Vol.2017-CSEC-78 No.37
Vol.2017-SPT-24 No.37
2017/7/15

演算の連続に対して安全な秘密分散法を用いた秘匿計算

MOHD KAMAL AHMAD AKMAL AMINUDDIN[†1]　岩村恵市[†1]

概要：従来の秘密分散を用いた秘匿計算において，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，元の情報を復元するために必要となる分散情報の数がk個から$2k-1$個に変化するという問題があった．それらの問題を解決するために神宮らによってTUS方式と呼ぶ2入力の秘匿演算法を提案したが，$ab+c$のように乗算と加算を連続的に行う場合，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，異なる演算の連続に対しても安全な秘密分散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においては，$2k-1>n$の場合，無条件に安全な計算は不可能とされている．ただしこれは，ある条件のもとでは$2k-1>n$における秘匿計算が安全に実行できる可能性があるということも意味する．よって，本論文は$2k-1>n$において異なる演算の連続を安全に実現するための条件を明らかにするための研究とも位置づけられる．

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

Secure Multiparty Computation using Secret Sharing Scheme against Consecutive Computation

AHMAD AKMAL AMINUDDIN MOHD KAMAL[†1]　KEIICHI IWAMURA[†1]

Abstract: When performing secrecy multiplication using secret sharing scheme, the result is a polynomial with degree of $2k-2$ instead of $k-1$. In order to reconstruct a multiplication result, the number of polynomial needed will increase from k to $2k-1$. Shingu et al. had proposed a method called the TUS method, allowing multiplication using secret sharing scheme to be performed without increasing the degree of polynomial by using the (scalar value×polynomial) approach instead of the typical (polynomial ×polynomial). However, the TUS method is not secure when consecutive computation such as product-sum operation of $ab+c$ is performed. In this paper, we propose a new method that is secure against product-sum operation without increasing the degree of polynomial. Typically, multiparty computation using secret sharing scheme is not unconditionally secure if $2k-1>n$. However, this also means that secure multiparty computation is realizable with conditions. Therefore, in this paper, we clarify the conditions needed to achieve conditionally secure multiparty computation using secret sharing scheme.

Keywords: secrecy computation, secret sharing scheme, multiparty computation, information-theoretic secure, $n<2k-1$

1. はじめに

近年，クラウドシステムを用いたビッグデータの利活用が注目されている．ビッグデータを収集し，分析することによって，研究やビジネスへの利活用が期待される．ただし，ビッグデータには利用者の個人情報なども含んでいるので，ビッグデータの分析において，利用者の個人情報およびプライバシーが保護できる技術が必要となる．

情報を守りながら演算を行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[4, 512, 13, 15, 16]と鍵を用いずにデータを秘匿化する秘密分散法を用いた秘匿計算[2, 6, 10, 14, 19, 20]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．そのためで，クラウドシステム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分散情報に変換し，分散する手法である．秘密分散法の1つであるShamirの(k,n)閾値秘密分散法[18]は，1つの秘密情報をn個の分散情報に変換し，n台のサーバに分散する．Shamirの秘密分散法の特徴は，分散したn個の分散情報から，k個の分散情報を集めれば，元の情報を復元することができるが，それ未満の情報からは，秘密情報に関する情報を一切得ることができない．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わかる．

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情報の数がk個から$2k-1$個に変化してしまうという問題である．それらの問題を解決するために神宮らによってTUS方式[19]と呼ぶ2入力の秘匿演算法が提案されている．TUS方式では秘匿乗算を行う際に，一時的に分散情報を復元し，（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

†1 東京理科大学

Vol.2017-CSEC-78 No.37
Vol.2017-SPT-24 No.37
2017/7/15

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ことが言える．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，攻撃者が一人の入力者の入力した乱数と演算結果を知ることができれば，残りの入力者の秘密情報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，異なる演算の連続に対しても安全な秘密分散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，無条件に安全な計算は不可能とされている．ただしこれは，ある条件のもとでは$2k-1>n$における秘匿計算が安全に実行できる可能性があるということも意味する．しかし，その条件は知られていない．その条件の実現が現実的であれば，$2k-1>n$における秘匿計算は実用的な秘匿計算法ということができる．よって，本論文は$2k-1>n$において異なる演算の連続，すなわち汎用的な秘匿計算を安全に実現するための条件を明らかにするための研究とも位置づけられる．

本論文の構成を以下に示す．第 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．第 3 章では$2k-1>n$において異なる演算の連続を実現する提案方式を説明し，それを実現するための条件を明らかにする．第 4 章では提案した方式の評価を行い，第 5 章で考察を行い，第 6 章でまとめを行う．

2. 従来方式

2.1 TUS 方式

2.1.1 TUS 方式

TUS 方式[19]は秘密情報をまず乱数とかけて秘匿化した秘密情報を生成し，それを Shamir の(k,n)-閾値秘密分散法[18]で分散する方式である．秘匿乗算を行う際に，秘匿化された秘密情報を一時的にスカラー量として復元し，もう一方の秘匿化された秘密情報の多項式と乗算を行うことによって，多項式の次数変化をさせず，秘匿乗算を実現できる手法である．以下に TUS 方式について説明する．なお，秘密情報a,bは$a,b\in Z/pZ$であり，分散処理および秘匿加算で生成する乱数$\alpha_j,\beta_j,\gamma_j$も$\alpha_j,\beta_j,\gamma_j\in Z/pZ$である（ただし，秘密情報$a,b$は 0 を除く）．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．

【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】

ユーザAの秘密情報：a

1. ユーザAはk個の乱数$\alpha_0,\alpha_1,\dots,\alpha_{k-1}$生成し，乱数$\alpha=\prod_{j=0}^{k-1}\alpha_j$を計算する．
2. 計算された乱数αと秘密情報aをかけて，αaを計算し，$\alpha a,\alpha_0,\alpha_1,\dots,\alpha_{k-1}$を Shamir の$(k,n)$-閾値秘密分散法で$n$台のサーバ$S_i\ (i=0,1,2,\dots,n-1)$に分散する．
3. サーバ$S_i\ (i=0,1,2,\dots,n-1)$は秘密情報aに関する分散情報として$[a]_i=\left(\overline{[\alpha a]}_i,\overline{[\alpha_0]}_i,\dots,\overline{[\alpha_{k-1}]}_i\right)$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$[a]_j\ (j=0,1,\dots,k-1)$を収集する．
2. 収集した分散情報の$\overline{[\alpha a]}_j,\overline{[\alpha_0]}_j,\dots,\overline{[\alpha_{k-1}]}_j$から$\alpha a,\alpha_0,\dots,\alpha_{k-1}$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：$[a]_j=\left(\overline{[\alpha a]}_j,\overline{[\alpha_0]}_j,\dots,\overline{[\alpha_{k-1}]}_j\right)\ (j=0,1,\dots,k-1)$
$[b]_j=\left(\overline{[\beta b]}_j,\overline{[\beta_0]}_j,\dots,\overline{[\beta_{k-1}]}_j\right)\ (j=0,1,\dots,k-1)$

出力：$[c]_i=[a\pm b]_i=\left(\overline{[\gamma(a\pm b)]}_i,\overline{[\gamma_0]}_i,\dots,\overline{[\gamma_{k-1}]}_i\right)\ (i=0,1,\dots,n-1)$

1. k台のサーバS_jは$\overline{[\alpha_0]}_j,\dots,\overline{[\alpha_{k-1}]}_j$と$\overline{[\beta_0]}_j,\dots,\overline{[\beta_{k-1}]}_j$を収集し，$\alpha_j$と$\beta_j$を復元する．それから，$k$台のサーバ$S_j$は乱数$\gamma_j$を生成し，サーバ$S_0$に$\gamma_j/\alpha_j,\gamma_j/\beta_j$を送信する．
2. サーバS_0は$\gamma_j/\alpha_j,\gamma_j/\beta_j$を用いて，以下の式より$\gamma/\alpha,\gamma/\beta$を一時的に復元し，全サーバ$S_i$に送信する．

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

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．

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

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に分散する．
5. サーバ$S_i\ (i=0,1,2,\dots,n-1)$は秘密情報$c=a\pm b$に関する分散情報として$[c]_i=[a\pm b]_i=\left(\overline{[\gamma(a\pm b)]}_i,\overline{[\gamma_0]}_i,\dots,\overline{[\gamma_{k-1}]}_i\right)$を保持する．

【秘匿乗除算】

入力：$[a]_j=\left(\overline{[\alpha a]}_j,\overline{[\alpha_0]}_j,\dots,\overline{[\alpha_{k-1}]}_j\right)\ (j=0,1,\dots,k-1)$
$[b]_j=\left(\overline{[\beta b]}_j,\overline{[\beta_0]}_j,\dots,\overline{[\beta_{k-1}]}_j\right)\ (j=0,1,\dots,k-1)$

出力：$[c]_i=[ab]_i=\left(\overline{[\alpha\beta ab]}_i,\overline{[\alpha_0\beta_0]}_i,\dots,\overline{[\alpha_{k-1}\beta_{k-1}]}_i\right)\ (i=0,1,\dots,n-1)$

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$\overline{[\alpha_0]}_j,\dots,\overline{[\alpha_{k-1}]}_j$と$\overline{[\beta_0]}_j,\dots,\overline{[\beta_{k-1}]}_j$を収集し，$\alpha_j$と$\beta_j$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に分散する．
5. サーバ$S_i\ (i=0,1,2,\dots,n-1)$は秘密情報$c=ab$に関す

Vol.2017-CSEC-78 No.37
Vol.2017-SPT-24 No.37
2017/7/15

る分散情報として $[c]_i = [ab]_i = (\overline{[\alpha\beta ab]_i}, \overline{[\alpha_0\beta_0]_i}, \dots, \overline{[\alpha_{k-1}\beta_{k-1}]_i})$ 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 $[c]_i := (\overline{[\beta b/\alpha a]_j}, \overline{[\beta_0/\alpha_0]_i}, \cdots, \overline{[\beta_{k-1}/\alpha_{k-1}]_i})$ が得られる．

よって，上記秘匿加減算および秘匿乗除算は図 1 のように表現することができる．また，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，秘匿演算処理中の $k-1$ 台のサーバ情報，及び 1 つの入力または出力が漏洩しても安全であることが証明されている．

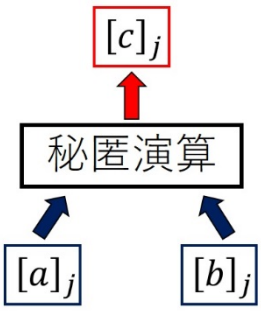

図 1　2 入力 1 出力の秘匿演算（TUS 方式）

2.1.2 TUS 方式の問題点

TUS 方式を組み合わせて，$f(x) = ab + c$ のような演算を連続的行うとき，攻撃者が演算結果を復元するために必要な情報，入力した秘密情報 b とそれを秘匿化に使う乱数 β および演算手順で漏洩する情報を知るとすれば，攻撃者は残りの入力者が入力した秘密情報を知ることができてしまうという問題がある．TUS 方式を組み合わせた積和演算を以下に示す．なお，秘密情報 a, b, c は $a, b, c \in Z/pZ$ であり，分散処理および秘匿加算で生成する乱数 $\alpha_j, \beta_j, \lambda_j, \gamma_j$ も $\alpha_j, \beta_j, \lambda_j, \gamma_j \in Z/pZ$ である（ただし，秘密情報が 0 を除く）．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．

【$ab + c$ の秘匿積和演算】

入力：$[a]_j = (\overline{[\alpha a]_j}, \overline{[\alpha_0]_j}, \dots, \overline{[\alpha_{k-1}]_j})$ $(j = 0, 1, \dots, k-1)$

$[b]_j = (\overline{[\beta b]_j}, \overline{[\beta_0]_j}, \dots, \overline{[\beta_{k-1}]_j})$ $(j = 0, 1, \dots, k-1)$

$[c]_j = (\overline{[\lambda c]_j}, \overline{[\lambda_0]_j}, \dots, \overline{[\lambda_{k-1}]_j})$ $(j = 0, 1, \dots, k-1)$

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

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 $\overline{[\alpha\beta ab]_i} = \alpha a \times \overline{[\beta b]_i}$ を計算する．
3. k 台のサーバ S_j は $\overline{[\alpha_0]_j}, \dots, \overline{[\alpha_{k-1}]_j}$ と $\overline{[\beta_0]_j}, \dots, \overline{[\beta_{k-1}]_j}$ 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 に分散する．
5. サーバ S_i $(i = 0, 1, 2, \dots, n-1)$ は秘密情報 ab に関する分散情報として $[ab]_i = (\overline{[\alpha\beta ab]_i}, \overline{[\alpha_0\beta_0]_i}, \dots, \overline{[\alpha_{k-1}\beta_{k-1}]_i})$ を保持する．
6. k 台のサーバ S_j は $\overline{[\alpha_0\beta_0]_j}, \dots, \overline{[\alpha_{k-1}\beta_{k-1}]_j}$ と $\overline{[\lambda_0]_j}, \dots, \overline{[\lambda_{k-1}]_j}$ 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 $\gamma_j/\alpha_j\beta_j, \gamma_j/\lambda_j$ を送信する．
7. サーバ S_0 は $\gamma_j/\alpha_j\beta_j, \gamma_j/\lambda_j$ を用いて，以下の式より $\gamma/\alpha\beta, \gamma/\lambda$ を計算し，全サーバ S_i に送信する．

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

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．

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

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

攻撃者が秘密情報 b の入力者かつ復元者である場合，攻撃者は入力者が入力した秘密情報 b，乱数 β，演算結果を復元ための乱数 γ，演算結果 $ab + c$ および $k-1$ 台のサーバから漏洩する $\alpha a, \gamma/\alpha\beta, \gamma/\lambda$ の情報を持っている．攻撃者は乱数 $\beta, \gamma, \gamma/\alpha\beta$ の情報を用いて，乱数 α を求めることができる．攻撃者は求めた乱数 α と秘匿化した秘密情報 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，攻撃者は入力者 B 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っていれば，残りの入力者の情報が漏洩してしまうという問題がある．

3. 提案方式

TUS 方式は多項式の次数変化をさせず，秘匿乗算を実現できるが，TUS の方式を使って，演算の連続がある場合，上記のように安全性の面で問題があった．そこで，本論文では，安全に演算が連続できるための条件を検討し，下記に示す 3 つの前提条件のもとで，TUS 方式を使って安全な秘匿演算が実現できることを示す．提案方式は以下の前提条件を持つ．

(1) 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十分に準備されている．
(3) 同一サーバセットでの演算であり，かつ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．

(1)の条件は TUS 方式が元々持つ条件である．(2)の条件が前記 TUS 方式の単純な組み合わせによる異なるタイプの演算の繰り返しを安全にするための主要な条件となる．TUS 方式において，攻撃者が元々知る情報及び $k-1$ 台のサーバの情報より残りの入力を求めることができることが問題である．よって，提案方式では，攻撃者が知らない情報を導入し，攻撃者が知る情報を用いても，残りの入力が漏

洩しないようにする．今回では，以下のように攻撃者が知らない乱数によって構成された 1 に対する分散値集合が準備されていると仮定する．

$$[1]^{(1)}{}_i = \left(\overline{[\delta]}_i, \overline{[\delta_0]}_i, \dots, \overline{[\delta_{k-1}]}_i\right) \quad (i = 0, 1, 2, \dots, n-1)$$
$$[1]^{(2)}{}_i = \left(\overline{[\eta]}_i, \overline{[\eta_0]}_i, \dots, \overline{[\eta_{k-1}]}_i\right) \quad (i = 0, 1, 2, \dots, n-1)$$

この 1 に対する分散値集合は例えば，以下の手順で簡単に生成できる．

【1 に対する分散値集合の生成】

1. k個の乱数$\delta_0, \delta_1, \dots, \delta_{k-1}$生成し，乱数$\delta = \prod_{j=0}^{k-1} \delta_j$を計算する．
2. $\delta, \delta_0, \delta_1, \dots, \delta_{k-1}$を Shamir の$(k, n)$-閾値秘密分散法で$n$台のサーバ$S_i$ $(i = 0, 1, 2, \dots, n-1)$に分散する．
3. サーバS_i $(i = 0, 1, 2, \dots, n-1)$は乱数 1 に関する分散情報として$[1]_i = \left(\overline{[\delta]}_i, \overline{[\delta_0]}_i, \dots, \overline{[\delta_{k-1}]}_i\right)$を保持する．

(3)は TUS 方式が 1 回の演算を想定しているのに対して，提案方式では演算の繰り返しを想定するため，それに対応するための条件である．例えば，秘匿乗算におけるサーバS_iと，秘匿加算におけるサーバS_iが異なるサーバS_jであった場合，サーバS_jは秘匿乗算における乱数と秘匿加算における異なる乱数を知ることができる（秘匿乗算でS_iはα_t, β_tを知り，秘匿加算でα_f, β_fを知るため，S_iは$\alpha_t, \beta_t, \alpha_f, \beta_f$を知る）．このような場合，$k-1$台の不正サーバから$k$個の分散値または乱数が漏洩する．また，1 回目の秘匿演算を行った後に，その結果を用いて 2 回目の秘匿演算を行う場合も，(3)の条件が満たされていれば，1 つのサーバは常に同じ分散値を扱うため秘密情報は漏洩しない．ただし，1 回目の演算に用いられたサーバが故障して異なるサーバを用いて 2 回目の演算を行う場合，1 回目の演算において$k-1$台のサーバが不正サーバであれば，2 回目の演算に用いられるサーバは正当なサーバである（1 つのサーバセット中$k-1$を超える不正サーバはない）ので，問題は生じない．

以下に提案方式の具体的アルゴリズムを示す．各サーバは 2.1.1 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．

【秘匿積和演算】

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

出力：$[d]_i = [ab + c]_i =$
$\left(\overline{[\gamma'(ab + c)]}_i, \overline{[\gamma'_0]}_i, \dots, \overline{[\gamma'_{k-1}]}_i\right)$ $(i = 0, 1, \dots, n-1)$

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をスカラー量として復元する．
2. 全サーバS_iにαaを送信する．
3. 全サーバS_iはαaと$\overline{[\beta b]}_i$の乗算を行い，$\overline{[\alpha\beta ab]}_i = \alpha a \times \overline{[\beta b]}_i$を計算する．
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$\alpha\beta ab, \lambda c$のスカラー量を復元する．
5. 全サーバS_iに$\alpha\beta ab, \lambda c$を送信する．
6. 全サーバS_iは$\alpha\beta ab$と$[1]^{(1)}{}_i$中の$\overline{[\delta]}_i$との乗算を行い，$\overline{[\alpha\beta\delta ab]}_i = \alpha\beta ab \times \overline{[\delta]}_i$を計算する．
7. 全サーバS_iはλcと$[1]^{(2)}{}_i$中の$\overline{[\eta]}_i$との乗算を行い，$\overline{[\lambda\eta c]}_i = \lambda c \times \overline{[\eta]}_i$を計算する．
8. k台のサーバは各々$\overline{[\alpha_j]}_l, \overline{[\beta_j]}_l, \overline{[\lambda_j]}_l, \overline{[\delta_j]}_l, \overline{[\eta_j]}_l$ $(l = 0, \dots, k-1)$を収集し，$\alpha_j, \beta_j, \lambda_j, \delta_j, \eta_j$を復元し，乱数$\gamma_j$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．
$$\frac{\gamma}{\alpha\beta\delta} = \prod_{j=0}^{k-1} \frac{\gamma_j}{\alpha_j\beta_j\delta_j} \quad , \quad \frac{\gamma}{\lambda\eta} = \prod_{j=0}^{k-1} \frac{\gamma_j}{\lambda_j\eta_j}$$
11. 全サーバS_iは$\overline{[\gamma(ab + c)]}_i = \gamma/\alpha\beta\delta\left(\overline{[\alpha\beta\delta ab]}_i\right) + \gamma/\lambda\eta\left(\overline{[\lambda\eta c]}_i\right)$を計算する．
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$\gamma_j$を Shamir の$(k, n)$-閾値秘密分散法で全サーバ$S_i$に分散する．
13. サーバS_i $(i = 0, 1, 2, \dots, n-1)$は秘密情報$ab + c$に関する分散情報として
$[ab + c]_i = \left(\overline{[\gamma(ab + c)]}_i, \overline{[\gamma_0]}_i, \dots, \overline{[\gamma_{k-1}]}_i\right)$を保持する．

上記積和演算において$c = 0$とすれば秘匿乗算になり 4.–7.，9.–11.を省略できる．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，$a = 1\left(\alpha \text{も} 1 \text{とする}\right)$とすれば秘匿加算となり，1.–3.を省略できる．ただし，4.–7.は TUS 方式の秘匿乗算と秘匿加算の組み合わせにない部分であり，この処理によって後述のように安全性が補強される．また，8.以降では必要な乱数のみ復元・計算し，用いていない乱数は 1 として扱う．さらに，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四則演算の組み合わせが実現できることが言える．

4. 安全性の評価

4.1 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 2 入力 1 出力のボックスで表せる．図中のボックスにはn台のサーバがあり，その中で 2.1.1 に示される秘匿演算を行う．TUS 方式では，攻撃者をボックス中の$k-1$台のサーバ情報を知ることができる攻撃者 1，$k-1$台のサーバ情報に加えて 1 つの入力（1 つの秘密情報及びその秘匿化に用いられた乱数）を知る攻撃者 2，$k-1$台のサーバ情報に加えて出力を知ることができる攻撃者 3 について，その安全性

Vol.2017-CSEC-78 No.37
Vol.2017-SPT-24 No.37
2017/7/15

を示した．ただし，TUS 方式は 2 入力演算であるので，1 つの入力と出力を知ることができる攻撃者 4 は想定していない．なぜならば，2 入力演算では 1 つの入力と出力を知れば必然的にもう 1 つの秘密情報を知ることができるためである．

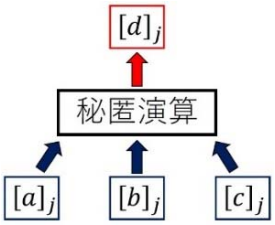

図 2　3 入力 1 出力の秘匿積和演算（提案方式）

提案方式は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．3 入力演算の場合，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．よって，安全な秘匿積和演算を以下の 2 つの安全性を満たす方式として定義する．

安全性 1：a, b, c中の任意の 1 入力（とそれを秘匿する乱数）と出力$d = ab + c$（とそれを秘匿する乱数）が漏洩しても，残りの 2 入力（とそれらを秘匿する乱数）は漏洩しない．

安全性 2：a, b, c中の任意の 2 つ入力（とそれらを秘匿する乱数）が漏洩しても，残りの 1 入力（とそれを秘匿する乱数）と出力d（とそれを秘匿する乱数）は漏洩しない．

例えば，2.1.1 に示した積和演算を図 2 のボックスとすると，a, b, c中の任意の 1 入力である$[b]_j$（秘密情報bとそれに用いられる乱数βからなる）と出力$[d]_j$（出力$d = ab + c$とそれを復元するための乱数γからなる）が漏洩した場合，ボックス内にあるn台のサーバ中$k-1$台のサーバから得られる情報によって乱数αとαaが漏洩するため，秘密情報aが漏洩し，攻撃者が持つ秘密情報と演算結果からもう 1 つの秘密情報cが漏洩するため安全性 1 が満たされていない（安全性 2 は満たされる）．よって，2.1.1 に示した積和演算は安全でないことが言える．

提案方式も秘密分散を用いているために，ボックス内にあるn台のサーバ中$k-1$台のサーバ情報が漏洩するという同様の前提および提案方式のアルゴリズムに示した 3 つの前提条件のもとで，上記 2 つの安全性を実現できることを以下に証明する．

【定理 1】

3 章に示した秘匿積和演算$ab + c$は，a, b, c中の任意の 1 入力（とそれを秘匿する乱数）と出力$d = ab + c$（とそれを秘匿する乱数）が漏洩しても，残りの 2 入力（とそれらを秘匿する乱数）は漏洩しない．

〈証明〉

a, b, c中の任意の 1 入力であるbおよびその出力$d = ab + c$が漏洩した場合，攻撃者は秘密情報aとそれを秘匿化する乱数αを知ることができる．よって，攻撃者は入力情報であるである$b, \beta, \beta_i\ (i = 0, \dots, k-1)$を知り，1.−2.において$\alpha a$，4.–5. において$\alpha\beta$ab,λc，8. において$\alpha_j, \beta_j, \lambda_j, \delta_j, \eta_j\ (j = 0, \dots, k-2)$，10.において$\gamma/\alpha\beta\delta\ , \gamma/\lambda\eta$を知り，出力結果の復元時に$\gamma_i, \gamma, ab + c$を知る．よって，攻撃者は，$b, \beta, \alpha a, \alpha\beta ab, \lambda c, \gamma/\alpha\beta\delta,\ \gamma/\lambda\eta\ , \gamma, \beta_i, \gamma_i\ (i = 0, \dots, k-1), \alpha_j, \beta_j, \lambda_j, \delta_j, \eta_j\ (j = 0, \dots, k-2), ab + c$から残りの秘密情報である$a, c$を求めようとする．

上記パラメータを次のように整理できる．$b, \beta, \alpha a, \lambda c, \alpha\delta, \lambda\eta, \gamma, \alpha_j, \lambda_j, \delta_j, \eta_j\ (j = 0, \dots, k-2), ab + c$から秘密情報$a, c$が漏洩するかを考える．

αaより秘密情報aを知るために，αを求める必要がある．αに関連する情報は$\alpha a, \alpha\delta, \alpha_j, \delta_j$である．しかし，これらの情報より$\alpha$が漏洩せず，秘密情報$a$も漏洩しないことが言える．ゆえに以下の式が成り立つ．

$$H(\alpha) = H\big(\alpha|\alpha_j\ (j = 0, \dots, k-2)$$
$$H(\delta) = H\big(\delta|\delta_j\ (j = 0, \dots, k-2)\big)$$
$$H(a) = H\big(a|\alpha a, \alpha\delta, \alpha_j, \delta_j\ (j = 0, \dots, k-2)\big)$$

一方，TUS 方式においては攻撃者が知らない乱数 1 であるδが導入されなかったので，$\alpha\delta$がαになり，秘密情報aが漏洩してしまう．

また，λcより秘密情報cを知るために，λを求める必要がある．秘密情報cに対しても同様に言える．よって，以下の式が成り立つ．

$$H(\lambda) = H\big(\lambda|\lambda_j\ (j = 0, \dots, k-2)$$
$$H(\eta) = H\big(\eta|\eta_j\ (j = 0, \dots, k-2)\big)$$
$$H(c) = H\big(c|\lambda c, \lambda\eta, \lambda_j, \eta_j\ (j = 0, \dots, k-2)\big)$$

秘密情報cに関しても攻撃者が知らない乱数ηを導入することによって，秘密情報cが漏洩しないことが言える．

次に，攻撃者は演算結果の復元時に$d = ab + c$を知るが，秘密情報a, cのどちらがわからなければ，bおよび$ab + c$がわかっても，残りの秘密情報を特定することができないため，以下の式が成り立つ．

$$H(a) = H\big(a|ab + c, \beta, b, \alpha a, \alpha\delta, \lambda c, \lambda\eta, \alpha_j, \delta_j,$$
$$\lambda_j, \eta_j\ (j = 0, \dots, k-2)\big)$$
$$(c) = H\big(c|ab + c, \beta, b, \alpha a, \alpha\delta, \lambda c, \lambda\eta, \alpha_j, \delta_j,$$
$$\lambda_j, \eta_j\ (j = 0, \dots, k-2)\big)$$

また，a, b, c中の入力aまたはcが漏洩したとしても同様の議論が成り立つ．そのため，提案方式は定理 1 を満たしていて，3 入力のうち 1 入力および出力が漏洩しても，残り 2 入力を漏洩しないことが言える．

また，上記の攻撃者に対して，攻撃者が知る 1 入力が定数などで既知である場合，TUS 方式で議論した攻撃者 3 と等価であることが考えられ，安全性を持つことが言える．

Vol.2017-CSEC-78 No.37
Vol.2017-SPT-24 No.37
2017/7/15

【定理 2】

3 章に示した秘匿積和演算$ab+c$は，a,b,c中の任意の 2 入力（とそれらを秘匿する乱数）が漏洩しても，残り 1 入力（とそれを秘匿する乱数）と出力d（とそれを秘匿する乱数）は漏洩しない．

〈証明〉

a,b,c中の任意の 2 入力であるa,bが漏洩した場合，攻撃者は秘密情報a,bとそれを秘匿化する乱数α,βを知ることができる．よって，攻撃者は入力情報であるである$a,b,\alpha,\beta,\alpha_i,\beta_i\ (i=0,\dots,k-1)$を知り，1.–2.において$\alpha a$，4.–5.において$\alpha\beta$ab,λc，8.において$\alpha_j,\beta_j,\lambda_j,\delta_j,\eta_j,\gamma_j\ (j=0,\dots,k-2)$，10.において$\gamma/\alpha\beta\delta,\gamma/\lambda\eta$を知る．よって，攻撃者は，$a,b,\alpha,\beta,\alpha a,\alpha\beta ab,\lambda c,\gamma/\alpha\beta\delta,\ \gamma/\lambda\eta,\alpha_i,\beta_i\ (i=0,\dots,k-1),\alpha_j,\beta_j,\lambda_j,\delta_j,\eta_j,\gamma_j\ (j=0,\dots,k-2)$から残りの秘密情報である$c$および出力$ab+c$を求めようとする．

上記パラメータを次のように整理できる．$a,b,\alpha,\beta,\lambda c,\gamma/\delta,\ \gamma/\lambda\eta,\lambda_j,\delta_j,\eta_j,\gamma_j\ (j=0,\dots,k-2)$から秘密情報$c$および出力結果$ab+c$が漏洩するかを考える．

λcより秘密情報cを知るために，λを求める必要がある．λに関連する情報は$\lambda c,\gamma/\lambda\eta,\lambda_j,\eta_j,\gamma_j$である．しかし，これらの情報より$\lambda$が漏洩せず，秘密情報$c$も漏洩しないことが言える．ゆえに以下の式が成り立つ．

$$H(\lambda)=H\big(\lambda|\lambda_j\ (j=0,\dots,k-2)\big)$$
$$H(\eta)=H\big(\eta|\eta_j\ (j=0,\dots,k-2)\big)$$
$$H(\gamma)=H\big(\gamma|\gamma_j\ (j=0,\dots,k-2)\big)$$
$$H(c)=H\big(c|\lambda c,\gamma/\lambda\eta,\lambda_j,\eta_j,\gamma_j\ (j=0,\dots,k-2)\big)$$

一方，TUS 方式においては攻撃者が知らない乱数 1 であるη,δが導入されなかったので，$\gamma/\lambda\eta$がγ/λになり，$\gamma/\alpha\beta\delta$が$\gamma/\alpha\beta$になる．元々に漏洩したα,βの情報および$\gamma/\alpha\beta$よりγが漏洩し，γ/λとγよりλが漏洩し，秘密情報cが漏洩してしまう．

また，攻撃者は$\gamma(ab+c)_j\ (j=0,\dots,k-2)$を知るが，それらの情報より$\gamma(ab+c)$が漏洩しないので，次の式が成り立つ．

$$H(\gamma(ab+c)=H\big(\gamma(ab+c)|\gamma(ab+c)_j\ (j=0,\dots,k-2)\big)$$

また，演算結果を暗号化している乱数γに関連する情報は，$\gamma/\delta,\ \gamma/\lambda\eta,\lambda_j,\delta_j,\eta_j,\gamma_j\ (j=0,\dots,k-2)$である．しかし，それらの情報より$\gamma$が漏洩しないことが言える．よって，次式が成り立つ．

$$H(\lambda)=H\big(\lambda|\lambda_j\ (j=0,\dots,k-2)\big)$$
$$H(\delta)=H\big(\delta|\delta_j\ (j=0,\dots,k-2)\big)$$
$$H(\eta)=H\big(\eta|\eta_j\ (j=0,\dots,k-2)\big)$$
$$H(\gamma)=H\big(\gamma|\gamma/\delta,\ \gamma/\lambda\eta,\lambda_j,\delta_j,\eta_j,\gamma_j\ (j=0,\dots,k-2)\big)$$

それに対して，TUS 方式においては，攻撃者が知らない乱数であるδ,ηが導入されなかったので，$\gamma/\lambda\eta$がγ/λになり，γ/δがγになり，γが漏洩したことがわかる．一方，提案方式では，攻撃者が知らない乱数の導入によって，γを漏洩しないことが言える．

最後に，攻撃者が秘密情報a,bを知るが，出力結果$ab+c$またはcのどちらが分からなければ，残りの入力及び出力結果を特定できない．よって，以下が言える．

$$H(c)=H\big(c|a,b,\alpha,\beta,\gamma/\delta,\ \gamma/\lambda\eta,\lambda_j,\delta_j,\eta_j,\gamma_j,\gamma(ab+c)_j\ (j=0,\dots,k-2)\big)$$
$$H(ab+c)=H\big(ab+c|a,b,\alpha,\beta,\gamma/\delta,\ \gamma/\lambda\eta,\lambda_j,\delta_j,\eta_j,\gamma_j,\gamma(ab+c)_j\ (j=0,\dots,k-2)\big)$$

また，上記の議論は，b,cまたはa,cの入力が漏洩したときにも成り立つ．以上より，提案方式は定理 2 を満たすことができ，3 入力のうち，2 入力が漏洩しても残り 1 入力及び出力が漏洩しないことが言える．

また，上記の攻撃に対しては，攻撃者が知る 2 入力のうち 1 入力が定数などで既知である場合，TUS 方式で議論した攻撃者 2 と等価であることが考えられ，安全性を持つことが言える．

4.2 積和演算の組み合わせに関する安全性

一般に，四則演算からなる任意の演算は積和演算$f(a,b,c)=d$の組み合わせに分解できる．

例えば，$a=f(a_1,a_2,\dots,a_{2m},a_{2m+1})=a_{2m+1}(a_1a_2+a_3a_4+\cdots+a_{2m-1}a_{2m})$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合せて表せば，図 3 のようになる（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）．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を前記のように分解して演算すれば，図 4 の中身が図 3 であるともいえる．

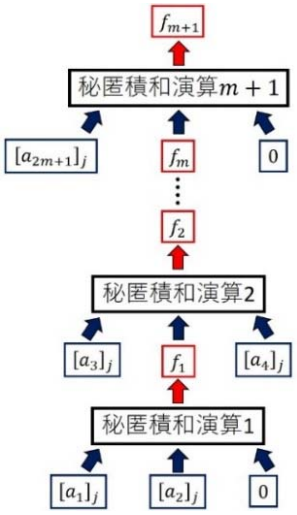

図 3　積和演算の組み合わせからなるaの演算

Vol.2017-CSEC-78 No.37
Vol.2017-SPT-24 No.37
2017/7/15

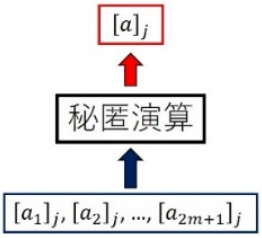

図 4　aの演算に関する一般的な積和演算ボックス

例えば，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．これを図 3 及び安全性 1，安全性 2 を用いて説明すると，図 3 においてa_1を入力とするボックスは最初の積和演算ではその出力を公開しないため，a_2, a_3が漏洩していても安全性 2 よりa_1は漏洩しない．残りのボックスにおいてもa_1を含むボックスからの出力が秘匿化されていれば，その入力も秘匿化される．しかし，最終ボックスにおいてその出力を攻撃者が得られるとすれば，3 入力中のa_1を含むボックス入力以外の 2 入力からa_1を含むボックスの出力も漏洩し，最終的にa_1が漏洩する．

一方，用いる秘匿演算が安全であるならば，一般的にaの演算において 2 つの入力（例えば，a_1, a_2）が漏洩していなければ，その 2 入力は漏洩しないことが言える．それは，図 4 では$a_1 = f^{-1}(a, a_2, \ldots, a_{2m}, a_{2m+1})$となるため，$a_2$が分からなければ$a_1$を特定できず，その逆も言えるためである．これを図 3 と安全性 1，安全性 2 によって説明する場合，a_1, a_2を含むボックス以外は前述のように全入力が特定されるが，a_1, a_2を含むボックスではその出力とa_1, a_2以外の入力が分かっても，ボックスで用いられる秘匿演算が安全であれば安全性 2 から残りの 2 入力(a_1, a_2)は漏洩しないことが言える．

また，図 3 を用いると，各ボックスに 1 つ以上の入力が漏洩していなければその出力は漏洩せず，最終ボックスでその出力を攻撃者が得たとしても，全入力が漏洩しないため元から漏洩している入力以外は漏洩しない．よって，以下の安全性を定義する．

安全性 3：複数の積和演算の組み合わせからなる演算において，2 つ以上の入力（とそれらを秘匿する乱数）が漏洩しなければ，秘匿化された入力は漏洩しない．

以下に提案方式の組み合わせが安全性 3 を満たすことを証明する．

【定理 3】

提案方式において，複数の積和演算の組み合わせからなる演算では，2 つ以上の入力（とそれらを秘匿する乱数）が漏洩しなければ，秘匿化された入力は漏洩しない．

〈証明〉

提案方式が定理 3 を満たすことを言うために以下の 2 つの場合に分けて考える．

(i). 1 つの積和演算ボックスに秘匿かされた 2 入力が入力され，後の入力は漏洩している場合．

1 つのボックスに秘匿された 2 入力が入力され，残りの入力が漏洩する．すなわち，1 つの積和演算ボックスの 3 入力のうち 1 入力および出力が漏洩する場合と同じである．提案方式が定理 1 を満たすので，1 つのボックスにおいて，1 入力及び出力が漏洩しても，残りの入力が漏洩しないことが言える．

また，この積和演算ボックスが例えば図 3 に示す積和演算ボックス 2 であるとした場合，残りの入力が漏洩するが，ボックス 2 の積和演算は定理 2 を満たしているので，残りの 2 入力が漏洩しないことが言える．

(ii). 秘匿化された 2 入力が異なるボックスに入力され，後の入力は漏洩している場合．

秘匿化された 2 入力が異なるボックスに入力される．すなわち，3 入力のうち 2 入力が漏洩する．例えば，秘匿化された 2 入力がそれぞれ図 3 に示すボックス 1 およびボックス 2 に入力された場合，ボックス 1 の積和演算が定理 2 を満たしているので，2 入力が分かっても，残りの 1 入力及び出力が漏洩せず，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 において，3 入力のうち 2 入力が漏洩しない．ボックス 2 における積和演算が定理 1 を満たしているので，残りの 2 入力が漏洩しないことが言える．

5. 考察

今回の提案方式とともに提案した 3 つの前提条件の実現性について検討する．

提案方式が安全であるための 3 つの前提条件の 1 つ目は，秘密情報に 0 を含まないことである．この理由として，秘密情報が 0 になるとき，秘匿乗算においてαa=0 となり，秘密情報が 0 であることが漏洩してしまう（乱数として 0 は選択されない）ためである．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．例えば，脈拍や血圧などは全て正の値であり，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．0 を含む場合，前述のように秘匿乗算においてのみ問題が生じるが，秘匿加減算ではαaのように途中で復元されないため 0 を含んでも問題ない．さらに，除算において 0 で割ることは禁止されており，演算途中で分母となる数が 0 であるか検証する必要がある．提案方式による秘匿除算は秘匿乗算と同様に秘密情報が 0 であれば，$\alpha a = 0$となり検出されるが，これは 0 による除算を防ぐという意味から問題はない．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

るため，その実現性に問題はないと言える．

次の前提条件は，各サーバ上に異なる乱数からなる 1 に対する分散値集合が十分に準備されてあることである．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（機関）が 1 に対する分散値集合を提供するとすることである．また，信頼できる第三者でなくても，1 に対する分散値集合は生成が容易であり入力に依存しないので，多くのユーザから広くそれを集め，サーバ内でシャッフルなどをして 1 に対する分散値集合の生成者と紐付けをできなくするなども考えられる．よって，この条件の回避は今後の課題であるが，現実問題としてその実現性は問題ないと考えられる．

最後に，演算がすべて同一サーバセットであり，かつ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ことである．この条件に関して提案法は決められたプロトコルに従う Passive Adversary を仮定するため，規則を決めれば実現に関するハードルは低いと言える．

6. 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十分に準備されている．
(3) 同一サーバセットでの演算であり，かつ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．

今後の課題として上記の条件を回避する方法を検討したいと考える．

参考文献

[1] Beaver D.: “Commodity-based cryptography.” In Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing (STOC ’97). ACM. El Paso, Texas, USA, pp. 445-455. (1997)
[2] Ben-Or M., Goldwasser S., Wigderson A.: “Completeness Theorems for Non-Cryptographic Fault-Tolerent Distributed Computation.” In Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing (STOC ’88). ACM, New York, NY, USA, pp. 1-10. (1988)
[3] Blakley G. R.: “Safeguarding Cryptographic Keys.” In Proceedings of the AFIPS 1979 National Computer Conference, vol. 48, pp. 313-317(1979)
[4] 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)
[5] Brakerski Z., Vaikuntanathan V.: “Fully Homomorphic Encryption from Ring-LWE and Security for Key Dependent Messages.” In: Rogaway P. (eds) Advances in Cryptology – CRYPTO 2011. CRYPTO 2011. Lecture Notes in Computer Science, vol 6841. Springer, Berlin, Heidelberg. (2011)
[6] Chaum D., Crépeau C., Damgård I.: “Multiparty Unconditionally Secure Protocols.” In Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing (STOC ’88), ACM, New York, NY, USA, pp. 11-19. (1988)
[7] 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)
[8] 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)
[9] Cleve R.: “Limits on The Security of Coin Flips When Half the Processors are Faulty.” In 18th Annual ACM Symposium on Theory of Computing (STOC ’86), pp. 364-369, ACM Press. (1986)
[10] Cramer R., Damgård I., Maurer U.: “General Secure Multi-Party Computation from any Linear Secret-Sharing Scheme.” In Preneel B. (eds) Advances in Cryptology-EUROCRYPT 2000. Lecture Notes in Computer Science, vol 1807, pp. 316-334. Springer, Berlin, Heidelberg. (2000)
[11] Damgård I., Ishai Y., Krøigaard M.: “Perfectly Secure Multiparty Computation and the Computational Overhead of Cryptography.” In Gilbert H. (eds) Advances in Cryptology-EUROCRYPT 2010. Lecture Notes in Computer Science, vol. 6110, pp. 445-465. Springer, Berlin, Heidelberg. (2010)
[12] Damgård I., Pastro V., Smart N., Zakarias S.: “Multiparty Computation from Somewhat Homomorphic Encryption.” In Safavi-Naini R., Canetti R., (eds) Advances in Cryptology-CRYPTO 2012. Lecture Notes in Computer Science, vol 7417, pp. 643-662. Springer, Berlin, Heidelberg, (2012)
[13] Damgård I., Keller M., Larraia E., Pastro V., Scholl P., Smart N.P.: “Practical Covertly Secure MPC for Dishonest Majority – Or: Breaking the SPDZ Limits.” In: Crampton J., Jajodia S., Mayes K. (eds) Computer Security – ESORICS 2013. ESORICS 2013. Lecture Notes in Computer Science, vol. 8134. Springer, Berlin, Heidelberg. (2013)
[14] Gennaro R., Rabin M. O., Rabin T.; “Simplified VSS and Fast-Track Multiparty Computations with Applications to Threshold Cryptography.” In Proceedings of the seventeenth annual ACM Symposium on Principles of Distributed Computing (PODC ’98). ACM, New York, NY, USA, pp. 101-111. (1998)
[15] Gentry C.: A Fully Homomorphic Encryption Scheme, Ph.D Thesis, Stanford University, Stanford, CA, USA, Sept 2009
[16] 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)
[17] 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)
[18] Shamir A.: “How to Share a Secret”, Communications of the ACM, 22, (11), pp. 612-613, (1979)
[19] Shingu T., Iwamura K., Kaneda K.: “Secrecy Computation without Changing Polynomial Degree in Shamir’s (k,n) Secret Sharing Scheme.” In Proceedings of the 13th International Joint Conference on e-Business and Telecommunications-Volume 1: DCNET, pp. 89-94. (2016)
[20] Watanabe T., Iwamura K., Kaneda K.: “Secrecy Multiplication Based on a (k,n)-Threshold Secret-Sharing Scheme Using Only k Servers.” In Park J., Stojmenovic I., Jeong H., Yi G. (eds) Computer Science and Its Applications. Lecture Notes in Electrical Engineering, vol. 330, pp. 107-112. Springer, Berlin, Heidelberg, (2015)
[21] Yao A. C.: "Protocols for secure computations." 23rd Annual Symposium on Foundations of Computer Science (sfcs 1982), Chicago, IL, USA, 1982, pp. 160-164. (1982)