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

松本 茉倫[1] 小口 正人[1]

概要：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・乗算が可能な完全準同型暗号が注目されている．しかし，一般的に共通鍵暗号よりも公開鍵暗号は低速であり，公開鍵暗号である完全準同型暗号は処理時間がかかるため，計算能力の低い IoT デバイス上での実装が課題である．本研究では，低速な公開鍵暗号を高速に利用することを目的として，高速な共通鍵暗号と鍵共有が容易な公開鍵暗号を組み合わせたハイブリッド暗号を応用し，共通鍵暗号と完全準同型暗号を組み合わせた暗号化を IoT デバイスにおいて提案・実装する．

A Study of Speeding Up Sensor Data Encryption with a Common Key Cryptosystem combined with Fully Homomorphic Encryption on IoT Devices

MARIN MATSUMOTO[1] MASATO OGUCHI[1]

1. はじめに

スマートフォン・スマートウォッチといった IoT デバイスの普及によって，位置情報・心拍数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取得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，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．しかし通常の暗号では暗号化されたセンサデータを復号せずにクラウド上で分析処理を行う事が出来ない．そこで，暗号文同士の加算・乗算が可能な性質を持つ完全準同型暗号 (以下 FHE: Fully Homomorphic Encryption) が注目されている．しかし，一般的に共通鍵暗号に比べて公開鍵暗号は低速で，公開鍵暗号である FHE による暗号化は共通鍵暗号に比べて処理に時間がかかる．また，FHE の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．

先行研究 [1] では現在主流な共通鍵暗号である AES と FHE を組み合わせた暗号化の高速化と通信量削減が提案され，先行研究 [2] では Laptop を実験に用いて実装した結果が示された [3]．

本研究では，計算能力の低いスマートフォンのような IoT デバイスを実験に用いて，共通鍵暗号と FHE を組み合わせることで FHE による暗号化の高速化と通信量削減を提案する．共通鍵暗号には AES だけではなく，ブロック暗号の KATAN[12] と欧州におけるストリーム暗号の評価プロジェクトの eSTREAM で選定された暗号の一つである TRIVIUM[15] を使用した．AES・KATAN・TRIVIUM と FHE を組み合わせて，IoT デバイスへの負荷・通信量・クラウド側への負荷という 3 つの指標で比較を行った．また，ブロック暗号の AES と KATAN は ECB モードと CTR モードで実装し，暗号化モードによる比較も行った．

[1] お茶の水女子大学
Ochanomizu University

2. 完全準同型暗号 (FHE)

2.1 特徴

FHE とは式 (1)，(2) のように暗号文同士の加法・乗法の演算が成立する性質をもつ暗号である．

完全準同型暗号

$$Encrypt(m) \oplus Encrypt(n) = Encrypt(m+n) \quad (1)$$
$$Encrypt(m) \otimes Encrypt(n) = Encrypt(m \times n) \quad (2)$$

FHE は公開鍵暗号の一つであり，秘密鍵で復号することなく暗号文同士の演算から平文同士の演算を暗号化した値を導くことができる．

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．しかしこの手法を用いた場合，1bit の平文を暗号化するとその暗号文は 1GB 程にもなってしまうなど，提案された当時は計算量の大きさから実用性がないとされていた．近年では効率の良い実装について多くの研究がなされて高速化や改良が進められており，実用化への期待が高まっている．

しかしながら，FHE には問題点もある．FHE の暗号文中の解読不可能性を高めるために加えられたランダムなノイズは，暗号文同士で計算を行うたびに増加する．特に乗算を行うことで大きく増加し，ノイズが閾値を超えると復号が不可能となる．演算可能な回数を決定するパラメータであるレベルを高く設定する，または bootstrapping と呼ばれるノイズをリセットする手法の導入することで演算回数の限定は解決することが出来るが，bootstrapping は計算量が非常に大きくなる．

2.2 leveled FHE

bootstrapping は暗号文同士の演算回数の制限を取り払う手法であるが，コストが高く実用的とは言い難い．本研究では bootstrapping は利用せずに，評価可能な回路の最大の深さを表すパラメータのレベルを任意の値にあらかじめ設定する leveled FHE という手法を用いる．レベル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．

2.3 ライブラリ

FHE を実装しているライブラリには HElib[6]，SEAL[7]，PALISADE[8] 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++で実装されている．PALISADE もまた現在 bootstrapping をサポートしていない．また，SEAL と PALISADE は外部ライブラリに依存していないが，HElib は GMP[9]，NTL[10] といった外部ライブラリに依存しているという特徴もある．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 を実装に用いる．

3. AES

AES(Advanced Encryption Standard)[11] とは，アメリカ連邦政府標準の暗号方式として 2000 年に採用された共通鍵暗号の一つである．また AES は高度な暗号化方式であり，現時点での解読方法は存在していないとされている．

AES は共通鍵暗号の一種であるブロック暗号で，鍵長は 128bit・192bit・256bit の 3 つが利用でき，128bit ずつ平文を区切って暗号化・復号を行う．本研究では，鍵長が 128bit の AES を用いた実装を行った．

4. KATAN

4.1 概要

KATAN[12][13] とはハードウェアで非常に効率的で RFID タグなどの低性能デバイスに適した軽量なブロック暗号である．鍵長は 80bit でブロック長が 32bit，48bit，64bit をサポートしている．本研究では，ブロック長が 64bit の KATAN を実装した．図 1 にあるように，KATAN の構造はストリーム暗号でしばしば用いられる非線形シフトレジスタに基づいている．十分に混合するためにラウンド数は 254 とされている．

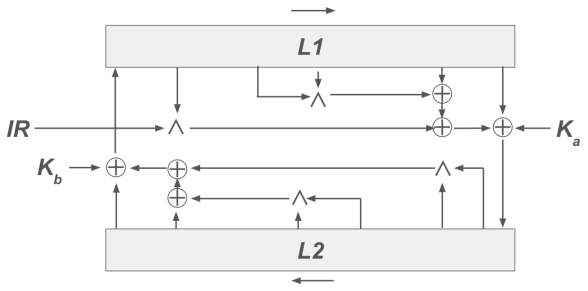

図 1: KATAN の構造

4.2 アルゴリズム

KATAN が AES と異なる特徴の一つに拡大鍵の生成方法がソースコード 1 に示しているように AES よりもシンプルであることが挙げられる．

ソースコード 1: KATAN Key Expansion

```
for(i=0;i<80;++i)
  k[i]=key[i];

for(i=80;i<2*254;++i)
  k[i]=k[i-80]^k[i-61]^k[i-50]^k[i-13] ;
```

平文を暗号化する際はソースコード 2 の処理を行う．

ソースコード 2: KATAN Encryption

```
for(i=0;i<39;++i)
  L2[i]=plain[i];
for(i=0;i<25;++i)
  L1[i]=plain[i+39];

for(i=0;i<80;++i)
  k[i]=key[i];
for(i=80;i<2*rounds;++i)
  k[i]=k[i-80]^k[i-61]^k[i-50]^k[i-13] ;

for(i=0;i<rounds;++i) {
  fa_2=L1[24]^L1[15]^(L1[20]&L1[11])^(L1[9]&IR[i
      ])^k[2*i];
  fa_1=L1[24-1]^L1[15-1]^(L1[20-1]&L1[11-1])^(L1
      [9-1]&IR[i])^k[2*i];
  fa_0=L1[24-2]^L1[15-2]^(L1[20-2]&L1[11-2])^(L1
      [9-2]&IR[i])^k[2*i];
  fb_2=L2[38]^L2[25]^(L2[33]&L2[21])^(L2[14]&L2
      [9])^k[2*i+1];
  fb_1=L2[38-1]^L2[25-1]^(L2[33-1]&L2[21-1])^(L2
      [14-1]&L2[9-1])^k[2*i+1];
  fb_0=L2[38-2]^L2[25-2]^(L2[33-2]&L2[21-2])^(L2
      [14-2]&L2[9-2])^k[2*i+1];

  for(j=24;j>2;--j)
    L1[j]=L1[j-3];
  for(j=38;j>2;--j)
    L2[j]=L2[j-3];
  L1[2]=fb_2;
  L1[1]=fb_1;
  L1[0]=fb_0;
  L2[2]=fa_2;
  L2[1]=fa_1;
  L2[0]=fa_0;
}

for(i=0;i<39;++i)
  cipher[i]=L2[i];
for(i=0;i<25;++i)
  cipher[i+39]=L1[i];
```

5. 暗号化モード

ブロック暗号である AES･KATAN にはブロック長 (AES では 128bit，KATAN では 64bit) よりも長い平文に対応するために暗号化モードを利用する．最も単純な仕組みの暗号化モードである ECB モードと CRYPTREC[14] で推奨され一般的に使用されている CTR モードについて示す．

5.1 ECB モード

図 2 は ECB モードの暗号化処理，図 3 は復号処理の概要である．図 2，3 の様に同じ平文ブロックを暗号化すると全て同じ暗号文になり，解読されやすくなってしまうという弱点がある．

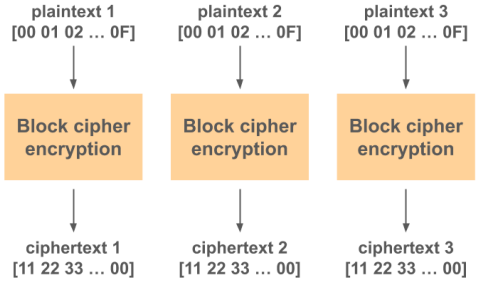

図 2: ECB モードによる暗号化

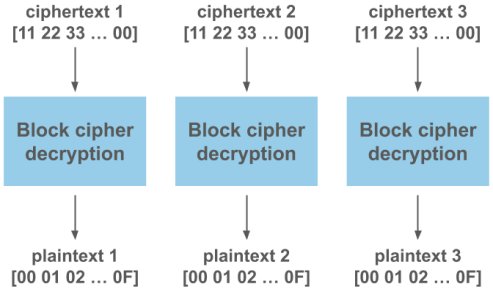

図 3: ECB モードによる復号

5.2 CTR モード

図 4 には CTR モードの暗号化処理を示す．ECB モードと異なるのはブロック暗号のアルゴリズムで暗号化するのはランダムに決められた Nonce と何番目のブロックかを示す Counter を組み合わせた値で，暗号化された Nonce，Counter と平文の XOR 演算を行うことで暗号文を生成する．ECB モードの様に同じ平文ブロックが存在しても同じ暗号文になることはない．

また，図 5 にあるように CTR モードの特徴として，復号する場合もブロック暗号の暗号化アルゴリズムを利用することが挙げられる．

6. TRIVIUM

6.1 概要

TRIVIUM[15] とは，2008 年に eSTREAM という欧州における共通鍵暗号の一種，ストリーム暗号の評価プロジェクトで選定された暗号の一つである．TRIVIUM は構造が

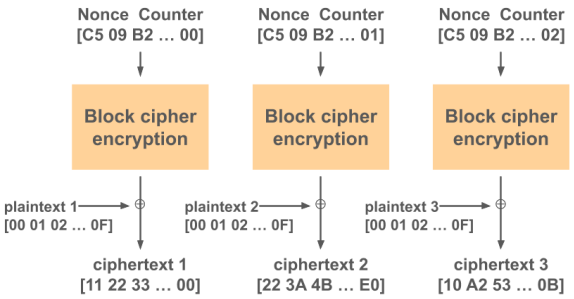

図 4: CTR モードによる暗号化

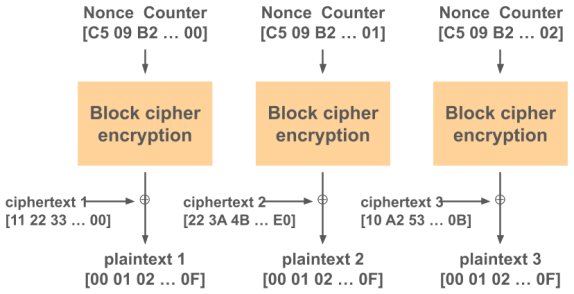

図 5: CTR モードによる復号

単純かつ高速であるため，eSTREAM に選定された 7 つの暗号の中でも特に高い評価を与えられた [16]．また，現時点では解読方法は存在していないとされている．

6.2 アルゴリズム

TRIVIUM では，80bit の鍵 (K)・80bit の初期化ベクトル (IV)・288bit の初期状態 (s) をパラメータとして使用し，以下のように K と IV で s の初期化を行う．

Key and IV setup

$(s_1, s_2, ..., s_{93}) \leftarrow (K_1, ..., K_{80}, 0, ..., 0)$
$(s_{94}, s_{95}, ..., s_{177}) \leftarrow (IV_1, ..., IV_{80}, 0, ..., 0)$
$(s_{178}, s_{279}, ..., s_{288}) \leftarrow (0, ..., 0, 1, 1, 1)$
$for\, i = 1\, to\, 4 \cdot 288\, do$
$t_1 \leftarrow s_{66} + s_{91} \cdot s_{92} + s_{93} + s_{171}$
$t_2 \leftarrow s_{162} + s_{175} \cdot s_{176} + s_{177} + s_{264}$
$t_3 \leftarrow s_{243} + s_{286} \cdot s_{287} + s_{288} + s_{69}$
$(s_1, s_2, ..., s_{93}) \leftarrow (t_3, s_1, ..., s_{92})$
$(s_{94}, s_{95}, ..., s_{177}) \leftarrow (t_1, s_{94}, ..., s_{176})$
$(s_{178}, s_{279}, ..., s_{288}) \leftarrow (t_2, s_{178}, ..., s_{287})$
$end\, for$

暗号化・復号の際は，初期化した s を利用して以下のように生成した Key stream(z) と平文の XOR 演算を行う．

Key stream generation

$for\, i = 1\, to\, N\, do$
$t_1 \leftarrow s_{66} + s_{93}$
$t_2 \leftarrow s_{162} + s_{177}$
$t_3 \leftarrow s_{243} + s_{288}$
$z_i \leftarrow t_1 + t_2 + t_3$
$t_1 \leftarrow t_1 + s_{91} \cdot s_{92} + s_{171}$
$t_2 \leftarrow t_2 + s_{175} \cdot s_{176} + s_{264}$
$t_3 \leftarrow t_3 + s_{286} \cdot s_{287} + s_{69}$
$(s_1, s_2, ..., s_{93}) \leftarrow (t_3, s_1, ..., s_{92})$
$(s_{94}, s_{95}, ..., s_{177}) \leftarrow (t_1, s_{94}, ..., s_{176})$
$(s_{178}, s_{279}, ..., s_{288}) \leftarrow (t_2, s_{178}, ..., s_{287})$
$end\, for$

7. ハイブリッド暗号

ハイブリッド暗号とは共通鍵暗号と公開鍵暗号を組み合わせることで，それぞれの暗号方式の欠点を補い，長所を組み合わせた暗号方式である．図 6 にハイブリッド暗号の概要を示す．

まずクライアントは共通鍵，サーバは公開鍵と秘密鍵を生成し，公開鍵をサーバからクライアントに送信して共有する．次にクライアントは公開鍵で共通鍵を暗号化し，サーバに送信して共有する．そうすることで，サーバでは秘密鍵でこの鍵を復号し，共通鍵を安全に手に入れることができる．このように共通鍵を共有することによって，これ以降，高速にクライアントで暗号化，サーバで復号することができる．

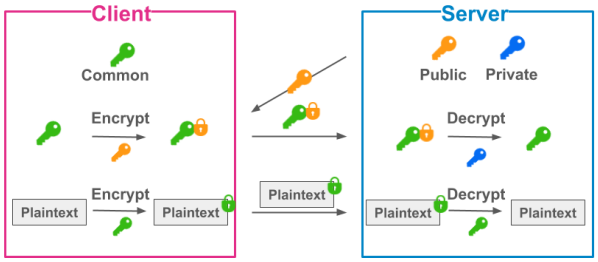

図 6: ハイブリッド暗号

8. 先行研究

Gentry ら (2015) は 10.1 節で述べる提案手法の (6) の処理にあたる，FHE による AES の復号処理の実装と評価を行った [2]．AES を評価に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 (10 ラウンド+1) × 128bit の AES の鍵を使い復号する設計になっている．実験に使用した Laptop の性能を

表 1 に示す． 実装はシングルスレッドであるため，1 つ

表 1: 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

のコアのみ使用している．2880B の AES 暗号文の復号に 18 分，16B あたり 6 秒を要した．

9. 従来手法

提案手法の比較として，FHE のみを暗号化に使用する従来システムの概要を図 7 に示し，FHE only とする．

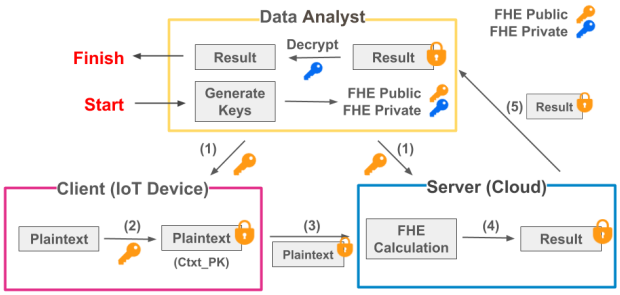

図 7: 従来システム (FHE only)

(1) Data Analyst が FHE の公開鍵と秘密鍵を生成し，Client(IoT デバイスユーザ) と Server(クラウドサービス) に公開鍵を送信．
(2) Client は FHE の公開鍵で暗号化された暗号文 (Ctxt_PK) を生成．
(3) Ctxt_PK を Server に送信．
(4) Server は Ctxt_PK を使って分析．
(5) Server は分析結果を Data Analyst に送信．

従来システムの問題点として，Client における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挙げられる．

10. 提案手法

10.1 概要

本研究では，IoT デバイスにおいて図 8 の共有鍵暗号と公開鍵暗号の FHE を組み合わせた暗号化システムを提案する．図 8 中の CommonKey は共通鍵暗号の AES・KATAN・TRIVIUM の 3 通りで，それぞれの暗号を使った提案システムを AES+FHE・KATAN+FHE・TRIVIUM+FHE とする．

(1) Data Analyst が FHE の公開鍵と秘密鍵を生成し， Client(IoT デバイスユーザ) と Server(クラウドサービス) に公開鍵を送信．
(2) Client は共通鍵 (CommonKey) を生成．

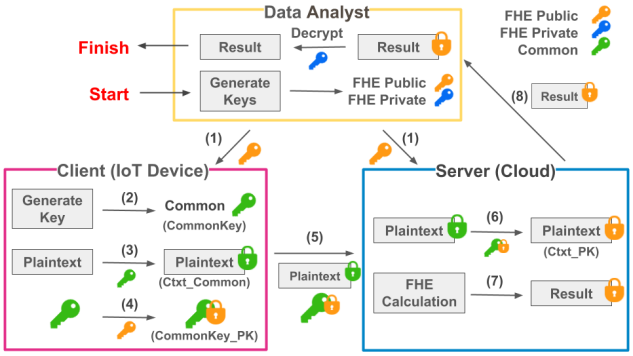

図 8: 提案システム (AES+FHE，TRIVIUM+FHE)

(3) Client は共通鍵で暗号化された暗号文 (Ctxt_Common) を生成．
(4) Client は FHE の公開鍵で暗号化された共通鍵 (CommonKey_PK) を生成．
(5) Client は Ctxt_Common と CommonKey_PK を Server に送信．
(6) Server は Ctxt_Common を CommonKey_PK で復号し，FHE の暗号文 (Ctxt_PK) を取得．
(7) Server は Ctxt_PK を使って分析．
(8) Server は分析結果を Data Analyst に送信．

平文が増加しても共通鍵で暗号化すれば良いため，従来手法よりも高速に暗号化できることが予想される．

提案手法では，7 節で説明したようなハイブリッド暗号と同様に共通鍵暗号と公開鍵暗号の FHE を組み合わせているが，ハイブリット暗号と異なるのは，サーバ (クラウド) は公開鍵と秘密鍵の作成も秘密鍵の所有もしないということである．サーバは FHE の性質を利用することで，FHE の公開鍵によって暗号化された共通鍵 (CommonKey_PK) を復号せずに共通鍵暗号によって暗号化された暗号文 (Ctxt_Common) を復号できる．

11. 実験

11.1 実験環境

実験に使用したマシンの性能を表 2 に示す．

表 2: マシン性能

Device	OS	CPU	Number of Cores	Processor Speed	RAM
Raspberry Pi 3 Model B+	Raspbian 10.0	ARM Cortex-A53	4	1.4GHz	1GB
Google Pixel 3	Android 9.0	ARM Cortex-A75 ARM Cortex-A55	8	2.5 GHz 1.6 GHz	4GB
MacBook Pro	OS X 10.15	Intel Core i5	4	1.4GHz	8GB

11.2 予備実験 1

FHE による 1B の暗号化時間を MacBook Pro，Google Pixel 3，Raspberry Pi で比較した結果を図 9 に示す．IoT

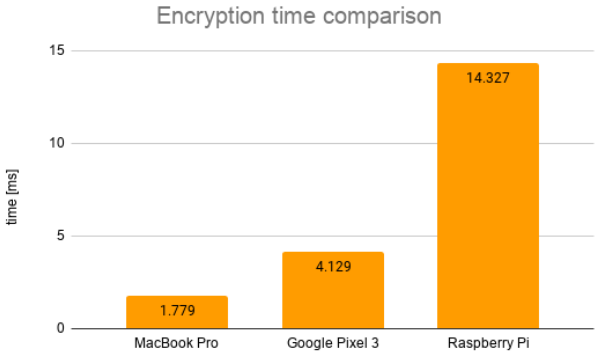

図 9: デバイスごとの FHE よる暗号化時間

デバイスである，Google Pixel 3 と Raspberry Pi に注目すると，処理速度は Google Pixel 3 が Raspberry Pi のおよそ 3.5 倍と格段に速いことがわかる．

また，IoT デバイスの中では高性能である Google Pixel 3 であっても，MacBook Pro に比べて約 2.3 倍低速である．

11.3 予備実験 2

提案手法で用いる共通鍵暗号の AES・KATAN・TRIVIUM と公開鍵暗号の FHE で 1B の暗号化時間を比較した結果を図 10 に示す．実験には Google Pixel 3 を使用した．

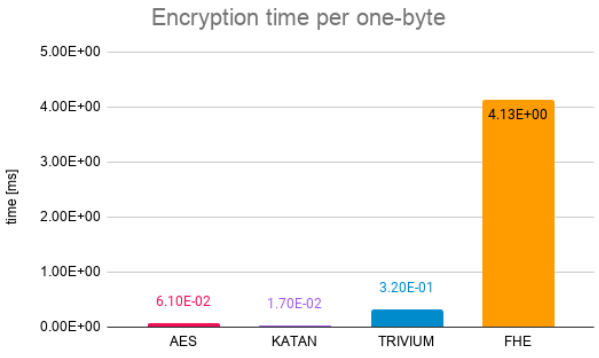

図 10: 暗号方式ごとの暗号化時間

共通鍵暗号に比べ，公開鍵暗号の FHE が低速であることは明らかである．

11.4 実験概要

Client は Google Pixel 3，Server は MacBook Pro を使用した．平文サイズを 16B，64B，128B，192B，256B，320B と変化させ，従来手法 (FHE only) と提案手法 (AES+FHE, KATAN+FHE，TRIVIUM+FHE) で以下の 3 つの比較を行った．また，ブロック暗号の AES と KATAN には 5 節で説明した ECB モードと CTR モードを実装した．

(1) Client における実行時間
(2) Client から Server に送信するファイルサイズ
(3) Server における共通鍵暗号の復号時間

11.5 実験結果

図 11 に (1) Client における実行時間 を示す．従来手法の実行時間には FHE による平文の暗号化が含まれ，提案手法の実行時間には共通鍵の生成時間・FHE による共通鍵の暗号化・共通鍵暗号による平文の暗号化が含まれている．

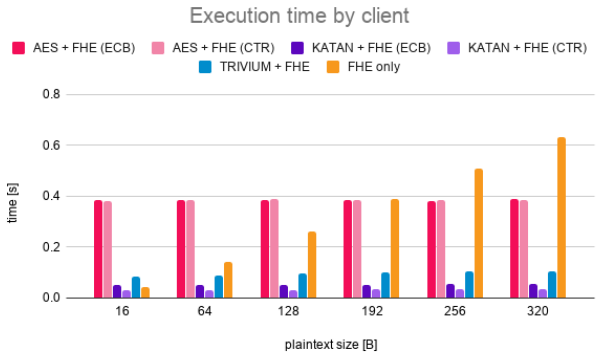

図 11: Client における実行時間

図 12 には (2) Client から Server に送信するファイルサイズ を示す．従来手法のファイルサイズには FHE の暗号文のみが含まれ，提案手法のファイルサイズには，FHE で暗号化された共通鍵と共通鍵で暗号化された暗号文が含まれる．

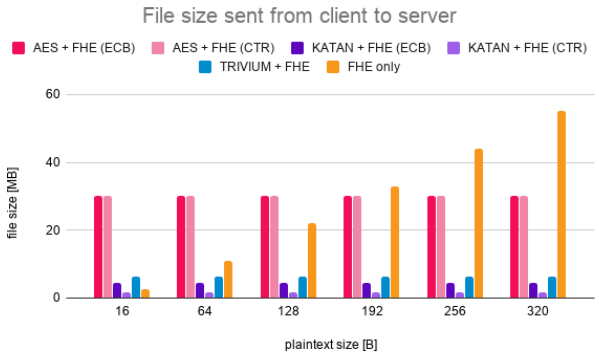

図 12: Client から Server に送信するファイルサイズ

図 13 に (3) Server における共通鍵暗号の復号時間を示す．FHE only では共通鍵暗号の復号処理は行わないため 0 秒である．

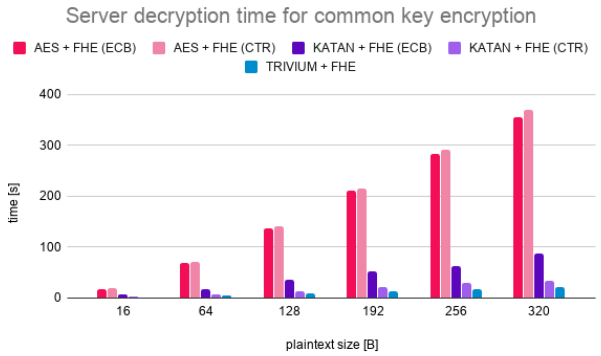

図 13: Server における共通鍵暗号の復号時間

11.6 実験結果まとめ

平文が長い場合 (plaintext size > 192) の従来手法と提案手法の比較を表 3 にまとめた．

表 3: 平文が長い場合

	Client load	Traffic	Server load	Safety
AES + FHE (ECB)	△	△	×	△
AES + FHE (CTR)	△	△	×	○
KATAN + FHE (ECB)	○	○	△	×
KATAN + FHE (CTR)	○	○	△	△
TRIVIUM + FHE	○	○	△	△
FHE only	×	×	○	○

表 4 には平文が短い場合 (plaintext size < 64) の従来手法と提案手法の比較をまとめた．

表 4: 平文が短い場合

	Client load	Traffic	Server load	Safety
AES + FHE (ECB)	×	×	×	△
AES + FHE (CTR)	×	×	×	○
KATAN + FHE (ECB)	○	○	△	×
KATAN + FHE (CTR)	○	○	△	△
TRIVIUM + FHE	△	△	△	△
FHE only	○	○	○	○

なお，表 3 と表 4 に示した Safety は，それぞれの手法で用いる暗号の鍵長から判断した簡易的なものである．AES + FHE と KATAN + FHE については，3.3 節にあるように ECB モードで暗号化した場合に同じ平文ブロックから同じ暗号文が生成されて強度が下がる．そのため，CTR モードの方が ECB モードよりも安全性が高いとした．

12. 考察

12.1 Client における実行時間 (Client への負荷)

図 11 より， 平文サイズが増加するにつれて，従来手法 (FHE only) と提案手法 (AES+FHE，KATAN + FHE, TRIVIUM+FHE) の差が顕著になっている．KATAN + FHE, TRIVIUM+FHE, AES+FHE の順番で高速な理由は，FHE で暗号化される共通鍵の鍵長が KATAN, TRIVIUM, AES の順で短いためである．提案手法におけるクライアントへの負荷は，共通鍵暗号の暗号化速度よりも共通鍵の鍵長が影響する．

一方で，平文サイズが小さい場合は，FHE only の方が高速であるという結果になった．提案手法で FHE によって暗号化される共通鍵の長さが，従来手法で FHE によって暗号化される平文より長い場合にそのような結果になる．

12.2 Client から Server に送信するファイルサイズ (通信量)

図 12 より，平文サイズが増加するにつれて，従来手法 (FHE only) と提案手法 (AES+FHE，KATAN+FHE, TRIVIUM+FHE) の差が顕著になっていることが分かる．これは，FHE の暗号文サイズが共通鍵暗号の暗号文サイズよりもはるかに大きいことが影響している．KATAN+FHE のファイルサイズが最も小さいのは，FHE で暗号化される共通鍵の鍵長が KATAN + FHE の場合に最も短いためである．

一方で，平文サイズが小さい場合は，FHE only の方が通信量が少ないという結果になった．提案手法で FHE によって暗号化される共通鍵の長さが，従来手法で FHE によって暗号化される平文より長い場合にそのような結果になる．

12.3 Server における共通鍵暗号の復号時間 (Server への負荷)

図 13 より，AES+FHE に比べて KATAN + FHE と TRIVIUM+FHE の方がサーバへの負荷が少ない．これは，AES の復号処理よりも KATAN + FHE と TRIVIUM+FHE の復号処理の方が加算・乗算の回数が少ないかつシンプルなためである．

12.4 暗号化モード

ECB モードと CTR モードとでは 5 節にあるように，サーバにおける暗号文の復号処理は異なるが，図 13 の AES + FHE (ECB) と AES + FHE(CTR) に注目すると，暗号化モードによるサーバへの負荷はあまり変わらない．

一方で, KATAN + FHE では ECB モードより CTR モードの方が高速で通信量も少ない．その理由は，KATAN で

は ECB モードによる復号処理は CTR モードの復号処理よりも演算回数が多いことから，2.2 節にあるような FHE で演算可能な回数を決定するパラメータであるレベルを高く設定する必要があるためである．図 14 は Google Pixel 3 におけるレベルごとの FHE による暗号化時間，図 15 ではレベルごとの暗号文サイズの比較を示している．これらの図から分かるように，レベルを高く設定するほど暗号化時間が長くなり，暗号文サイズが大きくなってしまう．

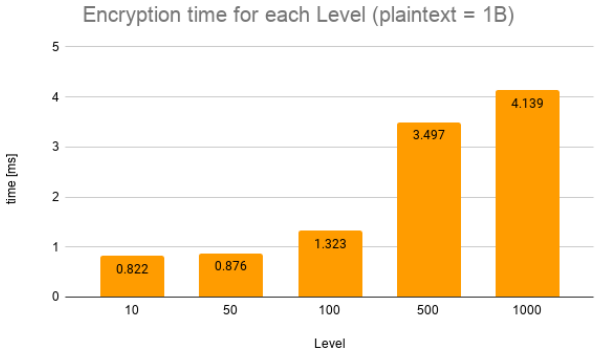

図 14: レベルごとの FHE による暗号化時間

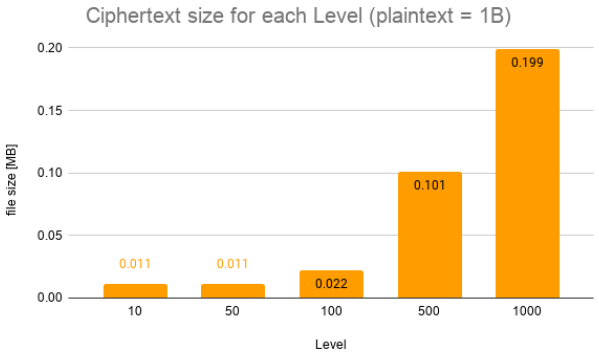

図 15: レベルごとの FHE の暗号文サイズ

12.5 結論

平文が短い場合には FHE only が有利であるが，全体の実行時間が短いため，提案手法との実行時間・通信量の差はわずかである．よって平文が長い場合に有利な手法を選択すべきである．また，一般的に IoT デバイスのようなクライアントはサーバに比べて性能が低いため，サーバに多少負荷がかかったとしてもクライアントへの負荷を減らすことを重視すべきである．

以上の理由から，平文が長い場合に有利でクライアントへの負荷が少ない提案手法が優れていると言える．ここで，安全性とパフォーマンスはトレードオフの関係にあるため，安全性を重視すれば遅くなりパフォーマンスを重視すれば安全性が下がる．したがって，目的に応じて適切な提案手法を選択すれば良い．

13. まとめと今後の課題

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有用な完全準同型暗号による暗号化を高速化することを目的として，共通鍵暗号の AES・KATAN・TRIVIUM と FHE を組み合わせた暗号を実装した．その結果，平文が長くなればなるほど従来手法よりも提案手法の方が IoT デバイスへの負荷と通信量を減らすことが可能になった．

今後は提案手法よりもクライアントへの負荷を減らせる構成の実装を検討している．

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, Vinod Vaikuntanathan: Can homomorphic encryption be practical?, Proc. of the 3rd ACM workshop on CCSW ’ 11, pp. 113–124, 2011
[2] Craig Gentry, Shai Halevi, Nigel P. Smart: Homomorphic Evaluation of the AES Circuit, January 3, 2015
[3] 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，第 15 回情報科学技術フォーラム，2016
[4] R. L. Rivest, L. Adleman, M. L. Dertouzos, et al.: On data banks and privacy homomorphisms, Foundations of secure computation 4.11 (1978), pp. 169–180.
[5] Craig Gentry, et al: Fully homomorphic encryption using ideal lattices. In STOC, Vol. 9, pp. 169–178, 2009
[6] HElib, `https://github.com/homenc/HElib` (2019/08 閲覧)
[7] Microsoft SEAL, `https://github.com/Microsoft/SEAL` (2019/08 閲覧)
[8] PALISADE, `https://palisade-crypto.org/software-library/` (2020/02 閲覧)
[9] GMP, `https://gmplib.org/` (2019/08 閲覧)
[10] NTL, `https://www.shoup.net/ntl/` (2019/08 閲覧)
[11] Joan Daemen and Vincent Rijmen: AES submission document on Rijndael, June, 1998
[12] Christophe De CannièreOrr DunkelmanMiroslav Knežević: KATAN and KTANTAN — A Family of Small and Efficient Hardware-Oriented Block Ciphers, Cryptographic Hardware and Embedded Systems - CHES 2009, pp. 272-288
[13] Lightweight Block Ciphers, `https://www.cryptolux.org/index.php/Lightweight_Block_Ciphers` (2019/08 閲覧)
[14] CRYPTREC 暗号リスト (電子政府水推奨暗号リスト), `https://www.cryptrec.go.jp/list.html` (2020/02 閲覧)
[15] Christophe De Canni‘ere, Bart Preneel: Trivium Specifications, eSTREAM, ECRYPT Stream Cipher Project, 2006
[16] 森井 昌克，寺村 亮一: ストリーム暗号の現状と課題，電子情報通信学会 基礎・境界ソサイエティ Fundamentals Review Vol.2 No.3，2009