マルチパーティ計算を用いたOblivious RAMの利便性及び安全性の向上(2) ー プロトコルの設計と評価 ー

和田 紘帆† 西田 直央† 加藤 遼† 吉浦 裕†

†電気通信大学 大学院情報理工学研究科
182-8585 東京都調布市調布ヶ丘 1-5-1
wada.hiroho@uec.ac.jp , n.nishida@uec.ac.jp

あらまし Oblivious RAMは，データのアクセスパターンを秘匿できるが，クライアントが暗号鍵を管理しなければならず，一人のクライアントの利用に限られる．一方で，秘密分散したデータをマルチパーティ計算で検索する手法がある．アクセスパターンの秘匿は困難だが，鍵の管理が不要で，複数クライアントによる利用が可能である．著者らは，Oblivious RAMに秘密分散とマルチパーティ計算を取り入れることで，その特徴を維持しつつ，問題点を解決する手法を提案している．本稿では，提案手法を実現するマルチパーティプロトコルを設計し，通信コストと安全性を評価する．

Improving Security of Oblivious RAM (2)

Hiroho Wada† Naohisa Nishida Ryo Kato† Hiroshi Yoshiura†

†Graduate School of Informatics and Engineering, The University of Electro-Communications
1-5-1, Choufugaoka, Choufu, Tokyo 182-8585, JAPAN
wada.hiroho@uec.ac.jp , n.nishida@uec.ac.jp

Abstract It becomes more and more important to balance information usability and confidentiality. Oblivious RAM is expected to meet this challenge. In this paper, we describe details of methods to improve security of Oblivious RAM.

1 はじめに

秘密のデータをサーバに預けて，クライアントが自由に検索を行うことが可能な技術として，Oblivious Random Access Machine (ORAM) [1][2][3][4] が注目されている．ORAMは，暗号データに対して演算が行われる度に，その暗号データの格納位置を変化させてデータを管理する方法である．暗号化によるデータの秘匿に加え，データのアクセスパターンを秘匿することが可能である．内外どちらの攻撃者も，どのデータがアクセスされたかということや検索頻度，さらにデータ間の関係も知ることができない．

しかし，ORAMの問題点として，クライアントがサーバにおける各データの格納位置を知っておく必要があるため，データの格納位置の情報を紛失すると，データの検索が不可能になってしまうことが挙げられる．

さらに，データを復号するための鍵をクライアントが持つ必要があるため，クライアントの鍵が漏洩すると，サーバに預けたデータが不正に復号される危険性があり，鍵を紛失するとデータの復号が不可能になってしまう．クライアントはITやセキュリティに精通しているとは限らないため，これらは利便性及び安全性の

面で大きな問題となる．さらに，複数のクライアントが同一の鍵を利用すると安全性が低下するため，鍵を所持しているただ一人のクライアントの利用のみに限られてしまうという利便性の問題がある．

著者らは，文献 [5] において，ORAM に秘密分散とマルチパーティ計算を取り入れることにより，アクセスパターンを秘匿しつつ，鍵の管理を不要化し，複数クライアントの利用を可能にする手法を提案した．本稿では，提案方式を具体化するマルチパーティプロトコルを設計し，通信コストと安全性の概略評価を行う．

2 先行研究

2.1 ORAM

クライアントの鍵を用いて暗号化されたデータを，サーバ等の外部記憶領域に預ける．そして，クライアントからの検索が行われる度に，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．

- データの値の秘匿
  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るため，サーバの管理者と外部からの攻撃者共に，データの値を知ることができない．
- アクセスパターンの秘匿
  データがアクセスされる度にデータの格納位置が変更されるので，同一データに対するアクセス頻度を推定することが困難である．また，アクセスを観察しても，データ間の関係を推定することができない．

2.2 PathORAM

ORAM の具体例として，現在最も効率が良いとされている E.Stefanov ら [3] による PathORAM を取り上げる．この手法は，検索や書き換えが要求される度に，サーバに預けたデータの木構造を変化させることでアクセスパターンを秘匿する．

2.2.1 サーバの記憶領域

サーバに預けた暗号データは，木構造を用いて記憶される．木は多分木でもよいが，本稿では簡単のために二分木を用いて説明する．

二分木 深さ $L+1$，葉の数 2^L の二分木を所持する．ノード数を N として，$L = \lceil \log_2 N \rceil - 1$ である．二分木の根を深さ 0 番目，葉を L 番目とする．

バケット 二分木の各ノードをバケットと呼び，各バケットには Z 個の暗号データが格納されている．実際の格納データが Z よりも少なければ，ダミーデータを格納することにより，バケットのサイズ Z を維持する．

パス $x \in \{0, 1, \ldots, 2^L - 1\}$ を，左から x 番目の葉ノードとする．葉ノード x から根ノードへ辿る経路をパスと呼び，$P(x)$ で表す．任意のデータは，一つ以上のパス上に存在する．また，$\ell \in \{0, 1, \ \ldots \ , L\}$ として，$P(x)$ 上の ℓ 階層目のバケットを $P(x, \ell)$ と表す．

2.2.2 クライアントの記憶領域

クライアントは，スタッシュとポジションマップの二つの記憶領域を平文で所持する．

スタッシュ プロトコルの実行中に，少数のデータが木のバケットから溢れてしまう可能性がある．そのため，クライアントはスタッシュと呼ばれる記憶領域に，これらの溢れたデータを一時的に格納する．

ポジションマップ 二分木における各データの格納位置を表す表である．格納位置をポジションと呼ぶ．具体的には，各データがどのパス上にあるかを表し，データ d について，$d \in P(x)$ となる x を d のキー値と共に記憶する．データは，$P(x)$ かスタッシュのどちらかに存在する．

2.2.3 検索・書き換えプロトコル

検索及び書き換えは，以下に示す手順で実行される．概要を図 1 に示す．例として，キー値 24 が要求されたものとする．

1. ポジションの再配置
   ポジションマップ上にあるキー値 24 のポ

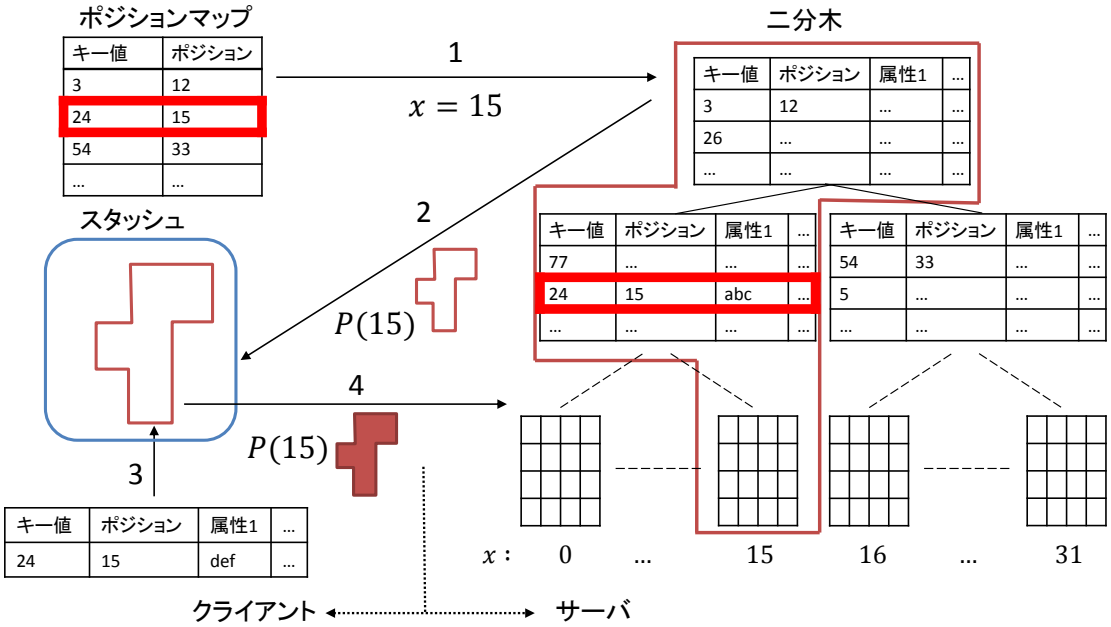

図 1: PathORAM の検索及び書き換え

ジションを取得し，ランダムに再配置する．この際，キー値 24 の元のポジション $x = 15$ を保持しておく．

2. パスの取得
   二分木から $P(15)$ を取得し，スタッシュに格納する．

3. 暗号データの更新
   書き換え要求の場合は，スタッシュ上にあるキー値 24 のデータを書き換える．

4. パスの書き換え
   ポジションマップを参照しつつ，スタッシュに格納されている $P(15)$ 上のデータの配置を並び替えて変化させたものを暗号化して，二分木のパスに新たに書き込む．

2.2.4 再帰的実行

パス内の検索及び書き換えにかかる計算量は $O(\log N)$ で見積もられるが，ポジションマップの参照にかかる計算量は $O(N)$ で見積もられる．そのため，2.2.3 項のプロトコル全体の計算量は $O(N)$ となってしまい，効率が悪い．

これに対し，上述のプロトコルを再帰的に実行することにより，ポジションマップの参照にかかる計算量を抑制し，プロトコル全体の計算量を $O(\log^2 N)$ にすることが可能である [3]．ここで，データのサイズを可変にすることにより，$O(\log N)$ に抑制することが可能だが，再帰の実行回数，つまり，サーバとクライアント間の通信回数が増加してしまうため，本稿では $O(\log^2 N)$ の方式を取り上げることにする．

2.2.5 利便性と安全性

データはクライアントの鍵により暗号化されているため，サーバはデータの値を知ることができない．さらに，検索の度に二分木のパスを変更するので，アクセスパターンを秘匿することが可能である．安全性の詳細については，Stefanov らによる文献 [3] を参照していただきたい．

一方で，1 章で述べたクライアントによる格納位置の保持，クライアントの鍵管理，一クライアントのみの利用という問題点は残っている．

2.2.6 Bandwidth

ORAM の通信コストの評価は，主に Bandwidth によって見積もられる．Bandwidth とは，一つのデータを検索するために必要な通信量のことである．必要なデータを含むパス全体を通信するため，一回の検索にかかる Bandwidth はパス全体のデータ量，つまり $O(\log N)$ で見積もられる．また，再帰の回数はパラメータ χ を用いて，$O(\log N / \log \chi)$ で見積もることができ，再帰的実行を適用すると，Bandwidth は $O(\log^2 N / \log \chi)$ で見積もられる．詳細は文献 [3] を参照していただきたい．

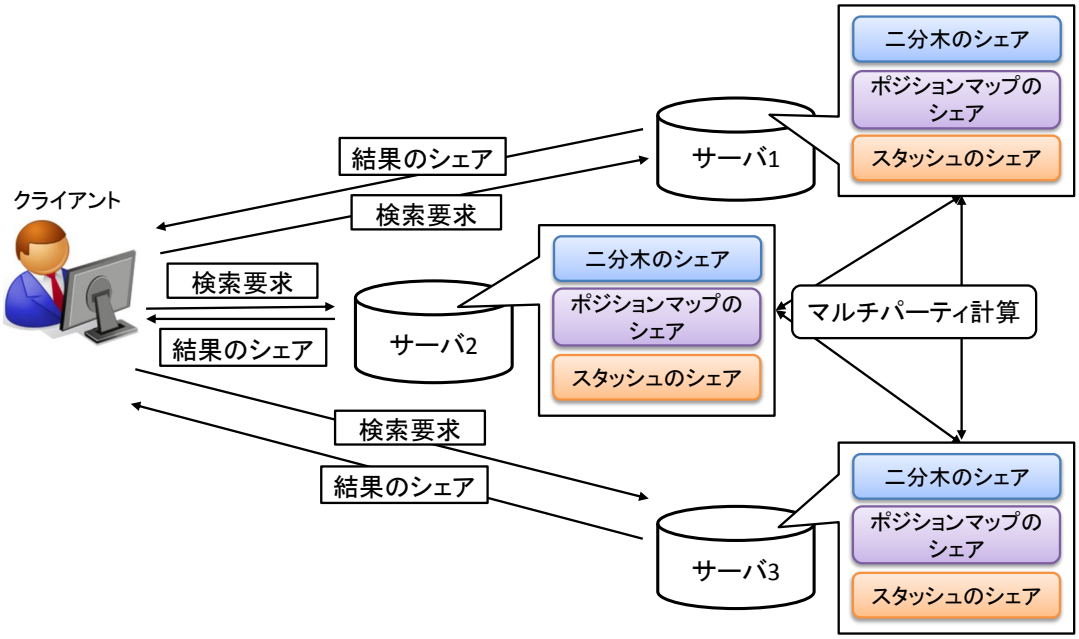

図 2: 秘密分散 PathORAM の概要

2.3 秘密分散とマルチパーティ計算

秘密分散法 [6] を用いてデータを複数のサーバに分散して秘匿し，サーバ間でマルチパーティ計算 (MPC) を行うことにより，データを復号することなく，各種の検索や統計計算を行う．秘密分散法の性質より，閾値以上のサーバが一度に侵入されない限り，元のデータは漏洩しない．また，閾値以上のサーバ管理者が結託しない限り，データの漏洩には繋がらない．さらに，データの復号に鍵を用いないので，クライアント及びサーバは鍵を管理する必要がない．

2.4 達成条件

提案手法の達成条件として以下が挙げられる．

- 鍵管理を不要化し，複数クライアントの利用を可能にする．
- 外部の攻撃者とサーバの管理者に対して，データの値とアクセスパターンを秘匿する．
- MPC の通信コスト (ラウンド数及び通信量) を少なく抑える．

3 提案手法の基本構成

3.1 基本構成

本稿では基本方針として，2.2 節で述べた PathORAM に，2.3 節で述べた秘密分散と MPC を取り入れる．

PathORAM に用いる 3 種類の記憶領域（二分木・ポジションマップ・スタッシュ）を全てサーバ側に配置する．概要を図 2 に示す．二分木・ポジションマップ・スタッシュの情報を秘匿するために，これらを秘密分散して各サーバにシェアを配置する．

次に，各記憶領域中のデータが所持する情報を述べる．なお，これらのデータは全てシェアの形で格納される．

- 二分木
  バケット中のデータは，キー値 (key)，ポジション情報 (pos)，データの値 ($data$) 及び，ダミーデータか否かを表すビット値 ($isDummy$) を所持する．
- ポジションマップ
  ポジションマップ中のデータは，キー値 (key) とポジション情報 (pos) を所持する．
- スタッシュ
  スタッシュ中の暗号データは，キー値 (key)，ポジション情報 (pos)，データの値 ($data$) の他に，データがパス上の ℓ 階層目に格納可能かを示すビット列 ($include_\ell$)，データがバケットに格納済みかを示すビット値 ($done$) 及び，格納可能だが，まだ格納していないことを示すビット値 ($flag = include \cdot (1 - done)$) を所持する．

以上の構成の上で，2.2.3 項で述べた PathO-

RAMの検索プロトコルに相当する処理を，MPCにより実現する．この構成によると，クライアントはポジションマップやスタッシュ等，何も管理しなくてよいため，クライアントに関わる利便性及び安全性の問題を解決することができる．

3.2 課題と目標

PathORAMでは，クライアントは攻撃者として想定されていないので，ポジションマップ及びスタッシュの検索をクライアント側で平文により行うことが可能である．しかし，提案手法ではこれらの処理をサーバ側で行う．サーバは攻撃者になり得るため，クライアントが平文で行っていた処理を，MPCにより安全に行う必要がある．

なお，ORAMでは通信コストをBandwidthで見積もっていたが，提案手法はMPCを用いるので，ラウンド数及び通信量で見積もることにする．ラウンド数の目標を，(マルチパーティプロトコルのラウンド数)·$PathORAM$の再帰の回数　$(\mathrm{O}(\log N))$にする．また，通信量の目標を，(マルチパーティプロトコルの通信量)·$Bandwidth(\mathrm{O}(\log^2 N))$にする．

3.3 記法と定義

本稿で用いる記法及びその定義を以下に示す．

- ℓ: 法となる素数のビット長
- $[a]$: 値aのシェア
- $a \leftarrow b$: bをaに代入，格納
- $d.inf$: データdの情報(inf)の値
  例として，スタッシュに格納されたデータsのポジション情報を，$s.pos$と記す．
- X_i: 集合Xのi番目の要素

3.4 主要プロトコル

- ポジションマップの参照プロトコル
  ポジションマップを参照して，キー値のポジション情報を出力する．
- 検索プロトコル
  ポジション情報を入力して，2.2.3項で述べた検索・書き換えプロトコルと等価な検索処理を行う．
- 再帰的実行プロトコル
  上述した検索プロトコルを再帰的に実行し，2.2.4項で述べた，再帰的実行と等価な処理を実現する．

上記のプロトコルで用いるデータは全てシェアの形で配置される．4, 5, 6章で，これらのプロトコルについて説明する．

4 ポジションマップの参照プロトコル

検索または書き換え要求されたデータのキー値のシェアを入力し，ポジションマップを参照して，そのキー値を持つデータのポジションをシェアの形で出力する．

入力するキー値を$[KEY]$，出力されるポジションを$[POS]$，ポジションマップに格納された任意のデータをpとする．

$[p.key = KEY]$は，等号判定プロトコルを用いて計算され，真なら[1]を，偽なら[0]を出力する．等号判定プロトコルとして，本稿では西出ら[7]による手法を用いることにする．その通信コストは8ラウンド，通信量81ℓである．

ポジションマップの参照プロトコルは以下の式で表される．

$$[POS] = \sum_{\forall p} [p.key = KEY] \cdot [p.pos] \quad (1)$$

式1により，本プロトコルのラウンド数は9，通信量は，バケット数をNとして$(81\ell+1)ZN$で見積もられる．通信量が$\mathrm{O}(N)$のため効率が悪いが，6章で述べる再帰的実行プロトコルにより，通信量を抑制することが可能である．

5 検索プロトコル

検索要求されたデータのキー値のシェア$[KEY]$を入力とし，当該データのシェア$[data]$を出力

する．また，書き換え要求の場合は，上記に加えて，書き換えたいデータの値のシェア $[data^*]$ を入力とし，データを書き換える．

5.1 プロトコル

2.2.3 項の手順に沿ったマルチパーティプロトコルを，サブプロトコル 1, 2, 3, 4 に示す．また，初期状態のスタッシュに格納されているデータの個数を $size$ とする．

サブプロトコル 1 ポジションの再配置

1: ポジションマップの参照プロトコルを実行
2: $[POS]$ を公開
3: ランダムなビット値 $[r_i](0 \leq i \leq L)$ を生成
4: $[newPOS] = \sum_{i=0}^{L}[r_i] \cdot 2^i$ を計算して，入力されたキー値の新しいポジションを設定
5: **for all** $p \in$ ポジションマップ **do**
6: $[p.pos] = [p.key \neq KEY] \cdot [p.pos] + [p.key = KEY] \cdot [newPOS]$
7: **end for**

サブプロトコル 1 の手順 3 では，西出ら [7] による手法を用いて，ランダムなビット値のシェアを生成する．手順 5 から 7 では，入力されたキー値 ($[KEY]$) に該当するポジションを書き換える．この際，他のキー値のポジションは変更せずに，同じ値の異なるシェアで上書きする．これにより，どのキー値のポジションが書き換えられたのかを秘匿する．

サブプロトコル 2 パスの取得

1: 二分木の $P(POS)$ 上にあるバケットを取得
2: 取得したバケットに格納されている全てのデータ s をスタッシュに格納
3: **for all** $s \in$ スタッシュ **do**
4: $[s.pos] = [s.key \neq KEY] \cdot [s.pos] + [s.key = KEY] \cdot [newPOS]$
5: **end for**

サブプロトコル 1 の手順 5 から 7 と同様の処理を，サブプロトコル 2 の手順 3 から 5 で実行し，スタッシュに格納されたデータのポジション情報を変更する．また，サブプロトコル 3 の

サブプロトコル 3 暗号データの更新

1: **for all** $s \in$ スタッシュ **do**
2: **if** 書き換え要求 **then**
3: $[s.data] = [s.key \neq KEY] \cdot [s.data] + [s.key = KEY] \cdot [data^*]$
4: **end if**
5: **end for**

手順 1 から 5 においても同様の処理を行い，スタッシュに格納されたデータの値を更新する．

サブプロトコル 4 の手順 2 から 4 では，西出ら [7] による手法を用いて，区間判定が行われる．手順 6 から 19 について述べる．ここでは，パス上のデータを葉から根のバケットへ逐次格納することで，パスを書き換える．手順 7 から 9 では，スタッシュに格納されたデータがバケットに格納可能だが，まだ格納されていないか ($flag$) を計算する．そして，濱田ら [8] による手法を Shamir(k, n) 秘密分散法に変換したソートの手法を用いて，手順 10 でデータを $flag$ に基づいて降順ソートすることにより，バケットに格納可能なデータ，つまり $flag$ が [1] であるデータがスタッシュ内の先頭に集約される．そこで，手順 12 から 16 において，スタッシュの先頭から Z 個のデータを取り出すことにより，格納可能なデータを効率的にバケットに格納することが可能である．この際，格納可能なデータが Z 個未満の場合は，その不足分にダミーデータを格納することで，バケットのサイズ Z を維持する．最後に，手順 17 で $done$ の更新を行う．データの $flag$ が [1] の場合，そのデータの $done$ を [1] に変更する．

サブプロトコル 2 でパスを取得する際，パスに格納されたデータの $isDummy$ を用いて，データの $done$ を初期化する．実データと同様にダミーデータの $done$ を [0] で初期化すると，サブプロトコル 4 の手順 8 により，$flag$ が [1] になる可能性がある．このとき，$flag$ が [1] であるデータがダミーデータ含めて Z 個以上存在する場合，優先してバケットに格納すべき実データではなく，ダミーデータが格納される可能性が生じる．そのため，ダミーデータ ($isDummy = [1]$) の $done$ を [1] で初期化することで，ダミーデータ

の $flag$ を常に [0] にする．

サブプロトコル 4 パスの書き換え

1: **for all** $s \in$ スタッシュ **do**
2: **for** $\ell \in \{0, 1, \ldots, L\}$ **do**
3: $[s.include_\ell] \leftarrow s$ が $P(POS, \ell)$ に格納可能か
4: **end for**
5: **end for**
6: **for** $\ell \in \{L, L-1, \ldots, 0\}$ **do**
7: **for all** $s \in$ スタッシュ **do**
8: $[s.flag] = [s.include_\ell] \cdot (1 - [s.done])$
9: **end for**
10: スタッシュに格納されたデータを，$flag$ に基づいて降順ソート
11: **for** $i \in \{0, 1, \ldots, Z-1\}$ **do**
12: **if** $[Stash_i.flag = 1]$ **then**
13: $P(POS, \ell) \leftarrow Stash_i.data$
14: **else**
15: $P(POS, \ell) \leftarrow$ ダミーデータ
16: **end if**
17: $[Stash_i.done] = [Stash_i.done] + [Stash_i.flag]$
18: **end for**
19: **end for**
20: スタッシュに格納されたデータを，$done$ に基づいて昇順ソート
21: スタッシュに格納された，先頭から $size+1$ 個のデータを保持し，残りのデータを消去

5.2 評価

検索プロトコルの安全性と通信コストの評価を行う．

まず，安全性を評価する．プロトコルは全て，サーバが所持するシェアに対して MPC で実行する．ただし，サブプロトコル 1 の手順 2 において，ポジション情報 (POS) を公開する．この公開は，サブプロトコル 2 の手順 1 において $P(POS)$ 上にあるバケットを取得するために行われる．しかし，PathORAM においても，サーバに平文のポジション情報を渡していることから，この処理の安全性は PathORAM と同等であると考えられる．他の処理においては，シェアを一度も復号しないことから，安全性は Shamir(k, n) 秘密分散法に帰着すると考えられる．

また，サブプロトコル 4 におけるパスの書き換えは，PathORAM における 2.2.3 節のパスの書き換えと等価な処理を MPC で実行している．そのため，PathORAM と同様に，アクセスパターンを秘匿することが可能である．

次に，通信コストを評価する．サブプロトコル 1 については，4 章で述べた通りである．また，ラウンド数は，どの手順も N に依存しないことから O(1) で見積もられる．サブプロトコル 2 及び 3 における通信量は，スタッシュに格納されたパスのバケット数に依存するため O($\log N$)，ラウンド数は O(1) で見積もられる．

サブプロトコル 4 では，手順 3 及び手順 8 が最も通信量を要し，共に O($\log^2 N$) で見積もられる．ラウンド数は，サブプロトコル中の for ループに依存するが，特に，手順 6 における for ループは，葉から根のバケットへ逐次処理を行うため，並列に実行することが困難である．これより，ラウンド数は手順 6 における for ループに依存し，O($\log N$) で見積もられる．

以上より，検索プロトコル全体の通信量は O(N)，全体のラウンド数は O($\log N$) で見積もられる．通信量が O(N) のため効率が悪いが，6 章で述べる再帰的実行プロトコルを適用することにより，通信量を抑制することが可能である．

6 再帰的実行プロトコル

6.1 プロトコル

上述した二分木を $TREE_0$ と呼ぶ．

ポジションマップ内のデータ (キー値とポジション) を M 個まとめたものを一つのデータとして，$TREE_0$ とは別の二分木 $TREE_1$ に格納する．$TREE_1$ のバケット数は N/M となり，$TREE_0$ よりも木の深さ及びバケット数が少なくなる．この時，$TREE_1$ に対するポジションマップでは，キー値が区間で表される．

この再帰を考慮した検索プロトコル (再帰的実行プロトコル) は，ポジションマップの参照

プロトコル及び検索プロトコルを部品として実現される．

$TREE_X$ に対応したポジションマップを PM_X とする．検索または書き換え要求がされると，まず，ポジションマップの参照プロトコルを用いて PM_X を参照する．この際，PM_X のキー値が区間で表されているため，式 1 において，等号判定プロトコルではなく二回の大小比較プロトコルが用いられる．大小比較プロトコルとして文献 [7] の手法を用いる場合，通信コストは 16 ラウンド，通信量 $558\ell + 11$ である．

次に，サブプロトコル 1，2，4 を実行する．これにより，出力として PM_{X-1} が得られる．以上を PM_0 が得られるまで再帰的に実行する．

最後に，PM_0 及び $TREE_0$ に対して，5.1 節で述べた検索プロトコルを実行することにより，当該データのシェアを出力する．

6.2 評価

再帰的実行プロトコルの安全性と通信コストの評価を行う．

まず，安全性を評価する．再帰的実行プロトコルは，検索プロトコルを再帰的に実行しているため，検索途中のデータのシェアに関しては，再帰を行わない検索プロトコルと同等の安全性を保持すると考えられる．再帰の深さなどのプログラムの動作については，PathORAM においてもサーバが知っていることから，この処理の安全性は PathORAM と同等であると考えられる．

次に，通信コストを評価する．一回の再帰により，新しい二分木のサイズが $1/M$ になるとすると，再帰の回数 X は，$X = \mathrm{O}(\log N/\log M)$ で見積もられる．i 番目の再帰における通信量は，ポジションマップに格納されたデータ数が M であり，二分木に格納されたバケット数が N/M^i であることから，$\mathrm{O}(M + \log^2(N/M^i))$ で見積もられる．

以上より，再帰的実行プロトコル全体の通信量は，$\mathrm{O}(\sum_{i=0}^{X}(M + \log^2(N/M^i)))$ で計算されるため，$\mathrm{O}(M \log N/\log M + \log^3 N/\log M)$ で見積もられる．ここで，$M = \log^2 N$ とすることにより，通信量を $\mathrm{O}(\log^3 N/\log\log N)$ で見積もることができる．ラウンド数は，再帰の回数が $\mathrm{O}(\log N/\log\log^2 N)$ であることから，$\mathrm{O}(\log^2 N/\log\log N)$ で見積もられる．

7 結論と今後の課題

ORAM に秘密分散法と MPC を取り入れることでアクセスパターンを秘匿しつつ，鍵の管理を不要化し，複数クライアントの利用を可能にする手法を提案し，具体的なプロトコルを設計した．また，安全性の概略と通信コストのオーダーを見積もった．

今後は，安全性と通信コストの厳密な評価を行う．また，ラウンド数と通信コストが目標値よりも大きかったため，プロトコルを改良する．

参考文献

[1] O.Goldreich. Towards a theory of software protection and simulation by oblivious rams. In STOC, pp.182-194, 1987.

[2] C.Gentry, K.Goldman, S.Halevi, C.Julta, M.Raykova, and D.Wichs. Optimizing oram and using it efficiently for secure computation. In PETS, pp1-18, 2013.

[3] E.Stefanov, M.Dijk, E.Shi, C.Fletcher, L.Ren, X.Yu, and S.Devadas. Path ORAM: An Extremely Simple Oblivious RAM Protocol. In CCS, pp. 299-310, 2013.

[4] E.Shi, T.-H.H.Chan, E.Stefanov, and M.Li. Oblivious RAM with $\mathrm{O}(\log N)^3$ worst-case cost. In ASIACRYPT, pp.197-214, 2011.

[5] 西田直央, 和田紘帆, 加藤遼, 吉浦裕. マルチパーティ計算を用いた Oblivious RAM の利便性及び安全性の向上 (1)— 構想と方式概要 —. In CSS, 2014.

[6] A.Shamir. How to Share a Secret. In Communications of the ACM, Vol.22, No.11, pp.612-613, 1979.

[7] T.Nishide, K.Ohta. Multiparty Computation for Interval, Equality, and Comparison Without Bit-Decomposition Protocol. In PKC 2007. LNCS, vol.4450, pp. 343-360, 2007.

[8] 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.