属性情報と履歴情報の秘匿統合分析に向けた秘密計算による高速な等結合アルゴリズムとその実装

桐淵 直人[1,a)]　五十嵐 大[1]　諸橋 玄武[1]　濱田 浩気[1]

概要：本稿では，秘密計算による 2 つの表の等結合（または垂直結合）において，属性情報と履歴情報の組み合わせのように，一方の表は重複の無い一意なキーを持ち，他方は同じキーを持つデータが重複して現れるケースに着目し，このようなケースに適用可能な高速アルゴリズムを提案する．これにより，ビッグデータや IoT の潮流により増加している履歴情報と，属性情報を結合した統合分析を秘密計算を用いて安全かつ現実的な処理コストで行うことができるようになる．従来の手法には両方の表に重複が無いケースのアルゴリズムと，両方に重複が有るケースのアルゴリズムのみがあり，片方に重複があるケースでは後者を適用する必要があった．その場合結合する 2 つ表の大きさを共に n，キーの重複要素数の上限をパラメータ k として，$O(kn \log n)$ の通信量が必要であったが，提案手法では $O(n \log n)$ に改善する．重複要素数は典型的には不確定であり，しかも特に履歴の取得期間の長さに応じて大きくなる増加がちのパラメータであるから，この改善は大きいと言える．さらに提案手法を実装し，先行研究よりおよそ 1,000 倍高速で，かつ 1,000 万行の大規模な表の結合をも現実的な時間の内に完了することを示す．

キーワード：秘密計算，等結合，データベース

An Efficient Equi-join Algorithm for Secure Computation and Its Implementation toward Secure Comprehensive Analyses of Users' Attribute and History Information

Naoto Kiribuchi[1,a)]　Dai Ikarashi[1]　Gembu Morohashi[1]　Koki Hamada[1]

Abstract: We propose an efficient multi-party computation algorithm for joining two tables which one table has unique keys like attribute information and the other has non-unique keys like history information. The algorithm enables us to analyze history information, which are increasing by currents of Big-Data and IoT, joined to attribute information securely. There are existing algorithms only for tables which both tables have unique keys and for tables which both tables have non-unique keys, therefore when only one table has have non-unique keys, one should apply the latter algorithms, and these costs $O(kn \log n)$ communication cost with the parameter k is the number of maximum duplication of keys and n is the number of rows of the tables, while proposed algorithm improves it to $O(n \log n)$. The improvement is significant since the parameter k is typically uncertain and in addition, it increases as a period of history becomes long. Furthermore, we implemented the proposed algorithm and show that our implementation of it is one thousand times faster than an existing combination of an algorithm and an implementation, and ours can process the joining of tables which have 10 millions of rows in a realistic time.

Keywords: Secure Computation, Equi-join, Database

[1] NTT セキュアプラットフォーム研究所
NTT Secure Platform Laboratories
[a)] kiribuchi.naoto@lab.ntt.co.jp

1. はじめに

1.1 背景

データ分析技術の進展に伴い，ユーザの行動履歴などのデータ利活用の機運が高まる中で，ユーザのプライバシに配慮したデータ保護が課題となっている．こうしたデータの保護と利活用の両立に向けた技術的なアプローチとして，プライバシ保護データマイニングの研究が盛んである．その中でも秘密計算は，ここ数年で実装報告が数多くなされており，実用的な技術として注目されている [2], [3], [18].

秘密計算は，暗号化や秘密分散法によって秘匿されたデータを参加者に預け，参加者に対して元のデータを知らせずに所望の関数を計算する技術である．特に複数の参加者が協調して実行する方式をマルチパーティ計算と呼ぶ．

秘密計算技術を使うと，例えば機関 $\mathcal{A}$ と $\mathcal{B}$ がそれぞれ保持するデータを秘匿した状態で参加者に預け，$\mathcal{A}$ と $\mathcal{B}$ が互いにデータを見せ合うことなく，参加者にもデータを見せない状態で，$\mathcal{A}$ のデータと $\mathcal{B}$ のデータに跨った横断的な分析処理が可能になる．

データの横断分析の一般的な手法として，データベース処理における表の等結合がある．多くのデータベースでは，データを行と列からなる表を単位として管理しており，表の等結合は**図 1** の例に示されるように，同じキー（ここでは「No.」）を元に 2 つの表 L, R を結合して表 J を得る操作である．キーの情報を使うことで，別々の表に格納されたデータを結びつけることができるため，複数の機関からデータを持ち寄る横断分析に欠かせない処理となっている．

横断分析において特に典型的かつ重要と考えられるのが，**属性情報と履歴情報の関連性の導出**である．例えば図 1 において，身長や体重といったユーザの属性情報と購入品といったユーザの履歴情報とを結合することで，「おいしい水は大柄のユーザによく売れる傾向がある」といった関連性を見出すことが考えられる．とりわけ，ビッグデータや IoT の進展に伴って，履歴情報への注目が高まっている．

ここで，ユーザの属性情報は 1 ユーザに対して一意であるためキーの重複がなく，履歴情報には同じユーザによる履歴が蓄積されるためキーの重複が含まれる．このように，データ分析における表の等結合では，キーの重複がない表とキーの重複のある表の結合が重要な処理となる．

さらに，計算時間のオーダーの面でも等結合の中で片方のみ重複ありのケースに着目する理由がある．結合する 2 つの表の行数をそれぞれ m, n とするとき，両方の表に重複が含まれる場合の等結合では，キーが等しい行同士を全て結合して出力する．例えば図 1 の表 L に No.9 の行がもう 1 行あったと仮定した場合，出力は掛け合わせで $2 \times 2 = 4$ 行の No.9 が出力される．このように，出力行数は最大で mn となる．処理コストは出力データ量より決して小さくならないため，原理的に計算時間は最低でも mn となってしまう．しかし，重複が片方の表にのみ含まれる場合，出力される表の行数は高々 n であり，アルゴリズムの工夫によって処理コストが改善される望みがある．

そこで本稿ではキーに重複のない表と重複を含む表の等結合に着目し，効率的な秘密計算アルゴリズムを提案する．

1.2 関連研究

表の等結合を秘密計算で実現する手法として，以下の提案がなされてきた．これらはいずれもマルチパーティ計算による方式である．マルチパーティ計算においては参加者間の通信量が主たる処理コストとして指標にされおり [4]，ここに挙げる手法は，通信量削減の歴史であるとも言える．

志村らによる手法 [14], [15] では，結合した表の行数を秘匿したまま処理可能である．しかし，入力する 2 つの表の行数を m, n としたとき，常に mn 行の表が出力され，表の中に空を示す行が非常に多く含まれるため非効率である．

濱田らは，結合後の表に含まれる行のみを出力する手法 [11] を提案した．この手法では，入力される表の行数の和を n とすると $O(n \log n)$ の通信量で実現可能である．さらに，結果の行数を秘匿する手法 [13] も提案している．しかしながら，これらの手法はいずれも結合に使うキーの要素に重複がないことを前提としていた．

Laur らは，キーに重複がない場合の等結合の手法 [7] と重複がある場合の等結合の手法 [8] を提案している．後者の手法では，重複する要素数の上限を k とし，k が既知であることを前提として，$O(k^2 n \log n)$ の通信量で表の等結合を実現している．また，濱田らは Laur と同様の前提の下で，通信量が $O(kn \log n)$ の手法を提案している [12].

1.3 本研究の貢献

提案アルゴリズムでは，秘密計算によるキーに重複のない表とキーに重複のある表の等結合について，キーの重複要素数の上限が既知という前提の排除と，処理コストの削減を達成した．さらに，本稿では提案アルゴリズムを実装し，測定によって得られた処理時間も報告する．

1.3.1 キーの重複要素数の上限が既知という前提の排除

Laur らの手法 [8] や濱田らの手法 [12] では，両方の表のキーに重複が許されるが，重複する要素数の上限を k とし，k が既知という前提である．すなわち，参加者が k を知る必要がある．k は，履歴情報では例えば「どの店で何回購入したか」など，日々増大し，かつ秘匿されることが望ましいであろうデータの上限値であり，正確に近い値を設定することは難しい．とはいえ，余裕を持って大きな値とすれば，比例して計算時間が悪化してしまう．

一方，提案アルゴリズムは，横断分析で重要となる属性情報と履歴情報のようなキーに重複の無い表と重複を含む表の結合に着目したアルゴリズムであり，参加者が k を知らない状態で等結合を行うことができる．

入力される表 L
（キーに重複の無い属性情報）

No.	身長 (cm)	体重 (kg)
3	200	100
5	110	19
9	160	85

入力される表 R
（キーに重複のある履歴情報）

No.	購入品
3	おいしい水
7	ミックスオレ
9	傷薬
9	おいしい水

表 L と表 R の等結合により出力される表 J
（属性情報と履歴情報を同じキーで結合）

No.	身長 (cm)	体重 (kg)	購入品
3	200	100	おいしい水
9	160	85	傷薬
9	160	85	おいしい水

図 1 表の等結合の例
Fig. 1 An example of an equi-join on two tables.

1.3.2 処理コストの削減

濱田らの手法 [12] は，キーに重複が含まれる場合に重複するキーの要素数の上限を k，結合する表の大きさを n とするとき，$O(kn \log n)$ の通信量が必要である．提案手法は，重複するキーの要素数の上限 k が未知の場合にも $O(n \log n)$ の通信量で実現でき，k が通信量に影響しない．

2. 準備

2.1 記法と定義

本稿で用いる主な記法と定義を示す．

$\mathbb{Z}_N$：0 から N までの整数．有限環をなす．
$[\![x]\!]$ ：$x \in \mathbb{Z}_N$ を暗号化ないし秘密分散で秘匿した値．
$[\![x]\!] + [\![y]\!]$：秘密計算による加算．$[\![x+y]\!]$ を出力する．
$[\![x]\!] \times [\![y]\!]$：秘密計算による乗算．$[\![x \times y]\!]$ を出力する．
$\texttt{PrefixSum}([\![x_1]\!], \ldots, [\![x_n]\!])$：2.2.2 節に示す Prefix-sum．
$\langle\sigma\rangle$：秘密計算による置換 σ．詳細は 2.2.3 節を参照．
$[\![\boldsymbol{f}]\!]$：$([\![f_1]\!], \ldots, [\![f_n]\!])$ を要素に持つ要素列（太字表記）．
$[\![\sigma([\![\boldsymbol{f}]\!])]\!]$：置換 σ によって要素列 $[\![\boldsymbol{f}]\!]$ を置換した要素列．
$\texttt{Sort}([\![x_1]\!], \ldots, [\![x_n]\!])$：安定ソート．置換 $\langle\sigma\rangle$ を出力する．

2.2 部品とする基礎的アルゴリズム

本節では，提案アルゴリズムの部品として用いられる基礎的アルゴリズムを示す．提案アルゴリズムは，基礎的アルゴリズムを組み合わせることで構成される．

2.2.1 加算および乗算

加算は，$a, b \in \mathbb{Z}_N$ の $[\![a]\!], [\![b]\!]$ が与えられたとき，$c = a + b$ となる $[\![c]\!]$ を秘匿した状態で求めるアルゴリズムである．具体的には Ben-Or らの手法 [1] が知られており，マルチパーティ計算における参加者間の通信は不要である．

乗算は，$a, b \in \mathbb{Z}_N$ の $[\![a]\!], [\![b]\!]$ が与えられたとき，$c = a \times b$ となる $[\![c]\!]$ を秘匿した状態で求める手法で，具体的には五十嵐らの方法 [5], [17] を用いることができる．この手法は，マルチパーティ計算における参加者間の通信が必要となる．

2.2.2 Prefix-sum

順序立てられた複数の要素に対して，ある要素と，それまでに現れた全ての要素との和を求める操作およびその結果の要素列を Prefix-sum と呼ぶ．すなわち，$([\![x_1]\!], \ldots, [\![x_n]\!])$ が与えられたとき，$y_i = \sum_{j=1}^{i} x_j$ となる $([\![y_1]\!], \ldots, [\![y_n]\!])$ を求める操作である．本稿では，この操作を $([\![y_1]\!], \ldots, [\![y_n]\!]) \leftarrow \texttt{PrefixSum}([\![x_1]\!], \ldots, [\![x_n]\!])$ と記載する．前節の加算アルゴリズムより，Prefix-sum は参加者間の通信不要で実行可能である．

2.2.3 置換

順序立てられた要素を並べ替える操作や，その写像を置換と呼ぶ．例えば $(1,2,3)$ を $(3,1,2)$ に並べ替える操作は，$\sigma(1) = 3, \sigma(2) = 1, \sigma(3) = 2$ を満たす写像 σ による置換とみなされる．写像の性質から，複数の置換を合成できる．すなわち置換 $\sigma \cdot \pi$ は，要素 x を $\sigma(\pi(x))$ に写す．

秘密計算による置換 $\langle\sigma\rangle$ を実現する方法の一つとして，五十嵐らの手法 [16] を利用できる．この方法ではマルチパーティ計算において例えば参加者を X, Y, Z とするとき，置換 σ を $\sigma = \sigma_{ZX} \cdot \sigma_{YZ} \cdot \sigma_{XY}$ の合成とする．σ_{XY} は参加者 X と Y のみに共有された置換であり，参加者 Z には知らされない．Z は，X と Y からの再分散によって σ_{XY} による置換後の秘匿要素列を得る．σ_{YZ}, σ_{ZX} も同様であり，いずれの参加者にとっても自分の知らない置換が含まれるため，全ての参加者が対応関係を知らないまま置換 $\langle\sigma\rangle$ を実行できる．ここで，$\langle\sigma\rangle$ は $\sigma_{XY}, \sigma_{YZ}, \sigma_{ZX}$ に分割されて適用順序が決められた複製秘密分散 [6] とみなせる．これは，σ をランダム置換とした濱田らのシャッフル [10] の一般化であり，通信コストは入力サイズに対して線形である．

2.2.4 逆置換

置換は全単射であるため，逆写像を持つ．ある置換 σ の逆写像を逆置換と呼び，σ^{-1} で記す．すなわち $\sigma(x) = y$ のとき，$\sigma^{-1}(y) = x$ である．特に，$\sigma^{-1} \cdot \sigma$ は恒等置換と呼ばれる順序を変えない置換である．

秘密計算による逆置換 $\langle\sigma^{-1}\rangle$ の方法は，置換の場合と同様である．置換 $\langle\sigma\rangle$ が $\sigma = \sigma_{ZX} \cdot \sigma_{YZ} \cdot \sigma_{XY}$ となる $\sigma_{XY}, \sigma_{YZ}, \sigma_{ZX}$ の合成として与えられたとき，逆置換は $\sigma^{-1} = \sigma_{XY}^{-1} \cdot \sigma_{YZ}^{-1} \cdot \sigma_{ZX}^{-1}$ より，各参加者 X, Y, Z が持つ置換の逆置換を順序を反転して適用すればよい．

2.2.5 安定ソート

同じ要素の順序関係が保存される並べ替えを安定ソートという．すなわち $(x_1, \ldots, x_n)$ を $(y_1, \ldots, y_n)$ に置換する安定ソート σ において，$x_i = x_j$ のとき $\sigma(x_i) = y_u, \sigma(x_j) = y_v$ とすると，$i < j$ のときのみ $u < v$ となる．

入力される表 L

$[\![\ell_1]\!]$	$[\![v_{1,2}]\!]$	$\cdots$	$[\![v_{1,a}]\!]$
$[\![\ell_2]\!]$	$[\![v_{2,2}]\!]$	$\cdots$	$[\![v_{2,a}]\!]$
$\vdots$	$\vdots$	$\ddots$	$\vdots$
$[\![\ell_m]\!]$	$[\![v_{m,2}]\!]$	$\cdots$	$[\![v_{m,a}]\!]$

入力される表 R

$[\![r_1]\!]$	$[\![u_{1,2}]\!]$	$\cdots$	$[\![u_{1,b}]\!]$
$\vdots$	$\vdots$	$\ddots$	$\vdots$
$[\![r_n]\!]$	$[\![u_{n,2}]\!]$	$\cdots$	$[\![u_{n,b}]\!]$

表 L と表 R の等結合により出力される表 J

$[\![r'_1]\!]$	$[\![v'_{1,2}]\!]$	$\cdots$	$[\![v'_{1,a}]\!]$	$[\![u'_{1,2}]\!]$	$\cdots$	$[\![u'_{1,b}]\!]$
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
$[\![r'_n]\!]$	$[\![v'_{n,2}]\!]$	$\cdots$	$[\![v'_{n,a}]\!]$	$[\![u'_{n,2}]\!]$	$\cdots$	$[\![u'_{n,b}]\!]$

図 2 提案アルゴリズムの入出力
Fig. 2 The input and output of the proposed algorithm.

秘密計算による手法としては，例えば五十嵐らの方法 [16] が挙げられる．本稿では，安定ソートによる出力を置換 $\langle\sigma\rangle$ とし，$\langle\sigma\rangle \leftarrow \mathtt{Sort}([\![x_1]\!], \ldots, [\![x_n]\!])$ と記す．

3. 提案アルゴリズム

3.1 入出力

図 2 に提案アルゴリズムの入出力を図示した．それぞれの内容を次節以降に説明する．

3.1.1 入力

結合する 2 つの表をそれぞれ L, R とする．表の大きさは，L が m 行 a 列，R が n 行 b 列とする．L, R ともにキー情報を 1 列目にもち，それぞれ $([\![\ell_1]\!], \ldots, [\![\ell_m]\!])$，$([\![r_1]\!], \ldots, [\![r_n]\!])$ とする．表の要素は，キー情報に加えて i 行目 j 列目にそれぞれ $[\![v_{i,j}]\!]$，$[\![u_{i,j}]\!]$ を持つこととする．

$(\ell_1, \ldots, \ell_m)$ には同じ値の要素が存在しないこととし，$(r_1, \ldots, r_n)$ には重複する値が存在してもよい．また，入力される全ての要素について，値が 0 でないこととし，入力に含まれる可能性がある場合には，例えば一律に加減算することでエスケープしておく．

3.1.2 出力

i 行目が $([\![r'_i]\!], [\![v'_{i,2}]\!], \ldots, [\![v'_{i,a}]\!], [\![u'_{i,2}]\!], \ldots, [\![u'_{i,b}]\!])$ である n 行 $a+b-1$ 列の表 J で，r_i と同じ値が $(\ell_1, \ldots, \ell_m)$ に存在しないとき，i 行目の値は全て 0，すなわち $2 \le \alpha \le a, 2 \le \beta \le b$ について $r'_i = 0, v'_{i,\alpha} = 0, u'_{i,\beta} = 0$ となる．$r_i = \ell_j$ のとき，$r'_i = r_i, v'_{i,\alpha} = v_{j,\alpha}, u'_{i,\beta} = u_{i,\beta}$ となる．

3.2 処理手順

アルゴリズム 1 に提案方式の手順を示す．提案方式のアイディアは，キーに重複のない表 L の値を重複のある表 J に 1 列ずつコピーすることである．最初に結合するキーの安定ソートを用いて入力された行と出力する行の対応関係を置換 $\langle\sigma\rangle$ としておき，列ごとに $\langle\sigma\rangle$ を適用してコピー先の変数を並べてから Prefix-sum によりコピーする．

アルゴリズム中の各ステップの挙動については，次節以降に図 1 の例に示した値を代入することで，提案アルゴリズムの具体的な流れを追いながら説明する．

3.2.1 ステップ 1

入力された 2 つの表のキー情報をソートする置換 σ を得る．置換 σ は，以下の 1 行目の要素列を 2 行目の順序に並べ替える．

$[\![3]\!], [\![5]\!], [\![9]\!], ③, ⑦, ⑨, ⑨, [\![3]\!], [\![5]\!], [\![9]\!]$

$[\![3]\!], ③, [\![3]\!], [\![5]\!], [\![5]\!], ⑦, [\![9]\!], ⑨, ⑨, [\![9]\!]$

ここでは説明のため，置換前後の位置関係を $[\![x]\!], ⓨ, [\![z]\!]$ という記号を用いて同じ値についても区別して表現するが，実際には区別されずに処理される．$[\![x]\!]$ と $[\![z]\!]$ が表 L から，ⓨ が表 R から得られたキー情報である．

3.2.2 ステップ 2 から 8

表 L の列ごとに繰り返される処理である．図 1 の例では「身長」と「体重」の列があるが，ここでは「体重」の列について処理を追う．要素列 $[\![\boldsymbol{f}]\!]$ は，体重の列の値から次の通りとなる．

$[\![100]\!], [\![19]\!], [\![85]\!], ⓪, ⓪, ⓪, ⓪, [\![\text{-}100]\!], [\![\text{-}19]\!], [\![\text{-}85]\!]$

置換 σ によって並べ替えられた $[\![\boldsymbol{g}]\!]$ は，次の通り．

$[\![100]\!], ⓪, [\![\text{-}100]\!], [\![19]\!], [\![\text{-}19]\!], ⓪, [\![85]\!], ⓪, ⓪, [\![\text{-}85]\!]$

よって Prefix-sum によって得られる $[\![\boldsymbol{g'}]\!]$ は，以下のようになる．この手順により表 L の列の値 $[\![x]\!]$ が ⓨ にコピーされていることがわかる．また $[\![z]\!]$ は，$[\![x]\!]$ の値がコピーされる終端に位置し，プログラミングにおける番兵の役割を果たす．

$[\![100]\!], ⑩⓪, [\![0]\!], [\![19]\!], [\![0]\!], ⓪, [\![85]\!], ⑧⑤, ⑧⑤, [\![0]\!]$

逆置換 σ^{-1} によって得られる $[\![\boldsymbol{f'}]\!]$ は，次の通り．

$[\![100]\!], [\![19]\!], [\![85]\!], ⑩⓪, ⓪, ⑧⑤, ⑧⑤, [\![0]\!], [\![0]\!], [\![0]\!]$

よって出力される列の値は，⑩⓪, ⓪, ⑧⑤, ⑧⑤ となる．ここで，表 L に同じキーが存在せず結合されなかった行の値は ⓪ となる．

3.2.3 ステップ 9 から 13

前節において表 L の列の値が全て 1 だった場合の処理であり，$([\![e_1]\!], \ldots, [\![e_n]\!])$ の各要素 e_i は該当する行が結合された場合に 1，結合する相手がいなかった場合に 0 となる．今回は，①, ⓪, ①, ① となる．

3.2.4 ステップ 14 から 19

表 R の列の各行ごとに繰り返され，表 L に同じキーが存在せず結合されなかった行を 0 に置き換えることで秘匿する処理である．$([\![e_1]\!], \ldots, [\![e_n]\!])$ を元に表 R の中の結合されなかった行の値を 0 にし，結合された行の値をコピーする．今回は，結合されない「ミックスオレ」の行が 0 となり，他の行には元と同じ値が代入される．

アルゴリズム 1 表の等結合

入力： 図 2 に示した表 L, R
出力： 図 2 に示した表 J

1: $\langle\sigma\rangle \leftarrow \mathtt{Sort}([\![\ell_1]\!], \ldots, [\![\ell_m]\!], [\![r_1]\!], \ldots, [\![r_n]\!], [\![\ell_1]\!], \ldots, [\![\ell_m]\!])$
2: **for** $j = 2$ **to** a **do**
3: $\quad [\![\boldsymbol{f}]\!] \leftarrow ([\![v_{1,j}]\!], \ldots, [\![v_{m,j}]\!], \underbrace{[\![0]\!], \ldots, [\![0]\!]}_{n\text{ 個}}, [\![-v_{1,j}]\!], \ldots, [\![-v_{m,i}]\!])$
4: $\quad [\![\boldsymbol{g}]\!] \leftarrow [\![\sigma([\![\boldsymbol{f}]\!])]\!]$
5: $\quad [\![\boldsymbol{g'}]\!] \leftarrow \mathtt{PrefixSum}([\![\boldsymbol{g}]\!])$
6: $\quad [\![\boldsymbol{f'}]\!] \leftarrow [\![\sigma^{-1}([\![\boldsymbol{g'}]\!])]\!]$
7: $\quad ([\![\tilde{v}_{1,j}]\!], \ldots, [\![\tilde{v}_{n,j}]\!]) \leftarrow ([\![f'_{m+1}]\!], \ldots, [\![f'_{m+n+1}]\!])$
8: **end for**
9: $[\![\boldsymbol{f}]\!] \leftarrow (\underbrace{[\![1]\!], \ldots [\![1]\!]}_{m\text{ 個}}, \underbrace{[\![0]\!], \ldots, [\![0]\!]}_{n\text{ 個}}, \underbrace{[\![-1]\!], \ldots, [\![-1]\!]}_{m\text{ 個}})$
10: $[\![\boldsymbol{g}]\!] \leftarrow [\![\sigma([\![\boldsymbol{f}]\!])]\!]$
11: $[\![\boldsymbol{g'}]\!] \leftarrow \mathtt{PrefixSum}([\![\boldsymbol{g}]\!])$
12: $[\![\boldsymbol{f'}]\!] \leftarrow [\![\sigma^{-1}([\![\boldsymbol{g'}]\!])]\!]$
13: $([\![e_1]\!], \ldots, [\![e_n]\!]) \leftarrow ([\![f'_{m+1}]\!], \ldots, [\![f'_{m+n+1}]\!])$
14: **for** $i = 1$ **to** n **do**
15: $\quad [\![\tilde{r}_i]\!] \leftarrow [\![e_i]\!] \times [\![r_i]\!]$
16: $\quad$ **for** $j = 2$ **to** b **do**
17: $\quad\quad [\![\tilde{u}_{i,j}]\!] \leftarrow [\![e_i]\!] \times [\![u_{i,j}]\!]$
18: $\quad$ **end for**
19: **end for**
20: $\langle\tilde{\sigma}\rangle \leftarrow \mathtt{Sort}([\![e_1]\!], \ldots, [\![e_n]\!])$
21: $([\![r'_1]\!], \ldots, [\![r'_n]\!]) \leftarrow [\![\tilde{\sigma}([\![\tilde{r}_1]\!], \ldots, [\![\tilde{r}_n]\!])]\!]$
22: **for** $j = 2$ **to** a **do**
23: $\quad ([\![v'_{1,j}]\!], \ldots, [\![v'_{n,j}]\!]) \leftarrow \langle\tilde{\sigma}([\![\tilde{v}_{1,j}]\!], \ldots, [\![\tilde{v}_{n,j}]\!])\rangle$
24: **end for**
25: **for** $j = 2$ **to** b **do**
26: $\quad ([\![u'_{1,j}]\!], \ldots, [\![u'_{n,j}]\!]) \leftarrow \langle\tilde{\sigma}([\![\tilde{u}_{1,j}]\!], \ldots, [\![\tilde{u}_{n,j}]\!])\rangle$
27: **end for**

3.2.5 ステップ 20 から 27

結合しなかった行を上に寄せる処理である．ステップ 19 までの処理では，入力した表 R の行の順序が保たれた状態で結合されなかった行の各要素が全て 0 となり，復元した際に入力した表 R の何行目が結合されたのかが分かってしまう．ここでは，$([\![e_1]\!], \ldots, [\![e_n]\!])$ をソートすることで結合しなかった行を上に寄せる置換 $\langle\tilde{\sigma}\rangle$ を各列に適用する．

3.3 処理コスト

アルゴリズム 1 の中で通信が必要となる処理は，ステップ 1 における長さ $2m+n$ のソートが 1 回，ステップ 4, 10 における長さ $2m+n$ の置換が合わせて a 回，ステップ 6, 12 の逆置換も同様に合わせて a 回，ステップ 15, 17 における乗算が bn 回，ステップ 20 における長さ n のソートが 1 回，ステップ 21, 23, 26 における長さ n の置換が合わせて $a+b-1$ 回である．ソートの通信コストは，濱田らの手法 [10] により $O((m+n)\log(m+n))$，置換および逆置換については入力に対して線形，乗算については 1 回につき定数量の通信が必要となり，表の列数 a, b を定数とみなすと提案アルゴリズムの通信量は，入力する 2 つの表の行数を m, n とすると，$O((m+n)\log(m+n))$ となる．

アルゴリズム 2 変形：結合した行だけを出力する

入力： アルゴリズム 1 の $([\![e_1]\!], \ldots, [\![e_n]\!])$ および出力された表 J
出力： 要素が全て 0 である行を除去した表 J'

1: $[\![c]\!] \leftarrow \sum_{i=1}^{n} [\![e_i]\!]$
2: c を公開し，表 J の下から c 行を出力とする．

3.4 セキュリティモデル

提案アルゴリズムは，マルチパーティ計算における passive モデルないし active モデルにおいて秘匿性を持つ．passive モデルは，n 人のうち k 人未満の参加者が攻撃者としてプロトコルには従うが盗み見をするモデルである．active モデルは，盗み見だけでなく任意の不正な動作をする攻撃者を想定するモデルである．部品となる基礎的アルゴリズムが前提とするセキュリティモデルが全て active モデルのとき提案アルゴリズムも active モデルとなり，基礎的アルゴリズムに passive モデルが含まれるとき提案アルゴリズムも passive モデルとなる．

提案アルゴリズムは，処理中に一切のデータを公開することがなく，秘匿化される前のデータのみならず秘匿結合された行の箇所や重複したキーの要素数も参加者に対して秘匿する．

3.5 変形：結合した行だけを出力する

結合後の表の行数が参加者に知られてもよい場合，結合された行だけを出力することができる．具体的な手順は**アルゴリズム 2** に示した通り，結合した行を e_i によって数え上げ，その行数を公開して結合した表の下からその行数分だけ出力すればよい．

この変形による通信コストの増加は，ステップ 2 に必要な公開 1 回分のみである．

4. 実装

本節では，提案したアルゴリズムの実装について，性能測定の環境および測定結果を報告する．

4.1 測定環境

実装では，Shamir の $(2, 3)$ 閾値秘密分散法 [9] によって $p = 2^{61} - 1$ 未満の整数を入力として扱うマルチパーティ計算として，前節までに示したアルゴリズムを適用した．具体的には，passive モデルでの乗算には文献 [17] の方法を，active モデルでは文献 [5] の方法を用いた．passive モデルでの置換，逆置換，安定ソートには文献 [16] の方法を，active モデルでは文献 [5] の方法を用いた．また，プログラミング言語として C++を用い，passive モデルと active モデルの両方に対応した秘密計算の実装である MEVAL2[18] の上に実装した．

参加者の処理を担う計算機は，**表 1** の性能を持つ独立した 3 台のサーバで，各サーバ間の通信路は，1Gbps のイー

表 1　性能測定サーバ性能
Table 1　Specifications on the servers.

項目	情報
OS	CentOS 7.1.1503
CPU	Intel® Core™ i7-6700
コア周波数	3.4GHz
ソケット数	1
コア数	4
論理 CPU 数	8
メモリ	32GB
ディスク	SSD 256GB

サネットである．

測定に用いたのは変形を含まないアルゴリズム 1 であり，測定にあたっては，あらかじめ 2 つの表を秘密分散により秘匿した状態で各サーバに格納した．

4.2　測定結果

前節に示した環境下において，行数などの条件を変えながら各サーバの処理時間を測定した．各項目について 5 回実施した測定の平均値を報告する．

まず，passive モデルと active モデルについて表の行数を変えながら処理時間を測定した結果を**表 2** に示す．入力はそれぞれ 5 列の同じ行数を持つ 2 つの表で，各表の 1 列目をキー情報とした．出力は入力と同じ行数で 9 列の大きさの等結合された表である．**図 3** は，縦軸に処理時間，横軸に表の行数をとり，表 2 と先行研究である文献 [8] の報告値をプロットした両対数グラフである．なお，キーの重複を認める濱田らの手法 [12] については，実装の記載がないため，本稿では性能測定の比較を行わない．

文献 [8] での測定環境は，12 コア 3GHz の CPU に 48GB のメモリを搭載した 3 台のサーバで，ネットワークは 1Gbps である．測定に用いたデータは，本稿と同じく 5 列同士の表の等結合でキー情報は 1 列である．この文献の手法は，passive モデルに限定されており，重複するキーの要素数の上限 k が既知の前提である．k が処理時間に影響するため，表 2 には $k = 2$ および $k = 16$ の場合を記載した．ただし，文献には正確な処理時間が記載されていなかったため，グラフから読み取った値をプロットした．また，1 万行以上のデータに対する測定については報告されていない．

先行研究 [8] と同じ passive モデル同士を比較すると，1,000 行のデータにおける処理時間について，先行研究の処理時間は，$k = 2$ のとき約 675 倍，$k = 16$ のとき約 1,000 倍かかっている．提案アルゴリズムは，理論的に k が処理時間に影響しないため，重複する要素数の上限 k に関わらず提案アルゴリズムの方が先行研究より高速だと言える．

また，提案アルゴリズムの active モデルは，1,000 行のデータにおける処理時間が提案アルゴリズムの passive モデルと比べて約 7.1 倍掛かっているが，それでもなお先行研究の passive モデルのみの実装結果 [8] より高速である．

表 2　性能測定結果
Table 2　Performance results of the proposed algorithm.

	処理時間（秒）	
行数	passive	active
1×10^3	0.04	0.285
1×10^4	0.248	0.777
1×10^5	1.822	5.066
1×10^6	15.132	44.043
1×10^7	156.301	441.584

5.　おわりに

本稿では，データを秘匿した状態での横断分析の実現に向け，属性情報と履歴情報のように結合するキーに重複が含まれる表と含まれない表を効率よく等結合する秘密計算アルゴリズムを提案した．従来の手法では，重複する要素数の上限 k を既知であるという前提のもと，表の大きさ n に対して $O(kn \log n)$ の通信量が必要であったが，提案手法は，k が既知という前提が不要で，k によらず $O(n \log n)$ の通信量で実行できる．

さらに，提案したアルゴリズムを実装し，実行時間を測定することで，1,000 万行の表同士でも処理可能かつ先行研究と比較しておよそ 1,000 倍高速であることを示した．

参考文献

[1] Ben-Or, M., Goldwasser, S. and Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation, *Proceedings of the twentieth annual ACM symposium on Theory of computing*, ACM, pp. 1–10 (1988).

[2] Bogdanov, D., Laur, S. and Willemson, J.: Sharemind: A framework for fast privacy-preserving computations, *European Symposium on Research in Computer Security*, Springer, pp. 192–206 (2008).

[3] Burkhart, M., Strasser, M., Many, D. and Dimitropoulos, X.: SEPIA: Privacy-preserving aggregation of multi-domain network events and statistics, *Proceedings of the 19th USENIX Security Symposium*, pp. 223–239 (2010).

[4] Damgård, I., Fitzi, M., Kiltz, E., Nielsen, J. B. and Toft, T.: Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation, *Theory of Cryptography Conference*, Springer, pp. 285–304 (2006).

[5] Ikarashi, D., Kikuchi, R., Hamada, K. and Chida, K.: Actively Private and Correct MPC Scheme in $t < n/2$ from Passively Secure Schemes with Small Overhead, Cryptology ePrint Archive, Report 2014/304 (2014).

[6] Ito, M., Saito, A. and Nishizeki, T.: Secret sharing scheme realizing general access structure, *Electronics and Communications in Japan (Part III: Fundamental Electronic Science)*, Vol. 72, No. 9, pp. 56–64 (1989).

[7] Laur, S., Talviste, R. and Willemson, J.: From oblivious AES to efficient and secure database join in the multiparty setting, *International Conference on Applied Cryptography and Network Security*, Springer, pp. 84–101 (2013).

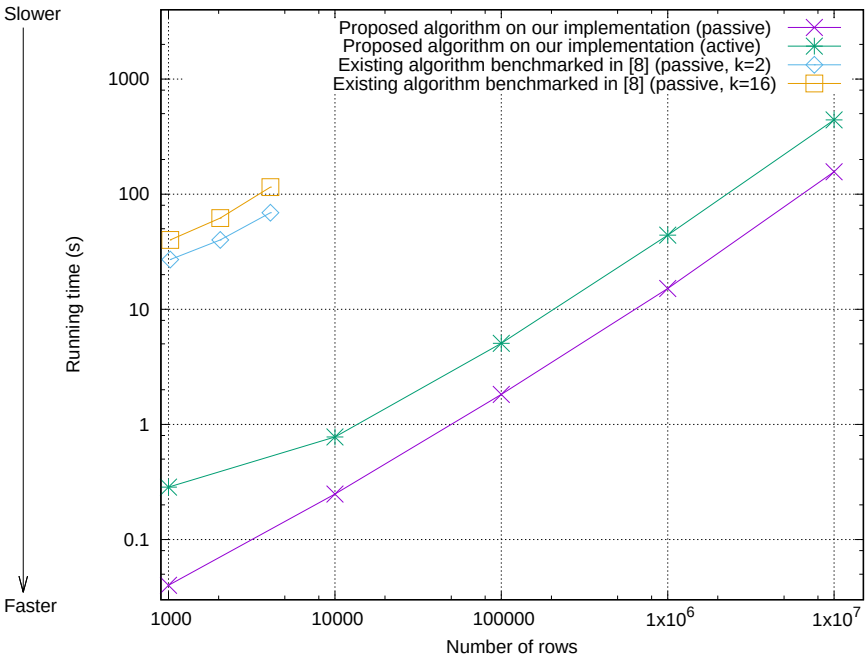

図 3 先行研究との性能比較

Fig. 3 Performance comparisons between our implementation and existing work.

[8] Laur, S., Talviste, R. and Willemson, J.: From oblivious AES to efficient and secure database join in the multiparty setting, Cryptology ePrint Archive, Report 2013/203 (2013).

[9] Shamir, A.: How to share a secret, *Communications of the ACM*, Vol. 22, No. 11, pp. 612–613 (1979).

[10] 濱田浩気，五十嵐大，千田浩司，高橋克巳：3 パーティ秘匿関数計算上のランダム置換プロトコル，コンピュータセキュリティシンポジウム (CSS)2010，情報処理学会コンピュータセキュリティ研究会 (2010).

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

[12] 濱田浩気，桐淵直人，五十嵐大：キーに重複がある場合の秘密計算向け結合アルゴリズム，暗号と情報セキュリティシンポジウム (SCIS) 2015，電子情報通信学会 (2015).

[13] 濱田浩気，菊池　亮，五十嵐大，千田浩司：秘密計算上の表の大きさを秘匿した等結合，暗号と情報セキュリティシンポジウム (SCIS) 2013，電子情報通信学会 (2013).

[14] 志村正法，遠藤つかさ，宮崎邦彦，吉浦　裕：安全で機能制限のないデータベースを実現するマルチパーティプロトコルを用いた関係代数演算，情報処理学会研究報告コンピュータセキュリティ (CSEC)， Vol. 2008, No. 71, pp. 187–193 (2008).

[15] 志村正法，宮崎邦彦，西出隆志，吉浦　裕：秘密分散データベースの構造演算を可能にするマルチパーティプロトコルを用いた関係代数演算，情報処理学会論文誌，Vol. 51, No. 9, pp. 1563–1578 (2010).

[16] 五十嵐大，濱田浩気，菊池　亮，千田浩司：インターネット環境レスポンス 1 秒の統計処理を目指した，秘密計算基数ソートの改良，暗号と情報セキュリティシンポジウム (SCIS) 2014，電子情報通信学会 (2014).

[17] 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，暗号と情報セキュリティシンポジウム (SCIS) 2011，電子情報通信学会 (2011).

[18] 菊池　亮，五十嵐大，濱田浩気，千田浩司：改ざん検知機能付きの実用的な秘密計算システム MEVAL2，暗号と情報セキュリティシンポジウム (SCIS) 2015，電子情報通信学会 (2015).