情報処理学会論文誌 Vol.59 No.9 1581–1595 (Sep. 2018)

秘密分散法を用いた四則演算の組み合わせに対して安全な次数変化のない秘匿計算

ムハンマド カマル アフマド アクマル アミヌディン[1,a)] 岩村 恵市[1]

受付日 2017年11月27日, 採録日 2018年6月8日

概要：一般に，Shamir の (k, n) 閾値秘密分散法を用いた秘匿計算では，乗算を行う際に乗算結果の多項式の次数が $k-1$ から $2k-2$ に変化してしまうため，復元に必要な分散値の個数が k から $2k-1$ に変化してしまうという問題があった．神宮らが提案した方式では，スカラー量 × 多項式というアプローチを用いて，次数変化の問題を解決したが，積和演算を実行すると，秘密情報が漏洩するという問題があった．そこで，本論文では，Shamir の (k, n) 閾値秘密分散を用いた秘匿計算において，積和演算に対しても次数が変化せず，かつ秘密情報が漏洩しない安全な秘匿計算手法を提案する．これによって秘匿四則演算の組合せを安全に実行できる．本論文では，passive な攻撃者を仮定し，(1) 秘匿乗算において秘密情報に 0 を含まない，(2) 攻撃者が知らない乱数を用いた 1 に対する分散値集合がある，(3) 演算の連続において各サーバが扱う分散値集合内の分散値の位置は固定されるという 3 つの前提条件をおく．この条件のもと，情報理論的な安全性を実現する秘匿計算が実現できることを示す．

キーワード：秘匿計算，秘密分散，$n < 2k-1$，マルチパーティ計算，情報理論的安全性

Conditionally Secure Multiparty Computation When $n < 2k-1$

Ahmad Akmal Aminuddin Mohd Kamal[1,a)] Keiichi Iwamura[1]

Received: November 27, 2017, Accepted: June 8, 2018

Abstract: We propose a new Secure Multiparty Computation (MPC) scheme using Shamir's (k, n)-threshold secret sharing scheme that enable computation of multiplication and addition to be done in consecutive manner while maintaining the condition of $n < 2k-1$. Typically, secure multiparty computation that employs a secret sharing scheme is not unconditionally secure if $n < 2k-1$. However, this also means that secure multiparty computation with $n < 2k-1$ is realizable using a conditionally secure approach. Therefore, in this study, we clarify the conditions needed as well as present a detailed method necessary to achieve conditionally secure multiparty computation using secret sharing scheme for $n < 2k-1$. By assuming a passive adversary, we propose a conditionally secure multiparty computation that is secure against $k-1$ number of adversary with a lower processing time compare to secure multiparty computation method proposed by Damgård et al. In this paper, we also evaluate the security of our proposed method.

Keywords: secure multiparty computation, MPC, secret sharing scheme, $n < 2-1$, information-theoritically secure

1. はじめに

近年，ビッグデータおよび IoT 環境の進歩にともない，個人に関する情報の活用が期待されている．ただし，個人情報の活用においては，その情報が漏洩すると個人のプライバシに影響を与える可能性がある．そのため，ビッグデータの利活用においては，個人に関する情報を守る技術と両立させる研究がさかんに行われている．

本論文では，ビッグデータの利活用と個人情報の保護の両立を目指して，データを守りながら演算を行うことができる秘匿計算技術を提案する．情報を守りながら演算を行

[1] 東京理科大学
Tokyo University of Science, Katsushika, Tokyo 125–8585, Japan
[a)] ahmad@sec.ee.kagu.tus.ac.jp

う手法は大きくて分けると，主に鍵を用いてデータを秘匿する準同型暗号 [3], [5], [14], [15], [17], [26], [28] と，鍵を用いずにデータを秘匿する秘密分散法を用いた秘匿計算 [2], [10], [12], [16], [24], [25], [29] がある．ただし，準同型暗号は一般的に計算量が多く，演算の処理に多大な時間がかかるという問題がある [5]．そのため，ビッグデータへの適用に対しては，計算量が重い準同型暗号よりも，計算量が軽い秘密分散を用いるというアプローチが検討されている．

秘密分散法とはユーザが持っている秘密情報を複数の異なる値（以降，分散値）に変換し，分散させる手法である．秘密分散法の 1 つである Shamir の (k, n) 閾値秘密分散法 [23] は，1 つの秘密情報を n 個の分散値に変換し，n 台のサーバに分散させる．Shamir の秘密分散法の特徴は，分散した n 個の分散値から，k 個の分散値を集めれば，元の秘密情報を復元することができるが，k 個未満の情報からは，秘密情報に関する情報をいっさい得ることができないということである．このことより，Shamir の秘密分散を用いると，$n > k$ の場合，一部のサーバの欠損に耐性を持つことが分かる．

一方，従来の秘密分散を用いた秘匿計算において，秘匿加算は容易に実現できるが，秘匿乗算を行う際に問題が生じることが知られている．すなわち，秘匿乗算を行うと多項式の次数が $k-1$ から $2k-2$ に変化してしまうので，元の情報を復元するために必要となる分散値の数が k 個から $2k-1$ 個に変化してしまうという問題である [2], [16]．それらの問題を解決するために著者らと同一の研究グループの Shingu らによって TUS 方式と呼ぶ 2 入力の秘匿演算法が提案されている [24]．TUS 方式では秘密情報を乱数によって秘匿し，秘匿乗算を行う際に，一時的にその秘匿化秘密情報を復元し，(*スカラー量* × *多項式*) の形で乗算を行うため，次数変化をさせず，秘匿乗算を実現できる．ただし，TUS 方式は，$ab + c$ のように乗算と加算を連続的に行う場合，秘密情報の安全性に問題がある．すなわち，TUS 方式を用いて $ab + c$ のような積和演算を実行すると，攻撃者が 1 つの入力（秘密情報と乱数）と演算結果を知ることができれば，残りの 2 つの入力を特定することができるという問題点である．

そこで，本論文では，上記問題を解決し，四則演算の連続に対しても安全な秘密分散法を用いた秘匿計算法を提案する．一般に，秘密分散を用いた秘匿計算，いわゆるマルチパーティプロトコルにおいては，$2k-1 > n$ の場合，無条件に安全（Unconditionally Secure）な計算は不可能とされている．しかし，ある条件をつけることによって，$2k-1 > n$ の場合でも秘匿計算が安全に実行できる可能性がある．よって，本論文では $2k-1 > n$ における秘匿計算の連続においても情報理論的な安全性を実現できるための条件を検討する．その条件の実現が現実的であれば，その秘匿計算は実用的な秘匿計算法ということができる．ただし，本論文では passive な攻撃者を仮定する．また，提案方式が SPDZ 2 方式 [15] などの従来方式に比べて，秘匿計算の効率が優れていることも示す．

本論文の構成を以下に示す．2 章では秘密分散法や TUS 方式などの関連技術について説明する．3 章では $2k-1 > n$ において積和演算の連続に対して安全な提案方式を説明し，それを実現するための条件を明らかにする．4 章では従来方式との比較および評価を行い，5 章でまとめを行う．

2. 関連研究

2.1 Shamir の (k, n) 閾値秘密分散法

Shamir の (k, n)-閾値秘密分散法 [23] とはある秘密情報を n 個の分散情報に分散し，そのうちの k 個以上の分散情報を集めることによって，元の秘密情報を復元することができるという方法である．ただし，それ未満の分散情報より元の秘密情報を得ることがいっさいできない．以下に Shamir の (k, n) 閾値秘密分散法の分散処理と復元処理を示す．

【分散処理】

1. ユーザは $s < p$ かつ $n < p$ の条件を満たす任意の素数 p を選択する．
2. ユーザは GF(p) の元から，n 個の x_i ($i = 0, 1, 2, \ldots, n-1$) を選び，サーバ ID とする．
3. ユーザは GF(p) の元から，$k-1$ 個の乱数 a_l（$l = 1, 2, \ldots, k-1$）を選び，以下の式を生成する．

$$W_i = s + a_1 x_i + a_2 {x_i}^2 + \cdots + a_{k-1} x_i^{k-1} \quad (mod\ p)$$

4. ユーザは上記式の x_i に各サーバ ID を代入し，分散値 W_i を計算し，各サーバ S_i に送信する．

【復元処理】

1. 復元に用いる分散情報を W_i（$i = 0, 1, 2, \ldots, k-1$）として，その分散情報に対応するサーバ ID を x_i とする．
2. 分散式に x_i と W_i を代入し，k 個の連立方程式を解いて，元の秘密情報 s を復元する．ただし，秘密情報 s を復元する際に，ラグランジュの補間公式を使うと便利である．

2.2 Damgård らの SPDZ 2 方式 [15]

Damgård らは，$n = k$ の設定において，不正者が参加者の半数以上の場合（dishonest majority）でも，安全なマルチパーティ計算 SPDZ 2 方式を提案した．SPDZ 2 方式は，秘密情報のオーナ自身が参加者の 1 人であり，自身以外のすべての参加者（$n-1$）が結託しても秘密情報の漏洩が発生しない．

SPDZ 2 方式は事前処理（preprocessing phase）とオンラインフェーズ（online phase）から構成されている．SPDZ 2 方式によるデータの秘匿性は加法的秘密分散により実

現される．SPDZ 2 により，秘匿加算処理は簡単に実現できるが，秘匿乗算に関しては，Beaver の回路ランダム化を用いて，multiplication triple と呼ぶ $a \cdot b = c$ の関係を満たすランダムな分散情報 $\langle a \rangle$，$\langle b \rangle$，$\langle c \rangle$ を生成し，乗算を実現する．SPDZ 2 方式の復元演算に関しては，たとえば，秘密情報 x の分散情報 $\langle x \rangle$ を入力とし，x を出力する処理を $x = open(\langle x \rangle)$ と表記する．SPDZ 2 方式の秘匿乗算処理をより具体的に説明すると，秘密情報 x，y の分散情報 $\langle x \rangle$，$\langle y \rangle$ を乗算するために，事前処理によって，$a \cdot b = c$ を満たすランダムな分散情報 $(\langle a \rangle, \langle b \rangle, \langle c \rangle)$ を生成し，$d = open(\langle x \rangle - \langle a \rangle)$，$e = open(\langle y \rangle - \langle b \rangle)$ を復元し，$\langle x \cdot y \rangle = d \cdot e + e \cdot \langle a \rangle + d \cdot \langle b \rangle + \langle c \rangle$ の式を利用し，乗算を実現する．ただし，SPDZ 2 の欠点としては，multiplication triple の生成は完全準同型暗号 [5] を利用しているので，計算量が多くなり，処理に非常に時間がかかる．

2.3 Damgård らの方式 [13]

Damgård らは秘密分散法をベースにした秘匿計算法を提案した．Damgård らが提案した方式は Franklin らの秘密分散法 [21] を用いて，分散情報どうしの秘匿加算，秘匿乗算および H-ゲートを実現する．ただし，H-ゲートの演算は，$\mathrm{H}(x, y, c) = (cx + (1-c)y, cy + (1-c)x)$ の演算である．秘匿加算処理に関しては，分散情報どうしのローカル演算で実行することができる．一方，秘匿乗算を実現するには，次のような処理が必要となる．たとえば，秘密情報 x，y に対する乗算結果 xy を求めたいとする．まず，事前処理として各サーバに d 次および $2d$ 次の多項式で分散された乱数 r の分散情報 $[r]_d$，$[r]_{2d}$ を準備する必要がある．ただし，$[r]_d$ という表記は，d 次多項式で表される r の分散情報を意味する．各サーバはそれらを用いて，$[xy+r]_{2d} = [x]_d[y]_d + [r]_{2d}$ の演算を行い，RobustReshare プロトコルによって，d 次の多項式で $[xy+r]_{2d}$ の結果を $[xy+r]_d$ として再分散をする．最後に，各サーバは $[r]_d$ を用いて，$[xy]_d = [xy+r]_d - [r]_d$ を計算することによって，d 次多項式で分散された xy の分散情報が得られる．

2.4 Gennaro らの方式 [16]

Gennaro らは Ben-Or らが提案した秘匿計算方式 [2] を改良し，より簡単に実現できる秘匿計算方式を提案した．Gennaro らが提案した方式において，秘匿加減算は簡単に実現できるが，秘匿乗算を行う際に，Ben-Or らの方式と同様に乗算結果の次数変化を防ぐ処理が必要となる．ここでは，Gennaro らが提案した passive な攻撃者に対する秘匿乗算プロトコル Simple-Mult を対象にして 4.1 節の比較を行う．秘匿乗算プロトコル Simple-Mult では，たとえば，秘密情報 a，b に対する乗算結果 ab を求めたいとする．各サーバは秘密情報 a，b に対する分散値 $f_a(i)$，$f_b(i)$ から $f_{ab}(x) = ab + \gamma_1 x + \cdots + \gamma_{2t} x^{2t}$ を計算する．ここで，$1 \le i \le 2t+1$，$f_{ab}(i) = f_a(i) f_b(i)$ とし，次のように書ける．

$$A \begin{bmatrix} ab \\ \gamma_1 \\ \vdots \\ \gamma_{2t} \end{bmatrix} = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & 2 & \cdots & 2^{2t} \\ \vdots & & & \\ 1 & 2t+1 & \cdots & 2t+1^{2t} \end{bmatrix} \begin{bmatrix} ab \\ \gamma_1 \\ \vdots \\ \gamma_{2t} \end{bmatrix} = \begin{bmatrix} f_{ab}(1) \\ f_{ab}(2) \\ \vdots \\ f_{ab}(2t+1) \end{bmatrix}$$

ここで，A は $(2t+1) \times (2t+1)$ のヴァンデルモンドの行列式であり，たとえば，A^{-1} の 1 行目を $(\lambda_1, \ldots, \lambda_{2t+1})$ とすれば，$ab = \lambda_1 f_{ab}(1) + \cdots + \lambda_{2t+1} f_{ab}(2t+1)$ より ab が求められる．

2.5 Nishide らの方式 [22]

Nishide らはビット分解プロトコルを用いない区間，等価性，比較のためのマルチパーティ計算を提案した．Nishide らの方式は，honest-but-curious の攻撃者を想定する．Nishide らが提案したマルチパーティ計算を実現するために，秘匿定数倍乗算，定数との秘匿加算，秘匿加算および秘匿乗算の演算が必要となる．たとえば，秘密情報 a，b および定数 c があるとする．定数との秘匿加算 $c + f_a(i) \bmod p$，定数倍乗算 $c f_a(i) \bmod p$ および秘匿加算 $f_a(i) + f_b(i) \bmod p$ の演算に関しては，すべてローカルで計算できる．ただし，秘匿乗算 $ab \bmod p$ の演算は複雑であり，サーバ間の通信が必要である．Nishide らは，秘匿乗算を実現するために 2.4 節で述べた Gennaro らが提案した乗算プロトコルを用いている．

2.6 TUS 方式

2.6.1 TUS 方式のプロトコル

以下に TUS 方式 [24] について説明する．なお，秘密情報 a，b は $GF(p)$ であり，分散処理および秘匿加算で生成する乱数は α_j，β_j，γ_j も α_j，β_j，$\gamma_j \in GF(p)$ である（ただし，秘匿乗算において秘密情報 a，b は 0 を含まず，秘匿加減算および秘匿除算において秘密情報に 0 を含んでもよい．また，いずれの場合でも乱数は 0 を含まない）．TUS 方式は秘密分散の処理も含めてすべての秘匿演算は p を法として行われる．また，TUS 方式におけるエンティティは各々の秘密情報を秘密分散して入力する入力者と，その分散値を用いて秘匿計算を行う n 台のサーバと，演算結果の分散値から復元を行う復元者からなる．

【記号定義】

- $\overline{[a]}_i$：値 a に対するサーバ S_i が保持する分散値．
- $[a]_i$：値 a に関連するサーバ S_i に保持する分散値集合．

【前提条件】
- 秘匿乗算において秘密情報に 0 を含まない．

【分散処理】
入力者 A の秘密情報：a

1. 入力者 A は k 個の乱数 $\alpha_0, \alpha_1, \ldots, \alpha_{k-1}$ を生成し，乱数 $\alpha = \prod_{j=0}^{k-1} \alpha_j$ を計算する．

2. 入力者 A は計算された乱数 α と秘密情報 a をかけて，αa を計算し，$\alpha a, \alpha_0, \alpha_1, \ldots, \alpha_{k-1}$ を Shamir の (k, n) 閾値秘密分散法で n 台のサーバ S_i $(i = 0, 1, \ldots, n-1)$ に分散させる．

3. サーバ S_i $(i = 0, 1, \ldots, n-1)$ は秘密情報 a に関する分散情報として $[a]_i = (\overline{[\alpha a]}_i, \overline{[\alpha_0]}_i, \ldots, \overline{[\alpha_{k-1}]}_i)$ を保持する．

【復元処理】

1. 復元者は k 台のサーバより k 個の分散情報 $[a]_j$ $(j = 0, 1, \ldots, k-1)$ を収集する．

2. 復元者は収集した分散情報の $\overline{[\alpha a]}_j, \overline{[\alpha_0]}_j, \ldots, \overline{[\alpha_{k-1}]}_j$ から $\alpha a, \alpha_0, \ldots, \alpha_{k-1}$ を復元し，乱数 $\alpha = \prod_{j=0}^{k-1} \alpha_j$ を計算する．

3. 復元者は復元した秘匿した秘密情報 αa と乱数 α を用いて，以下の式より秘密情報 a を復元する．

$$\alpha a \times \alpha^{-1} = a$$

【秘匿加減算】

入力：$[a]_j = (\overline{[\alpha a]}_j, \overline{[\alpha_0]}_j, \ldots, \overline{[\alpha_{k-1}]}_j)$ $(j = 0, 1, \ldots, k-1)$
$[b]_j = (\overline{[\beta b]}_j, \overline{[\beta_0]}_j, \ldots, \overline{[\beta_{k-1}]}_j)$ $(j = 0, 1, \ldots, k-1)$
出力：$[c]_i = [a \pm b]_i = (\overline{[\gamma(a \pm b)]}_i, \overline{[\gamma_0]}_i, \ldots, \overline{[\gamma_{k-1}]}_i)$ $(i = 0, 1, \ldots, n-1)$

1. k 台のサーバ S_j は $\overline{[\alpha_0]}_j, \ldots, \overline{[\alpha_{k-1}]}_j$ と $\overline{[\beta_0]}_j, \ldots, \overline{[\beta_{k-1}]}_j$ を収集し，α_j と β_j を復元する．それから，k 台のサーバ S_j は乱数 γ_j を生成し，サーバ S_0 に γ_j/α_j，γ_j/β_j を送信する．

2. サーバ S_0 は γ_j/α_j，γ_j/β_j を用いて，以下の式より γ/α，γ/β を一時的に復元し，全サーバ S_i に送信する．

$$\frac{\gamma}{\alpha} = \prod_{j=0}^{k-1} \frac{\gamma_j}{\alpha_j}$$

$$\frac{\gamma}{\beta} = \prod_{j=0}^{k-1} \frac{\gamma_j}{\beta_j}$$

3. 全サーバ S_i は以下の式を用いて，$\overline{[\gamma(a \pm b)]}_i$ を計算する．

$$\overline{[\gamma(a \pm b)]}_i = \frac{\gamma}{\alpha}(\overline{[\alpha a]}_i) \pm \frac{\gamma}{\beta}(\overline{[\beta b]}_i)$$

4. k 台のサーバ S_j は乱数 γ_j を Shamir の (k, n) 閾値秘密分散法で全サーバ S_i に分散させる．

5. サーバ S_i $(i = 0, 1, \ldots, n-1)$ は秘密情報 $c = a \pm b$ に関する分散情報として $[c]_i = [a \pm b]_i = (\overline{[\gamma(a \pm b)]}_i, \overline{[\gamma_0]}_i, \ldots, \overline{[\gamma_{k-1}]}_i)$ を保持する．

【秘匿乗除算】

入力：$[a]_j = (\overline{[\alpha a]}_j, \overline{[\alpha_0]}_j, \ldots, \overline{[\alpha_{k-1}]}_j)$ $(j = 0, 1, \ldots, k-1)$
$[b]_j = (\overline{[\beta b]}_j, \overline{[\beta_0]}_j, \ldots, \overline{[\beta_{k-1}]}_j)$ $(j = 0, 1, \ldots, k-1)$
出力：$[c]_i = [ab]_i = (\overline{[\alpha\beta ab]}_i, \overline{[\alpha_0\beta_0]}_i, \ldots, \overline{[\alpha_{k-1}\beta_{k-1}]}_i)$ $(i = 0, 1, \ldots, n-1)$

1. サーバ S_0 は k 台のサーバより $\overline{[\alpha a]}_j$ を収集し，一時的に αa のスカラー量を復元し，全サーバ S_i に送信する．

2. 全サーバ S_i は以下の式を用いて，$\overline{[\alpha\beta ab]}_i$ を計算する．

$$\overline{[\alpha\beta ab]}_i = \alpha a \times \overline{[\beta b]}_i$$

3. k 台のサーバ S_j は $\overline{[\alpha_0]}_j, \ldots, \overline{[\alpha_{k-1}]}_j$ と $\overline{[\beta_0]}_j, \ldots, \overline{[\beta_{k-1}]}_j$ を収集し，α_j と β_j を復元し，$\alpha_j\beta_j$ を計算する．

4. k 台のサーバ S_j は乱数 $\alpha_j\beta_j$ を Shamir の (k, n) 閾値秘密分散法で全サーバ S_i に分散させる．

5. サーバ S_i $(i = 0, 1, \ldots, n-1)$ は秘密情報 $c = ab$ に関する分散情報として $[c]_i = [ab]_i = (\overline{[\alpha\beta ab]}_i, \overline{[\alpha_0\beta_0]}_i, \ldots, \overline{[\alpha_{k-1}\beta_{k-1}]}_i)$ を保持する．

秘匿除算は秘匿乗算の 2. における計算を $\overline{[\beta b/\alpha a]}_i = \overline{[\beta b]}_i/\alpha a$ とし，3. における計算を β_j/α_j とすることによって実現され，その結果 $c = b/a$ の分散値集合 $[c]_i := (\overline{[\beta b/\alpha a]}_i, \overline{[\beta_0/\alpha_0]}_i, \cdots, \overline{[\beta_{k-1}/\alpha_{k-1}]}_i)$ が得られる．一般に，秘匿除算において除数が 0 の場合，解が定義できないため，除数が 0 である場合は排除する必要がある．提案方式は除数が 0 であったとしても，1. において $\alpha a = 0$ が検出されるため，そこで処理を止めることにより 0 による除数を排除できる．

TUS 方式は上記 2 入力の秘匿加減算および秘匿乗除算に関しては，以下の 3 種類の攻撃者に対して情報理論的安全性が実現できることが証明されている．ただし，攻撃者 1〜3 が知ろうとする情報が得られた場合，攻撃成功となるが，攻撃者はプロトコルに従う passive adversary を想定する．

攻撃者 1：サーバを乗っ取るなどして，$k-1$ 台のサーバが知る情報を知り，それをもとに秘匿計算の入力となる秘密情報，または秘匿計算の出力結果を知ろうとする．

攻撃者 2：秘匿計算に関する 1 つの値を入力する者が攻撃者となった場合であり，自らが入力した秘密情報および秘匿化秘密情報に用いられた乱数を知る．さらに，$k-1$ 台のサーバが知る情報も知ることができ，もう 1 人の入力者が入力した秘密情報，または秘匿計算出力を知ろうとする．

攻撃者 3：秘匿演算の復元を行う者が攻撃者となった場合であり，k 台のサーバから送られる，秘密情報を復元するために必要な情報を知る．さらに，$k-1$ 台のサーバが知る情報も知ることができ，秘匿計算の入力

となった秘密情報を知ろうとする．

ただし，攻撃者 4 として攻撃者が復元者かつ 1 つの値の入力者である場合も考えられるが，2 入力であればどのような秘匿計算法を用いても，攻撃者 4 は秘匿計算結果と自分の入力値からもう 1 人の入力者の秘密情報を知ることができる．よって，2 入力の秘匿計算に関しては，攻撃者 4 は想定しない．

2.6.2 TUS 方式の問題点

TUS 方式を組み合わせて，$f(x) = ab + c$ のような積和演算を行う場合を以下に示す．以下において，手順 1〜5 で秘匿乗算 ab を行い，6〜10 で秘匿加算 $ab+c$ を行う．なお，秘密情報 a，b，c は a，b，$c \in GF(p)$ であり，分散処理および秘匿加算で生成する乱数は α_j，β_j，λ_j，γ_j も α_j，β_j，λ_j，$\gamma_j \in GF(p)$ である（ただし，秘密情報と乱数は 0 を除く）．また，入力は a，b，c に関する分散値集合 $[a]_j$，$[b]_j$，$[c]_j$ である．秘密分散の処理も含めてすべての秘匿演算は p を法として行われる．

【$ab+c$ の秘匿演算】

入力：$[a]_j = (\overline{[\alpha a]}_j, \overline{[\alpha_0]}_j, \ldots, \overline{[\alpha_{k-1}]}_j)$ $(j = 0, 1, \ldots, k-1)$
$[b]_j = (\overline{[\beta b]}_j, \overline{[\beta_0]}_j, \ldots, \overline{[\beta_{k-1}]}_j)$ $(j = 0, 1, \ldots, k-1)$
$[c]_j = (\overline{[\lambda c]}_j, \overline{[\lambda_0]}_j, \ldots, \overline{[\lambda_{k-1}]}_j)$ $(j = 0, 1, \ldots, k-1)$
出力：$[ab+c]_i = (\overline{[\gamma(ab+c)]}_i, \overline{[\gamma_0]}_i, \ldots, \overline{[\gamma_{k-1}]}_i)$ $(i = 0, 1, \ldots, n-1)$

1. サーバ S_0 は k 台のサーバより $\overline{[\alpha a]}_j$ を収集し，一時的に αa のスカラー量を復元し，全サーバ S_i に送信する．
2. 全サーバ S_i は以下の式を用いて，$\overline{[\alpha\beta ab]}_i$ を計算する．

$$\overline{[\alpha\beta ab]}_i = \alpha a \times \overline{[\beta b]}_i$$

3. k 台のサーバ S_j は $\overline{[\alpha_0]}_j, \ldots, \overline{[\alpha_{k-1}]}_j$ と $\overline{[\beta_0]}_j, \ldots, \overline{[\beta_{k-1}]}_j$ を収集し，α_j と β_j を復元し，$\alpha_j\beta_j$ を計算する．
4. k 台のサーバ S_j は乱数 $\alpha_j\beta_j$ を Shamir の (k, n) 閾値秘密分散法で全サーバ S_i に分散させる．
5. サーバ S_i（$i = 0, 1, \ldots, n-1$）は秘密情報 ab に関する分散情報として $[ab]_i = (\overline{[\alpha\beta ab]}_i, \overline{[\alpha_0\beta_0]}_i, \ldots, \overline{[\alpha_{k-1}\beta_{k-1}]}_i)$ を保持する．
6. k 台のサーバ S_j は $\overline{[\alpha_0\beta_0]}_j, \ldots, \overline{[\alpha_{k-1}\beta_{k-1}]}_j$ と $\overline{[\lambda_0]}_j, \ldots, \overline{[\lambda_{k-1}]}_j$ を収集し，$\alpha_j\beta_j$ と λ_j を復元する．それから，k 台のサーバ S_j は乱数 γ_j を生成し，サーバ S_0 に $\gamma_j/\alpha_j\beta_j$，γ_j/λ_j を送信する．
7. サーバ S_0 は $\gamma_j/\alpha_j\beta_j$，γ_j/λ_j を用いて，以下の式より $\gamma/\alpha\beta$，γ/λ を計算し，全サーバ S_i に送信する．

$$\frac{\gamma}{\alpha\beta} = \prod_{j=0}^{k-1} \frac{\gamma_j}{\alpha_j\beta_j}$$

$$\frac{\gamma}{\lambda} = \prod_{j=0}^{k-1} \frac{\gamma_j}{\lambda_j}$$

8. 全サーバ S_i は以下の式を用いて，$\overline{[\gamma(ab+c)]}_i$ を計算する．

$$\overline{[\gamma(ab+c)]}_i = \frac{\gamma}{\alpha\beta}(\overline{[\alpha\beta ab]}_i) + \frac{\gamma}{\lambda}(\overline{[\lambda c]}_i)$$

9. k 台のサーバ S_j は乱数 γ_j を Shamir の (k, n) 閾値秘密分散法で全サーバ S_i に分散させる．
10. サーバ S_i（$i = 0, 1, 2, \ldots, n-1$）は秘密情報 $ab+c$ に関する分散情報として $[ab+c]_i = (\overline{[\gamma(ab+c)]}_i, \overline{[\gamma_0]}_i, \ldots, \overline{[\gamma_{k-1}]}_i)$ を保持する．

【復号処理】

1. 復元者は k 台のサーバより k 個の分散情報 $[ab+c]_j$ を収集する．
2. 復元者は収集した分散情報の $\overline{[\gamma(ab+c)]}_j, \overline{[\gamma_0]}_j, \ldots, \overline{[\gamma_{k-1}]}_j$ から $\gamma(ab+c), \gamma_0, \ldots, \gamma_{k-1}$ を復元し，乱数 $\gamma = \prod_{j=0}^{k-1} \gamma_j$ を計算する．
3. 復元者は復元した秘匿した秘密情報 $\gamma(ab+c)$ と乱数 γ を用いて，以下の式より秘密情報 $ab+c$ を復元する．

$$\gamma(ab+c) \times \gamma^{-1} = ab + c$$

積和演算は 3 入力 1 出力の演算であるため，入力者は 3 人，復元者は 1 人想定される．よって，攻撃者 4 として復元者かつ 1 つの値の入力者である場合を考える．たとえば，攻撃者 4 が秘密情報 b の入力者かつ復元者である場合，攻撃者 4 は入力者が入力した秘密情報 b，乱数 β，演算結果を復元するための乱数 γ，演算結果 $ab+c$ および $k-1$ 台のサーバから漏洩する αa，$\gamma/\alpha\beta$，γ/λ の情報を持っている．攻撃者 4 は乱数 β，γ，$\gamma/\alpha\beta$ の情報を用いて，乱数 α を求めることができる．攻撃者 4 は求めた乱数 α と演算途中に得られる αa より秘密情報 a を計算することができる．それから，攻撃者 4 は秘密情報 a，b と演算結果 $ab+c$ を用いて，秘密情報 c を知ることができる．これによって，TUS 方式は 1 つの入力情報，復元者の持つ情報および $k-1$ 台のサーバから漏洩する情報を持つ攻撃者 4 に対して，残りの入力者の情報を漏洩させてしまうという問題がある．ただし，積和演算は攻撃者 1〜3 に対しては安全である．

3. 提案方式

本章では，TUS 方式の問題である積和演算に対して，攻撃者 4 に対しても安全な手法を検討する．2.6.2 項から分かるように TUS 方式の単純な組合せでは秘密情報が漏洩する．これは，攻撃者が自ら知る情報と $k-1$ 台からの情報を用いることによって，他の入力を求めることができることが問題である．よって，攻撃者が知る情報を用いても，他の入力を求めることができないようにする必要がある．そのために攻撃者が知らない乱数を導入する．よって，攻撃者が知らない乱数を含む分散値が準備されているという条件を設定する．簡単のため，以下のように攻撃者が知らな

い秘密情報を 1 に限定した乱数 δ_j，η_j（$j=0,1,\ldots,k-1$）によって構成される乱数 δ，η を含む分散値集合が準備されていると仮定する．

$$[1]^{(\delta)}{}_i = (\overline{[\delta]}_i, \overline{[\delta_0]}_i, \ldots, \overline{[\delta_{k-1}]}_i)$$
$$[1]^{(\eta)}{}_i = (\overline{[\eta]}_i, \overline{[\eta_0]}_i, \ldots, \overline{[\eta_{k-1}]}_i)$$

これを 1 に対する分散値集合と呼ぶ．この 1 に対する分散値集合は，たとえば，以下のように容易に生成できる．ただし，下記処理 1，2 を実行する主体は 4.2 節において議論する．

【1 に対する分散値の生成】

1. k 個の乱数 $\delta_0,\delta_1,\ldots,\delta_{k-1}$ を生成し，乱数 $\delta=\prod_{j=0}^{k-1}\delta_j$ を計算する．

2. $\delta,\delta_0,\delta_1,\ldots,\delta_{k-1}$ を Shamir の (k,n) 閾値秘密分散法で n 台のサーバ S_i（$i=0,1,\ldots,n-1$）に分散させる．

3. サーバ S_i（$i=0,1,\ldots,n-1$）は 1 に関する分散情報として $[1]^{(\delta)}{}_i=(\overline{[\delta]}_i,\overline{[\delta_0]}_i,\ldots,\overline{[\delta_{k-1}]}_i)$ を保持する．

したがって，ここでは 2 つ目の条件として，以下を仮定する．

(2). 攻撃者に知られない乱数を用いた 1 に対する分散値集合が準備されている．

3.1 秘匿積和演算

各サーバは 2.6.1 項に示す分散処理によって演算に用いる秘密情報に関する分散値集合を持っているとする．秘密分散を含めてすべての演算は p を法として行う．

【記号定義】

- $\overline{[a]}_i$：値 a に対するサーバ S_i が保持する分散値．
- $[a]_i$：値 a に関連するサーバ S_i に保持する分散値集合．

【前提条件】

(1). 秘匿乗算において秘密情報と乱数に 0 を含まない．

(2). 攻撃者が知らない乱数（0 を除く）を用いた 1 に対する分散値集合が準備されている．ここでは，それを以下とする．

$$[1]^{(\delta)}{}_i=(\overline{[\delta]}_i,\overline{[\delta_0]}_i,\ldots,\overline{[\delta_{k-1}]}_i)\ (i=0,1,\ldots,n-1)$$
$$[1]^{(\eta)}{}_i=(\overline{[\eta]}_i,\overline{[\eta_0]}_i,\ldots,\overline{[\eta_{k-1}]}_i)\ (i=0,1,\ldots,n-1)$$

【秘匿積和演算】

入力：$[a]_j=(\overline{[\alpha a]}_j,\overline{[\alpha_0]}_j,\ldots,\overline{[\alpha_{k-1}]}_j)\ (j=0,1,\ldots,k-1)$
$[b]_j=(\overline{[\beta b]}_j,\overline{[\beta_0]}_j,\ldots,\overline{[\beta_{k-1}]}_j)\ (j=0,1,\ldots,k-1)$
$[c]_j=(\overline{[\lambda c]}_j,\overline{[\lambda_0]}_j,\ldots,\overline{[\lambda_{k-1}]}_j)\ (j=0,1,\ldots,k-1)$

出力：$[d]_i=[ab+c]_i=$
$(\overline{[\gamma(ab+c)]}_i,\overline{[\gamma_0]}_i,\ldots,\overline{[\gamma_{k-1}]}_i)\ (i=0,1,\ldots,k-1)$

1. サーバ S_0 は k 台のサーバより $\overline{[\alpha a]}_j$ を収集し，一時的に αa をスカラー量として復元する．

2. サーバ S_0 は全サーバ S_i に αa を送信する．

3. 全サーバ S_i は以下の式を用いて，αa と $\overline{[\beta b]}_i$ の乗算を行い，$\overline{[\alpha\beta ab]}_i$ を計算する．

$$\overline{[\alpha\beta ab]}_i=\alpha a\times\overline{[\beta b]}_i$$

4. サーバ S_0 は k 台のサーバより $\overline{[\alpha\beta ab]}_j$，$\overline{[\lambda c]}_j$ を収集し，一時的に $\alpha\beta ab$，λc のスカラー量を復元する．

5. サーバ S_0 は全サーバ S_i に $\alpha\beta ab$，λc を送信する．

6. 全サーバ S_i は以下の式を用いて，$\alpha\beta ab$ と $[1]^{(\delta)}{}_i$ 中の $\overline{[\delta]}_i$ との乗算を行い，$\overline{[\alpha\beta\delta ab]}_i$ を計算する．

$$\overline{[\alpha\beta\delta ab]}_i=\alpha\beta ab\times\overline{[\delta]}_i$$

7. 全サーバ S_i は以下の式を用いて，λc と $[1]^{(\eta)}{}_i$ 中の $\overline{[\eta]}_i$ との乗算を行い，$\overline{[\lambda\eta c]}_i$ を計算する．

$$\overline{[\lambda\eta c]}_i=\lambda c\times\overline{[\eta]}_i$$

8. k 台のサーバは各々 $\overline{[\alpha_j]}_l$，$\overline{[\beta_j]}_l$，$\overline{[\lambda_j]}_l$，$\overline{[\delta_j]}_l$，$\overline{[\eta_j]}_l$（$l=0,\ldots,k-1$）を収集し，α_j，β_j，λ_j，δ_j，η_j を復元し，乱数 γ_j を生成する．

9. k 台のサーバ S_j は各々 $\gamma_j/\alpha_j\beta_j\delta_j$，$\gamma_j/\lambda_j\eta_j$ を計算し，サーバ S_0 に送信する．

10. サーバ S_0 は $\gamma_j/\alpha_j\beta_j\delta_j$，$\gamma_j/\lambda_j\eta_j$ を用いて，以下の式より $\gamma/\alpha\beta\delta$，$\gamma/\lambda\eta$ を計算し，全サーバ S_i に送信する．

$$\frac{\gamma}{\alpha\beta\delta}=\prod_{j=0}^{k-1}\frac{\gamma_j}{\alpha_j\beta_j\delta_j}$$
$$\frac{\gamma}{\lambda\eta}=\prod_{j=0}^{k-1}\frac{\gamma_j}{\lambda_j\eta_j}$$

11. 全サーバ S_i は以下の式を用いて，$\overline{[\gamma(ab+c)]}_i$ を計算する．

$$\overline{[\gamma(ab+c)]}_i=\frac{\gamma}{\alpha\beta\delta}(\overline{[\alpha\beta\delta ab]}_i)+\frac{\gamma}{\lambda\eta}(\overline{[\lambda\eta c]}_i)$$

12. k 台のサーバ S_j は，秘匿乗算 ab のみであれば，$\gamma_j=\alpha_j\beta_j$ とし，秘匿加算を含む場合は $\gamma_j=\gamma_j$ として γ_j を Shamir の (k,n)-閾値秘密分散法で全サーバ S_i に分散させる．

13. サーバ S_i（$i=0,1,2,\ldots,n-1$）は秘密情報 $ab+c$ に関する分散情報として $[ab+c]_i=(\overline{[\gamma(ab+c)]}_i,\overline{[\gamma_0]}_i,\ldots,\overline{[\gamma_{k-1}]}_i)$ を保持する．

上記秘匿積和演算では，1.〜3. で TUS 方式と同様の秘匿乗算を行い，8.〜13. で TUS 方式に準じる秘匿加算を行う．よって，4.〜7. が 1 に対する分散値集合を用いた新たな処理になる．また，積和演算 $ab+c$ において $c=0$ とすれば秘匿乗算となり，$a=1$ とすれば秘匿加算となることが知られている．よって，上記秘匿積和演算においても $c=0$ とすれば秘匿乗算になり，4.〜7.，9.〜11. を省略できる．ただし，8. では α_j，β_j のみ復元し，γ_j を生成せず，

$\alpha_j\beta_j$ を分散させる．さらに，3. における乗算を αa による除算とし，12. の $\gamma_j = \alpha_j\beta_j$ を $\gamma_j = \beta_j/\alpha_j$ とすれば秘匿除算となる．また，$a = 1$（α も 1 とする）とすれば秘匿加算となり，1.〜3. を省略できる．ただし，4. 以降における $\overline{[\alpha\beta ab]}_j$ と $\alpha\beta ab$ は 3. において乗算を行っていないので $\overline{[\beta b]}_i$ と βb となり，6. における $\overline{[\alpha\beta\delta ab]}_i$ は $\overline{[\beta\delta b]}_i$ となり，8. では α_j の復元は行わず，9. 以降で $\alpha_j = \alpha = 1$ とする．また，加算を減算とすれば秘匿減算を実現できるので，上記アルゴリズムによって秘匿除算と秘匿減算を含むすべての四則演算と積和演算が実現できる．ただし，4.〜7. の処理の追加による安全性の向上については次節で議論する．

3.2 1 回の積和演算に関する安全性

TUS 方式では 2 入力演算であるので，**図 1** に示すように $[a]_j$，$[b]_j$ が入力され，$[c]_j$ が出力される 2 入力 1 出力の（秘匿演算と書かれた）ボックス内で，2.6.1 項に示す秘匿加減算および秘匿乗除算がボックス内で行われると考えることができる．TUS 方式では，ボックスから $k-1$ 台のサーバ情報を得ることができる攻撃者 1，$k-1$ 台のサーバ情報に加えて入力 $[a]_j$ または $[b]_j$ を知る攻撃者 2，$k-1$ 台のサーバ情報に加えて出力 $[c]_j$ を知る攻撃者 3 について，その安全性を示した．

提案方式は 3 入力 1 出力の積和演算に対するものであるため，**図 2** に示すように $[a]_j$，$[b]_j$，$[c]_j$ を入力とし，$[d]_j$ を出力とする 3 入力 1 出力のボックスで表せる．ただし，1 つの入力を既知（$a = 1$ または $c = 0$）とすれば，TUS 方式と同等な 2 入力 1 出力の秘匿演算（秘匿加減算と秘匿乗除算）を実現できる．

ここでは，以下のような攻撃者 4 と攻撃者 5 を定義し，各攻撃者が知ろうとする情報を得ることができれば，攻撃成功とする．

攻撃者 4：秘匿積和演算に関する 1 つの入力と出力を知る者が攻撃者となった場合であり，自らが入力した秘密情報および秘匿化秘密情報に用いられる乱数と復元時に得られる情報を知る．さらに，$k-1$ 台のサーバが知る情報も知ることができ，それらをもとに残り 2 つの入力情報を個々に知ろうとする．

攻撃者 5：秘匿積和演算に関する 2 つの入力を知る者が攻撃者となった場合であり，自らが入力した秘密情報および秘匿化秘密情報に用いられた乱数を知る．さらに，$k-1$ 台のサーバが知る情報も知ることができ，それらをもとに残り 1 つの入力情報および秘匿演算結果を知ろうとする．

ここで，攻撃者 4 の知る入力が定数などで既知であり，特に解読に利用できないとする場合，攻撃者 4 と攻撃者 3 は等価と考えられ，攻撃者 5 の知る入力の 1 つが既知である場合，攻撃者 5 は攻撃者 2 と等価であると考えられる．また，攻撃者 1 は攻撃者 4 または攻撃者 5 に含まれることは明らかである．また，3 入力 1 出力演算の場合，どのように安全な秘匿演算を用いても 2 入力と 1 出力が分かれば残りの 1 入力が漏洩し，3 入力が分かれば出力が漏洩するので，これ以上の情報を知る攻撃者を想定することは無意味である．よって，攻撃者 4 と攻撃者 5 に対して提案方式が安全であれば，提案方式は安全な積和演算を実現するといえる．

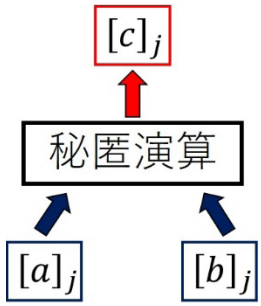

図 1 2 入力 1 出力の秘匿計算（TUS）
Fig. 1 2-input 1-output secrecy computation (TUS).

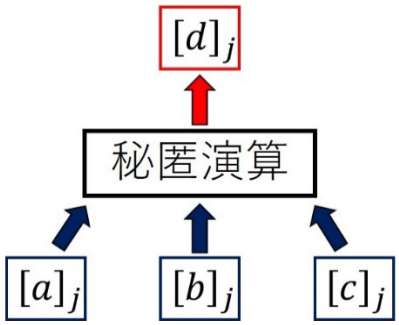

図 2 3 入力 1 出力の秘匿積和演算（提案方式）
Fig. 2 3-input 1-output secrecy computation (proposed method).

以下に，攻撃者 4 と攻撃者 5 に対する安全性を示す．

【攻撃者 4 に対する安全性】

b の入力を行う入力者が攻撃者である場合，彼は $k-1$ 台のサーバ情報とその出力も知る．よって，攻撃者 4 は入力時において b，β，β_i（$i = 0,\ldots,k-1$）を知り，1.〜2. において αa，4.〜5. において $\alpha\beta ab$，λc，8. において α_j，β_j，λ_j，δ_j，η_j，γ_j（$j = 0,1,\ldots,k-2$），10. において $\gamma/\alpha\beta\delta$，$\gamma/\lambda\eta$ を知り，復元時に γ，γ_i，$ab+c$ を知る．よって，攻撃者 4 に対する安全性は，b，β，αa，$\alpha\beta ab$，λc，$\gamma/\alpha\beta\delta$，$\gamma/\lambda\eta$，γ，β_i，γ_i（$i = 0,1,\ldots,k-1$），α_j，β_j，λ_j，δ_j，η_j，γ_j（$j = 0,1,\ldots,k-2$），$ab+c$ から a，c を求めることができるかという問題となる．

まず，問題を整理するために上記パラメータを，重複を避けた情報に分離する．その結果，b，β，αa，λc，γ，$\alpha\delta$，$\lambda\eta$，$ab+c$，β_i，α_j，λ_j，δ_j，η_j（$j = 0,1,\ldots,k-2$）から a，c を求めることができるかという問題に変形できる．

αa から a を知るためには α が分かればよいが，α と関連する情報は αa，$\alpha\delta$，α_j，δ_j（$j = 0,1,\ldots,k-2$）である

(b, β, c, λ, η は α, a と独立に定められる）ので，以下が成り立ち，a は漏洩しないことがいえる．

$$H(\alpha) = H(\alpha \mid \alpha_j \ (j = 0, 1, \ldots, k-2))$$
$$H(\delta) = H(\delta \mid \delta_j \ (j = 0, 1, \ldots, k-2))$$
$$H(a) = H(a \mid \alpha a, \alpha\delta, \alpha_j, \delta_j \ (j = 0, 1, \ldots, k-2))$$

TUS 方式では δ を乱数として持つ 1 の分散値情報が利用されていなかったので，$\alpha\delta$ が α となり，秘密情報 a が漏洩した．それに対して，提案方式では攻撃者が知らない乱数 δ を持つ 1 の分散値集合を導入したことによって秘密情報 a が漏洩しないことがいえる．

また，c を知るためには λ が分かればよいが，c に関しても同様に以下がいえる．

$$H(\lambda) = H(\lambda \mid \lambda_j \ (j = 0, 1, \ldots, k-2))$$
$$H(\eta) = H(\eta \mid \eta_j \ (j = 0, 1, \ldots, k-2))$$
$$H(c) = H(c \mid \lambda c, \lambda\eta, \lambda_j, \eta_j \ (j = 0, 1, \ldots, k-2))$$

これも攻撃者が知らない乱数 η を持つ 1 の分散値集合を導入したことにより，TUS 方式で漏洩した λ が漏洩せず秘密情報 c が守られていることが分かる．

最後に，攻撃者 4 は復元結果も知るため $d = ab + c$ と b を知るが，a，c のどちらが分からなければ残りを特定できない．

また，攻撃者が a の入力者，または c の入力者であっても同様の議論が成り立つ．よって，提案方式は攻撃者 4 に対して安全であり，残りの秘密情報を漏洩しないことがいえる．

【攻撃者 5 に対する安全性】

b, c の入力を行う入力者が攻撃者である場合，彼らは $k-1$ 台のサーバ情報も知る．よって，攻撃者は入力時において b, β, c, λ, β_i, λ_i $(i = 0, \ldots, k-1)$ を知り，1.〜2. において αa，4.〜5. において $\alpha\beta ab$, λc, 8. において α_j, β_j, λ_j, δ_j, η_j, γ_j $(j = 0, 1, \ldots, k-2)$，10. において $\gamma/\alpha\beta\delta$, $\gamma/\lambda\eta$ を知る．よって，攻撃者 5 に対する安全性は，b, β, c, λ, αa, $\alpha\beta ab$, λc, $\gamma/\alpha\beta\delta$, $\gamma/\lambda\eta$, β_i, λ_i $(i = 0, 1, \ldots, k-1)$, α_j, β_j, λ_j, δ_j, η_j, γ_j $(j = 0, 1, \ldots, k-2)$ から残りの入力 a または出力 $ab + c$ を求めることができるかという問題となる．

まず，問題を整理するために上記パラメータを，重複を避けた情報に分離する．その結果，b, β, c, λ, αa, $\gamma/\alpha\delta$, γ/η, α_j, δ_j, η_j, γ_j, $\gamma(ab + c)_j$ $(j = 0, 1, \ldots, k-2)$ が得られる．

αa から a を知るためには α が分かればよいが，α と関連する情報は αa, $\gamma/\alpha\delta$, α_j, δ_j, γ_j $(j = 0, 1, \ldots, k-2)$ であるので，以下が成り立ち，a は漏洩しないことがいえる．

$$H(\alpha) = H(\alpha \mid \alpha_j \ (j = 0, 1, \ldots, k-2))$$
$$H(\gamma) = H(\gamma \mid \gamma_j \ (j = 0, 1, \ldots, k-2))$$
$$H(\delta) = H(\delta \mid \delta_j \ (j = 0, 1, \ldots, k-2))$$
$$H(a) = H(a \mid \gamma/\alpha\delta, \alpha_j, \delta_j, \gamma_j \ (j = 0, 1, \ldots, k-2))$$

TUS 方式では δ, η を乱数として持つ 1 の分散値情報が利用されていなかったので，$\gamma/\lambda\eta$ が γ/λ となり，$\gamma/\alpha\delta$ が γ/α となり，γ/λ と λ の情報より乱数 γ が漏洩し，γ/α, γ より乱数 α が漏洩するので，最後に α と αa より秘密情報 a が漏洩する．漏洩した a と攻撃者が知る b, c より秘匿演算結果 $ab + c$ も漏洩する．それに対して，提案方式では攻撃者が知らない乱数 δ, η を持つ 1 の分散値集合を導入したことによって秘密情報 a が漏洩しないことがいえる．

また，攻撃者は $\gamma(ab + c)_j$ の情報を知るが，それより $\gamma(ab + c)$ が漏洩しないことがいえる．よって，以下がいえる．

$$H(\gamma(ab + c)) = H(\gamma(ab + c) \mid \gamma(ab + c)_j \ (j = 0, 1, \ldots, k-2))$$

さらに，秘匿演算結果を秘匿化する乱数 γ が漏洩するかについて考える．γ と関連する情報は $\gamma/\alpha\delta$, γ/η, α_j, δ_j, γ_j, η_j $(j = 0, 1, \ldots, k-2)$ であるので，以下が成り立ち，γ は漏洩しないことがいえる．

$$H(\gamma) = H(\gamma \mid \gamma_j \ (j = 0, 1, \ldots, k-2))$$
$$H(\alpha) = H(\alpha \mid \alpha_j \ (j = 0, 1, \ldots, k-2))$$
$$H(\delta) = H(\delta \mid \delta_j \ (j = 0, 1, \ldots, k-2))$$
$$H(\eta) = H(\eta \mid \eta_j \ (j = 0, 1, \ldots, k-2))$$
$$H(\gamma) = H(\gamma \mid \gamma/\alpha\delta, \gamma/\eta, \alpha_j, \delta_j, \gamma_j, \eta_j \ (j = 0, 1, \ldots, k-2))$$

TUS 方式では η を乱数として持つ 1 の分散値情報が利用されていなかったので，$\gamma/\lambda\eta$ が γ/λ となり，γ/λ と λ の情報より乱数 γ が漏洩した．それに対して，提案方式では攻撃者が知らない乱数 η を持つ 1 の分散情報を導入したことによって乱数 γ が漏洩しないことがいえる．

最後に，攻撃者 5 は入力情報 b, c を知るが，$ab + c$ が分からなければ残りを特定できない．よって，以下がいえる．

$$H(a) = H(a \mid b, \beta, c, \lambda, \alpha a, \gamma/\alpha\delta, \gamma/\eta, \alpha_j, \delta_j, \eta_j, \gamma_j, \gamma(ab + c)_j)$$
$$H(ab + c) = H(ab + c \mid b, \beta, c, \lambda, \alpha a, \gamma/\alpha\delta, \gamma/\eta, \alpha_j, \delta_j, \eta_j, \gamma_j, \gamma(ab + c)_j)$$

また，攻撃者が a，c の入力者，または a，b の入力者であっても同様の議論が成り立つ．以上より，提案方式は攻撃者 5 に対して安全であり，残りの秘密情報および演算結果を漏洩しないことがいえる．

3.3 積和演算の組合せとその安全性

一般に，四則演算からなる任意の演算は積和演算

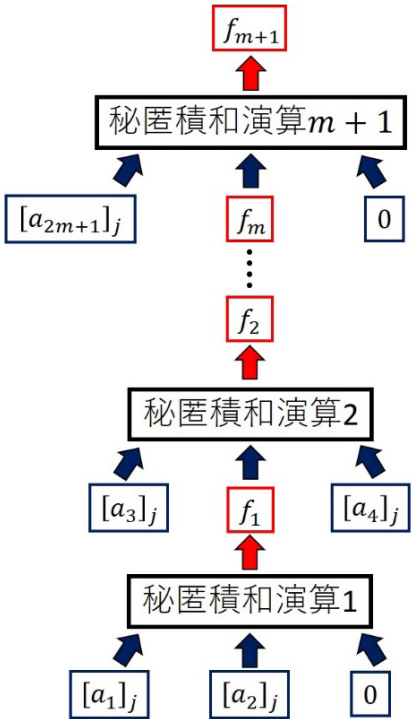

図 3 積和演算の組合せからなる a の演算

Fig. 3 Computation of a from combination of multiple product-sum operation.

$f(a,b,c)=d$ の組合せに分解できる．

たとえば，$a=f(a_1,a_2,\ldots,a_{2m},a_{2m+1})=a_{2m+1}(a_1a_2+a_3a_4+\cdots+a_{2m-1}a_{2m})$ は以下のように積和に分解される．

積和 1：$f_1=f(a_1,a_2,0)=a_1a_2$

積和 2：$f_2=f(a_3,a_4,f_1)=a_3a_4+a_1a_2$

⋮

積和 m：$f_m=f(a_{2m-1},a_{2m},f_{m-1})=a_1a_2+a_3a_4+\cdots+a_{2m-1}a_{2m}$

積和 $m+1$：$f_{m+1}=f(a_{2m+1},f_m,0)=a_{2m+1}(a_1a_2+a_3a_4+\cdots+a_{2m-1}a_{2m})=a$

これを図 2 に示すボックスを組み合わせて表せば，**図 3** のようになる．ただし，途中結果は復元されないので，ボックス間の接続においてその出力は復元されずそのまま入力される．また，各ボックスの演算は同一のサーバセットによって実行される．一方，a の演算は一般的に**図 4** のように表すこともできるが，a を前記のように積和演算に分解すれば，図 4 の中身が図 3 であるということもできる．

ここで，3.1 節に示した秘匿積和演算の組合せについて考える．たとえば，図 4 のボックスにどんなに安全な秘匿演算手法を用いていても，a_1 以外の入力と出力が知られている場合，$a_1=f^{-1}(a,a_2,\ldots,a_{2m},a_{2m+1})$ により a_1 も漏洩する．

それに対して，a の演算において 2 つの入力（たとえば，a_1, a_2）が漏洩していなければ，その 2 入力は個々に漏洩しないことがいえる．それは，$a_1=f^{-1}(a,a_2,\ldots,a_{2m},a_{2m+1})$ となるため，a_2 が分からなければ a_1 を特定できず，その逆もいえるためである．

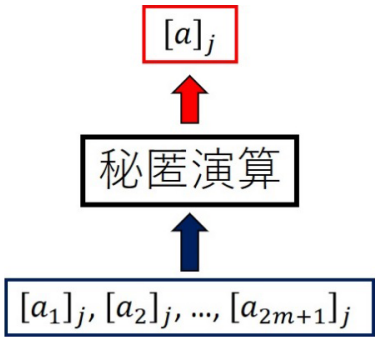

図 4 a の演算に関する一般的な積和演算ボックス

Fig. 4 Computation of a represented as a single product-sum operation.

よって，以下の攻撃者を定義する．

攻撃者 6：t 入力 1 出力演算に関して，$t-1$ 個以下の入出力を知る者が攻撃者となった場合である．残り 2 つの入力または入出力のみが未知数であり，自らが知る情報と $k-1$ 台のサーバが知る情報から，残り 2 つの未知数を個々に知ろうとする．

攻撃者 6 に対する安全性を以下に示す．ただし，積和演算の連続を 1 つのサーバセットで実行するため，以下の条件を加える．

(3). 演算の連続において，各サーバが扱う分散値集合内の分散値の位置は固定される．

Shamir 法による秘匿演算では，秘匿積和演算が定義され，それを図 2 のようなボックスで表し，秘匿演算の連続を図 3 のように表すとき，各ボックスは独立になる．すなわち，ある秘匿演算を実行するサーバの組は次の演算では前の組にとらわれず，自由にサーバの組合せを変えることができる．それに対して，提案方式では秘匿演算に参加したサーバは 1 つの演算で秘密情報の断片を知る．その後，連続する演算で前の演算と異なる乱数を扱えば，そのサーバは 2 つの乱数に関する情報を得ることができるため，$k-1$ 台のサーバから k 台相当のサーバ情報が漏洩する可能性がある．よって，提案方式では条件 (3) によって，演算が繰り返されても 1 つのサーバが扱う乱数は前の演算で扱った乱数に限定する．すなわち，秘匿乗算において α_j と β_j を復元し，$\alpha_j\beta_j$ を分散したサーバは次の演算においても，$\alpha_j\beta_j$ を復元するサーバとなる．

ただし，$n>k$ でサーバが故障した場合などでは，たとえば，α_j，β_j を扱ったサーバ S_j が故障すれば今まで秘匿演算に参加していない他のサーバが S_j に代わって秘匿演算に参加する．ここで，n 台のサーバ中情報漏洩するサーバは $k-1$ 台以下という前提が秘密分散には存在する．よって，ある演算において秘匿演算に参加しているサーバ中，$k-1$ 台が攻撃者に情報漏洩する不正サーバとするなら，秘匿演算に参加している 1 台のサーバに代わって秘匿演算

に参加する新たなサーバは不正サーバではないことがいえる．よって，条件 (3) を追加しても新たな問題は発生しない．また，各サーバは秘密情報の断片を知るがそれだけであり，それが k 個集められることはないため安全である．

以下に，上記条件の下，攻撃者 6 に対する安全性を積和演算の組合せを用いて示す．

【攻撃者 6 に対する安全性】

攻撃者 6 に対して，以下の 3 つの場合に分けて考える．

1. 1 つの積和演算ボックスの 3 入力中の 2 入力が未知であり，残りの入力と最終出力が漏洩している場合．

1 つの積和演算ボックスの 2 入力が未知である，すなわちボックスの 1 入力と 1 出力を知る攻撃者は攻撃者 4 と同じである．提案方式は攻撃者 4 に対して，残りの 2 入力を秘密に保つことがいえる．

次に，このボックスが，たとえば，図 3 の秘匿積和演算 2 のボックスとして，残りのボックスの全入力と最終出力が漏洩しているとする．この場合，秘匿積和演算 2 のボックス以外の情報は漏洩するが，秘匿積和演算 2 のボックスは攻撃者 4 に対する安全性から，残り 2 入力は漏洩しない．また，これはボックスの位置に依存しない．

2. 未知の 2 入力が各々異なるボックスに入力され，残りの入力と最終出力が漏洩している場合．

未知の 2 入力が異なるボックスに入力されると，そのボックスの 3 入力のうち 2 入力が漏洩する．たとえば，秘匿化された 2 入力がそれぞれ図 3 のボックス 1 およびボックス 2 に入力され，それ以外の入力が漏洩しているとする．ボックス 1 では 2 入力が分かっても 1 入力不明であればその出力は分からず，ボックス 2 において漏洩しない 1 入力が加わる．よって，ボックス 2 でも 1 つのボックス中 2 入力が漏洩していないため，前記と同様に攻撃者 4 の安全性からその 2 入力は漏洩しないことがいえる．これもボックスの組合せに依存しない．

3. 最終出力と 1 入力が未知であり，残りの全入力が漏洩している場合．

未知の入力が入力されるボックスでは，前述のようにその出力も不明になる．よって，その出力につながる入力も不明となるため，最終ボックスにおいて 1 入力と 1 出力が不明となる．この場合，最終ボックスでは 2 入力が知られるが，攻撃者 5 の安全性によりその 1 入力と 1 出力は知られないことがいえる．

4. 従来方式との比較および考察

4.1 従来方式との比較

秘密分散を用いた秘匿計算の方式として，Damgård らの方式 [13]，Gennaro らの方式 [16]，Nishide らの方式 [22]，SPDZ 方式 [14] および SPDZ 2 方式 [15] がある．ただし，SPDZ 2 方式は SPDZ 方式の改良版であるので，これ以降は，Damgård らの方式，Gennaro らの方式，Nishide らの方式および SPDZ 2 方式のみを比較対象とする．

SPDZ 2 方式は $n = k$ の設定に限定されており，自身がプレイヤの 1 人で，n 人のプレイヤ中自分以外の $n-1$ 人までの結託に対して安全な方式となっている．特に，SPDZ 2 方式は active な攻撃者を想定する dishonest majority を達成している．また，SPDZ 2 方式は事前計算フェーズ（pre-processing phase）とオンラインフェーズ（online phase）から構成され，事前計算フェーズでは準同型暗号で処理を行うため，SPDZ 2 方式の安全性は計算量的安全性となり，事前処理に関する処理量は大きい．

Damgård らの方式は $n \geq 2k-1$ の設定に限定されており，active な攻撃者に対する安全性を持つ．また，Damgård ら方式は秘匿加減算と秘匿乗算のような秘匿計算の組合せに対して安全である．

また，Gennaro らの方式と Nishide らの方式は $n \geq 2k-1$ の設定に限定されており，passive な攻撃者を想定する情報理論的安全性を達成している．

それに対して，提案方式は $n = k$ の設定に限定されず，$k \leq n$ の設定に対して有効である（提案方式は $n < 2k-1$ に対してその有効性を発揮するが，$n \geq 2k-1$ に対しても適用できる）．ただし，提案方式は Gennaro らの方式や Nishide らの方式と同様に，攻撃者はプロトコルに従い，漏洩した情報を基に秘密情報の解析などを行う passive な攻撃者を想定しており，それに対しては情報理論的安全性を実現する．また，提案方式は異なる秘匿演算を組み合わせても情報理論的安全性を達成する．ただし，提案方式には 3 つの条件が必要である．また，提案方式は 3.1 節に示すように a と b の分散値から秘匿除算を直接実行できる．一般に，除算は除数の逆元をべき乗演算などにより計算した後，秘匿乗算を行うが，提案方式は αa を復元した後，べき乗演算ではなく逆元を変換表などによって求めることができ，効率的である．以上の比較を，**表 1** にまとめて示す．

表 1 提案方式と従来方式との比較
Table 1 Comparison of proposed method and conventional methods.

	提案方式	SPDZ 2	Damgård らの方式	Gennaro らの方式	Nishide らの方式
n と k の設定	$n \geq k$	$n = k$	$n \geq 2k-1$	$n \geq 2k-1$	$n \geq 2k-1$
想定する攻撃者	Passive	Active	Active	Passive	Passive
実現する安全性	情報理論的安全性	計算量的安全性	情報理論的安全性	情報理論的安全性	情報理論的安全性
必要な条件	3	0	0	0	0

表 2 提案方式と従来方式の比較
Table 2 Comparison of proposed method and conventional methods.

		提案方式	SPDZ 2	Damgård らの方式	Gennaro らの方式	Nishide らの方式
シェア容量 1		$3(k+1)nd_1$	$3nd_1$	$3nd_1$	$3nd_1$	$3nd_1$
シェア容量 2		$2(k+1)nd_1$	$3nd_1$	$2nd_1$	−	−
計算量	事前処理	$2(k+1)C_1$	$3C_1+2C_2$	$2C_1$	−	−
	分散処理	$3(k+1)C_1$	$3C_1$	$3C_1$	$3C_1$	$3C_1$
	復元処理	$(k+1)C_1$	C_1	C_1	C_1	C_1
	秘匿乗算 $a\times b$	$(3k+1)C_1$	$2nC_1$	$2nC_1$	$(2k-1)nC_1$	$(2k-1)nC_1$
	秘匿加算 $a+b$	$(5k+2)C_1$	nA	nA	nA	nA
	秘匿積和演算 $ab+c$	$(6k+3)C_1$	$2nC_1$	$2nC_1$	$(2k-1)nC_1$	$(2k-1)nC_1$
通信量	事前処理	$2(k+1)nd_1$	$3nd_1+2nd_2$	$2nd_1$	−	−
	分散処理	$3(k+1)nd_1$	$3nd_1$	$3nd_1$	$3nd_1$	$3nd_1$
	復元処理	$(k+1)kd_1$	kd_1	kd_1	$(2k-1)d_1$	$(2k-1)d_1$
	秘匿乗算 $a\times b$	$(2k^2+k+nk+n)d_1$	$(2n^2-2n)d_1$	$(2k-1+n)d_1$	$(2k-1)n^2d_1$	$(2k-1)n^2d_1$
	秘匿加算 $a+b$	$(4k^2+4k+nk+4n)d_1$	0	0	0	0
	秘匿積和演算 $ab+c$	$(5k^2+5k+nk+5n)d_1$	$(2n^2-2n)d_1$	$(2k-1+n)d_1$	$(2k-1)n^2d_1$	$(2k-1)n^2d_1$
ラウンド数	事前処理	1	2	1	−	−
	分散処理	1	1	1	1	1
	復元処理	1	1	1	1	1
	秘匿乗算 $a\times b$	4	1	2	1	1
	秘匿加算 $a+b$	6	0	0	0	0
	秘匿積和演算 $ab+c$	8	1	2	1	1

また，提案方式と SPDZ 2 方式，Damgård らの方式，Gennaro らの方式，Nishide らの方式との計算量や通信量およびシェア容量に関する比較を**表 2** 以降に示す．ただし，通信回数は通信の方向に応じてラウンド数という観点で評価する．また，提案方式はサーバで行う処理が異なる場合（S_0 は $\overline{[\alpha a]}_j$ を集めて αa を復元するなど）があることから，表 2 以降は全体の計算量や通信量およびシェア容量の評価となる．ただし，分散は 3 人の入力者，復元処理は 1 人の復元者が行う処理である．以下に使われる記号の定義を示す．

【記号定義】

- d_1：秘密分散におけるシェアのサイズ
- d_2：Somewhat-準同型暗号（SHE）におけるデータサイズ
- C_1：秘密分散に関する計算量
- C_2：Somewhat-準同型暗号（SHE）に関する計算量
- A：1 回の加算に関する計算量

ただし，d_1 は秘密情報のサイズとほぼ同程度である．それに対して，d_2 は一般に秘密情報より大きい．よって $d_2 > d_1$ とする．さらに，C_1 は C_2 に比べて十分小さいと考えられる．よって，$C_1 \ll C_2$ とする．ただし，秘密分散において分散処理と復元処理に必要な計算量は異なるが，秘密分散の分散処理および復元処理は一般に C_2 より十分小さいので簡単のため秘密分散の分散処理および復元処理を C_1 で表す．ただし，A は 1 回の加算に関する計算量であり，$C_1, C_2 \gg A$ のため，C_1 または C_2 を含む場合 A の

計算量を無視する．また，ゼロ知識証明や MAC などの検証に関する処理量の評価も提案方式が検証処理を含まないため比較を省略する．なお，従来方式および提案方式の中には事前処理を必要とするものがあるため，それを分けて評価する．提案手法における事前処理では，1 回の秘匿加算または秘匿積和演算を実現するために必要となる情報（2 組の 1 に対する分散値集合）を生成する．SPDZ 2 方式および Damgård らの方式における事前処理では，1 回の秘匿乗算または秘匿積和演算を実現するために必要となる情報（SPDZ 2 方式では分散情報 $\langle a\rangle$，$\langle b\rangle$，$\langle c\rangle$，Damgård らの方式では分散情報 $[r]_d$，$[r]_{2d}$）を生成する．よって，表 2 のシェア容量 1 は，1 回の秘匿積和演算を実現するために必要となる秘密情報 a，b，c のシェア容量を示し，シェア容量 2 は事前処理によって生成した情報（SPDZ 2 方式では分散情報 $\langle a\rangle$，$\langle b\rangle$，$\langle c\rangle$，Damgård らの方式では分散情報 $[r]_d$，$[r]_{2d}$，提案方式では 2 組の 1 の分散値集合）に関する容量を示す．また，分散処理の計算量および通信量は秘密情報 a，b，c を分散させるために必要となる計算量および通信量を示す．また，事前処理の計算量および通信量には前述の情報を生成するための計算量および通信量を示す．秘匿乗算，秘匿加算および秘匿積和演算の計算量および通信量はそれぞれの秘匿演算を実現するために必要となる全体の計算量および通信量を示す．また，Gennaro らの方式と Nishide らの方式における復元処理の通信量は秘匿乗算および秘匿積和演算の演算結果を復元するための通信量を示す．

表 2 より，計算量において，提案方式の計算量は Somewhat-準同型暗号（SHE）の計算量 C_2 を含まないため，SPDZ 2 方式より優れると考えられる．さらに，提案方式では $n < 2k-1$ の設定でも秘匿計算を安全に実現できるが，そのトレードオフとして，Damgård らの方式，Gennaro らの方式および Nishide らの方式より計算量が多いことが分かる．通信量に関しては，n，k，d_1，d_2 の関係によって優劣が異なると考えられる．最後に，ラウンド数から比較すると，提案方式では秘匿演算処理において乱数の生成・分散および演算中に乱数の復元・分散のために，ラウンド数が SPDZ 2 方式より多い．

また，シェア容量において，提案方式では 1 つの秘密情報に対して，1 個の秘匿化した情報と k 個の乱数を生成するので，$(k+1)d_1$ のシェアサイズが必要となる．一方，SPDZ 2 方式，Damgård ら方式，Gennaro らの方式および Nishide らの方式では，1 つの秘密情報に対して d_1 のシェアサイズが必要となる．

以上より，提案方式は Damgård ら方式，Gennaro らの方式および Nishide らの方式でできない $n < 2k-1$ 上の秘匿演算を実現できるが，それらの方式に比べて全般的に計算量や通信量・ラウンド数などが大きい．

一方，$n = k$ の設定で秘匿演算を実現する SPDZ 2 方式に対して，提案方式の計算量は SHE の処理を行わないため，SPDZ 2 方式の計算量より軽いことがいえる．それを検証するために，C_2 が C_1 に比べて十分大きいことを提案方式の実装による処理時間の比較によって示す．ただし，SPDZ 2 方式は文献 [15] にある値を用いるため設定や実装環境が異なるが，上記評価の参考とできる．SPDZ 2 方式の分散処理では，CPU が 2.80 GHz の Intel Core i7 で，メモリが 4 GB の計算環境における $n = 2$，$p = 2^{32}$ の設定で，SHE の処理を含んで 1 個の multiplication triple を生成するために必要な平均時間が 19.55 ms であった [15]．それに対して，提案方式の分散処理では，前記のように d_1 は d_2 より小さくできるため，$n = 2$，$p = 2^9$ の設定で，CPU が Intel Core i7-7700 で，メモリが 32 GB という市販のノート PC を用いて 1 個の 1 に対する分散値集合を生成するために必要な時間は 6.240 μs であった．これによって，C_1 は C_2 に比べてはるかに速いと考えられる．提案方式はいくつかの乗算や加算処理を除けば，基本的に秘密分散処理で構成される．これにより，表 2 における提案方式と SPDZ 2 方式の計算量については，C_2 を含まない提案方式が優れるといえる．ただし，提案方式のラウンド数および通信量は，表 2 より SPDZ 2 方式より多い．

また，SPDZ2 方式は $n = k$ 以外の $n < 2k-1$ の設定では秘匿演算できない．よって，提案方式は $n < 2k-1$ において前記 3 条件の下で汎用的に秘匿演算できる最速の手法ということができる．

条件を同じにして比較するために，提案方式の k と SPDZ 2 以外の従来方式の $2k-1$ が等しいとする場合を**表 3**，**表 4** に示す．提案方式の k と SPDZ 2 以外の従来方式の $2k-1$ が等しいとする場合，必要なサーバ数を同じとできるので，各々の n に対する k の値を表 3 に示す．表 3 から分かるように，SPDZ 2 以外の従来方式の k は提案方式の k の約半分になり安全性が低下する．表 4 に $n = 5$ を同じとした場合に対する各方式の不正なサーバ数の最大数を具体値として求めた全体の計算量および通信量を示す．表 4 より，

表 3 従来方式との比較（サーバ数 n を同じにした場合）
Table 3 Comparison with conventional methods (number of server n is made the same).

	提案方式	Damgård らの方式	Gennaro らの方式	Nishide らの方式
$n = 2$ の場合	$k \le 2$	$k \le 3/2$	$k \le 3/2$	$k \le 3/2$
$n = 3$ の場合	$k \le 3$	$k \le 2$	$k \le 2$	$k \le 2$
$n = 4$ の場合	$k \le 4$	$k \le 5/2$	$k \le 5/2$	$k \le 5/2$
$n = 5$ の場合	$k \le 5$	$k \le 3$	$k \le 3$	$k \le 3$

表 4 提案方式と従来方式の比較（サーバ数 $n = 5$ を同じにした場合）

Table 4 Comparison of proposed method and conventional methods (number of server $n = 5$).

		提案方式	Damgård らの方式	Gennaro らの方式	Nishide らの方式
不正なサーバ数の最大数		$k = 5$	$k = 3$	$k = 3$	$k = 3$
シェア容量 1		$90d_1$	$15d_1$	$15d_1$	$15d_1$
シェア容量 2		$60d_1$	$10d_1$	−	−
計算量	事前処理	$12C_1$	$2C_1$	−	−
	分散処理	$18C_1$	$3C_1$	$3C_1$	$3C_1$
	復元処理	$6C_1$	C_1	C_1	C_1
	秘匿乗算 $a \times b$	$16C_1$	$10C_1$	$25C_1$	$25C_1$
	秘匿加算 $a + b$	$27C_1$	$5A$	$5A$	$5A$
	秘匿積和演算 $ab + c$	$33C_1$	$10C_1$	$25C_1$	$25C_1$
通信量	事前処理	$60d_1$	$10d_1$	−	−
	分散処理	$90d_1$	$15d_1$	$15d_1$	$15d_1$
	復元処理	$30d_1$	$3d_1$	$5d_1$	$5d_1$
	秘匿乗算 $a \times b$	$85d_1$	$10d_1$	$125d_1$	$125d_1$
	秘匿加算 $a + b$	$165d_1$	0	0	0
	秘匿積和演算 $ab + c$	$200d_1$	$10d_1$	$125d_1$	$125d_1$

n を同じとした場合，全体の計算量および通信量は従来方式のほうが小さいことが分かる．

次に，結託するサーバ数が等しいとする場合について，**表 5** に同じ k に対する必要なサーバ数を示し，**表 6** にその場合の全体の計算量と通信量を示す．なお，表 6 の計算量および通信量は，同じ $k = 5$ に対する各方式が必要なサーバ数の最小数を用いた場合の全体の計算量と通信量である．表 5 より，従来方式は $n < 2k - 1$ を実現できないため，提案方式の約 2 倍の数のサーバ数が必要となることが分かる．表 6 より，事前処理，分散処理，復元処理および秘匿加算の計算量や通信量は，サーバ数が多くても従来方式の方が小さいことが分かる．ただし，提案方式の秘匿乗算と秘匿積和演算の計算量および通信量は，Gennaro らの方式および Nishide らの方式より小さい．

また，サーバ数 n が大きい場合，その設置場所や維持費などの実用的な観点からの問題が発生する．一般に，秘密分

表 5 従来方式との比較（結託するサーバ数 k を同じとした場合）

Table 5 Comparison with conventional methods (number of adversary k is made the same).

	提案方式	Damgård らの方式	Gennaro らの方式	Nishide らの方式
$k = 2$ の場合	$n \geq 2$	$n \geq 3$	$n \geq 3$	$n \geq 3$
$k = 3$ の場合	$n \geq 3$	$n \geq 5$	$n \geq 5$	$n \geq 5$
$k = 4$ の場合	$n \geq 4$	$n \geq 7$	$n \geq 7$	$n \geq 7$
$k = 5$ の場合	$n \geq 5$	$n \geq 9$	$n \geq 9$	$n \geq 9$

表 6 提案方式と従来方式の比較（結託するサーバ数 $k = 5$ を同じとした場合）

Table 6 Comparison of proposed method and conventional methods (number of adversary $k = 5$).

		提案方式	Damgård らの方式	Gennaro らの方式	Nishide らの方式
必要なサーバ数の最小数		$n = 5$	$n = 9$	$n = 9$	$n = 9$
シェア容量 1		$90d_1$	$27d_1$	$27d_1$	$27d_1$
シェア容量 2		$60d_1$	$18d_1$	−	−
計算量	事前処理	$12C_1$	$2C_1$	−	−
	分散処理	$18C_1$	$3C_1$	$3C_1$	$3C_1$
	復元処理	$6C_1$	C_1	C_1	C_1
	秘匿乗算 $a \times b$	$16C_1$	$18C_1$	$81C_1$	$81C_1$
	秘匿加算 $a + b$	$27C_1$	$9A$	$9A$	$9A$
	秘匿積和演算 $ab + c$	$33C_1$	$18C_1$	$81C_1$	$81C_1$
通信量	事前処理	$60d_1$	$18d_1$	−	−
	分散処理	$90d_1$	$27d_1$	$27d_1$	$27d_1$
	復元処理	$30d_1$	$5d_1$	$9d_1$	$9d_1$
	秘匿乗算 $a \times b$	$85d_1$	$18d_1$	$729d_1$	$729d_1$
	秘匿加算 $a + b$	$165d_1$	0	0	0
	秘匿積和演算 $ab + c$	$200d_1$	$18d_1$	$729d_1$	$729d_1$

散によるデータ保管に対して，安全性と故障などに対する可用性から k と n が定まっていても，従来方式では秘匿計算を行う場合，$n \geqq 2k-1$ という条件から n を増加させる必要がある．それに対して，提案方式は秘匿計算を行っても k と n の関係を変える必要がない．よって，提案方式は実用性という観点からも最も優れた方式ということができる．

4.2 考察

以下にその 3 つの条件の実現性について検討する．

(1). 秘匿乗算において秘密情報に 0 を含まない．

秘匿乗算において秘密情報が 0 であれば，秘匿乗算において $\alpha a = 0$ となり，秘密情報が 0 であることが漏洩する（乱数は 0 を含まない）．ただし，0 を含まない情報は医療データなど多く存在する．たとえば，脈拍や血圧などはすべて正の値であり，0 は死亡していることを意味しており，医療的な統計計算に用いられない．また血糖値なども正の値である．よって，病院に入院中または治療中の患者などの情報を秘匿統計計算する場合，0 を含まないことは問題とならない場合が多い．また，秘匿加減算および秘匿除算においては秘密情報に 0 を含んでも問題ない．よって，この条件の回避は今後の課題であるが，0 を含まない情報は多数存在し，提案方式はそのような情報に対する秘匿演算法としては有効である．

(2). 攻撃者が知らない乱数を用いた 1 に対する分散値集合がある．

この条件を満たすために最も簡単な方法として，信頼できる第三者（サーバ）が 1 に対する分散値集合を提供することが考えられる．秘密分散に信頼できるサーバを仮定する手法としては，文献 [1], [18] などがあり，それによって処理の効率化などを実現する．よって，信頼できるサーバの設置は実用的には有効である．しかし，1 に対する分散値集合は 3 章の冒頭に示した処理から分かるように，秘密情報に依存せず，k 個の乱数の生成とその乗算，および秘密分散処理によって容易に実現できる．また，本論文では情報漏洩はしても処理に従う Passive Adversary を仮定している．よって，全サーバに異なる乱数を持つ 1 の分散値集合を生成させる処理を追加し，文献 [27] に示すシャッフル処理を実行すれば，自分が作った分散値集合への対応がとれなくなる．さらに，提案方式を実行する複数のサーバセットがあり，互いに利害関係がなければ，それらのサーバ間で 1 に対する分散値集合を交換・混合・削除しながら，シャッフルすれば信頼できる第三者に近い仕組みとなると考えられる．よって，この条件の回避も今後の課題であるが，種々の実現方法が想定され，実用的な問題はないと考えられる．

(3). 演算の連続において各サーバが扱う分散値集合内の分散値の位置は固定される．

提案方式は決められたプロトコルに従う Passive Adversary を仮定しているため，秘匿演算を実行するサーバに対して，これを規則として定めればよい．

5. まとめ

本論文では，以下の 3 つの前提条件のもとで，$2k-1 > n$ における演算の連続に対して安全な秘密分散を用いた秘匿計算を実現できた．

(1). 秘匿乗算において秘密情報と乱数に 0 を含まない．

(2). 攻撃者に知られない乱数を用いた 1 に対する分散値集合がある．

(3). 演算の連続において各サーバが扱う分散値集合内の分散値の位置は固定される．

今後は上記 3 条件を撤廃または緩和できる方式の検討が課題である．

参考文献

[1] Beaver, D.: Commodity-based cryptography, *Proc. 29th Annual ACM Symposium on Theory of Computing* (*STOC '97*), pp.445–455, ACM (1997).

[2] Ben-Or, M., Goldwasser, S. and Wigderson A.: Completeness Theorems for Non-Cryptographic Fault-Tolerent Distributed Computation, *Proc. 20th Annual ACM Symposium on Theory of Computing* (*STOC '88*), pp.1–10, ACM (1988).

[3] Bendlin, R., Damgård, I., Orlandi, C. and Zakarias, S.: Semi-homomorphic Encryption and Multiparty Computation, *Advances in Cryptology-EUROCRYPT 2011*, Paterson, K.G. (Ed.) Lecture Notes in Computer Science, Vol.6632, pp.169–188, Soringer, Berlin, Heidelberg (2011).

[4] Blakley, G.R.: Safeguarding Cryptographic Keys, *Proc. AFIPS 1979 National Computer Conference*, Vol.48, pp.313–317 (1979).

[5] Brakerski, Z., Gentry, C. and Vaikuntanathan, V.: Leveled Fully Homomorphic Encryption without Bootstrapping, *ITCS 2012*, Mitzenmacher, M. (Ed.), pp.309–325, Cambridge, MA, USA (Jan. 2009).

[6] Brakerski, Z. and Vaikuntanathan, V.: Fully Homomorphic Encryption from Ring-LWE and Security for Key Dependent Messages, *Advances in Cryptology – CRYPTO 2011*, Rogaway, P. (Eds.), Lecture Notes in Computer Science, Vol.6841, Springer, Berlin, Heidelberg (2011).

[7] Chaum, D., Crépeau, C. and Damgård, I.: Multiparty Unconditionally Secure Protocols, *Proc. 20th Annual ACM Symposium on Theory of Computing* (*STOC '88*), pp.11–19, ACM (1988).

[8] 千田浩司，五十嵐大，濱田浩気，高橋克巳：エラー検出可能な軽量 3 パーティ秘匿関数計算の提案と実装評価，情報処理学会論文誌，Vol.52, No.9, pp.2674–2685 (2011).

[9] 千田浩司，五十嵐大，濱田浩気，菊池　亮，冨士　仁，高橋克巳：マルチパーティ計算に適用可能な計算量的ショート秘密分散，SCIS2012 (2012).

[10] 千田浩司，五十嵐大，菊池　亮，濱田浩気：計算量的秘密分散およびランプ型秘密分散のマルチパーティ計算拡張，研究報告コンピュータセキュリティ（CSEC），Vol.2012-CSEC-58, No.40, pp.1–5 (2012).

[11] Cleve, R.: Limits on The Security of Coin Flips When Half the Processors are Faulty, *18th Annual ACM Sym-*

posium on Theory of Computing (*STOC '86*), pp.364–369, ACM Press (1986).
[12] Cramer, R., Damgård, I. and Maurer, U.: General Secure Multi-Party Computation from any Linear Secret-Sharing Scheme, *Advances in Cryptology-EUROCRYPT 2000*, Preneel, B. (Ed.), Lecture Notes in Computer Science, Vol.1807, pp.316–334, Springer, Berlin, Heidelberg (2000).
[13] Damgård, I., Ishai, Y. and Krøigaard, M.: Perfectly Secure Multiparty Computation and the Computational Overhead of Cryptography, *Advances in Cryptology-EUROCRYPT 2010*, Gilbert, H. (Ed.), Lecture Notes in Computer Science, Vol.6110, pp.445–465, Springer, Berlin, Heidelberg (2010).
[14] Damgård, I., Pastro, V., Smart, N. and Zakarias, S.: Multiparty Computation from Somewhat Homomorphic Encryption, *Advances in Cryptology-CRYPTO 2012*, Safavi-Naini, R. and Canetti, R. (Eds.), Lecture Notes in Computer Science, Vol.7417, pp.643–662, Springer, Berlin, Heidelberg (2012).
[15] Damgård, I., Keller, M., Larraia, E., Pastro, V., Scholl, P. and Smart, N.P.: Practical Covertly Secure MPC for Dishonest Majority – Or: Breaking the SPDZ Limits, *Computer Security – ESORICS 2013, ESORICS 2013*, Crampton, J., Jajodia, S. and Mayes, K. (Eds.), Lecture Notes in Computer Science, Vol.8134, Springer, Berlin, Heidelberg (2013).
[16] Gennaro, R., Rabin, M.O. and Rabin, T.: Simplified VSS and Fast-Track Multiparty Computations with Applications to Threshold Cryptography, *Proc. 17th annual ACM Symposium on Principles of Distributed Computing* (*PODC '98*), pp.101–111, ACM (1998).
[17] Gentry, C.: A Fully Homomorphic Encryption Scheme, Ph.D. Thesis, Stanford University, Stanford, CA, USA (Sep. 2009).
[18] 濱田浩気，菊池　亮：事前計算が効率的で不正者が多くても安全なマルチパーティ計算，コンピュータセキュリティシンポジウム 2015 論文集，pp.995–1002 (2015).
[19] 五十嵐大，千田浩司，高橋克巳：高効率 3 パーティ秘匿関数計算の情報理論的安全性，研究報告コンピュータセキュリティ（CSEC），Vol.2010-CSEC-50, No.46, pp.1–8 (2010).
[20] Mell, P. and Grance, T.: *The NIST Definitionof Cloud Computing*, National Institute of Standards and Technology (2011).
[21] Franklin, M.K. and Yung, M.: Communications Complexity of Secure Computation, *Proc. 24th Annual ACM Symposium on Theory of Computing*, pp.699–710, ACM (1992).
[22] Nishide, T. and Kazuo, O.: Multiparty Computation for Interval, Equality, and Comparison Without Bit-Decomposition Protocol, *Public Key Cryptography*, Vol.4450, pp.343–360 (2007).
[23] Shamir, A.: How to Share a Secret, *Comm. ACM*, Vol.22, No.11, pp.612–613 (1979).
[24] Shingu, T., Iwamura, K. and Kaneda, K.: Secrecy Computation without Changing Polynomial Degree in Shamir's (k, n) Secret Sharing Scheme, *Proc. 13th International Joint Conference on e-Business and Telecommunications-Volume 1: DCNET*, pp.89–94 (2016).
[25] 神宮武志，岩村恵市：ランプ型秘密分散法を用いた秘匿乗算手法の提案，コンピュータセキュリティシンポジウム 2015，3B2-3 (2015).
[26] Smart, N.P. and Vercauteren, F.: Fully Homomorphic Encryption with Relatively Small Key and Ciphertext Sizes, *Public Key Cryptography – PKC 2010, PKC 2010*, Nguyen, P.Q. and Pointcheval, D. (Eds.), Lecture Notes in Computer Science, Vol.6056, Springer, Berlin, Heidelberg (2010).
[27] 辻下健太郎，岩村恵市：パスワード付き秘密分散法の秘匿検索への応用，研究報告コンピュータセキュリティ（CSEC）(2017).
[28] van Dijk, M., Gentry, C., Halevi, S. and Vaikuntanathan, V.: Fully Homomorphic Encryption over the Integers, *EUROCRYPT 2010*, Gilbert, H. (Ed.), Lecture Notes Computer Science, Vol.6110, pp.24–43, Nice, France (2010).
[29] Watanabe, T., Iwamura, K. and Kaneda K.: Secrecy Multiplication Based on a (k, n)-Threshold Secret-Sharing Scheme Using Only k Servers, *Computer Science and Its Applications, Lecture Notes in Electrical Engineering*, Park, J., Stojmenovic, I., Jeong, H. and Yi, G. (Eds.), Vol.330, pp.107–112, Springer, Berlin, Heidelberg (2015).
[30] Yao, A.C.: Protocols for secure computations, *23rd Annual Symposium on Foundations of Computer Science* (*SFCS 1982*), Chicago, IL, USA, pp.160–164 (1982).

ムハンマド カマル アフマド アクマル アミヌディン

平成 29 年東京理科大学工学部電気工学科卒業．平成 30 年東京理科大学大学院工学研究科電気工学専攻修士課程在学．

岩村 恵市　（正会員）

昭和 55 年九州大学工学部情報工学科卒業．昭和 57 年同大学大学院情報工学研究科修士課程修了．同年キャノン（株）入社．平成 6 年東京大学博士（工学）．現在，東京理科大学工学部電気工学科教授．主に符号理論，並列処理，情報セキュリティ，電子透かしの研究に従事．IEEE，電子情報通信学会，電気学会各会員．エンリッチド・マルチメディア（EMM）研究会委員長，情報ハイディングおよびその評価基準（IHC）研究会委員長，本会フェロー．