LOUDS と準同型暗号による秘匿決定木評価

須藤　弘貴† 　清水　佳奈†

† 早稲田大学基幹理工学部情報理工学科　〒 169–8555 東京都新宿区大久保 3 – 4 – 1

E-mail: †hsudo108@ruri.waseda.jp, ††shimizu.kana@waseda.jp

あらまし　ビッグデータ解析で得られた知見をエンドユーザーが利用できる機械学習サービスの提供が普及し始める一方で，学習済み分類器の公開による機密情報漏えいの危険性が指摘されており，システムを安全に運用するための技術開発が急務となっている．このような背景を鑑み，本研究ではユーザーのデータとサーバーのモデルを互いに明かすことなく，決定木による分類結果を計算する秘匿計算技術を提案する．決定木の分類結果を秘匿計算する従来研究として，計算量が木の高さの指数オーダーとなる手法が知られている．提案手法では木の簡潔データ構造である LOUDS と紛失通信と呼ばれる暗号技術を効果的に組み合わせることにより，木の高さに線形な計算量を実現した．

キーワード　準同型暗号, 秘匿計算, LOUDS, プライバシー保護, 機械学習, 決定木

1. 序　　論

本研究の背景と目的： ビッグデータの解析結果をエンドユーザが利用できるシステムが普及しつつある．特に，機械学習を用いたシステムは実社会でサービス化され始めており [10, 2]，大きな注目を集めている．このようなサービスの一つとして次のようなシナリオが想定される．企業などのデータ解析拠点（あるいは個人）が独自に収集したデータを用いて分類器の学習を行った後，学習済みの分類器を web 上で API を通じて利用可能にする．サービスの利用者は，自身のデータを API 経由で分類器に入力し，予測結果を得ることができる．例えば，遺伝子診断企業 A が多数の患者のゲノム情報をもとに疾患の診断を行う分類器の学習を行ったとする．利用者 B は A の提供する分類器に対して，自身のゲノム情報を入力することにより，将来の疾患リスクの予測を行うことができる．

こういったサービスでは多くの利用者が容易にビッグデータの恩恵にあずかれるという利点があるものの，セキュリティリスクという重大な問題がある．前述の遺伝子診断の例では，サービスに用いられるクエリ（ゲノム情報）には利用者のプライバシ情報が含まれるため，利用者のプライバシリスクがある．一方で，クエリの送信をせずに分類器を利用可能にするには，分類器自体を公開する必要性が生じるが，このような措置には別のリスクがある．例えば，分類器の学習に用いる訓練データにはゲノムのようなデータ提供者の個人情報の他，企業がコストをかけて収集したノウハウが含まれるが，最近の研究 [8] において，学習済みの分類器から訓練データに関する情報が漏洩する危険性が指摘されている．また，分類サービスの提供により収益を上げることを想定した場合は，学習器自体の公開はビジネスモデルとして適さない．上記に述べたセキュリティの問題を解決するために，学習器に集約されたビッグデータの解析結果を安全に活用するための技術開発が急務となっている．

このような問題の解決には，データの中身を秘匿したまま情報解析を行う**秘匿計算**の応用が効果的であると考えられる．そこで本研究では，**ユーザーがクエリの中身をサーバーに対して秘密にしたまま，サーバーの持つ分類器を用いて分類結果を計算し，ユーザーだけがその結果を得ることのできる秘匿計算技術の開発を目的**とする．より具体的には，本研究で用いる分類器を**決定木**と定め，以下の要件を満たす二者間秘匿計算プロトコルの開発を行う．図 1 は下記要件の模式図である．

要件 1. サーバーは学習済みの決定木 T（木構造，及び各ノードで定めるパラメータの集合）を持つ．ユーザーは決定木に対する入力 $\boldsymbol{x}$ を持つ．ユーザーは $\boldsymbol{x}$ をサーバーに知らせることなく，決定木 T による分類結果を得る．ユーザーは分類結果以外の情報を得ず，サーバーは $\boldsymbol{x}$ に関するいかなる情報も得ない．

決定木は予測根拠の説明が容易な性質を持つことから実社会での利用と親和性が高く，医療診断やマーケティングなどで広く普及している．そのため本提案手法により，利用者とデータ提供者双方のプライバシーを損なうことなく，医療機関や企業などのデータ収集拠点において得られた知見を速やかに社会に還元できることが期待できる．

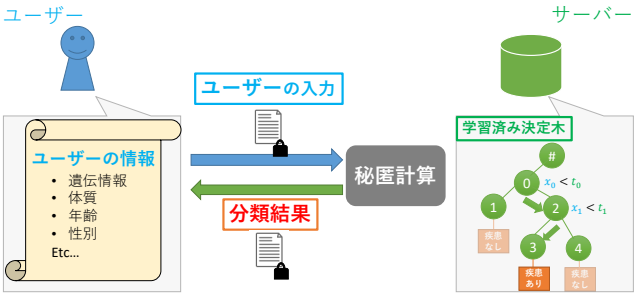

図 1: 提案手法の問題設定.

本研究の貢献： 秘匿計算とは単一の技術を指すものではなく，データの中身を秘匿したまま情報解析を行う技術の総称である．秘匿計算の実現には様々な方法が提案されているが，汎用性と計算性能を両立する手法の開発は極めて困難である．例えば Garbled Circuit や完全準同型暗号は，理論上ではいかなる演算も秘匿化したまま計算することが可能とされているが，実

際には膨大な計算コストを要する．そのため，秘匿計算技術を開発する際には，アプリケーションに応じて計算の内容をよく精査し，重いコストが必要となる暗号計算を最小限にとどめるように工夫を凝らしたアルゴリズムを個別に設計する必要がある．関連研究との詳細な比較は次節以降にて述べるが，本研究では以下の技術的工夫により，安全性を損なうことなく，従来手法よりも計算性能及び汎用性に優れた手法を開発した．

（1） 決定木の木構造を LOUDS と呼ばれる簡潔データ構造で表現し，木の走査に必要な計算を大幅に効率化した．(4.4 節にて詳細を述べる)

（2） 決定木の各ノードで行われる分岐判定は暗号化されたまま行われる．この判定結果を復号することなく，枝の選択を行うことのできる新しい暗号技術「eROT」を開発した．(4.3.3 節にて詳細を述べる)

従来手法では決定木の高さ d に対して $O(2^d)$ の計算量が必要であったのに対し，提案手法は d に線形な計算量のみが必要となる．4.5 節にて計算量の理論的な議論を行い，5. 節の実験にて実際の計算量と理論性能がよく一致することを示す．また，提案手法は木構造を自由に走査することが可能なため，決定木のみに特化して設計された従来手法とは異なり，木による検索など多様な応用も検討可能な汎用性を持つ．

2. 関 連 研 究

決定木による分類結果を計算する二者間秘匿計算プロトコルはこれまでにいくつか提案されている．Brickell らの手法 (2007) [4] と Barni らの手法 (2009) [1] はともに加法準同型暗号と Yao's grabled circuit を組合わせた手法である．一方，Bost ら (2014) [3] は SWHE を用いる手法を提案した．[3] の手法は決定木を決定変数の多項式として扱い，SWHE を用いてこれを計算し決定木を評価する．これらの手法は計算量・通信量の面でコストが大きいという問題があった．より実用的な計算時間，通信量を実現した手法として Wu らの手法 (2016) [14] が挙げられる．[14] は加法準同型暗号を用いるアプローチを採る．[14] では，木の構造を完全二分木に制約し，各節点の分類規則を表現したビット列を暗号技術に適用する手法が提案されている．しかし，このような方法で分類規則を表現すると，ビット列のパターンが木の高さに対して指数的に増加し，計算量・通信量が指数時間となる問題があった．これに対し，提案手法は高さに対して線形なオーダーであり，決定木の高さが高い場合の効率化を実現した．[14] が性能的に最も実用的であることから，6. 節では提案手法とのオーダーにおける比較を行う．

提案手法の採る効率的なデータ構造と暗号技術を組合わせるアプローチは清水らの手法 (2016) [12] の流れを汲むものである．[12] は PBWT というデータ構造と暗号技術を組合わせることにより効率的に文字列検索を行う手法を提案した．[12] では再帰的にアクセス可能なデータ構造と暗号技術を組合わせる手法 (ROT) が提案され，これを用いることにより PBWT と暗号技術の組み合わせを実現した．提案手法は ROT と本稿で提案する ROT の拡張技術 (eROT) を LOUDS と組合わせる．これにより，効率的に決定木の分類結果を秘匿計算する．

3. 準　　備

本節では，提案手法の説明に用いる記法や定義，要素技術，セキュリティモデルについて説明する．個々の要素技術の説明に移る前に，提案手法の概要と必要な要素技術について述べる．提案手法では決定木の出力を得るために必要な計算を大きく二つに分ける．(1) ユーザーの持つ属性値とサーバーの持つ決定木から，各節点で選択される枝を決定する計算と，(2) 枝の選択に基づき木構造を順次辿り，葉に関連付けられた値を出力する計算である．提案手法では**加法準同型暗号**を両者においてプライバシー保護の最も基礎となる暗号技術として用いる．選択される枝の決定には量的変数の場合，属性値と閾値との比較が必要である．そのため，お互いの機密情報を漏洩せずに比較を行うために**加法準同型暗号**を用いた二者間の比較プロトコルを使用する．木構造の走査においては，高さに線形なオーダーを実現するために **LOUDS** の性質を利用し，**紛失通信**と呼ばれる暗号技術と組み合わせる．

3.1 基本的な記法

本稿ではベクトル $\boldsymbol{v}$ を $(v_0, \ldots, v_n)$ のように表記する．また，$\boldsymbol{v}$ の i 番目の要素を $\boldsymbol{v}[i]$ と表記する．$\{a_0, \ldots, a_n\}$ は集合を表す表記とする．$\{a_i\}_{i=0}^{n}$ は $\{a_0, \ldots, a_n\}$ と同じ集合を示す．さらに，ベクトルの「循環」操作を定義する．本稿では n 次元ベクトル $\boldsymbol{v}$ について，各要素が $\hat{\boldsymbol{v}}[(i+r)_{\mathsf{mod}\ n}] = \boldsymbol{v}[i]$ であるベクトル $\hat{\boldsymbol{v}}$ を「ベクトル $\boldsymbol{v}$ を r 循環」したベクトルとする．

3.2 加法準同型暗号

加法準同型暗号とは，暗号化したままで二つの暗号文同士の加算を行うことのできる性質を持つ暗号方式である．本稿ではこのような性質を持つ公開鍵暗号方式の使用を想定する．例としては lifted-Elgamal 暗号 [5] などが知られている．

公開鍵暗号方式は次の 3 つのアルゴリズムを備えている．

（1） KeyGen: 公開鍵と秘密鍵のペア $(\mathsf{pk}, \mathsf{sk})$ を出力する．

（2） $\mathsf{Enc}(m)$: 公開鍵 pk を用いて平文 m を暗号化し，暗号文 $[\![m]\!]$ を出力する．

（3） $\mathsf{Dec}([\![m]\!])$: 秘密鍵 sk を用いて暗号文 $[\![m]\!]$ を復号し，平文 m を出力する．

$[\![m]\!]$ は平文 m の暗号文を表す記法である．また，本稿では，平文ベクトル $\boldsymbol{v}$ の各要素を暗号化したベクトルも $[\![\boldsymbol{v}]\!]$ のように略記する場合がある．

これらのアルゴリズムに加え，加法準同型暗号では暗号文上での演算により，2 つの暗号文の表す平文同士の加算，暗号文の平文の定数倍を計算することが可能である．本稿ではそれぞれに次のような記法を用いる．

- 暗号文同士の加算: $[\![m_1 + m_2]\!] = [\![m_1]\!] \oplus [\![m_2]\!]$
- 暗号文の定数倍: $[\![k \cdot m]\!] = k \otimes [\![m]\!]$

ただし，m, m_1, m_2 は平文 (整数)，k は整数定数である．

演算 $\otimes$ は演算 $\oplus$ を繰り返すことにより実現できる．暗号文の符号反転については $\ominus[\![m]\!]$ と表記する．

本稿では用いる暗号方式が強秘匿性を有することを前提として議論を行う．強秘匿性とは暗号文から平文の情報を何も得られないような性質である．

3.3 紛失通信

紛失通信 (OT) はユーザー・サーバ間で行われる二者間暗号プロトコルの一種である．1-out-of-n OT では，サーバーが持つ n 個の要素からなるベクトル $\boldsymbol{v}$ から，ユーザーが要素のインデックス $t \in \{0, \ldots, n-1\}$ 番目を指定して $\boldsymbol{v}[t]$ を受信することができる．この時，ユーザーが指定した t **はサーバーに知られず，ユーザーはサーバーの他の要素を知ることはない．**

1-out-of-n OT は加法準同型暗号により実装できる．ここで，二つの操作を定義する．

- $\mathsf{PrepQuery}(t)$： t 番目の要素のみが 1 でそれ以外の要素が 0 である長さ n のベクトル $\boldsymbol{q}$(クエリベクトル) を用意し，各要素を暗号化したベクトル $[\![\boldsymbol{q}]\!]$ を返す．
- $\mathsf{InnerProduct}(\boldsymbol{v}, [\![\boldsymbol{q}]\!])$： 長さが n の平文ベクトル $\boldsymbol{v}$ と暗号文ベクトル $[\![\boldsymbol{q}]\!]$ について，内積 $\boldsymbol{v} \cdot \boldsymbol{q}$ の暗号文を計算する．

1-out-of-n OT は上記の操作を用いて次のように実行される．

（1） ユーザーは $\mathsf{PrepQuery}(t)$ を実行し，$[\![\boldsymbol{q}]\!]$ をサーバーに送信する．

（2） サーバーは $\mathsf{InnerProduct}(\boldsymbol{v}, [\![\boldsymbol{q}]\!]) = [\![\boldsymbol{v}[t]]\!]$ を実行し，結果をユーザーに送る．

（3） ユーザーは $\mathsf{Dec}([\![\boldsymbol{v}[t]]\!])$ を計算し，$\boldsymbol{v}[t]$ を得る．

この時，サーバーはユーザーから送られたクエリを復号することはできないので，クエリが何番目を指定しているのかを知ることはできない．一方，0 との乗算により $\boldsymbol{v}[t]$ 以外の要素の情報は失われるため，ユーザーは $\boldsymbol{v}[t]$ 以外の $\boldsymbol{v}$ の要素の情報を得ることはできない．

3.4 LOUDS

LOUDS [9, 15] は木の簡潔データ構造の一つであり，情報理論的下限に近いデータサイズを実現しつつも木の走査が可能な優れたデータ構造である．

本稿では LOUDS 表現の詳細については割愛し，決定木の計算で現れる，親から子への移動に必要な計算に絞って説明する．

LOUDS では木構造を長さ $2N+1$ のビット列 B で表現する．木の各節点は B 上の二つのビットにより 0 と 1 の二通りで表現される．したがって，B 上の位置 p は，木のいずれかの節点を一意に表現する．つまり，木構造上での親から子への移動は B 上での位置の移動に対応する．以上を踏まえ，以下にビット列 B に対する二つの操作を定義する．

定義 1. ビット列に対する操作

- $\mathsf{Rank}_b(B, p)$：ビット列 B の位置 p までに含まれる $b \in \{0, 1\}$ の数を返す．
- $\mathsf{Select}_b(B, i)$：ビット列 B の i 番目の $b \in \{0, 1\}$ の B 上の位置を返す．

親から子への移動は上述した二つの操作により実現することができる．ひとまずここでは B 上のビット'0' で表現された位置について注目する．まず，B 上の位置 p が与えられたとする．この時，$B[p] = 0$ ならば，p に対応する節点の x 番目の子を示す位置 p'（$B[p'] = 0$）は，次の式で計算することができる．

$$p' = \mathsf{Select}_1(B, \mathsf{Rank}_0(B, p) + 1) + x \qquad (1)$$

以降では簡単のため，式 (1) の第一項を $\mathsf{SelRan}(B, p)$ と記述する．p' は $\mathsf{SelRan}(B, p)$ と子の順序を表す x（以降，オフセットと呼称する）の加算により算出される．ここで注意すべきは，$\mathsf{SelRan}(B, p)$ は全ての $p \in \{i \mid i \in \{0, \ldots, 2N\}, B[i] = 0\}$ に対して事前計算可能だという点である．提案手法では，$\mathsf{SelRan}(B, p)$ の事前計算結果をベクトル $\boldsymbol{v}$ に保存し（以降では SelRan ベクトルと呼称する），1-out-of-n OT により必要な情報を取得する．式 (1) を計算するためには $\boldsymbol{v}$ から取得した値に x を加算する必要があるが，この操作の実現には 4.4 節で説明するオフセットベクトルと再帰的紛失通信を用いる．

ところで，ビット列 B にはビット'1' も含まれており，$B[p] = 1$ の場合についても適切な事前計算が必要となる．式 (1) により子の位置を計算すると，位置 p に対応する節点が葉である場合，$B[p'] = 1$ となる（p' も到達した葉に対応する位置）．提案手法では決定木の根から葉まで辿るために，親から子へ移動する操作を必ず木の高さ d 回再帰的に行う (詳細は 4.4 節)．よって，途中で葉に到達し $B[p] = 1$ である場合にも式 (1) に相当する計算が必要となる．そこで便宜上，$B[p] = 1$ となる位置 p に対しては $\boldsymbol{v}[p] = p$ とする．

上述のように、節点は二種類のビットにより表現されるが、以降では、$B[p] = 0$ となるビットを 0 ベース表現、$B[p] = 1$ となるビットを 1 ベース表現と呼称する．

3.5 決定木

決定木は機械学習モデルの一種である．本稿で扱う決定木を以下のように定める．

- 決定木の木構造は**多分木**かつ**順序木**であるとする．
- 決定木の各内部節点にはそれぞれ入力に対しどの枝を選択するかを返す**分岐関数**が対応付けられている．
- n 次元の**入力ベクトル** $\boldsymbol{x}$ が与えられると，各内部節点に対応付けられた分岐関数に従い根から葉までを辿り到達した葉 l に対応する値 z_l を返す．ただし，$\boldsymbol{x} \in \mathbb{Z}^n, z_l \in \mathbb{Z}$.

$\langle P(x) \rangle$ を論理式 $P(x)$ が成り立つとき 1，それ以外の時に 0 を表す記法とする．各内部節点 j の分岐関数 $\mathsf{Choose}(\mathcal{X}; t_0, \ldots, t_u)$ は，入力値 $\mathcal{X}$，u 個の閾値 $t_k (t_k \in \mathbb{Z})$ について $\sum(\langle t_k \leqq \mathcal{X} \rangle) + 1$ を返す関数である．入力値，閾値のビット長を ℓ とする．入力値，閾値については固定小数点表現を用いることにより実数値に拡張することができる．この場合，ℓ は固定小数点表現の精度とみなすことができる．

3.6 SelRan ベクトルによる決定木の表現

決定木の木構造は SelRan ベクトルにより表現することができる．提案手法では，決定木の表現に SelRan ベクトルと追加の 3 つの補助データ構造を用いる．以降では決定木を表現する上で必要な 3 つの補助データ構造 (1) **閾値行列** Θ，(2) **節点情報ベクトル** $\boldsymbol{L}$，(3) **出力値ベクトル** $\boldsymbol{z}$ について順に説明する．

（1） **閾値行列** Θ： $B[p] = 0$ となる p 番目の列ベクトルのみに閾値のベクトルを格納する $t_{max} \times (2N+1)$ の行列．(t_{max} は閾値の数の上限)

（2） **節点情報ベクトル** $\boldsymbol{L}$： LOUDS の各位置に，位置の種類を格納した長さ $2N+1$ のベクトル．位置の種類とは LOUDS 上の位置が 1 ベース表現，葉の 0 ベース表現，内部節点の 0

ベース表現のいずれに対応するものであるかを区別するものである．3 つの種類には次のように識別子を振ることとする．(i)1 ベース表現 : ONEBASE (ii) 葉の 0 ベース表現 : LEAF (iii) 内部節点の 0 ベース表現 : INTERNAL.

（3） **出力値ベクトル $\boldsymbol{z}$**： 各葉に対応する出力値を葉に対応する 1 ベース表現の位置に格納した長さ $2N+1$ のベクトル．ただし，それ以外の位置には出力値の取りうる値の範囲内のランダムな値を格納する．

また，$\Theta[i][j]$ は LOUDS 上で i 番地の節点の j 番目の閾値である．これら決定木を表現するデータ構造を T と総称する．また，T に入力ベクトル $\boldsymbol{x}$ を与え出力値 z_l を返すことを $T(\boldsymbol{x}) = z_l$ と表記する．

3.7 セキュリティモデル

本稿ではセキュリティモデルとして semi-honest モデルを想定する．semi-honest モデルではプロトコルに参加する二者がどちらもプロトコルの仕様から逸脱しない．しかし，プロトコルの仕様上得られる出力や中間結果などから相手の秘密情報を得ようとする．

一方，semi-honest モデルより攻撃者の自由度が高いモデルとして malicious モデルがある．malicious モデルでは攻撃者はプロトコルの仕様から自由に逸脱し相手の秘密情報を得ようとする．malicious モデルへの対応については本稿では詳細を説明しないが，ゼロ知識証明と呼ばれる暗号技術を用いて実現することが可能である．

4. 手　　法

4.1 問題設定

提案手法で想定する問題を次のように定義する．

ユーザーは秘密情報である入力ベクトル $\boldsymbol{x}$ を持ち，サーバーは秘密情報として決定木 T を保持する．ユーザーとサーバーは二者間の秘匿決定木評価プロトコルに参加し，最終的な結果としてユーザーは入力 $\boldsymbol{x}$ に対する決定木の出力 $T(\boldsymbol{x})$ のみを得，サーバーは何も得ない．さらに，プロトコル中にユーザー，サーバーは他方の秘密情報に関する情報を何も得ない．ただし，ユーザーとサーバーは (1) SelRan ベクトルの長さ，(2) 決定木の高さ，(3) 入力ベクトルの長さは予め共有するものとする．

4.2 提案手法の概観

この節では提案手法の概観を示す．

提案手法では，まず，ユーザーが KeyGen により鍵生成を行い鍵ペア $(\mathsf{sk}, \mathsf{pk})$ を得る．ユーザーは公開鍵 pk のみをサーバーに送信する．ここで，ユーザーは sk を持つため暗号文を復号することが出来るが，サーバーは pk のみを持つため復号することができない．したがって，サーバーが暗号文から平文の情報を得られないことは公開鍵暗号の強秘匿性により保証される．

鍵交換後は大きく 2 つのフェーズに分かれる．時系列順に比較フェーズ，評価フェーズである．以下にそれぞれのフェーズで行われる操作の概要を示す．また，図 2 に模式図を示す．

a） **比較フェーズ**

比較フェーズではユーザーの入力ベクトル $\boldsymbol{x}$ から各節点に紐づけられた分岐関数の出力を秘匿計算し，各節点でどの枝が

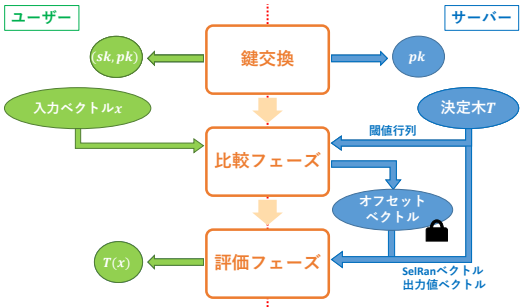

図 2: 提案手法のプロトコル概観

選択されるかを全て暗号文としてサーバーが保持する．分岐関数の出力の計算には後述する比較プロトコルを用いる．何番目の枝を選択するのかは LOUDS 上での位置オフセット，すなわち，式 1 の x に対応する．最終的にこのフェーズでは，対応する LOUDS 上の位置にオフセットの値を格納したベクトルであるオフセットベクトルをサーバーが構築する．

b） **評価フェーズ**

評価フェーズではサーバーの持つ SelRan ベクトルと比較フェーズで構築したオフセットベクトルを繰り返し参照し，LOUDS 上を根から葉まで順次辿る．葉までたどり着くと，到達した節点に対応する値を出力値ベクトルから抽出する．

比較フェーズ，評価フェーズは実際には二者間の暗号プロトコルを用いた秘匿計算により行われる．まず，このとき用いられる暗号プロトコルについて 4.3 節で述べたのちに，4.4 節にて比較フェーズ，評価フェーズの詳細について述べる．

4.3 構成技術

4.3.1 DGK 比較プロトコル

比較フェーズで分岐関数の出力を秘匿計算するためには，$\langle x < y \rangle$ を秘匿計算する暗号プロトコルが必要である．加法準同型暗号を用いる方式でこれを実現する二者間の秘匿計算プロトコルとして，DGK 比較プロトコル [6, 13] が知られている．二者間の比較プロトコルにはいくつかの問題設定が考えられるが，提案手法ではユーザーとサーバーが平文の入力値 x，y を持ち，プロトコルの終了時にサーバーのみが暗号化された結果 $[\![\langle x < y \rangle]\!]$ を得る問題設定を想定する．比較プロトコルについては，本稿ではブラックボックスとして扱う．なお，実装には [13] の手法を用いた．[13] の手法を用いる場合，x，y が ℓ ビット整数であるとき，一回の比較における計算量，通信量のオーダーはサーバー・ユーザーともに $O(\ell)$ となる．

4.3.2 再帰的紛失通信

評価フェーズではユーザーの指定したクエリに基づいてサーバーの持つベクトルの要素を参照し，その値から次のクエリを生成する，という操作を繰り返す必要がある．この再帰的な繰り返しをお互いの秘密情報を明かさずに行うためには，クエリの隠ぺいとともに**中間結果 (サーバーから送信する値) も隠ぺいする必要がある**．この要件を満たすために，**再帰的紛失通信** (ROT) [12] と呼ばれる技術を導入する．

ROT ではサーバーが長さ M の平文ベクトル $\boldsymbol{v}$ を持ち，ユーザーがクエリ p_0 を指定したとき，何回かの繰り返しの後ユー

ザーが $\boldsymbol{v}[\boldsymbol{v}[\ldots\boldsymbol{v}[p_0]\ldots]]$ を得，サーバーは何も情報を得ない．

ROT の概要は次のとおりである．ROT ではサーバーが OT の結果をランダム化することにより，ユーザーに対し OT の情報を隠ぺいする．ユーザーはランダム化された値を元に次の回の OT のクエリベクトルを生成する．このとき生成されたクエリベクトルはランダム化されているためそのまま用いることは出来ない．しかし，サーバーは乱数を知っているのでクエリベクトルからランダム化される前の「真の」クエリベクトルを復元することができる．サーバーは「真の」クエリに基づき OT の操作を実行し，別の乱数でランダム化する．これを条件を満たすまで繰り返す．

具体的には ROT は次の 2 つの操作に PrepQuery を加えた 3 つの操作により構成される．

- $\mathsf{RanOT}(\boldsymbol{v}, [\![\boldsymbol{q}]\!], r)$: それぞれ長さ M の平文ベクトル $\boldsymbol{v}$ と暗号文ベクトル $[\![\boldsymbol{q}]\!]$ 及び乱数 r を入力とする．$\boldsymbol{q}[t] = 1$ であるとする．ランダム化された結果 $[\![(\boldsymbol{v}[t] + r)_{\mathsf{mod}\ M}]\!]$ を返す．
- $\mathsf{RecQuery}([\![\hat{\boldsymbol{q}}]\!], r)$: 長さ M の暗号文ベクトル $[\![\hat{\boldsymbol{q}}]\!]$，及び乱数 r を入力とする．ただし，$[\![\hat{\boldsymbol{q}}]\!]$ は乱数 r によってランダム化されたクエリから生成されたクエリベクトルとする．$[\![\hat{\boldsymbol{q}}]\!]$ のランダム化を解いた $[\![\boldsymbol{q}]\!]$ を返す．

RanOT 操作では長さ M の平文ベクトルと暗号文ベクトルの内積が計算される．RecQuery 操作は $[\![\hat{\boldsymbol{q}}]\!]$ を r 循環させることにより実現される．なぜなら，クエリ (整数) の加算はクエリベクトルの 1 が立つ位置の移動に対応するからである．

ROT のサーバー側の計算量で主要となるのは暗号文ベクトルの内積操作である．したがって，サーバー側の一回あたりの計算量は $O(M)$ である．一方，ユーザー側の計算量はクエリベクトルの生成が主であり，一回あたり $O(M)$ の計算量となる．また，一回あたりの通信量も同じく $O(M)$ である．

4.3.3 拡張再帰的紛失通信

ROT はサーバーが平文でベクトルを保持していなければならないという制約があった．しかし，評価フェーズで参照する必要があるオフセットベクトルは暗号化されている必要がある．ユーザーが選択する枝の情報を保護するためである．

この節では ROT を拡張し，特定の状況下においてサーバーが**暗号化された状態で情報を保持しているときに適用可能な手法**を提案する．本稿ではこの手法を**拡張再帰的紛失通信 (eROT)** と呼称する．eROT ではサーバーは $M \times N$ の暗号文行列 $\boldsymbol{V}$ を持つ．$\boldsymbol{V}$ の各行 $\boldsymbol{V}[i]$ は N ビットの単進符号の各ビットを暗号化したベクトルである．$\boldsymbol{V}[i]$ は平文のベクトル $\boldsymbol{v}$ の要素 $\boldsymbol{v}[i]$ の値を表すとする．ユーザーがクエリ p_0 を指定したとき，何回かのラウンドの繰り返しの後ユーザーが $\boldsymbol{v}[\boldsymbol{v}[\ldots\boldsymbol{v}[p_0]\ldots]]$ を得，サーバーは何も情報を得ない．

eROT も ROT 同様ランダム化により情報を隠ぺいする．しかし，加法準同型暗号上で任意の法の mod 演算を行うことは困難であり，$\oplus$ 演算のみで乱数の加算を安全に行うことができない．eROT ではこの問題を，乱数の加算及び mod 演算が，単進符号を表現したベクトルの循環に対応することを利用して解決する．

eROT では新たに 2 つの操作を定義する．

（1） $\mathsf{RandRot}(\boldsymbol{V}, [\![\boldsymbol{q}]\!], r', r)$: $\boldsymbol{V}$ を $M \times N$ の暗号文の行列，r'，r を乱数とする．$\boldsymbol{q}$ は $\boldsymbol{q}[p] = 1$ それ以外の要素が 0 であるような長さ M のベクトルである．各要素 $\boldsymbol{u}[i]$ が $\boldsymbol{u}[i] = \bigoplus_{j=0}^{N-1}(\boldsymbol{V}[i][j] \otimes (j + r)_{\mathsf{mod}\ M})$ のように定義されたベクトル $\boldsymbol{u}$ を構築する．$\boldsymbol{u}$ を r' 循環させたベクトルを $\hat{\boldsymbol{u}}$ とする．$\hat{\boldsymbol{u}}'$ を各要素が $\hat{\boldsymbol{u}}'[i] = \hat{\boldsymbol{u}} \oplus [\![r_{2,i} \otimes ([\![1]\!] \oplus (\ominus \boldsymbol{q}[i]))]\!]$ であるベクトル $\hat{\boldsymbol{u}}'$ を返す．$r_{2,i}$ はそれぞれ乱数である．このとき p 番地には $[\![0]\!]$ が足される．

（2） $\mathsf{Extract}(\hat{\boldsymbol{u}}, p)$: 暗号ベクトル $\hat{\boldsymbol{u}}$ から $\hat{\boldsymbol{u}}[p]$ を選び，復号した値 $\mathsf{Dec}(\hat{\boldsymbol{u}})$ を返す．この値は $(\boldsymbol{v}[p] + r)_{\mathsf{mod}\ M}$ である．

eROT は PrepQuery, RecQuery, RandRot, Extract の 4 つの操作からなる．eROT はこれら 4 つの操作を以下のように繰り返し実行する．

（1） ユーザーが $\mathsf{PrepQuery}(p_i)$ を実行し，クエリベクトル $[\![\hat{\boldsymbol{q}}]\!]$ をサーバーに送信する．

（2） サーバーは前回セットした乱数 r_i を用いて $\mathsf{RecQuery}([\![\hat{\boldsymbol{q}}]\!], r_i)$ を実行し「真の」クエリベクトル $[\![\boldsymbol{q}]\!]$ を得る．

（3） サーバーは新たな乱数 r_{i+1} をセットして $\mathsf{RandRot}(\boldsymbol{V}, [\![\boldsymbol{q}]\!], r_{i+1})$ を実行し，$\hat{\boldsymbol{u}}$ を全てユーザーに送信する．

（4） ユーザーは $\mathsf{Extract}(\hat{\boldsymbol{u}}, p)$ を実行し，次のクエリ p_{i+1} を $(\boldsymbol{v}[p] + r_{i+1})_{\mathsf{mod}\ M}$ としたうえで (1) に戻る．

eROT におけるサーバー側の計算量の主要な部分は ROT と同様に内積演算である．したがって，サーバー側の一回あたりの計算量は $O(MN)$ となる．ユーザー側の計算量はクエリ生成が主要であり，ROT の場合と変わらず $O(M)$ である．また，一回あたりの通信量はクエリベクトルと $\hat{\boldsymbol{v}}$ がともに長さ M であるため $O(M)$ である．

4.4 LOUDS と準同型暗号による秘匿決定木評価

4.4.1 比較フェーズ

4.2 節で述べた通り，比較フェーズではユーザーの入力ベクトル $\boldsymbol{x}$ から各節点に関連付けられた分岐関数の出力を計算し，それらを元に**オフセットベクトルを構築する**．ただし，オフセットベクトルを eROT により参照するため，実際には単進符号の暗号文行列とする必要がある．以降ではオフセットベクトルを改めて**オフセット行列**と呼ぶことにする．比較フェーズの詳細は Algorithm 1 に示した．

フラグ行列 $\boldsymbol{F}$ の構築： LOUDS 上の j 番地に対応する節点の分岐関数 $\mathsf{Choose}_j(\mathcal{X}_j; \Theta[j][0], \ldots, \Theta[j][u_j - 1])$ の出力は $\sum_{k=0}^{u_j-1}(\Theta[j][k] \leqq \mathcal{X}_j)_1$ により計算できる．ただし，$\mathcal{X}_j$ は j 番地に対応する入力ベクトルの要素，u_j は j 番地に対応する閾値の個数である．まず，Step (1) でサーバーとユーザーは入力値と閾値の比較結果を全て計算し，結果の暗号文ベクトルを構築する．閾値と入力値との比較結果を秘匿計算するために 4.3.1 節で導入した DGK 比較プロトコルを用いる．比較結果の暗号文行列 $\boldsymbol{F} = [\![\langle \Theta[j][k] \leqq \mathcal{X}_j \rangle]\!]$ を**フラグ行列**と呼ぶ．ただし，$k = 0, \ldots, u_j - 1$，$j = 0, \ldots, 2N + 1$ である．

$\boldsymbol{W}$ の構築： 次に，サーバーは Step (2) でフラグ行列の各行を，分岐関数の出力を単進符号の形式に変換して各ビットを暗号化した行列 $\boldsymbol{W}$ を構築する．フラグ行列 $\boldsymbol{F}$ が与えられたとき，$\mathsf{Choose}_j(\mathcal{X}_j; \Theta[j][0], \ldots, \Theta[j][u_j - 1]) - 1$ を表す $u_j + 1$

Algorithm 1 提案手法の比較フェーズの詳細

- 共通の入力: LOUDS 表現の長さ $2N+1$，決定木の高さ d，入力ベクトルの長さ n，比較回数 m
- サーバーの入力: 閾値ベクトル Θ，節点情報ベクトル $\boldsymbol{L}$
- ユーザーの入力: 入力ベクトル $\boldsymbol{x}$

Step (1): サーバー，ユーザーが DGK 比較プロトコルを実行し，フラグベクトル $\boldsymbol{F}$ を構築する．

Step (2): サーバーはフラグ行列 $\boldsymbol{F}$ から $\boldsymbol{W}$ を構築する．

for $j \in S, k \in \{0,\dots,u_j-1\}$ **do**
  $\boldsymbol{W}[j][k] \leftarrow \boldsymbol{F}[j][k-1] \oplus (\ominus \boldsymbol{F}[j][k])$

Step (3): サーバーは $\boldsymbol{W}$ を元にオフセット行列 $\boldsymbol{O}$ を構築する．

for $j \in \{0,\dots,2N+1\}$ **do**
  if $\boldsymbol{L}[j] = \mathsf{ONEBASE}$ **then**
    $\boldsymbol{O}[j] \leftarrow ([\![1]\!])$
  else if $\boldsymbol{L}[j] = \mathsf{LEAF}$ **then**
    $\boldsymbol{O}[j] \leftarrow ([\![1]\!])$
  else if $\boldsymbol{L}[j] = \mathsf{INTERNAL}$ **then**
    $\boldsymbol{O}[j] \leftarrow (\boldsymbol{W}[j], [\![0]\!])$

$(\boldsymbol{W}[j], [\![0]\!])$ は $\boldsymbol{W}[j]$ に $[\![0]\!]$ を連結したベクトルを意味する．

ビットの単進符号表現の暗号文ベクトル $\boldsymbol{W}[j]$ は以下のように各要素を計算することで構築することができる．

$$\boldsymbol{W}[j][k] = \boldsymbol{F}[j][k-1] \oplus (\ominus \boldsymbol{F}[j][k])$$

ただし，$\boldsymbol{F}[j][u_j] = [\![0]\!]$，$\boldsymbol{F}[j][-1] = [\![1]\!]$ とする．

オフセット行列 $\boldsymbol{O}$ の構築： 最後に，サーバーは Step (3) で $\boldsymbol{W}$ 及び節点情報ベクトル $\boldsymbol{L}$ からオフセット行列 $\boldsymbol{O}$ を構築する．**$\boldsymbol{O}$ は各行が LOUDS 上のオフセットを単進符号で表した暗号ベクトルであるような行列**である．$\boldsymbol{O}$ の各行 $\boldsymbol{O}[j]$ に格納すべき値はまず，位置 j が 1 ベース表現に対応するか否かで異なる．1 ベース表現の場合はオフセットは 0 とする．これは 1 ベース表現にたどり着いた後にその位置にとどまり続けるためである．0 ベース表現の場合はさらに 2 つの場合に分かれる．(1) 節点が葉の場合は 1 ベース表現にとどめるためオフセットを 0 とする．(2) 内部節点の場合分岐関数の出力 $\mathsf{Choose}_j(\mathcal{X}_j; \Theta[j][0], \dots, \Theta[j][u_j-1])$ をオフセットとする．以上より，$\boldsymbol{O}[j]$ は Step(3) のように決定される．

Algorithm 1 のように $\boldsymbol{O}$ を構築することで比較回数以外のサーバー，ユーザーの情報は明かされない．

また，$\boldsymbol{O}$ は各行の長さが異なるが，**eROT により参照できる形式となっている．**RandRot の内積操作においてビット列の長さに応じて $\bigoplus_{j=0}^{u_j}(\boldsymbol{V}[i][j] \cdot (j+r)_{\mathsf{mod}\ M})$ と計算すればよいからである．さらに，これにより計算量を削減することができる．

4.4.2 評価フェーズ

この節では 4.2 節に述べた LOUDS 上の木の操作を秘匿計算技術を用いてどのように実現するかを説明する．LOUDS 上の位置 p_i を与えられたとき，次の位置 p_{i+1} を算出するためには SelRan ベクトル $\boldsymbol{v}$ とオフセット行列 $\boldsymbol{O}$ の p_i 番目の要素の参照を行い，前者に後者を足した値を計算する．根から葉まで辿る操作を行うためには初期位置 p_0 を 0 と指定して繰り返し $\boldsymbol{v}$ 及び $\boldsymbol{O}$ を参照する．ただし，サーバーはユーザーがどの位置を指定したかを知らず，ユーザーはサーバーの持つ $\boldsymbol{v}$ 及び $\boldsymbol{O}$ の内容を知らないようにしなければならない．サーバー，ユーザーがお互いの秘密情報を得ずに 2 つのベクトルの参照を繰り返し行うために，4.3.2，4.3.3 節で導入した ROT 及び eROT を用いる．

評価フェーズの疑似コードを Algorithm 2 に示した．最初にサーバー，ユーザーともに Step (1) で初期化を行う．サーバーは乱数 $r'_1 = r'_2 = 0$ を設定する．r'_i は一つ前の繰り返しで設定した乱数を記録する変数である．ユーザーは初期位置 $p_0 = 0$ を設定して評価フェーズを始める．

Step (2) では $\boldsymbol{v}$ と $\boldsymbol{O}$ に対しそれぞれ ROT と eROT を繰り返し適用し，LOUDS 上の位置を更新する．各回の ROT,eROT により得られる β'_{k+1} 及び ω'_{k+1} はそれぞれ $\mathsf{SelRan}(B, p'_k)$ とオフセットの値がランダム化された値である．また，LOUDS 上の子の位置は $\mathsf{SelRan}(B,p)+x$ によって定まる．よって，次のクエリ $p'_{k+1} = (\beta'_{k+1} + \omega'_{k+1})_{\mathsf{mod}\ 2N+1}$ となる．

この繰り返しは木の高さと同じ d 回行われる．ここで，葉に到達した場合の動作について注意が必要である．$\boldsymbol{O}$ の定義から，葉に到達した場合 LOUDS 上の位置は到達した葉の位置 (1 ベース表現) に固定される．これにより葉の深さにかかわらず d 回の繰り返しの後にはいずれかの葉の 1 ベース表現に到達することが保証される．

Step (3) では OT により出力値ベクトルを参照し，ユーザーが決定木の出力 $T(\boldsymbol{x}) = z_l$ を得る．

評価フェーズでは ROT と eROT を組合わせることにより，サーバー，ユーザーの秘密情報を隠したまま LOUDS 上で木を辿る操作を実現した．評価フェーズで共有する情報は $\boldsymbol{v}$ の長さ $2N+1$ のみである．

4.5 計 算 量

本節では提案手法の時間計算量・通信量について考察する．

まず，比較フェーズについて考察する．DGK 比較プロトコルに要する計算量は精度 ℓ と比較回数 $m(<N)$ に比例し，サーバー側で $O(\ell m)$ である．ユーザー側では入力ベクトルの暗号化と比較フェーズにおける復号化を考慮すると $O(\ell(n+m))$ の計算量となる．また，フラグ行列からオフセット行列を構築するのに必要な計算量はオフセット行列の行ベクトルの総長に比例し，$O(N)$ である．したがって，比較フェーズのサーバーの計算量は $O(\ell m + N)$，ユーザーの計算量は $O(\ell(n+m))$ となる．また，通信量については m 回の DGK 比較プロトコルに要する通信量が ℓm であることから，サーバー側で $O(\ell m)$ である．ユーザー側の通信量は入力ベクトルの長さが n であり，暗号化された結果を m 個送信するため $O(\ell n + m)$ である．

評価フェーズの計算量はサーバー側，ユーザー側ともに $O(dN)$ である．ROT の計算量は 1 回当たり $O(N)$ であることから $O(dN)$ である．また，eROT の計算量も $O(dN)$ である．eROT においてサーバー側の計算量は行列の大きさに比例するが，オフセット行列の各行の長さの総和は $O(N)$ であるため，eROT1 回当たりの計算量は $O(N)$ となるからである．通信量については長さ $2N+1$ の暗号文ベクトルを送信するため，$O(dN)$ である．以上の計算量及び通信量を表 1 にまとめた．

Algorithm 2 提案手法の比較フェーズの詳細

- 共通の入力: LOUDS 表現の長さ $2N+1$，決定木の高さ d
- サーバーの入力: SelRan ベクトル $\boldsymbol{v}$，オフセット行列 $\boldsymbol{O}$

Step (1): パラメータを初期化する．

ServerSide:

$r'_1 \leftarrow 0, r'_2 \leftarrow 0$

ClientSide:

$p'_0 \leftarrow 0$

Step (2): ROT，eROT を繰り返し LOUDS 上の位置を更新する．

for $k = 0$ **to** $d-1$ **do**

ClientSide:

$\hat{\boldsymbol{q}} \leftarrow \mathsf{PrepQuery}(p'_k)$

$\hat{\boldsymbol{q}}$ をサーバーに送信する．

ServerSide:

$\hat{\boldsymbol{q}}$ を受信する．

$\boldsymbol{q} \leftarrow \mathsf{RecQuery}(\hat{\boldsymbol{q}}, r'_1 - r'_2)$

r_1, r_2 に乱数を設定する．

ROT:

$[\![\beta'_{k+1}]\!] \leftarrow \mathsf{RanOT}(\boldsymbol{v}, [\![\boldsymbol{q}]\!], r_1)$

eROT:

$\hat{\boldsymbol{o}}_{k+1} \leftarrow \mathsf{RandRot}(\boldsymbol{O}, [\![\boldsymbol{q}]\!], r'_1 - r'_2, r_2)$

$[\![\beta'_{k+1}]\!]$ と $\hat{\boldsymbol{o}}_{k+1}$ をユーザーに送信する．

$r'_1 \leftarrow r_1, r'_2 \leftarrow r_2$

ClientSide:

$[\![\beta'_{k+1}]\!]$ と $\hat{\boldsymbol{o}}_{k+1}$ を受信する．

$\beta'_{k+1} \leftarrow [\![\beta'_{k+1}]\!]$

$\omega'_{k+1} \leftarrow \mathsf{Extract}(\hat{\boldsymbol{o}}_{k+1}, p'_k)$

$p'_{k+1} \leftarrow (\beta'_{k+1} + \omega'_{k+1}) \bmod 2N+1$

Step (3): OT により出力ベクトル $\boldsymbol{z}$ から決定木の出力 z_ℓ を得る．

ClientSide:

$[\![\hat{\boldsymbol{q}}]\!] \leftarrow \mathsf{PrepQuery}(p'_d)$

$[\![\hat{\boldsymbol{q}}]\!]$ をサーバーに送信する．

ServerSide:

$[\![\hat{\boldsymbol{q}}]\!]$ を受信する．

$[\![\boldsymbol{q}]\!] \leftarrow \mathsf{RecQuery}(\hat{\boldsymbol{q}}, r'_1 - r'_2)$

$[\![z_\ell]\!] \leftarrow \mathsf{InnerProduct}(\boldsymbol{z}, [\![\boldsymbol{q}]\!])$

$[\![z_\ell]\!]$ をユーザーに送信する．

ClientSide:

$[\![z_\ell]\!]$ を受信する．

$z_\ell \leftarrow \mathsf{Dec}[\![z_\ell]\!]$

表 1: 提案手法の各フェーズと全体の時間計算量と通信量．表中の S はサーバー，U はクライアントの略である．

	時間計算量	通信量
比較フェーズ (S)	$O(\ell m + N)$	$O(\ell m)$
比較フェーズ (U)	$O(\ell(n+m))$	$O(\ell n)$
評価フェーズ (S)	$O(dN)$	$O(dN)$
評価フェーズ (U)	$O(dN)$	$O(dN)$

5. 実験結果

4.4 節に説明した semi-honest モデルに対応した提案手法を実装した．実装に用いた言語は C++である．加法準同型暗号方式としては lifted-Elgamal を採用し，その C++実装である mcl ライブラリ [11] を用いた．lifted-Elgamal のセキュリティパラメータには secp192k1 [7] を用いた．

テストデータはランダムに生成した決定木と入力ベクトルである．高さ d を 10 から 20 まで 1 刻みで変化させ，節点数 N が 500 である決定木をサーバーの持つ決定木とした．比較回数 m は 180 程度となった．分岐数は 2 か 3 であるよう設定した．入力ベクトルの長さ n は比較回数 m と同じとした．

実験にはユーザー用，サーバー用にそれぞれデスクトップコンピューター台ずつを用いた (それぞれ Xeon 2.40GHz，Xeon 3.40GHz を搭載)．また，どちらも 1 スレッドのみ使用した．

図 3 は計算時間のグラフである．図 3 から，計算時間の合計は高さ 20 においても 11 秒程度と実用的な時間であることが確認できた．また，サーバー，ユーザーともに高さに比例する計算時間となった．

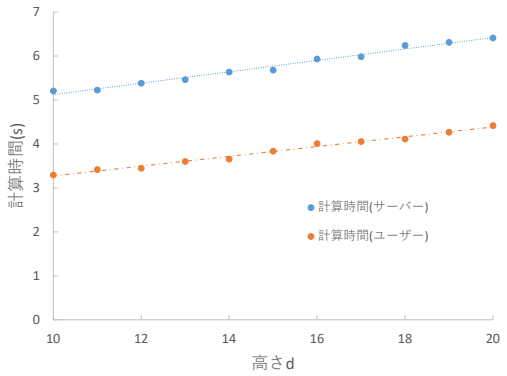

図 3: 提案手法におけるユーザー，サーバーそれぞれの計算時間．

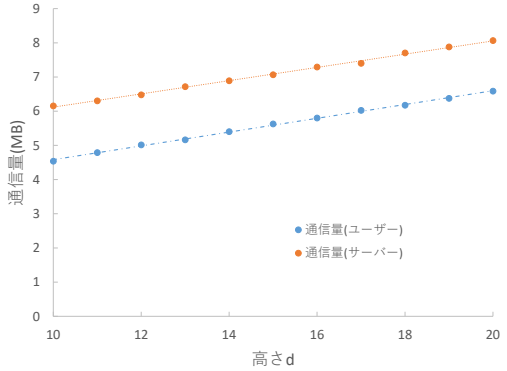

図 4: 提案手法におけるユーザー，サーバーそれぞれの総通信量 (アップロード)．

図 4 は通信量のグラフである．図 4 から，総通信量は高さ 20 においてサーバーの上りで 8MB，ユーザーの上りで 7MB 程度となり，現実的な値と言える．また，通信量についてもサーバー，ユーザーともに高さに対しておおよそ線形であり，表 1 に示した理論性能に従うことが確かめられた．

6. 従来手法との比較

本節では semi-honest モデルに対して安全なプロトコルについて従来手法 [14] との比較を行う．まず，計算量及び通信量のオーダーの比較について考察する．表 2 に従来手法と提案手法の計算量及び通信量のオーダーを示した．[14] の手法ではサーバー側の計算量と通信量が指数オーダーとなる．提案手法はサーバー側の計算量・通信量を高さ d に対して線形のオーダーに改善した．一方で，ユーザー側の計算量及び通信量のオーダーは従来手法よりも悪化した．しかし，提案手法のユーザー側の評価フェーズの計算時間・通信量が全体に占める割合は小さい．よって，プロトコル全体としてはユーザー側のオーダー悪化の影響は小さいと思われる．以上から，提案手法は [14] と比較して決定木の高さが高い場合にも対応することができる汎用性を持ち，特に木がスパースな場合に優れた性能となると言える．木の高さが低い場合では提案手法の実測値も十分に小さく，[14] との差は小さいことが予想される．

次に，実行時間の実測値について比較を行う．ただし，従来手法 [14] は実装が非公開であるため， [14] に記載の実験結果と，二分決定木を用いパラメータを同一にした場合の提案手法の実験結果との比較を行う．$d = 17, m = 57$ の場合，提案手法，従来手法の実行時間はそれぞれ 5.7 秒，17 秒であり，11 秒程度，約 3 倍の高速化を達成した．また，$d = 4, m = 13$ の場合，それぞれ 0.52 秒，0.37 秒と従来手法の方が上回る結果となった．しかし，差は 0.15 秒程度と小さい．また，実験結果は計算量及び通信量のオーダーの比較において述べた性質とよく一致すると言える．なお，上記の提案手法の実験結果は CPU の周波数について補正した値である．暗号ライブラリやセキュリティパラメータなどの差異は補正していない．

	時間計算量	通信量
従来手法 (S)	$O(\ell m + 2^d)$	$O(\ell m + 2^d)$
従来手法 (U)	$O(\ell(n+m)+d)$	$O(\ell n + m)$
提案手法 (S)	$O(\ell m + dN)$	$O(\ell m + dN)$
提案手法 (U)	$O(\ell(n+m)+dN)$	$O(\ell n + m + dN)$

表 2: 従来手法 [14] と提案手法それぞれの時間計算量と通信量のオーダー．表中の S はサーバー，C はユーザーの略である．

7. 結　　論

本研究ではサーバー，クライアント双方の秘密情報を秘匿したまま決定木の出力を効率的に計算する二者間の秘匿決定木評価手法を提案した．提案手法は高さに線形な計算量・通信量を達成した．従来手法の高さに対して指数オーダーの計算量・通信量に対して大幅な改善となった．また，提案手法は malicious モデルに対して安全なプロトコルに拡張することができる．提案手法は従来手法と同等のセキュリティを維持しつつ決定木の高さが高い場合にも対応できる汎用性を期待できる．さらに，提案手法は木構造を自由に走査することが可能なため，決定木のみに特化して設計された従来手法とは異なり，木による検索など多様な応用も期待される．

文　　献

[1] Mauro Barni, Pierluigi Failla, Vladimir Kolesnikov, Riccardo Lazzeretti, Ahmad Reza Sadeghi, and Thomas Schneider. Secure evaluation of private linear branching programs with medical applications. In *ESORICS*, pp. 424–439, 2009.

[2] BigML Website. Machine Learning for everyone. *BigML.com*, 2016. URL accessed January 30, 2017.

[3] Raphaël Bost, Ra Popa, Stephen Tu, and Shafi Goldwasser. Machine Learning Classification over Encrypted Data. In *NDSS*, pp. 1–14, 2015.

[4] Justin Brickell, Donald E Porter, Vitaly Shmatikov, and Emmett Witchel. Privacy-preserving remote diagnostics. In *CCS*, pp. 498–507, 2007.

[5] Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers. A secure and optimally efficient multi-authority election scheme. In *EUROCRYPT*, pp. 103–118. Springer-Verlag, 1997.

[6] Ivan Damgård, Martin Geisler, and Mikkel Krøigaard. Efficient and Secure Comparison for On-Line Auctions. In *ACISP*, pp. 416–430, 2007.

[7] Daniel R. L. Brown. Standards for Efficient Cryptography 2 (SEC 2) : Recommended Elliptic Curve Domain Parameters. *Standards for Efficient Cryptography*, pp. 1–33, 2010.

[8] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. Model Inversion Attacks that Exploit Confidence Information and Basic Countermeasures. In *SIGSAC*, pp. 1322–1333, 2015.

[9] Guy Jacobson. Space-efficient static trees and graphs. In *FOCS*, pp. 549–554, 1989.

[10] Microsoft Azure. Machine Learning - Microsoft Azure, 2016. URL accessed January 30, 2017.

[11] Mitsunari Shigeo. C++ library implementing elliptic curve elgamal crypto system. URL accessed January 30, 2017.

[12] Kana Shimizu, Koji Nuida, and Gunnar Ratsch. Efficient privacy-preserving string search and an application in genomics. *Bioinformatics*, Vol. 32, No. 11, pp. 1652–1661, 2016.

[13] Thijs Veugen. Improving the DGK comparison protocol. In *WIFS*, pp. 49–54, 2012.

[14] David J Wu, Tony Feng, Michael Naehrig, and Kristin Lauter. Privately Evaluating Decision Trees and Random Forests. *PoPETS*, Vol. 2016, No. 4, pp. 1–21, 2016.

[15] 岡之原大輔. 高速文字列解析の世界. 岩波書店, 2012.