

秘密分散法を用いた次数変化のない秘匿計算手法

神宮 武志¹ 青井 健¹ 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6月19日, 採録日 2017年12月8日

概要: 本論文では, Shamir の (k, n) しきい値秘密分散法を用いて, 秘密情報に 0 を含まないという条件の下で $2k - 1 > n$ の場合でも秘匿四則演算が可能な手法を提案する. 一般に, Shamir 法を用いた乗算は $k - 1$ 次の多項式からなる分散値どうしの乗算となるため, 乗算結果の多項式の次数が $2k - 2$ となってしまう問題点があり, 分散数 n は $2k - 1$ 以上にしなければならないという制限があった. 提案手法では秘密情報を分散する際, 乱数を掛けて秘匿化した秘匿化秘密情報を Shamir 法によって分散し, 乗算を行う際に, 一方の秘匿化秘密情報の分散値を集めて, 一時的に分散した秘匿化秘密情報を復元する. 秘匿化秘密情報はスカラー量であるので, 多項式で表現されるもう一方の分散値と乗算しても多項式の次数が増えないためこの問題点を解決できる. さらに, 本論文では提案方式の安全性を検討し, 多入力の場合の秘匿加減算および秘匿乗除算に対しても情報理論的安全性を持つことを示す. これによって, 秘匿乗算を含む場合でもサーバ台数などの運用を変える必要のないシステムが構築できる. ただし, 提案方式の安全性は同じタイプの演算の組合せに制限され, 異なるタイプの演算の安全な組合せ (たとえば, 秘匿積和演算) に関しては今後の課題になる.

キーワード: 秘匿計算, (k, n) しきい値秘密分散法, マルチパーティ計算, 情報理論的安全性

Secure Computation without Changing Polynomial Degree in (k, n) Secret Sharing Scheme

TAKESHI SHINGU¹ KEN AOI¹ AHMAD AKMAL AMINUDDIN MOHD KAMAL^{1,a)} KEIICHI IWAMURA¹

Received: June 19, 2017, Accepted: December 8, 2017

Abstract: We propose a new secrecy multiplication scheme without changing the polynomial degree even in case of $2k - 1 > n$ under the condition that 0 is not included in secret information in Shamir's (k, n) secret sharing scheme. In general, the multiplication in Shamir's scheme has a problem that the degree of the polynomial increases to $2k - 2$ and there is a limitation that the number n must be $2k - 1$ or more. Our scheme generates a scalar value called a concealed secret, which multiplies a secret by a random number, and distributes the concealed secret by using Shamir's scheme. When secure multiplying, we temporarily reconstruct the concealed secret, and multiply it with a share. Therefore, we can perform secrecy multiplication without changing the degree of polynomials by multiplying a polynomial and scalar value. We evaluate the security of our schemes, and show a possible application.

Keywords: secure computation, (k, n) secret sharing scheme, multiparty computation, information theoretical security

1. はじめに

ビッグデータの利活用とプライバシー保護の両立が可能な秘匿計算や秘匿検索は, IoT 社会において有用な技術であ

る. 暗号化したデータを暗号化したまま演算が可能な秘匿計算と, 暗号化したまま検索が可能な秘匿検索技術をビッグデータに適用することにより, 様々な統計演算や処理が可能になるため, プライバシーを保護しながら種々の知見を含む統計情報などの取得が期待できる. また, 暗号化したまま演算を行う技術として準同型暗号があげられるが, 一般に処理が重く, IoT のような比較的小さなデバイスへの適用を考えた

¹ 東京理科大学
Tokyo University of Science, Katsushika, Tokyo 125-8585, Japan

^{a)} ahmad@sec.ee.kagu.tus.ac.jp

場合、適していない。そこで、軽量かつ高速での実行が可能な秘密分散法を用いた秘匿計算技術の研究が多くされている [7], [8], [10], [11], [13], [14], [16], [17], [18], [19], [20], [21].

秘密分散法の 1 つである Shamir の (k, n) しきい値秘密分散法 [1] は、1 つの情報を異なる複数の情報 (分散値) に変換し、その分散値のうち k 個以上が集まれば元の情報の復元が可能だが、 k 個未満では元の情報は復元されないという方式である。これによって、サーバやネットワークの障害などによりデータの一部が使えなくても、それが $n - k$ 以下ならば元の情報が復元でき、さらに k 以上の情報が漏洩しない限り情報漏洩は起こらないという安全な情報管理システムが実現できる。

また、Shamir の (k, n) しきい値法を用いた従来の秘匿計算では、加減算は容易であるが、乗算に関しては乗算結果の復元に必要な分散値の数が k 個から $2k - 1$ 個に変化してしまうという問題がある。これは分散値どうしの乗算により、分散値を生成する多項式の次数が $k - 1$ から $2k - 2$ へと変化してしまうからである。そこで、Ben-Or ら [7] は復元に必要な $2k - 1$ 個の分散値数を k 個に戻す手法を提案した。また、Gennaro ら [10] は Ben-Or らの手法を改良し、より効率の良い秘匿乗算手法を提案した。しかし、復元時に最初に準備する分散値の数が $n \geq 2k - 1$ という制限は変わっていない。よって、秘匿乗算を行う場合、従来方式ではサーバ台数を $n \geq 2k - 1$ としなければならないため、サーバ台数設定など運用に制限が発生する。

本論文では、秘密情報に乱数をかけて秘匿化した秘匿化秘密情報を生成して、それを多項式で表現した分散値として秘密分散する。乗算の際はその秘匿化秘密情報の分散値を集めて、一時的にスカラー量として復元することで、多項式どうしの演算をスカラー量 $\times$ 多項式の形に変形し、復元に必要な分散値の数を変化させず、すなわち次数変化をさせずに乗算を行う手法を提案する。この手法はスカラー量を用いるため秘匿除算にも容易に対応できる。また、秘匿化秘密情報を用いた秘匿加減算も提案し、四則演算の秘匿演算において次数変化がなく、かつ情報理論的安全性を持つ手法を構成する。これによって、秘匿乗算を含む場合でもサーバ台数などに制限のないシステムが構築できる。

以降、2 章において代表的な秘密分散法と秘密分散法を用いた秘匿計算手法を紹介する。3 章では提案方式について説明し、4 章では提案方式の安全性について考察する。5 章では従来方式との比較および運用に関する考察を行い、6 章で提案方式の応用を紹介し、最後にまとめる。

2. 関連研究

2.1 Shamir の (k, n) しきい値秘密分散法

次の 2 つの条件を満たす秘密分散法を (k, n) しきい値秘密分散法と呼ぶ。

1. 秘密情報 s から生成された n 個の分散値のうち、任意

の k 個から秘密情報 s を復元することができる。

2. $k - 1$ 個以下の分散値からは、秘密情報 s に関する情報はいっさい得ることができない。

Shamir の提案した多項式補間による方法では、以下のようにして (k, n) しきい値秘密分散法 (以降、Shamir 法) を実現する。

[分散処理]

- (1) ディーラは $s < p$ かつ $n < p$ である任意の素数 p を選ぶ。
- (2) ディーラは $\mathbf{Z}/p\mathbf{Z}$ から、異なる n 個の x_i ($i = 1, 2, \dots, n$) を選び出し、サーバ ID とする。
- (3) ディーラは $\mathbf{Z}/p\mathbf{Z}$ から、 $k - 1$ 個の乱数 a_l ($l = 1, 2, \dots, k - 1$) を選んで、以下の式を生成する。

$$f(x) = s + a_1x + a_2x^2 + \dots + a_{k-1}x^{k-1} \pmod{p}$$

- (4) ディーラは上記式の x に各サーバ ID を代入して、分散値 $f_i = f(x_i)$ を計算し、各サーバ P_i に f_i を配布する。

[復元処理]

- (1) ユーザは任意の k 台のサーバから k 個の分散値 f_i ($i = 1, 2, \dots, k$) を収集する。
- (2) ユーザは k 個の分散値より多項式 $f(x)$ を復元し、 $f(0) = s$ より秘密情報 s を得る。

2.2 Shamir 法を用いた秘匿計算

Shamir の (k, n) しきい値秘密分散法に基づく秘匿加減算は 2.1 節の分散処理に示す多項式の次数を変化させず、簡単に実現できる。ただし、秘匿乗算に関しては、多項式どうしに乗算を行うと、生成された演算結果の多項式の次数が $k - 1$ から $2k - 2$ に変化してしまうという問題が発生する。具体的に、各サーバ P_i は a, b に関する 2.1 節に示す分散処理により生成された分散値 $f(x_i), g(x_i)$ を保持しているとし、以下の秘匿乗算を行う。ただし、秘密分散を含むすべての演算は a, b の演算結果よりも大きな素数 p を法として行われる。

[秘匿乗算]

- (1) サーバ P_i は $h(x_i) = f(x_i) \times g(x_i)$ を計算する。 P_i はこれを $a \times b$ についての分散値とし、ユーザへ送信する。

$$\begin{aligned} f(x_i) &= a + a_1x_i + a_2x_i^2 + \dots + a_{k-1}x_i^{k-1} \\ \times g(x_i) &= b + b_1x_i + b_2x_i^2 + \dots + b_{k-1}x_i^{k-1} \\ \hline h(x_i) &= ab + (a_1b + ab_1)x_i + \dots + a_{k-1}b_{k-1}x_i^{2k-2} \end{aligned}$$

- (2) ユーザは $2k - 1$ 個の分散値 $h(x_i)$ より多項式 $h(x)$ を復元し、 $h(0) = ab$ より ab を得る。

以上の秘匿乗算処理により、2.1 節の分散処理で生成した $k - 1$ 次多項式を用いて、多項式どうしの乗算を行うと、 $2k - 2$ の多項式になってしまう。つまり、Shamir 法を用

いた秘匿乗算の結果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が生じる。すなわち、分散値の数が $n < 2k-1$ であれば、秘匿乗算結果の復元が不可能となる。よって、各サーバが1つの分散値を持つと仮定すれば、乗算を行う場合は、サーバの台数を $2k-1$ 以上する必要がある。運用におけるコストを考慮すれば、サーバの台数の増加は、全体システムのコスト増につながる。

2.3 Ben-Or らの秘匿乗算

2.2 節の秘匿乗算は、 $h(x) = f(x) \times g(x)$ の次数が $2k-2$ となり、乗算結果の復元に必要な分散値の数が $2k-1$ 個必要となる。そこで、Ben-Or らは、 $h(x)$ の次数を $k-1$ へと下げ、 $h(x)$ の各係数をランダム化する手法を提案した。しかし、この手法は秘密情報の復元に必要な分散値の数が $n \geq 2k-1$ という制限は変わっておらず、また、この変換処理に必要な計算量も大きいという欠点がある。

2.4 Damgård らの SPDZ 2 方式

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 [18] を提案した。SPDZ 2 方式は、秘密情報のオーナー自身が参加者の1人であり、自身以外のすべての参加者 ($n-1$) 人が結託しても秘密情報の漏洩が発生しないというマルチパーティ計算モデルを想定している。また、SPDZ 2 における秘密情報の秘匿性は加法的秘密分散法を用いて実現する。

SPDZ 2 方式を用いた秘匿乗算は、定数倍乗算のアプローチを用いて実現し、多項式の次数変化を防ぐ。具体的には、秘密情報の分散値 $\langle x \rangle$, $\langle y \rangle$ を乗算するために、Beaver の回路ランダム化を用いた $a \cdot b = c$ を満たすランダムな分散情報 $(\langle a \rangle, \langle b \rangle, \langle c \rangle)$ を用いて、 $d = \text{open}(\langle x \rangle - \langle a \rangle)$, $e = \text{open}(\langle y \rangle - \langle b \rangle)$ 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 の事前処理では、 $(\langle a \rangle, \langle b \rangle, \langle c \rangle)$ を生成するために somewhat 準同型暗号を用いているので、秘匿乗算を簡単に実現できるが、somewhat 準同型暗号の実現にかかる計算量が非常に大きいことが問題点である。つまり、SPDZ 2 方式により、Shamir 法を用いた秘匿乗算に関わる $n \geq 2k-1$ の制限を回避することができるが、そのトレードオフとして、somewhat 準同型暗号にかかる計算のオーバーヘッドが大きくなってしまいう問題が発生する。

2.5 濱田らの方式

濱田らは SPDZ 2 の事前処理をより効率化する手法を提案した。濱田らの提案方式では、Beaver によって提案された commodity-based cryptography のモデルを用いて、計

算に参加する n 人のパーティに加えて、信用できる m 台のサーバの存在を仮定し、事前処理の支援を行い、 $(\langle a \rangle, \langle b \rangle, \langle c \rangle)$ の生成を効率化する。

濱田らの方式により、秘匿乗算に関わる $n \geq 2k-1$ の制限を回避できると同時に、SPDZ 2 における計算のオーバーヘッドを減らすことができた。ただし、Beaver の提案した commodity-based cryptography を利用することによって、信用できるサーバの存在を仮定する必要がある。運用を考えれば、信頼できるサーバの実現が難しいので、運用できる場面に制限があると考えられる。

3. 提案方式

秘密情報に乱数を掛けて秘匿化したものを秘匿化秘密情報と呼び、それを秘密分散し、秘匿乗算や除算の際は秘匿化秘密情報を一時的に復元してスカラー量とし、それをもう一方の分散値と掛け合わせるにより、多項式の次数を変化させずに秘匿乗算を行う手法を提案する。なお、提案方式は加法準同型と定数乗算に対する準同型性を持つ任意の秘密分散法に適用できる。ここでは、Shamir 法を想定する。また、秘密情報は $a, b \in \mathbb{Z}/p\mathbb{Z}$ であり、3.1 節で生成する α_j も $\alpha_j \in \mathbb{Z}/p\mathbb{Z}$ であり、一様分布する乱数である (乱数は 0 を除く)。ただし、秘匿乗算において秘密情報が 0 であれば、3.3 節の (2) で復元される $\alpha a = 0$ となり、秘密情報が 0 であることが漏洩してしまう。それを防ぐために、秘匿乗算において秘密情報は 0 を除くという条件を設定する。秘密分散を含むすべての演算は p を法として行われる。

[記号定義]

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

3.1 分散処理

入力: a

出力: $[a]_i := (\overline{[\alpha a]}_i, \overline{[\alpha 0]}_i, \dots, \overline{[\alpha_{k-1}]}_i)$ ($i=0, 1, \dots, n-1$)

- (1) デイーラは k 個の乱数 $\alpha_0, \dots, \alpha_{k-1}$ を生成し、 $\alpha = \prod_{j=0}^{k-1} \alpha_j$ を計算する。
- (2) デイーラは αa を計算し、 $\alpha a, \alpha_0, \dots, \alpha_{k-1}$ を Shamir 法を用いて n 台のサーバに分散する。
- (3) サーバ P_i は秘密情報 a に関する分散値集合として $[a]_i := (\overline{[\alpha a]}_i, \overline{[\alpha 0]}_i, \dots, \overline{[\alpha_{k-1}]}_i)$ を保持する。

3.2 復元処理

入力: $[a]_j := (\overline{[\alpha a]}_j, \overline{[\alpha 0]}_j, \dots, \overline{[\alpha_{k-1}]}_j)$ ($j=0, 1, \dots, k-1$)

出力: a

- (1) ユーザは k 台のサーバ P_j から $[a]_j$ を収集する。
- (2) ユーザは $\overline{[\alpha a]}_j, \overline{[\alpha 0]}_j, \dots, \overline{[\alpha_{k-1}]}_j$ ($j=0, 1, \dots, k-1$) から $\alpha a, \alpha_0, \dots, \alpha_{k-1}$ を復元し、以下より a を復元する。

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

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

3.3 秘匿乗算

a, b の分散値集合から $c = ab$ の分散値集合を生成する手順を以下に示す. c は 3.2 節に示す復元処理によって得られる. ここで, 乱数 α_j, β_j の復元を行う k 台のサーバ P_j ($j = 0, \dots, k-1$) は n 台のサーバのうちからあらかじめ指定されているとする. また, αa を復元するサーバを以下では P_0 とするが, αa は全サーバが知るのでどのサーバでもよい. i は全サーバを表すときに用い, j は特に指示がない場合, 指定された k 台のサーバを表すときに用いる. また, 秘匿乗算において秘密情報は 0 を含まないとする.

入力: $[a]_j, [b]_j$ ($j = 0, 1, \dots, k-1$)

出力: $[c]_i := [ab]_i$ ($i = 0, 1, \dots, n-1$)

- (1) P_0 は k 台のサーバ P_j から $[\alpha a]_j$ を収集する.
- (2) P_0 は αa を復元し, すべてのサーバへ αa を送信する.
- (3) P_i は $[\alpha \beta ab]_i = \alpha a \times [\beta b]_i$ を計算する.
- (4) P_j は $[\alpha_j]_0, \dots, [\alpha_j]_{k-1}, [\beta_j]_0, \dots, [\beta_j]_{k-1}$ を k 台のサーバから収集し α_j, β_j を復元する.
- (5) P_j は $\alpha_j \beta_j$ を計算し, サーバ $P_0, \dots, P_{n-1}$ へ秘密分散する.
- (6) P_i は $[c]_i := ([\alpha \beta ab]_i, [\alpha_0 \beta_0]_i, \dots, [\alpha_{k-1} \beta_{k-1}]_i)$ を $c = ab$ の分散値集合とする.

3.4 秘匿除算

秘匿除算は 3.3 節秘匿乗算の (3) における計算を $[\beta b / \alpha a]_i = [\beta b]_i / \alpha a$ とし, (5) 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 $[c]_i := ([\beta b / \alpha a]_i, [\beta_0 / \alpha_0]_i, \dots, [\beta_{k-1} / \alpha_{k-1}]_i)$ が得られる.

入力: $[a]_j, [b]_j$ ($j = 0, 1, \dots, k-1$)

出力: $[c]_i := [b/a]_i$ ($i = 0, 1, \dots, n-1$)

秘匿乗算と秘匿除算を合わせてタイプ 1 の秘匿計算と呼ぶ. 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 である場合を排除する必要がある. 提案方式は除数が 0 である場合, (2) において $\alpha a = 0$ となるため, そこで処理をやめることにより 0 による除算を排除できる.

3.5 秘匿加減算

入力: $[a]_j, [b]_j$ ($j = 0, 1, \dots, k-1$)

出力: $[c]_i := [a \pm b]_i$ ($i = 0, 1, \dots, n-1$)

- (1) P_j は $[\alpha_j]_0, \dots, [\alpha_j]_{k-1}, [\beta_j]_0, \dots, [\beta_j]_{k-1}$ を収集し α_j, β_j を復元する. P_j は乱数 $\gamma_j \in \mathbb{Z}/p\mathbb{Z}$ を生成し, P_0 に $\gamma_j / \alpha_j, \gamma_j / \beta_j$ を送信する.
- (2) P_0 は以下より $\gamma / \alpha, \gamma / \beta$ を復元し全サーバに送信す

る. ただし, $\gamma = \prod_{j=0}^{k-1} \gamma_j$ とする.

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

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

- (3) P_i は $(\frac{\gamma}{\alpha})[\alpha a]_i \pm (\frac{\gamma}{\beta})[\beta b]_i = [\gamma(a \pm b)]_i$ を計算する.
- (4) P_j は γ_j を $P_0, \dots, P_{n-1}$ へ秘密分散する.
- (5) P_i は $c = a \pm b$ の分散値集合として以下を持つ.

$$[c]_i := ([\gamma(a \pm b)]_i, [\gamma_0]_i, \dots, [\gamma_{k-1}]_i)$$

ここでは, 秘匿加減算をタイプ 2 の秘匿計算と呼ぶ.

4. 安全性

秘匿計算に関するエンティティは入力者, n 台のサーバ, 復元者であり, 各エンティティはプロトコルに従った処理を行い, 各エンティティ間の通信は安全であるとする. ただし, 攻撃に関しては以下の四者を考える. また, 同一のサーバセットにおいて, 1 つのサーバはつねに同じ乱数を扱うとする (たとえば, サーバ P_j はつねに α_j と β_j を復元・合成・分散する).

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, それを元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.

攻撃者 4: 秘匿演算の 1 つの入力および復元を行う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 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, もう 1 人の入力者が入力した秘密情報を知ろうとする.

ただし, 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 1~3 のみを想定する. 各秘匿計算が単独または組み合わせて実行された場合におい

て、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。すなわち、2 入力 a, b および出力 c において、各攻撃者が知ることができる情報を Y と定義すると、攻撃者 1、攻撃者 2 および攻撃者 3 は Y から入力 a, b および出力 c を知ることができれば攻撃成功となるが、 Y から a, b および c に関して何も情報が得られない以下の式が成り立つ場合、攻撃失敗とする。

$$H(a) = H(a|Y)$$

$$H(b) = H(b|Y)$$

$$H(c) = H(c|Y)$$

4.1 秘匿乗除算に関する安全性

4.1.1 攻撃者 1 の場合

攻撃者 1 は全サーバが知る αa を知ることができる。また、攻撃者 1 は $k-1$ 台のサーバから α_j, β_j ($j = 0, \dots, k-2$) を知ることができる。よって、攻撃者 1 が知ることができる暗号文相当の情報は αa と α_j, β_j ($j = 0, \dots, k-2$) となり、それを元に平文に相当する a, b が漏洩するかについて考える。まず、 $k-1$ 個の α_j, β_j からは α, β は漏洩しない。

$$H(\alpha) = H(\alpha|\alpha_0, \dots, \alpha_{k-2})$$

$$H(\beta) = H(\beta|\beta_0, \dots, \beta_{k-2})$$

また、 αa は a および α が 0 ではないので、 αa から α, a を特定できず、以下が成り立つ。

$$H(\alpha) = H(\alpha|\alpha a)$$

また、その組合せに関しても以下がいえる。

$$H(a) = H(a|\alpha a, \alpha_0, \dots, \alpha_{k-2})$$

また、 β_j ($j = 0, \dots, k-2$) は αa および α_j ($j = 0, \dots, k-2$) と独立であるため、以下がいえる。

$$H(a) = H(a|\alpha a, \alpha_0, \dots, \alpha_{k-2}, \beta_0, \dots, \beta_{k-2})$$

$$H(b) = H(b|\alpha a, \alpha_0, \dots, \alpha_{k-2}, \beta_0, \dots, \beta_{k-2})$$

すなわち、 $k-1$ 台のサーバが持つ情報、および演算中にサーバが知る情報を攻撃者 1 が得ても、秘密情報 a, b は漏洩しない。

次に、秘匿演算結果である ab が漏洩するかについて考える。 αa は分散値 $[\beta b]_i$ に掛けられ保存されるが、 $\alpha a \beta b$ はたとえ $k-1$ 個のサーバの情報が漏洩しても復元されない。ゆえに以下の式が成り立つ。

$$H(\alpha a \beta b) = H(\alpha a \beta b|\alpha a, \alpha_0, \dots, \alpha_{k-2}, \beta_0, \dots, \beta_{k-2})$$

加えて、 ab は $\alpha \beta$ が分からなければ漏洩しない。ゆえに以下の式が成り立つ。

$$H(\alpha \beta) = H(\alpha \beta|\alpha a, \alpha_0, \dots, \alpha_{k-2}, \beta_0, \dots, \beta_{k-2})$$

よって、以下のように攻撃者 1 は $k-1$ 台のサーバが持つ情報および演算中に知られる情報を得ても、下記のように秘匿計算結果である ab を知ることはできない。

$$H(ab) = H(ab|\alpha a, \alpha_0, \dots, \alpha_{k-2}, \beta_0, \dots, \beta_{k-2})$$

4.1.2 攻撃者 2 の場合

攻撃者 2 が b の入力者である場合を考える。この場合、攻撃者 2 は 4.1.1 項の場合の情報に加えて β と b を知るが、 α, a と β, b はまったく独立であるので、 β と b を知っても 4.1.1 項で示した以上の α と a に関する情報はまったく得られない。また、 α の入力者が攻撃者 2 である場合も同様に α, a と β, b はまったく独立であり、4.1.1 項で示した以上の β と b に関する情報は得られない。

4.1.3 攻撃者 3 の場合

攻撃者 3 は 4.1.1 項の場合の情報に加えて $\alpha \beta$ および $\alpha \beta ab$ を知る。よって、攻撃者 3 が知る暗号文相当の情報は $\alpha a, \alpha_j, \beta_j$ ($j = 0, \dots, k-2$) と $\alpha \beta, \alpha \beta ab$ である。 $\alpha \beta$ および $\alpha \beta ab$ から ab は得られるが、 a, b または α, β に分離することができない。また、4.1.1 項で議論したように αa と α_j, β_j ($j = 0, \dots, k-2$) から α, β は得られない。ゆえに以下の式が成り立つ。

$$H(a) = H(a|ab, \alpha \beta, \alpha_0, \dots, \alpha_{k-2}, \beta_0, \dots, \beta_{k-2})$$

$$H(b) = H(b|ab, \alpha \beta, \alpha_0, \dots, \alpha_{k-2}, \beta_0, \dots, \beta_{k-2})$$

以上より、攻撃者 3 は $k-1$ 台のサーバが持つ情報および演算中および復元中に知ることができる情報を得ても、秘密情報 a, b を知ることはできない。

ただし、 $a = b$ の場合、復元者は得られた a^2 から a を知るが、これはどの秘匿計算を用いても同様であるので対象としない（後述の秘匿加減算に対しても同様）。

4.1.1~4.1.3 項は秘匿除算に対しても同様にいえるため、タイプ 1 の秘匿計算は単独演算であれば情報理論的安全性を持つ。

4.2 秘匿加減算に関する安全性

4.2.1 攻撃者 1 の場合

攻撃者 1 は全サーバが知る γ/α と γ/β を知ることができる。また、攻撃者 1 は $k-1$ 台のサーバから $\gamma_j, \alpha_j, \beta_j$ ($j = 0, \dots, k-2$) を知ることができる。よって、攻撃者 1 が知ることができる暗号文相当の情報は $\gamma/\alpha, \gamma/\beta$ と α_j, β_j ($j = 0, \dots, k-2$) となり、それを元に平文に相当する a, b が漏洩するかについて考える。

$\gamma_j, \alpha_j, \beta_j$ は独立に定められており、 $k-1$ 個の $\gamma_j, \alpha_j, \beta_j$ からは γ, α, β は漏洩しない。また、 γ/α と γ/β は γ, α, β に分離されない。ゆえに以下の式が成り立つ。

$$H(\gamma) = H(\gamma|\gamma/\alpha, \gamma/\beta, \alpha_0, \dots, \alpha_{k-2}, \beta_0, \dots, \beta_{k-2},$$

$$\gamma_0, \dots, \gamma_{k-2})$$

$$H(\alpha) = H(\alpha | \gamma/\alpha, \gamma/\beta, \alpha_0, \dots, \alpha_{k-2}, \beta_0, \dots, \beta_{k-2}, \gamma_0, \dots, \gamma_{k-2})$$

$$H(\beta) = H(\beta | \gamma/\alpha, \gamma/\beta, \alpha_0, \dots, \alpha_{k-2}, \beta_0, \dots, \beta_{k-2}, \gamma_0, \dots, \gamma_{k-2})$$

また、攻撃者は全サーバが知る γ/α と γ/β の情報より秘密情報 a, b を秘匿化している乱数 α, β の比（たとえば、 α/β ）を知ることができる。ただし、秘匿加減算では秘匿乗除算のように αa または βb が復元されないので、乱数の比が分かっても秘密情報 a, b は漏洩しない。よって、攻撃者 1 は $k-1$ 台のサーバが持つ情報および演算中に知ることができる情報を得ても、秘密情報 a, b を知ることはできない。

次に、秘匿演算結果が漏洩するかを考える。演算結果は途中で復元されないので、たとえ $k-1$ 個のサーバの情報が漏洩しても復元できない。ゆえに以下の式が成り立つ。

$$H(a+b) = H(a+b | \gamma/\alpha, \gamma/\beta, \alpha_0, \dots, \alpha_{k-2}, \beta_0, \dots, \beta_{k-2}, \gamma_0, \dots, \gamma_{k-2})$$

以上により攻撃者 1 は $k-1$ 個のサーバが持つ情報、および演算中に知られる情報から秘匿計算結果を知ることができない。

4.2.2 攻撃者 2 の場合

a の入力者を攻撃者 2 とした場合、攻撃者 2 は α を知るので γ/α と γ/β から γ と β を知る。 b の入力者が攻撃者 2 である場合も同様に β から γ と α を知る。さらに、 $k-1$ 個の α_j, β_j と α, β からすべての α_j, β_j を知る。しかし、 $k-1$ 個の分散値 $[\alpha a]_j$ または $[\beta b]_j$ から $\alpha a, \beta b$ は復元できず、さらに秘匿加減算において、 αa または βb を復元する処理はない。よって攻撃者 2 は乱数 γ, α, β を知るが秘密情報 a, b を知ることはできない。これは、従来の Shamir 法による分散値どうしを用いた秘匿加算において、乱数 $\alpha = \beta = \alpha_j = \beta_j = 1$ が知られている状態と同じと考えることができる。すなわち、Shamir 法は提案法において乱数が漏洩している場合と考えられるが、 $k-1$ 個の分散値から秘密情報である a と b が漏洩しないのと同様である。よって、秘匿化に用いられる乱数が漏洩するだけでは攻撃は成功せず、安全性に問題はない。

4.2.3 攻撃者 3 の場合

攻撃者 3 は γ および $\gamma(a \pm b)$ を知り、 $k-1$ 個のサーバの情報を知る。さらに、攻撃者 3 は演算中に得られる $\gamma/\alpha, \gamma/\beta$ も知ることができる。よって、攻撃者 3 が知ることができる暗号文相当の情報は $\gamma, \gamma(a \pm b)$ および $\gamma/\alpha, \gamma/\beta, \alpha_j, \beta_j$ ($j = 0, \dots, k-2$) となり、それを元に平文に相当する a, b が漏洩するかについて考える。まず、 γ を知ることによって $\gamma/\alpha, \gamma/\beta$ から α, β を知ることができるが、秘匿加減算において αa および βb は復元されないで、秘密情報

a, b を知ることはできない。ゆえに以下の式が成り立つ。

$$H(a) = H(a | a+b, \gamma, \gamma/\alpha, \gamma/\beta, \alpha_0, \dots, \alpha_{k-2}, \beta_0, \dots, \beta_{k-2}, \gamma_0, \dots, \gamma_{k-2})$$

$$H(b) = H(b | a+b, \gamma, \gamma/\alpha, \gamma/\beta, \alpha_0, \dots, \alpha_{k-2}, \beta_0, \dots, \beta_{k-2}, \gamma_0, \dots, \gamma_{k-2})$$

よって、攻撃者 3 は $k-1$ 台のサーバが持つ情報、および演算中・復元中に知られる情報を得ても、秘密情報 a, b を知ることはできない。よって、攻撃者 3 も乱数 γ, α, β を知るが、秘密情報 a, b を知ることはできない。これも、Shamir 法において乱数が知られている場合と同じと考えられる。

4.2.1~4.2.3 項より 3.5 節に示す秘匿加減算、すなわちタイプ 2 の秘匿計算も情報理論的安全性を持つ。

4.3 秘匿乗除算と秘匿加減算の組合せに関する安全性

タイプの異なる 2 つの演算 $[c]_i := [ab]_i, [d]_i := [a+b]_i$ ($i = 0, 1, \dots, n-1$) を行う場合について考える。攻撃者 3 は秘匿乗除算から αa 、秘匿加減算から γ, α, β を知ることができ、 αa と α から秘密情報 a を知ることができる。また、 a と秘匿計算結果から b も知る。よって、3 章に示す秘匿計算はそのままでは演算の組合せに対して安全性を持たない。本論文では、2 入力演算の組合せを以下のように分類して定義する。

演算の組合せ：秘匿演算結果を入力とせず、同一または異なるタイプの秘匿演算を行うことである。たとえば、秘匿乗算 ab と秘匿乗算 ac を行う、または秘匿乗算 ab と秘匿除算 a/c を行う、または秘匿乗算 ab と秘匿加算 $a+b$ を行うなどである。

多入力演算：秘匿演算結果を用いて同一タイプの演算を繰り返すことである。具体例については 4.4 節で議論する。

演算の連続：秘匿演算結果を用いて任意の演算を連続させることである。たとえば、秘匿乗算 ab をした後、秘匿加算を行い秘匿積和演算 $ab+c$ を計算するなどである。

本節では、上記演算の組合せに対する安全性を議論する。

そこで、3.1 節に示した分散処理の特徴を利用して、異なる乱数を用いた複数の分散値集合を用いることを考える。すなわち、3.3, 3.4 節の秘匿乗除算に用いる乱数 α_j を $\alpha_j^{(1)}$ とし、秘匿加減算に用いる乱数 α_j を $\alpha_j^{(2)}$ とする ($\alpha_j^{(1)}$ と $\alpha_j^{(2)}$ は独立な一様乱数である)。それを用いて同じ秘密情報 a に対して 2 つの分散値集合 $[a]_i^{(1)}$ と $[a]_i^{(2)}$ を以下のよう

$$[a]_i^{(1)} = ([\alpha^{(1)}a]_i, [\alpha_0^{(1)}]_i, \dots, [\alpha_{k-1}^{(1)}]_i)$$

$$[a]_i^{(2)} = ([\alpha^{(2)}a]_i, [\alpha_0^{(2)}]_i, \dots, [\alpha_{k-1}^{(2)}]_i)$$

$$\left(\alpha^{(1)} = \prod_{j=0}^{k-1} \alpha_j^{(1)}, \alpha^{(2)} = \prod_{j=0}^{k-1} \alpha_j^{(2)} \right)$$

これを用いて秘匿乗除算と秘匿加減算を行えば、攻撃者

2 または攻撃者 3 は秘匿乗除算から $\alpha^{(1)}a$, 秘匿加減算から $\gamma, \alpha^{(2)}, \beta$ を知ることができるが $\alpha^{(1)}a$ と $\alpha^{(2)}$ は独立の乱数であるので, その組合せから秘密情報 a を知ることはできない. すなわち, 4.1 節に示した秘匿乗除算と 4.2 節に示した秘匿加減算に関する安全性が各々独立に成り立ち, タイプの異なる演算の組合せに対しても情報理論的安全性を実現できる. ただし, 演算結果の ab と $a+b$ の組合せから a, b が漏洩するのは他の秘匿計算手法を用いても同様であるので対象としない.

また, 同じタイプの演算の組合せの場合, 分散値集合 $[a]_i^{(1)}$ と $[b]_i^{(1)}$ を用いた秘匿乗算と秘匿除算を組み合わせても漏洩する情報は同じ $\alpha^{(1)}a$ であり, $[a]_i^{(2)}$ と $[b]_i^{(2)}$ を用いた秘匿加算と秘匿減算を組み合わせても漏洩する乱数は同じである. よって, 同じタイプの演算の組合せに対しては乱数の異なる分散値集合を用いる必要がない. よって, 少なくともタイプに応じた 2 つの分散値集合を持てばよい.

4.4 多入力の秘匿計算に関する安全性

4.1~4.3 節では 2 入力かつ 1 回の演算に関する安全性が示された. 秘匿計算では多入力を扱うことが多いので, 多入力の秘匿計算すなわち Πa または Σa に関する安全性を検討する. この場合, 入力者が入力の一部を知っていても, 残りの入力が多数であるので, 残りの入力を構成する個別の入力が分からなければ安全とする. よって, ここでは攻撃者が復元者かつ 1 つの値の入力者となる攻撃者 4 に対する安全性も検討する.

4.4.1 多入力の秘匿乗除算に関する安全性

多入力の乗算を, 3.3 節に示した 2 入力の秘匿乗算の連続によって実現することも可能であるが, 以下のように変形した方が効率的であり, 分かりやすい. ただし, 以下の (1) において P_0 で行う復元を, 複数のサーバで分担して行うことも可能である.

入力: $[a]_j, [b]_j, \dots, [z]_j$ ($j = 0, 1, \dots, k-1$)

出力: $[c]_i := [ab \cdots z]_i$ ($i = 0, 1, \dots, n-1$)

- (1) P_0 は k 台のサーバ P_j から $[\alpha a]_j$ を除く $[\beta b]_j, \dots, [\zeta z]_j$ を収集する.
- (2) P_0 は $\beta b, \dots, \zeta z$ を復元し, すべてのサーバへ送信する.
- (3) P_i は $[\alpha \cdots \zeta a \cdots z]_i = \beta b \times \cdots \times \zeta z \times [\alpha a]_i$ を計算する.
- (4) P_j は $[\alpha]_0, \dots, [\alpha]_{k-1}, \dots, [\zeta]_0, \dots, [\zeta]_{k-1}$ を k 台のサーバから収集し $\alpha_j, \dots, \zeta_j$ を復元する.
- (5) P_j は $\alpha_j \cdots \zeta_j$ を計算し, n 台のサーバ $P_0, \dots, P_{n-1}$ へ秘密分散する.
- (6) P_i は $c = a \cdots z$ の分散値集合として以下を持つ.

$$[c]_i := ([\alpha \cdots \zeta a \cdots z]_i, [\alpha_0 \cdots \zeta_0]_i, \dots, [\alpha_{k-1} \cdots \zeta_{k-1}]_i)$$

攻撃者 1 は αa 以外の秘匿化秘密情報 $\beta b, \dots, \zeta z$ と $k-1$ 個の $\alpha_j, \dots, \zeta_j$ を知る. 各秘匿化秘密情報および各部分乱

数からの情報漏洩については 4.1 節と同じ議論が成り立つため, 各乱数 $\alpha, \dots, \zeta$ は漏洩しない. また, それらを組み合わせても秘密情報および秘匿計算結果は漏洩しないことも 4.1 節の議論と同様である. また, 攻撃者 2 の場合も, 攻撃者 3 の場合も 4.1.2 項および 4.1.3 項と同様の議論によって秘密情報が漏洩しないことがいえる.

ここでは攻撃者が復元者かつ 1 つの値の入力者である攻撃者 4 について考える. まず, a の入力者が攻撃者 4 である場合, 攻撃者 4 は $a, \alpha, \beta b, \dots, \zeta z, \alpha \beta \cdots \zeta, \alpha \beta \cdots \zeta a \cdots z$ を知る. この場合, a または α から残りの $\beta \cdots \zeta$ および $b \cdots z$ を知ることができるが, 各秘密情報および各乱数を分離できないため, $\beta, \dots, \zeta$ が漏洩しないことも 4.1.1 項の議論から明らかである. また, a 以外の入力者が攻撃者 4 である場合も同様である.

よって, 多入力の秘匿乗除算に関しても情報理論的安全性を持つことがいえる.

一方, 本項に示した秘匿乗算は上記 (3) における乗算の繰返しに対して, モンゴメリ法や中国人の剰余定理のような従来知られた高速化手法を適用することができる. それに対して, 従来の分散値どうしを用いる乗算では 1 回乗算を行うごとに 2.3 節に示す次数変換処理をしなければ多項式の次数が膨大となり, 復元のための分散値の数も膨大となるため, 効率的なべき乗演算を実現できない.

4.4.2 多入力の秘匿加減算に関する安全性

多入力の加減算を 3.5 節に示した秘匿加減算の連続で実現することもできるが, 以下のような効率化を行う.

入力: $[a]_j, [b]_j, \dots, [z]_j$ ($j = 0, 1, \dots, k-1$)

出力: $[c]_i := [a \pm b \pm \cdots \pm z]_i$ ($i = 0, 1, \dots, n-1$)

- (1) P_j は $[\alpha]_0, \dots, [\alpha]_{k-1}, \dots, [\zeta]_0, \dots, [\zeta]_{k-1}$ を収集し $\alpha_j, \dots, \zeta_j$ を復元する. P_j は乱数 $r_j \in Z/pZ$ を生成し, P_0 に $\frac{r_j}{\alpha_j}, \dots, \frac{r_j}{\zeta_j}$ を送信する.
- (2) P_0 は以下より $\frac{r}{\alpha}, \dots, \frac{r}{\zeta}$ を復元し全サーバに送信する.

$$\begin{aligned} \frac{r}{\alpha} &= \prod_{j=0}^{k-1} \frac{r_j}{\alpha_j} \\ &\vdots \\ \frac{r}{\zeta} &= \prod_{j=0}^{k-1} \frac{r_j}{\zeta_j} \end{aligned}$$

- (3) P_i は $(\frac{r}{\alpha})[\alpha a]_i \pm \cdots \pm (\frac{r}{\zeta})[\zeta z]_i = [r(a \pm \cdots \pm z)]_i$ を計算する.
- (4) P_j は r_j を n 台のサーバ P_i に秘密分散する.
- (5) P_i は $c = a \pm b \pm \cdots \pm z$ に関する分散値集合として以下を持つ.

$$[c]_i := ([r(a \pm \cdots \pm z)]_i, [r_0]_i, \dots, [r_{k-1}]_i)$$

攻撃者 1 は $\frac{r}{\alpha}, \dots, \frac{r}{\zeta}$ を知る. これから, 各乱数 $\alpha, \dots, \zeta$ の比を知ることができる. また, 攻撃者 1 は $k-1$ 台のサー

バから $r_j, \alpha_j, \dots, \zeta_j$ ($j = 0, \dots, k-2$) を知ることができる。よって、残りの部分乱数の比も知ることができる。しかし、 $k-1$ 台のサーバの情報だけから各秘匿化秘密情報および秘匿計算結果を知ることにはできないので 4.2.1 項と同様の議論により攻撃者 1 に秘密情報は漏洩しない。攻撃者 2 または攻撃者 3 の場合、攻撃者は各乱数 $r, \alpha, \dots, \zeta$ を知るが、やはり各秘匿化秘密情報値および秘匿計算結果を知ることにはできないので 4.2.2 項および 4.2.3 項と同様の議論により秘密情報は漏洩しない。

最後に、攻撃者 4 は各乱数 $r, \alpha, \dots, \zeta$ および $r(a \pm \dots \pm z)$ を知るが、各秘密情報に分解できないので、自分を除く加減算結果を知るのみである。すなわち、多入力秘匿加減算についても情報理論的安全性を持つことがいえる。

4.4.3 多入力の秘匿乗除算と秘匿加減算の組合せに関する安全性

多入力の秘匿乗除算と秘匿加減算の組合せは 4.3 節に示したように同じ秘密情報に対して異なる乱数を用いた分散値集合を用いることによって、独立の演算とできるため問題ない。以上のように、多入力の秘匿乗除算、秘匿加減算およびその組合せについても情報理論的安全性を持つことがいえる。

4.5 秘匿計算 $m(a+b+c)$ に関する安全性

タイプの異なる演算の連続に関する安全性確立は今後の課題であるが、 $m(a+b+c)$ の秘匿演算は a, b, c に関して情報理論的安全性を持つことがいえる。この場合の演算手順は以下のようになる (c, m も乱数 γ, μ を用いて 3.1 節の手順で分散されているとする)。

- 入力: $[a]_j, [b]_j, [c]_j, [m]_j$ ($j = 0, 1, \dots, k-1$)
 出力: $[d]_i := [m(a+b+c)]_i$ ($i = 0, 1, \dots, n-1$)
- (1) P_j は $[\alpha_j]_0, \dots, [\alpha_j]_{k-1}, [\beta_j]_0, \dots, [\beta_j]_{k-1}, [\gamma_j]_0, \dots, [\gamma_j]_{k-1}$ を収集し、 $\alpha_j, \beta_j, \gamma_j$ を復元する。
 - (2) P_j は乱数 $\delta_j \in \mathbb{Z}/p\mathbb{Z}$ を生成し、 P_0 に $\frac{\delta_j}{\alpha_j}, \frac{\delta_j}{\beta_j}, \frac{\delta_j}{\gamma_j}$ を送信する。
 - (3) P_0 は $\frac{\delta}{\alpha}, \frac{\delta}{\beta}, \frac{\delta}{\gamma}$ を復元し全サーバに送信する。
 - (4) P_i は $(\frac{\delta}{\alpha})[\alpha a]_i + (\frac{\delta}{\beta})[\beta b]_i + (\frac{\delta}{\gamma})[\gamma c]_i = [\delta(a+b+c)]_i$ を計算する。
 - (5) P_0 は k 台のサーバ P_j から $[\mu m]_i$ を集めて μm を復元し、全サーバに送る。
 - (6) P_i は $[\delta \mu m(a+b+c)]_i = \mu m[\delta(a+b+c)]_i$ を計算する。
 - (7) P_j は $[\mu_j]_0, \dots, [\mu_j]_{k-1}$ を収集し μ_j を復元し、 $\delta_j \mu_j$ を計算して n 台のサーバに秘密分散する。
 - (8) P_i は $m(a+b+c)$ の分散値集合として以下を持つ。

$$[d]_i := ([\delta \mu m(a+b+c)]_i, [\delta_0 \mu_0]_i, \dots, [\delta_{k-1} \mu_{k-1}]_i)$$

上記において、(1)~(4) を秘匿加算、(5)~(7) を秘匿乗算に分けることができ、秘匿加算で知られた乱数は秘匿

乗算では復元されない。よって、攻撃者 1 は単独では $\frac{\delta}{\alpha}, \frac{\delta}{\beta}, \frac{\delta}{\gamma}, \mu m$ および $k-1$ 個の $\alpha_j, \beta_j, \gamma_j, \delta_i, \mu_i$ を知るのみであり、それらから秘密情報および秘匿計算結果は漏洩しない。

また、 a の入力者を攻撃者 2 とした場合、攻撃者 2 は $\frac{\delta}{\alpha}$ から δ を知るが、 $\delta(a+b+c)$ が復元されることはなく、最終的な演算結果も知りえない。また、 m は a と独立であるので、個別の秘密情報を知ることにはできない。これは b, c の入力者が攻撃者 2 であっても同様である。 m の入力者が攻撃者 2 の場合、 m と a, b, c は独立であるので、 m の入力者は攻撃者 1 以上の情報を得ない。

攻撃者 3 は攻撃者 1 が知る情報に加えて $\delta \mu m(a+b+c)$ と $\delta \mu$ を知る。しかし、 δ と μ を分離できず a, b, c, m を個別に知ることはできない。よって、攻撃者 1~3 に関して $m(a+b+c)$ の秘匿演算は情報理論的安全性を持つことがいえる。

a の入力者が攻撃者 4 になる場合、攻撃者 4 は δ を知る所以 $\delta \mu$ を分離でき μm から m を知ることができる。さらに、演算結果である $m(a+b+c)$ と自らが知る a から $b+c$ を得ることができる。しかし、 $b+c$ を b と c に分離できないので、 b, c を個別に得ることはできない。 m の入力者が攻撃者 4 となっても、 m と a, b, c は独立であるので a, b, c に関する情報は漏洩しない。よって、どの場合でも a, b, c に関しては各々情報理論的安全性が保証される。

5. 従来方式との比較および運用に関する考察

$n = k$ における秘匿計算を実現する従来方式として文献 [17]、文献 [18] がある。ただし、文献 [18] に示される SPDZ 2 方式は文献 [17] に示される SPDZ 方式の改良版であるので、以降に、提案方式と SPDZ 2 方式との比較を示す。

SPDZ 2 方式は $n = k$ の設定に限定されており、自身がプレイヤーの 1 人で、 n 人のプレイヤー中自分以外の $n-1$ 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 は準同型暗号で事前処理を行う preprocessing phase と秘密分散を用いて秘匿計算を実行する online phase によって構成される。よって、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。また、SPDZ 2 方式は秘匿加算と秘匿乗算のような秘匿計算の組合せに対しても安全である。ただし a と b の分散値から秘匿除算を行う場合、 $1/a$ と b の分散値が必要となるが、 a の分散値から $1/a$ の分散値を求めるために、事前計算を含む余計な計算が必要となる（たとえば、秘密の乱数 r を秘密分散し、 a の分散値と乗算し、 ar を公開し、乱数 r の分散値に $1/ar$ をかけて、 $1/a$ を求める）。

それに対して、提案方式は $n = k$ の設定に限定されず、

表 1 提案方式と SPDZ 2 との比較

Table 1 Comparison of proposed method and SPDZ 2.

	提案方式	SPDZ 2
n と k の設定	$n \geq k$	$n = k$
想定する攻撃者	Passive adversary	Active adversary
実現する安全性	情報理論的安全性	計算量的安全性
組み合わせ可能な演算	同一タイプの演算	制限なし
秘匿除算	直接実行可能	逆数の分散値を求める演算が別に必要

$k \leq n$ 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, 提案方式において, 攻撃者は情報漏洩とその情報を基に秘密情報の解析などを行うが, プロトコルに従う passive adversary 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, 提案方式はタイプ 1 の秘匿計算 (乗算と除算) とタイプ 2 の秘匿計算 (加算と減算) のような同一タイプの秘匿計算の組合せであれば, 入力数に制限なく繰り返すことができる. ただし, 提案方式は異なるタイプの秘匿計算の連続に対しては安全性の観点から問題を残す (4.5 節においては a, b, c は秘匿できるが, 攻撃者 4 に対しては m は漏洩する). ただし, 提案方式は 3.4 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. 以上の比較を, 表 1 にまとめて示す.

また, 提案方式の通信量と通信回数および計算量の定量評価を表 2, 表 3 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. さらに, 提案方式と SPDZ 2 の計算量や通信量に関する比較を表 4 に示す. ここで, Shamir の (k, n) 閾値秘密分散におけるシェアのサイズを d_1 とし, 提案方式では秘密分散を $k+1$ 回行うことによって, 分散処理において全体のシェアサイズを $d_1(k+1)$ となる. さらに, somewhat 準同型暗号 (SHE) におけるデータサイズを d_2 とし, 秘密分散に関する計算量を C_1 , somewhat 準同型暗号 (SHE) に関する計算量を C_2 とする. また, 通常の乗除算に関する計算量を M , 加減算に関する計算量を A で表す. Shamir の (k, n) 閾値秘密分散におけるシェアサイズ d_1 は秘密情報のサイズとほぼ同程度とでき, somewhat 準同型暗号 (SHE) のデータサイズ 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. ただし, 提案方式のシェアサイズは Shamir の (k, n) 閾値秘密分散のシェアサイズに比べて $k+1$ 倍大きくなるので, somewhat 準同型暗号 (SHE) のデータサイズ d_2 より大きくなるが, SPDZ 2 方式では秘密情報より生成されたシェア d_1 に加えて, 秘匿乗算において消費される multiplicative

表 2 提案方式の通信量とラウンド数

Table 2 Communication and number of rounds of the proposed method.

処理	手順番号	各通信量	合計
分散処理 (秘密情報 a, b)	(2) $[a]_i$ の分散	$nd_1(k+1)$	ラウンド 数: 1 通信量: $2nd_1(k+1)$
	(2) $[b]_i$ の分散	$nd_1(k+1)$	
秘匿乗算 $a \times b$	(1)	kd_1	ラウンド 数: 4 通信量: $\{(k+n)(k+1) + k^2\}d_1$
	(2)	nd_1	
	(4)	$2k^2d_1$	
	(5)	$nk d_1$	
秘匿加算 $a + b$	(1) $[\alpha_j]_0, \dots, [\alpha_j]_{k-1},$ $[\beta_j]_0, \dots, [\beta_j]_{k-1}$ の収集	$2k^2d_1$	ラウンド 数: 4 通信量: $\{(2k+2n)(k+1) - nk\}d_1$
	(1) $\gamma_j/\alpha_j, \gamma_j/\beta_j$ の送信	$2kd_1$	
	(2)	$2nd_1$	
	(4)	$nk d_1$	
復元処理	(1)	$k(k+1)d_1$	ラウンド 数: 1 通信量: $k(k+1)d_1$

triple のシェアも考慮する必要があるので, 全体的に提案方式のシェアサイズが SPDZ 2 より小さいことがいえる. また, C_1 と C_2 は一般に, $C_1 \ll C_2$ の関係であり, C_2 は C_1 に比べて十分大きいと考えられる. 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, どちらも C_2 より十分小さいので簡単のため C_1 で表す. また, M と A は C_1, C_2 より十分小さいと考えられるので, 表 2, 表 3 には示すが合計値で比較する場合においては省略する. また, 零知識証明や 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省略する.

表 4 から計算量に関しては, 提案方式は分散処理時および秘匿演算中に k 個の乱数の復元と分散を行うため, 秘匿加算に関しては SPDZ 2 より劣るが, 秘匿乗算に関しては somewhat 準同型暗号 (SHE) を用いた事前計算を行わないため C_2 を含まず, 計算量においては SPDZ 2 より大きく優れると考えられる. また, 提案方式と SPDZ 2 の秘匿加算と秘匿乗算の計算量を合わせた全体の差分を比べると, 提案方式は C_2 に依存する計算量を含まないため, 提案方式は全体として SPDZ 2 より高速な計算が可能といえる. また, 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. ただし, ラウンド数に関し

表 3 提案方式の計算量

Table 3 Computational complexity of the proposed method.

処理	手順番号	各通信量	合計
分散処理 (秘密情報 a, b)	(1)	$2(k-1) \cdot M$	$2(k+1) \cdot C_1$ $+ 2k \cdot M$
	(2)	$2M,$ $2(k+1) \cdot C_1$	
秘匿乗算 $a \times b$	(2)	C_1	$(n+k) \cdot M$ $+ (3k+1)$ $\cdot C_1$
	(3)	$n \cdot M$	
	(4)	$2k \cdot C_1$	
	(5)	$k \cdot M,$ $k \cdot C_1$	
秘匿加算 $a + b$	(1)	$2k \cdot C_1,$ $2k \cdot M$	$3k \cdot C_1 + nA$ $+ 2(2k+n$ $-1) \cdot M$
	(2)	$2(k-1) \cdot M$	
	(3)	$2n \cdot M + n \cdot A$	
	(4)	$k \cdot C_1$	
復元処理	(2)	$(k+1) \cdot C_1,$ $k \cdot M$	$(k+1) \cdot C_1$ $+ k \cdot M$

表 4 提案方式と SPDZ 2 方式の比較

Table 4 Comparison of proposed method and SPDZ 2.

	処理	提案方式	SPDZ 2
計算量	秘匿乗算 $a \times b$	$(6k+4) \cdot C_1$	$6 \cdot C_1 + 2 \cdot C_2$
	秘匿加算 $a + b$	$(6k+3) \cdot C_1$	$3 \cdot C_1$
通信量	秘匿乗算 $a \times b$	$\{3n(k+1)$ $+ k(3k+2)\}d_1$	$2(6n-1)d_2$
	秘匿加算 $a + b$	$\{n(3k+4)$ $+ 3k(k+1)\}d_1$	$3nd_2$
ラウンド数	秘匿乗算 $a \times b$	6	5
	秘匿加算 $a + b$	6	2

て提案方式は k 個の乱数の復元と分散を行うため SPDZ 2 より多い。

以上より、提案方式と SPDZ 2 を比べると、全体として本提案方式の方が軽量な処理を実現するといえる。

一方、運用を考慮した場合、SPDZ 2 方式は、秘密情報を保持する入力者が演算を行うプレイヤーの 1 人として参加することを想定している。しかし、現代のビッグデータ解析において、秘密情報の入力者が直接秘匿計算に参加することは運用的にも現実的でない。また、大量の秘密情報を用いて秘匿計算する場合、SPDZ 2 ではその秘密情報のオーナーが秘匿計算にプレイヤーとして参加するため、非常に大きな $n = k$ を設定する必要がある。一般に、ビッグデータ解析においては、大量の情報がクラウドサーバなどに秘匿されて保管されており、それらを復元することなく、それら

の平均値や分散などの統計値を得ることが求められる。その場合、4.4 節に示すように多入力 $a \sim z$ の各オーナーがそれらを提案方式によって n 台のクラウドサーバに分散しており、秘匿計算はそのうちの k 台のサーバが行うとした方が運用も容易であり、現実的である。

6. 提案方式の応用

提案方式は秘密情報として 0 を含まず、 $2k-1 > n$ の場合においても秘匿計算が可能である。まず、0 を含まない情報は、医療データなどには多く存在する。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は用いられない。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。

以降に、ビッグデータにおける運用例として、多くの秘密情報が n 台のクラウドサーバに分散・保管されている場合を考える。たとえば、秘密情報 $a_1, \sim, a_z$ を患者 $O_1, \sim, O_z$ の血糖値などのデータとし、それを病院が 3.1 節の手法により n 台のクラウドサーバに秘密分散して管理しているとする。このクラウドサーバは複数の病院に利用されており、膨大なデータが秘匿管理されているとする。このデータは基本的に個人情報であるので秘匿されているが、学術目的などの場合のみ情報が漏洩しないことを条件に利用が許可されるとする。そこで、ある研究者が復元者として、その秘匿管理されているデータの平均値などの統計情報を計算したい場合を考える。ただし、その復元者は統計計算に必要な母数となる患者の総数 z を指定しており、 $1/z$ を計算して小数点をずらして $1/z$ を整数化して m とし、そのずらし幅などを保存しているとする。さらに、復元者は m の分散値集合 $[m]_i$ を 3.1 節の手法により分散しているとする。

[平均値の秘匿計算]

(1) 各サーバは、4.5 節の入力 $[a]_i, [b]_i, [c]_i, [m]_i$ を $[a_1]_i', \sim, [a_z]_i', [m]_i$ として、多入力演算を行い、その平均値 μ の分散値集合 $[\mu]_i := ([r\mu]_i, [r_0]_i, \dots, [r_{k-1}]_i)$ を計算する。

(2) 復元者は k 個の分散値集合 $[\mu]_i$ を集め、平均値 μ を得る。

上記秘匿計算は、4.5 節に示した $m(a+b+c)$ の秘匿計算と同様の処理であるので、4.5 節において議論したように、各患者の秘密情報 $a_1, \sim, a_z$ に関しては情報理論的安全性を持ち、漏洩することがない（この場合、復元者は m の入力者となる攻撃者 4 に相当する）。

提案方式では、サーバの規模を最小にするため、 $k = n = 2$ に設定できる。Shamir 法を用いる従来の秘匿計算では $n \geq 2k-1$ 台のサーバを必要とするため、 $k = n = 2$ では秘匿乗算を行うことができず、SPDZ 2 では膨大な事前計

算を必要とする。また、提案方式はそのクラウドサーバのサーバ故障などを考慮して、 $k = 3$, $n = 4$ または $k = 4$, $n = 5$ とすることもできる（従来方式では最小でも $k = 3$, $n = 5$ または $k = 4$, $n = 7$ とする必要がある）。また、SPDZ 2 は $k = n$ 以外に対応できない。以上より、提案方式は秘匿乗算の有無に関わりなく当初見積もったサーバ規模を設定でき、かつ小さな計算量で秘匿計算を実行できるため、自由な運用を可能にする。

7. まとめ

本論文では、秘匿乗算において秘密情報に 0 は含まないという条件のもとで $2k - 1 > n$ においても有効な (k, n) しいき値秘密分散法を用いた、新たな秘匿計算手法を提案した。この手法は $n < 2k - 1$ においても秘匿計算が実行可能であり、情報理論的安全性を持つ。これによって、秘匿乗算を含んでもサーバ台数を変化させる必要がなく、自由な運用を可能とする。ただし、異なるタイプの演算の組合せ（たとえば、秘匿積和演算）を安全に実現できる手法の検討およびその安全性の評価は今後の課題になる。

参考文献

- [1] Shamir, A.: How to share a secret, *Comm. ACM*, Vol.22, No.11, pp.612–613 (1979).
- [2] Yao, A.C.: On the succession problem for Byzantine Generals, manuscript (1983).
- [3] Blakley, G.R.: Security of ramp schemes, *CRYPTO '84*, pp.242–268 (Nov. 1984).
- [4] Kawamoto, Y. and Yamamoto, H.: (k, L, n) Ramp Secret Sharing Systems for Functions, *IEIC*, Vol.J68-A, No.9, pp.945–952 (1985).
- [5] Yao, A.C.: How to generate and exchange secrets, *STOC86* (1986).
- [6] Ito, M., Saito, A. and Nishizeki, T.: Secret Sharing Scheme Realizing General Access Structure, *Proc. IEEE Global Telecommunications Conference, Globecom '87*, pp.99–102 (1987).
- [7] Ben-Or, M. and Goldwasser, A.S.: Wigderson, Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation, *ACM* (1988).
- [8] Beaver, D.: Efficient multiparty protocols using circuit randomization, Feigenbaum, J. (Ed.), *CRYPTO*, Vol.576 of Lecture Notes in Computer Science, pp.420–432, Springer (1991).
- [9] Krawczyk, H.: Secret Sharing Made Short, *CRYPTO '93*, pp.136–146 (1994).
- [10] Gennaro, R., Rabin, M.O. and Rabin, T.: Simplified VSS and Fast-track Multiparty Computations with Applications to Threshold Cryptography, *17th ACM Symposium on Principles of Distributed Computing*, pp.101–112 (1998).
- [11] Beerliová-Trubíniová, Z. and Hirt, M.: Perfectly-Secure MPC with Linear Communication Complexity, *Theory of Cryptography*, Vol.4948 of the series Lecture Notes in Computer Science, pp.213–230 (2008).
- [12] Mell, P. and Grance, T.: The NIST Definition of Cloud Computing, *National Institute of Standards and Technology* (2011).
- [13] Ben-Sasson, E., Fehr, S. and Ostrovsky, R.: Near-linear unconditionally-secure multiparty computation with a dishonest minority, *IACR Cryptology ePrint Archive*, Vol.2011, No.629 (2011).
- [14] 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).
- [15] Kurihara, J. and Uyematsu, T.: A Novel Realization of Threshold Schemes over Binary Field Extensions, *IEICE Trans. Fundamentals*, Vol.E94-A, pp.1375–1380 (2011).
- [16] Asharov, G., Jain, A., López-Alt, A., Tromer, E., Vaikuntanathan, V. and Wichs, D.: Multiparty computation with low communication, computation and interaction via threshold FHE, Pointcheval, D. and Johansson, T. (Eds.), *EUROCRYPT*, Vol.7237 of Lecture Notes in Computer Science, pp.483–501, Springer (2012).
- [17] Damgård, I., Pastro, V., Smart, N. and Zakarias, S.: Multiparty Computation from Somewhat Homomorphic Encryption, *CRYPTO*, pp.643–662 (2012).
- [18] 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, *Proc. ESORICS*, Vol.8134 of Lecture Notes in Computer Science, pp.1–18, Springer (2013).
- [19] 渡辺泰平, 金田北洋, 岩村恵市: サーバ台数の変化が生じない (k, n) しいき値秘密分散法に基づく乗算手法, *電子情報通信学会論文誌 D*, Vol.J98-D, No.3, pp.428–436 (2015).
- [20] 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).
- [21] Kikuchi, R., Chida, K., Ikarashi, D., Ogata, W., Hamada, K. and Takahashi, K.: Secret Sharing with Share-Conversion: Achieving Small Share-Size and Extendibility to Multiparty Computation, *IEICE Trans. Fundamentals*, Vol.E98-A, pp.213–222 (2015).



神宮 武志

平成 27 年東京理科大学工学部電気工学科卒業。平成 29 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修了。



青井 健

平成 28 年東京理科大学工学部電気工学科卒業。平成 29 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。



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。同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 年同大学院情報工学研究科修士課程修了。同年キャノン株式会社入社。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。