え，暗号文の第 1 成分とする．

$$c_0 := [bv + 2e_0 + ptxt]_q.$$

pk に含まれる a を v 倍したものにノイズ e_1 の 2 倍を加え暗号文の第 2 成分とする．

$$c_1 := [av + 2e_1]_q$$

(c_0, c_1) が暗号文 ct となる．

- **暗号文同士の加算**

2 つの暗号文を $ct = (c_0, c_1), ct' = (c'_0, c'_1)$ とする． ct, ct' の各係数の加算を行い，係数ごとに q で割った余りをとる．

$$d := ([c_0 + c'_0]_q, [c_1 + c'_1]_q)$$

暗号文 d が暗号文同士の加算結果 $ct + ct'$ となる．

- **復号**

c_0 から第 2 成分 c_1 の s 倍を引き，q で割った余りをとり，2 で割った余りをとると平文 $ptxt$ となる．

$$ptxt = [[c_0 - sc_1]_q]_2$$

正しく復号できるかを確認する．

$$\begin{aligned} c_0 - sc_1 &\equiv bv + 2e_0 + ptxt - s(av + 2e_1) \\ &\equiv (b - as)v + 2e_0 - 2se_1 + ptxt \\ &\equiv 2ev + 2e_0 - 2se_1 + ptxt \\ &\equiv ptxt + 2(ev + e_0 - se_1) (\mathrm{mod} q) \end{aligned}$$

ここで q と比べて e, e_0, e_1, v, s は小さいので，$2(ev + e_0 - se_1)$ も q にくらべて小さい．$ptxt$ もその各係数は 0 または 1 である．よって，$ptxt + 2(ev + e_0 - se_1)$ も q に比べて小さく，その各係数は $(-q/2, q/2]$ の範囲に収まっているとしてよい．したがって，

$$[c_0 - sc_1]_q = ptxt + 2(ev + e_0 - se_1)$$

となり，2 で割った余りとして $ptxt$ を取り出せる．

3 関 連 研 究

本節では 2 種類の関連する研究として共通鍵暗号を組み合わせた LHE・FHE の暗号化処理高速化，IoT デバイスとサーバ連携時の準同型暗号処理を扱った研究について説明する．

3.1 共通鍵暗号による LHE・FHE の暗号化処理高速化

Lauter ら [1] 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．この手法では，クライアントでは平文を AES で暗号化し AES の鍵のみを FHE で暗号化する．クラウドサーバでは AES の復号処理を FHE で暗号化したままで行うことで FHE の暗号文が得られる．本稿では，FHE または LHE で暗号化したままで行う復号処理を FHE または LHE への変換と呼ぶ．Gentry ら [7] は実際に AES から FHE・LHE への変換処理を実装した．

AES 以外では， National Security Agency(NSA) によってリリースされた SIMON [8] [9] や低レイテンシで小面積実装が可能な PRINCE [10] [11] 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る結果，変換処理に多くの時間がかかる．そこで，Albrecht ら [12] は，乗算の数が少ないブロック暗号 LowMC を提案した．しかしながら，LowMC の脆弱性がその後指摘された [13] [14]．Canteaut ら [14] 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[1]で高い評価を得た Trivium [15] 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．

1：https://www.ecrypt.eu.org/stream/project.html

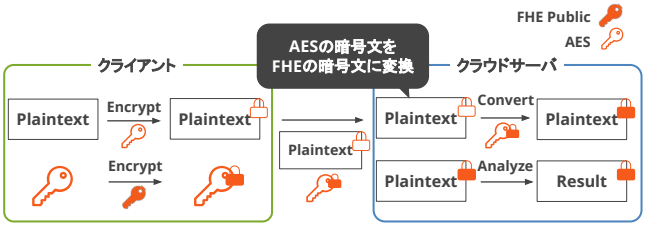


図 2: Lauter ら [1] の提案した手法．平文を AES で暗号化し，AES の鍵だけを FHE で暗号化することで暗号化の高速化が可能．FHE で暗号化するデータが最大でも AES の鍵長になる．

著者ら [16] 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 [17] 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，暗号化時間・暗号文サイズ・変換処理の実行時間を比較した．

このように，どの共通鍵暗号と LHE または FHE を組み合わせれば効率的かを比較する研究がなされている．

3.2 IoT デバイスとサーバ連携時の準同型暗号処理

本項では，IoT デバイスへの負荷が大きい準同型暗号のアルゴリズムの改良やハードウェア最適化を行った Ma ら [18]，Natarajan ら [19] と Hagen ら [20] の研究について説明する．

データを集約せずに分散した状態で機械学習を行う Federated Learning [21] はクライアントが送信した勾配から情報漏洩が引き起こされることが指摘されている [22] [23]．Ma ら [18] は準同型暗号を組み合わせることでプライバシ保護した Federated Learning を 10 台の Jetson Nano(4 コア 1.43GHz の ARM Cortex-A57，4GB の RAM，128NVIDIA CUDA® コアの GPU) をクライアントとして実装した．このとき，サーバに集められたパラメータは平均化することでアップデートされ [24]，暗号文同士の加算のみが必要となる．実験では 3 種類の加法準同型性を持つ暗号 (Paillier 暗号 [25], MK-CKKS [26], xMK-CKKS) で Jetson Nano の消費電力，暗号文サイズ，暗号化の実行時間を比較した．xMK-CKKS は Ma らが提案した手法で，近似固定小数点演算をサポートした LHE スキームである CKKS [27] を複数の異なる鍵での暗号化を可能にした MK-CKKS [26] のセキュリティ強度を高めたものである．実験の結果，暗号文サイズと暗号化時間は MK-CKKS と xMK-CKKS が優れており，消費電力には大きな差がないことが示された．

Natarajan ら [19] は Microsoft 社の CKKS [27] 方式の準同型暗号ライブラリ SEAL[2]を IoT デバイス向けに拡張した SEAL-Embedded[3]を提案した．実験に使用したデバイスは Azure Sphere MT3620(1 コア 494MHz の ARM Cortex-A7 CPU, 256 KB の RAM) と Nordic nRF52840(64 MHz の ARM Cortex-M4 CPU，256 KB の RAM) の 2 種類の組み込み機器である．Number Theoretic Transform (NTT) より効率的な negacyclic NTT を使った多項式乗算の高速化やメモリの再利用などにより高速でメモリ消費の少ない実装が可能になったこ

2：https://github.com/Microsoft/SEAL
3：https://github.com/microsoft/SEAL-Embedded

とが示された．

[20] では， 準同型暗号上の推論処理のサーバへの委託においてクライアントへの負荷を削減するためのシステムである CHOCO-TACO(Client-aided HE for Opaque Compute Offloading Through Accelerated Cryptographic Operations) を提案した．クライアントとして NXP IMX6 evaluation kit(528MHz の ARM Cortex-A7 CPU, 4GB の DDR3L SDRAM) を使用し， SHE は Microsoft 社のライブラリの SEAL で実装している．CHOCO-TACO では暗号化処理の並列化によるハードウェア最適化を施すことで，最適化前の 417 倍の高速化と 603 倍の消費電力削減が可能であることが示された．

このような関連研究で IoT デバイスとして使われているのは主にマイコンであり，Android OS のスマートフォンに LHE または FHE を実装した研究は現時点では著者らのものを除いて見つからなかった．関連研究では準同型暗号のアルゴリズムを改善したり並列化を取り入れることで IoT デバイス向けの暗号としている．これに対し本稿で提案した最も IoT デバイスへの負荷が低い 4.4 項のデザイン 3 では，IoT デバイスは軽量な暗号で平文を暗号化しクラウドサーバが代わりに高機能な LHE に変換する手法を提案している．

4 システムデザイン

本節では，Baseline のデザイン 1・既存手法のデザイン 2・提案手法のデザイン 3 のアルゴリズムとその特徴について説明する．表 2 にはそれぞれのデザインの処理内容をまとめた．

4.1 概　　要

システムデザインでは図 3 のように IoT デバイス，クラウドサーバ，復号サーバの 3 つのエンティティを想定する．また，それぞれのエンティティは共謀しないことを前提とする．それぞれのエンティティの役割を以下に示す．

- **IoT デバイス (IoT)**

IoT デバイスはデータ所有者として，復号サーバから受信した公開鍵を用いて自身の個人情報を暗号化しクラウドサーバに送信する．

- **クラウドサーバ (CS)**

クラウドサーバは高性能ではあるが外部からアクセスされる恐れのあるマシンを想定する．したがって，クラウドサーバには平文は保存されず，常に暗号化されたデータのみが保存される．なお，本稿では具体的なアプリケーションは実装せず，その構成要素となる各演算処理の評価を行っているが，機械学習の推論や統計分析などの演算を LHE で暗号化したままで実行することを想定する．演算結果は復号サーバへ送信される．

- **復号サーバ (DS)**

復号サーバは限られたデータ分析者のみがアクセス可能なマシンを想定し，鍵生成とクラウドサーバから受信した演算結果の復号を行う．なお，それぞれの IoT デバイスで暗号化したデータは復号せず演算結果だけを復号するものとする．

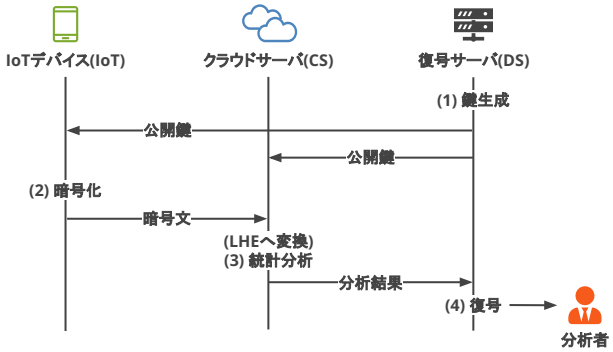


図 3: IoT デバイス (IoT)・クラウドサーバ (CS)・復号サーバ (DS) の役割

4.2 デザイン 1 (Baseline)

デザイン 1 では復号サーバは LHE の公開鍵を配布し，IoT デバイスは平文を LHE で暗号化する．シンプルで実装が容易であるが暗号化する平文が長くなるほど IoT デバイスで暗号化に時間がかかりクラウドサーバへ送信する暗号文サイズが大きくなる．

（1） 復号サーバによる鍵生成
LHE の秘密鍵 sk_{LHE} と公開鍵 pk_{LHE} を生成．
IoT と CS に LHE の公開鍵 pk_{LHE} を送信．
（2） IoT デバイスにおける暗号化
LHE で平文 (センサデータ等) を 1 ビットずつ暗号化．
LHE の暗号文 $LHE(ptxt)$ を CS に送信．
（3） クラウドサーバにおける分析
$LHE(ptxt)$ を使って統計分析等の演算を行う．
分析結果 $LHE(result)$ を DS に送信．
（4） 復号サーバによる分析結果の復号
分析結果 $LHE(result)$ を LHE の秘密鍵 sk_{LHE} で復号．

4.3 デザイン 2 (既存手法)

Lauter らの手法 [1] を基にしたデザイン 2 ではデザイン 1 と同じように復号サーバが LHE の公開鍵を配布するが，IoT デバイスが LHE で暗号化するのは AES 鍵のみで平文は AES で暗号化する点が異なる．したがって，デザイン 1 よりも IoT デバイスでの実行時間と IoT–クラウドサーバ間の通信量が抑えられる．AES を選択した理由は，現在広く使われている共通鍵暗号であり LHE への変換の際の並列処理に適しているためである．

AES 暗号文を LHE の暗号文へ変換する処理は Gentry ら [7] とほぼ同じ実装である．この実装では複数の暗号文を 1 つの暗号文として処理する暗号文パッキング [7] が用いられているため，複数ブロックをまとめて AES 暗号文から LHE 暗号文に変換可能になる．

（1） 復号サーバによる鍵生成
LHE の秘密鍵 sk_{LHE} と公開鍵 pk_{LHE} を生成．
IoT と CS に LHE の公開鍵 pk_{LHE} を送信．
（2） IoT デバイスにおける暗号化
AES 鍵 sk_{AES} を生成．

AES 鍵から 11 の $RoundKey$ を生成.

AES 鍵 sk_{AES} で平文を暗号化し $AES(ptxt)$ とする.

$RoundKey$ を LHE で暗号化し $LHE(RoundKey)$ とする.

AES の暗号文 $AES(ptxt)$ と暗号化した $RoundKey$ の $LHE(RoundKey)$ を CS に送信.

（3） クラウドサーバによる LHE 暗号文への変換と分析

$AES(ptxt)$ と $LHE(RoundKey)$ の準同型加算を行い $LHE(s)$ とする．以下のように $LHE(s)$ を AES の復号関数 ($AddRoundKey$, $InvShiftRows$,$InvSubBytes$, $InvMixColumns$) の入力とすることで最終的に LHE で暗号化された平文 $LHE(ptxt)$ へ変換される.

$LHE(s) \leftarrow$
$AddRoundKey(AES(ptxt), LHE(RoundKey)[10])$;
for $i = 9$ *to* 1 **do**
  $InvShiftRows(LHE(s))$;
  $InvSubBytes(LHE(s))$;
  $AddRoundKey(LHE(s), LHE(RoundKey)[i])$;
  $InvMixColumns(LHE(s))$;
end
$InvShiftRows(LHE(s))$;
$InvSubBytes(LHE(s))$;
$AddRoundKey(LHE(s), LHE(RoundKey)[0])$;
$LHE(ptxt) \leftarrow LHE(s)$;

$LHE(ptxt)$ を使って統計分析等の演算を行い，分析結果 $LHE(result)$ を DS に送信.

（4） 復号サーバによる分析結果の復号

分析結果 $LHE(result)$ を LHE の秘密鍵 sk_{LHE} で復号.

4.4 デザイン 3 (提案手法)

提案手法であるデザイン 3 では 2.2 項で説明した RLWE 問題ベースの AHE を LHE に組み合わせる.

デザイン 3 では既存手法のデザイン 1・2 とは 2 点が異なる．まず，復号サーバは AHE の公開鍵を IoT デバイスに，LHE で暗号化した AHE の秘密鍵をクラウドサーバに配布する．2 つ目に，IoT デバイスでは LHE を一切使わず AHE で平文を暗号化する．このような 2 点の特徴によって，IoT デバイスの実行時間が短縮され，AHE の公開鍵と暗号文は LHE よりも小さいため通信量も削減可能である．デザイン 3 ではデザイン 2 で行っていたような暗号文パッキングは現時点では実装していない.

RLWE 問題をベースとした AHE を選択した理由として復号処理が準同型演算で行えること，復号処理の乗算回数が少ないことが挙げられる．例えば，楕円曲線暗号は秘密鍵が暗号化された状態では復号処理が困難であり，RSA 暗号の復号処理は乗算回数が RLWE 問題をベースとした AHE よりも多くなる.

（1） 復号サーバによる鍵生成

LHE の秘密鍵 sk_{LHE} と公開鍵 pk_{LHE} を生成.

AHE の秘密鍵 sk_{AHE} と公開鍵 pk_{AHE} を生成.

LHE の公開鍵 pk_{LHE} で AHE の秘密鍵 sk_{AHE} を暗号化し $LHE(sk_{AHE})$ とする．ここで，AHE の秘密鍵は 1 ビットずつ暗号化する.

IoT に AHE の公開鍵 pk_{AHE}，CS に LHE で暗号化した AHE の秘密鍵 $LHE(sk_{AHE})$ を送信.

（2） IoT デバイスにおける暗号化

AHE の公開鍵 pk_{AHE} で平文を暗号化し $AHE(ptxt)$ とする.

AHE の暗号文 $AHE(ptxt)$ を CS に送信.

（3） クラウドサーバによる LHE 暗号文への変換と分析

AHE の暗号文 $AHE(ptxt)$ の第 1 成分を c_0，第 2 成分を c_1 とする.

LHE で暗号化した AHE の秘密鍵 $LHE(sk_{AHE})$ と c_1 の準同型乗算を行って c_0 から引き，AHE の暗号文 $AHE(ptxt)$ を LHE の暗号文 $LHE(ptxt)$ へ変換.

$$LHE(ptxt) = c_0 - LHE(sk_{AHE}) * c_1$$

$LHE(ptxt)$ を使って統計分析等の演算を行う.

分析結果 $LHE(result)$ を DS に送信.

（4） 復号サーバによる分析結果の復号

分析結果 $LHE(result)$ を LHE の秘密鍵 sk_{LHE} で復号.

5 実　　験

5.1 実験概要

高性能な IoT デバイスとしてスマートフォンの Google Pixel 3，低性能な IoT デバイスとして Raspberry Pi を実験に使用した．デザイン 1–3 で 256B の平文を LHE で暗号化する場合を想定する．実験プログラムの LHE は準同型暗号ライブラリの HElib [4] で実装し，AHE の実装は 2.2 項に示した．表 3 は実験に使用したマシンの性能である.

5.2 評価指標

IoT デバイスへの負荷 (IoT デバイスの実行時間，CPU 使用率，メモリ使用量)・クラウドサーバへの負荷 (クラウドサーバで LHE 暗号文へ変換する場合の実行時間，変換処理の CPU 使用率，メモリ使用量)・通信量 (IoT デバイス–クラウドサーバ間，復号サーバ–IoT デバイス間で IoT デバイスが送受信するファイルサイズ) という 3 つの指標で比較を行った．実行時間を示す値は 5 回実行した平均値を採用した．また，IoT デバイスの CPU 使用率とメモリ使用量については top コマンドによって 5 秒ごとに取得した値の平均を指す．CPU 使用率は該当するプロセスの CPU 項目をコア数で割った値で，メモリ使用量は実際に使用した物理メモリの値を示す RES 項目を抽出した．クラウドサーバの CPU 使用率とメモリ使用量については htop コマンドによって 30 秒ごとに取得した値の平均である．CPU 使用率は全てのコアの使用率を平均したもので，メモリ使用量は RES 項目を抽出した.

5.3 パラメータ

表 4 のパラメータについて，全てのデザインでクラウドサーバが LHE 暗号文への変換処理後，Level は 4 程度残るように選

4：https://github.com/homenc/HElib.git

表 2: 各システムデザインにおける IoT デバイス・クラウドサーバ・復号サーバの処理内容

デザイン 1 (Baseline)	IoT デバイス	LHE で平文を暗号化.
	クラウドサーバ	LHE 暗号文を分析.
	復号サーバ	LHE の公開鍵を IoT デバイスとクラウドサーバに配布.
デザイン 2 (既存手法)	IoT デバイス	AES で平文を暗号化し LHE で AES 鍵を暗号化.
	クラウドサーバ	AES 暗号文を LHE 暗号文へ変換して LHE 暗号文を分析.
	復号サーバ	LHE の公開鍵を IoT デバイスとクラウドサーバに配布.
デザイン 3 (提案手法)	IoT デバイス	AHE で平文を暗号化.
	クラウドサーバ	AHE 暗号文を LHE 暗号文へ変換して LHE 暗号文を分析.
	復号サーバ	LHE で暗号化した AHE の秘密鍵はクラウドサーバに AHE の公開鍵を IoT デバイスに配布.

表 3: 実験環境

Google Pixel 3 (IoT デバイス)	OS	Android 9.0
	CPU	ARM Cortex-A75(2.5 GHz) 4Cores ARM Cortex-A55(1.6 GHz) 4Cores
	Main Memory	4GB
Raspberry Pi 3 Model B+ (IoT デバイス)	OS	Raspbian GNU/Linux 10.0
	CPU	ARM Cortex-A53(1.4 GHz) 4Cores
	Main Memory	1GB
クラウドサーバ	OS	CentOS 6.10
	CPU	Intel®Xeon®Processor E5-2643 v3 (3.4GHz) 6 Cores × 2 Sockets
	Main Memory	512GB

表 4: パラメータ

デザイン 1	LHE	Level: 7, Security:130 bit
デザイン 2	LHE	Level: 45, Security: 131 bit
	AES	Key length: 128 bit
デザイン 3	LHE	Level: 8, Security: 130 bit
	AHE	Degree of a polynomial: 128, Security: 128 bit

択し，同程度のセキュリティレベルになるように設定した．ここで，Level が 4 程度残るとは，3 回程度の準同型乗算は可能であることを指す．デザイン 2(既存手法) では AES から LHE への変換処理で準同型乗算の回数が他 2 つのデザインよりも多いため Level は高く設定されている．LHE のセキュリティは，HElib で実装されている関数を用いて算出した値である．また，AHE のセキュリティは LWE Estimator[5]で算出した．

5.4 実験結果

図 4 は実験結果を 3 つの指標で比較したグラフを示している．図 4(a)・4(b)・4(c) の IoT デバイスへの負荷と図 4(d)・4(e)・4(f) のクラウドサーバへの負荷のトレードオフの関係が顕著に現れた．

5.4.1 IoT デバイスへの負荷

図 4(a) より，IoT デバイスが LHE を一切使わないデザイン 3(提案手法) では Google Pixel 3 の実行時間はデザイン 1(Baseline) の約 2670 分の 1，デザイン 2(既存手法) の約 590 分の 1 で格段に低い．IoT デバイスの CPU 使用率・メモリ使用量についてもデザイン 3(提案手法) が優れていることが図 4(b)・図 4(c) からも読み取れる．デザイン 2(既存手法) で図 4(c) のメモリ使用量がデザイン 1(Baseline) と 3(提案手法) に比べて大きい理由は図 4(h) に示されている，暗号化に使用する鍵が大きいためであると考えられる．その影響で，Raspberry Pi では既存手法がメモリ不足のため実行不可能であった．したがって，既存手法はリソースの限られた低性能な IoT デバイスへの実装は不向きであり，提案手法は低性能なデバイスであっても実現可能なシステムである．

5.4.2 クラウドサーバへの負荷

デザイン 1(Baseline) では変換処理が必要ないため，図 4(d) のクラウドサーバの実行時間は 0 秒である．デザイン 2(既存手法) に比べてデザイン 3(提案手法) の方が実行時間を要した．この理由としてはデザイン 3(提案手法) では複数の暗号文を 1 つの暗号文として扱う暗号文パッキングを実装していないことが挙げられ，今後の改善が必要である．図 4(e)・図 4(f) に示したクラウドサーバの CPU 使用率とメモリ使用量からも，デザイン 3(提案手法) の方がデザイン 2(既存手法) よりもクラウドサーバへ負荷が高いことが読み取れる．デザイン 2(既存手法) では暗号文パッキングを用いて一度に複数の暗号文の演算が行うことで，デザイン 3(提案手法) より高速な LHE 暗号文への変換処理が可能になっていることが分かる．

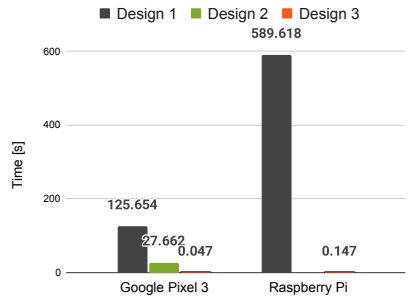
5.4.3 通　信　量

図 4(g) の IoT デバイスからクラウドサーバへ送信するファイルサイズについては，平文を LHE で暗号化するデザイン 1(Baseline) ではファイルサイズが最も大きい．デザイン 2(既存手法) では AES 鍵のみを LHE で暗号化するためデザイン 1 よりも IoT デバイスからクラウドサーバへ送信するファイルサイズは小さい．しかしながら，パラメータの Level が高く設定されている影響で図 4(h) の通信量，つまり IoT デバイスに配布する LHE の公開鍵が約 2.4GiB とデザイン 1(Baseline)・3(提案手法) よりも大きくなる．デザイン 3(提案手法) では IoT デバイスが LHE 暗号文を生成しないため，図 4(g) のクラウドサーバへ送信する暗号文のサイズも図 4(h) の復号サーバから受け取る鍵のサイズも最も小さくなる．

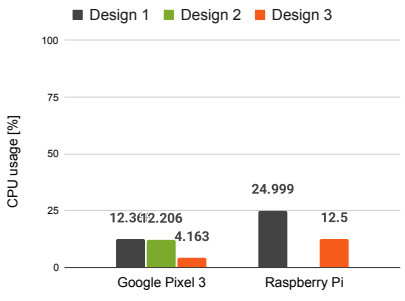
5.4.4 結　　論

どのシステムデザインを採用するべきかは以下のように目的によって異なる．

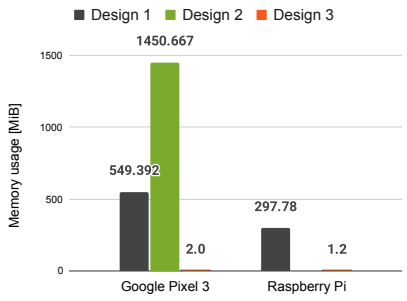
5：https://lwe-estimator.readthedocs.io/en/latest/index.html



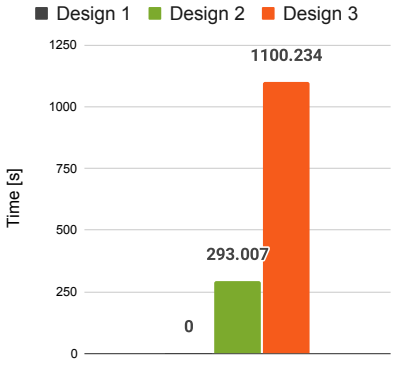
(a) IoT デバイスの実行時間



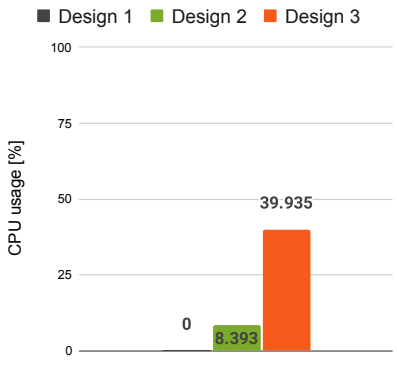
(b) IoT デバイスの CPU 使用率



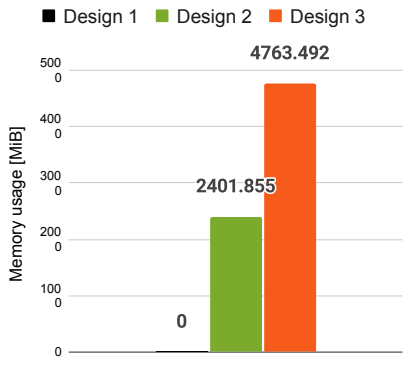
(c) IoT デバイスのメモリ使用量



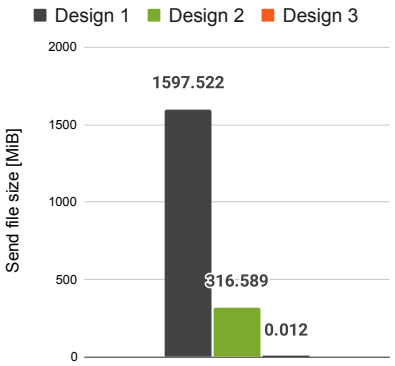
(d) クラウドサーバの実行時間



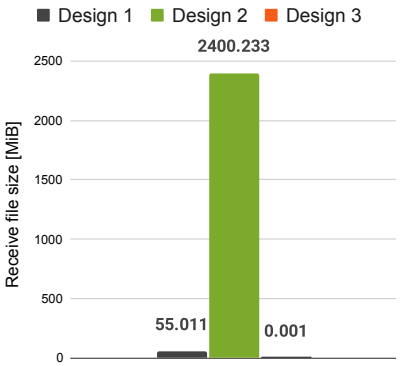
(e) クラウドサーバの CPU 使用率



(f) クラウドサーバのメモリ使用量



(g) 通信量 (IoT–CS 間)



(h) 通信量 (DS–IoT 間)

図 4: 256B の平文を LHE の暗号文にする場合の実験結果．デザイン 1 は最もシンプルな Baseline，デザイン 2 は既存手法，デザイン 3 が提案手法である． (a)IoT デバイスの実行時間は平文の暗号化時間，(b) と (c) はその暗号化処理の CPU 使用率とメモリ使用量である．(d) クラウドサーバの実行時間は LHE の暗号文へ変換する場合の実行時間で，(e) と (f) はその変換処理の CPU 使用率とメモリ使用量を指す．(g) の通信量は IoT デバイスがクラウドサーバに送る暗号文のファイルサイズ，(h) は復号サーバが IoT デバイスに配布する公開鍵のファイルサイズを指す．

- IoT デバイスの負荷削減のみを優先する場合

– デザイン 3(提案手法) を選択する．ただし，クラウドサーバへの負荷は高くなる．

- クラウドサーバの負荷削減のみを優先する場合

– デザイン 1(Baseline) を選択する．ただし，IoT デバイスへの負荷と IoT デバイス–クラウドサーバ間の通信量は多くなる．

- IoT デバイスとクラウドサーバの負荷削減を優先する場合

– デザイン 2(既存手法) を選択する．ただし，IoT デバイスに配布する鍵のサイズが大きくなり， Raspberry Pi のような低性能なデバイスではメモリ不足で実行不可能になる．

- 通信量削減を優先する場合

– デザイン 3(提案手法) を選択することで，IoT デバイス–クラウドサーバ間と復号サーバ–IoT デバイス間の両方の通信量が抑えられる．

一般論としては，クラウドサーバにおけるリソースは豊富であるのに対し，スマートフォンを含む IoT デバイスの負荷や通

信量は出来るだけ減らしたい場合が多いと考えられ，そのような場合に提案手法のデザイン 3 は有効であるといえる．

6 まとめと今後の課題

IoT デバイスの LHE 暗号化の負荷を削減するため，IoT デバイス側では LHE よりも軽量な AHE によって平文を暗号化し，代わりにクラウドサーバ側で AHE から LHE の暗号文へ変換する手法を提案した．実験の結果，既存手法に比べて，提案手法では IoT デバイスへの負荷を格段に減らすことが可能であり，Raspberry Pi のようなメモリの限られたデバイスでも実行可能であることを示した．よって，提案手法の方が優れたシステムデザインであると考えられる．

今後の課題として，クラウドサーバにおける変換処理の高速化が挙げられる．

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] Frederik Armknecht, C. Boyd, Christopher Carr, Kristian Gjøsteen, Angela Jäschke, Christian A. Reuter, and M. Strand. A guide to fully homomorphic encryption. *IACR Cryptol. ePrint Arch.*, Vol. 2015, p. 1192, 2015.

[3] 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.

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

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

[6] 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.

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

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] Martin R. Albrecht, Christian Rechberger, Thomas Schneider, Tyge Tiessen, and Michael Zohner. Ciphers for mpc and fhe. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015*, pp. 430–454, 2015.

[13] Itai Dinur, Yunwen Liu, Willi Meier, and Qingju Wang. Optimized interpolation attacks on lowmc. In Tetsu Iwata and Jung Hee Cheon, editors, *Advances in Cryptology – ASIACRYPT 2015*, pp. 535–560, 2015.

[14] 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.

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

[16] 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.

[17] 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.

[18] Jing Ma, Si-Ahmed Naas, Stephan Sigg, and Xixiang Lyu. Privacy-preserving federated learning based on multi-key homomorphic encryption. arXiv preprint arXiv:2104.06824, 2021.

[19] Deepika Natarajan and Wei Dai. Seal-embedded: A homomorphic encryption library for the internet of things. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, Vol. 2021, No. 3, p. 756–779, Jul. 2021.

[20] McKenzie van der Hagen and Brandon Lucia. Practical encrypted computing for iot clients. arXiv preprint arXiv:2103.06743, 2021.

[21] Jakub Konečný, H. Brendan McMahan, Felix X. Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. arXiv preprint arXiv:1610.05492, 2017.

[22] Le Trieu Phong, Yoshinori Aono, Takuya Hayashi, Lihua Wang, and Shiho Moriai. Privacy-preserving deep learning via additively homomorphic encryption. *IEEE Transactions on Information Forensics and Security*, Vol. 13, No. 5, pp. 1333–1345, 2018.

[23] Hongxu Yin, Arun Mallya, Arash Vahdat, Jose M. Alvarez, Jan Kautz, and Pavlo Molchanov. See through gradients: Image batch recovery via gradinversion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 16337–16346, June 2021.

[24] H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-efficient learning of deep networks from decentralized data. arXiv preprint arXiv:1602.05629, 2017.

[25] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *Advances in Cryptology — EUROCRYPT '99*, pp. 223–238, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[26] Hao Chen, Wei Dai, Miran Kim, and Yongsoo Song. Efficient multi-key homomorphic encryption with packed ciphertexts with application to oblivious neural network inference. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, CCS '19, p. 395–412, New York, NY, USA, 2019. Association for Computing Machinery.

[27] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology – ASIACRYPT 2017*, pp. 409–437, Cham, 2017. Springer International Publishing.