秘匿計算上の集約関数中央値計算アルゴリズム

濱田 浩気 †　　五十嵐 大 †　　千田 浩司 †

† 日本電信電話株式会社
180-8585 東京都武蔵野市緑町 3-9-11
{hamada.koki,ikarashi.dai,chida.koji}@lab.ntt.co.jp

あらまし 本稿では秘匿計算による安全なデータベースの実現を目指し，秘匿計算で中央値計算の集約関数を実現するアルゴリズムを提案する．集約関数は表として与えられたデータを特定の列でグループ分けし，グループごとの集計値を計算する演算の総称である．総和や集計，最大値，最小値の計算に対しては秘匿計算で集約関数を実現する手法が提案されているが，実用上重要な統計値である中央値の計算には適用することができなかった．提案手法は中央値計算の集約関数をデータ数 n に対して $O(\log n)$ ラウンド，比較プロトコル換算で $O(n \log n)$ 回の通信量で実現する．さらに，中央値計算を一般化し，四分位数を含む分位数の計算の集約関数を実現するアルゴリズムも提案する．

An Algorithm for Computing Aggregate Median on Secure Function Evaluation

Koki Hamada†　　Dai Ikarashi†　　Koji Chida†

†NTT Corporation
3-9-11, Midori-cho Musashino-shi, Tokyo 180-8585 Japan
{hamada.koki,ikarashi.dai,chida.koji}@lab.ntt.co.jp

Abstract We propose an algorithm that computes aggregate median on secure function evaluation. Aggregate operations are a set of operations that compute certain function for each set of values grouped by given attributes. Although algorithms that realize aggregate operations for computing sum, count, max and min are proposed, algorithm for aggregate median is not known despite its practical importance. Our algorithm computes aggregate median in $O(\log n)$ rounds and $O(n \log n)$ comparisons where n is the number of input values. In addition, we generalize our algorithm so that we can compute quantile including median and quartiles.

1　はじめに

近年，個人に関する様々な情報を容易に取得できる環境が整い，データ分析技術の進歩と相まって，個人に関する情報の活用への期待が高まっている．その一方で，個人情報保護やプライバシの観点から個人に関する情報は極めて慎重な取扱いが必要とされ，データの保護と活用をどう両立させるかが問題となっている．

この問題の解決のため，我々は秘匿計算と呼ばれる技術を用いて，出力以外の情報を一切漏らさない安全なデータベースの実現を目指している．秘匿計算は Yao により提案された技術であり，データを秘匿したまま明かすことなく計算する [13]．しかし，秘匿計算は通常の計算機上の計算に比べて処理効率が低下してしまうという問題がある．

処理速度の低下の原因は大きく二つある．一つは，基本演算のオーバーヘッドである．秘匿計算ではデータの秘匿性を保つため，乗算のような通常の計算機では一命令で実行可能な基本演算にも大きな処理時間を要する．これに対し

表 1: 入力データの例

出身地	年齢	身長
兵庫	28	170
兵庫	23	180
神奈川	31	190
東京	20	190
兵庫	45	170
神奈川	25	180

表 2: 出身地でグループ分けされたデータ

出身地	年齢	身長
東京	20	190
神奈川	31	190
神奈川	25	180
兵庫	28	170
兵庫	23	180
兵庫	45	170

ては近年改良が進んでおり，効率のよい基本演算を備えたフレームワークの提案，実装が行われている．

もう一つは，計算量の悪化である．Goldreich らや Ben-Or らにより，秘匿計算では基本演算を組み合わせて回路を作ることによって，任意の関数をデータを秘匿したまま計算できることが示されている [7, 2]．しかし，回路に基づいた一般的な構成方法を用いると，実用上重要なアルゴリズムの多くで計算量が大きくなってしまう．基本演算の効率化は定数倍の改善に過ぎず，多量のデータを扱うためには，計算量の悪化の解決が不可欠である．このため，特定の処理ごとに秘匿計算上の効率的なアルゴリズムを設計する研究が進んでおり，Aggarwal らによる k 番目の要素の選択 [1] や Damgård らによるビット分解 [5]，Nishide らによる比較 [11]，Ning らによる剰余計算 [10]，Goodrich によるソート [8] などが提案されている．

さらに，これらのアルゴリズムの組み合わせにより多くの処理が秘匿計算で効率よく実現されてきており，濱田らによる表の結合 [15] などの複雑なデータベース処理も現実的な時間で実現されている．

1.1 集約関数演算

データベース処理では多くの場合，データを表 1 のような表形式で扱う．データはいくつかの属性値 (表 1 では出身地，年齢，身長の各値) の組からなるレコード (表 1 の各行) の集合からなる．表形式のデータの分析でよく行われる処理の一つに**集約関数演算**と呼ばれる処理がある．集約関数演算は表形式のデータを特定の属性 (表 1 では出身地) についてグループ分けし，

表 3: 出身地による集約関数の計算例

(a) 年齢の平均値

出身地	年齢
東京	20
神奈川	28
兵庫	32

(b) 身長の中央値

出身地	身長
東京	190
神奈川	180
兵庫	170

指定された集計値をグループごとに計算する演算の総称であり，SQL では GROUP BY 句として扱われる．

表 1 に対して出身地により年齢の平均値の集約関数の計算を行った結果を表 3(a) に例示する．表 1 を出身地でグループ分けすると表 2 のようになり，例えば出身地が兵庫であるレコードは $(\text{兵庫}, 28, 170)$，$(\text{兵庫}, 23, 180)$，$(\text{兵庫}, 45, 170)$ の 3 つである．これら 3 つの中での年齢の平均値は $(28+23+45)/3 = 32$ であるので，出力の表 3(a) には $(\text{兵庫}, 32)$ のレコードが含まれる．同様に出身地ごとに中央値を計算した結果を表 3(b) に示す．

五十嵐らにより総和や集計，最大値，最小値の集約関数を秘匿計算で実現する手法が提案されている [16]．この手法は総和や集計，最大値，最小値の計算が二項演算の反復で書けることを利用している．

1.2 本研究の貢献

本稿では，中央値の集約演算を秘匿計算で実現するアルゴリズムと，中央値の一般化である q 分位数の集約演算を秘匿計算で実現するアルゴリズムの提案を行う．

中央値は重要な統計値の一つであり，大まかにいえば集合の真ん中の点の値である．中央値の詳細な定義はいくつか存在するが，本稿では大きさ n の集合の $\lfloor (n+1)/2 \rfloor$ 番目の値を中央値と呼ぶ (詳細は 2.2 節で述べる)．中央値は二項演算の反復で記述することは自明ではなく，五十嵐らの手法 [16] では求めることができなかった．本稿ではまず，中央値の集約関数を計算するアルゴリズムを提案する．さらに，中央値を求めるアルゴリズムを一般化し，$[1, n]$ を $a : b$ に内分する点である $\frac{a}{a+b}$ 分位数 ($\lceil \frac{a}{a+b} n \rceil$ 番目の値) の集約関数を計算するアルゴリズムを提案する．これらのアルゴリズムはいずれも $O(\log n)$ ラウンドで，比較プロトコル換算で $O(n \log n)$ 回の通信量である．

2 準備

本稿で提案するアルゴリズムは，秘匿計算上の演算の組み合わせにより構築される．提案アルゴリズムが必要とする秘匿計算上の演算は，加算，減算，乗算，等号判定，比較，安定ソートである．これらを全て提供する秘匿計算を実現するためには，例えば，Ben-Or らによる加算，減算，乗算を提供する秘匿計算 [2] 上で Nishide らによる等号判定と比較 [11]，濱田らが提案した安定ソート [14] を用いればよい．

2.1 記法，定義

本稿のアルゴリズムで扱う値はすべて有限環 $\mathbb{Z}_N$ 上の値とする．ベクトル $\boldsymbol{a}$ の第 i 要素を a_i で参照する． $a \in \mathbb{Z}_N$ を暗号化や秘密分散などの手段で秘匿化した値を a の秘匿文と呼び，$[\![a]\!]$ と表記する．また，a を $[\![a]\!]$ の平文と呼ぶ．ベクトル $\boldsymbol{v}$ の各要素を秘匿化したベクトルを $[\![\boldsymbol{v}]\!]$ と表記する．

2.2 中央値，分位数

中央値は大まかにいえば集合の真ん中の点の値であり，集合の大きさ n が奇数の場合には $(n+1)/2$ 番目の要素の値である．n が偶数の場合には $n/2$ 番目と $n/2+1$ 番目に真ん中の点があるため，いくつか定義がある．本稿では Cormen らによる定義 [3] を用い，n の偶奇に関わらず $\lfloor (n+1)/2 \rfloor$ 番目を小さい方の中央値，$\lceil (n+1)/2 \rceil$ 番目を大きい方の中央値と呼び，単に中央値と書いた場合は小さい方の中央値のこととする．

集合を $p : 1-p$ $(0 < p \leq 1)$ に内分する点の値を p 分位数と呼ぶ．本稿では Hyndman らによる定義 [9] を用い，$\lceil pn \rceil$ 番目の点の値を p 分位数と呼ぶ．

2.3 計算効率の尺度

本稿で提案するアルゴリズムの効率はサブルーチンとして使用する各演算の効率に依存して決まる．本稿では秘匿計算がマルチパーティプロトコルとして構成されるものとして計算効率の評価を行う．マルチパーティプロトコルは複数のパーティ間で通信を行いながら協調計算を行う方式である．一般的な構成では，各パーティが単独で行うローカルの計算に比べて通信に要する時間が著しく大きい．このため，ローカルの計算は無視できるものとみなし，通信したデータの量（通信量）と一度に送ることのできるデータ量に制限がない場合に要する通信回数（ラウンド数）の 2 つの尺度で計算効率の評価を行う．本稿では通信量は比較プロトコルを単位として評価する．

2.4 既存の秘匿計算上の演算

本稿で提案するアルゴリズムは既存の秘匿計算上の演算の組み合わせにより構成する．本稿で用いる各演算は以下で述べる機能を安全に実現し，出力以外の一切の情報を漏らさないものとする．

2.4.1 加算，減算，乗算

加算，減算，乗算の各演算は 2 つの値 $a, b \in \mathbb{Z}_N$ の秘匿文 $[\![a]\!], [\![b]\!]$ を入力とし，それぞれ $a+b$, $a-b$, ab の計算結果 c_1，c_2，c_3 の秘匿文 $[\![c_1]\!]$, $[\![c_2]\!]$, $[\![c_3]\!]$ を計算する．これらの演算の実行をそれぞれ

$$[\![c_1]\!] \leftarrow \mathsf{Add}([\![a]\!], [\![b]\!]),$$
$$[\![c_2]\!] \leftarrow \mathsf{Sub}([\![a]\!], [\![b]\!]),$$
$$[\![c_3]\!] \leftarrow \mathsf{Mul}([\![a]\!], [\![b]\!])$$

と記述する．誤解を招く恐れのない場合には，$\mathsf{Add}([\![a]\!],[\![b]\!])$，$\mathsf{Sub}([\![a]\!],[\![b]\!])$，$\mathsf{Mul}([\![a]\!],[\![b]\!])$ をそれぞれ $[\![a]\!]+[\![b]\!]$，$[\![a]\!]-[\![b]\!]$，$[\![a]\!]\times[\![b]\!]$ と略記する．

2.4.2 論理演算

論理和，論理積，否定の各演算は 2 つの値 $a,b\in\{0,1\}$ の秘匿文 $[\![a]\!],[\![b]\!]$ を入力とし，a と b の論理和，論理積，a の否定の計算結果 c_1，c_2，c_3 の秘匿文 $[\![c_1]\!]$，$[\![c_2]\!]$，$[\![c_3]\!]$ を計算する．これらの演算の実行をそれぞれ

$$[\![c_1]\!]\leftarrow[\![a]\!]\vee[\![b]\!],$$
$$[\![c_2]\!]\leftarrow[\![a]\!]\wedge[\![b]\!],$$
$$[\![c_3]\!]\leftarrow\neg[\![a]\!]$$

と記述する．

2.4.3 比較，等号判定

比較，等号判定の各演算は 2 つの値 $a,b\in\mathbb{Z}_N$ の秘匿文 $[\![a]\!],[\![b]\!]$ を入力とし，それぞれ $a<b$，$a=b$ の真偽値 c_1，c_2 の秘匿文 $[\![c_1]\!]$，$[\![c_2]\!]$ を計算する．真偽値は真のとき 1，偽のとき 0 とする．これらの演算の実行をそれぞれ

$$[\![c_1]\!]\leftarrow([\![a]\!]\overset{?}{<}[\![b]\!]),$$
$$[\![c_2]\!]\leftarrow([\![a]\!]\overset{?}{=}[\![b]\!])$$

と記述する．

2.4.4 安定ソート

ソートはベクトルの要素を昇順に並べ替えたベクトルを計算する演算である．特に安定ソートは，ソート演算で同じ値が存在した場合に同じ値の要素同士の順序を保存する．形式的には，大きさ n のベクトル $\boldsymbol{k}\in\mathbb{Z}_N^n$ と ℓ 個（$1\le\ell$）の大きさ n のベクトル $\boldsymbol{a}^{(1)},\dots,\boldsymbol{a}^{(\ell)}\in\mathbb{Z}_N^n$ の秘匿文 $[\![\boldsymbol{k}]\!],[\![\boldsymbol{a}^{(1)}]\!],\dots,[\![\boldsymbol{a}^{(\ell)}]\!]$ を入力とし，$\pi_s(i)<\pi_s(j)$ である任意の $i,j\in\{1,\dots,n\}$ に対して $k_i<k_j\vee(k_i=k_j\wedge i<j)$ を満たす全単射 $\pi_s:\{1,\dots,n\}\to\{1,\dots,n\}$ について $a_j^{(i)}=b_{\pi_s(j)}^{(i)}$（$1\le i\le\ell,1\le j\le n$）を満たすベクトル $\boldsymbol{b}^{(1)},\dots,\boldsymbol{b}^{(\ell)}\in\mathbb{Z}_N^n$ の秘匿文 $[\![\boldsymbol{b}^{(1)}]\!],\dots,[\![\boldsymbol{b}^{(\ell)}]\!]$ を計算する．この演算の実行を

$$[\![\boldsymbol{b}^{(1)}]\!],\dots,[\![\boldsymbol{b}^{(\ell)}]\!]\leftarrow\mathsf{StableSort}([\![\boldsymbol{a}^{(1)}]\!],\dots,[\![\boldsymbol{a}^{(\ell)}]\!];[\![\boldsymbol{k}]\!])$$

と記述する．

表 4: 入出力の平文の例

(a) 整列済みの入力

出身地	身長
神奈川	180
神奈川	190
東京	190
兵庫	170
兵庫	170
兵庫	180

(b) 階段値と出力

階段+	階段-	出力
0	1	1
1	0	0
0	0	1
0	2	0
1	1	1
2	0	0

3 提案手法

3.1 問題の概要

本稿で解く問題は要素数 n のベクトル $\boldsymbol{k},\boldsymbol{v}\in\mathbb{Z}_N^n$ の秘匿文 $[\![\boldsymbol{k}]\!],[\![\boldsymbol{v}]\!]$ を入力とし，要素数 n のベクトル $\boldsymbol{f}\in\mathbb{Z}_N^n$ の秘匿文 $[\![\boldsymbol{f}]\!]$ を出力する問題である．$\boldsymbol{k}$ と $\boldsymbol{v}$ はあらかじめ対応する各要素の対 (k_i,v_i) について整列済みのキーと値の列である．すなわち，$\boldsymbol{k}$ の同一の値を持つ要素はすべて連続しており，同一の値を持つ $k_{i_g+1},k_{i_g+2},\dots,k_{i_g+n_g}$ に対して $v_{i_g+1}\le v_{i_g+2}\le\cdots\le v_{i_g+n_g}$ である（$\boldsymbol{k}$，$\boldsymbol{v}$ の例をそれぞれ表 4(a) の出身地，身長に示す）．これらの要素からなるレコードをグループと呼ぶ．出力 $\boldsymbol{f}$ の要素 f_i は k_i を含むレコードがグループ内の中央値ならば 1，そうでなければ 0 とする（$\boldsymbol{f}$ の例を表 4(b) の出力に示す）．

入力の $\boldsymbol{k}$，$\boldsymbol{v}$ が整列済みでない場合は，

1: $[\![\boldsymbol{k}']\!],[\![\boldsymbol{v}']\!]\leftarrow\mathsf{StableSort}([\![\boldsymbol{k}]\!],[\![\boldsymbol{v}]\!];[\![\boldsymbol{v}]\!])$
2: $[\![\boldsymbol{k}'']\!],[\![\boldsymbol{v}'']\!]\leftarrow\mathsf{StableSort}([\![\boldsymbol{k}']\!],[\![\boldsymbol{v}']\!];[\![\boldsymbol{k}']\!])$

を実行した出力 $[\![\boldsymbol{k}'']\!],[\![\boldsymbol{v}'']\!]$ を入力とすればよい．

3.2 集約関数中央値計算アルゴリズム

提案する集約関数中央値計算のアルゴリズムを説明する．基本的なアイディアは，各グルー

Algorithm 1 階段+の計算

Notation: Sawtooth-Inc($[\![\boldsymbol{k}]\!]$)
Input: 大きさ n のソート済みのキー $[\![\boldsymbol{k}]\!]$
Output: i 番目の要素がグループ内で上から j 番目のとき $c_i = j-1$ である $[\![\boldsymbol{c}]\!]$

1: **for** $i = 1$ **to** $n-1$ **do in parallel**
2: $\quad [\![e_i]\!] := ([\![k_i]\!] \stackrel{?}{=} [\![k_{i+1}]\!])$
3: $[\![b_i^{(0)}]\!] := \begin{cases} [\![0]\!] & \text{if } i = 1 \\ [\![e_{i-1}]\!] & \text{otherwise} \end{cases}$
4: $[\![a_i^{(0)}]\!] := [\![b_i^{(0)}]\!]$
5: $t := \lceil \log_2 n \rceil$
6: **for** $j = 0$ **to** $t-1$ **do**
7: $\quad$ **for** $i = 0$ **to** n **do in parallel**
8: $\quad\quad [\![b_i^{(j+1)}]\!] := \begin{cases} [\![b_i^{(j)}]\!] \times [\![b_{i-2^j}^{(j)}]\!] & \text{if } 2^j < i, \\ [\![b_i^{(j)}]\!] & \text{otherwise,} \end{cases}$
$\quad\quad [\![a_i^{(j+1)}]\!] := \begin{cases} [\![a_i^{(j)}]\!] + [\![b_i^{(j)}]\!] \times [\![a_{i-2^j}^{(j)}]\!] & \text{if } 2^j < i, \\ [\![a_i^{(j)}]\!] & \text{otherwise.} \end{cases}$
9: **return** $[\![\boldsymbol{a}^{(t)}]\!]$

プ内の上下端からの距離が等しい点を検出することにより中央値を得ることである．

まず，グループごとにグループ内の上から i 番目の各レコードに対して $i-1$，$n_g - i$ を計算する．ここで n_g はグループの要素数である．これにより $0, 1, \ldots, n_g - 1$ と $n_g - 1, n_g - 1, \ldots, 0$ の 2 つの n 個の列が得られる．例を表 4(b) の階段+と階段-の属性に示す．この計算を階段計算と呼び，それぞれ Algorithm 1，Algorithm 2 に示す．

階段計算により，各グループごとに計算した $0, 1, \ldots, n_g - 1$ と $n_g - 1, n_g - 1, \ldots, 0$ の 2 つの n_g 個の列はそれぞれ上端，下端からの距離を表している．中央値を求めるためには上下端から等しい距離の点を求めればよい．すなわち，2 つの値が等しい点を検出すればよい．これは要素数 n_g が奇数の時は正しい．n_g が偶数の場合は，等しい点は存在せず，差が 1 の点が連続してちょうど 2 回出現する．これらは小さい中央値と大きい中央値に合致する．すなわち，グループ内の i 番目の点の階段+，階段-をそれぞれ c_i，d_i とすると，

$$c_i = d_i \vee c_i + 1 = d_i$$

が成り立つ点を求めれば中央値である．以上の内容を書き下したアルゴリズムを Algorithm 3 に示す．

Algorithm 3 集約関数中央値計算

Notation: Lower-Median($[\![\boldsymbol{k}]\!]$)
Input: 大きさ n のソート済みのキー $[\![\boldsymbol{k}]\!]$
Output: i 番目の点が小さい方の中央値なら $f_i = 1$，そうでなければ $f_i = 0$ である $[\![\boldsymbol{f}]\!]$

1: $[\![\boldsymbol{c}]\!] \leftarrow$ Sawtooth-Inc($[\![\boldsymbol{k}]\!]$)
$\quad [\![\boldsymbol{d}]\!] \leftarrow$ Sawtooth-Dec($[\![\boldsymbol{k}]\!]$)
2: **for** $i \leftarrow 1$ **to** n **do in parallel**
3: $\quad [\![t_i]\!] \leftarrow ([\![c_i]\!] \stackrel{?}{=} [\![d_i]\!])$,
$\quad\quad [\![u_i]\!] \leftarrow ([\![c_i]\!] + 1 \stackrel{?}{=} [\![d_i]\!])$.
4: $\quad [\![f_i]\!] \leftarrow [\![t_i]\!] \vee [\![u_i]\!]$.
5: **return** $[\![\boldsymbol{f}]\!]$

3.3 集約関数 $\frac{a}{a+b}$ 分位数計算アルゴリズム

次に，集合を $a : b$ に内分する点である $\frac{a}{a+b}$ 分位数をグループごとに計算するアルゴリズムを説明する．このアルゴリズムは集約関数中央値計算アルゴリズムのアイディアを一般化したアルゴリズムである．中央値を求める際に行ったことは，x と $-x+n$ が等しくなる $x(=\frac{n}{2})$ を検出したと見ることができる．これら 2 つの x の一次関数の傾きをそれぞれ b 倍，a 倍した関数 $f(x) = bx$，$g(x) = -a(x-n)$ を考えると，$f(x) = g(x)$ となるとき $x = \frac{a}{a+b}n$ であり，この x は区間 $[0, n]$ を $a : b$ に内分する点である．よって $\frac{a}{a+b}$ 分位数を求めるには，階段+の値を

Algorithm 2 階段-の計算

Notation: Sawtooth-Dec($[\![\boldsymbol{k}]\!]$)
Input: 大きさ n のソート済みのキー $[\![\boldsymbol{k}]\!]$
Output: i 番目の要素がグループ内で下から j 番目のとき $d_i = j - 1$ である $[\![\boldsymbol{d}]\!]$

1: **for** $i = 1$ **to** $n - 1$ **do in parallel**
2: $\quad [\![e_i]\!] := ([\![k_i]\!] \stackrel{?}{=} [\![k_{i+1}]\!])$
3: $[\![b_i^{(0)}]\!] := \begin{cases} [\![0]\!] & \text{if } i = n \\ [\![e_i]\!] & \text{otherwise} \end{cases}$
4: $[\![a_i^{(0)}]\!] := [\![b_i^{(0)}]\!]$
5: $t := \lceil \log_2 n \rceil$
6: **for** $j = 0$ **to** $t - 1$ **do**
7: $\quad$ **for** $i = 0$ **to** n **do in parallel**
8: $\quad\quad [\![b_i^{(j+1)}]\!] := \begin{cases} [\![b_i^{(j)}]\!] \times [\![b_{i+2^j}^{(j)}]\!] & \text{if } i + 2^j \le n, \\ [\![b_i^{(j)}]\!] & \text{otherwise,} \end{cases}$
$\quad\quad [\![a_i^{(j+1)}]\!] := \begin{cases} [\![a_i^{(j)}]\!] + [\![b_i^{(j)}]\!] \times [\![a_{i+2^j}^{(j)}]\!] & \text{if } i + 2^j \le n, \\ [\![a_i^{(j)}]\!] & \text{otherwise.} \end{cases}$
9: **return** $[\![\boldsymbol{a}^{(t)}]\!]$

b 倍，階段-の値を a 倍したものを比較して大小が逆転する点を求めればよい．より正確には，グループ内の i 番目の点の階段+，階段-をそれぞれ c_i，d_i とすると $c_i = i - 1$，$d_i = n - i$ であることから，

$$
\begin{aligned}
& i = \lceil \frac{a}{a+b} n \rceil \\
\iff & i - 1 < \frac{a}{a+b} n \le i \\
\iff & \frac{a}{a+b} n \le i \wedge i < \frac{a}{a+b} n + 1 \\
\iff & a(n-i) \le bi \wedge b(i-1) < a(n-(i-1)) \\
\iff & a(d_{i+1}+1) \le bc_{i+1} \wedge bc_i < a(d_i+1)
\end{aligned}
$$

であり，$bc_i < a(d_i + 1)$ の真偽値を ℓ_i とおくと $\neg\ell_{i+1} \wedge \ell_i$ を満たす点を求めれば $\frac{a}{a+b}$ 分位数である．$i + 1$ 番目の要素が存在しない場合に対処し，以上の内容を書き下したアルゴリズムを Algorithm 4 に示す．

3.4 提案アルゴリズムの正当性

まず，次の補題により Algorithm 1 が正しく階段+を計算することを示す．

補題 1 (階段+) i_g を g 個めのグループの先頭の添字 -1，グループの要素数を n_g とする．このとき，整数 j に対する条件

$$
b_{i_g+i}^{(j)} = \begin{cases} 0 & \textit{if } i \le 2^j, \\ 1 & \textit{otherwise,} \end{cases}
$$

$$
a_{i_g+i}^{(j)} = \begin{cases} i-1 & \textit{if } i \le 2^j, \\ 2^j & \textit{otherwise.} \end{cases}
$$

を $P_g(j)$ とすると，$0 \le j \le t$ のすべての整数 j に対して $P_g(j)$ が成立する．

証明 $k_{i_g+1} = k_{i_g+2} = \cdots = k_{i_g+n_g}$，$k_{i_g+1} \neq k_{i_g}$ であるので，$b_{i_g+1}^{(0)} = a_{i_g+1}^{(0)} = 0$，$b_{i_g+2}^{(0)} =$

Algorithm 4 集約関数 $\frac{a}{a+b}$ 分位数計算

Notation: Quantile($[\![\boldsymbol{k}]\!], a, b$)
Input: 大きさ n のソート済みのキー $[\![\boldsymbol{k}]\!]$，値 a，b
Output: i 番目の点が $\frac{a}{a+b}$ 分位数なら $f_i = 1$，そうでなければ $f_i = 0$ である $[\![\boldsymbol{f}]\!]$

1: $[\![\boldsymbol{c}]\!] \leftarrow$ Sawtooth-Inc($[\![\boldsymbol{k}]\!]$),
$\quad [\![\boldsymbol{d}]\!] \leftarrow$ Sawtooth-Dec($[\![\boldsymbol{k}]\!]$).
2: **for** $i \leftarrow 1$ **to** n **do in parallel**
3: $\quad [\![\ell_i]\!] \leftarrow ([\![c_i]\!] \times b \stackrel{?}{<} ([\![d_i]\!] + 1) \times a)$.
4: $\quad [\![t_i]\!] \leftarrow \begin{cases} [\![0]\!] & \text{if } i = n, \\ (k_i \stackrel{?}{=} k_{i+1}) & \text{otherwise.} \end{cases}$
5: $\quad [\![f_i]\!] \leftarrow [\![\ell_i]\!] \wedge ([\![t_i]\!] \vee \neg[\![\ell_{i+1}]\!])$.
6: **return** $[\![\boldsymbol{f}]\!]$.

$\cdots = b^{(0)}_{i_g+n_g} = 1$，$a^{(0)}_{i_g+2} = \cdots = a^{(0)}_{i_g+n_g} = 1$ である．すなわち，$P_g(0)$ は成立する．

$x \geq 0$ を満たす整数 x に対して $P_g(x)$ が成り立っていると仮定する．

(i) $1 \leq i \leq 2^x$ のとき，$b^{(x)}_{i_g+i} = 0$ であるので，

$$\begin{aligned} b^{(x+1)}_{i_g+i} &= 0, \\ a^{(x+1)}_{i_g+i} &= a^{(x)}_{i_g+i} \\ &= i - 1. \end{aligned}$$

(ii) $2^x < i \leq 2^{x+1}$ のとき，

$$\begin{aligned} b^{(x+1)}_{i_g+i} &= b^{(x)}_{i_g+i} \times b^{(x)}_{i_g+i-2^x} \\ &= 1 \times 0 \\ &= 0, \\ a^{(x+1)}_{i_g+i} &= a^{(x)}_{i_g+i} + b^{(x)}_{i_g+i} \times a^{(x)}_{i_g+i-2^x} \\ &= 2^x + 1 \times (i - 2^x - 1) \\ &= i - 1. \end{aligned}$$

(iii) $2^{x+1} < i$ のとき，

$$\begin{aligned} b^{(x+1)}_{i_g+i} &= b^{(x)}_{i_g+i} \times b^{(x)}_{i_g+i-2^x} \\ &= 1 \times 1 \\ &= 1, \\ a^{(x+1)}_{i_g+i} &= a^{(x)}_{i_g+i} + b^{(x)}_{i_g+i} \times a^{(x)}_{i_g+i-2^x} \\ &= 2^x + 1 \times 2^x \\ &= 2^{x+1}. \end{aligned}$$

以上より，$P_g(x+1)$ も成り立つ．

よって $0 \leq j \leq t$ を満たすすべての整数 j について $P_g(j)$ は成立する． □

同様にして，Algorithm 2 の正当性も示される．

紙面の都合により Algorithm 3，Algorithm 4 の正当性の証明は省略し，3.2 節，3.3 節の説明で代用する．

3.5 提案アルゴリズムの安全性

提案アルゴリズムはすべて既存の演算の呼び出しのみからなる．既存の演算はすべて 2.4 節で述べた機能を安全に実現しているため，提案アルゴリズムは一切の情報を漏らさない．

3.6 提案アルゴリズムの計算コスト

2 節で例としてあげた構成では，提案アルゴリズムが呼び出したすべての既存の演算はラウンド数は $O(1)$ で通信量は比較プロトコルが最大である．Algorithm 1 と Algorithm 2 は $O(n)$ の並列計算を $\lceil \log_2 n \rceil$ 回行っており，$O(\log n)$ ラウンドで $O(n \log n)$ 回の比較を要する．Algorithm 3 と Algorithm 4 は Algorithm 1 と Algorithm 2 の呼び出し以外の部分は $O(1)$ ラウンドで $O(n)$ 回の比較であるので，全体では $O(\log n)$ ラウンドで $O(n \log n)$ 回の比較である．

4 性能評価

提案手法を実装し，性能測定を行った．

4.1 実験の設定

提案手法の実用性を確認するため，n を 1 から 10^6 まで動かしながら，実装した提案手法の処理時間を測定した．

4.1.1 実装した秘匿計算の方式

3 パーティのマルチパーティプロトコルによる秘匿計算を実装し，実験を行った．実装した方式は環 $\mathbb{Z}_N$ 上の Shamir による秘密分散 [12] に基づく，2 節で例としてあげた構成である．ただし，等号判定は Cramer らの方式 [4]，比較は Damgård らの方式 [5]，乗算は Ben-Or らの方式 [2] の改良である Gennaro らの手法 [6] を実装した．N は素数 $4294967291 = 2^{32} - 5$ とした．

4.1.2 実験環境

3 台のラップトップ PC を 1 台のハブを介して 1Gbps の有線 LAN で互いに接続し，実験を行った．3 台のラップトップ PC は CPU が Intel Core i5 2540M 2.6 GHz，メモリが 8 GB，SSD が 128 GB，OS が Linux（Ubuntu 11.10）である．プログラミング言語は C++ を，コンパイラは g++ 4.6.1 を用いた．

表 5: 実行時間

n	入力のソート	集約関数中央値計算	集約関数分位数計算
10^0	0.372 [s]	0.027 [s]	0.248 [s]
10^1	0.408 [s]	0.058 [s]	0.283 [s]
10^2	0.483 [s]	0.074 [s]	0.318 [s]
10^3	0.969 [s]	0.159 [s]	0.609 [s]
10^4	4.995 [s]	0.952 [s]	3.342 [s]
10^5	42.582 [s]	8.300 [s]	27.874 [s]
10^6	603.261 [s]	123.784 [s]	362.148 [s]

4.2 測定結果

提案手法を実装し，入力の大きさ n を動かしながら実行時間を測定した．主な実行時間を表 5 に示す．提案手法はソート済みの入力に対して $n = 10^6$ でもそれぞれ 123.784 秒，362.148 秒と現実的な実行時間を達成できていることが確認できる．

5 おわりに

本稿では秘匿計算で中央値計算と $\frac{a}{a+b}$ 分位数計算の集約関数を実現するアルゴリズムを提案した．提案アルゴリズムはともに入力データ数 n に対して $O(\log n)$ ラウンドで比較プロトコル換算で $O(n \log n)$ 回の通信量である．

また，これらの提案アルゴリズムを実装し，$n = 10^6$ の場合にもソート済みの入力に対してはそれぞれ 123.784 秒，362.148 秒と現実的な時間で動作することを確認した．

参考文献

[1] Gagan Aggarwal, Nina Mishra, and Benny Pinkas. Secure computation of the k th-ranked element. In *EUROCRYPT*, pp. 40–55, 2004.

[2] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *STOC*, pp. 1–10. ACM, 1988.

[3] Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition, 2001.

[4] Ronald Cramer and Ivan Damgård. Secure distributed linear algebra in a constant number of rounds. In Joe Kilian, editor, *CRYPTO*, Vol. 2139 of *LNCS*, pp. 119–136. Springer, 2001.

[5] Ivan Damgård, Matthias Fitzi, Eike Kiltz, Jesper Buus Nielsen, and Tomas Toft. Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In *TCC*, pp. 285–304, 2006.

[6] Rosario Gennaro, Michael O. Rabin, and Tal Rabin. Simplified vss and fact-track multiparty computations with applications to threshold cryptography. In Brian A. Coan and Yehuda Afek, editors, *PODC*, pp. 101–111. ACM, 1998.

[7] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *STOC*, pp. 218–229. ACM, 1987.

[8] Michael T. Goodrich. Randomized shellsort: A simple oblivious sorting algorithm. In *SODA*, pp. 1262–1277, 2010.

[9] Rob J. Hyndman and Yanan Fan. Sample Quantiles in Statistical Packages. *The American Statistician*, Vol. 50, No. 4, pp. 361–365, November 1996.

[10] Chao Ning and Qiuliang Xu. Multiparty computation for modulo reduction without bit-decomposition and a generalization to bit-decomposition. In *ASIACRYPT*, pp. 483–500, 2010.

[11] Takashi Nishide and Kazuo Ohta. Multiparty computation for interval, equality, and comparison without bit-decomposition protocol. In *PKC*, pp. 343–360, 2007.

[12] Adi Shamir. How to share a secret. *Commun. ACM*, Vol. 22, No. 11, pp. 612–613, 1979.

[13] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *FOCS*, pp. 162–167, 1986.

[14] 濱田浩気, 五十嵐大, 千田浩司, 高橋克巳. 秘匿関数計算上の線形時間ソート. In *SCIS*, 2011.

[15] 濱田浩気, 菊池亮, 五十嵐大, 千田浩司. 秘匿計算上の結合アルゴリズム. 人工知能学会全国大会 (第 26 回) 論文集, June 2012.

[16] 五十嵐大, 千田浩司, 濱田浩気, 高橋克巳. 軽量検証可能 3 パーティ秘匿関数計算の効率化及びこれを用いたセキュアなデータベース処理. In *SCIS*, 2011.