Computer Security Symposium 2018
22 - 25 October 2018

Oblivious RAM を利用したセキュアな決定木の実装評価

市川 敦謙[1]　濱田 浩気[1]

概要：決定木学習は，機械学習の中でもポピュラーな学習法である．決定木学習の目的は，決定木の学習 (構築) それ自体に留まらず，構築した決定木を用いた新規データの分類・推定も包含する．分類対象のデータにプライバシ情報が含まれる場合に，マルチパーティ計算によりプライバシ保護して，分類計算を効率良く行う手法が提案されている．本論文ではその手法を実装し，実行時間を評価する．

キーワード：マルチパーティ計算，Oblivious RAM，決定木

Implementation and Evaluation of Oblivious Decision Tree for Secure Computation

Atsunori Ichikawa[1]　Koki Hamada[1]

Abstract: Decision tree learning is a popular one of machine learning methods. The purpose of decision tree learning is not only a learning (constructing a decision tree), but also classifying (using a constructed decision tree). For classify data which include privacy, an efficient privacy-preserving classification scheme for MPC has been proposed. We implement and evaluate that scheme in this paper.

Keywords: Multi-Party Computation, Oblivious RAM, Decision Tree.

1. はじめに

1.1 背景

データマイニングは膨大なデータから特徴となる要素を抽出し，新たな知見を得る技術である．データマイニングではデータ解析手法として機械学習が用いられることも多く，その 1 つに決定木学習が挙げられる．

決定木学習はデータ群から特徴量を抽出し，分類・回帰推定を行うための決定木を構築する手法である．機械学習技術として決定木学習を論じる場合，多くは学習そのもの，すなわちデータからいかに良質な決定木を構築するかが着目される．

しかし，現実に決定木学習を利用する場合，その目的は決定木の構築のみに留まらず，決定木を用いた新規データの分類や回帰推定までを包含する．具体的には，例えば医療機関の持つ患者データから，ある疾患に罹るリスクを推定する決定木を構築し，その木を用いて個人向けに罹患リスクを推定するようなサービスを行う，といったものが考えられる．

このように，構築された決定木を利用して，個人へのフィードバックを返すサービスは，様々な属性情報の整理や分析を手助けする非常に有益なものとなり得る．一方で，こうしたサービスでは個人と深く密接する情報，すなわちプライバシ情報を得ることが不可欠となる．

したがって，こうしたサービスの実現には，サービスを受ける個人 (以後ユーザと呼ぶ) のプライバシ情報を秘匿したままで，決定木へ入力して分類・推定結果を得られることが望ましい．

こうした問題に対し，ユーザ・サービス提供者が互いに自身の持つ情報を秘匿したまま決定木による分類を行う方法が市川らによって提案されている [13], [14].

本論文では，市川らによるマルチパーティ計算可能な決定木の構成方法を実装し，実行時間を測定することで方式の有効性について考察する．

[1] NTT セキュアプラットフォーム研究所
NTT Secure Platform Laboratories

© 2018 Information Processing Society of Japan

1.2 関連研究

本研究は，プライバシ保護データマイニング (privacy preserving data mining, PPDM) に関連する研究である．決定木学習においては，教師データのプライバシを保護して学習を行う手法が知られている [1], [12].

また学習とはやや異なるが，ゲノム解析手法の 1 つであるフィッシャーの正確検定をプライバシ保護しながら効率良く計算するために，検定アルゴリズムを決定木の形に変換して実行する手法が知られている [5].

既製の決定木を用いてプライバシを保護した分類計算を行う手法については，決定木を持つ側と入力データを持つ側の 2 者間で準同型暗号によって互いにデータを秘匿しながら分類を行う手法 [10]，およびそれをランダムフォレストへと拡張した手法 [6], [11] が知られている．マルチパーティ計算を用いてデータを秘匿したまま分類を行う手法は [5] の他，[13], [14] にて論じられている．

2. 準備

2.1 決定木

決定木 (decision tree) とは，各ノードが数または文字列を入力とする述語を持つ，木構造のグラフである．

決定木には大きく分けて回帰木と分類木の 2 種類がある．それぞれ用途は異なるが，木グラフの各枝ノードが等号・不等号の比較述語を持ち，各葉ノードが推定結果を持つという点は共通である．よって本論文では簡単のため，これら 2 つをまとめて「決定木」と表記し，また決定木にデータを入力し推定結果を得ることを「決定木による分類計算」または単に「分類計算」と呼称する．

本論文において扱う決定木を以下に定義する．

定義 1. T をノード $(N_{1,0}, N_{2,0}, N_{2,1}, N_{3,0}, \ldots)$ からなる木構造のグラフであるとする．ノード $N_{1,0}$ は根ノードとし，$N_{2,0}, N_{2,1}$ は $N_{1,0}$ の子であるとする．同様に，T の全てのノード $N_{i,j}$ は 2 つの子ノード $N_{i+1,2j}, N_{i+1,2j+1}$ を持つか，または子を持たない葉であるとする．この時，子を持つ全てのノード $N_{i,j}$ に対して比較データ $Y_{i,j}$ と比較演算子 $\circ \in \{=, <\}$，入力変数 X_i が紐付けられ，また子を持たない全てのノード（葉）$N_{k,l}$ に対して分類結果 C_l が紐付けられているならば，T を決定木であると呼ぶ．

決定木による分類計算は，入力列 $(X_1, \ldots, X_h)$ に対して，根ノードから順に紐付けられたデータを参照し，述語 $(X_i \circ Y_{i,j})_?$ を判定する．次に判定結果が真ならばノード $N_{i+1,2j+1}$ を，偽ならばノード $N_{i+1,2j}$ を参照する．これを繰り返し，葉ノード $N_{k,l}$ に至ったならば，分類結果 C_l を出力する．

2.2 複製秘密分散

複製秘密分散 (replicated secret sharing)[7] は秘密分散法，特に (k, n)-しきい値法と呼ばれる方式の 1 つである．

(k, n)-しきい値法を以下のように，2 つの関数 **Share**, **Reveal** により定義する．

定義 2. **Share** : $\mathcal{R} \to \mathcal{R}'^n$ は，ある値 $r \in \mathcal{R}$ から n 個のシェア列 $([\![r]\!]_1, \ldots, [\![r]\!]_n) \in \mathcal{R}'^n$ への写像である．ただし，$\mathcal{R}, \mathcal{R}'$ はいずれも環とする．**Reveal** : $\mathcal{R}'^k \to \mathcal{R}$ は，k 個のシェアの組 $([\![r]\!]_{\sigma(1)}, \ldots, [\![r]\!]_{\sigma(k)}) \in \mathcal{R}'^k$ から値 $r \in \mathcal{R}$ への写像である．ただし，σ は単射 $\{1, 2, \ldots, k\} \to \{1, 2, \ldots, n\}$ とする．この時，任意の単射 σ に対してある係数列 $(\lambda_1, \ldots, \lambda_k)$ が存在し，任意の $r \in \mathcal{R}$ について $r = \mathbf{Reveal}(\{\mathbf{Share}(r)_{\sigma(i)}\}_{i \le k}) = \sum_{i=1}^{k} \lambda_i [\![r]\!]_{\sigma(i)}$ が成り立つならば，かつ，任意の単射 $\tilde{\sigma} : \{1, \ldots, k-1\} \to \{1, \ldots, n\}$ および任意のアルゴリズム A について $H([\![r]\!]_{\tilde{\sigma}(1)}, \ldots, [\![r]\!]_{\tilde{\sigma}(k-1)}) = H([\![r]\!]_{\tilde{\sigma}(1)}, \ldots, [\![r]\!]_{\tilde{\sigma}(k-1)} \mid A([\![r]\!]_{\tilde{\sigma}(1)}, \ldots, [\![r]\!]_{\tilde{\sigma}(k-1)}))$ が成り立つならば[*1]，関数の組 $\{\mathbf{Share}, \mathbf{Reveal}\}$ は（情報理論的安全な）(k, n)-しきい値法である．

(k, n)-しきい値法においては通常，**Share** によって生成された n 個のシェアを，n 人のプロトコル参加者 (以後「パーティ」と呼ぶ) に，重複がないよう 1 つずつ配布し，また **Reveal** においては k パーティ以上の合意をもって k 個のシェアから秘密を復元する．

複製秘密分散においては，関数 **Share**, **Reveal** は次のように定義される．

Share: $m = {}_nC_{k-1}$ 個の，$r = \sum_{i=1}^{m} r_i$ を満たす数 r_i をランダムに選ぶ．次に，n パーティ中の $k-1$ パーティにより構成される全ての部分集合 (仮に i 番目の部分集合を $\mathcal{P}_i$ とする) について，$\mathcal{P}_i$ に属さないパーティ全員に r_i を与える．

Reveal: 上記の **Share** により，k 個以上のシェアが集まることで初めて全ての $r_1, \ldots, r_m$ が揃うことから，$r = \sum_{i=1}^{m}$ により秘密を復元する．

例えば，(2,3)-複製秘密分散においては，秘密 $r = r_1 + r_2 + r_3$ のシェアは $(r_1, r_2), (r_2, r_3), (r_3, r_1)$ となる．

2.3 マルチパーティ計算

マルチパーティ計算はパーティ $\{P_i\}_{i=1,\ldots,n}$ によって実行されるプロトコル **Eval** であり，本研究においては特に秘密分散法を利用するものを指す．以降では簡単のため，秘密分散において各パーティが持つシェア $[\![\cdot]\!]_i$ の添え字 i を省略し，単に $[\![\cdot]\!]$ と記述する．

Eval : $([\![\pi_1]\!], \ldots, [\![\pi_t]\!], F) \mapsto [\![\pi']\!]$ は，各参加者が持つ複数のメッセージのシェア $[\![\pi_1]\!], \ldots, [\![\pi_t]\!]$ と，乗算・加算により構成される回路 F から，新たなシェア $[\![\pi']\!]$ を生成し，各参加者に出力するプロトコルである．**Eval** は以下の性質を満たす．

[*1] $H(\cdot)$ は平均情報量とする

© 2018 Information Processing Society of Japan

$$\mathbf{Reveal}(\mathbf{Eval}([\![\pi_1]\!],\ldots,[\![\pi_t]\!],F)) = F(\pi_1,\ldots,\pi_t)$$

2.4 Oblivious RAM

Oblivious RAM(ORAM)[9] は，外部ストレージを用いた安全なデータ管理を可能にするためのアクセスパターン秘匿アルゴリズムである．

ユーザが秘匿化したデータを外部ストレージ (サーバ) へ預け，必要に応じてその中の特定のデータを参照・更新するモデルは，検索可能暗号などにおいてしばしば用いられる．しかし，たとえデータそのものや検索用インデクスの安全性が破られなかったとしても，サーバ上の各記憶領域に対する参照の有無や重複 (＝アクセスパターン) によって，検索されたデータに関する部分情報が漏洩してしまうことが考えられる．

こうした問題に対し，Oblivious RAM はデータアクセスのたびにデータの再秘匿化とランダム置換を適切に行い，複数回のデータアクセスによって生じるアクセスパターンの差異を識別不可能にすることでデータの安全性を高めている．

[3] に述べられる，Oblivious RAM の安全性の定義を以下に示す．

定義 3. $\vec{q} = ((u_1, op_1, S_i), \ldots, (u_M, op_M, S_M))$ を，*Oblivious RAM* に対する M 回のデータ参照クエリとする．ただし，u_i は参照するデータのインデクス，$op_i \in \{\mathrm{read}, \mathrm{write}\}$ はデータに対する操作，S_i は書き込み用データとする．また，$\mathcal{AC}(\vec{q})$ をクエリ $\vec{q}$ におけるアクセスパターン，すなわち記憶領域に対する参照の順序であるとする．この時，等しい長さのクエリ $\vec{q}_1, \vec{q}_2$ について，$\mathcal{AC}(\vec{q}_1)$ と $\mathcal{AC}(\vec{q}_2)$ が計算量的に識別不可能であるならば，*Oblivious RAM* は安全であると呼ぶ．

2.4.1 Path ORAM

Path ORAM[4] は，完全二分木型のデータ構造を用いた Oblivious RAM である．

Path ORAM はデータを格納するためのデータ構造 “tree” と，tree における各データの格納場所を保存する “position map” によって構成される．

Tree は “bucket” と呼ばれる，Z 個のデータを格納できる箱をノードとして持つ高さ p の完全二分木である．全てのデータは秘匿化 (暗号化) され，tree 上のいずれかの bucket に格納される．また，格納される (正規の) データが Z 個に満たない bucket は，空きを全てダミーデータによって埋められ，必ず全 bucket が (正規・ダミー問わず)Z 個のデータを持つようにする．全ての正規データは tree のいずれかの葉 $L_i (i = 0, \ldots, 2^{p-1} - 1)$ と紐付けられ，tree の根 bucket から葉 $L_i$bucket までの path 上のどこかへ格納される．Position map はこのデータ (のインデクス) と葉との対応を記録する．

2.4.2 2 者間 Path ORAM

2 者間で実行される Path ORAM プロトコル [4] は，tree を持つ外部ストレージサーバと，position map を持つユーザの間で，以下のようにデータの読み書きが行われる．

(1) ユーザは操作したいデータのインデクス u を用いて position map を検索し，葉 L を見つけサーバへ送る

(2) サーバは根から葉 L までの path 上にある全ての bucket を取り出し，ユーザへ送る

(3) ユーザは送られた bucket 列にあるデータを全て復号し，その中から目的のデータを取り出す．また，必要なら新規データへ書き換える

(4) ユーザは取り出したデータと紐付く葉を，新たにランダムに選んだ $\tilde{L}$ へと更新し，position map を書き換える

(5) ユーザは復号した全データを再度暗号化し，bucket 列へと格納する．この際，取り出したデータのみ根 bucket へ格納し，それ以外のデータは元の bucket へ格納する

(6) ユーザは bucket 列に対して「追い立て (evict)」を行う．具体的には，bucket 列にある全ての正規データを，自身が紐付く葉と L との最小共通祖先となる bucket へ移動させる．例えば，前の手順にて取り出され根 bucket へ再格納されたデータは，$\tilde{L}$ と L の最小共通祖先にあたる bucket へと移動する

(7) 全ての「追い立て」が完了したら，ユーザは bucket 列をサーバへ送る

(8) サーバは送られた bucket 列によって，根から葉 L までの全ての bucket を上書きする

このプロトコルにおいてサーバが観測しうるのは，ユーザから送られる葉 L と bucket 列である．葉 L はデータやインデクスと独立してランダムに選ばれる値であり，データに関する情報を持たない．bucket 列は全てユーザによって再暗号化されており，サーバがユーザに送った bucket 列と比較しても計算量的に識別不可能である．[*2] したがって Path ORAM は安全である．

また，[4] によれば，tree のサイズは格納するデータ数が N の時，理論的に高さが $p = \lceil \log_2 N \rceil$ 必要，実験上では高さ $p = \lceil \log_2 N \rceil - 1$ で十分とされている．よって Path ORAM の計算量はデータ数 N に対し，position map の探索が $O(\log_2 N)$[*3]，その後のデータ読み出し・再格納が $O(\log_2 N)$ である．

[*2] 利用する暗号方式の安全性に依存する．
[*3] 単純な二分探索の場合．

© 2018 Information Processing Society of Japan

2.4.3 マルチパーティ Path ORAM

Path ORAM を複数台のサーバによるマルチパーティ計算で実行する方式 (以下「マルチパーティ Path ORAM」) が [8] にて提案されている．

マルチパーティ Path ORAM のプロトコルでは，ユーザとサーバという立場の差がなく，全てのサーバが等しく秘密分散された tree および position map を持つ．

データの読み出しの際はマルチパーティ計算により，インデクスのシェア $[\![u]\!]$ を用いて position map を探索し，葉のシェア $[\![L]\!]$ を取り出す．その後 $L \leftarrow \mathbf{Reveal}([\![L]\!])$ を得て，根から葉 L までの全ての bucket を $[\![u]\!]$ によって探索することで目的のデータを得る．データを再格納する際には，position map をランダムな葉のシェア $[\![\check{L}]\!]$ によって更新し，マルチパーティ計算による「追い立て」を行う．この際，path 全体に対する **Reveal** および **Share**(復号と再暗号化) は行わない．

このマルチパーティ Path ORAM の計算量は，position map の探索が $O(N)$，その後のデータ読み出し・再格納が $O(\log_2^3 N)$ となっており，通常のものと比べやや計算量が増している．

2.4.4 再帰的 Path ORAM

上記の 2 者間 Path ORAM では，データ数 N に対し，ユーザの記憶領域が $O(N)$ だけ必要となってしまう (position map のサイズのため)．

そこで，position map を小さな Path ORAM へと再帰的に格納してゆくことで，ユーザに要求される記憶領域を小さくする方法が提案されている [2], [4]．

この方式は，ユーザ・サーバの立場の差が無いマルチパーティ Path ORAM においても，記憶領域ではなく計算量の削減の点で有効となることが知られている [15], [16]．マルチパーティ Path ORAM においては，position map が分散・秘匿化されているため，二分探索のような効率の良い探索手法が使えず線形時間の探索にならざるを得ない．しかし，再帰的に position map を小さな tree へ格納してゆくことで，探索時間を $O(\log_2^2 N)$ に抑えることができる．

3. セキュアな決定木モデルと既存の構成方法

本章では，市川らによって議論・提案された，安全にマルチパーティ計算可能な決定木 [13], [14] のモデルと構成方法を示す．なお，本論文で以下に示すプロトコルは全て (2,3)-複製秘密分散と，複製秘密分散を用いたマルチパーティ計算によって構成されるとする．また，秘密分散は全て 1-bit ごとに行われるとし，l-bit の数 $X = (x_{l-1}, \ldots, x_0)$, $x_i \in \{0, 1\}$ に対応するシェアを $[\![X]\!] = ([\![x_{l-1}]\!], \ldots, [\![x_0]\!])$ と表記する．

3.1 セキュアな決定木のモデル

[13], [14] では，サーバ群とユーザからなる決定木計算モデルが示されている．

Protocol 1 Comp$([\![X]\!], [\![Y]\!], [\![e]\!]) \rightarrow [\![z]\!]$

Require: $[\![X]\!], [\![Y]\!]$: l-bit 数のシェア，$[\![e]\!]$: 1-bit 数のシェア
Ensure: $z = (X = Y)_?(e = 0$ の時)，$z = (X < Y)_?(e = 1$ の時)
  for all $0 \le i \le l-1$ **do**
    $[\![p_i]\!] \leftarrow [\![x_i]\!] \oplus [\![y_i]\!]$
  end for
  $[\![q_{l-1}]\!] \ldots [\![q_0]\!] \leftarrow \mathbf{PrefixOR}([\![p_{l-1}]\!] \ldots [\![p_0]\!])$
  $[\![w^{X<Y}]\!], [\![w^{Y<X}]\!] \leftarrow [\![0]\!]$
  for all $0 \le i \le l-1$ **do**
    $[\![q_i]\!] \leftarrow [\![q_i]\!] \oplus [\![q_{i+1}]\!]$ **if** $i < l-1$
    $[\![\tilde{x}_i]\!] \leftarrow [\![x_i]\!] \cdot [\![q_i]\!]$
    $[\![\tilde{y}_i]\!] \leftarrow [\![y_i]\!] \cdot [\![q_i]\!]$
    $([\![w_i^{x<y}]\!], [\![w_i^{y<x}]\!]) \leftarrow (([\![\tilde{x}_i]\!] \oplus 1) \cdot [\![\tilde{y}_i]\!], [\![\tilde{x}_i]\!] \cdot ([\![\tilde{y}_i]\!] \oplus 1))$
    $[\![w^{X<Y}]\!] \leftarrow [\![w^{X<Y}]\!] \oplus [\![w_i^{x<y}]\!]$
    $[\![w^{Y<X}]\!] \leftarrow [\![w^{Y<X}]\!] \oplus [\![w_i^{y<x}]\!]$
  end for
  $[\![w^{X=Y}]\!] \leftarrow [\![w^{X<Y}]\!] \oplus [\![w^{Y<X}]\!] \oplus 1$
  $[\![z]\!] \leftarrow [\![w^{X=Y}]\!] \cdot ([\![e]\!] \oplus 1) \oplus [\![w^{X<Y}]\!] \cdot [\![e]\!]$

Protocol 2 DT-Online1$([\![X_1]\!], \ldots, [\![X_{h-1}]\!]) \rightarrow [\![C]\!]$

Require: $[\![X_1]\!], \ldots, [\![X_{h-1}]\!]$: 決定木 (高さ h) への入力
Ensure: $[\![C]\!]$: 分類結果
  $[\![z_1]\!] \leftarrow \mathbf{Comp}([\![X_1]\!], [\![Y_1]\!], [\![e_1]\!])$
  $([\![Y_2]\!], [\![e_2]\!])$
    $\leftarrow ([\![Y_{2,0}]\!] \cdot (1 - [\![z_1]\!]) + [\![Y_{2,1}]\!] \cdot [\![z_1]\!], [\![e_{2,0}]\!] \cdot (1 - [\![z_1]\!]) + [\![e_{2,1}]\!] \cdot [\![z_1]\!])$
  $[\![z_2]\!] \leftarrow \mathbf{Comp}([\![X_2]\!], [\![Y_2]\!], [\![e_2]\!])$
  $[\![u_3]\!] \leftarrow [\![z_1]\!] \,\|\, [\![z_2]\!]$
  for $i = 3$ **to** $h-1$ **do**
    $([\![Y_i]\!], [\![e_i]\!]) \leftarrow \mathbf{PO}_i.\mathbf{ReadAndRemove}([\![u_i]\!])$
    $[\![z_i]\!] \leftarrow \mathbf{Comp}([\![X_i]\!], [\![Y_i]\!], [\![e_i]\!])$
    $[\![u_{i+1}]\!] \leftarrow [\![u_i]\!] \,\|\, [\![z_i]\!]$
  end for
  $[\![C]\!] \leftarrow \mathbf{PO}_h.\mathbf{ReadAndRemove}([\![u_h]\!])$

Protocol 3 DT-Offline1$([\![C]\!], ([\![Y_1]\!], [\![e_1]\!]), \ldots, ([\![Y_{h-1}]\!], [\![e_{h-1}]\!]))$

Require: $[\![C]\!], ([\![Y_1]\!], [\![e_1]\!]), \ldots, ([\![Y_{h-1}]\!], [\![e_{h-1}]\!])$: Protocol 2 で使用したデータ
  for $i = 3$ **to** $h-1$ **do**
    $\mathbf{PO}_i.\mathbf{Add}(([\![Y_i]\!], [\![e_i]\!]))$
  end for
  $\mathbf{PO}_h.\mathbf{Add}([\![C]\!])$

このモデルでは，サーバ群が決定木に関する情報を秘密分散した状態で保持している．決定木への入力データを持つユーザは秘密分散した入力値を各サーバへ送信し，入力を受けたサーバ群はマルチパーティ計算によって入力データへの分類計算を行い結果をユーザへと返す．

このモデルにおけるプロトコルの安全性は以下のように定義されている．

定義 4. 各サーバは結託しないと仮定する．決定木計算の過程で，それぞれのサーバがユーザの入力に関する情報を一切得られないならば，決定木の構成が安全であると呼ぶ．

以下，各構成方法の安全性について詳細は [13], [14] を参照されたし．

© 2018 Information Processing Society of Japan

Protocol 4 DT-Online2$(\llbracket X_1 \rrbracket, \ldots, \llbracket X_{h-1} \rrbracket) \to \llbracket C \rrbracket$

Require: $\llbracket X_1 \rrbracket, \ldots, \llbracket X_{h-1} \rrbracket$: 決定木 (高さ h) への入力
Ensure: $\llbracket C \rrbracket$: 分類結果

$\llbracket z_1 \rrbracket \leftarrow \mathbf{Comp}(\llbracket X_1 \rrbracket, \llbracket Y_1 \rrbracket, \llbracket e_1 \rrbracket)$
$(\llbracket Y_2 \rrbracket, \llbracket e_2 \rrbracket, \llbracket L'_{3,0} \rrbracket, \llbracket L'_{3,1} \rrbracket)$
  $\leftarrow (\llbracket Y_{2,0} \rrbracket \cdot (1 - \llbracket z_1 \rrbracket) + \llbracket Y_{2,1} \rrbracket \cdot \llbracket z_1 \rrbracket,$
  $\llbracket e_{2,0} \rrbracket \cdot (1 - \llbracket z_1 \rrbracket) + \llbracket e_{2,1} \rrbracket \cdot \llbracket z_1 \rrbracket,$
  $\llbracket L_{3,0} \rrbracket \cdot (1 - \llbracket z_1 \rrbracket) + \llbracket L_{3,2} \rrbracket \cdot \llbracket z_1 \rrbracket,$
  $\llbracket L_{3,1} \rrbracket \cdot (1 - \llbracket z_1 \rrbracket) + \llbracket L_{3,3} \rrbracket \cdot \llbracket z_1 \rrbracket)$
$\llbracket z_2 \rrbracket \leftarrow \mathbf{Comp}(\llbracket X_2 \rrbracket, \llbracket Y_2 \rrbracket, \llbracket e_2 \rrbracket)$
$\llbracket u_3 \rrbracket \leftarrow \llbracket z_1 \rrbracket \parallel \llbracket z_2 \rrbracket$
$\llbracket L_3 \rrbracket \leftarrow \llbracket L'_{3,0} \rrbracket \cdot (1 - \llbracket z_2 \rrbracket) + \llbracket L'_{3,1} \rrbracket \cdot \llbracket z_3 \rrbracket$
for $i = 3$ **to** $h - 1$ **do**
  $L_i \leftarrow \mathbf{Reveal}(\llbracket L_i \rrbracket)$
  for $j = 1$ **to** i **do**
    $\mathbf{B} \leftarrow \mathbf{PO'}_i.\mathbf{GetBucket}(L_i, j)$
    $(\llbracket Y_i^{(j)} \rrbracket, \llbracket e_i^{(j)} \rrbracket, \llbracket L_{i+1,0}^{(j)} \rrbracket, \llbracket L_{i+1,1}^{(j)} \rrbracket)$
      $\leftarrow \mathbf{B}.\mathbf{ReadAndRemove}(\llbracket u_i \rrbracket)$
  end for
  $(\llbracket Y_i \rrbracket, \llbracket e_i \rrbracket, \llbracket L_{i+1,0} \rrbracket, \llbracket L_{i+1,1} \rrbracket)$
    $\leftarrow (\sum_j \llbracket Y_i^{(j)} \rrbracket, \sum_j \llbracket e_i^{(j)} \rrbracket, \sum_j \llbracket L_{i+1,0}^{(j)} \rrbracket, \sum_j \llbracket L_{i+1,1}^{(j)} \rrbracket)$
  $\llbracket z_i \rrbracket \leftarrow \mathbf{Comp}(\llbracket X_i \rrbracket, \llbracket Y_i \rrbracket, \llbracket e_i \rrbracket)$
  $\llbracket u_{i+1} \rrbracket \leftarrow \llbracket u_i \rrbracket \parallel \llbracket z_i \rrbracket$
  $\llbracket L_{i+1} \rrbracket \leftarrow \llbracket L_{i+1,0} \rrbracket \cdot (1 - \llbracket z_i \rrbracket) + \llbracket L_{i+1,1} \rrbracket \cdot \llbracket z_i \rrbracket$
end for
$L_h \leftarrow \mathbf{Reveal}(\llbracket L_h \rrbracket)$
for $j = 1$ **to** h **do**
  $\mathbf{B} \leftarrow \mathbf{PO'}_h.\mathbf{GetBucket}(L_h, j)$
  $\llbracket C^{(j)} \rrbracket \leftarrow \mathbf{B}.\mathbf{ReadAndRemove}(\llbracket u_h \rrbracket)$
end for
$\llbracket C \rrbracket \leftarrow \sum_j \llbracket C^{(j)} \rrbracket$

Protocol 5 DT-Offline2$(\llbracket C \rrbracket, \{(\llbracket Y_i \rrbracket, \llbracket e_i \rrbracket, \llbracket L_{i+1,0} \rrbracket, \llbracket L_{i+1,1} \rrbracket)\}_{i<h})$

Require: $\llbracket C \rrbracket, \{(\llbracket Y_i \rrbracket, \llbracket e_i \rrbracket, \llbracket L_{i+1,0} \rrbracket, \llbracket L_{i+1,1} \rrbracket)\}_{i<h}$: Protocol 4 で使用したデータ

$\llbracket \tilde{L}_h \rrbracket \leftarrow \mathbf{GetRandomLeaf}(h - 1)$
$\mathbf{PO'}_h.\mathbf{Root}.\mathbf{Add}(\llbracket C \rrbracket, \llbracket \tilde{L}_h \rrbracket)$
$\mathbf{PO'}_h.\mathbf{Evict}(L_h)$
for $i = h - 1$ **to** 3 **do**
  $\llbracket L_{i+1,0} \rrbracket \leftarrow \llbracket L_{i+1,0} \rrbracket \cdot \llbracket z_i \rrbracket + \llbracket \tilde{L}_{i+1} \rrbracket \cdot (1 - \llbracket z_i \rrbracket)$
  $\llbracket L_{i+1,1} \rrbracket \leftarrow \llbracket L_{i+1,1} \rrbracket \cdot (1 - \llbracket z_i \rrbracket) + \llbracket \tilde{L}_{i+1} \rrbracket \cdot \llbracket z_i \rrbracket$
  $\llbracket \tilde{L}_i \rrbracket \leftarrow \mathbf{GetRandomLeaf}(i - 1)$
  $\mathbf{PO'}_i.\mathbf{Root}.\mathbf{Add}((\llbracket Y_i \rrbracket, \llbracket e_i \rrbracket, \llbracket L_{i+1,0} \rrbracket, \llbracket L_{i+1,1} \rrbracket), \llbracket \tilde{L}_i \rrbracket)$
  $\mathbf{PO'}_i.\mathbf{Evict}(L_i)$
end for
$\llbracket L_{3,0} \rrbracket$
  $\leftarrow \llbracket L_{3,0} \rrbracket \cdot (1 - (1 - \llbracket z_1 \rrbracket) \cdot (1 - \llbracket z_2 \rrbracket)) + \llbracket \tilde{L}_3 \rrbracket \cdot (1 - \llbracket z_1 \rrbracket) \cdot (1 - \llbracket z_2 \rrbracket)$
$\llbracket L_{3,1} \rrbracket \leftarrow \llbracket L_{3,1} \rrbracket \cdot (1 - (1 - \llbracket z_1 \rrbracket) \cdot \llbracket z_2 \rrbracket) + \llbracket \tilde{L}_3 \rrbracket \cdot (1 - \llbracket z_1 \rrbracket) \cdot \llbracket z_2 \rrbracket$
$\llbracket L_{3,2} \rrbracket \leftarrow \llbracket L_{3,2} \rrbracket \cdot (1 - \llbracket z_1 \rrbracket \cdot (1 - \llbracket z_2 \rrbracket)) + \llbracket \tilde{L}_3 \rrbracket \cdot \llbracket z_1 \rrbracket \cdot (1 - \llbracket z_2 \rrbracket)$
$\llbracket L_{3,3} \rrbracket \leftarrow \llbracket L_{3,3} \rrbracket \cdot (1 - \llbracket z_1 \rrbracket \cdot \llbracket z_2 \rrbracket) + \llbracket \tilde{L}_3 \rrbracket \cdot \llbracket z_1 \rrbracket \cdot \llbracket z_2 \rrbracket$

3.2 方式 1: ナイーブなセキュア決定木の構成方法

通常，決定木は 2.1 節に示すように，属性値の比較を行いながら根ノードから葉ノードへと適切な路を辿って分類結果を得る．

しかしマルチパーティ計算においては，このような入力データ依存の計算方法では秘密の漏洩に繋がる恐れがある．例えば，根ノードで入力の属性値 X_0 と定数 Y_0 の比較を行い，その結果によって次の枝で属性値 X_1 と定数 Y_1 を用いるか，または X_2 と Y_2 を用いるかが変わるような決定木があるとする．この決定木を通常の方法で計算する場合，たとえセキュアなマルチパーティ計算により秘密の入力 X_0 を秘匿したままで値 Y_0 との比較を行ったとしても，次に用いるデータが (X_1, Y_1) であるか (X_2, Y_2) であるかという判別において，秘密 X_0 が Y_0 より大きいか小さいかという比較結果が漏洩してしまう．

従って，入力を秘匿したまま安全に決定木での分類計算を行う場合，入力の値に依存しない計算方法によって分類結果を得る必要がある．最もナイーブな方法として，[13] では以下のプロトコルが示されている．

(1) 各サーバはあらかじめ，決定木の各ノードで比較されるデータ (定数 Y_i) を秘密分散し保持する

(2) ユーザは決定木への入力を秘密分散により秘匿化し，各サーバへ送る

(3) 入力を受けたサーバは，決定木の全てのノードにおける比較をマルチパーティ計算によって行い，それぞれの葉に対応する真理値 (のシェア) の集合を得る

(4) サーバは全ての葉の出力データと，葉に対応する真理値の集合との総積を計算し，それらの総和をユーザへ返す

3.3 方式 2: Path ORAM を用いたセキュア決定木の構成方法

前述のナイーブな構成法は，入力値に関わらず全てのノードの比較計算を行うため，決定木の高さ h に対して $O(2^h)$ の計算時間がかかってしまい，効率が悪い．

そこで [14] では効率を改善するため，[8] のマルチパーティ Path ORAM を利用し，決定木の必要なノードだけを計算して安全に分類結果を得る構成方法が示されている．

その際，各ノードでの比較演算子の差異 $(=, <)$ による入力情報の漏洩を防ぐため，通常の比較回路の代わりに Protocol 1 を用いる．このプロトコルでは，比較演算子 $\{=, <\}$ を 1-bit 数 $\{0, 1\}$ のシェアとして秘匿し，プロトコルの入力に含めることで等号・不等号のいずれが計算されているかを判別不能としている．ただし Protocol 1 中の "PrefixOR" は，入力ビット列に "1" の値が含まれる時，値 1 となるビットより右に位置する全てのビットの値を 1 と

© 2018 Information Processing Society of Japan

する秘密計算プロトコルである．

この比較プロトコルと，決定木の各分岐ごとの「入力データ参照パターン」を一様にする冗長化 [13] により，次のように安全な決定木を構成できる．

(1) サーバは冗長化された決定木について，木の深さごとに Path ORAM を構築し，ノードのデータ (定数 Y_i，比較演算子 $e_i \in \{=, <\}$) を秘密分散して格納する．この時，データに付与される Path ORAM 上のインデクスは，対応するノードに至るための比較結果の連結とする．すなわち，決定木における深さ 3 のノードのデータ $(Y_{3,0}, e_{3,0}), \ldots, (Y_{3,3}, e_{3,3})$ はそれぞれインデクス $00, 01, 10, 11$ が与えられ，データ格納木の高さが 3 である Path ORAM，$\mathbf{PO}_3$ へ格納される．決定木における深さ 4 のデータは $(Y_{4,0}, e_{4,0}), \ldots, (Y_{4,7}, e_{4,7})$ の 8 つであり，それぞれインデクス $000, 001, \ldots, 111$ を与えられて，データ格納木の高さが 4 である Path ORAM，$\mathbf{PO}_4$ に格納される．ただし，決定木における深さ 1 のデータ (Y_1, e_1)，深さ 2 のデータ $(Y_{2,0}, e_{2,0}), (Y_{2,1}, e2, 1)$ については Path ORAM を用いず，シェアを通常のメモリに格納する

(2) ユーザは入力データを秘密分散し各サーバへ送る

(3) 入力を受けたサーバは，Protocol 2 を実行し，結果をユーザへ返す

(4) サーバは Protocol 3 により各データを再格納する

ただし，Protocol 2, 3 における **ReadAndRemove**, **Add** は [8] に示される Path ORAM へのアクセス及びデータ格納のプロトコルである．

また，Protocol 2, 3 は各 Path ORAM に対するデータ読み出しを一挙に行い，ユーザへ結果を返した後でまとめて再格納を行っている．これは通常の Path ORAM とは異なる用法の，[14] に倣った Online/Offline 型のプロトコルである．4.1 節の実験結果に示すように Path ORAM のデータ格納は計算コストが高いため，ユーザへのレスポンス効率向上のため再格納の処理を分けている．

3.4 方式 3: 芋づる式 Path ORAM によるセキュア決定木の構成方法

今回想定する決定木の構築においては，各ノードでの比較用データは比較的小さなものである．[*4] そのため，上記のように単純に Path ORAM を利用した場合，Path ORAM 上のデータ格納位置を記録する position map のサイズがノードのデータ全体より大きくなり，効率の低下を招く恐れがある．

そこで [14] ではプロトコルを改良し，各 Path ORAM の呼び出し時に position map が不要となる方式を提案している．その方式を以下に示す．

(1) 単純に Path ORAM を利用する場合と同様に，サーバは冗長化した決定木の深さごとに Path ORAM，$\mathbf{PO'}_i$ を構築し，各深さ i のノードのデータを格納する．ただしこの時，深さ i のデータのシェア $([\![Y_{i,j}]\!], [\![e_{i,j}]\!])$ に対して，「次に呼ばれるべきデータの，$\mathbf{PO'}_{i+1}$ における格納先の葉」を付加する．すなわち，例えば深さ 2 のノードのデータ $([\![Y_{2,0}]\!], [\![e_{2,0}]\!])$ において，入力 X との比較結果が “0” だったなら $([\![Y_{3,0}]\!], [\![e_{3,0}]\!])$ が，“1” だったなら $([\![Y_{3,1}]\!], [\![e_{3,1}]\!])$ が次の比較計算で用いられるとした時，$\mathbf{PO'}_3$ における $([\![Y_{3,0}]\!], [\![e_{3,0}]\!])$ の格納先 $L_{3,0}$ と，$([\![Y_{3,1}]\!], [\![e_{3,1}]\!])$ の格納先 $L_{3,1}$ を付加し $([\![Y_{2,0}]\!], [\![e_{2,0}]\!], [\![L_{3,0}]\!], [\![L_{3,1}]\!])$ とする．この付加情報は，葉ノードのデータ (=分類結果) と根ノードのデータ $([\![Y_1]\!], [\![e_1]\!])$ を除く全てのデータに付加しておく

(2) ユーザは入力データを秘密分散し各サーバへ送る

(3) 入力を受けたサーバは Protocol 4 を実行し，結果をユーザへ返す

(4) サーバは Protocol 5 により各データを更新・再格納する．

ただし，Protocol 4, 5 で用いられる **GetBucket**, **B.ReadAndRemove**, **GetRandomLeaf**, **Root.Add**, **Evict** はそれぞれ，該当の path 上にある bucket を取り出す関数，bucket からデータを読み出す関数，ランダムな葉 (引数-bit の数) を得る関数，Path ORAM の根ノード bucket にデータを格納する関数，path 上の全てのデータを「追い立て」る関数である．

4. 実験

筆者らは，3 章で紹介した安全にマルチパーティ計算可能な決定木の構成を，それぞれ実装し実行時間を測定した．本章ではその結果を示し，各構成でかかる実行時間を比較する．

なお，以下の実験は全て，CPU: 3.20GHz 6 core Intel Core i7, RAM: 20GB の端末上で動作する仮想マシン (OS: Ubuntu16.04, Processor: 6 core, RAM: 12GB, clang version 3.8, OpenSSL 1.0.2) において行った．

4.1 Path ORAM の Online/Offline 計算時間

まず初めに，[14] および本論文 3.3 節・3.4 節の構成において Online/Offline 型のプロトコルが用いられた根拠である，Path ORAM の Online 計算時間と Offline 計算時間を計測する．ただし，Path ORAM へ格納するデータのサイズは 8-bit 数とする．

[*4] 例として，[13] ではデータを全て 8-bit 数と想定している．

© 2018 Information Processing Society of Japan

表 1　マルチパーティ Path ORAM(非再帰) の Online/Offline 計算時間 (ms)

高さ p	Online	Offline
6	1028	4552
7	1889	5837
8	3123	7291
9	6018	8673
10	12969	11219

表 2　再帰的マルチパーティ Path ORAM(分割要素数 2) の Online/Offline 計算時間 (ms)

高さ p	Online	Offline
6	2202	11215
7	3451	17176
8	4632	23413
9	7448	37758
10	10029	48546

表 3　再帰的マルチパーティ Path ORAM(分割要素数 4) の Online/Offline 計算時間 (ms)

高さ p	Online	Offline
6	1556	6519
7	2253	9613
8	3177	13765
9	5016	20911
10	5549	23435

表 4　再帰的マルチパーティ Path ORAM(分割要素数 8) の Online/Offline 計算時間 (ms)

高さ p	Online	Offline
6	1203	5227
7	2468	8960
8	3323	11227
9	4522	16194
10	6615	18736

表 1 は，マルチパーティ Path ORAM(非再帰) における “tree の” 高さ p ごとの，データアクセス 1 回における「データアクセスクエリを受けてからデータを取り出すまでの時間＝ Online 計算時間」と「取り出したデータを再格納し次のクエリを受け付け可能になるまでの時間＝ Offline 計算時間」をそれぞれ計測したものである．

表 2–4 は，再帰的マルチパーティ Path ORAM について，表 1 と同様の計測を行った結果である．表 2 は各再帰で position map を分割する際，1 ブロックあたりの要素数を 2 としたときの計測である．表 3 は各再帰で position map を分割する際，1 ブロックあたりの要素数を 4 としたときの計測である．表 4 は各再帰で position map を分割する際，1 ブロックあたりの要素数を 8 としたときの計測である．

いずれの場合でも，Offline 計算時間が全体の計算時間の中で大きな割合を有していることがわかる．

4.2　各構成方法による決定木の分類計算時間

次に，3 章で示したそれぞれの構成方法について，実行時間を計測する．ただし，方式 2 については，通常の (非再帰的な)Path ORAM を利用した場合を方式 2-1，再帰的 Path ORAM を利用した場合を方式 2-2 として，それぞれ実行時間を計測する．方式 2-2 について，各再帰的 Path ORAM の position map は分割要素数 8 とする．

また，決定木の各ノードで比較されるデータと出力される分類結果は全て 8-bit 数であるとし，決定木は完全二分木の形を取るものとした．[*5]

表 5 に，“決定木の” 高さ h ごとの，各構成に対する入力 1 回ごとの実行時間を示す．表の各数値は “Online 計算時間/Offline 計算時間” を表している．ただし，ナイーブな構成については Offline 計算が存在しないため，値を省略する．

5.　考察

5.1　Path ORAM の実行時間

通常の Path ORAM においては，tree の高さ p に対して position map の要素数は 2^{p-1} となることから，Online 計算時間も指数的に上昇していることがわかる．一方，Offline 計算時間は p の多項式オーダであるため，$p = 10$ においては Online 計算時間が Offline 計算時間を越える様子が見られる．

再帰的 Path ORAM では Online 計算時間が p の多項式となっており，position map へのアクセスコストが削減できたとわかる．Position map の分割要素数を大きくすることでより強くアクセスコストを削減できるが，小さな Path ORAM では分割要素数を過度に大きくしてしまうと，アルゴリズムの上では position map への線形探索を行うのと変わりない計算量となってしまう．実際に p が高々 1 桁の場合，分割要素数を 4 とした時と 8 とした時で，Online 計算時間にはほとんど差が見られないとわかる．[*6]

5.2　決定木の分類計算時間

方式 1 においては，h に対して $2^{h-1} - 1$ 回の比較を行うため，必然的に実測の計算時間も指数的な伸び方となる．また方式 2-1 も，各 Path ORAM での position map への探索が線形であることから，計算時間が指数的な伸びとなる．よって方式 2-1 では，原理的にナイーブな構成より高速に動作するのは難しいと考察できる．

方式 2-2 の Online 計算時間は，$h \geq 9$ で方式 2-1 の On-

[*5] 実際の決定木が完全二分木となることはおよそ無いと思われる．しかし，実際の決定木は目的や教師データによって木の構造 (剪定の度合い) が全く異なってしまうため，実験データとして対照性に欠ける恐れがある．そのため今回は，決定木の高さとノード数 (比較計算回数) の相関が明確である完全二分木を想定した．

[*6] Offline 計算時間の差は，position map を格納するための Path ORAM の個数 (p と分割要素数に依存) の差のためである．

© 2018 Information Processing Society of Japan

表 5　各構成による決定木の Online/Offline 計算時間 (ms)

高さ h	方式 1	方式 2-1	方式 2-2	方式 3
6	765/—	2515/9890	2848/10815	2142/10576
7	1688/—	4715/16758	5265/19016	3111/16466
8	4275/—	7520/24829	9225/32143	4803/25149
9	8578/—	14136/34629	12152/45882	6142/32532
10	19148/—	28355/49121	20912/72829	7878/39956

line 計算時間を下回ることから，指数オーダよりも小さな多項式オーダであると見られる．これは再帰的 Path ORAM の Online 計算時間が多項式オーダであることとも符合する．ただし方式 1 の Online 計算時間と比較すると，$h = 10$ でも方式 2-2 の Online 計算時間がやや大きくなっている．方式 2-2 が方式 1 より高速に動作できるのは，$h \geq 11$ の時となる．

方式 3 は，方式 2-1,2-2 よりも高速であることがわかる．さらに，Path ORAM を用いた 3 つの構成の中で唯一，$h \geq 9$ でナイーブな構成よりも効率良く応答 (Online 計算) 可能である．芋づる式 Path ORAM の特徴は，position map のアクセスにかかる計算量が $O(1)$ で済むことである．このことと実験結果から，Path ORAM においては position map へのアクセス効率が全体のデータアクセス速度を向上させる非常に重要な要素であると考察できる．

方式 1，すなわちナイーブな計算に対する優位性という点では，Path ORAM によるいずれの構成も不十分である．今回は決定木を完全二分木と仮定したが，現実の決定木は枝の剪定により一部のノードを削除する．そのため実際の分類計算において，方式 1 の応答速度は実験値より高速となることが想定され，他の方式が優位となる場面はより少なくなると考えられる．また，効率・精度に優れるとされる決定木学習法であるランダムフォレストでは，木の高さは高々 5 ないし 1 となる場合が多いため，方式 1 でも十分高速に動作できると想定される．

ただし，葉の数 10000〜となるような巨大な決定木をマルチパーティ計算する事例 [5] もあり，こうした用途においては方式 3 が方式 1 より高速である．

6. 結論

本論文ではマルチパーティ計算可能な決定木を実装し，実行時間を評価した．その結果，決定木の高さが 9 以上ならば，芋づる式 Path ORAM による方式がナイーブな方法より高速であるとわかった．

参考文献

[1] A. Blum, C. Dwork, F. McSherry, and K. Nissim, “Practical privacy: the sulq framework,”Proc. ACM, pp.128–138, 2005.

[2] E. Shi, T.-H.H. Chan, E. Stefanov, and M. Li, “Oblivious RAM with $O(\log N)^3$ worst-case cost,”ASIACRYPT, pp.197–214, 2011.

[3] E. Stefanov, E. Shi, and D. Song, “Towards Practical Oblivious RAM, ”NDSS, 2012.

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

[5] K. Hamada, S. Hasegawa, K. Misawa, K. Chida, S. Ogishima, and M. Nagasaki, “Privacy-Preserving Fisher’s Exact Test for Genome-Wide Association Study,”International Workshop on Genome Privacy and Security (GenoPri), 2017.

[6] M. Backes, P. Berrang, M. Bieg, R. Eils, C. Herrmann, M. Humbert, and I. Lehmann, “Identifying Personal DNA Methylation Profiles by Genotype Inference,”IEEE Symposium on Security and Privacy, 2017.

[7] M. Ito, A. Saito, and T. Nishizeki, “Secret Sharing Schemes Realizing General Access Structures,”Proc. IEEE Global Telecommunication Conf., Globecom 87, pp. 99–102, 1987.

[8] M. Keller and P. Scholl, “Efficient, Oblivious Data Structures for MPC,”ASIACRYPT 2014, volume 8874 of LNCS, pp. 506–525.

[9] O. Goldreich and R. Ostrovsky, “Software Protection and Simulation on Oblivious RAMs,”J. ACM, 1996.

[10] R. Bost, R.A. Popa, S. Tu, and S. Goldwasser, “Machine Learning Classification over Encrypted Data,”NDSS, 2015.

[11] D.J. Wu, T. Feng, M. Naehrig, and K.E. Lauter, “Privately Evaluating Decision Trees and Random Forests,”PoPETs 2016, 2016(4):335–355.

[12] Y. Lindell and B. Pinkas, “Privacy Preserving Data Mining,”Journal of Cryptology, pp.36–54, 2000.

[13] 市川 敦謙, 菊池 亮, 尾形 わかは, “決定木の秘密計算における効率比較と事前計算を用いた効率化, ” 暗号と情報セキュリティシンポジウム, 2016.

[14] 市川 敦謙, 尾形 わかは, “Oblivious RAM を用いた効率的な決定木のマルチパーティ計算, ” 暗号と情報セキュリティシンポジウム, 2017.

[15] 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—,” コンピュータセキュリティシンポジウム, 2014.

[16] 和田 紘帆, 西田 直央, 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（2）— プロトコルの設計と評価—,” コンピュータセキュリティシンポジウム, 2014.

© 2018 Information Processing Society of Japan