定数ラウンドかつ対数サイズ通信量の秘匿配列アクセスプロトコル

樋渡 啓太郎[1,2,a)]　縫田 光司[1,2]

概要：秘匿配列アクセスは秘匿された配列から秘匿されたアクセス位置に対応する値をとる処理である．文字列解析やデータ分析において，配列アクセスは多用されているため，秘匿配列アルゴリズムの効率化は秘匿文字列解析などの効率的な構成に大きくかかわっており，盛んに研究されている．秘匿配列アクセスはその性質上，単純な方法では配列サイズ N に関して線形の通信，計算コストがかかってしまう．ORAM などの方式では，劣線形の通信量，計算量で秘匿配列アクセスを実現しているが，構成が複雑なため漸近表示に隠れる定数ファクターが大きい，通信ラウンドが多い，といった問題があり，ある程度の大きさの配列サイズでは線形コスト計算量の手法のほうが性能がよいという研究結果も存在する．定数ファクターが小さいような比較的簡単な構成で線形コスト計算量，定数ラウンドを達成する既存手法では，$\tilde{O}(\sqrt{N})$ 通信量が最も少ない通信量であったが，本研究では，秘匿配列アクセスをより簡単な問題に言い換えることにより，線形計算量ではあるものの，$O(\log N)$ 通信量，定数ラウンドを達成するアルゴリズムを構成した．

キーワード：秘密計算，秘匿配列アクセス

A Secure Array Access Protocol with Constant Communication Rounds and Log Size Communication Complexities

Keitaro Hiwatashi[1,2,a)]　Koji Nuida[1,2]

Abstract: Secure array access is a process that takes the value corresponding to the concealed address from the concealed array. Since array access is a major method used in various string or data analyses, the improvement of "secure" array access affects the efficiency of "secure" string or data analyses and is studied actively. It takes $O(N)$ computational/communication complexity to realize secure array access naively, where N is the size of the array. Though there are some techniques such as ORAM which achieve sublinear computation and communication costs, those techniques are very complicated, need a lot of communication rounds, and are inferior to some linear-complexity methods up to a certain size N. Among the methods with relatively simple construction and constant communication rounds, the best communication complexity is $\tilde{O}(\sqrt{N})$. In this paper, we propose a new method that achieves $O(\log N)$ communication complexity.

1. はじめに

秘密計算（Secure Multi-Party Computation，以下 MPC）は第三者への情報漏洩だけでなく，当事者への情報漏洩も防ぎつつ，データの処理を可能にする暗号技術である．特に近年，プライバシーを適切に守りつつ，一方で個人情報を有効に活用したい，という社会的な需要も多く，大きな注目を集めている．MPC を実現する手法としては秘密分散，秘匿回路，加法準同型暗号，完全準同型暗号などがある．本稿では，主に秘密分散に基づいた秘密計算を取り扱う．秘密分散に基づく手法は計算が比較的軽く大規模な秘密計算処理に向いている．また，スケーラビリティの観点からユーザー（クライアント）が情報を暗号化して計算サーバーに送り，計算サーバーが秘密計算処理を行う，と

[1] 東京大学
The University of Tokyo
[2] 産業技術総合研究所
AIST
[a)] keitaro_hiwatashi@mist.i.u-tokyo.ac.jp

いった Client-Server モデルという計算モデルを仮定する研究も多くある [1], [11], [16], [17], [18], [22] が，秘密分散に基づく秘密計算は Client-Server モデルに適している，という利点もある．

MPC は Yao[23] によって提唱されて以降盛んに研究されているが，その中で秘匿配列アクセスと呼ばれる技術が存在する．秘匿配列アクセスは，（秘匿もしくは公開されている）配列と秘匿されたインデックスを入力とし，インデックスに対応する（秘匿された）値を出力とする関数である．配列アクセスは文字列解析やデータ分析に多用されているため，秘匿配列アルゴリズムの効率化は秘匿文字列解析などの効率的な構成に大きくかかわっている．また，複雑な秘密計算処理を配列アクセスに置き換えることで効率化を行う研究 [7] も存在する．

秘匿配列アクセスに関しては，単純な方法だと配列サイズに関して線形の通信量が必要であるが，近年 [25], [26] において，定数ラウンドかつ劣線形通信量の手法が提案されている．しかし，これらの手法の内部では Demux[14] や oblivious AES[5] の計算などの複雑な処理がなされているため，ラウンド数が（定数ではあるものの）比較的大きいなどの問題がある．

1.1 関連研究

配列から値を取ってくる，という形式の機能を実現する上で配列やインデックスが暗号化されているか否か，によってさまざまな問題設定が考えられる．そのようなバリエーションとして，Oblivious Transfer (OT)[20], Private Information Retrieval (PIR)[6], Oblivious RAM (ORAM)[8] といったものが挙げられる．大まかな違いは以下の通りである．本研究で取り扱う秘匿配列アクセスは主に Client-Server モデルの途中において用いられる配列アクセスである．つまり，各パーティは配列の値もインデックスの値もシェアの形式でしか保持していない．OT はインデックスの平文を持っているパーティと配列の平文を持っているパーティがいる状態での配列アクセス，PIR はインデックスの平文を持っているパーティが，公開情報である配列を保持しているサーバーに対して行う配列アクセス，ORAM はクラウドサービスのように自身が外部に保存した配列に対して行う配列アクセスである．これらの問題設定をまとめたものが表 1 である．

プライベートな情報はシェアの特別な形とみなすことができるので，この中では秘匿配列アクセスが最も一般化された形の問題である．秘匿配列アクセスではインデックスの平文を持っている者がいないため，インデックスの平文を持っているパーティがインデックスに応じた複雑な処理を行うことの多い OT や PIR，ORAM の手法をそのまま適用することはできない．実際，OT や PIR，ORAM では，テーブルサイズに関して劣線形な通信コストでシングルクエリを処理する手法が古くから知られている [6], [21], [24] が，秘匿配列アクセスに関しては近年 [25], [26] などで劣線形通信コストのプロトコルが提案されたばかりであり，いまだ線形通信コストの手法も提案されている [2], [7].

表 1 秘匿配列アクセス, OT, PIR, ORAM における問題設定．「シェア」は分散された形式で持っていることを示し，「プライベート」はパーティの一部が（平文で）持っていることを示し，「パブリック」はすべてのパーティが知っている（もしくは知っても問題ない）情報であることを示す．

	インデックス	配列
秘匿配列アクセス	シェア	シェア
OT	プライベート	プライベート
PIR	プライベート	パブリック
ORAM	プライベート	プライベート

1.2 本研究の貢献

本研究では，少ないラウンド数の定数ラウンドかつ劣線形通信量の秘匿配列アクセスプロトコルを二つ構成する．特にそのうちの一つは定数ラウンドと対数サイズ通信量を両立する初の手法である．

二つの提案手法について，一つ目の手法は，[24], [25] と似た方針で $O(\sqrt{N})$ 通信量のプロトコルであり，もう一つは関数秘密分散 [3], [4] と呼ばれる技術を用いた $O(\log N)$ 通信量のプロトコルである．提案手法の特徴として，以下の点があげられる：

- 二つの手法はともに定数ラウンドであり，その定数も 1 ラウンド，2 ラウンドと，他の定数ラウンドの既存研究 [25], [26] と比較しても十分小さい．
- 定数ラウンドと対数サイズ通信量を両立する初の手法である．

また，これらに対して実装評価も行っており，配列サイズ $N = 2^{20}$ に対して，13.4 ミリ秒と，[26] における概算値 20.3 ミリ秒よりも高速に動作していることを確認した．

2. 準備

この章では，本稿で用いる記法と提案手法の構成に関わる基本事項について簡単にまとめる．まず，2.1 節で本稿で用いる記法に関して述べる．2.2 節では秘密計算の安全性を定式化し，2.3 節，2.4 節では，秘密分散，関数秘密分散について述べる．最後に 2.5 節で今回取り扱う秘匿配列アクセスの問題の定式化を行う．

2.1 記法

$x \overset{\mathrm{R}}{\in} A$ で x を集合 A から一様ランダムに選ぶことを意味する．$+_M$ で $\mathbb{Z}_M$ 上の和であることを表し，$x =_M y$ で $x \equiv y \bmod M$ であることを表す．本稿で扱う秘匿配列アクセスにおける配列は，長さ N であり，各配列要素は $\mathbb{Z}_{2^m}$ であるとする．また，N のビット長は l である．

2.2 安全性定義

本研究では攻撃者として semi-honest なものを想定し，各パーティはプロトコルに正しく従うものとする．また，計算パーティは P_1, P_2, P_3 の 3 つであるとする．このとき，1 パーティの corrupt に対する秘密計算の安全性は以下のように定義される [9].

定義 2.1

プロトコル π が functionality $f = (f_1, f_2, f_3)$ を安全に計算するとは, $i = 1, 2, 3$ に対して, ある確率的多項式時間シミュレータ $\mathcal{S}_i$ が存在し,

$$\{\mathbf{View}^{\pi}_{i,x_1,x_2,x_3}, \mathbf{Output}^{\pi}_{x_1,x_2,x_3}\} \equiv \{\mathcal{S}_i(x_i, f_i), f\}$$

が成り立つことを言う．ただし，$\mathbf{View}^{\pi}_{i,x_1,x_2,x_3}$ とは，プロトコル π において，入力が (x_1, x_2, x_3) であったとき，パーティ P_i に送られるメッセージと P_i が使用する乱数の集合であり，$\mathbf{Output}^{\pi}_{x_1,x_2,x_3}$ は入力が (x_1, x_2, x_3) であった時のプロトコル π の出力である．

この定義は，直感的には，プロトコルの途中で各パーティが手にする値はすべて，自身の入出力をもとに多項式時間でシミュレートすることができる，ということを言っている．semi-honest 安全性に関しては，このようなシミュレータの構成は難しくないため，安全性証明は直感的な説明にとどめ，厳密な証明は省略する．

また，semi-honest な攻撃者に対する安全性の場合，以下のような結合定理が成立することが知られている [9].

定理 2.1

プロトコル π が functionality f を安全に計算し，f をオラクルとして使用するプロトコル Π^f が functionality g を安全に計算するとき，オラクル f の呼び出しを π に変換したプロトコル Π^{π} は g を安全に計算する．

2.3 秘密分散

秘密分散は，情報をシェアと呼ばれるいくつかの情報に分割することで，いくつかのシェアが集まらない限り元の情報を復元できなくする技術である．情報の復元にシェアをどれほど集める必要があるかなどは秘密分散の具体的な構成に依存する．本節では t-out-of-t 秘密分散と 2-out-of-3 複製秘密分散を取り扱う．

2.3.1 t-out-of-t 秘密分散

本稿で扱う $\mathbb{Z}_M$ 上の t-out-of-t 秘密分散では, 情報 $x \in \mathbb{Z}_M$ から，以下のようにシェア $[\![x]\!]$ が作られる：

$$[\![x]\!] = \{[\![x]\!]_1, [\![x]\!]_2, \ldots, [\![x]\!]_t\}$$
$$\text{s.t. } [\![x]\!]_i \overset{\mathrm{R}}{\in} \mathbb{Z}_M \ (i = 1, \ldots, t-1), \quad \sum_{i=1}^{t} [\![x]\!]_i =_M x.$$

t-out-of-t 秘密分散では t 個のシェアをすべて足すことで元の情報を復元することができ，逆にすべてのシェアが集まらないと元の情報を復元することはできない．また，線形演算がローカルで計算できるという性質がある．つまり，$a[\![x]\!] + b[\![y]\!] = [\![ax + by]\!]$ である．

2.3.2 2-out-of-3 複製秘密分散

本稿で扱う Z_M 上の 2-out-of-3 複製秘密分散は，情報 $x \in \mathbb{Z}_M$ から，以下のようにシェア $\langle x \rangle$ が作られる：

$$\langle x \rangle = (\langle x \rangle_1, \langle x \rangle_2, \langle x \rangle_3) = ((x_{31}, x_{12}), (x_{12}, x_{23}), (x_{23}, x_{31}))$$
$$\text{s.t. } x_{12}, x_{23} \overset{\mathrm{R}}{\in} \mathbb{Z}_M, \ x_{12} + x_{23} + x_{31} =_M x.$$

2-out-of-3 複製秘密分散では，シェアを 2 つ集めると x_{12}, x_{23}, x_{31} が集まるため，それらを足すことで元の情報を復元でき，逆に 1 つのシェアからだけではもとの情報を復元することはできない．また，t-out-of-t 秘密分散と同じく，線形演算はローカルで計算できる．

2.4 関数秘密分散

関数秘密分散（Function Secret Sharing, FSS）は [3] にて提案された暗号技術である．通常の秘密分散を用いた秘密計算では，入力を分散し，（公開された）関数を計算する．FSS では，入力が公開されており，関数が秘匿されている．今回使用する関数クラスはポイント関数と呼ばれるクラス $\mathcal{F}$ で,

$$\mathcal{F} = \{f(a, \cdot)\}_{a \in \mathbb{Z}_M} \text{ s.t. } f(a, \cdot) : \mathbb{Z}_M \to \mathbb{Z}_{M'},$$
$$f(a, x) = \begin{cases} 1 & (x = a) \\ 0 & (\text{otherwise}) \end{cases}$$

で定義される．ポイント関数の定義域 $\mathbb{Z}_M$ と値域 $\mathbb{Z}_{M'}$ は一般的には異なる．本稿で扱う FSS の正当性，安全性は以下のように定義される [3]：

$$f_0, f_1 \leftarrow \mathrm{FSS}(1^{\lambda}, a, \mathcal{F}),$$
正当性： $f_0(x) + f_1(x) =_{M'} f(a, x),$
安全性： $\exists$ 多項式時間シミュレータ $\mathcal{S}_i$
$$\text{s.t. } \{f_i\} \equiv \{\mathcal{S}_i(1^{\lambda}, \mathcal{F})\}.$$

特に，ポイント関数に対する FSS は DPF (Distributed Point Function) と呼ばれ，一方向性関数から構成が可能である．実用的には AES を用いて効率的に実装することができる．具体的には [3], [4] を参照してほしい．以降の記述では，DPF の引数としてセキュリティパラメータや $\mathcal{F}$ を省略する．

2.5 秘匿配列アクセスの問題設定

秘匿配列アクセスは秘匿された配列から秘匿されたインデックスに対応する値をとってくる処理である．今回は配列やインデックスは 2-out-of-3 複製秘密分散を用いて秘匿されているものとする．このとき，秘匿配列アクセスは以下のように定式化される：

入力：$\langle T\rangle, \langle x\rangle$,

出力：$\langle T[x]\rangle$.

なお，配列 T に対する $\langle T\rangle$ は，各要素が 2-out-of-3 複製秘密分散のシェアになっているものとして定義する．今回は，簡単のため，$\langle T\rangle, \langle T[x]\rangle$ は $\mathbb{Z}_{2^m}$ 上のシェアとし，$\langle x\rangle$ は $\mathbb{Z}_N$ 上のシェアとする．ただし，N は配列の要素数である．

提案手法

3. 提案手法

この章では提案手法の具体的な構成について述べる．まず，3.1 節で秘匿配列アクセスを別の簡単な問題に言い換える．3.2 節，3.3 節では 3.1 節で言い換えた問題に対して，$O(\sqrt{N})$, $O(\log N)$ 通信量で実現するプロトコルを構成する．なお，和などの演算は特に断りがなければ $\mathbb{Z}_{2^m}$ 上の演算であるとし，$+_{2^m}$, $=_{2^m}$ を $+$, $=$ で略記する．

3.1 問題の言いかえ

秘匿配列アクセスは，配列そのものとインデックスがともに秘匿されているところが一つの難しい点である．本節では秘匿配列アクセスの問題を，P_1, P_2 に公開されている配列と，P_3 が持っているインデックスをもとに配列アクセスを行う問題に言い換える．このような言いかえを行うことで，PIR や OT の手法を適用できるようになる．実際一つ目の提案手法（3.2 節）では OT の技法 [24] を，二つ目の提案手法（3.3 節）では PIR の技法 [3] を用いて構成している．

元の問題における各パーティの入力は，配列，インデックスの 2-out-of-3 複製秘密分散値であった．P_i が持っている配列，インデックスのシェアを $(T_{i-1,i}, T_{i,i+1}), (x_{i-1,i}, x_{i,i+1})$ とする．配列，インデックスの平文は $T = T_{1,2} + T_{2,3} + T_{3,1}, x = x_{1,2} + x_{2,3} + x_{3,1}$ である．$T[x]$ のシェアをもとめるには，$T_{1,2}[x]$ のシェアが計算できればよい．実際，$T_{2,3}, T_{3,1}$ に対しても同様のことを行い，それらを足し合わせることで $T[x]$ のシェアを計算できる．よって，以下では $T_{1,2}[x]$ のシェアを計算することを考える．

P_1, P_2 は $T_{1,2}, x_{1,2}$ を共有しているので，新たな配列 $T'_{1,2}$ を以下のように構成する:

$$T'_{1,2}[k] = T_{1,2}[x_{1,2} +_N k].$$

すると，P_3 が計算できる値 $x' =_N x_{2,3} + x_{3,1}$ に対して，$T'_{1,2}[x'] = T_{1,2}[x]$ となる．まとめると，$T_{1,2}[x]$ のシェアを求めるという問題は**表 2** の問題へと言い換えることができる．以降，3.2 節，3.3 節では，表 2 の問題を解くことを考える．なお，各節で安全性について触れるが，直感的な説明にとどめ，厳密な証明は省略する．

表 2 問題の言いかえ

パーティ	P_1	P_2	P_3
入力	T'	T'	x'
出力	$\langle T'[x']\rangle_1$	$\langle T'[x']\rangle_2$	$\langle T'[x']\rangle_3$

3.2 $O(\sqrt{N})$ 通信量のプロトコル

3.2.1 プロトコルの構成

本節では $O(\sqrt{N})$ 通信量のプロトコルを構成する．構成の基本的なアイデアとしては [24], [25] と同じく，配列を行列で表現し，行列とベクトルの積で配列アクセスを実現する．具体的には以下のとおりである．簡単のため，$N = n^2$ とする．まず，配列 T' をもとに，$n \times n$ 行列 M を $M_{i,j} = T'[ni + j]$ で定める．次に x' を n で割った商 q と余り r を求め，q, r 番目の要素だけ 1 でそれ以外が 0 となる単位ベクトル $\mathbf{e}_q, \mathbf{e}_r$ を求める．最後に $\mathbf{e}_q^{\mathrm{T}} M \mathbf{e}_r$ を計算することで，$T'[x']$ が求まる．[25] では除算や単位ベクトルの生成，行列積などの演算を秘密計算により実現することで秘匿配列アクセスプロトコルを構成している．[24] では，秘匿配列アクセスではなく OT を扱っているためインデックス x の除算などはローカルで行うことができ，行列積に対応する演算を以下のように準同型暗号を用いて実現している．まず，インデックスを持つクライアントは，q, r に対する単位ベクトル $\mathbf{e}_q, \mathbf{e}_r$ の各要素を暗号化したベクトル $\{\mathrm{Enc}(u_i)\}_i, \{\mathrm{Enc}(v_i)\}_i$ ($u_q = 1, v_r = 1$，それ以外は 0) をサーバに送る．サーバは乱数 a_k を用いて，

$$y_k = \mathrm{Enc}(a_k) + \sum_{j=1}^{n} T'[kn + j]\mathrm{Enc}(v_j), \tag{1}$$

$$z = \sum_{j=1}^{n} a_j \mathrm{Enc}(u_j), \tag{2}$$

を計算し，クライアントに送る．準同型暗号の性質から，$y_k = \mathrm{Enc}(T'[kn + r] + a_k)$, $z = a_q$ となる．各 y_k の平文は乱数 a_k によりマスクされており，z を用いることで，$k = q$ の場合のみマスクを外し，$T'[nq + r]$ を得ることができる．これより，クライアントは $T'[x]$ を得つつ，それ以外の配列の値についての情報を得ることはない．

提案手法は [24] の方式にのっとる．着目するのは，配列を保持するパーティが P_1, P_2 の 2 つであるので，(1) における $\mathrm{Enc}(val)$ を val の 2-out-of-2 の加法的秘密分散としても同様の計算ができる点である．(一方，問題を言い換える前の秘匿配列アクセスにおいては，T も秘匿であるので，(1) の計算は準同型演算にならず，通信が必要となる．) ここで，[24] では OT の問題設定から，$T'[x']$ の平文がクライアントに知られてよかったが，今考えている問題では，出力もシェアの形であるので，(1) を以下のように変更する：

Protocol 1 表 2 の問題に対する $O(\sqrt{N})$ サイズ通信量のプロトコル

Input: (P_1, P_2, P_3) : (T', T', x')
Output: $\langle T'[x] \rangle$

1: P_3 は x' を $n = \sqrt{N}$ で割った商 q と余り r を計算し，対応する n 次元単位ベクトルの 2-out-of-2 のシェア $\{[\![u_i]\!]\}, \{[\![v_i]\!]\}$ ($u_q = 1, v_r = 1$, それ以外は 0) を P_1, P_2 に送る．
2: $P_i (i = 1, 2)$ は一様乱数 R_i を計算する．
3: P_1, P_2 は $k = 1, \ldots, n$ に対し一様乱数 a_k を共有し，$[\![y_k]\!], [\![z]\!]$ を以下のように計算する：
$P_1 : [\![y_k]\!]_1 = -R_1 + a_k + \sum_{j=1}^{n} T'[kn + j][\![v_j]\!]_1$,
$\quad [\![z]\!]_1 \ = \sum_{j=1}^{n} a_j [\![u_j]\!]_1$,
$P_2 : [\![y_k]\!]_2 = -R_2 + \sum_{j=1}^{n} T'[kn + j][\![v_j]\!]_2$,
$\quad [\![z]\!]_2 \ = \sum_{j=1}^{n} a_j [\![u_j]\!]_2$.
4: $P_i (i = 1, 2)$ は $[\![y_k]\!], [\![z]\!]$ を P_3 に送り，P_3 は y_k, z を復元する．
5: P_1, P_2, P_3 は $R_1, R_2, y_q - z$ を 2-out-of-3 のシェア $\langle R_1 \rangle, \langle R_2 \rangle, \langle y_q - z \rangle$ に分割し，自分以外のパーティに分配する．
6: $\langle R_1 \rangle + \langle R_2 \rangle + \langle y_q - z \rangle$ を出力する．

$$y_{k,i} = -R_i + [\![a_k]\!]_i + \sum_{j=1}^{n} T'[kn + j][\![v_j]\!]_i, \tag{3}$$

$$z_i = \sum_{j=1}^{n} a_j [\![u_j]\!]_i. \tag{4}$$

なお，上記の式は P_i $(i = 1, 2)$ の計算式である．a_k は P_1, P_2 が共有している乱数であり，R_i は P_i が保持する乱数である．こうすることで，$(R_1, R_2, y_{q,1} + y_{q,2} - z_1 - z_2)$ は $T'[x]$ の 3-out-of-3 のシェアになっている．3-out-of-3 のシェアから 2-out-of-3 複製秘密分散のシェアへの変換は一回の通信で容易に可能である．プロトコルの擬似コードを Protocol 1 に示す．なお，問題を言い換えたうえで $\mathbf{e}_q^{\mathrm{T}} M \mathbf{e}_r$ を愚直に乗算を用いて計算する，という方針でも計算可能であるが，乗算の際に P_1, P_2 間での通信が発生するため，通信ラウンドの観点で上記の方針が優れる．

3.2.2 安全性

P_1, P_2 がほかのパーティからもらう値は，Step 1 で P_3 からもらうシェアと Step 5 でほかのパーティからもらうシェアである．これらのシェアの分布は一様乱数と同じ分布であるので，シミュレート可能である．また，P_3 がほかのパーティからもらう値は，Step 4 で P_1, P_2 からもらうシェアと Step 5 でほかのパーティからもらうシェアである．Step 4 で P_3 が得る情報は y_k $(k = 1, \ldots, n)$ と z であるが，

$$y_k = -R_1 - R_2 + a_k + T'[kn + r]$$
$$z = a_q$$

であるため，R_1, R_2, a_k が一様乱数であることからこれらはシミュレート可能である．Step 5 でもらう値もシェアの性質からシミュレート可能である．これより，P_i の View は P_i の入力のみからシミュレート可能であり，安全である．

Protocol 2 表 2 の問題に対する $O(\log N)$ サイズ通信量のプロトコル

Input: (P_1, P_2, P_3) : (T', T', x')
Output: $\langle T'[x] \rangle$

1: P_3 は $(f_1, f_2) \leftarrow \mathrm{DPF}(x')$ を計算し，f_i を P_i に送る．
2: $P_i (i = 1, 2)$ は $y_i = \sum_{k=1}^{N} T'[k] f_i(k)$ を計算する．
3: $P_i (i = 1, 2)$ は y_i を 2-out-of-3 のシェアに分割し，自分以外のパーティに分配する．
4: $\langle y_1 \rangle + \langle y_2 \rangle$ を出力する．

3.2.3 コスト

ローカルの計算コストに関しては Step 3 が律速である．各 y_k の計算に $O(n)$ かかり，$k = 1, \ldots, n$ に対して計算するので，合計で $O(n^2) = O(N)$ の計算コストとなる．通信ラウンドに関しては Step 1, Step 4, Step 5 で各 1 ラウンドの合計 3 ラウンドである[*1]．また通信量に関しては，Step 1, Step 3 が律速であり，m ビット整数を $O(n)$ 個分の合計 $O(mn) = O(m\sqrt{N})$ ビットの通信量となる．なお，出力として 3-out-of-3 のシェアを許すならば，Step 5 が不要であるため通信ラウンドが 2 ラウンドに減る．

3.3 $O(\log N)$ 通信量のプロトコル

3.3.1 プロトコルの構成

本節では $O(\log N)$ 通信量のプロトコルを構成する．構成において着目するのは，3.1 節で言い換えた問題が 2 サーバーの PIR と同じ設定になっている点である．2 サーバーの PIR では FSS を用いた効率的な構成 [3] が知られており，その手法を適用することができる．具体的には以下のとおりである．構成で用いる DPF の定義域は Z_N，値域は Z_{2^m} とする．まず，P_3 が $(f_1, f_2) \leftarrow \mathrm{DPF}(x')$ を計算し，P_i に f_i を送る $(i = 1, 2)$．その後，P_i は $y_i = \sum_{k=1}^{N} T'[k] f_i(k)$ を計算する．すると，(y_1, y_2) は $T'[x']$ の 2-out-of-2 の加法的なシェアになっている．実際，

$$\begin{aligned} y_1 + y_2 &= \sum_{k=1}^{N} T'[k](f_1(k) + f_2(k)) \\ &= T'[x'] \ \ (\because \text{DPF の性質}) \end{aligned}$$

であるため，確かに，2-out-of-2 のシェアになっていることがわかる．2-out-of-2 のシェアから 2-out-of-3 複製秘密分散のシェアへの変換は一回の通信で容易に可能である．プロトコルの擬似コードを Protocol 2 に示す．

3.3.2 安全性

$P_i (i = 1, 2)$ がほかのパーティから受け取る値は Step 1 における f_i と，Step 3 におけるシェアである．f_i に関しては DPF の安全性要件から多項式時間でシミュレート可能である．Step 3 で受け取るシェアに関しては，一様乱数と

[*1] Step 3 での P_1, P_2 間の乱数の共有に関しては，Step 1 と並行して行うこともできるし，PRF の鍵をあらかじめ共有することでローカルで計算もできるため，通信ラウンドにカウントしていない．

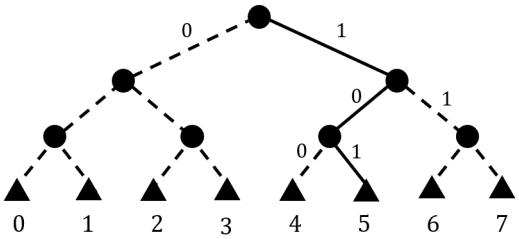

図 1 $f_i(5)$ の計算における遷移の例．根から始まり実線に沿って計算が遷移していく．

同じ分布であるためシミュレート可能である．また，P_3 が受け取る値は Step 3 におけるシェアのみであるため，同様にシミュレート可能である．これより，P_i の View は各々の入力のみから多項式時間でシミュレート可能であり，安全である．なお，このプロトコルの安全性は DPF の安全性に依拠するため，計算量的安全性であることに注意する．

3.3.3 コスト

ローカルの計算コストに関しては，Step 2 が律速であり，各 $f_i(\cdot)$ の計算に $O(\log N)$，合計で $O(N \log N)$ かかっている．しかし，3.3.4 節で述べる工夫により，計算量は線形オーダーにまで落とすことができる．通信ラウンドに関しては，Step 1, Step 3 でそれぞれ 1 回の通信が発生しているので合計で 2 ラウンドである．通信量に関しては，Step 1 での通信において，DPF で生成される関数のサイズがおよそ $m + \lambda \log N$ であり，Step 3 の通信では，2-out-of-3 複製秘密分散のシェア 3 個分のやり取りが発生する．これらを合わせると，通信量は $O(m + \lambda \log N)$ ビットとなる．なお，出力として 2-out-of-2 のシェアを許すならば，P_1, P_2 がそれぞれ y_1, y_2 を出力とすればよく，Step 3 が不要となり，通信ラウンドは 1 ラウンドになる[*2]．

3.3.4 計算量を線形オーダーに落とすための工夫

3.3.3 節で述べたように，f_i の計算を要素数回行う，という単純な実装では，計算量が $O(N \log N)$ になってしまう．この節では，計算量を $O(N)$ に落とすための構成を説明する．

ポイント関数に対する FSS では生成された関数 $(f_1, f_2) \leftarrow \mathrm{DPF}(a)$ に関して，$f_i(x)$ の計算では直感的には以下のような処理が行われる（詳細に関しては，文献 [3], [4] を参照してほしい）: まず, x をビット分解し, $x = x_1 x_2 \dots x_l$ を得る．その後 $s_k = h(x_k, s_{k-1}),\ (k = 1, \dots, l)$ を計算していき，s_l を出力とする．ただし，s_0 は f_i として送られてきた文字列である．h の具体的な記述は省略するが h の計算は定数時間で可能である．例として，$l = 3, x = 5$ の場合の計算の遷移を図 1 に示す．$x = 5$ のビット分解が 101 であることより，図 1 の実線に従って計算が進んでいく．

[*2] 言い換えた問題の上では 2-out-of-2 のシェアとなるが，元の秘匿配列アクセスに適用する際は P_1, P_2, P_3 の役割を巡回的に入れ替え，それぞれの問題で得られた出力を足すので最終的には 3-out-of-3 のシェアになる．

$f_i(\cdot)$ の計算には二分木の高さ l に等しい回数の h を計算する必要があるため，計算量は $O(l)$ である．この計算を二分木の葉の数 2^l 回だけ愚直に行うと，合計の計算量が $O(l2^l) = O(N \log N)$ になる．しかし，図 1 において例えば $x = 4$ に対しても $f_i(x)$ を計算したい場合，根から 2 段階目までは $f_i(5)$ と同じ計算をすることになるため無駄がある．すべての葉に対して $f_i(\cdot)$ を計算したい場合，各節点での分岐をすべてたどればよいため，二分木の辺の数だけ h を計算すればよいとわかる．この場合，辺の数が $2^{l+1} - 2 = O(N)$ 本であるため，$O(N)$ 回の h の呼び出しで全ての葉に対する $f_i(\cdot)$ が得られるとわかる．

4. 評価

この章では提案手法の評価を行う．4.1 節では理論的に評価し，先行研究とも比較する．4.2 節では提案手法の実装評価を行う．

4.1 理論評価

提案手法と先行研究の計算量，通信量，通信ラウンドなどを**表 3** にまとめる．提案手法において，出力は 3-out-of-3 のシェアを許しているため，3 章で評価したラウンド数より 1 ラウンド小さくなっている．なお，漸近表示において，各配列要素のビットサイズ m，セキュリティパラメータ λ は定数であるとして無視している．また，PathORAM は ORAM であるため，1.1 節で述べたとおり問題設定が異なることに注意する．秘匿配列アクセスの問題設定で PathORAM を用いる場合は [13] のように，PathORAM におけるクライアントの処理をすべて秘密計算によってエミュレートする必要があるが，漸近的には polylog であることに変わりはなく，計算量の観点では PathORAM が最も優れている．しかし，秘密計算では実行環境によっては通信コストが律速になることも多くあるため，通信コストに注目すると，通信量の観点，通信ラウンドの観点では提案手法（3.3 節）が最も優れる．なお，[25], [26] の手法における通信ラウンドは定数ラウンドであるものの，公開値除算 [19]，単位ベクトルの生成 [14]，擬似ランダム置換の計算 [5] などの処理があるため，定数はある程度大きいと考えられる．配列アクセスは用途に応じて配列サイズ N が非常に大きくなることも考えられ，その場合は提案手法のような $O(N)$ 計算量の手法は性能が悪くなることも考えられるが，ある程度の大きさであれば通信が律速になると考えられるため，提案手法の性能が良くなると考えられる．

4.2 実装評価

各配列要素として 64 ビット整数を用いて実装を行った．また，DPF としては [4] のものを用い，内部で用いる擬似乱数生成器は 128 ビットの AES を用いた．さらに，乱数の生成なども AES を用いており，また，乱数の共有に関して

表 3　提案手法と先行研究の理論比較.

	計算量	通信量	通信ラウンド	安全性
PathORAM[21]	$O(\mathrm{polylog}(N))$	$O(\mathrm{polylog}(N))$	$O(\log N)$	情報論的
[25]	$O(N)$	$O(\sqrt{N} \times \mathrm{polylog}(N))$	$O(1)$	情報論的
[26]	$O(\sqrt{N} \times \mathrm{polylog}(N))$	$O(\sqrt{N} \times \mathrm{poylog}(N))$	$O(1)$	計算量的
提案手法（3.2 節）	$O(N)$	$O(\sqrt{N})$	2	情報論的
提案手法（3.3 節）	$O(N)$	$O(\log N)$	1	計算量的

は AES の鍵をあらかじめ共有し，ローカルで乱数を計算している．使用言語は C++であり，1 つのラップトップ PC 内でソケット通信を用いたプロセス間通信を行い実行している．用いた PC の CPU は Intel Core i7-8565U 1.8 GHz，メモリが 16GB である．また，PC の OS は Windows 10 であるが，WSL 上で実験，計測を行った．結果は**表 4** のとおりである．なお，実行時間は 100 回の試行の平均であり，通信量は 1 つのパーティが送受信するデータ量である．[26] では $N = 2^{20}$ の場合の一回のアクセスに（机上の計算で）20.3 ms かかっているが，（環境は異なるものの）提案手法（3.2 節）では 13.4 ms と，高速になっていることが確認できる．$O(\log N)$ 通信量の提案手法は，通信量は小さいものの，ローカルの計算が $O(\sqrt{N})$ 通信量の提案手法に比べやや複雑なため，実行時間が多くかかるという結果になっている．しかし，本来の秘密計算の設定どおり 3 台の PC を用いて実行する，並列化処理を行う，などによりさらなる実行時間の改善が見込める．また，今回はプロセス間通信での計測のため通信のオーバーヘッドが比較的小さいが，例えば WAN 環境といった通信遅延が大きく，通信帯域が小さい環境では性能は逆転すると考えられる．

謝辞　本研究開発は総務省 SCOPE（受付番号 182103105）の委託を受けたものであり，また JST CREST（課題番号 JPMJCR19F6）の援助を受けている．

参考文献

[1] Assaf Ben-David, Noam Nisan, and Benny Pinkas. Fairplaymp: a system for secure multi-party computation. In *Proceedings of the 2008 ACM Conference on Computer and Communications Security, CCS 2008, Alexandria, Virginia, USA, October 27-31, 2008*, pp. 257–266, 2008.

[2] Marina Blanton, Ah Reum Kang, and Chen Yuan. Improved building blocks for secure multi-party computation based on secret sharing with honest majority. In *Applied Cryptography and Network Security, 18th International Conference, ACNS 2020, Roma, Italy, October 19-22*, 2020.

[3] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II*, Vol. 9057 of *Lecture Notes in Computer Science*, pp. 337–367. Springer, 2015.

[4] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing: Improvements and extensions. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pp. 1292–1303. ACM, 2016.

[5] Koji Chida, Koki Hamada, Dai Ikarashi, Ryo Kikuchi, and Benny Pinkas. High-throughput secure AES computation. In Michael Brenner and Kurt Rohloff, editors, *Proceedings of the 6th Workshop on Encrypted Computing & Applied Homomorphic Cryptography, WAHC@CCS 2018, Toronto, ON, Canada, October 19, 2018*, pp. 13–24. ACM, 2018.

[6] Benny Chor, Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan. Private information retrieval. *J. ACM*, Vol. 45, No. 6, pp. 965–981, 1998.

[7] Ghada Dessouky, Farinaz Koushanfar, Ahmad-Reza Sadeghi, Thomas Schneider, Shaza Zeitouni, and Michael Zohner. Pushing the communication barrier in secure computation using lookup tables. In *24th Annual Network and Distributed System Security Symposium, NDSS 2017, San Diego, California, USA, February 26 - March 1, 2017*, 2017.

[8] Oded Goldreich. Towards a theory of software protection and simulation by oblivious rams. In Alfred V. Aho, editor, *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*, pp. 182–194. ACM, 1987.

[9] Oded Goldreich. *The Foundations of Cryptography - Volume 2: Basic Applications*. Cambridge University Press, 2004.

[10] Seny Kamara, Payman Mohassel, and Mariana Raykova. Outsourcing multi-party computation. *IACR Cryptology ePrint Archive*, Vol. 2011, p. 272, 2011.

[11] Seny Kamara, Payman Mohassel, and Ben Riva. Salus: a system for server-aided secure function evaluation. In *the ACM Conference on Computer and Communications Security, CCS'12, Raleigh, NC, USA, October 16-18, 2012*, pp. 797–808, 2012.

[12] Seny Kamara and Mariana Raykova. Secure outsourced computation in a multi-tenant cloud. In *IBM Workshop on Cryptography and Security in Clouds*, pp. 15–16, 2011.

[13] Marcel Keller and Peter Scholl. Efficient, oblivious data structures for MPC. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014, Proceedings, Part II*, Vol. 8874 of *Lecture Notes in Computer Science*, pp. 506–525. Springer, 2014.

[14] John Launchbury, Iavor S. Diatchki, Thomas DuBuisson, and Andy Adams-Moran. Efficient lookup-table protocol in secure multiparty computation. In Peter Thiemann and Robby Bruce Findler, editors, *ACM SIGPLAN International Conference on Functional Programming, ICFP'12, Copenhagen, Denmark, Septem-*

表 4　配列サイズと実行時間，通信量．単位はそれぞれミリ秒，キロバイトで記載．

配列サイズ		2^4	2^6	2^8	2^{10}	2^{12}	2^{14}	2^{16}	2^{18}	2^{20}
提案手法（3.2 節）	実行時間	0.166	0.139	0.137	0.148	0.198	0.412	1.20	4.23	13.4
	通信量	0.288	0.544	1.06	2.08	4.13	8.22	16.4	32.8	65.6
提案手法（3.3 節）	実行時間	0.124	0.119	0.121	0.237	0.685	2.56	9.35	33.5	133
	通信量	0.384	0.512	0.640	0.768	0.896	1.02	1.15	1.28	1.41

ber 9-15, 2012, pp. 189–200. ACM, 2012.

[15] Payman Mohassel, Ostap Orobets, and Ben Riva. Efficient server-aided 2pc for mobile phones. *PoPETs*, Vol. 2016, No. 2, pp. 82–99, 2016.

[16] Payman Mohassel and Peter Rindal. Aby^3: A mixed protocol framework for machine learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pp. 35–52, 2018.

[17] Payman Mohassel and Yupeng Zhang. Secureml: A system for scalable privacy-preserving machine learning. In *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*, pp. 19–38, 2017.

[18] Hiraku Morita, Nuttapong Attrapadung, Tadanori Teruya, Satsuya Ohata, Koji Nuida, and Goichiro Hanaoka. Constant-round client-aided secure comparison protocol. In *Computer Security - 23rd European Symposium on Research in Computer Security, ESORICS 2018, Barcelona, Spain, September 3-7, 2018, Proceedings, Part II*, pp. 395–415, 2018.

[19] Chao Ning and Qiuliang Xu. Multiparty computation for modulo reduction without bit-decomposition and a generalization to bit-decomposition. In *International Conference on the Theory and Application of Cryptology and Information Security*, pp. 483–500. Springer, 2010.

[20] Michael O Rabin. How to exchange secrets with oblivious transfer. Technical report, TR-81, Aiken Computation Laboratory, Harvard U., 1981.

[21] Emil Stefanov, Marten van Dijk, Elaine Shi, Christopher W. Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. Path ORAM: an extremely simple oblivious RAM protocol. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*, pp. 299–310. ACM, 2013.

[22] Sameer Wagh, Divya Gupta, and Nishanth Chandran. Securenn: 3-party secure computation for neural network training. *PoPETs*, Vol. 2019, No. 3, pp. 26–49, 2019.

[23] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th Annual Symposium on Foundations of Computer Science, Toronto, Canada, 27-29 October 1986*, pp. 162–167, 1986.

[24] Bingsheng Zhang, Helger Lipmaa, Cong Wang, and Kui Ren. Practical fully simulatable oblivious transfer with sublinear communication. In Ahmad-Reza Sadeghi, editor, *Financial Cryptography and Data Security - 17th International Conference, FC 2013, Okinawa, Japan, April 1-5, 2013, Revised Selected Papers*, Vol. 7859 of *Lecture Notes in Computer Science*, pp. 78–95. Springer, 2013.

[25] 濱田浩気. 劣線形通信量で定数ラウンドの秘密計算配列アクセスアルゴリズム. In *CSS*, 2017.

[26] 濱田浩気. 劣線形ローカル計算量で定数ラウンドの秘密計算配列アクセスアルゴリズム. In *SCIS*, 2019.