IoT デバイスにおける 準同型暗号を用いた暗号化手法の比較

松本 茉倫[1] 小口 正人[1]

概要：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大き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，スマートフォン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てシステムデザインを 3 つ実装し，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．

Comparison of Encryption Algorithms using Homomorphic Encryption on IoT devices

MARIN MATSUMOTO[1] MASATO OGUCHI[1]

1. はじめに

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) 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 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 を使ってシステムを実装した．

しかし，LHE・FHE 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大き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．そこで，Lauter ら [1] はクライアントへの負荷削減のため，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 の暗号文へ変換することが可能である．本稿では，Lauter らの手法を基にして，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) で暗号化し，クラウドサーバ側で LHE の暗号文へ変換する手法と Lauter らの手法を含め 3 つのシステムデザインを実装した．

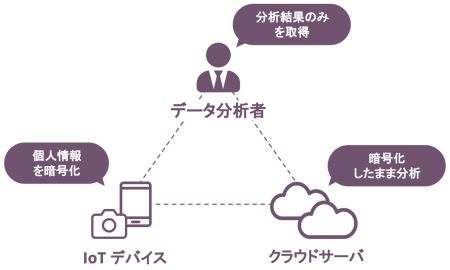


図 1: 準同型暗号を用いたビッグデータ活用システム

本稿の貢献を以下に示す．

[1] お茶の水女子大学
Ochanomizu University

- LHE でデータ暗号化した際の IoT デバイスへの負荷を把握するため，実際にスマートフォンを用いて実験を行った．
- シンプルに LHE で暗号化する手法・AES と LHE を組み合わせた手法・AHE と LHE を組み合わせた手法の 3 つを 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た．

2. 準備

本章では，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．

本稿で使われる記号の一覧を表 1 にまとめた．

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，Somewhat 準同型暗号 (以下 SHE:Somewhat Homomorphic Encryption)，LHE，FHE の 3 種類に分類される [2][3]．

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．

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．

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．

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．

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．
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．

n を 2 の巾乗，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．

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

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．

$$s \leftarrow R_2 \tag{2}$$

$$a \leftarrow R_q \tag{3}$$

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．

$$e \leftarrow N(0, \sigma^2) \tag{4}$$

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．

$$b := [as + 2e]_q \tag{5}$$

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．

$$v \leftarrow R_2 \tag{6}$$

$$e_0, e_1 \leftarrow N(0, \sigma^2) \tag{7}$$

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．

$$c_0 := [bv + 2e_0 + ptxt]_q. \tag{8}$$

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．

$$c_1 := [av + 2e_1]_q \tag{9}$$

(c_0, c_1) が暗号文 $ctxt$ となる．

暗号文同士の加算

2 つの暗号文を $ctxt = (c_0, c_1), ctxt' = (c'_0, c'_1)$ とする．$ctxt, ctxt'$ 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．

$$d := ([c_0 + c'_0]_q, [c_1 + c'_1]_q) \tag{10}$$

暗号文 d が暗号文同士の加算結果 $ctxt + ctxt'$ 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．

$$ptxt = [[c_0 - sc_1]_q]_2 \tag{11}$$

正しく復号できるかを確認する．

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

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. LHE・FHE の暗号化処理高速化の関連研究

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 またはへの変換と呼ぶ．Gentry ら [7] は実際に AES から FHE 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りその結果，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

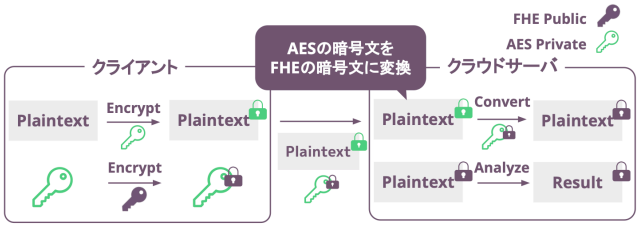


図 2: Lauter ら [1] の提案した手法．平文を AES で暗号化し，AES の鍵だけを FHE で暗号化することで暗号化の高速化が可能．FHE で暗号化するデータが最大でも AES の鍵長になる．

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[15] で高い評価を得た Trivium[16] 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 [17] 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，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．

4. システムデザイン

4.1 概要

本節のシステムデザインでは 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．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)

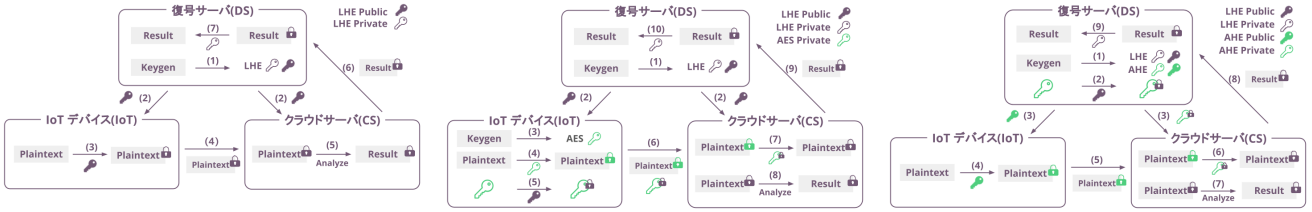
IoT デバイスはデータ提供者として，復号サーバから受信した公開鍵を用いて自身の個人情報を暗号化しクラウドサーバに送信する．

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．IoT デバイスから集めたデータを使って機械学習や統計分析などの演算を準同型暗号で暗号化したままで実行する．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．



(a) デザイン 1: IoT デバイスは平文を LHE で暗号化する．

(b) デザイン 2: 平文は AES で暗号化し，AES 鍵は LHE で暗号化する．

(c) デザイン 3: IoT デバイスは平文を AHE で暗号化する．

図 3: システムデザイン

アルゴリズム 1: IoT デバイスでは LHE で平文を暗号化しクラウドサーバが統計分析等の演算を行う．(デザイン 1)

復号サーバ
(1) Key generation
$sk_{LHE}, pk_{LHE} \leftarrow Keygen_{LHE}()$
(2) Send pk_{LHE} to IoT and CS
$Send(pk_{LHE}, IoT)$
$Send(pk_{LHE}, CS)$
IoT デバイス
(3) Encrypt $ptxt$ with LHE
for $i = 1$ to $ptxt.length$ **do**
$LHE(ptxt)[i] \leftarrow Enc_{LHE}(ptxt[i])$
end for
(4) Send $LHE(ptxt)$ to CS
$Send(LHE(ptxt), CS)$
クラウドサーバ
(5) Statistical analysis (homomorphic)
$LHE(result) \leftarrow Analyze(LHE(ptxt))$
(6) Send $result$ to DS
$Send(LHE(result), DS)$
復号サーバ
(7) Decrypt $LHE(result)$ with sk_{LHE}
$result \leftarrow Dec_{LHE}(LHE(result))$

4.2 デザイン 1

デザイン 1(図 3a，アルゴリズム 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．

4.3 デザイン 2

図 3b，アルゴリズム 2 には Lauter らの手法 [1] を基にしたデザイン 2 を示した．このデザインではデザイン 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 暗号文を LHE の暗号文へ変換する処理の実装は Gentry ら [7] とほぼ同じである．この実装では複数の暗号文を一つの暗号文として処理する暗号文パッキング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 暗号文に変換可能である．

アルゴリズム 2: Lauter らの手法 [1] を基にして，IoT デバイスは平文は AES で，AES 鍵は LHE で暗号化する．(デザイン 2)

復号サーバ
(1) Key generation
$sk_{LHE}, pk_{LHE} \leftarrow Keygen_{LHE}()$
(2) Send pk_{LHE} to IoT and CS
$Send(pk_{LHE}, IoT)$
$Send(pk_{LHE}, CS)$
IoT デバイス
(3) AES key generation
$sk_{AES} \leftarrow Keygen_{AES}()$
// Produces 11 round keys
$RoundKey \leftarrow AESKeyExpansion(sk_{AES})$
(4) Encrypt $ptxt$ with AES
for $i = 1$ to $ptxt.length/128$ **do**
$AES(ptxt)[i] \leftarrow Enc_{AES}(ptxt[(i-1) * 128])$
end for
(5) Encrypt $RoundKey$ with LHE
for $i = 1$ to 11 **do**
$LHE(RoundKey)[i] \leftarrow Enc_{LHE}(RoundKey[i])$
end for
(6) Send $AES(ptxt)$ and $LHE(RoundKey)$ to CS
$Send(AES(ptxt), CS)$
$Send(LHE(RoundKey), CS)$
クラウドサーバ
(7) AES decryption (homomorphic)
for $i = 1$ to 11 **do**
$LHE(state) \leftarrow$
$AddRoundKey(LHE(RoundKey)[i], AES(ptxt))$
$LHE(state) \leftarrow InvMixColumns(LHE(state))$
$LHE(state) \leftarrow InvShiftRows(LHE(state))$
$LHE(state) \leftarrow InvSubBytes(LHE(state))$
end for
$LHE(ptxt) \leftarrow LHE(state)$
(8) Statistical analysis (homomorphic)
$LHE(result) \leftarrow Analyze(LHE(ptxt))$
(9) Send $result$ to DS
$Send(LHE(result), DS)$
復号サーバ
(10) Decrypt $LHE(result)$ with sk_{LHE}
$result \leftarrow Dec_{LHE}(LHE(result))$

4.4 デザイン 3

デザイン 3(図 3c，アルゴリズム 3) ではデザイン 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

アルゴリズム 3: IoT デバイスは AHE で平文を暗号化し，クラウドサーバが LHE の暗号文へ変換する．(デザイン 3)

復号サーバ
(1) Key generation
$sk_{AHE}, pk_{AHE} \leftarrow Keygen_{AHE}()$
$sk_{LHE}, pk_{LHE} \leftarrow Keygen_{LHE}()$
(2) Encrypt sk_{AHE} with LHE
for $i = 1, \cdots, sk_{AHE}.length$ **do**
$LHE(sk_{AHE})[i] \leftarrow Enc_{LHE}(sk_{AHE}[i])$
end for
(3) Send $LHE(sk_{AHE})$ to IoT
$Send(LHE(sk_{AHE}), IoT)$
(3) Send pk_{LHE} to CS
$Send(pk_{LHE}, CS)$
IoT デバイス
(4) Encrypt $ptxt$ with AHE
// n: Degree of a polynomial
for $i = 1$ to $ptxt.length/n$ **do**
$AHE(ptxt)[i] \leftarrow Enc_{AHE}(ptxt[(i-1) * n])$
end for
(5) Send $AHE(ptxt)$ to CS
$Send(AHE(ptxt), CS)$
クラウドサーバ
(6) AHE decryption (homomorphic)
for $i = 1$ to $ptxt.length/n$ **do**
$(c_0, c_1) \leftarrow AHE(ptxt)[i]$
$LHE(ptxt)[i] \leftarrow c_0 - LHE(sk_{AHE}) * c_1$
end for
(7) Statistical analysis (homomorphic)
$LHE(result) \leftarrow Analyze(LHE(ptxt))$
(8) Send $result$ to DS
$Send(LHE(result), DS)$
復号サーバ
(9) Decrypt $LHE(result)$ with sk_{LHE}
$result \leftarrow Dec_{LHE}(LHE(result))$

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．

AHE から LHE の暗号文へ変換する処理では，LHE で暗号化した AHE の秘密鍵を AHE 暗号文の第 2 成分 c_1 に掛けたものを第 1 成分 c_0 から引く．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．

5. 実験

5.1 実験環境

実験プログラムの LHE は準同型暗号ライブラリの HElib[*1]で実装し，AHE の実装は 2.2 節に示した通りである．実験に使用したマシンの性能を表 2 に示した．

5.2 実験概要

IoT デバイスとしてスマートフォンの Google Pixel 3 を使用し，デザイン 1–3 で 256B の平文を LHE で暗号化する場合で IoT デバイスへの負荷 (IoT デバイスにおける実行時間)・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)・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．

表 2: 実験環境

IoT デバイス (Google Pixel 3)	OS	Android 9.0
	CPU	ARM Cortex-A75(2.5 GHz) 4 Cores ARM Cortex-A55(1.6 GHz) 4 Cores
	Main Memory	4GB
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 3: パラメータ

デザイン 1	LHE	Level: 4, Security:140 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

表 3 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，統計分析等で暗号文同士の演算が同じ回数行えるように Level を選択し，同程度のセキュリティレベルになるように設定した．デザイン 2 では AES から LHE への変換処理で暗号文同士の乗算が他 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[*2]で算出した．

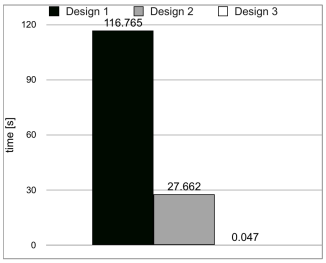
5.3 実験結果

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．実行時間を示すグラフ (図 4a，図 4b) の値は 5 回実行した場合の平均値である．図 4a の IoT デバイスへの負荷と図 4b クラウドサーバへの負荷はトレードオフの関係にある．

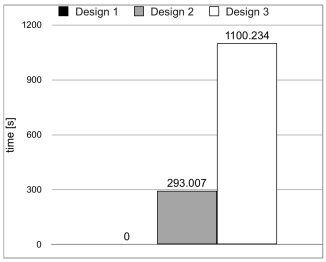
平文を LHE で暗号化して送信するだけのデザイン 1 では Lauter らの手法 [1] を基にしたデザイン 2 よりも IoT デバイスへの負荷が約 4 倍になるという結果だった．図 4c の IoT デバイスからクラウドサーバへ送信するファイルサイズについては，デザイン 2 と大差ないように見えるが，平文が長くなると，平文を LHE で暗号化するデザイン 1 ではファイルサイズが大きく増加していくが，デザイン 2 では AES 鍵のみを LHE で暗号化するため増加幅は小さい．

デザイン 2 では，IoT デバイスへの負荷とクラウドサー

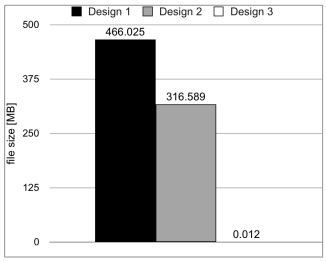
[*1] https://github.com/homenc/HElib.git
[*2] https://lwe-estimator.readthedocs.io/en/latest/index.html



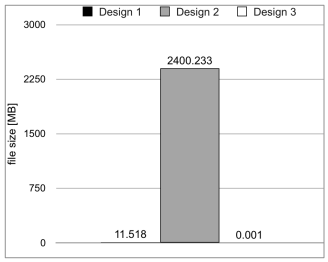
(a) IoT デバイスへの負荷



(b) クラウドサーバへの負荷



(c) 通信量 (IoT–CS 間)



(d) 通信量 (DS–IoT 間)

図 4: 256B の平文を LHE の暗号文にした場合の実験結果．(a)IoT デバイスへの負荷は IoT デバイスの実行時間，(b) クラウドサーバへの負荷は LHE の暗号文へ変換する場合の実行時間である．(c) 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，(d) 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．

バへの負荷のバランスが良いが，パラメータの Level が高く設定されている影響で図 4d の通信量，つまり IoT デバイスに配布する LHE の公開鍵が約 2.4GB とデザイン 1・3 よりも大きくなる．

IoT デバイスが LHE を一切使わないデザイン 3 では IoT デバイスへの負荷はデザイン 1 の約 2480 分の 1，デザイン 2 の約 590 分の 1 で格段に低い．一方で，クラウドサービスへの負荷はデザイン 2 よりも高くなっている．この理由としてはデザイン 3 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．

結論として，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．

- IoT デバイスの負荷削減のみを優先する場合
  - デザイン 3 を選択する．だたし，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 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大きくなる．
- 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 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る．
- 通信量削減を優先する場合
  - デザイン 3 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．

6. まとめ

本稿では，スマートフォンから収集したデータをクラウドサーバで安全に活用するためのシステムにおいて課題となる LHE による暗号化手法を 3 つのシステムデザインを実装し比較した．

デザイン 1 では IoT デバイスが LHE を使って平文を暗号化する手法で実装がシンプルであるものの，IoT デバイスでの実行時間が長く，暗号文サイズが大きくなることが示された．デザイン 2 は AES で平文を暗号化し，AES 鍵を LHE で暗号化する手法で，クラウドサーバが LHE の暗号文に変換する処理が必要である．IoT デバイスへの負荷とクラウドサーバへの負荷のバランスが優れているが，IoT デバイスに配布する公開鍵のサイズが 2GB を超え非常に大きいことが分かった．デザイン 3 は IoT デバイスが LHE よりも軽量な AHE で平文を暗号化する手法で，クラウドサーバが LHE の暗号文に変換する処理が必要である．IoT デバイスへの負荷と通信量の点で優れているがクラウドサーバでの変換処理の重いため改善が必要であることが分かった．

今後の課題として，AHE 暗号文を LHE 暗号文に変換する処理の高速化や LHE を使った統計分析や機械学習モデルの構築など具体的なアプリケーションを実装した場合のシステムデザインの比較が挙げられる．

謝辞

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

参考文献

[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 ENCRYP-*

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

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

[17] 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.