DEIM Forum 2023 5b-9-4

圧縮センシングを利用した準同型暗号化データの通信量削減

泉　湖雪† 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: †koyuki@ogl.is.ocha.ac.jp, marin@ogl.is.ocha.ac.jp, oguchi@is.ocha.ac.jp

あらまし　近年の IoT デバイスやクラウドサービスの普及により，クラウド上で安全にデータを処理する必要がある．そこで，暗号文同士の加算や乗算が可能な準同型暗号が注目されている．しかし，準同型暗号は送信するデータが多次元であるほど通信量が大きくなってしまうという課題がある．また，収集するデータがスパースな場合は無駄が多くなる．そこで本研究では，零成分を多く含む (以下 スパースな) データを圧縮してから復元する「圧縮センシング」を利用して，通信量を削減してスパースな多次元データを送信することを提案する．実験では乱数により生成したスパースな多次元データを用いて，圧縮ありと圧縮なしの場合で，準同型暗号文のサイズ，サーバでの復元結果の精度，実行時間を比較する．

キーワード　準同型暗号，圧縮センシング, スパース

1 はじめに

スマートフォンなどの IoT デバイスが普及したことで，クラウドなどのサーバにデータが蓄積され，分析に利用されるようになった．しかし，収集したデータは個人に関する情報を含む場合があるため，外部からの攻撃などによるデータの漏洩に備える必要がある．そこで，暗号文同士の加算や乗算が可能な準同型暗号を用いることで，盗聴のリスクを減らせるだけでなく，暗号状態のままデータ分析を行うことが可能となる．準同型暗号の課題として，送信するデータが多次元であるほど通信量が大きくなってしまうことが挙げられる．また，収集するデータがスパースな場合は無駄が多くなる．したがって本研究では，図 1 に示すような圧縮センシングを利用した多次元データの送信を提案する．

圧縮センシングは，スパース性を利用することにより，少ないサンプル数からもとのデータを復元する技術である．圧縮行列 A を掛けることで低次元データを得て，得られた低次元データと圧縮行列 A から推定することで，元の多次元データを復元する．この技術を用いて圧縮されたスパースな多次元データを準同型暗号化してクライアントからサーバに送信することで，クライアントとサーバ間の通信量の削減が期待できる．

提案手法では，圧縮センシングによる圧縮と準同型暗号ライブラリの Microsoft SEAL を使った平文の準同型暗号化をクライアント側で行い，圧縮センシングの復元をサーバ側で行う．代表的な再構成アルゴリズムとして Orthogonal Matching Pursuit(以下 OMP) があるが，OMP で復元する際には比較演算を行う．暗号化された状態では比較演算が困難であるため，比較演算を使用しないような復元アルゴリズムを用いる必要がある．そこで，今回は一般逆行列による L_2 ノルムの最小化によって圧縮前のデータを推定する．実験では，乱数により生成したスパースな多次元データを用いて，Baseline と提案手法で，クライアントからサーバに送信する準同型暗号文のサイズ，サーバでの圧縮センシングによる復元結果の精度，実行時間の比較を行う．ここでは単純に，スパースな多次元データをクライアントで準同型暗号化して，圧縮せずにサーバに送信する方法を Baseline とする．

実験の結果，圧縮率を 50%に設定した場合は提案手法は圧縮せずに送信した場合に比べて 50%通信量を削減できることを示した．このとき，復元後のベクトル $\hat{\boldsymbol{x}}$ の二乗平均平方根誤差 (以下 RMSE: Root Mean Squared Error) はおよそ 0.15 となった．

2 既存技術

2.1 準同型暗号

2.1.1 特　　徴

準同型暗号とは，以下のような特徴を持つ公開鍵暗号である．

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

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

暗号文同士の加算が可能であるという特徴を表す (1) を加法準同型性といい，暗号文同士の乗算が可能であるという特徴を表す (2) を乗法準同型性という．準同型暗号には多くの種類が存在し，加法準同型性のみを持つ加法準同型暗号には Paillier 暗号 [1]，乗法準同型性のみを持つ乗法準同型暗号には RSA 暗号 [2] が挙げられる．

また，加法準同型性と乗法準同型性の両方を満たす準同型暗号は演算可能回数によって 3 つに分類できる．演算可能回数に制限がある場合は Somewhat Homomorphyc Encryption(以下 SHE) といい, パラメータにより演算可能回数が決まる場合は Leveled Homomorphic Encryption(以下 LHE) という [3]. 演算可能回数に制限がない準同型暗号は完全準同型暗号 (以下 FHE: Fully Homomorphic Encryption) という．ブートストラップを行い, 演算による誤差の蓄積をリセットすることにより，

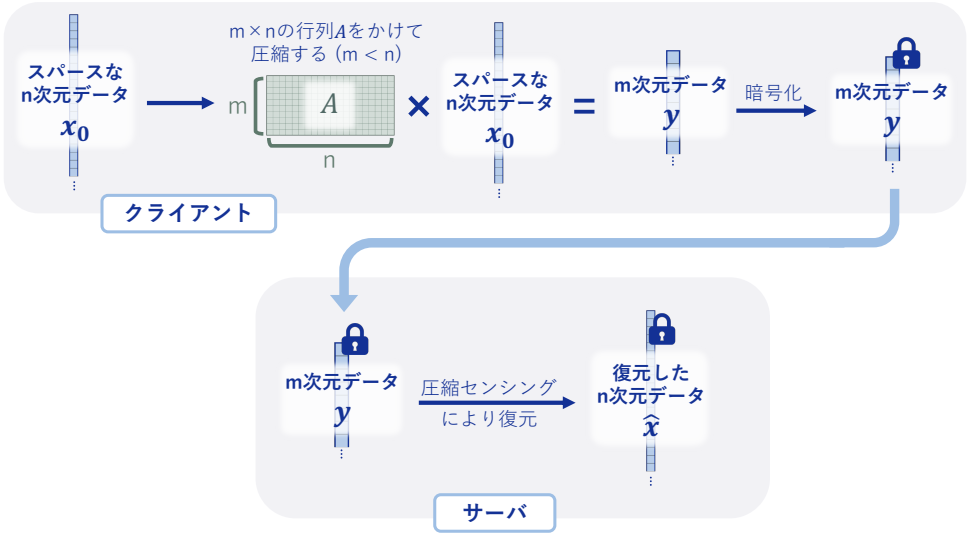

図 1　提案手法．クライアント側は圧縮センシングによる平文の圧縮と準同型暗号化を行い，サーバ側は圧縮センシングの復元を行う．圧縮センシングにより，Baseline に比べて，クライアントからサーバに送信する暗号文のサイズは n 次元から m 次元に減少する．したがって，クライアントとサーバ間の通信量の削減が可能となる．

加算と乗算の任意回数の演算を実現している．FHE は，1978 年に Rivest ら [4] によって構想され，2009 年に Craig Gentry ら [5] によって格子暗号を用いた実現手法が提案された．

SHE は LHE よりも演算にかかる時間は短いが乗算可能回数に制限があり，LHE は演算可能回数をあらかじめ設定できるが，演算可能回数の増加と演算にかかる時間は比例関係にあるという課題がある．FHE には暗号文同士の加算と乗算が任意回数可能であるが，暗号文のサイズが大きく，クライアントとサーバ間の通信量が大きくなってしまうといった課題がある [6]．本研究では LHE を使用する．

2.1.2　暗号化方式 [7] とライブラリ

現在，準同型暗号は，第 1 世代 イデアル格子による方式，第 2 世代 BGV [8] や BFV 方式 [9]，第 3 世代 GSW 方式や TFHE [10] [11], 第 4 世代 CKKS 方式 [12] の 4 つの世代に分類される．特に，CKKS 方式は整数だけではなく実数や複素数も計算することができるという特徴を持ち，機械学習に応用可能であることから近年注目されている．スケールというパラメータによって実数を整数に見立てて計算するため，計算結果に若干のノイズが生じ，計算を繰り返すとノイズが増加するため，リスケール処理を行う．

また，現在では多くの準同型暗号ライブラリが公開されており，有名なライブラリの 1 つとして，Microsoft SEAL [13] がある．SEAL は Microsoft が開発したライブラリで，BFV 方式と CKKS 方式の 2 つを利用でき，C++で実装されている．他にも，BGV 方式，C++で実装され，ブートストラップもサポートしている HElib [14] も有名である．

本研究では，実数を扱いたいため，CKKS 方式の SEAL を用いて提案手法を実装した．

2.2　圧縮センシング [15]

2.2.1　概　　要

圧縮センシング (compressed sensing) は，スパース性 (零成分が多いという性質) を持つ高次元の信号を少ない観測から復元する枠組みである．未知の n 次元ベクトル $\boldsymbol{x_0}$ が，m 行 n 列の行列 A を用いて以下のように線形変換されているとする．このとき，$m < n$ とする．

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} = \begin{bmatrix} a_{11} & \cdots & a_{1i} & \cdots & a_{1n} \\ \vdots & \ddots & & & \vdots \\ a_{i1} & & a_{ii} & & a_{2n} \\ \vdots & & & \ddots & \vdots \\ a_{m1} & \cdots & a_{mi} & \cdots & a_{mn} \end{bmatrix} \times \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

$$\therefore \boldsymbol{y} = A\boldsymbol{x_0} \tag{3}$$

元のベクトル $\boldsymbol{x_0}$ の零ではない要素が k 個あると仮定して，低次元な m 次元ベクトル $\boldsymbol{y}$ と圧縮行列 A から (3) を満たす $\boldsymbol{x_0}$ を求める．一般には，未知数の数が方程式の数より多いため不定となり，解を一意に定めることができない．しかし，$\boldsymbol{x_0}$ の要素の多くが零であったとすると，解を一意に推定できる．

2.2.2　再構成アルゴリズム - OMP

圧縮センシングに使用される再構成アルゴリズムは多数存在する．代表的なアルゴリズムには，例えば OMP(Orthogonal Matching Pursuit)，IHT(Iterative Hard Tresholding), IST(Iterative Soft Tresholding) などがある．再構成アルゴリズムを選定する際には，暗号化状態のまま演算が可能かを考慮した．暗号化状態のまま可能な演算は限られており，暗号化状態での演算が困難な例の 1 つとして比較演算が挙げられる．ここでは OMP のアルゴリズムを紹介する．また，

本研究では一般逆行列による L_2 ノルムの最小化によって圧縮前のデータを推定したが，詳細については 4.2 で説明する．

OMP とは貪欲法の一種であり，$\boldsymbol{y}$ を列ベクトル a_i の成分のみで近似する．残差を最も良く近似する順に A の列ベクトルの添字を添字集合 T に加えていくことで復元する．OMP の詳細は以下の通りである．

- 入力：観測信号 $\boldsymbol{y} \in \mathbb{R}^m$, 観測行列 $A \in \mathbb{R}^{m \times n}$
  (非零要素数 $k \in \mathbb{N}$)
- 初期化：推定値 $\hat{\boldsymbol{x}} = \boldsymbol{0}$, 添字集合 $T = \phi$

  終了条件を満たすまで以下を繰り返す.
  （1） 残差 $\boldsymbol{y} - A\hat{\boldsymbol{x}}$ と最も相関の高い
  (内積の絶対値の大きい)A の列ベクトルの
  添字を T に追加する.
  （2） $\hat{\boldsymbol{x}} = \underset{\hat{\boldsymbol{x}}}{\operatorname{argmin}} ||\boldsymbol{y} - A_T \boldsymbol{x_T}||_2^2$ を一般逆行列で
  評価することによって $|T|$ スパースな推定値 $\hat{x}$
  を求める (一般逆行列で評価する).
- 出力：推定値 $\hat{\boldsymbol{x}} \in \mathbb{R}^n$

「残差を最も良く近似する」というのは,「内積の絶対値が最も大きい」と言い換えられる．また，$\underset{\hat{\boldsymbol{x}}}{\operatorname{argmin}} ||\boldsymbol{y} - A_T \boldsymbol{x_T}||_2^2$ の解である $\hat{\boldsymbol{x}}$ は，$\hat{\boldsymbol{x}} = (A_T^\top A_T)^{-1} A_T^\top \boldsymbol{y}$，つまり，$\hat{\boldsymbol{x}} = (A_T$の一般逆行列) と $\boldsymbol{y}$ の積 で求められることが一般に知られている．

3 関 連 研 究

ここでは関連する研究として，圧縮とプライバシー保護を実現する multi-class privacy-preserving cloud computing scheme(以下 MPCC 方式) の提案，暗号化領域での圧縮アルゴリズムの実行の 2 つを挙げて説明する．

3.1 圧縮センシングに基づくマルチクラスプライバシー保護型クラウドコンピューティング

Kuldeep ら [16] は，センサデータを正確に取得できるスーパーユーザと，平均値や分散などの統計データのみを取得できる準権限ユーザの 2 クラスの秘匿性の実現を目的として，以下のような MPCC 方式を提案した．

（1） IoT デバイスは圧縮センシングにより継続的にデータをセンシングし，センシングしたデータをクラウドに送信，保存する．
（2） スーパーユーザにはマスターキーを，準権限ユーザには統計キーを持たせる．
（3） データを取得するために，ユーザはクラウドにクエリを送信する．
（4） クラウドがスパース信号を復元し，復元された暗号文をユーザに送信する．
（5） ユーザは自分のアクセス権に従って，暗号文に鍵を適用し，スーパーユーザは平文の復元結果を，準権限ユーザは統計キーで並べ替えられた平文の復元結果のみを得る．

MPCC 方式では，クラウドサービスプロバイダーに対するデータの機密性を損なうことなく，計算量の多いスパース信号の復元をクラウド上で行うことができる．

また，MPCC 方式に対して暗号文のみの攻撃を適用することは計算上不可能であることが証明されている．

3.2 暗号化領域での圧縮アルゴリズムの実行

Canard ら [17] は，FHE を用いて入力データが暗号化されている場合に圧縮アルゴリズムを実行することを目的として，最も基本的な圧縮アルゴリズムである Run-Length Encoding(以下 RLE) を分析し，圧縮アルゴリズムを実行する際に多くの問題が生じることを示した．

RLE とは，$\alpha_1, \alpha_2, ...$ のような記号列が与えられたとき，{ 回数，記号 } のペアからなる列を生成することで，記号列を圧縮するものである．RLE の例は (4) に示す．

$$0,0,2,3,3,4,0,0,0 \rightarrow \{2,0\},\{1,2\},\{2,3\},\{1,4\},\{3,0\} \quad (4)$$

RLE アルゴリズムは単純であるにも関わらず，そのままそれ自体を準同型暗号で実装することはできない．FHE の演算能力の特徴として以下が挙げられる．

- FHE は暗号化領域のデータに依存する条件を持つ ”if...then...else”構文を少なくとも直接実行できない．
- 暗号化領域のデータに依存する終了条件を持つループ構造を直接実行できない．

つまり，暗号化領域のデータに対して FHE はいわゆる静的制御構造プログラムしか実行できないといえる．しかし，動的制御構造プログラムを静的制御構造プログラムに変換するための正規化を行えば FHE でも実行できる．正規化の 1 つの例として (5),(6) を示す．

$$x := c?a : b \quad (5)$$
$$\equiv x$$
$$:= c \otimes a \oplus (1 \oplus c) \otimes b \quad (6)$$

”if...then...else”構文は (5) の条件代入演算子で表すことができ，値 x,a,b,c に関する情報を一切学習することなく，この構文の両方の分岐を実行し，(6) のように演算子を用いて条件値にしたがって結果を適切に再結合することができる．このように，アルゴリズム中に動的な部分が存在する場合にはアルゴリズム制御フローを正規化することが必要となる．また，この研究では，実際に FanVercauteren 暗号と Armadillo FHE コンパイラを用いた具体的な実験も行っている．

4 提案手法

4.1 概　　要

ここでは単純に，スパースな多次元データをクライアントで準同型暗号化して，圧縮せずにサーバに送信する方法を Baseline とする．本研究では，図 1 で示すような，スパースなデータを圧縮してから復元する「圧縮センシング」を利用して，通信量を削減してスパースな多次元データを送信することを提案する．ここで m, n は $m < n$ であるものとする．

クライアント

（1） スパースな n 次元データ $\boldsymbol{x_0}$ に，m 行 n 列の圧縮行列 A をかけて圧縮する
$\boldsymbol{y} = A\boldsymbol{x_0}$

（2） 得られた m 次元データ $\boldsymbol{y}$ を暗号化する
$Encrypt(\boldsymbol{y})$

（3） 暗号化状態の $\boldsymbol{y}$ をサーバに送信する

サーバ

（4） 暗号化状態の $\boldsymbol{y}$ を受け取る

（5） 再構成アルゴリズムにより復元した n 次元データ $\hat{\boldsymbol{x}}$ を得る
$\hat{\boldsymbol{x}} = Reconstruction(Encrypt(\boldsymbol{y}))$

Baseline に比べて，クライアントからサーバに送信する暗号文のサイズは n 次元から m 次元に減少するため，クライアントとサーバ間の通信量の削減が可能となる．

4.2 提案手法で使用した再構成アルゴリズム

代表的な再構成アルゴリズムの 1 つである OMP では，2.2.2 でも述べたように，残差を最も良く近似する，つまり，残差と最も内積の絶対値が大きい A の列ベクトルがどれかを見つける際に比較演算が必要となる．しかし，準同型暗号での比較演算は非常に難しい．なぜなら，暗号文どうしの比較が簡単に可能であるとすると挟み込みなどによって値の特定ができてしまうからである．そのため提案手法では，圧縮センシングの再構成アルゴリズム $Reconstruction(\cdot)$ において，一般逆行列によって最小ノルム解，つまり L_2 ノルムが最小となる推定結果を得る．すなわち $encrypt(\hat{\boldsymbol{x}}) = (A^\top A)^{-1}A^\top \times encrypt(\boldsymbol{y})$ のように一般逆行列と暗号化された低次元行列との積を求める．

5 実　　験

5.1 概　　要

実験ではクライアントには Raspberry Pi，サーバには Ubuntu のマシンを使用し，Baseline と提案手法の 2 つを Microsoft SEAL ライブラリを用いて C++で実装した．Baseline，提案手法ともに乱数により生成されたスパースな多次元データを利用し，スパースな多次元データは 0,1 の 2 値で構成されているものとした．提案手法では，スパースな多次元データを圧縮センシングにより圧縮してから準同型暗号化した暗号文をクライアントからサーバに送信し，サーバで圧縮センシングの再構成アルゴリズムにより復元をするものとした．Baseline では，スパースな多次元データをクライアントで準同型暗号化して，圧縮せずにクライアントからサーバに送信するものとした．これらの Baseline と提案手法で，クライアントからサーバに送信する準同型暗号文のサイズ比較，圧縮センシングの再構成アルゴリズムによるサーバでの復元結果の精度測定，クライアントとサーバでの実行時間の測定を行った．精度の評価方法には RMSE を用いた．また，スパースな多次元データを生成する際の乱数には，メルセンヌ・ツイスタ [18] を使用した．

表 1　マシン性能

	Raspberry Pi	サーバ
OS	Raspbian 11	Ubuntu 22.04.1
CPU	ARM Cortex-A72	Intel(R) Xeon(R) Gold 5115 CPU @ 2.40GHz
コア数	4	10
プロセッサ速度	600MHz	1GHz
RAM	4GB	192GB

5.2 実 験 環 境

実験に使用したマシンの性能を表 1 に示す．

5.3 パラメータ

5.3.1 準同型暗号に関するパラメータ [19]

実験で使用した SEAL の CKKS 方式のパラメータを表 2 に示す．

暗号文の長さ $slot\_count$ は，使用する多項式の次元を示す $poly\_modulus\_degree$ に 0.5 をかけた値となる．ここでは，$poly\_modulus\_degree = 8192$ としたため，$slot\_count = 4096$ となる．また，暗号文同士の乗算可能回数を示す Leveled や復号時の精度，復号時の実数値の精度は，チェインの設定によって決まる．実験ではチェインは次のように設定した．

$$modulus\_chain = \{60, 40, 40, 60\} \tag{7}$$

(7) は最初の数値 (以下 プライマリビット) である 60，最後の数値 (以下 ラストビット) である 60, それ以外の (以下 スケーリングビット)2 つの 40 に分けることができる．

Leveled はスケーリングビットの個数に依存するため，$Leveled = 2$ となり，2 回乗算可能となる．また，プライマリビットは復号時の精度と対応しており，復号時の実数値の整数部分の精度はプライマリビットとスケーリングビットの差と，復号時の実数値の小数部分の精度はスケーリングビットと復号時の実数値の整数部分の精度ビットの差と対応している．

表 2　SEAL の CKKS 方式に関するパラメータ

	値
暗号文の長さ $slot\_count$	4096
セキュリティ	128 bit
Leveled	2
復号時の精度	60 bit
復号時の実数値の精度	整数部分　20 bit
	小数部分　20 bit

5.3.2　圧縮センシングに関するパラメータ

実験の圧縮センシングには，(n, m)=$(1024, 512)$,$(1024, 256)$, $(1024, 128)$ のように圧縮率を変化させたものと，(n, m)=$(1024, 512)$,$(256, 128)$,$(128, 64)$ のように n, m の大小を変化させたものを用いた．実験では要素の 95 %が 0 であると仮定し，乱数により元のベクトル $\boldsymbol{x_0}$ を生成した．

また，表 2 で示したとおり，長さが 4096 の暗号文を使用するため，4096 の要素すべてが埋められた状態でクライアントからサーバに渡す必要がある．そのため，$slot\_count \div m$ 個のスパースな n 次元データ $\boldsymbol{x_0}$ を用意し，圧縮後のベクトル $\boldsymbol{y}$ を $slot\_count \div m$ 個を詰めた 1 つのベクトルを渡す形としている．

5.4　評価方法

5.1 で述べたように，Baseline と提案手法での圧縮センシングの再構成アルゴリズムによるサーバでの復元結果の精度の評価方法には RMSE を用いた．RMSE は以下の式で算出される．

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=0}^{n-1}(y_i - \hat{y}_i)^2} \tag{8}$$

式 (8) 中の y_i は実際の値，$\hat{y}_i$ は予測値，n はデータの総数を表す．RMSE の値が小さいほど誤差の小さいモデルであるといえる．

5.5　実験結果

5.5.1　通　信　量

図 2 にクライアントとサーバ間の通信量，つまり，クライアントからサーバに送信する準同型暗号文のサイズを示す．Baseline に比べ，提案手法では約半分の通信量に抑えられていることが分かる．これは n と m の比率に対応していることが分かる．

5.5.2　精　　度

スパース値 k と RMSE の関係性を示した図 3 と，圧縮率と RMSE の関係性を示した図 4 にあるように，スパース値 k が小さいと精度が高まるという性質を持つ．要素の 95 %が 0 であると仮定した時の提案手法における RMSE を測定した結果を示す．圧縮率 50 % $(n = 1024, m = 512)$ としたときの 8 個の $\hat{\boldsymbol{x}}$ の RMSE は，おおよそ 0.15 となった．

また，同様のパラメータにおいて，図 5 に元のベクトル $\boldsymbol{x_0}$ と再構成アルゴリズムにより復元した n 次元データ $\hat{\boldsymbol{x}}$ の比較を，図 6 に閾値により 0,1 の 2 値化を行った結果を示す．

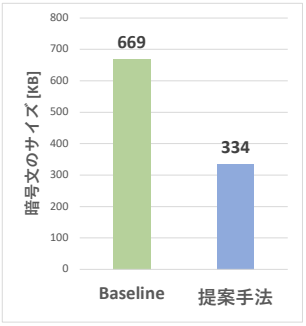

図 2　圧縮率 50 %の場合のクライアントとサーバ間の通信量．
$n = 1024, m = 512$ とした．縦軸は暗号文のサイズである．
提案手法では Baseline に比べて約半分の通信量に抑えられており，通信量の削減比率は n と m の比率に対応している．

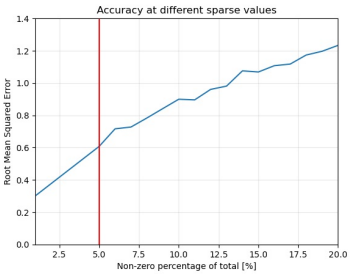

図 3　スパース値と RMSE

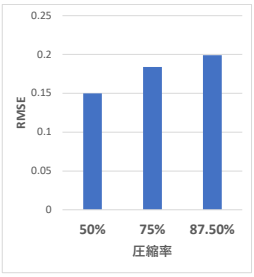

図 4　圧縮率と RMSE

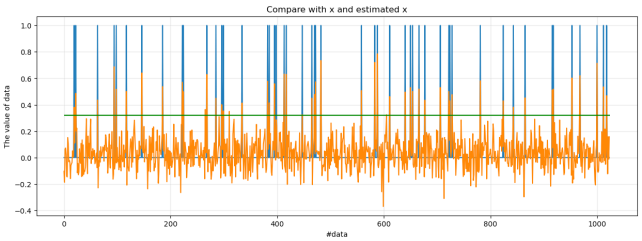

図 5　元のベクトル $\boldsymbol{x_0}$ と復元後のベクトル $\hat{\boldsymbol{x}}$ の比較．
青が元のベクトル $\boldsymbol{x_0}$，オレンジが復元後のベクトル $\hat{\boldsymbol{x}}$ を表す．
緑は今回設定した閾値である．

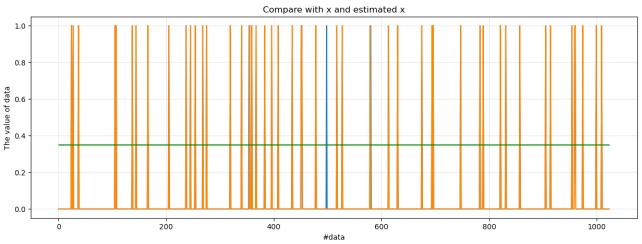

図 6　図 5 での復元後のベクトル $\hat{\boldsymbol{x}}$ を閾値により 2 値化した結果．
青が元のベクトル $\boldsymbol{x_0}$，オレンジが復元後のベクトル $\hat{\boldsymbol{x}}$ を 2 値化したものである．緑は今回設定した閾値である．

図 6 は閾値 0.32 として 2 値化している．このとき，1 である部分は 91%一致していた (5 回の平均値)．

5.5.3　実行時間

Baseline と提案手法における実行時間を図 7 に示す．提案手法では，クライアントからサーバに送信する暗号文のサイズの

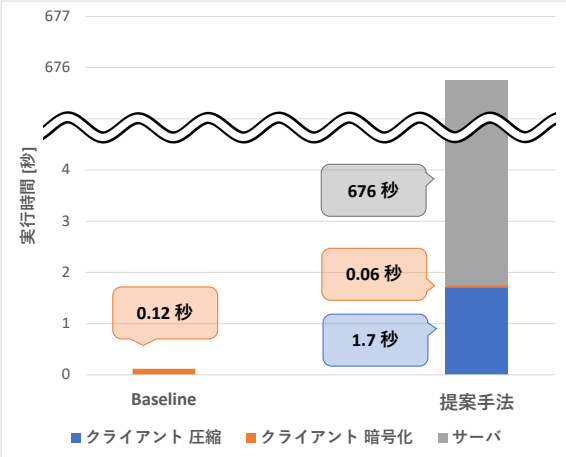

図 7　圧縮率 50 %のときの実行時間.$n = 1024, m = 512$ とした.
クライアントからサーバに送信する暗号文のサイズの減少に伴い暗号化にかかる時間は減っている.
しかし, クライアントで圧縮させる際の行列計算や, 特にサーバで復元する際の再構成アルゴリズムの計算に時間がかかっている.

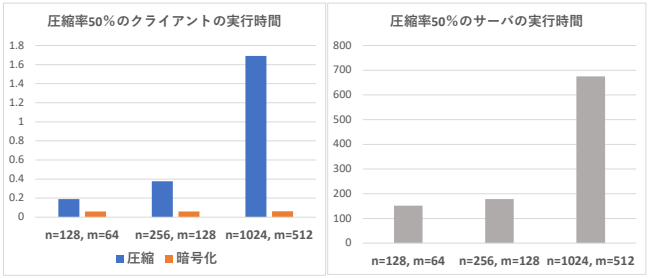

図 8　圧縮率 50%時の実行時間.
n, m の値が小さいほどクライアントでの圧縮時間やサーバでの復元時間が短くなっている.

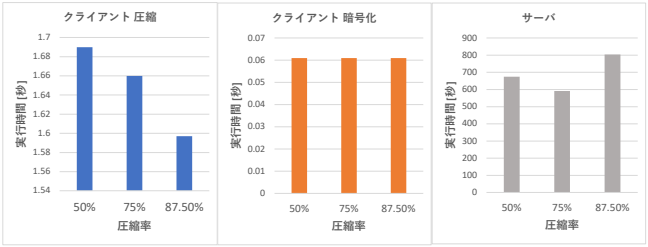

図 9　圧縮率を変化させたときの実行時間の変化

減少に伴い，暗号化にかかる時間は減っているが，クライアントで圧縮させる際の行列計算や，特にサーバで復元する際の再構成アルゴリズムの計算に時間がかかってしまっていることが分かる．

5.6　パラメータの値を変更した場合の実験結果に対する考察

パラメータの値 n, m を変更して，圧縮率 50 %の場合のクライアントの実行時間の変化を図 8 に，圧縮率を変化させたときの実行時間の変化を図 9 と図 10 に示す．実行時間，平均の RMSE 値に関して考察する．

まず実行時間について見てみると，n, m の値が小さいほどクライアントでの圧縮時間やサーバでの復元時間が短いことが

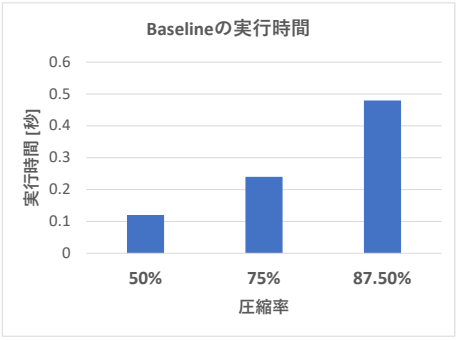

図 10　圧縮率を変化させたときの Baseline の実行時間の変化.
n と m の比率と比例している.

読み取れる．これは圧縮行列 A が $n \times m$ であり，n, m の大きさと行列計算の時間には正の関係があるからであると考えられる．したがって，長いベクトルを復元するよりも短いベクトルを複数復元する方が高速であり適しているといえる．また，Baseline の実行時間の変化について，n と m の比率と比例している．これは，Baseline で暗号化する際に，長さ n のスパースなデータを $n \div m$ 個の長さ $slot\_count$ のベクトルにしているからである．

平均の RMSE 値に関して，図 4 でも示したように，やはり圧縮率を高くするほど RMSE 値が大きくなって精度が低くなってしまっていることが読み取れる．したがって圧縮率と精度はトレードオフの関係にあることがわかる．

6　まとめと今後の課題

IoT デバイスの普及により，クラウドなどのサーバに個人に関する情報を含むデータが蓄積され，データ分析に利用されるようになった．外部からの攻撃などによるデータの漏洩に備えるために，暗号文同士の加算や乗算が可能な準同型暗号が用いることが有効である．しかし，準同型暗号の問題点として，暗号文のサイズが大きく通信量が大きくなってしまうことが挙げられる．

本研究では，圧縮センシングを利用して，通信量を削減してスパースな多次元データを送信することを提案した．圧縮センシングは，スパース性を利用することにより，少ないサンプル数からもとのデータを復元する技術である．

実験の結果，圧縮率 50%のとき，クライアントとサーバ間の通信量を約半分に減らすことができた．復元後のベクトル $\hat{\boldsymbol{x}}$ の RMSE はおよそ 0.15 であり，閾値 0.32 として 2 値化すると 1 である部分は 91%一致した．

今後は，スパースな多次元データの構成を実数にも対応させることや，他の再構成アルゴリズムの使用により，より精度を高めること，実行時間の改善を検討している．

文　献

[1] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *International conference on the theory and applications of cryptographic techniques*, pp. 223–238. Springer, 1999.

[2] Ronald L Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, Vol. 21, No. 2, pp. 120–126, 1978.

[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, Michael L Dertouzos, et al. On data banks and privacy homomorphisms. *Foundations of secure computation*, Vol. 4, No. 11, pp. 169–180, 1978.

[5] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pp. 169–178, 2009.

[6] 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 F-002, 2016.

[7] 準同型暗号の世代区分. `https://www.eaglys.co.jp/news/column/homomorphicencryption/`.

[8] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory (TOCT)*, Vol. 6, No. 3, pp. 1–36, 2014.

[9] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical gapsvp. In *Advances in Cryptology–CRYPTO 2012: 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, pp. 868–886. Springer, 2012.

[10] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Tfhe: fast fully homomorphic encryption over the torus. *Journal of Cryptology*, Vol. 33, No. 1, pp. 34–91, 2020.

[11] TFHE. `https://tfhe.github.io/tfhe/`.

[12] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. In *International conference on the theory and application of cryptology and information security*, pp. 409–437. Springer, 2017.

[13] Microsoft SEAL. `https://github.com/microsoft/SEAL`.

[14] HElib. `https://github.com/homenc/HElib`.

[15] 圧縮センシング: 疎情報の再構成とそのアルゴリズム. `https://www.kurims.kyoto-u.ac.jp/~kyodo/kokyuroku/contents/pdf/1803-03.pdf`.

[16] Gajraj Kuldeep and Qi Zhang. Compressive sensing based multi-class privacy-preserving cloud computing. In *GLOBECOM 2020-2020 IEEE Global Communications Conference*, pp. 1–6. IEEE, 2020.

[17] Sebastien Canard, Sergiu Carpov, Donald Nokam Kuate, and Renaud Sirdey. Running compression algorithms in the encrypted domain: a case-study on the homomorphic execution of rle. In *2017 15th Annual Conference on Privacy, Security and Trust (PST)*, pp. 283–28309. IEEE, 2017.

[18] Makoto Matsumoto and Takuji Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, Vol. 8, No. 1, pp. 3–30, 1998.

[19] 格子暗号の CKKS 形式のパラメータ、精度ビットなどの解説（SEAL ライブラリ）. `https://qiita.com/kenmaro/items/931835476ca8781de9f9`.