情報処理学会論文誌 Vol.58 No.9 1464–1482 (Sep. 2017)

秘密計算を用いた時系列情報の安全な集計方法

奈良 成泰[1,a)] 天田 拓磨[1] 西出 隆志[2] 土井 洋[3] 吉浦 裕[1]

受付日 2016年12月5日, 採録日 2017年6月6日

概要：個人や組織の活動にともなって時系列的に発生する情報をサーバで安全に集計することは実世界で大きなニーズがある．この集計において，サーバは受け取る時系列情報の範囲をあらかじめ予想できないため，情報の受取りにともなって集計表の値の加算だけではなく，集計表の拡張を行う必要がある．本論文では，時系列情報の安全な集計問題を新たに定義したうえで，秘密分散によって時系列情報を秘匿しながらマルチパーティ計算によって集計する方式を検討する．まず，秘密分散とマルチパーティ計算によって個々の値を秘匿しても，アクセスパターンを通じて集計表の推定が可能になることを示す．表の全探索によって値の加算と表の拡張を行う方法を提案し，アクセスパターンは秘匿できるが通信量が大きいという問題点を明らかにする．この分析に基づいて，全探索を避けながらアクセスパターンを秘匿するために，再帰的 Path ORAM を用いる手法を提案し，通信量のオーダーレベルの削減効果を明らかにする．

キーワード：マルチパーティ計算，秘密分散，時系列情報，セキュリティ，プライバシ保護

Masahiro Nara[1,a)] Takuma Amada[1] Takashi Nishide[2] Hiroshi Doi[3] Hiroshi Yoshiura[1]

Received: December 5, 2016, Accepted: June 6, 2017

Abstract: People often need to use servers to count on time-series information that is generated during activities of people and organizations. In this counting, because a server cannot predict range of information to accept in future, a server needs not only to add values on the counting table but also to extend the table. This paper provides new definition of secure counting on time-series information. Based on this definition, the paper describes methods that hide time-series information by secret sharing and that count on it by multiparty computation. It is shown that tabulated values can be estimated on the bases of observed access patterns even if each value is hidden by secret sharing and multiparty computation. A method that accesses the table exhaustively to update and extend the table is proposed and evaluated to show its confidentiality against access pattern observation while showing its problem of a large communication amount. Based on these analyses, a method that uses recursive Path ORAM to hide access patterns while avoiding exhaustive accesses is proposed and evaluated to show its effects on reducing communication amounts in the order level.

Keywords: multiparty computation, secret sharing, time series information, security, privacy

1. 前書き

個人や組織に関する情報を電子的に集積し，活用することがさかんになっている．このように集積・活用される情報の一種として，個人や組織の活動にともなって時系列的に発生する情報が重要である．たとえば，人の移動履歴やWebサイトの閲覧履歴，重要インフラ施設の状態の履歴があげられる．これらの時系列的な情報には，個人に関わる情報や，組織の機密情報が含まれているため，プライバシや機密の保護が不可欠である．たとえば，移動履歴の利用例として，WiFi基地局が交信した端末のMACアドレスを基地局番号，時刻とともにサーバに送信し，集積・活用することを考える．このような移動履歴からは，端末保持者の職場や住所の位置，生活パターン，保持者間の人間関係を推測することができ，プライバシの侵害につながる可

[1] 電気通信大学
University of Electro-Communications, Chohu, Tokyo 182–8585, Japan
[2] 筑波大学
University of Tsukuba, Tsukuba, Ibaraki 305–8577, Japan
[3] 情報セキュリティ大学院大学
Institute of Information Security, Yokahama, Kanagawa 221–0835, Japan
[a)] m.nara@uec.ac.jp

能性がある．

時系列情報の利用において，表や行列の形で集計したのち分析することが多い．上述した移動履歴の活用の場合，基地局ごとの MAC アドレスの訪問回数を集計することが考えられる．たとえば，MAC アドレス i の基地局 j への訪問回数を行列の形で集計や，時間帯 k における基地局 j への訪問回数を行列の形で集計，MAC アドレス i が基地局 j を時間帯 k において訪問した回数を 3 次元配列の形で集計することなどが考えられる．これらの行列や多次元配列から，各個人の訪問傾向，各基地局付近の時間帯ごとの混雑度などを分析することができる．

本論文では，時系列的に発生する情報を，プライバシや機密を保護しながら，表や行列，多次元配列の形で集計する問題を検討する．まず最初に，この問題を時系列情報のセキュア集計問題として定式化する．次に，問題の解決方法として，時系列情報を秘密分散によって秘匿し，マルチパーティ計算（Multiparty computation: MPC）で集計する方法を提案する．行列や多次元配列に対するアクセスパターン（同一データに対する読み書きの回数，データ間の読み書きの順序など）の秘匿も検討する．

本論文の貢献は以下の 4 点である．

(1) 時系列情報のセキュア集計という一般性の高い興味深い問題を定式化した（2 章）．

(2) 集計表（および索引）の値の加算と表自体の拡張を，アクセスパターンを秘匿しながら実行する新しい MPC プロトコルを提案した（7 章の表 3，9.1 節の表 7）．

(3) Keller らによる Oblivious Random Access Machine（ORAM）の MPC 実現方式 [17] を取り入れるが，時系列データの集計に必要な新しい手法として，(a) ハッシュ関数による索引の圧縮，(b) データのブロック化によるハッシュ値の衝突回避を提案した（9.2 節）．

(4) 上記 (2)，(3) のプロトコルを用い，アクセスパターンを秘匿しながら MPC の通信量を $O(\ell(\log M)^3)$ に削減する時系列情報のセキュア集計方式を提案した（9.2 節）．

2. 時系列情報のセキュア集計問題

まず最初に，WiFi 基地局を介して集計される人の移動履歴を例として，本論文で扱う時系列情報の集計問題を定義する．WiFi 機能を搭載した端末（スマートフォンやゲーム機など）が WiFi 基地局からビーコンを受信すると，端末の MAC アドレスや RSSI（信号の強度）を含むプローブリクエストを基地局に送信する．基地局は，プローブ情報およびその受信時刻を記録するとともに，MAC アドレス，RSSI，プローブリクエスト受信時刻，および基地局番号の情報をサーバに送信する．送信するタイミングとして，端末と交信するたび，1 分間隔，10 分間隔などがありうる．サーバは多数の基地局から受信した情報を，表や行列の形で集計する．

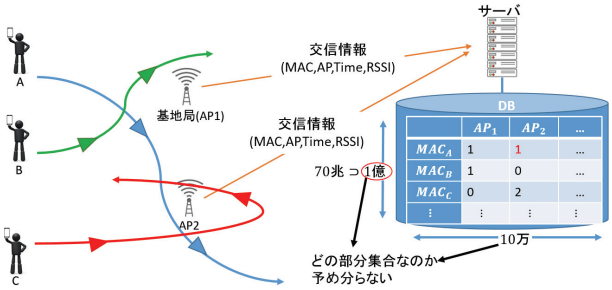

図 1 移動履歴の集計例

Fig. 1 Example aggregation of position history.

東京 23 区内の人の移動履歴を集計する場合を考えると，利用する WiFi 基地局が 10 万局程度 [1]，MAC アドレス数が 1 億程度だと考えられる．しかし，MAC アドレスの可能な個数は，70 兆程度である [2]．サーバは，受け取る MAC アドレスが 70 兆のどの部分集合にあたるのかあらかじめ知ることはできない（図 1）．また，受け取る MAC アドレスの順序もあらかじめ知ることはできない．

本論文において，時系列情報を，以下のように定義する．

(1) イベント情報は $(ID_1, ID_2, \cdots, ID_E, t, V)$ により定義され，1 つ以上の識別子 $ID_i (1 \le i \le E, E \ge 1)$ と時刻 t とその他の情報 V が含まれている．識別子，時刻，その他の情報の値域をそれぞれ $D(ID_i), D(t), D(V)$ と表す．2 つのイベント情報 $(ID_1, ID_2, \cdots, ID_E, t, V)$ と $(ID'_1, ID'_2, \cdots, ID'_{E'}, t', V')$ が $E = E', D(ID_i) = D(ID'_i)(1 \le i \le E), D(t) = D(t'), D(V) = D(V')$ である場合，2 つのイベント情報は同じ型であるとする．

(2) 時系列情報は，イベント情報の列である．

上述した例においては，MAC アドレス，基地局番号，RSSI，プローブリクエスト受信時刻の 4 つ組がイベント情報，イベント情報の列が時系列情報となる．時系列情報の集計問題は以下の (a) から (d) の AND で定義される．なお，以下では，イベント情報の型は 1 つとし，型が複数の場合は別々に収集する．

(a) サーバは $T_1 \le t \le T_2$ の時間帯で時系列情報を受け取り集計する．

(b) サーバは $E' (1 \le E' \le E)$ 個の属性 $(\{ID'_1, \cdots, ID'_{E'}\} \subseteq \{ID_1, \cdots, ID_E\})$ を用い，E' 次元の配列を生成する．本論文では $E' = 1$ のときこの配列を表，$E' = 2$ のとき行列と呼ぶ．

(c) サーバは時系列情報をすべて受け取った後にまとめて集計するのではなく，受け取るごとに集計に加える．

(d) 少なくとも 1 つの $ID_j (1 \le j \le E)$ については，$T_1 \le t \le T_2$ の間に受け取る値が，可能な値 $D(ID_j)$ のうち，どの部分集合であるのかサーバはあらかじめ予測できず，また，受け取る値の順序も予測できない．

時系列情報の集計問題は一般的な問題であり，移動履歴

の集計以外に，Web サイトの閲覧履歴やポイントカードの使用履歴，ネットショッピングの購買履歴の集計などがあげられる．Web サイトの閲覧履歴の集計の場合は，ユーザから一定間隔でイベント情報がサーバに送られてくる．このイベント情報には，ユーザ ID，URL，時刻などが含まれている．サーバは，たとえばユーザ ID と URL の 2 つの識別子を用いて，ユーザの URL への訪問回数を行列の形で集計する．これ以外にサーバが，ユーザの性別や年齢などの情報を保持している場合もあり，性別，あるいは年代別の URL の訪問回数を集計することも考えられる．このとき，URL の可能な値のうちサーバが受け取る値がどの部分集合なのかあらかじめ分からず，値の受け取る順序も分からない．

時系列情報のセキュア集計問題は，以下の AND で定義される．

- 時系列情報のセキュア集計問題は，時系列情報の集計問題である．
- 各イベント情報と生成する多次元配列（表・行列など）が秘匿される．
- 多次元配列へのアクセスパターン（同じデータに対する読み書きの回数，データ間の読み書きの順序）が秘匿される．

想定する攻撃者および上記の秘匿の範囲は応用によって異なる．本論文の以下の章では，攻撃者としてサーバの管理者とサーバへの不正侵入者を想定し，秘匿の範囲はサーバがイベント情報を受け取ってから多次元配列に反映させるまでとする．なお，現実の運用では，通信路の観測者や多次元配列の利用者が攻撃する場合もあり，その場合は，WiFi 基地局からサーバまでの通信路上での暗号化や多次元配列の利用時のプライバシ保護など，周辺部分の対応も必要となる．また，基地局が攻撃者になる可能性もあり，その場合には基地局の認証が必要となる．

3. システムモデル

WiFi 基地局を介して収集される人の移動履歴を例として，システムモデルを説明する（**図 2**）．WiFi 基地局において，秘密分散法を用いてイベント情報を秘匿し，シェアを複数のサーバに送信する．各サーバは，秘密分散法によって秘匿された基地局 j ごとの MAC アドレス i の訪問回数を示す行列のシェアを生成する．すなわち，各サーバは，各基地局からイベント情報のシェアを受信するたびに，マルチパーティ計算（MPC）を用いて，MAC アドレス i と基地局 j に対応する行列の要素に 1 を加算する．サーバは送られてくる MAC アドレス i をあらかじめ知ることはできないため，MAC アドレス i が行列に存在しない場合がある．その場合は MAC アドレス i の行を追加し，1 で初期化する．また，行列のシェアへのアクセスパターンを秘匿するために，ORAM の MPC 実装を用いる場合もある．

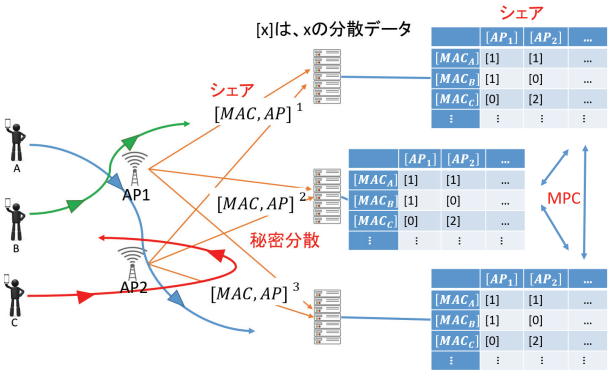

図 2 システムモデルの概要
Fig. 2 Overview of system model.

上記のように，秘密分散法，マルチパーティ計算，ORAM を用いて情報を秘匿しながら，基地局ごとの端末の訪問回数を示す行列を更新する．

4. 先行研究

4.1 秘密分散法

秘匿したい情報を n 個のシェアに分割する手法である．秘密分散法の一種として，(k, n) 閾値秘密分散法 [3], [4] がある．この手法では，秘密情報 s を n 個のシェアに分割して管理する．秘密情報 s を復元するには k 個以上のシェアを集める必要があり，k 個未満では s に関する情報をまったく知ることはできない．

4.2 マルチパーティ計算（MPC）

秘密分散法を用いて秘匿されたデータの関数値のシェアを，データを秘匿したまま計算する手法である [5], [6], [7], [9], [10], [11]．MPC の基礎的なプロトコルとして，通信を必要としない和プロトコルと，通信を必要とする積プロトコルが存在する．これらを組み合わせることで等号判定や大小比較などの上位プロトコルを構成できる．上位プロトコルのコストの評価指標には，並列処理を最大限行った場合の積プロトコルの実行段数であるラウンド数と，積プロトコルの総実行回数である通信量がある．

本論文では上位プロトコルとして**表 1** の方式を用いる．なお，表中の ℓ は，秘密分散の法となる素数のビット長，m はデータ数，n はシェアを保持するサーバ数を表す．濱田らの比較ソートプロトコル [9] の処理コストは，表中の大小比較プロトコル [6] とランダム置換プロトコル [10] を前提として算出した．

4.3 Oblivious Random Access Machine（ORAM）

ORAM は，Goldreich [13] により提案された，データを安全に検索・管理するための手法である．クライアントの鍵を用いて暗号化されたデータを，クラウドなどの外部のサーバに預ける．そして，クライアントから検索が行わ

表 1 本論文で利用する上位プロトコル
Table 1 High-level protocol used in this paper.

プロトコル	ラウンド数	通信量
ビット乱数生成[5]	2	2
ビット分解[5]	25	$93\ell + 47\log\ell$
等号判定[6]	4	$\ell + 4\log\ell$
大小比較[6]	6	$3\ell - 2$
1ビットソート[8]	$n+1$	$(n+1)m$
比較ソート[9]	$n(1+12\log m)$	$nm\{m + 2(3\ell-2)\log m\}$
ランダム置換[10]	n	nm^2

れる度に，アクセスされたデータの格納位置を変更する．ORAM は以下の要件を満たす．

- 暗号化により，サーバはデータの値を知ることができない．
- データの格納位置の変更により，サーバはデータへのアクセスパターン（同一データに対する読み書きの回数やデータ間の読み書きの順序）を推定することができない．

ORAM における一般的なアクセスパターンの秘匿性定義は文献 [15] に示されている．そこでは攻撃者はクライアント以外のすべての者（すなわちサーバおよび外部の攻撃者）としている．攻撃者の観測可能な情報はクライアントがサーバに送信する検索コマンドである．したがって，この観測情報から，攻撃者は，クライアントによるサーバ上のデータへのアクセスパターンを推定しようとする．

クライアントは，サーバ上の記憶領域に対して，識別子を用いて M 回の検索を行う．クライアントから見た検索列は，$\overrightarrow{y} = ((op_1, a_1, data_1), (op_2, a_2, data_2), \cdots, (op_i, a_i, data_i), \cdots, (op_M, a_M, data_M))$ で表される $(1 \le i \le M)$．ここで，M は検索列の長さ，$(op_i, a_i, data_i)$ は i 回目の検索を表す．op_i は $read$ または $write$，a_i は識別子，$data_i$ は op_i が $write$ の場合の書き込みデータである．

検索列 $\overrightarrow{y}$ を実行するためにクライアントがサーバに送る検索コマンドをアクセス列と呼び，$A(\overrightarrow{y})$ で表す．サーバはアクセス列 $A(\overrightarrow{y})$ を観測し，検索列 $\overrightarrow{y}$ のアクセスパターンを推定しようとする．このとき，アクセスパターンの秘匿に関する ORAM の安全性は，「同じ長さの任意の 2 つの検索列 $\overrightarrow{y_1}$，$\overrightarrow{y_2}$ に対して，$A(\overrightarrow{y_1})$ と $A(\overrightarrow{y_2})$ が，クライアント以外のすべての者にとって識別不能である」と定義される．なお，ここでの識別問題とは，検索列の組とアクセス列の組の対応関係の識別問題である．

3 章で述べたように，本システムの全体を MPC で実現するため，ORAM もデータをシェアの形で保持し，検索や格納の処理を MPC で行う必要がある．そこで以下では，代表的な ORAM の例として，MPC での実行方法が明らかになっている Stefanov らの Path ORAM について説明する [14], [16]．

4.4 Path ORAM

データの検索（読み出しと書き換え）が可能な記憶領域がサーバ上にあり，クライアントは M 個のデータを暗号化して，この記憶領域に預託する．クライアントは，キー値を用いて，この記憶領域への検索を行う．以下では，サーバ上のデータはクライアントの鍵で暗号化されているものとする．

4.4.1 サーバの保持するデータ

- 完全二分木（以下，二分木を略す）
  サーバは，二分木を保持する．1 つのノードに配置できるデータ数は，一定数である．そのノードにクライアントから預かったデータを格納する．二分木のサイズはクライアントのデータ数よりも大きく取り，余りの領域にはダミーデータを格納する．
- パス
  二分木の左から x 番目の葉ノードを x と表記する．葉ノード x から根ノードへたどる経路をパスと呼び，$P(x)$ で表す．任意のデータは，1 つ以上のパス上に存在する．

4.4.2 クライアントの保持するデータ

- 索引
  各データの二分木における格納位置を示す表である．格納位置は，当該データが二分木のどのパス上にあるかによって表現され，当該データのポジションと呼ばれる．すなわち，データ d のポジションは，$d \in P(x)$ となる x によって表現される．索引は，各データ d について，（d のキー値，d のポジション）を記憶する．
- スタッシュ
  二分木に格納できなかったデータを保持する．データは二分木のパス $P(x)$ 上か，またはスタッシュに格納される．

4.4.3 検索・書換えプロトコル

例として，キー値 24 のデータを検索するものとする（**図 3**）．

(1) クライアントは索引を参照して，キー値のポジション $x = 15$ を取得し，[0〜二分木の葉ノードの総数 −1] の範囲で一様ランダムに選ばれた新たなポジション x' に変更する．
(2) 取得したポジション $x = 15$ をサーバに送信する．
(3) サーバは受け取ったポジションのパス $P(15)$ をクライアントに送信する．
(4) クライアントはパス $P(15)$ を復号したデータとスタッシュのデータを作業領域に展開し，キー値 24 のデータを検索する．
(5) クライアントは，索引と二分木の整合性を維持しつつ，作業領域内のパス $P(15)$ 上のデータを再配置する．
(6) 再配置後の $P(15)$ 上のデータを再暗号化してサーバに送信し，二分木のパスに上書きする．

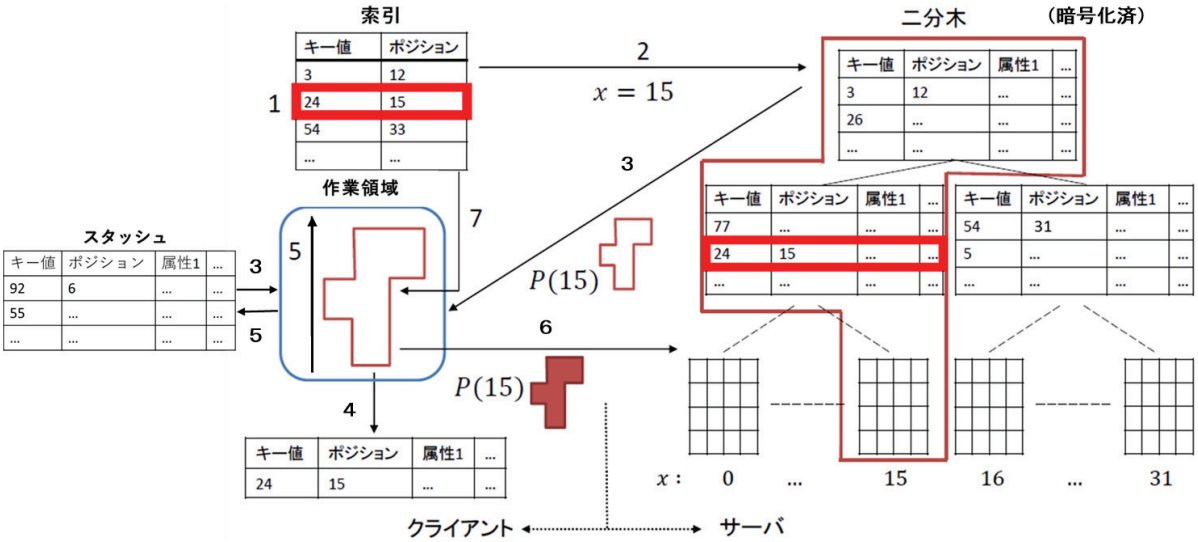

図 3 Path ORAM の検索
Fig. 3 Search protocol in Path ORAM.

書換えの際は，ステップ (4) の実行時に当該データを新しいデータに書き換える．

4.4.4 パスの再配置

ステップ (5) のパスの再配置について詳しく説明する．最初に，各データについて，パスのどのノードに配置可能かを決定する．この決定は，再配置の対象となるパスの番号と，当該データのポジションに依存する．なお，検索対象だったデータのポジションはランダムに変更されている．たとえば，4.4.3 項の例の場合，対象となるパスは $P(15)$ なので，ポジションが 15 のデータは葉ノード以上すなわち $P(15)$ のどのノードにも配置できる．しかし，ポジションが 14 のデータの場合，葉ノードに配置すると，そのデータを検索する際，$P(14)$ 上にないため検索できなくなる．そのため，ポジションが 14 のデータは，$P(14)$ と $P(15)$ の共通ノードすなわち下から 2 番目のノード以上に配置する必要がある．

次に，データを 1 つずつ取り上げ，以下の手順で配置対象パスに再配置する．すなわち，葉ノードから上に向かって，順に，データがノードに配置可能であり，ノードにまだ空があれば，データをノードに配置する．なお，4.4.1 項で述べたように，1 つのノードに配置可能なデータ数はプロトコルのパラメータとしてあらかじめ決まっている．データがノードに配置不能であるか，または，ノードに空きがない場合には，1 つ上のノードに対してデータの配置を試みる．もしルートノードまで試みても当該データが配置できなかった場合には，そのデータはパスに再配置できず，クライアント側のスタッシュに格納される．

4.4.5 再帰的プロトコル

クライアントの保持する索引のサイズは $O(N)$ となり膨大であるため，再帰的実行により定数オーダに削減する．

再帰的実行のための Path ORAM の構成手順を図 4 に示す．初期状態として，クライアントが保持する索引を $Index_1$，サーバが保持する二分木を $ORAM_1$ と表す．クライアントは，$Index_1$ を χ 行単位で分割し，新たな二分木 $ORAM_2$ へ格納する．このとき，$ORAM_2$ のノード数は $ORAM_1$ のノード数の $\frac{1}{\chi}$，クライアントは $Index_1$ の代わりに $Index_2$ を保持する．$Index_2$ は，キー値の範囲と $ORAM_2$ のポジションとの対応を示す．

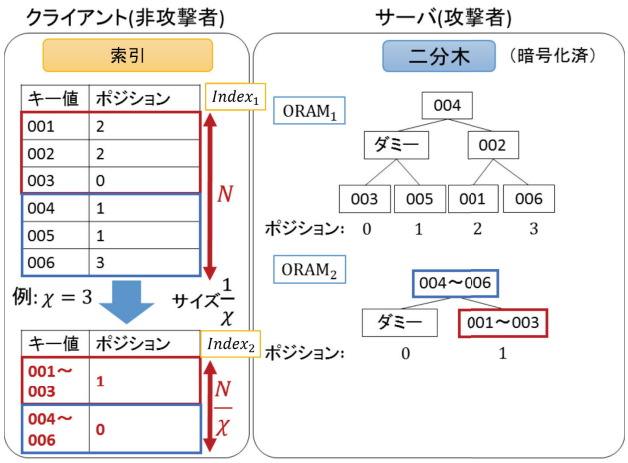

図 4 再帰的 Path ORAM の構成
Fig. 4 Structure of Recursive Path ORAM.

次に，再帰的実行の手順を図 5 に示す．$Index_R$ を用いて $ORAM_R$ から $Index_{R-1}$ を取り出す．この操作を $Index_1$ が出力されるまで再帰的に繰り返し，$Index_1$ を用いて該当データを $ORAM_1$ から出力する．

4.4.6 Path ORAM の安全性

サーバに格納するデータはクライアントの鍵を用いて暗号化されるため，値の秘匿性は暗号方式の秘匿性に帰着する．

アクセスパターン秘匿の安全性を，4.3 節の安全性定義

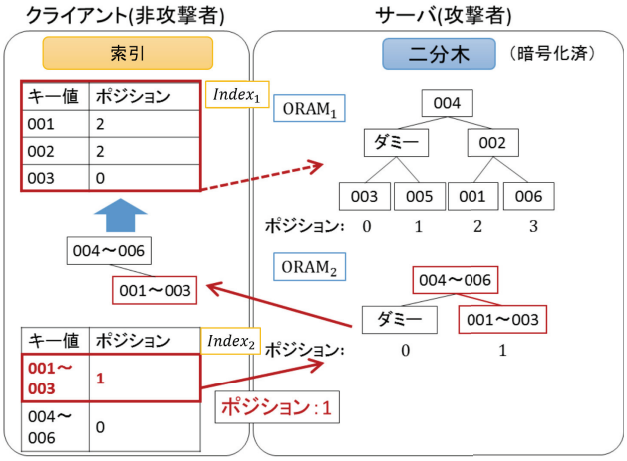

図 5 再帰的 Path ORAM の実行
Fig. 5 Execution of recursive Path ORAM.

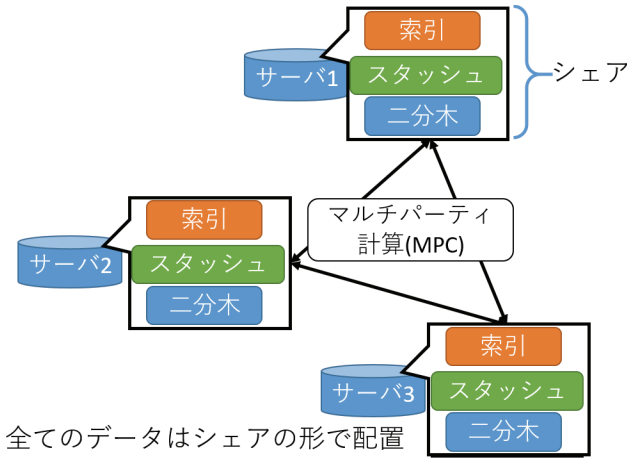

図 6 秘密分散と MPC による Path ORAM の実現
Fig. 6 Implementation of Path ORAM using secret sharing and MPC.

に沿って示す．4.3 節の識別子 a_i を Path ORAM における i 回目の検索のキー値とみなせば，Path ORAM の検索列は 4.3 節の $\overrightarrow{y}$ で表される．4.4.3 項のステップ (1) により，$A(\overrightarrow{y_1})$ と $A(\overrightarrow{y_2})$ は，いずれも，[0～二分木の葉ノードの総数 −1] の範囲の一様ランダムな長さ M の数列であり，識別不能である．したがって，Path ORAM は，アクセスパターン秘匿に関して安全である．

4.4.7 データロスの確率

Path ORAM では 4.4.3 項のステップ (5) において，データをパス上に再配置する際，パス上にデータを格納しきれず，スタッシュにデータが残ることがある．スタッシュへのデータの格納が連続し，スタッシュサイズを超えると，データの取りこぼしが起こる．たとえば，スタッシュサイズが 4 で，$P(1)$ に 4 つまでのデータを配置することができるとき，ステップ (1) において，新たなポジションが $x' = 1$ であることが 9 回以上続くと，1 つ以上のデータがパスにもスタッシュにも配置できなくなる．この取りこぼしが起こる確率をデータロス確率という．

データロスの確率は，二分木の 1 つのノードに格納可能なデータ数（ノードサイズ）とスタッシュに格納可能なデータ数（スタッシュサイズ）に依存する．文献 [14] では，データロス確率 2^{-80}，2^{-128}，2^{-256} 以下を保証するためのノードサイズとスタッシュサイズが示されている．たとえば，2^{-80} 以下を保証するには，ノードサイズが 4 でスタッシュサイズが 89，あるいは，ノードサイズが 5 でスタッシュサイズが 63 となっている．

4.5 MPC による Path ORAM の実現

Keller らは，上記の Path ORAM を秘密分散法と MPC で実現した（**図 6**）[17]．Path ORAM ではクライアントとサーバでシステムを構築するが，Keller らの方式では，サーバがクライアントの役割も兼ねている．想定される攻撃者は，MPC の参加者自身と外部からの侵入者である．

4.5.1 方式

- 秘密分散法によるデータの秘匿
  Path ORAM では，サーバに預けたデータはクライアントの鍵により暗号化されていたが，この手法では秘密分散法を用いてデータを複数のサーバに分散して管理する．
- 秘密分散法による索引とスタッシュの秘匿
  Path ORAM では，クライアントが索引とスタッシュを平文で管理したが，この方式では，参加者自身も攻撃者であるため秘密分散法により秘匿し，シェアの形で管理する．
- MPC による検索・更新
  ある 1 つのサーバがクライアントになり，あるキー値のデータを検索する場合，MPC を用いて索引のシェアからキー値に対応するポジションを検索し，その値を開示する．開示されたポジションに対応するパスのシェアに対して，4.4.3 項のステップ (4)～(6) の処理を MPC で実行する．
- 処理コスト
  この手法では，すべての処理を MPC により実行するため，ラウンド数および通信量で処理コストを見積もる．
- 再帰による通信量の削減
  想定攻撃者である参加者が索引を保持するので，索引を参照する際，アクセスパターンを秘匿するために全探索を行う．そのため，索引の参照に必要な通信量は，Path ORAM に含まれる全データ数を N としたとき，$O(N)$ となってしまう．しかし 4.4.5 項で述べた再帰を用いることで，通信量は $O((\log N)^3)$ となる．つまり，Path ORAM ではクライアントが保持する索引のサイズを削減するために再帰を行ったが，この手法では索引の検索をする際の通信コストを削減するために再帰を行う．

4.5.2 安全性

データの値の秘匿性は，秘密分散法と MPC の秘匿性に帰着する．アクセスパターンの秘匿性およびデータロス確率は，Path ORAM のアクセスパターンの秘匿性およびデータロス確率に帰着する．

5. MPC を用いた時系列情報の安全な集計のためのアクセスパターンの秘匿性

2 章で述べた時系列情報のセキュア集計問題と 3 章のシステムモデルを前提とし，次章以下の提案方式の安全性分析に必要な定義を定める．

5.1 前提

攻撃者は，クライアント以外のすべての者（すなわちサーバおよび外部の攻撃者）とする．アクセスパターンの秘匿を論じる前提として，データの値自体は，秘密分散および MPC によって秘匿されているとし，攻撃者が復号に必要な個数のシェアを入手することはないとする．

検討の対象とする MPC プロトコルについて，以下の前提を設ける．すなわち，MPC プロトコルは，命令と変数から構成される[*1]．MPC プロトコルは変数に対してのみ情報の読み書きを行うとする．たとえば，起動時の引数として外部から情報を受け取る場合は，引数が変数に相当し，その変数から情報を読む．MPC プロトコルは変数から情報を読むことを通じてのみ，外部から情報を入力する[*2]．MPC プロトコルは，その 1 回の実行中に，変数への読み書きを 1 回以上実行する．

変数への 1 回の読み書きは $(op, a, data)$ の形式で抽象化され，op は $read$ または $write$，a は識別子，$data$ は op が $write$ の場合の書き込み情報を表す．MPC プロトコル P の 1 回の実行を $E(P)$ で表し，$E(P)$ における変数への読み書きの列を $A(E(P))$ で表す．$A(E(P))$ は，$((op_1, a_1, data_1), (op_2, a_2, data_2), \cdots, (op_T, a_T, data_T))$ の形式の読み書きの列である[*3]．ここで，T は $E(P)$ における変数への読み書きの回数を表す．

5.2 アクセスの秘匿性

MPC プロトコルのアクセスに関する秘匿性を以下のように定義する．

定義 1：時系列情報の安全な集計のための MPC プロトコル P におけるアクセスの秘匿性

任意の入力に対する P の 1 回の実行を $E_1(P)$，別の 1 回の実行を $E_2(P)$ とする．$E_1(P)$ の入力と $E_2(P)$ の入力は，入力の定義域内の任意の値であり，異なっていてもよい．このとき，P がアクセスの秘匿に関して安全とは，クライアント以外のすべての者にとって，$E_1(P)$ における変数への読み書きの列 $A(E_1(P))$ と，$E_2(P)$ における変数への読み書きの列 $A(E_2(P))$ が識別不能であることである．

定理 1：MPC プロトコルにおけるアクセス秘匿性の十分条件

MPC プロトコル P について，以下の 3 条件がすべて成立するときに，P はアクセスの秘匿に関して安全である．(1) P の用いている秘密分散法が情報理論的に安全である．(2) P の入力に平文が含まれている場合には，P の任意の 2 回の実行 $E_1(P)$，$E_2(P)$ における入力の平文 $IP_1(P)$，$IP_2(P)$ が識別不能である．(3) P の実行途中に開示される平文あるいは出力として開示される平文がある場合には，P の任意の 2 回の実行 $E_1(P)$，$E_2(P)$ において開示される平文 $DP_1(P)$，$DP_2(P)$ が識別不能である．

(証明)

秘密分散法が情報理論的に安全な場合，同じ法を用いて秘密分散した 2 つの平文 v_1 と v_2 のシェア $[v_1]$ と $[v_2]$ は識別不能である[*4]．したがって，条件 (1) が成立すれば，$E_1(P)$ の入力のシェアの部分と $E_2(P)$ の入力のシェアの部分は識別不能であり，実行途中に生成されるシェアも識別不能である．これに条件 (2) が加われば，P への入力全体が識別不能である．さらに条件 (3) が成立すれば，開示された平文を P が読むことで，$A(E_1(P))$ と $A(E_2(P))$ の識別につながる情報を得ることはない．以上から，P は，$A(E_1(P))$ と $A(E_2(P))$ に識別可能な差をもたらす情報を得ることがない．また，P の入出力および実行過程を観測するサーバも，$A(E_1(P))$ と $A(E_2(P))$ を識別するための情報を得ることはない．

なお，(k, n) しきい値秘密分散法が情報理論的に安全な場合，同じ法を用いて秘密分散した平文 v_1 の k 個未満のシェアの集合と平文 v_2 の k 個未満のシェアの集合は識別不能である．そのため，情報理論的に安全な (k, n) しきい値秘密分散法を用いる場合，上記の定理は，k 未満のサーバが結託する場合でも成立する．また，情報理論的に安全な (k, n) しきい値秘密分散法の例としては，Shamir の (k, n) しきい値秘密分散法などがある．

5.3 アクセスパターンの秘匿性

MPC プロトコルのアクセスパターンに関する秘匿性を以下のように定義する．

定義 2：時系列情報の安全な集計のための MPC プロトコ

[*1] 命令と変数以外の要素を含んでいてもいい．

[*2] たとえば引数としての直接的な入力，大域変数を読むことによる間接的な入力がありうる．

[*3] 複数の読み書きの間で実行順序が指定されている場合と任意な場合（たとえば並列実行が可能な場合）がある．したがって，厳密には読み書きの列ではなく，半順序集合である．ここでは，説明の分かりやすさのため，読み書きの列（全順序集合）としたが，半順序集合に一般化することは可能である．

[*4] $[v1]$ と $[v2]$ が識別不能でなければ，たとえば $[v1]$ の平文が $v1$ か $v2$ かを有意な確率で推定できるので，秘密分散法は情報理論的に安全ではない．なお，攻撃者が復号に必要な個数のシェアを入手しないことを前提としている．

ル P におけるアクセスパターンの秘匿性

初期状態（時系列情報を 1 つも受け取っていない状態）からの，P の M 回の実行の列を $E_i(P)(1 \leq i \leq M)$ で表すことにする．P の任意の 2 つの実行列 $E_{1i}(P)$ と $E_{2i}(P)(1 \leq i \leq M)$ を考える．各 $E_{ji}(P)(1 \leq i \leq M, 1 \leq j \leq 2)$ の入力は，入力の定義域内の任意の値であり，異なっていてもよい．このとき，P がアクセスパターンの秘匿に関して安全とは，クライアント以外のすべての者にとって，2 つのアクセス列 $A(E_{1i}(P))$ と $A(E_{2i}(P))(1 \leq i \leq M)$ が識別不能であることである．

上記の定義の妥当性について説明する．2 章で述べた時系列情報のセキュア集計問題が MPC プロトコル P によって実現される場合を考える．ここで P への入力は，2 章 (1) 項で定義したイベント情報のシェアおよび，これまでに生成された集計用配列（2 章 (b) 項）のシェアである．イベント情報が P に時系列的に M 回到着する場合，P は M 回実行され，そのつど，上記のイベント情報および集計用配列のシェアを入力する．P が上記の定義 2 を満たす場合，受け取るイベント情報の列および集計用配列の状態列にかかわらず，P の集計用配列への読み書きの列は識別不能である．すなわち，*read*/*write* の別，識別子（すなわち集計用配列のどの要素への読み書きか），*write* の場合どのデータを書き込んだかが識別不能である．以上のように，上記の安全性定義は，時系列情報のセキュア集計問題において期待するアクセスパターンの秘匿性を表現している．

上記の定義 2 に関して，以下の定理が成立する．

定理 2：MPC プロトコル P におけるアクセスパターン秘匿性の十分条件

P の任意の 2 つの実行列 $E_{1i}(P)$ と $E_{2i}(P)(1 \leq i \leq M)$ を考える．MPC プロトコル P について，次の 3 条件がすべて成立するときに，P はアクセスパターンの秘匿に関して安全である．(1) P の用いている秘密分散法が情報理論的に安全である．(2) P の入力に平文が含まれている場合には，任意の $i(1 \leq i \leq M)$ について，$E_{1i}(P)$ と $E_{2i}(P)$ における入力の平文 $IP_{1i}(P)$，$IP_{2i}(P)$ が識別不能である．(3) P の実行途中に開示される平文あるいは出力として開示される平文がある場合には，任意の $i(1 \leq i \leq M)$ について，$E_{1i}(P)$ と $E_{2i}(P)$ において開示される平文 $DP_{1i}(P)$ と $DP_{2i}(P)$ が識別不能である．

(証明)

条件 (1)〜(3) が成立すれば，定理 1 より，任意の $i(1 \leq i \leq M)$ について，$A(E_{1i}(P))$ と $A(E_{2i}(P))$ は識別不能である．そのため，2 つのアクセス列 $A(E_{1i}(P))$ と $A(E_{2i}(P))$ は識別不能である．

6. 既存技術の組合せによる方式

アクセスパターンの漏洩問題を明確化し，7 章以降の提案手法との比較を可能にするために，本章では既存の MPC プロトコルを組み合わせた方式について述べる．

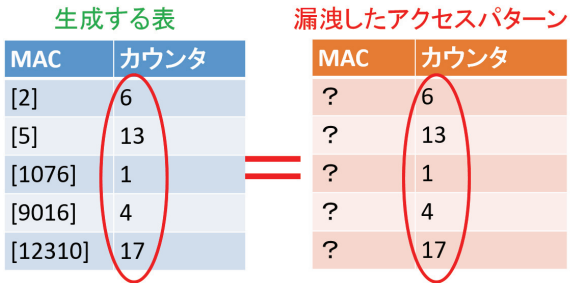

図 7 アクセスパターンと Table
Fig. 7 Access-pattern and Table.

表 2 二分探索利用方式のプロトコル
Table 2 Protocol for the method using a binary search.

```
Input : [MAC] ←送られてきたデータのMAC
Output :  なし
1       二分探索を用いてTableから[MAC]を探索
2       if Tableに[MAC]が存在する場合 then
3         対応するカウンタに1を加算
4       else if Tableに[MAC]が存在しない場合 then
5         Tableに新しい[MAC]の行を追加する
6         対応するカウンタを1に初期化
7       end if
```

WiFi 基地局を介して収集される人の移動履歴を例として，3 章のシステムモデルを具体化する．移動履歴のイベント情報には，MAC アドレス，基地局番号，RSSI，時刻が含まれる．3 章では MAC アドレスと基地局番号の 2 次元で行列を生成したが，ここでは説明を分かりやすくするため，図 7 の左側のように MAC アドレスのみの 1 次元の表で集計することとする．各 MAC アドレスとその出現回数のシェアが集計される．

以下では，値の開示が明記されている場合以外は，値は秘密分散で秘匿されているものとし，計算は MPC で行われる．また，集計表を *Table*，集計表の出現回数の部分をカウンタと呼ぶことにする．

6.1 記法と定義

- N : |D(MAC アドレス)| すなわち世界の MAC アドレスの数（70 兆程度）
- M：集計期間内にサーバが受け取る異なる MAC アドレスの数（東京都 23 区内で集計する場合 1 億程度）
- ℓ：秘密分散法の法となる素数のビット長（$\ell \simeq \log N$）
- $[a]$：値 a のシェア
- row_i：集計表の i 行目
- $d.inf$：データ d の情報 inf の値（たとえば，データ d のポジションは $d.pos$）

6.2 既存技術の組合せによる方式

表 2 に示すように，二分探索を用いて，基地局から送られてきた MAC アドレスを *Table* から見つけ出す．二分探索を行うため，*Table* 内の MAC アドレスはソートされて

いる．送られてきたMACアドレスが$Table$に存在する場合，対応するカウンタに1を加算する．$Table$にMACアドレスが存在しない場合，MACアドレスの順序が維持されるように，送られてきたMACアドレスの行を挿入し，対応するカウンタを1に初期化する．

6.2.1 安全性

データの値は秘密分散し，計算はMPCにより実行するため，データの値の秘匿性は秘密分散とMPCの秘匿性に帰着する．一方，二分探索では，探索対象データ（送られてきたMAC）を，配列の中央のデータ（$Table$内のMACアドレスの中央値）と大小比較し，その結果に従って，配列の中央値未満の半分または中央値より大きな半分について，再帰的処理を行う．そのためには，大小比較の結果を平文として開示する必要がある．大小比較の結果は送られてきたMACアドレスに依存し，5.3節で述べたアクセスパターンの秘匿に関する定理2の条件(3)を満たさないので，アクセスパターンの漏洩の可能性がある．実際，大小比較の結果の開示をトレースすることで，表2のステップ3において，$Table$のどの行のカウンタが加算されたか分かり，同じMACアドレスに対するアクセスの回数が漏洩する．また，ステップ5，6により，新しくMACアドレスを挿入した行も分かる．その結果，$Table$のカウンタの値を推定することができる．なぜなら，MACアドレスを挿入した場所と，1を加算した場所と回数が分かれば，各MACアドレスが送られて来た回数が分かるためである．攻撃者は図7の右側に示す情報を得ることができる．

6.2.2 コスト

メモリ使用量は$2M\ell \simeq 2M \log N$ビットとなる．ラウンド数は$6\log M$，通信量は$2\log M\,(2\ell + 2\log \ell - 1)$．詳細は付録A.1を参照いただきたい．

7. アクセスパターンを秘匿する並列集計方式

$Table$の初期状態として，行番号，MACアドレス，カウンタをそれぞれM行用意する．Mは送られてくるMACアドレス数の上限である（**図8**）．初期値は，行番号が1〜M，MACアドレスがM個のダミーデータ(0)，カウンタがM個の0である．また，先頭のダミーデータを送られてきたMACアドレスで書き換えるために，先頭のダミーデータの行番号を保持するポインタpを用意する．pの初期値は1である．

並列集計方式のプロトコルを**表3**に示す．ステップ1で送られてきたMACアドレス（以下MACとする）が$Table$のi行目に存在するかどうかを判別するフラグx_iを0（存在しない状態）に初期化する．ステップ2〜5でMACが$Table$に存在した場合，フラグxに1を代入し，対応するカウンタに1を加算する．ステップ6で，テーブルに存在すれば，$x = 1$となる．ステップ7でMACが$Table$に存在する場合には$y = 0$となり，存在しない場合にはyにはダミーデータの先頭の行番号が代入される．ステップ8で，ポインタを更新する．存在しない場合に1を加算する．ステップ9〜12でMACが$Table$に存在しない場合，先頭のダミーデータのmacにMACを代入し，対応するカウンタに1を加算する．詳しく説明すると，MACが$Table$に存在する場合には$y = 0$なので，$y = i$はつねに0となる（iは1から始まることに注意）．そのため，ステップ10，11で$row_i.mac$および$row_i.counter$は変化しない．MACが$Table$に存在しない場合には，yにダミーデータの先頭の行番号が代入されているので，ステップ10，11において，row_iがダミーデータの先頭行のときのみ$y = i$となり，等号が成立し，$row_i.mac$にMACが代入され，$row_i.counter$に1が加算される．

#	MAC	カウンタ
1	2	6
2	5	12
⋮	⋮	⋮
p	ダミー	0
⋮	⋮	⋮
M-1	ダミー	0
M	ダミー	0

図8 並列集計方式のTable
Fig. 8 Table of the parallel aggregation method.

表3 並列集計方式のプロトコル
Table 3 Protocol for the parallel aggregation method.

Input :	$[MAC]$ ←送られてきたデータのMAC
Output :	なし
1	$[x_1], \cdots, [x_M] \leftarrow [0]$
2	**for** $i = 1$ to M **do parallel**　　(row_iは$Table$のi行目を示す．)
3	$[x_i] \leftarrow [x_i] + [MAC = row_i.mac]$
4	$[row_i.counter] \leftarrow [row_i.counter] + [MAC = row_i.mac]$
5	**end for**
6	$[x] = [x_1] + \cdots + [x_M]$
7	$[y] \leftarrow [p] \times ([1] - [x])$
8	$[p] \leftarrow [p] + [1] - [x]$
9	**for** $i = 1$ to M **do parallel**
10	$[row_i.mac] \leftarrow [row_i.mac] \times (1 - [y = i]) + [MAC] \times [y = i]$
11	$[row_i.counter] \leftarrow [row_i.counter] + [y = i]$
12	**end for**

7.1 安全性

はじめにデータの値の秘匿性について述べる．各値は秘密分散されており，計算はMPCを用いており，計算の途中で一度も復号しないので，値の秘匿性は秘密分散法とMPCの秘匿性に帰着する．

次にアクセスパターンの秘匿性について述べる．本プロトコルは，5章の安全性定義の前提を満たしている．すなわち，本プロトコルは命令と変数から構成され，変数に対してのみ情報の読み書きを行う．たとえば，表3のステップ1では，変数$[x_1], \cdots, [x_M]$に対して，$[0]$を書き込んでいる．起動時の引数として外部から情報を受け取る場合

は，引数が変数に相当し，その変数から情報を読む．表 3 では，変数 $[MAC]$ から引数の情報を読んでいる．本プロトコルは変数から情報を読むことを通じてのみ，外部から情報を入力する．表 3 の場合，引数変数 $[MAC]$，大域変数 $row_i.mac$，$row_i.counter$ および i からの読みを通じてのみ，外部から情報を入力している．本プロトコルは，1 回の実行中に変数への読み書きを 1 回以上実行する．

本プロトコルでは入力に平文が含まれず，計算の途中および出力で平文を開示しないので，5 章の定理 2 の条件 (2) および (3) が成立する．したがって，本プロトコルで情報理論的に安全な秘密分散法を用いれば，定理 2 の条件 (1)〜(3) がすべて成立するので，アクセスパターンの秘匿に関して安全である．

7.2 コスト

メモリ使用量は $3M\ell \simeq 3M \log N$ ビットとなる．ラウンド数は積 2 回，等号判定 2 回となり，通信量は積 $M+1$ 回，等号判定 $2M$ 回となる．Catrina の等号判定と大小比較プロトコルを例として，積の実行回数で見積もるとラウンド数が 10，通信量が $2M(\ell + 4\log\ell + 1) + 1$ となる（詳細は付録 A.2 を参照）．本方式は，$Table$ の全行に対して同じ処理を行うため，二分探索利用方式に比べると通信量が大きい．東京都 23 区内であれば，1 億行程度に処理を行うこととなる．

7.3 新規性

秘匿された表の全データに並列にアクセスし，アクセスパターンを秘匿しながら，キー値のデータを検索する MPC プロトコルは，下記の形で従来から知られている [12]．

for $i = 1$ to M **do parallel**

$[data_i] \leftarrow [data_1] \times [KEY = key_i]$

end for

$[output] \leftarrow [data_1] + \cdots + [data_M]$

ここで，M はデータ数，KEY は検索したいデータのキー値，$data_i$ は i 番目のデータ，key_i は i 番目のデータのキー値である．提案手法のうち集計表への加算（表 3 ステップ 2〜6）は，上記の従来手法の単純な応用である．

しかし，アクセスパターンを秘匿しながら，表自体の拡張すなわち表になかったデータを追加する MPC プロトコルは従来知られていなかった．提案手法では，集計表への加算の中で，要素の有無を表すフラグ (x) を設定し（表 3 ステップ 3），これを用いることで，ダミー要素の先頭位置を表すポインター (p) に必要に応じた更新を加える（ステップ 8）．そして，p と x を用いて，キー値の要素があれば 0 なければ p となる変数 (y) を設け，全要素への並列アクセスによりダミーの先頭位置に新データを追加することで，アクセスパターンを秘匿しながら表を拡張する．MPC において，アクセスパターンを秘匿しながら，表への値の加算だけではく，表自体の拡張を行う技術は新規である．

8. Path ORAM 利用のための準備

全行にアクセスしないでアクセスパターンを秘匿するために，Keller らの MPC による再帰的 Path ORAM 実現方法 [17]（4.5 節）を用いる．本章では，その準備として，再帰のない Path ORAM 実現方法の利用について述べ，問題点を明らかにする．8.2 節，8.3 節の方式では，$Table$ の各行のデータを 2 分木のノードに格納する．なお，Path ORAM では，データロスの確率を小さくするために，実データの 10 倍程度のデータが格納できるように 2 分木のサイズを設定しているので，ここでも同様にする．

8.1 記法と定義

- n：サーバの数
- Z：二分木のノード
- $Z.size$：1 つのノードに入るデータの数．なお，各データは 5 章の $Table$ の 1 行に相当．
- H：二分木の高さ（$H = \lceil \log\left(\frac{10M}{Z.size}\right) \rceil$）
- R：再帰の深さ
- χ：再帰の際にまとめる索引の行数
- $P(x)$：ポジション x のパス，すなわち二分木のルートと，左から x 番目の葉ノードとを結ぶパス
- St：スタッシュ
- $St.size$：スタッシュのサイズ
- W：作業領域

8.2 再帰のない Path ORAM 利用方式

初期状態として二分木の各ノードに，MAC アドレスのダミー (0) と，カウンタの 0，ダミーのポジション (0) からなるデータを $Z.size$ 個ずつ格納する．また，二分木に対応する索引として，行番号（1〜M）と，MAC アドレスのダミー (0)，ダミーポジション (0) を M 行分用意する．なお，二分木と索引の間でダミーのポジションが一致する必要はない．スタッシュのサイズは，文献 [14] に従うものとする．

本方式は，(1) 送られてきた MAC アドレスに対して索引を検索し，当該 MAC アドレスがあれば対応するポジションを出力し，なければ索引に当該項目を設けるプロトコル（索引参照プロトコル），(2) ポジションを用いて 2 分木からパスを取り出し，MAC アドレスに該当するデータがあればカウンタを 1 つ加算し，なければダミーデータを当該 MAC アドレスのデータに変えてカウンタを 1 に初期化するプロトコル（カウンタ加算プロトコル），(3) パスの再配置を行うプロトコル（パス再配置プロトコル）から構成される．

(1)〜(3) のうち，Keller らはパス再配置プロトコルのみを示しており，索引参照プロトコルとカウンタ加算プロト

表 4 最下層共通先祖ノード（LCA）探索プロトコル
Table 4 Protocol for searching least common ancestor.

Input : $[a]$ ←再配置対象データのポジション POS ←再配置対象パスの番号
$[e]$ ←ダミーデータであるかどうかを示すビットの値(ダミーなら[1])
Output : $[r_0],\cdots,[r_{H-1}]$ ← $e=0$なら最下位共通親ノード$[r_i]=[1]$となり,$e=1$ならすべて[0]
1 $[a]_B \leftarrow \mathrm{BitDec}([a], H)$
2 $POS_B \leftarrow POS$のビット分解(ローカル計算)
3 $[c] \leftarrow 1-[e]$
4 **for** $i = 0$ *to* $H-1$ **do**
5 $[t] \leftarrow 1 - \mathrm{XOR}([a_i], POS_i)$
6 $[r_i] \leftarrow [c] - [c] \times [t]$
7 $[c] \leftarrow [c] \times [t]$
8 **end for**

表 5 配置階層算出（ComputePos）プロトコル
Table 5 Protocol for computing position.

Input :POS ←再配置対象パスの番号 $[w_i]$ ←作業領域上のデータ $[e]$ ←ダミーデータかどうか
Output :$[a]$ ←パスに格納できたかをどうかを示す1ビットの値(スタッシュに格納する場合[1],
パス上に配置する場合[0])
$[H_E]$ ←データを格納するパス上のノードの階層
1 $[r_0],\cdots,[r_{H-1}] \leftarrow \mathrm{LCA}([w_i.pos], POS, [e])$
2 $[b_0],\cdots,[b_{H-1}] \leftarrow [0]$ H_Eをビットの配列で示した値となる。ここでは0で初期化する
3 **if** データをパスに格納できる場合 **then**
4 $[b_i] = [1]$ iはデータを格納する階層を示す
5 **end**
6 $[a] \leftarrow 1 - [e] - \sum_j [b_j]$
7 $[H_E] \leftarrow \sum_j [b_j] \times j$

表 6 パス再配置プロトコル
Table 6 Protocol for path relocation.

Input :POS ←再配置対象パスの番号
$[w_0],\cdots,[w_{m-1}]$ ←作業領域Wに展開されている全データ
Output :なし
1 **for** $i = 0$ to $m-1$ **do**
2 $([a_i],[H_i]) \leftarrow \mathrm{ComputePos}(POS, [w_i], [w_i.dummy()])$
3 **end for**
4 ダミーデータの配置階層を決定する
5 $\mathrm{Shuffle}([w_i],[H_i]))$
6 **for** $i = 0$ to $m-1$ **do**
7 $H_i \leftarrow \mathrm{Reveal}([H_i])$
8 $[w_i]$をH_i階層目のノードに配置する
9 **End**

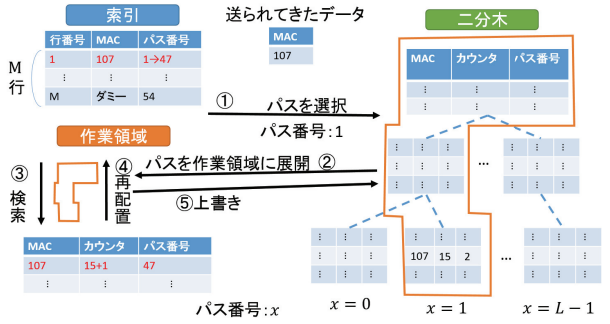

図 9 Path ORAM を利用した集計方式
Fig. 9 Aggregation method using Path ORAM.

コルは示していない．そこで，これらについては 9 章で新たに提案する．

以下では，**表 4**，**表 5**，**表 6** を用いて，パス再配置プロトコルを説明する．表 6 はメインプロトコルであり，表 4，表 5 はそのサブプロトコルを示している．

最下層共通先祖ノード探索プロトコル（表 4）は，配置対象データのポジション a と，再配置対象パスの番号 POS を入力し，データがパスの何階層以上に配置可能かを求める．これは，葉ノード a と葉ノード POS の共通祖先ノードのうち最下層にあるノード（Least Common Ancestor）を求めることに相当する．ステップ 1，2 で 2 つの葉ノードのポジションをビット分解する．ステップ 4〜8 で最上位ビットから順に同じ値かを判定する．違う値となる 1 つ手前の階層が LCA となる．**図 10** の場合は，葉ノード 0 と 3 の LCA は $r_1=1$，$r_0=r_2=r_3=0$ より，2 階層目のノードとなる．データがダミーの場合，$r_1=r_2=r_3=0$ となる．

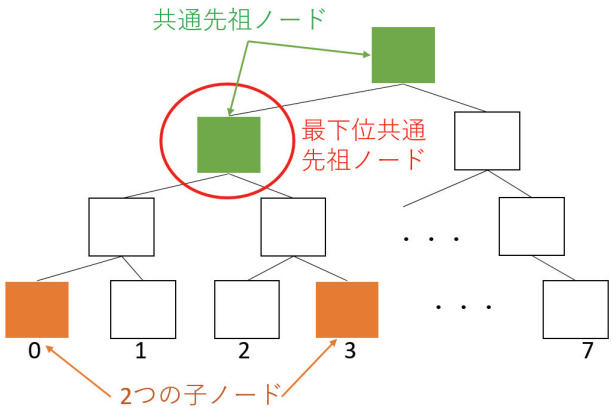

図 10 最下層共通先祖ノード
Fig. 10 Least Common Ancestor.

配置階層算出プロトコル，パス再配置プロトコルについては，説明を分かりやすくするため，簡略したアルゴリズムを示している．詳細については，付録 A にて述べる．

配置階層算出プロトコル（表 5）は，作業領域 W にある 1 つのデータ w_i をパス $P(POS)$ 上の何階層目に配置するかを決定する．H_E は配置する階層を表し，ルートに配置する場合は $H_E=0$，葉ノードの場合 $H_E=H-1$ とする．また，配置できたことを表す a を 0 にする．配置できなかった場合はスタッシュに格納する．その場合，$a=1$，$H_E=0$ とする．ステップ 1 で，最下層共通先祖ノード LCA を求める．ステップ 3〜5 で，データを配置する階層 i を求め，$b_i=1$ とする．配置できなかった場合は，$b_i=0$ $(0 \le i \le H-1)$ のままである．ステップ 3〜5 の判定は，データの配置可能ノード（LCA）と階層のノードの空きに依存する．ステップ 6 で，パス上にデータを配置できた場合，またはデータがダミーの場合には $a=0$，データが実データかつ配置できなかった場合には $a=1$ とする．ステップ 7 で配置できた階層を H_E に代入する．図 10 の例では，ルートの直下のノードに配置可能であり，そのノードに空きがあった場合には，配置することに決定し，$H_E=1$ とする．w_i がダミーデータの場合は $a=0$，$H_i=0$ となる．

パス再配置プロトコル（表 6）は，作業領域内の全データ w_0〜w_{m-1} をパス $P(POS)$ 上に再配置する．なお，$m = Z.sizeH + St.size$ である．ステップ 1〜3 で，配置階層算出プロトコル（ComputePos）を用いて，作業領域上の各データ w_i を $P(POS)$ の何階層目に配置するか，または配置できないかを判定する．その結果，$([w_i],[H_i])$

が m 組算出される．ここで，H_i は，w_i が格納される階層を示す．なお，w_i がダミーデータの場合は，$H_i = 0$ となる．ある階層のノードが実データで埋まらない場合，ダミーデータを配置する必要がある．そこで，ステップ 4 で，ダミーデータを配置する階層を決める．ステップ 5 でデータをシャッフルし，再配置前と再配置後のデータの関係を追跡不能とする．ステップ 6〜8 で配置階層を公開し，再配置を行う．

8.2.1 安全性

索引参照プロトコルとカウンタ加算プロトコルについては，9 章で提案した後，安全性を分析する．なお，9 章で提案する索引参照プロトコルは，ポジション（POS）として，[0, 二分木の葉ノード総数 −1] の一様ランダムな乱数を出力するように設計する．パス再配置プロトコルおよびそのサブプロトコルは Keller らの方式と同様である．パス再配置プロトコルの入力は，POS および作業エリア内のデータ (実データ, ダミーデータ) であり，出力は再配置されたパスおよびスタッシュの新たな状態であるが，これらも Keller らの方式と同様である．以上から，パス再配置プロトコルとその入出力はすべて Keller らの方式と同様であるため，その安全性は Keller らの方式に帰着する．

パス再配置プロトコルのアクセスパターンについて，5 章の安全性定義の観点から分析する．この分析は，7.1 節で述べた並列集計方式の安全性証明と同じ手順を取る．まず，パス再配置プロトコルは，5 章の安全性定義の前提を満たしている．すなわち，命令と変数から構成され，変数に対してのみ情報の読み書きを行う．また，変数から情報を読むことを通じてのみ，外部から情報を入力する．1 回の実行中に変数への読み書きを 1 回以上実行する．パス再配置プロトコルの入力する平文はポジション POS であるが，これは一様ランダムな乱数であるため，識別不能であり，条件 (2) を満たす．パス再配置プロトコルは，表 6 のステップ 7 においてのみ，平文 H_i を開示する（$0 \leq i \leq n-1$）．H_i は，再配置の対象となるデータ i を階層の何段目に配置するか（データ i の配置レベル）を表す．データ i が階層の何段目以上に配置可能か（配置可能レベル）は，再配置の対象となるパスの番号とデータ i のポジションとの関係で決まり，実際の配置レベルは，他のデータの配置可能レベルとの関係で決まる．

再配置の対象となるパスの番号は，[0〜二分木の葉ノード総数 −1] の一様乱数である．検索対象以外の各データのポジションも [0〜二分木の葉ノード総数 −1] の一様乱数である．また，検索対象データの新しいポジションも，上記区間の一様乱数である．そこで，パス再配置プロトコルの任意の 2 つの実行系列において，データの配置レベル H_i に識別可能な差をもたらす情報は存在しないため，H_i は識別不能である．以上から，定理 2 の条件 (2) が成立する．そこで，パス再配置プロトコルで情報理論的に安全な秘密分散法を用いれば，定理 2 の条件 (1)〜(3) がすべて成立するので，アクセスパターンの秘匿に関して安全である．

データロスは，パスの再配置の際に生じるが，パス再配置プロトコルとその入出力はすべて Keller らの方式と同様であるため，その安全性は Keller らの方式すなわち Path ORAM のデータロス確率に帰着する．

8.2.2 コスト

ここでは，パス再配置プロトコルのコストを分析する．

スタッシュサイズ $\alpha \log M$ とし，作業領域に展開するデータの総数を $m = Z.sizeH + \alpha \log M$ とする．最下層共通先祖ノード探索プロトコルでは，通信量は積が $2H$ 回，ビット分解が 1 回，XOR が H 回となり，ラウンド数はステップ 5〜7 にかけての $c \times t$ と XOR を並列にできるため，積が H 回，ビット分解 1 回となる．配置階層算出プロトコルとパス再配置プロトコルのコストは，付録の**表 A·1** と**表 A·2** に基づいて算出する．表 A·1 の配置階層算出プロトコルは並列に処理を行うことができないため通信量とラウンド数ともに，積が $4H$ 回，XOR が $4H$ 回，最下層共通先祖ノード探索プロトコルが 1 回となる．表 A·2 のパス再配置プロトコルの通信量は積が m 回，reveal が m 回，ソートが 2 回，ランダム置換が 1 回，配置階層算出プロトコルが m 回となり，ラウンド数はステップ 5〜9 と 11〜19 で処理を並列に行うことができるため，積が 1 回，reveal が 1 回，ソートが 2 回，ランダム置換が 1 回，配置階層算出プロトコルが m 回となる．

9. 再帰的 Path ORAM 利用方式

本章では，9.1 節において，Keller らの方式で欠けていた索引参照プロトコルとカウンタ加算プロトコルを新たに提案し，安全性とコストを分析する．その後，索引参照プロトコルとカウンタ加算プロトコル，および 8.2 節で述べたパス再配置プロトコルを部品として，9.2 節にて，再帰的 Path ORAM 利用方式を提案する．

9.1 索引参照およびカウンタ加算プロトコル

索引参照プロトコル（**表 7**）は，並列集計方式のプロトコル（表 3）を拡張したものである．並列集計方式において $row_i.counter$ を加算する代わりに，本プロトコルでは $row_i.pos$ を新たなポジション（$newPOS$）で書き換える．送られてきた MAC が索引に存在した場合，ステップ 7 で，対応するポジション $row_i.pos$ が POS_i に代入される．また，ステップ 10〜12 において，$[x] = [1]$, $[y] = [0]$, $[p] = [p]$ となる．送られてきた MAC が索引に存在しなかった場合には，ステップ 10〜12 において，$[x] = [0]$, $[y] = [p]$, $[p] = [p] + 1$ となる．また，ステップ 15 では，先頭のダミーデータのポジション $row_i.pos$ が POS_i に代入される．ステップ 18，19 で，POS が公開される．

カウンタ加算プロトコル（**表 8**）のステップ 1 は，送られ

表 7 索引参照プロトコル
Table 7 Protocol for index reference.

Input : $[MAC]$ ←送られてきたデータのMAC
Output : POS←再配置対象パスの番号 $[newPOS]$ ←生成した新しいポジション
1 ランダムなビット値$[r_i](0 \le i \le L)$を生成
2 $[newPOS] = \sum_{i=0}^{L}([r_i] \times 2^i)$
3 $[x_1], \cdots, [x_M] \leftarrow [0]$
4 $[POS_1], \cdots, [POS_M] \leftarrow [0]$
5 **for** $i = 1$ *to* M **do parallel**
6 $[x_i] \leftarrow [x_i] + [MAC = row_i.mac]$
7 $[POS_i] \leftarrow [POS_i] + ([row_i.pos] \times [MAC = row_i.mac])$
8 $[row_i.pos] \leftarrow [newPOS] \times [MAC = row_i.mac] + [row_i.pos] \times (1 - [MAC = row_i.mac])$
9 **end for**
10 $[x] = [x_1] + \cdots + [x_M]$
11 $[y] \leftarrow [p] \times ([1] - [x])$
12 $[p] \leftarrow [p] + [1] - [x]$
13 **for** $i = 1$ *to* M **do parallel**
14 $[row_i.mac] \leftarrow [row_i.mac] \times (1 - [y = i]) + [MAC] \times [y = i]$
15 $[POS_i] \leftarrow [POS_i] + [row_i.pos] \times [y = i]$
16 $[row_i.pos] \leftarrow [newPOS] \times [y = i] + [row_i.pos] \times (1 - [y = i])$
17 **end for**
18 $[POS] = [POS_1] + \cdots + [POS_M]$
19 $[POS]$を公開

表 8 カウンタ加算プロトコル
Table 8 Protocol for counter increment.

Input : $[MAC]$ ←送られてきたデータのMAC $[newPOS]$ ←生成した新しいポジション
$[x]$ ←索引参照プロトコルで生成したフラグで索引にMACがあれば[1],なければ[0]
Output : なし
1 作業領域Wに$P(POS)$とスタッシュStを展開
2 **for** $i = 1$ *to* $Z.sizeH + St.size$ **do parallel**
3 $[w_i.counter] \leftarrow [w_i.counter] + [MAC = w_i.mac]$
4 $[w_i.pos] \leftarrow [newPOS] \times [MAC = w_i.mac] + [w_i.pos] \times (1 - [MAC = w_i.mac])$
5 $[w_i.dm] \leftarrow [w_i.mac = 0]$ ダミーであるかどうかを示すフラグ
6 **end for**
7 w_iを$w_i.dm$に基づいて昇順に1ビットソート
8 **for** $i = 2$ *to* $Z.sizeH + St.size$ **do**
9 $[w_i.dm] = [w_i.dm] - [w_{i-1}.dm]$
10 **end for**
11 **for** $i = 1$ *to* $Z.sizeH + St.size$ **do parallel**
12 $[w_i.counter] \leftarrow [w_i.counter] + (1 - [x]) \times [w_i.dm]$
13 $[w_i.pos] \leftarrow [newPOS] \times (1 - [x]) \times [w_i.dm] + [w_i.pos] \times [x] \times [w_i.dm]$
14 $[w_i.mac] \leftarrow [MAC] \times (1 - [x]) \times [w_i.dm] + [w_i.mac] \times [x] \times [w_i.dm]$
15 **end for**

てきたMACアドレスのポジションPOSに対応するパスとスタッシュを作業領域Wに展開する．$Z.sizeH + St.size$は展開したデータの総数であり，w_iは各データを表す．ステップ2〜6で，送られてきたMACアドレスに対応するカウンタに1を加算し，ポジションを新しいポジション$newPOS$で書き換える．また，ダミーデータであるかどうかを示すフラグの更新をする．ステップ7〜10では，ダミーデータの先頭において，$[w_i.dm] = [1]$とし，それ以外において，$[w_i.dm] = [0]$とする．ステップ11〜15では，送られてきたMACアドレスが存在しない場合，先頭のダミーデータのカウンタに1を加算し，ポジションを新しいポジション$newPOS$に書き換え，ダミーデータを送られてきたMACに書き換える．

9.1.1 安全性

最初に値の秘匿性について述べる．索引参照プロトコルは，ステップ1，2，7，15，18，19を除くと，7章（表3）で述べた並列集計方式と同様である．ステップ1，2，7，15，18はMPCによる計算であり，値の開示を含まないので，値の秘匿性は秘密分散とMPCの秘匿性に帰着する．ステップ19は，データの存在するパス番号POSを開示するが，パス番号を開示して，そのパスからデータを探索する点は，Kellerらの方式と同様であるため，Kellerらの方式と同様に値は漏洩しない．カウンタ加算プロトコルはMPCによる計算であり，値の開示を含まないので，値の秘匿性は秘密分散とMPCの秘匿性に帰着する．

次にアクセスパターンの秘匿性について述べる．索引参照プロトコルとカウンタ加算プロトコルは，5章の安全性の前提を満たしている．索引参照プロトコルの入力には平文が含まれない．また，開示する平文（POS）は，送られてきたMACに依存しない一様ランダムな乱数であるため任意の2つのアクセス列において識別不能である．そこで，索引参照プロトコルにおいて情報理論的に安全な秘密分散法を用いれば，定理2の条件(1)〜(3)がすべて成立し，アクセスパターンの秘匿に関して安全である．カウンタ加算プロトコルの入力には平文が含まれず，計算の途中で平文を開示しないので，情報理論的に安全な秘密分散法を用いれば，アクセスパターンの秘匿に関して安全である．

9.1.2 コスト

索引参照プロトコルの通信量は積が$8M + H + 1$回，等号判定が$2M$回，1ビット乱数生成がH回となる．ラウンド数については，2カ所のforループ内の並列処理を考慮すると，積が3回，等号判定が2回，1ビット乱数生成が1回となる．

カウンタ加算プロトコルの通信量は積が$7m$回，等号判定が$2m$回，1ビット安定ソートが1回となり，ラウンド数は，forループ内の並列処理を考慮すると，積が3回，等号判定が1回，1ビット安定ソートが1回となる．なお，mは8.2.2項で述べたように，$Z.sizeH + \alpha \log M$である．

この2つのプロトコルとPath再配置プロトコルのコストにより，再帰のないPath ORAMのメモリ使用量が分かる．索引では行番号，ポジション，MACアドレスの3属性を保持しており，各要素のビット長はℓビットであるため，このメモリ使用量は$3\ell M$ビットとなる．二分木とスタッシュはMACアドレス，ポジション，カウンタの3属性を保持しているため，$2^{t-1} - 1 < \frac{10M}{Z} \le 2^t - 1$としたとき，このメモリ使用量は$3\ell(2^t - 1)Z + 3\alpha\ell \log M$となる．したがって，再帰のないPath ORAM利用方式全体における具体的な処理コストはサーバ数をnとするとメモリ使用量は，$3M\ell + 3\ell(2^t - 1)Z + 3\alpha\ell \log M$ビットとなる．

また，パス再配置プロトコルと合わせて，ラウンド数が$(H + 24)m + 24n \log m + 4n + 9$で，通信量が$2M(\ell + 4log\ell + 4) + m[n\{3m + 4(3\ell - 2)\log m + 1\} + \ell(47 \log \ell + 98) + 8 \log \ell + H + 5] + H + 1$となる．

再帰のないPath ORAM利用方式は，索引参照プロトコルにおいて並列集計方式と同様に処理を行い，さらに，カウンタ加算およびパス再配置を行うので，ラウンド数，通信量ともに並列集計方式より大きい．特に，索引参照プロトコルは索引の全行にアクセスするため通信量が大きい．

9.2 再帰的 Path ORAM 利用方式

索引に対するアクセスの回数を減らすため，再帰的 Path ORAM を用いる．Keller らの方式を参考に，時系列情報のセキュア集計に対応するために，9.1 節で述べたオリジナルの索引参照プロトコルおよびカウンタ加算プロトコルを用い，さらに下記の改良を加える．

Keller らの方式では，索引がキー値（MAC アドレス）についてソートされていることを前提とし，索引を χ 行ずつまとめて小さくしていた（図 4）．しかし，時系列情報のセキュア集計では，キー値は巨大な集合の部分集合であり，どの部分集合かはあらかじめ分からず，また，キー値の到着順序もあらかじめ分からない．そのため，9.1 節の索引参照プロトコルでは，MAC アドレスについてソートされておらず，到着順に並んでおり，Keller らの再帰方式を利用できない．到着した MAC アドレスだけを登録し，かつソートを維持しようとすると，6.2 節の二分探索利用方式のように，アクセスパターンから，MAC アドレスごとの到着回数が漏洩する．

一方，あらかじめ世界の MAC アドレスの総数 N 個分の索引を設け，MAC アドレス順にソートしておけば，上記の問題は解決できる．しかし，$N = 70$ 兆であるため，索引のメモリ容量を試算すると 100 TB となる．今回は MAC アドレスだけの索引としたが，実運用では，MAC アドレス × 基地局あるいは，MAC アドレス × 時間帯などの索引を設けることがあり，その場合には，索引のサイズがさらに数百～数 10 万倍に増大する．

そこで，ソートの維持とアクセスパターンの秘匿を可能とし，かつメモリ容量を節約するために，MAC アドレスをキー値とするのではなく，MAC アドレスのハッシュ値をキー値として索引を構成する．基地局にハッシュ関数計算機能を設ける．基地局は，これまでのイベント情報 (MAC アドレス, RSSI, ⋯ 基地局番号) に加えて，MAC アドレスのハッシュ値を秘密分散してサーバに送る．サーバは MAC アドレスのハッシュ値をキー値として索引への検索・追加を行う．

上記のようにハッシュ関数を用いて索引を構成する場合，ハッシュ値の衝突が問題となるので，さらに 2 つの対策を行う．1 つ目の対策として，索引のサイズを M から βM に拡大する（β は 1 以上の定数，$\beta \log M$ は正整数である）．

2 つ目の対策はデータのブロック化である．まず，ハッシュ値の衝突数の上限を MAX と想定する．ハッシュ値の衝突が MAX より多い場合にはデータロスとみなす．MAX は，M 種類の MAC アドレスのハッシュ値が値域 1～βM において衝突する回数の上限である．

8.2 節の再帰のない Path ORAM 利用方式では，1 つのデータは (MAC アドレス, カウンタ, ポジション) の形式を取っていた．ここで i 番目のデータを $data_i$ とし，以下のように j 番目データブロック $DATA_j$ を定義する．

$$DATA_j = \{data_{j_i} | j_i \in S_1 \cup S_2,$$
$$|S_1 \cup S_2| = MAX, S_1 \cap S_2 = \phi,$$
$$Hash(data_{j_p}.mac) = Hash(data_{j_q}.mac)$$
$$for\ j_p, j_q \in S_1,\ data_{j_k}\ is\ dammy\ for\ j_k \in S_2\}$$

ただし，$Hash()$ はハッシュ関数．

すなわち，データブロック $DATA_j$ は，MAC アドレスのハッシュ値が等しい MAX 個のデータの集合である．ハッシュ値が等しいデータが MAX 個未満の場合には，残りをダミーデータとする．

データの二分木への配置および再配置はデータブロック単位で行う．すなわち，同じデータブロック内の MAX 個のデータは，つねに二分木の同じノードに格納するか，またはすべてスタッシュに格納する．8.2 節の再帰のない Path ORAM 利用方式では，二分木の各ノードに Z 個のデータを格納したが，本節の再帰的 Path ORAM 利用方式では，各ノードに Z 個のデータブロックを格納する．そのため，データを格納する二分木の容量は MAX 倍になる．スタッシュのサイズも，再帰のない Path ORAM 利用方式では $\alpha \log M$ 個のデータ分であったが，再帰的 Path ORAM 利用方式では，$\alpha \log M$ 個のデータブロック分とするため，MAX 倍になる．

索引，および索引を格納する二分木については，Keller らの方式と同様であり，索引参照プロトコルは再帰のない Path ORAM 利用方式（表 7）と同様である．データの格納，カウンタの加算，パスの再配置については，データを格納する二分木の容量が MAX 倍になる点，カウンタ加算プロトコル（表 8）が作業領域に展開するデータ数が MAX 倍になる点，パス再配置プロトコル（表 6）がデータではなくデータブロック単位となる点を除いて，再帰のない Path ORAM 利用方式と同様である．

部分索引展開プロトコル（**表 9**）は，次の再帰で利用する索引を取り出す．ステップ 1 では，索引を格納した二分木とスタッシュからデータ $w_i (1 \le i \le Z.sizeH + St.size)$ を作業領域に展開する．たとえば，図 5 において，$ORAM_2$ のパス 1 から 2 つのデータ（001～003 の部分索引）と（004～006 の部分索引）を作業領域に展開する．ステップ 2 では，次の再帰で使う索引 $Index$ を初期化する．ステップ 3～6 では，次の再帰で使用する索引を w_i から取り出す（ス

表 9 部分索引展開プロトコル
Table 9 Protocol for expading partial index.

Input : $[Hash]$ ←送られてきたデータのハッシュ値 $[newPOS]$ ←生成した新しいポジション
Output : $[Index]$ ←次の再帰で使用する索引
1 作業領域Wに$P(POS)$とスタッシュStを展開
2 $[Index_1], \cdots, [Index_M] \leftarrow [0]$
3 **for** $i = 1$ *to* $Z.sizeH + St.size$ **do parallel**
4 $[Index_i] \leftarrow w_i.index \times [w_i.hash_{bottom} \le Hash \le w_i.hash_{top}]$
5 $[w_i.pos] \leftarrow [newPOS] \times [w_i.hash_{bottom} \le Hash \le w_i.hash_{top}]$ $+[w_i.pos] \times (1 - [w_i.hash_{bottom} \le Hash \le w_i.hash_{top}])$
6 **end for**
7 $[Index] = [Index_1] + \cdots + [Index_M]$

表 10 再帰的 Path ORAM 利用プロトコル
Table 10 Protocol using recursive Path ORAM.

Input : MAC ←送られてきたデータのMAC
Output : なし
1 for $r = R$ to 2 do rは何回目の再帰なのかを示す
再帰r回目の$Index_r$(索引)$Tree_r$(二分木)に対してハッシュ値で
・索引参照プロトコル
2 ・部分索引展開プロトコル
・パス再配置プロトコル
を実行
3 次の再帰で利用する$Index_{r-1}$(索引)を出力
4 end for
5 $Index_1$に対してハッシュ値で索引参照プロトコルを実行
6 W_1(作業領域)に対してMACでカウンタ加算プロトコルを実行
7 W_1と$Tree_1$に対してハッシュ値でパス再配置プロトコルを実行

テップ 4)．ここで，$w_i.hash_{bottom}$ と $w_i.hash_{top}$ は，w_i の保持する部分索引の下限と上限である（図 5 の場合は 1 と 3)．また，w_i の二分木 $ORAM_2$ におけるポジションを $newPOS$ に変更する（ステップ 5)．ステップ 7 は，次の再帰で使う索引 $Index$ を求める．

再帰的 Path ORAM 利用方式のプロトコルを**表 10** を用いて説明する．ステップ 1〜4 でハッシュ値をキーとして再帰的に Path ORAM 利用方式を実行する．ステップ 5 は，データを格納する二分木 $Tree_1$ に対する索引の検索であり，MAC アドレスの代わりにハッシュ値をキーとする点以外は，再帰のない Path ORAM 利用方式と同様である．ステップ 6 は，作業領域に展開されたデータのうち，MAC アドレスが送られてきた MAC アドレスに等しいものを選択し，そのカウンタを 1 加算する．Path ORAM 利用方式のカウンタ加算プロトコル（表 8）を用いるが，作業領域内のデータ数は MAX 倍になっている．ステップ 7 は，Path ORAM 利用方式のパス再配置プロトコル（表 6）を用いるが，データの代わりにデータブロック単位で再配置する．また，スタッシュは $\alpha \log M$ 個のデータの代わりに，同じ個数のデータブロックを格納できるサイズとする．

9.2.1 安全性

本方式は，Keller らの再帰的 Path ORAM の MPC 実行方式に，9.1 節で述べた索引参照プロトコルとカウンタ加算プロトコルを加え，MAC アドレスの代わりにハッシュ値をキーとして用い，データの代わりにデータブロック単位で再配置を行う．索引参照プロトコルとカウンタ加算プロトコルの値の秘匿性については，秘密分散法と MPC の秘匿性に帰着し，アクセスパターンの秘匿性については 5 章の安全性定義を満たすことを 9.1.1 項で示した．また，パス再配置プロトコルについては，索引参照プロトコルおよびカウンタ加算プロトコルと組み合わせても，値の秘匿性については Keller らの方式の秘匿性（すなわち秘密分散法と MPC の秘匿性）に帰着し，アクセスパターンの秘匿性については 5 章の安全性定義を満たすことを 8.2.1 項で示した．MAC アドレスの代わりにハッシュ値をキーとして用いても，索引参照プロトコル中の MAC が $Hash(MAC)$ に変わるだけで，その処理は変わらないので，安全性は変わらない．データの代わりにデータブロック単位で再配置を行っても，配置階層算出プロトコルとパス再配置プロトコル中のデータ w_i がデータブロック W_i に変わるだけで，その処理は変わらないので，安全性は変わらない．以上から，本方式の安全性は，値については秘密分散法と MPC の安全性に帰着し，アクセスパターンについては，5 章の安全性定義を満たす．

9.2.2 コスト

本方式の索引のメモリ使用量は，再帰のない Path ORAM 利用方式と異なり，再帰の階層が深くなるごに $\frac{1}{\chi}$ になるため，$3(2^t-1)\beta\ell\lceil\frac{\chi^R-1}{\chi^{R-1}(\chi-1)}\rceil$ ビットとなる．方式全体のメモリ使用量は $3(2^t-1)\ell\lceil\frac{\chi^R-1}{\chi^{R-1}(\chi-1)}\rceil MAX + 3M\beta\ell + 3\alpha\beta\ell\log M$ ビットとなる．

次にラウンド数と通信量を求める．本方式は Path ORAM 利用方式を R 回実行する．基本的にはループ 1 回あたりのラウンド数と通信量は再帰のない Path ORAM 利用方式と同様であるが，索引参照プロトコルについては，索引の大きさが M ではなく χ である．また，最初の $R-1$ 回のループでは，送られてきた MAC アドレス（MAC）を等号判定で探すのではなく，MAC を含む区間を区間判定（大小比較 2 回の並列実行）で探すので，索引参照プロトコルのループ 1 回あたりのラウンド数は積 3 回，大小比較 1 回となり，通信量が積 $2\chi+1$ 回，大小比較 4χ 回となる．最後のループでは，MAC を等号判定で探すため，索引参照プロトコルのラウンド数は積 3 回，等号判定 1 回となり，通信量が積 $2\chi+1$ 回，等号判定 2χ 回となる．

部分索引展開プロトコルについては，何回目の再帰であるかによって，二分木とスタッシュから受け取るデータ数が変わる．i 回目の再帰の際の，二分木とスタッシュから受け取るデータ数を m_i とする．部分索引展開プロトコルは $R-1$ 回のループで行われ，m_i 個の索引から，区間判定によって MAC を含む索引を探索するので，ループ 1 回あたりのラウンド数は積が 1 回，大小比較が 1 回，通信量は積が $3m_i$ 回，大小比較が $2m_i$ 回となる．カウンタ加算プロトコルは最後の 1 回のループで行われ，m_i 個のデータから，等号判定によって MAC に該当するデータを探索するので，ラウンド数は積が 3 回，等号判定が 1 回，1 ビット安定ソートが 1 回，通信量は積が $7m_i$ 回，等号判定が $2m_i$ 回，1 ビット安定ソートが 1 回となる．パス再配置プロトコルについては，何回目の再帰であるかによって，再配置するデータの数 m_i および二分木の高さ h_i が変わり，ラウンド数と通信量に反映される以外は，8.2.2 項で述べた再帰のない Path ORAM 利用方式におけるパス再配置プロトコルの通信コストと同様である．

再帰的 Path ORAM 利用方式全体の処理コストはラウンド数が $R\{24n\log m_i + (h_i+3)m_i + 2n + 17\} - 7$ で，通信量は $(R-1)[2\chi(3\ell+2) + m_i[n\{2m_i + 2(3\ell-2)\log m_i\} +$

$\ell(47\log\ell+94)+3\ell+16h_i+11]+3h_i+3]+2\chi(\ell+4log\ell+4)+m_R[n\{3m_R+4(3\ell-2)\log m_R\}+\ell(47\log\ell+94)+\beta(4\log\ell+4)+16h_R+11]+3h_R+4$ となる．

9.2.3 新規性

9.2 節の冒頭で述べたように，Keller らの方式は，索引がキー値についてソートされていることを前提とするが，時系列情報のセキュア集計では大小のキー値が順不同で到着するので索引はソートされない．そのため，Keller らの方式をそのまま利用することはできない．また，時系列情報のセキュア集計ではキー値は巨大な集合の部分集合であり，どの部分集合かあらかじめ予測できない．そのため，到着したキー値だけを索引に登録し，かつソートを維持しようとするとアクセスパターンが漏洩する一方，巨大な集合全体に対して索引を設けるとメモリ使用量が膨大になり，実用性が失われる．

これらの問題を解決するために，キー値のハッシュ値をキー値とみなすことで，索引をハッシュ関数の値域サイズに圧縮するとともに，索引をハッシュ値についてソートすることで，Keller らの方式を利用することを考えた．しかし，ハッシュ値を用いる場合，ハッシュ値の衝突すなわちキー値の衝突が生じる．そこで，ハッシュ値の衝突数の上限のデータをブロック化することで衝突の問題を回避した．

上記のハッシュ関数による索引の圧縮とブロック化による衝突回避の技術（9.2 節）は，Path ORAM の MPC 実現において新規である．また，7 章で述べた表への値の加算と表自体の拡張を行う手法を，索引への値の加算と索引の拡張に利用した点（9.1 節の表 7）も新規である．これらの 3 つの新技術により，再帰的 Path ORAM の MPC 実現を時系列情報のセキュア集計問題に適用可能とし，アクセスパターンを秘匿しながら通信量のオーダーを $O(\ell(\log N)^3)$ に削減した．

10. 提案方式の比較

各方式の概要を**表 11** に示す．各方式の安全性とコストを**表 12** に示す．具体値は，サーバが受け取る MAC アドレス数 $M=1$ 億，二分木の段数 $H=28$，1 ノードに格納可能なデータ数 $=4$，二分木に格納可能なデータ数 $(2^{28}-1)\times 4=1{,}073{,}741{,}820$，$\chi=10$，再帰の回数 $=8$，$St.size=280$，秘密分散の法のビット数 $\ell=\lceil\log(7\times10^{13})\rceil=46$，$n=3$，$k=2$ として算出した．再帰的 Path ORAM 利用方式の場合は，索引サイズを $10M=10$ 億とした．

再帰的 Path ORAM 利用方式におけるハッシュ値の衝突数の上限 MAX について考察する．値域が 1〜10 億（索引サイズ）のハッシュ関数に 1 億回（MAC アドレス数）の入力を行って，衝突の最大値を計測する実験を 100 万回実施した．その結果，衝突の最大値が 4 の場合が 528 回，5 の場合が 879,287 回，6 の場合が 118,329 回，7 の場合が 1,836 回，8 の場合が 20 回，9 以上の場合が 0 回となった（**図 11**）．そこで，衝突数が 10 を超える確率はほぼ 0 と考え，$MAX=10$ とした．

なお，MAC アドレスの可能な個数は約 70 兆であるが，時系列情報の集計システムが実際に受け取る 1 億個の MAC アドレスは，そのうちのランダムな 1 億個ではない．しかし，本評価において，システムが実際に受け取る MAC アドレスを予想することは困難であったため，ハッシュ関数をランダム関数（入力に対して，値域内の一様ランダムな値を出力する関数）として理想化した．このようにハッシュ関数をランダム関数とみなす仮定の下では，実際に受け取る MAC アドレスを入力としても，定義域内のランダ

表 11 方式の比較
Table 11 Comparison of methods.

二分探索	ソートされた集計表から二分探索を用いてデータを検索する． あれば，カウンタを+1，なければ項目を挿入する．
並列集計	集計表を並列に検索する．あれば，カウンタを+1，なければ，全要素への並列処理により，ダミーデータの先頭位置に項目を登録する．
PathORAM	索引を用いて，二分木を検索する． あれば，カウンタを+1，なければ，索引と二分木に項目を登録する． （索引への登録は，並列集計と同様に行う．）
再帰的PathORAM	索引を新たな二分木に格納し、 再帰的にPathORAMを実行しながら検索する

表 12 安全性とコストの比較
Table 12 Comparison in security and cost.

安全性	既存技術の組み合わせ	並列集計	PathORAM利用	再帰的PathORAM利用
値の秘匿性	秘密分散とMPCの秘匿性に帰着			
アクセスパターンの秘匿性	漏洩	秘匿（変数への読み書きが識別不能）		

メモリ使用量	既存技術の組み合わせ	並列集計	PathORAM利用	再帰的PathORAM利用
オーダ	$O(\ell M)$			
(Byte)	1.15GB	1.73GB	29.51GB	325.95GB

ラウンド数	既存技術の組み合わせ	並列集計	PathORAM利用	再帰的PathORAM利用
オーダ	$O(log\ M)$	$O(1)$	$O(\log M)$	$o((\log M)^2)$
具体値	162	10	7927	36360

通信量	既存技術の組み合わせ	並列集計	PathORAM利用	再帰的PathORAM利用
オーダ	$O(\ell\ log\ M)$	$O(\ell M)$	$O(\ell M)$	$O(\ell(log\ M)^3)$
具体値	5076	1.42×10^{10}	1.48×10^{10}	2.56×10^{7}

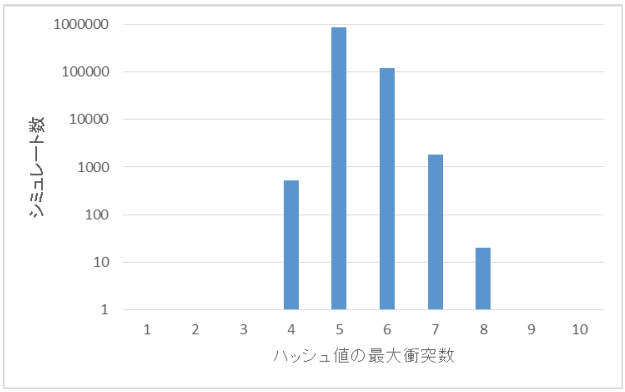

図 11 ハッシュ値の最大衝突数
Fig. 11 Maximum number of collisions of hash value.

ムな値を入力としても，出力は同様にランダム値となるので，ランダムな値を入力として評価しても，一定の妥当性があると考えた．

表 12 を見ると，二分探索利用方式がコスト面では最も優れている．しかし，この方式はアクセスパターンからカウンタの値が漏洩するため，安全ではない．Path ORAM 利用方式は，並列集計方式と同様の処理を行う索引参照プロトコルに加え，カウンタ加算とパス再配置を行うため，並列集計方式よりもメモリ使用量，ラウンド数，通信量が大きい．

並列集計方式と再帰的 Path ORAM 利用方式を比較すると，ラウンド数については並列集計が $\frac{1}{3650}$ である一方，通信量については再帰的 Path ORAM 利用方式が $\frac{1}{550}$ である．メモリ使用量については，アクセスパターン利用方式が優れているが，再帰的 Path ORAM 利用方式の場合でも 330 GB であり，サーバにとってはそれほど大きな負荷ではない．そのため，両方式は，ラウンド数と通信量について一長一短があるといえる．なお，五十嵐ら [11] によれば，大規模なシステムの場合，通信量の方が処理時間に与える影響が大きい．

以上のように，本論文ではラウンド数の小さい方式と通信量の小さい方式の 2 方式を提案することができた．

11. 結論

本論文では，時系列情報のセキュア集計問題を定式化した．この問題の特徴は，集計のキーとなる識別子が巨大な集合に属し，集計期間内に受け取る識別子が当該集合のどの部分集合か予測できず，受け取る順序も予測できない状況において，時系列情報，その集計値および集計表へのアクセスパターンを秘匿しながら集計することである．サーバは受け取る時系列情報の範囲をあらかじめ予想できないため，情報の受取りにともなって集計表の値の加算だけではなく，表の拡張を行う必要がある．

定式化した問題に対して，秘密分散によって時系列情報を秘匿しながら MPC によって集計する手法を検討した．集計表の加算・拡張の際に，表の全データにアクセスすればアクセスパターンを秘匿できるが，MPC の通信量がデータ数の線形オーダーとなり，実用性が失われる．一方，二分探索などの従来手法を用いて表の一部のみにアクセスすると，アクセスパターンが漏洩する．そこで，全要素へのアクセスを避けながらアクセスパターンを秘匿する ORAM を利用した．具体的には，Keller らによる再帰的 Path ORAM の MPC 実現方式を参考とし，索引への追加と索引自体の拡張，ハッシュ関数を用いた索引の圧縮，ハッシュ値の衝突回避の技術を新たに考案し，アクセスパターンを秘匿しながら MPC の通信量を $O(\ell(\log M)^3)$ に削減する手法を提案した．

参考文献

[1] 総務省：Wi-Fi 整備についての現状と課題—総務省 (2013)，入手先 ⟨http://www.soumu.go.jp/main_content/000322502.pdf⟩.

[2] IEEE-SA: Standard Group MAC Addresses:A Tutorial Guide (2011), available from ⟨http://standards.ieee.org/develop/regauth/tut/macgrp.pdf⟩.

[3] Shamir, A.: How to Share a Secret, *Comm. ACM*, Vol.22, No.11, pp.612–613 (1979).

[4] Bogdanov, D., Laur, S. and Willemson, J.: Sharemind: A Framework for Fast Privacy-Preserving Computations, *ESORICS 2008*, LNCS, Vol.5283, pp.192–206 (2008).

[5] Nishide, T. and Ohta, K.: Multiparty Computation for Interval, Equality, and Comparison Without Bit-Decomposition Protocol, *PKC 2007*, LNCS, Vol.4450, pp.343–360 (2007).

[6] Catrina, O. and Hoogh, S.: Improved primitives for secure multiparty integer computation, *Security and Cryptography for Networks*, LNCS, Vol.6280, pp.182–199 (2010).

[7] Damgård, I., Fitzi, M., Kiltz, E., Nielsen, J. and Toft, T.: UnconditionallySecure Constant-Rounds Multiparty Computation for Equality, Comparison, Bits and Exponentiation, *TCC 2006*, Vol.3876, pp.285–304 (2006).

[8] 濱田浩気，五十嵐大，千田浩司ほか：秘匿関数計算上の線形時間ソート，暗号と情報セキュリティシンポジウム（SCIS 2011）予稿集，2C3-5 (2011).

[9] Hamada, K., Kikuchi, R., Ikarashi, D., Chida, K. and Takahashi, K.: Practically Efficient Multi-party Sorting Protocols from Comparison Sort Algorithms Information Security and Cryptology, *ICISC 2012*, Vol.7839, pp.202–216 (2012).

[10] Laur, S., Willemson, J. and Zhang, B.: Round-efficient oblivious database manipulation, *ISC*, LNCS, Vol.7001, pp.262–277 (2011).

[11] 五十嵐大，濱田浩気，菊池　亮，千田浩司：少パーティの秘密分散ベース秘密計算のための $O(\ell)$ ビット通信ビット分解および $O(|p'|)$ ビット通信 Modulus 変換法，CSS 2013 (2013).

[12] 濱田浩気，五十嵐大，千田浩司：秘密計算上の関係代数演算アルゴリズムの改良，信学技報，Vol.112, LOIS2012-82, pp.77–82 (2013).

[13] Goldreich, O.: Towards a theory of software protection and simulation by oblivious RAMs, *19th Annual ACM Symposium on Theory of Computing*, pp.182–194 (1987).

[14] Stefanov, E., Dijk, V.M., Shi, E., et al.: Path ORAM: An Extremely Simple Oblivious RAM Protocol, *2013 ACM SIGSAC Conference on Computer & Communications Security*, pp.299–310 (2013).

[15] Stefanov, E., Shi, E. and Song, D.: Towards practical oblivious RAM, NDSS 2012, arXiv:1106.3652 (2012).

[16] Shi, E., Chan, T.-H.H., Stefanov, E. and Li, M.: Oblivious RAM with $O(\log N)^3$ worst-case cost, *ASIACRYPT 2011*, LNCS, Vol.7073, pp.197–214 (2011).

[17] Keller, M. and Scholl, P.: Efficient, Oblivious Data Structures for MPC, *ASIACRYPT 2014*, LNCS, Vol.8874, pp.506–525 (2014).

[18] Martin, R. and Angeleka, S.: "Balls into Bins" – A Simple and Tight Analysis, *Randomization and Approximation Techniques in Computer Science*, pp.159–170 (1999).

付　　録

A.1 既存技術の組合せによる方式のコスト

Table のメモリ使用量と，1 回の更新を行う際のラウンド数と通信量を求める．

Table は MAC アドレスとカウンタの 2 属性を保持し，M 行 2 列の行列である．各要素のビット長は ℓ ビットであるため，メモリ使用量は $2M\ell \simeq 2M \log N$ ビットとなる．

次にラウンド数と通信量を求める．この方式では，何回目の更新かによって *Table* のサイズが異なり，ラウンド数と通信量が異なるため，上界を求める．上界のケースは，*Table* に MAC アドレスが M 個含まれている場合である．ステップ 1 で等号判定が $\log M$ 回，大小比較が $\log M$ 回行われる．この方式では，等号判定と大小比較の並列処理は可能だが，等号判定間および大小比較間の並列処理はできない．そのため，プロトコル全体のラウンド数は，ラウンド数がより大きい大小比較が $\log M$ 回，通信量は等号判定 $\log M$ 回と大小比較 $\log M$ 回の和になる．具体的なプロトコルとして Catrina ら [6] の等号判定と大小比較を用いた場合，等号判定のラウンド数が 4，通信量が $\ell + 4\log \ell$ となり，大小比較のラウンド数が 6，通信量が $3\ell - 2$ となる．したがってプロトコル全体のコストの上界はラウンド数が $6 \log M$，通信量が $2 \log M\,(2\ell + 2 \log \ell - 1)$ となる．

A.2 アクセスパターンを秘匿する並列集計方式のコスト

Table は行番号，MAC アドレス，カウンタの 3 属性を保持しており，M 行 3 列の Table である．各要素のビット長は ℓ ビットであるため，メモリ使用量は $3M\ell \simeq 3M \log N$ ビットとなる．

本方式では，更新回数による計算コストの変化はない．ステップ 2〜5 のラウンド数は等号判定が 1 回で，通信量は等号判定が M 回となる．ステップ 7 のラウンド数と通信量は積 1 回で，ステップ 9〜12 でラウンド数は等号判定と積が 1 回で，通信量は等号判定が M 回，積が $2M$ 回となる．したがって，プロトコル全体のラウンド数は積 2 回，等号判定 2 回となり，通信量は積 $M+1$ 回，等号判定 $2M$ 回となる．Catrina の等号判定と大小比較プロトコルを例として，積の実行回数で見積もるとラウンド数が 10，通信量が $2M(\ell + 4\log \ell + 1) + 1$ となる．

A.3 配置階層算出プロトコル

ステップ 2 と 4〜11 が 8.2 節で説明したアルゴリズムと異なる部分になっている．ステップ 2 の f はパスに格納できるか，または，格納すると決めたかを示すフラグであり，0 に初期化する．ステップ 4 から 11 は，葉ノードからルートに向かってループする．ステップ 5 において，階層 j が最下層共通祖先ノード LCA よりも下の場合は r_i が 0 なので，f は 0 のままとなり，ステップ 6，7 の左辺は 0 となり，ステップ 8〜10 の左辺は元値のままとなる．ステップ 5 にて，階層 j が LCA のときに，$r_i = 1$ なので $f = 1$ となる．$Z_j.num$ は j 階層目のノードにすでに配置されたデータ数を表し，$Z_j.num_2$ は $Z_j.num$ の MSB を表す．$Z_j.num_2$=0 は当該ノードに空きがあることを意味する．その場合，ステップ 6 において，$b_i = 1$ となり，データが i 階層目のノードに配置されることが決まる．ステップ 7 で f を 0 に戻す．ステップ 8〜10 で，$Z_i.num$ を +1 する．当該ノードに空きがない場合には，ステップ 6 で $b_i = 0$ のまま，$f = 1$ のままとなる．なお，w_i がダミーの場合は，r_0〜r_{H-1} までがすべて 0 なので，b_0〜b_{H-1} がすべて 0 になり，$a = 0$（パス上に格納できる）$H_E = 0$ となる．

表 A·1　配置階層算出（ComputePos）プロトコル
Table A·1　Protocol for computing position.

Input :POS ←再配置対象パスの番号　$[w_i]$ ←作業領域上のデータ　$[e]$ ←ダミーデータかどうか
Output :$[a]$ ←パスに格納できたかをどうかを示す 1 ビットの値(スタッシュに格納する場合[1]，パス上に配置する場合[0])
$[H_E]$ ←データを格納するパス上のノードの階層

1 $[r_0], \cdots, [r_{H-1}] \leftarrow \mathrm{LCA}([w_i.pos], POS, [e])$
2 $[a], [f] \leftarrow [0]$　　fは格納できるか，または格納したかを示すフラグである
3 $[b_0], \cdots, [b_{H-1}] \leftarrow [0]$　H_Eを 1 ビットの配列で示した値となる．ここでは0で初期化する
4 **for** $j = H - 1$ to 0 **do**
5 $[f] \leftarrow [f] + [r_j]$
6 $[b_j] \leftarrow [f] \times (1 - [Z_j.num_2])$　　$Z_j.num_x$は$Z_j.num$をビット分解した値，ここでは$Z.size = 4$とする
7 $[f] \leftarrow \mathrm{XOR}([f], [b_j])$
8 $[Z_j.num_2] \leftarrow \mathrm{XOR}([Z_j.num_2], [Z_j.num_1] \times [Z_j.num_0] \times [b_j])$　　$Z_j.num_0$がLSBである
9 $[Z_j.num_1] \leftarrow \mathrm{XOR}([Z_j.num_1], [Z_j.num_0] \times [b_j])$
10 $[Z_j.num_0] \leftarrow \mathrm{XOR}([Z_j.num_0], [b_j])$
11 **end for**
12 $[a] \leftarrow 1 - [e] - \Sigma_j[b_j]$
13 $[H_E] \leftarrow \Sigma_j[b_j] \times j$

表 A·2　パス再配置プロトコル
Table A·2　Protocol for path relocation.

Input :POS ←再配置対象パスの番号
$[w_0], \cdots, [w_{m-1}]$ ←作業領域Wに展開されている全データ
Output :なし

1 **for** $i = 0$ to $m - 1$ **do**
2 $([H_i], [a_i]) \leftarrow \mathrm{ComputePos}(POS, [w_i], [w_i.dummy()])$
3 **end for**
4 $\mathrm{Sort}(([w_i], [H_i], [a_i])_i)$　　$(0 \le i \le m - 1)$
5 **for** $j = 0$ to $H - 1$ **do parallel**
6 $([b_{j*z+0}], [H'_{j*z+0}]) \leftarrow ([Z_j.num_0] + [Z_j.num_2], j)$　　$Z.size = 4$と仮定
7 $([b_{j*z+1}], [H'_{j*z+1}]) \leftarrow ([Z_j.num_1] + [Z_j.num_2], j)$
8 $([b_{j*z+2}], [H'_{j*z+2}]) \leftarrow ([Z_j.num_1] + [Z_j.num_2], j)$
9 $([b_{j*z+3}], [H'_{j*z+3}]) \leftarrow ([Z_j.num_2], i)$
10 **end for**
11 $\mathrm{Sort}([b_j], H'_j)$　　b_jに基いてソートを行う
12 **for** $i = 0$ to $m - 1$ **do parallel**
13 **if** $i < Z.size \times H$ **then**
14 $[H_i] \leftarrow H_i + [w_i.dummy()] \times (H'_i - H_i)$
15 **else**
16 $[H_i] \leftarrow [H_i] + H \times [w_i.dummy()]$
17 **end for**
18 $\mathrm{Shuffle}(([w_i], [H_i]))$
19 **for** $i = 0$ to $m - 1$ **do**
20 $H_i \leftarrow \mathrm{Reveal}([H_i])$
21 $[w_i]$をH_i階層目のノードに配置する
22 **end for**

A.4 パス再配置プロトコル

上記同様に $Z.size = 4$ のアルゴリズムを示す．ステップ 1〜3 で実データの配置階層を決定している．ス

*5 ダミーデータの MAC アドレスが 0 なので $[w_x.mac = 0]$ により実現される．

テップ 4〜17 が 8.2 節で説明したアルゴリズムと異なる部分になっている．ステップ 4 では，2 つのデータ w_x, w_y について，比較関数 $((w_x, H_x, a_x) < (w_y, H_y, a_y) \Leftrightarrow (w_x.dummy() \times (a_y - 1) + 1 - a_x) = 0/1$ を用いて，パス上に格納される実際のデータ，ダミーデータ，スタッシュ上に格納される実際のデータの順にソートする．なお，$w_x.dummy()$ はデータ w_x がダミーならば 1，実データならば 0 を返す関数である[*5]．ステップ 5〜10 では，各ノードに格納した実データ数 $Z_j.num$ をビット分解した $Z_j.num_t$ を用いる．$Z_j.num_t$ は $Z_j.num$ の t ビット目の値であり，$Z_j.num_2$ は MSB である．それをもとにノードごとにデータが格納されていることを示す $Z_j.num$ 個の $([1], j)$，データが格納されていないことを示す $Z.size - Z_j.num$ 個の $([0], j)$ を求める．たとえば，Z_2 に配置されたデータ数が $Z_2.num = 3$ のとき，3 個の $([0], 2)$ と 1 個の $([1], 2)$ を求める．ステップ 11 でデータが配置される場所，配置されない場所の順にソートする．最初に $([0], j)$ のデータがきて，後ろに $([1], j)$ のデータが来る．ステップ 12〜17 では，ダミーデータを格納する階層を決めており，ステップ 14 がパスに格納する場合で，ステップ 16 がスタッシュに格納する場合の処理となる．

奈良 成泰 （学生会員）

2016 年電気通信大学情報理工学部総合情報学科卒業．2018 年同大学大学院情報理工学研究科情報学専攻修士課程修了見込み．情報セキュリティの研究に従事．

天田 拓磨

2016 年電気通信大学情報理工学部総合情報学科卒業．2018 年同大学大学院情報理工学研究科情報学専攻修士課程修了見込み．情報セキュリティの研究に従事．

西出 隆志 （正会員）

1997 年東京大学理学部情報科学科卒業．同年日立ソフトウェアエンジニアリング株式会社入社．セキュリティ，ネットワーク製品の設計開発に従事．2003 年南カリフォルニア大学修士課程修了．2008 年電気通信大学．博士（工学）．2009 年九州大学大学院システム情報科学研究院助教を経て，2013 年より筑波大学システム情報系准教授．暗号，情報セキュリティの研究に従事．

土井 洋 （正会員）

1988 年 3 月岡山大学理学部数学科卒業．同年日立ソフトウェアエンジニアリング株式会社入社．1994 年北陸先端科学技術大学院大学情報科学研究科博士前期課程修了．2000 岡山大学大学院自然科学研究科博士課程修了．同年より中央大学研究開発機構助教授．2004 年より情報セキュリティ大学院大学教授．暗号理論，情報セキュリティの研究に従事．博士（理学），電子情報通信学会会員．

吉浦 裕 （正会員）

1981 年東京大学理学部情報科学科卒業．日立製作所を経て，2003 年より電気通信大学勤務．現在，情報理工学研究科教授．情報セキュリティ，プライバシ保護の研究に従事．博士（理学）．日立製作所社長技術賞（2000 年），情報処理学会論文賞（2005 年，2011 年），システム制御情報学会産業技術賞（2005 年），IEEE IIH-MSP best paper award（2006），日本セキュリティ・マネジメント学会論文賞（2010 年，2016 年），IFIP I3E best paper award（2016）等受賞．電子情報通信学会，日本セキュリティ・マネジメント学会，人工知能学会，システム制御情報学会，IEEE 各会員．本会フェロー．