Vol.2019-CSEC-86 No.42
Vol.2019-SPT-34 No.42
2019/7/24

n<2k-1 において計算結果の正当性を検証可能な秘密分散を用いた秘匿計算

落合将吾　岩村惠市

概要：鴇田らによって n<2k-1 においても実行可能な秘密分散を用いた秘匿計算法が提案されている．しかし，その手法は passive な攻撃者を想定しており，active な攻撃者に対しては安全ではない．そこで，n<2k-1 における秘密分散を用いた秘匿計算において，計算結果が正しいことを検証できる秘匿計算法を提案する．この手法は active な攻撃者に対しても安全であり，効率的な秘匿計算を実現する．

キーワード：秘匿計算，　秘密分散，　マルチパーティ計算，　active，malicious

Secure Multiparty Computation for n<2k-1 based on Secret Sharing Scheme

SHOGO OCHIAI　KEIICHI IWAMURA

1. はじめに

クラウドシステムの発展に伴いビッグデータを構成する多種多様なデータを利活用してさらなる付加価値の創出が期待されているが，ビッグデータの利活用では情報保護に注意すべきである．ビッグデータには個人のプライバシー情報や企業の機密情報なども含まれ，それらはビッグデータの利活用者に対して漏洩されずに扱われるべきである．これを実現する手法の一つがデータを暗号化したまま計算を行える秘匿計算という技術である．

秘匿計算は準同型暗号や秘密分散法などを用いて実現できる．秘密分散法に関しては，1979 年に Shamir により提案された多項式補間を用いた(k,n)閾値秘密分散法[1](以降，Shamir 法)が有名である．一般的な(k,n)閾値秘密分散法は，秘密情報からn個の分散値を生成してそれらをn台の各サーバに 1 個ずつ配布する手法であり，以下の 2 つの性質(定義)を持つ．ただし，$n \geq k$である．

(1) n個の分散値のうちk個以上の分散値から秘密情報を復元できる．

(2) k個未満の分散値から秘密情報に関する情報は一切得られない．

Shamir 法では分散値が多項式から生成されるため，秘匿乗算を行うと次数変化が起こる．具体的には，乗算を 1 回行うと閾値kが$2k-1$へ増加するため$n < 2k-1$においては乗算結果が復元できないという問題があった．

この問題に対して，著者らと同一の研究グループである神宮ら，Aminuddin ら，鴇田らによって TUS（TUS1[2],TUS2[3],TUS3[4]）方式が提案された．TUS 方式は，乱数で秘密情報を秘匿化してスカラー量として乗算するというアプローチで次数変化のない秘匿乗算を実現している．しかし，TUS 方式は全て passive な攻撃者を想定しており，プロトコルから逸脱する active な攻撃者に対しては安全ではない．active な攻撃者に対する安全性は，分散値や復元値などの正当性を検証することで実現されており，検証可能な完全準同型暗号(VFHE)，検証可能な秘密分散法(VSS)，マルチパーティ計算(MPC)などとして提案されている．マルチパーティ計算とは、複数人のプレイヤーが持つプライベートな値を入力とする関数fを計算し、各プレイヤーはその関数fの出力のみを知ることができる技術である。近年注目されている Damgård らが提案した MPC 手法である SPDZ(SPDZ-1[9],SPDZ-2[10])は，Shamir 法の問題点であった秘匿乗算における次数変化が起こらず，MAC 認証やコミットメントを用いることで復元値の正当性を検証するが，全ての分散値の総和を秘密情報とする加法的秘密分散を用いているため，$n = k$という条件がある．

そこで，本論文では TUS 方式のアプローチを採用し，秘匿乗算における次数変化が起こらず，$n < 2k-1$で適用でき，かつ分散値や復元値の正当性を検証することで active な攻撃者に対して安全な方式を提案する．

以下，本論文の構成を示す．まず，2 章では従来方式として(k,n)閾値秘密分散法，TUS3 方式，従来の active な攻撃者に対する秘匿計算に関する説明を行う．3 章では active な攻撃者に対して安全な秘匿計算方法を提案する．4 章では安全性の議論を行い，5 章をまとめとする．

2. 従来方式

2.1. (k, n)閾値秘密分散法

(k,n)閾値秘密分散法は，パラメータ(k,n)に対して以下の 2 つの条件を満たす．
(1)k個未満の分散値からは，秘密情報に関する情報は一切得られない．
(2)任意のk個以上の分散値からは，秘密情報が一意に得られる．
代表例として，Shamir の多項式補間による方式 (Shamir 法), 栗原らの XOR による方式[5][6](XOR 法)などが提案されている．秘密分散法によりセキュリティの 3 要素の 1 つである機密性を実現できる．機密性は準同型暗号などの暗号化方式でも実現できるが，一般に(k,n)閾値秘密分散法のほうが遥かに計算量が少なく高効率である[7].

2.2. TUS3 方式

秘密分散法を用いた秘匿計算に関して，Shamir 法では加減算および乗算が可能であるが，乗算を行うと閾値がkから$2k-1$へ増加してしまう．この問題に対して，神宮らはスカラー倍では次数変化が起こらないことに着目し，秘密分散を用いた次数変化のない秘匿乗算を実現した(TUS1)．しかし，この方式は加減算と乗算の組み合わせ(積和演算)を行うと安全性に問題が生じた．この問題を AMINUDDIN らは攻撃者が知らない”1”の分散値集合を用いて解決した(TUS2)．この方式は，秘密情報に 0 は含まないなど３つの条件はあるが，情報理論的安全性をもつ．しかし，Shamir 法における分散値の復元が多いため，処理速度に問題があった．そこで鴇田らは，栗原らの提案した秘匿演算はできないが高速な XOR 法に着目し，分散値に対する演算を行わない部分を XOR 法に置き換えることにより，高速処理を実現した(TUS3)．TUS3 方式で生成される乱数や秘密情報はすべて有限体F_p上の値であり，全ての演算はpを法として行われる．また，TUS3 が提案された論文[4]では 2 つの提案方式が存在するが，ここでは秘密情報が 0 の場合に対応している２つ目のプロトコルを示す．
[前提条件]
(1) 生成される乱数はすべて非 0 とする．
(2) 攻撃者が知らない”1”の分散値集合がある．
(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．
[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に関するサーバS_iが保持する分散値集合．
2.2.1.分散処理と復元処理
[分散処理]
① 入力者は，k個の乱数$\alpha_0, \alpha_1, \dots, \alpha_{k-1}$ 生成し，XOR 法でn台のサーバに秘密分散する．
② 入力者は，乱数$\alpha = \sum_{j=0}^{k-1} \alpha_j$を計算し，秘密情報$a = \{0,1,\dots,p-2\}$に 1 を加算し，$(a+1)$に乱数$\alpha$を乗じて秘匿化秘密情報$\alpha(a+1)$を計算して$n$台のサーバにブロードキャストする．
③ サーバS_iは秘密情報sに関する分散情報として，以下を保持する．

$$[a]_i = \left(\alpha(a+1), \overline{[\alpha_0]}_i, \dots, \overline{[\alpha_{k-1}]}_i\right)$$

[復元処理]
① 復元者は，k台のサーバS_jを選択し，それらが持つ分散値集合$[a]_j$を収集する．
② 復元者は，収集した分散値集合を復元し，$\alpha(a+1), \alpha_0, \dots, \alpha_{k-1}$を得る．
③ 復元者は，以下のように秘密情報sを復元する．

$$a = \alpha(a+1) \times \alpha^{-1} - 1$$

$$= \alpha(a+1) \times \left(\sum_{j=0}^{k-1} \alpha_j\right)^{-1} - 1$$

TUS3 方式は，秘匿積和演算が可能であり，以下に示す攻撃者 6 に対して安全であることが示されている．
攻撃者 6 : t入力 1 出力の演算に関して，$t-1$個以下の入出力を知る passive な攻撃者．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 つの未知な入出力を知ろうとする．

2.3. 従来の active な攻撃者に対する秘匿計算

Active（malicious とも呼ばれる）な攻撃者に対応するための検証機能は，セキュリティの 3 要素の 1 つである完全性を実現するために必要である．近年では，Damgård らが提案した active な攻撃者に対して安全なマルチパーティ計算手法である SPDZ(SPDZ-1[9],SPDZ-2[10])が注目されている．SPDZ は，秘密情報xと MAC 鍵αに対し MAC 値$m = \alpha x$を生成し，復元値x'を用いて$\alpha x' = m$を検証することで復元値の正当性を確認する手法であり，その検証過程でコミットメント方式も用いられる．また，Shamir 法の問題点であった秘匿乗算による次数変化を，乗算用組$(a, b, c = ab)$を用いることで解決している．SPDZ を改良した方式(SPDZ family)として，Keller らによる紛失通信を用いた MASCOT[11]や，Cramer らによる環$\mathbb{Z}_{2^k}$で適用可能な SPD$\mathbb{Z}_{2^k}$[12]があるが，これらの手法の共通点は加法的秘密分散を用いることで，そのため$n = k$でしか適用できない．

3. 提案方式

3.1. 概要

本方式では，２つの乱数を用いて１つの秘密情報を加算と乗算で秘匿化し公開情報とする．また，加算に用いた乱数を別の乱数で秘匿し，これも公開情報とする．各サーバは，これらの乱数の断片を保持しており，復元や秘匿計算において使用する．また，本方式は$n \geq k$に適用できるので，加法的秘密分散を用いた従来方式[9][10][11][12]に対して可用性を高める．

3.1.1. 表記

$[x]_i$：xに対するサーバS_iが保持する Shamir 法での分散値
$[x]_i^X$：xに対するサーバS_iが保持する XOR 法での分散値

Vol.2019-CSEC-86 No.42
Vol.2019-SPT-34 No.42
2019/7/24

h_x：xに対するハッシュ値

$[x, y]$：関係の深いペア（見やすさのため）

S：分散値を保持するn台のサーバ集合

$$\mathrm{S} = \{S_0, S_1, \dots, S_{n-1}\}$$

P：秘匿積和演算に対する秘密情報の入力者または復元者

$$P = \{A, B, C, R\}, \qquad P \notin S$$

3.1.2. 前提

・生成される乱数はすべて非 0 とする．

・全ての演算は入力や計算結果に対して十分大きな素数pを法とした有限体F_pで行われ，生成される乱数や秘密情報はp未満である．

・S_iは事前に変換用乱数組$[\varepsilon_h]_i, \varepsilon_{h,i}$を保持している．$\varepsilon_h = \prod_{i=0}^{k-1} \varepsilon_{h,i} \quad h = 1, \dots,$必要個数

・変換用乱数組は毎回異なるものを使用する．

3.2. 分散

① ディーラは$4k$個の乱数$\alpha_{i,0}, \dots, \alpha_{i,k-1}\ (i = 1,..,4)$を生成し，乱数$\alpha_1, \alpha_2, \alpha_3, \alpha_4$を計算する．

$$\alpha_i = \prod_{j=0}^{k-1} \alpha_{i,j}\ (i = 1,..,4)$$

② ディーラは乱数a_1,a_2を生成し，それらのハッシュ値a_1, a_2を計算し，全サーバに送信する．

$$h_{a_1}, h_{a_2}$$

③ ディーラは以下を計算し，全サーバに送信する．また，$4k$個の乱数$\alpha_{i,0}, \dots, \alpha_{i,k-1}\ (i = 1,..,4)$を$n$台のサーバにXOR 法で秘密分散する。

$$\alpha_1 a_1 = \alpha_1 \times a_1$$
$$\alpha_2(a + a_1) = \alpha_2 \times (a + a_1)$$
$$\alpha_3 a_2 = \alpha_3 \times a_2$$
$$\alpha_4(a + a_2) = \alpha_4 \times (a + a_2)$$

分散処理終了後，サーバS_iは秘密情報aに対して以下を保持する。

$$\alpha_1 a_1, \alpha_2(a + a_1), \alpha_3 a_2, \alpha_4(a + a_2)$$

$$[\alpha_{1,j}]_i^X, [\alpha_{2,j}]_i^X, [\alpha_{3,j}]_i^X, [\alpha_{4,j}]_i^X,\ (j = 0,1, \dots, k-1)$$

$$h_{a_1}, h_{a_2}$$

3.3. 秘匿計算

本稿では 3 入力a, b, cから 1 出力$ab + c$を求める積和演算を考える．3 人のディーラA, B, Cによる 3 入力a, b, cに対する値を以下のように定義する．

[1] ディーラAが分散したaに対する値

$$\alpha_1 a_1, \alpha_2(a + a_1), \alpha_3 a_2, \alpha_4(a + a_2)$$

$$[\alpha_{1,j}]_i^X, [\alpha_{2,j}]_i^X, [\alpha_{3,j}]_i^X, [\alpha_{4,j}]_i^X,\ (j = 0,1, \dots, k-1)$$

$$h_{a_1}, h_{a_2}$$

[2] ディーラBが分散したbに対する値

$$\beta_1 b_1, \beta_2(b + b_1), \beta_3 b_2, \beta_4(b + b_2)$$

$$[\beta_{1,j}]_i^X, [\beta_{2,j}]_i^X, [\beta_{3,j}]_i^X, [\beta_{4,j}]_i^X,\ (j = 0,1, \dots, k-1)$$

$$h_{b_1}, h_{b_2}$$

[3] ディーラCが分散したcに対する値

$$\gamma_1 c_1, \gamma_2(c + \gamma_1), \gamma_3 c_2, \gamma_4(c + c_2)$$

$$[\gamma_{1,j}]_i^X, [\gamma_{2,j}]_i^X, [\gamma_{3,j}]_i^X, [\gamma_{4,j}]_i^X,\ (j = 0,1, \dots, k-1)$$

$$h_{c_1}, h_{c_2}$$

3.3.1 秘匿積和演算（演算中 0 が復元されない場合）

① サーバS_iは，乱数$\delta_{1,i}$を生成し，乱数の断片を復元し，以下ように乱数比の断片を計算してブロードキャストし，全サーバはその値を$i = 0,.., k-1$に対して乗算して乱数比を計算する．

$$\frac{\delta_{1,i}}{\gamma_{1,i}\varepsilon_{1,i}}, \frac{\delta_{1,i}}{\alpha_{1,i}\beta_{1,i}\varepsilon_{2,i}} \rightarrow \frac{\delta_1}{\gamma_1\varepsilon_1}, \frac{\delta_1}{\alpha_1\beta_1\varepsilon_2}$$

② サーバS_iは，$[\delta_1(c_1 - a_1 b_1)]_i$を計算してブロードキャストする．

$$[\delta_1(c_1 - a_1 b_1)]_i = \gamma_1 c_1 \times \left(\frac{\delta_1}{\gamma_1\varepsilon_1}\right) \times [\varepsilon_1]_i$$
$$-\alpha_1 a_1 \times \beta_1 b_1 \times \left(\frac{\delta_1}{\alpha_1\beta_1\varepsilon_2}\right) \times [\varepsilon_2]_i$$

③ 全サーバは，分散値から$\delta_1(c_1 - a_1 b_1)$を復元してブロードキャストする．復元値が 0 でなければ次の手順へ進む．※0 が復元された場合の処理は **3.3.2** で示す．

④ サーバS_iは，乱数$\delta_{2,i}$を生成し，乱数の断片を復元し，以下ように乱数比の断片を計算してブロードキャストし，全サーバはその値を$i = 0,.., k-1$に対して乗算して乱数比を計算する．

$$\frac{\delta_{2,i}}{\alpha_{2,i}\beta_{2,i}\varepsilon_{3,i}}, \frac{\delta_{2,i}}{\alpha_{2,i}\beta_{1,i}\varepsilon_{4,i}}, \frac{\delta_{2,i}}{\alpha_{1,i}\beta_{2,i}\varepsilon_{5,i}}, \frac{\delta_{2,i}}{\gamma_{2,i}\varepsilon_{6,i}}$$
$$\rightarrow \frac{\delta_2}{\alpha_2\beta_2\varepsilon_3}, \frac{\delta_2}{\alpha_2\beta_1\varepsilon_4}, \frac{\delta_2}{\alpha_1\beta_2\varepsilon_5}, \frac{\delta_2}{\gamma_2\varepsilon_6}$$

⑤ サーバS_iは，以下のように分散値を計算してブロードキャストする．

$$[\delta_2\{(ab + c) + (c_1 - a_1 b_1)\}]_i$$
$$= \alpha_2(a + a_1) \times \beta_2(b + b_1) \times \left(\frac{\delta_2}{\alpha_2\beta_2\varepsilon_3}\right) \times [\varepsilon_3]_i$$
$$-\alpha_2(a + a_1) \times \beta_1 b_1 \times \left(\frac{\delta_2}{\alpha_2\beta_1\varepsilon_4}\right) \times [\varepsilon_4]_i$$
$$-\alpha_1 a_1 \times \beta_2(b + b_1) \times \left(\frac{\delta_2}{\alpha_1\beta_2\varepsilon_5}\right) \times [\varepsilon_5]_i$$
$$+\gamma_2(c + c_1) \times \left(\frac{\delta_2}{\gamma_2\varepsilon_6}\right) \times [\varepsilon_6]_i$$

⑥ 全サーバは，分散値から$\delta_2\{(ab + c) + (c_1 - a_1 b_1)\}$を復元してブロードキャストする．

⑦ 同様にして，各サーバは$\delta_3(c_2 - a_2 b_2), \delta_4\{(ab + c) + (c_2 - a_2 b_2)\}$を得る。

⑧ サーバS_iは，$\delta_{1,i},\delta_{2,i},\delta_{3,i},\delta_{4,i}$を$n$台のサーバに XOR 法で秘密分散する。

1 回の積和演算終了後，サーバS_iは，演算結果$ab+c$に対して以下を保持する．

$$\delta_1(c_1-a_1b_1),\delta_2\{(ab+c)+(c_1-a_1b_1)\}$$
$$\delta_3(c_2-a_2b_2),\delta_4\{(ab+c)+(c_2-a_2b_2)\}$$
$$[\delta_{1,j}]_i^X,[\delta_{2,j}]_i^X,[\delta_{3,j}]_i^X,[\delta_{4,j}]_i^X,\ (j=0,1,\dots,k-1)$$

3.3.2 積和演算の手順③で 0 が復元された場合

この場合，$\delta_2\{(ab+c)+(c_1-a_1b_1)\}$も 0 が復元されれば，$ab+c=0$が漏洩する．よって，この場合手順③の直後に追加処理を加える．

① サーバS_iは，乱数$\tau_{1,i},w_{1,i}$を生成し，$\tau_{1,i}w_{1,i}$を計算してブロードキャストする．さらに，$w_{1,i}$のハッシュ値$h_{w_{1,i}}$をブロードキャストする．

② 全サーバは，$\tau_1 w_1=\prod_{j=0}^{k-1}\tau_{1,i}w_{1,i}$を計算する．

③ サーバS_iは，乱数の断片から以下ように乱数比の断片を計算してブロードキャストし，全サーバはその値を$i=0,..,k-1$に対して乗算して乱数比を計算する．

$$\frac{\tau_{1,i}}{\delta_{1,i}\varepsilon_{7,i}}\rightarrow\frac{\tau_1}{\delta_1\varepsilon_7}$$

④ $\tau_1([c_1+w_1]-a_1b_1)$を，$\delta_1(c_1-a_1b_1)$に代わる新たな公開値とする．分散値の計算は以下のように行う．

$$[\tau_1([c_1+w_1]-a_1b_1)]_i$$
$$=\delta_1(c_1-a_1b_1)\times\frac{\tau_1}{\delta_1\varepsilon_7}\times[\varepsilon_7]_i+\tau_1w_1$$

⑤ 以降, 3.3.1 の手順④以降を行う．ただし，$\delta_2\{(ab+c)+(c_1-a_1b_1)\}$は，$\tau_2\{(ab+c)+([c_1+w_1]-a_1b_1)\}$に置き換える．この値に対する分散値は以下のように計算できる．

$$[\delta_2\{(ab+c)+([c_1+w_1]-a_1b_1)\}]_i$$
$$=[\delta_2\{(ab+c)+(c_1-a_1b_1)\}]_i$$
$$+\tau_1w_1\times\left(\frac{\delta_2}{\tau_1\varepsilon_8}\right)\times[\varepsilon_8]_i$$

よって，以下の 2 値が生成される．

$$\tau_1([c_1+w_1]-a_1b_1)$$
$$\delta_2\{(ab+c)+([c_1+w_1]-a_1b_1)\}$$

$\delta_3(c_2-a_2b_2)$が 0 の場合も同様な処理を行う．また，演算の最後に$\delta_{1,i}$ではなく$\tau_{1,i}$を XOR 法で秘密分散する．

3.4. 復元

積和演算結果$ab+c$の復元を考える．

① 復元者は，k台のサーバS_iから$[\delta_{2,j}]_i^X,[\delta_{4,j}]_i^X$を収集して復元し，$\delta_2',\delta_4'$を計算する．また，$\delta_2\{(ab+c)+(c_1-a_1b_1)\},\delta_4\{(ab+c)+(c_2-a_2b_2)\}$も収集して$\{(c_1-a_1b_1)-(c_2-a_2b_2)\}$を計算する．

$$\{(c_1-a_1b_1)-(c_2-a_2b_2)\}'$$
$$=\frac{\delta_2\{(ab+c)+(c_1-a_1b_1)\}}{\delta_2'}-\frac{\delta_4\{(ab+c)+(c_2-a_2b_2)\}}{\delta_4'}$$

② ディーラ A,B,C は，復元者に乱数a_1,a_2,b_1,b_2,c_1,c_2を送信する．

③ サーバS_iは，復元者に$h_{a_1},h_{a_2},h_{b_1},h_{b_2},h_{c_1},h_{c_2}$を送信する．

④ 復元者は，手順②で受け取った乱数のハッシュ値を計算して，手順③で受けとった値と一致するか検証する．検証が問題なければ$\{(c_1-a_1b_1)-(c_2-a_2b_2)\}$を計算する．

⑤ 復元者は，手順①で計算した$\{(c_1-a_1b_1)-(c_2-a_2b_2)\}'$と手順④で計算した$\{(c_1-a_1b_1)-(c_2-a_2b_2)\}$が一致するか検証する．

$$\{(c_1-a_1b_1)-(c_2-a_2b_2)\}'-\{(c_1-a_1b_1)-(c_2-a_2b_2)\}$$
$$=0$$

[復元部分]

⑥ 復元者は，すべての検証が問題なければ，手順①で計算した$\frac{\delta_2\{(ab+c)+(c_1-a_1b_1)\}}{\delta_2'}$から手順④で計算した$(c_1-a_1b_1)$を引いて復元値とする．

$$\frac{\delta_2\{(ab+c)+(c_1-a_1b_1)\}}{\delta_2'}-(c_1-a_1b_1)$$
$$=ab+c$$

4. 安全性

本稿では, 以下ように安全性要件と, 安全性設定を定め, 安全性を証明する．

[安全性要件]

①機密性： honest なユーザが入力した秘密情報が攻撃者に漏洩しない．

②完全性：honest な復元者はディーラが分散した値，またはそれらから計算される値を復元処理で得る．

[安全性設定]

(1)攻撃者は変換用乱数組$[\varepsilon_h]_i,\varepsilon_{h,i}$で求まる$\varepsilon_h$を知らない．

(2)攻撃者の計算能力は有限でハッシュ値h_xからxを計算できない．

また，攻撃者は$k-1$台のサーバを乗っ取る者を考える．この攻撃者は，$k-1$台のサーバが保持する値を組み合わせてディーラの入力や復元結果を不正に知ろうとし，秘匿計算で偽物の分散値を送信するなどして，復元者に偽物の値を復元させることを目指す．

4.1. 機密性

攻撃者は，$k-1$台のサーバが保持する情報から，秘密情報a,b,cに関する以下の値を知る．

Vol.2019-CSEC-86 No.42
Vol.2019-SPT-34 No.42
2019/7/24

$$x_a = \{\alpha_1 a_1, \alpha_2(a+a_1), \alpha_3 a_2, \alpha_4(a+a_2)\}$$
$$x_b = \{\beta_1 b_1, \beta_2(b+b_1), \beta_3 b_2, \beta_4(b+b_2)\}$$
$$x_c = \{\gamma_1 c_1, \gamma_2(c+\gamma_1), \gamma_3 c_2, \gamma_4(c+c_2)\}$$
$$x_{ab+c} = \{\delta_1(c_1 - a_1 b_1), \delta_2\{(ab+c)+(c_1-a_1b_1)\}, \delta_3(c_2 - a_2 b_2), \delta_4\{(ab+c)+(c_2-a_2b_2)\}\}$$

$$[\alpha_{1,j}]_i^X, [\alpha_{2,j}]_i^X, [\alpha_{3,j}]_i^X, [\alpha_{4,j}]_i^X, [\beta_{1,j}]_i^X, [\beta_{2,j}]_i^X, [\beta_{3,j}]_i^X, [\beta_{4,j}]_i^X$$

$$[\gamma_{1,j}]_i^X, [\gamma_{2,j}]_i^X, [\gamma_{3,j}]_i^X, [\gamma_{4,j}]_i^X, [\delta_{1,j}]_i^X, [\delta_{2,j}]_i^X, [\delta_{3,j}]_i^X, [\delta_{4,j}]_i^X$$

$$h_{a_1}, h_{a_2}, h_{b_1}, h_{b_2}, h_{c_1}, h_{c_2}$$
$$(j = 0,1,\dots,k-1), (i = 0,1,\dots,k-2)$$

[1] 安全性設定(2)より，ハッシュ値からその入力は得られない．

$$H(x) = H(x|h_x)$$

[2] $k-1$個の分散値からは復元値は得られない．

$$H(x_{i,j}) = H\left(x_{i,j}|[x_{i,j}]_0, \dots, [x_{i,j}]_{k-2}\right)$$

[3] 全サーバが持つx_a, x_b, x_c, x_{ab+c}から，秘密情報は得られない．

$$\mathrm{H}(a) = H(a|x_a, x_b, x_c, x_{ab+c})$$
$$\mathrm{H}(b) = H(b|x_a, x_b, x_c, x_{ab+c})$$
$$\mathrm{H}(c) = H(c|x_a, x_b, x_c, x_{ab+c})$$

上記[1][2][3]より，サーバが保持すると定めた値から秘密情報が漏洩しないことがわかる．

次に考えるべき事は，サーバが保持しないと定めた値を攻撃者が乗っ取るサーバが不当に保持していた場合である．このような値には，演算中に得る乱数比などがある．この乱数比から個々の乱数が漏洩すれば，秘密情報も漏洩する．（例えば，α_1, α_2が漏洩すれば$\alpha_2(a+a_1)$からaが漏洩する．）しかし，生成される乱数比はすべて変換用乱数が乗じてあるため，安全性設定(1)および変換用乱数組は毎回異なるものを使用するという前提より，どの乱数比を何個組み合わせても個々の乱数が漏洩しないことが言える．よって，演算中に計算される乱数比から個々の乱数は漏洩しない．

3.3.2 の処理が行われると，$\delta_1(c_1 - a_1 b_1) = 0$から$c_1 - a_1 b_1 = 0$が漏洩する．しかし，個々の乱数$a_1, b_1, c_1$は安全性設定(2)よりそれぞれに対するディーラしか知らないため，攻撃者には漏洩しない．また，この乱数の部分は$c_1 + w_1 - a_1 b_1$に更新されるため，検証にも影響は無いと言える．

以上より，提案方式は機密性を持つことが言える．

4.2. 完全性

攻撃者は，$k-1$台のサーバを操作して任意の値を送信させるなどして計算結果を偽物の値に操作し，復元者に偽物の値を復元させることを目的とする．

復元に$\delta_1(c_1 - a_1 b_1), \delta_3(c_2 - a_2 b_2)$は用いないため，3.3.1 の④から考える．攻撃者に乗っ取られたサーバが偽物の値を送信することで，正しい値との比をq_1, q_2, q_3, q_4とした偽物の乱数比が計算された場合，手順④は以下のような計算となる．ここで，q_1, q_2, q_3, q_4は攻撃者が任意に調整することができる．

$$[\delta_2\{(ab+c)+(c_1-a_1b_1)\}]_i$$
$$= \alpha_2(a+a_1) \times \beta_2(b+b_1) \times \left(q_1 \frac{\delta_2}{\alpha_2\beta_2\varepsilon_3}\right) \times [\varepsilon_3]_i$$
$$-\alpha_2(a+a_1) \times \beta_1 b_1 \times \left(q_2 \frac{\delta_2}{\alpha_2\beta_1\varepsilon_4}\right) \times [\varepsilon_4]_i$$
$$-\alpha_1 a_1 \times \beta_2(b+b_1) \times \left(q_3 \frac{\delta_2}{\alpha_1\beta_2\varepsilon_5}\right) \times [\varepsilon_5]_i$$
$$+\gamma_2(c+c_1) \times \left(q_4 \frac{\delta_2}{\gamma_2\varepsilon_6}\right) \times [\varepsilon_6]_i$$

よって，復元値は以下となる．

$$\delta_2\{(ab+c)+(c_1-a_1b_1)\}'$$
$$= \delta_2\{q_1(a+a_1)(b+b_1) - q_2(a+a_1)b_1 - q_3 a_1(b+b_1) + q_4(c+c_1)\}$$

これを整理すると，

$$\delta_2\{(ab+c)+(c_1-a_1b_1)\}'$$
$$= \delta_2\{q_1 ab + (q_1-q_2)ab_1 + (q_1-q_3)a_1 b + (q_1-q_2-q_3)a_1 b_1 + q_4 c + q_4 c_1\}$$

同様に，差分をr_1, r_2, r_3, r_4とすると，

$$\delta_4\{(ab+c)+(c_2-a_2b_2)\}'$$
$$= \delta_4\{r_1 ab + (r_1-r_2)ab_2 + (r_1-r_3)a_2 b + (r_1-r_2-r_3)a_2 b_2 + r_4 c + r_4 c_2\}$$

さらに復元処理の手順①を考え，復元者が$\delta_2' = t_1\delta_2, \delta_4' = t_2\delta_4$を得たとすると，手順①で計算される値は以下となる．

$$\{(c_1 - a_1 b_1) - (c_2 - a_2 b_2)\}'$$
$$= \frac{\delta_2\{(ab+c)+(c_1-a_1b_1)\}'}{t_1\delta_2} - \frac{\delta_4\{(ab+c)+(c_2-a_2b_2)\}'}{t_2\delta_4}$$

手順④でのハッシュ値を用いた乱数の検証が問題ない場合，手順⑤の検証は以下のようになる．

$$\{(c_1 - a_1 b_1) - (c_2 - a_2 b_2)\}' - \{(c_1 - a_1 b_1) - (c_2 - a_2 b_2)\}$$
$$= \frac{\delta_2\{(ab+c)+(c_1-a_1b_1)\}'}{t_1\delta_2} - \frac{\delta_4\{(ab+c)+(c_2-a_2b_2)\}'}{t_2\delta_4} - \{(c_1 - a_1 b_1) - (c_2 - a_2 b_2)\}$$
$$= \left(\frac{q_1}{t_1} - \frac{r_1}{t_2}\right)ab + \left(\frac{q_4}{t_1} - \frac{r_4}{t_2}\right)c + \frac{(q_1-q_2)}{t_1}ab_1 + \frac{(q_1-q_3)}{t_1}a_1 b + \left\{\frac{(q_1-q_2-q_3)}{t_1} + 1\right\}a_1 b_1 - \frac{(r_1-r_2)}{t_2}ab_2 - \frac{(r_1-r_3)}{t_2}a_2 b - \left\{\frac{(r_1-r_2-r_3)}{t_2} + 1\right\}a_2 b_2 + \left(\frac{q_4}{t_1} - 1\right)c_1 - \left(\frac{r_4}{t_2} - 1\right) = 0$$

ここで，$a, b, c, a_1, a_2, b_1, b_2, c_1, c_2$はそれぞれに対するディーラしか知らないのでこの式が成り立つにはそれぞれの項が 0 である必要があり，その条件をまとめると，以下のようになる．

$$\frac{q_1}{t_1} = \frac{q_2}{t_1} = \frac{q_3}{t_1} = \frac{r_1}{t_2} = \frac{r_2}{t_2} = \frac{r_3}{t_2} = \frac{q_4}{t_1} = \frac{r_4}{t_2} = 1$$

この条件での復元値は

$$\frac{\delta_2\{(ab+c)+(c_1-a_1b_1)\}'}{t_1\delta_2}-(c_1-a_1b_1)$$
$$=\left\{\frac{q_1}{t_1}ab+\left(\frac{q_1}{t_1}-\frac{q_2}{t_1}\right)ab_1+\left(\frac{q_1}{t_1}-\frac{q_2}{t_1}\right)a_1b+\left\{\frac{q_1}{t_1}-\frac{q_2}{t_1}-\frac{q_3}{t_1}-1\right\}a_1b_1+\frac{q_4}{t_1}c+\left(\frac{q_4}{t_1}-1\right)c_1\right\}$$
$$=ab+c$$

よって，検証が成功する場合，復元値は正しい値となる．

次に，$\delta_1(c_1-a_1b_1)$に対して 3.3.2 の処理が行われた場合を考える．この時，計算される値は以下のようになる．(差分s_1)

$$\delta_2\{(ab+c)+([c_1+w_1]-a_1b_1)\}'$$
$$=\delta_2\{(ab+c)+(c_1-a_1b_1)\}'+s_1w_1$$

同様に$\delta_3(c_2-a_2b_2)$に対しても 3.3.2 の処理が行われた場合を考えると，(差分s_2)

$$\delta_4\{(ab+c)+([c_2+w_2]-a_2b_2)\}'$$
$$=\delta_4\{(ab+c)+(c_2-a_2b_2)\}'+s_2w_2$$

よって，それぞれにs_1w_1, s_2w_2が追加されるだけである．ここで，攻撃者が操るサーバが演算に使用した値とは異なる$w'_{1,i}, w'_{2,i}$に対するハッシュ値を計算してブロードキャストした場合，手順⑤の検証が通過する条件は，すでに議論したものに加え，以下の２条件が追加される．

$$s_1w_1-\prod_{i=0}^{k-1}w'_{1,i}=s_1w_1-u_1w_1=(s_1-u_1)w_1=0$$
$$s_2w_2-\prod_{i=0}^{k-1}w'_{2,i}=(s_2-u_2)w_2=0$$

しかし，w_1, w_2は 3.3.2 の手順②で定まる値とハッシュ値の入力から定まる値の２種類しかないため，どちらかを基準にし，基準にしたものを正しい値として考えて良い．よって，$s_1=s_2=1$または $u_1=u_2=1$となり，検証を成功させるためには$s_1=s_2=u_1=u_2=1$である必要がある．この場合に正しい値が復元されることは同様に言える．

以上より，3.3.2 の有無にかかわらず，秘匿積和演算および復元処理に不正が行われても，すべての検証が成功すれば復元値は正しい値となることがわかる．

5. 任意回の積和演算に対する安全性の考察

4 章までの議論で，積和演算 1 回に対する安全性を証明した．しかし，任意の関数を計算するには任意回の積和演算を連続させる必要があり，これに対する安全性を証明する必要がある．機密性は同様に証明できるが，問題は完全性である．まず，我々は 2 回の積和演算の連続に対して計算と復元を行い完全性の確認を行い，差分が満たすべき条件の変化を追った．設定した 2 回の積和演算を図 1 に示す．図 1 中のa,b,c,d,e,f,g,h,iはディーラが入力した値であり，正しい値とした．積和演算 1,2,3 は 4.2 で考えた攻撃方法を採用し，これらの出力$s=ab+c, t=de+f, u=gh+i$に対する分散値は 4.2 における$\delta_2\{(ab+c)+(c_1-a_1b_1)\}', \delta_4\{(ab+c)+(c_2-a_2b_2)\}'$と同じ形を取る（例えば，$t$に対する分散値は$\delta_6\{(de+f)+(f_1-d_1e_1)\}', \delta_8\{(de+f)+(f_2-d_2e_2)\}'$）．

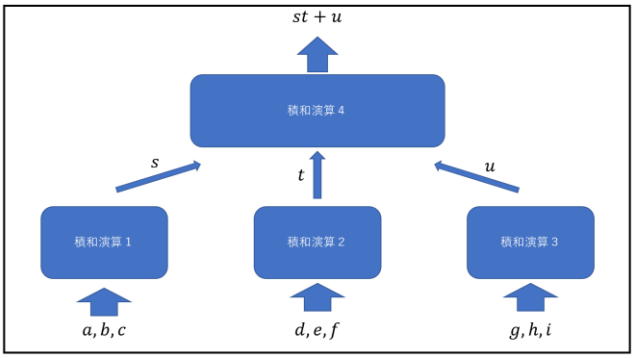

図 1　２回の積和演算の連続

そして，これらs,t,uを入力とする積和演算 4 を考えて完全性の検証を行った．その結果，積和演算 4 に対してのみ検証を行うだけで，検証が成功すれば復元値は正しい値となることが確認できた．また，以下の結果を得た．

［1］ 積和演算 4 での不正の有無にかかわらず，積和演算 4 での復元および検証が成功するためには，積和演算 1, 2, 3 で計算されるs,t,uに対する分散値はすべて以下の形で表せなければならない．

$$\phi_1(Q_1z_1-Q_2x_1y_1)$$
$$\phi_2\{(Q_3xy+Q_4z)+(Q_4z_1-Q_3x_1y_1)\}$$
$$\phi_3(Q_5z_2-Q_6x_2y_2)$$
$$\phi_4\{(Q_7xy+Q_8z)+(Q_8z_2-Q_7x_2y_2)\}$$

ただし，$(x,y,z)=\{(a,b,c),(d,e,f),(g,h,i)\}$であり，全体に掛けられている$\phi_i$や差分$Q_j$は積和演算 1, 2, 3 でそれぞれ独立した値である。また，積和演算 4 での不正がない場合は，以下のように書き換えることができる事を確認した．

$$Q_1\phi_1(z_1-x_1y_1)$$
$$Q_3\phi_2\{(xy+z)+(z_1-x_1y_1)\}$$
$$Q_5\phi_3(z_2-x_2y_2)$$
$$Q_7\phi_4\{(xy+z)+(z_2-x_2y_2)\}$$

ここで，4.2 における$\delta_2\{(ab+c)+(c_1-a_1b_1)\}', \delta_4\{(ab+c)+(c_2-a_2b_2)\}'$に対して，同様に検証が成功する条件を代入してみる．検証成功に必要な条件をt_1, t_2を用いない場合(復元処理を行う前)に書き換えると，以下のようになる．

$$q_1=q_2=q_3=q_4,\qquad r_1=r_2=r_3=r_4$$

よって，この条件を用いて書き換えると，

$$\delta_2\{(q_1ab+q_4c)+(q_4c_1-q_1a_1b_1)\}$$
$$\delta_4\{(r_1ab+r_4c)+(r_4c_2-r_1a_2b_2)\}$$

さらに，

$$q_1\delta_2\{(ab+c)+(c_1-a_1b_1)\}$$
$$r_1\delta_4\{(ab+c)+(c_2-a_2b_2)\}$$

となる．これは，２回の積和演算の連続に対する，積和演算４での不正がない場合と同じ形となる．

ここで重要な事は，不正が２回連続で起こるか否かにかかわらず，秘密情報の部分（例えばab）と，それに対応する乱数の部分（例えばa_1b_1）の差分が同じ値であることである．同じ差分が発生し得る理由は，積和演算の計算過程から確認できる．これにより，ディーラが用意する正しい乱数による検証によって乱数の部分の差分が 1 である事が条件として加わり，さらに差分が同じことから秘密情報の部分も差分は 1 となり，復元値は正しい値となる．

今までの議論をn入力$(x_1,\dots,x_n)$から1出力$f(x_1,\dots,x_n)$を求める計算に対して考える．この関数fはある一定の回数の積和演算を行う事で計算できる．まず，最終的な計算結果は以下のように表せる．

$$\varphi_2\{f'(\boldsymbol{x})+g'(\boldsymbol{x},\boldsymbol{x_1})+h'(\boldsymbol{x_1})\}$$
$$\varphi_4\{f''(\boldsymbol{x})+g''(\boldsymbol{x},\boldsymbol{x_2})+h''(\boldsymbol{x_2})\}$$

※ $\boldsymbol{x}=\{x_1,\dots,x_n\},\ \boldsymbol{x_1}=\{x_{1,1},\dots,x_{n,1}\},\ \boldsymbol{x_2}=\{x_{1,2},\dots,x_{n,2}\}$

※ 正しい値は，関数$f(\boldsymbol{x}),h(\boldsymbol{x_1}),h(\boldsymbol{x_2})$から計算される以下の値とする．

$$\varphi_2\{f(\boldsymbol{x})+h(\boldsymbol{x_1})\}$$
$$\varphi_4\{f(\boldsymbol{x})+h(\boldsymbol{x_2})\}$$

この場合に対して，3.2 の手順⑤は，

$$\frac{\{f'(\boldsymbol{x})+g'(\boldsymbol{x},\boldsymbol{x_1})+h'(\boldsymbol{x_1})\}}{t_1}-\frac{\{f''(\boldsymbol{x})+g''(\boldsymbol{x},\boldsymbol{x_2})+h''(\boldsymbol{x_2})\}}{t_2}-\big(h(\boldsymbol{x_1})-h(\boldsymbol{x_2})\big)$$
$$=\left(\frac{f'(\boldsymbol{x})}{t_1}-\frac{f''(\boldsymbol{x})}{t_2}\right)+\frac{g'(\boldsymbol{x},\boldsymbol{x_1})}{t_1}-\frac{g''(\boldsymbol{x},\boldsymbol{x_2})}{t_2}+\left(\frac{h'(\boldsymbol{x_1})}{t_1}-h(\boldsymbol{x_1})\right)-\left(\frac{h''(\boldsymbol{x_2})}{t_1}-h(\boldsymbol{x_2})\right)=0$$

よって，検証成功条件は，

$$\frac{f'(\boldsymbol{x})}{t_1}-\frac{f''(\boldsymbol{x})}{t_2}=0$$
$$\frac{g'(\boldsymbol{x},\boldsymbol{x_1})}{t_1}=0,\qquad\frac{g''(\boldsymbol{x},\boldsymbol{x_2})}{t_2}=0$$
$$\frac{h'(\boldsymbol{x_1})}{t_1}-h(\boldsymbol{x_1})=0,\qquad\frac{h''(\boldsymbol{x_2})}{t_1}-h(\boldsymbol{x_2})=0$$

となる．この時，復元値は，

$$\frac{\{f'(\boldsymbol{x})+g'(\boldsymbol{x},\boldsymbol{x_1})+h'(\boldsymbol{x_1})\}}{t_1}-h(\boldsymbol{x_1})=\frac{f'(\boldsymbol{x})}{t_1}+\frac{g'(\boldsymbol{x},\boldsymbol{x_1})}{t_1}+\left(\frac{h'(\boldsymbol{x_1})}{t_1}-h(\boldsymbol{x_1})\right)=\frac{f'(\boldsymbol{x})}{t_1}$$

ここで，今までの議論より秘密情報の部分と，それに対応する乱数の部分の差分が同じ値であるから，

$$\frac{f'(\boldsymbol{x})}{t_1}=\frac{f'(q_1x_1,q_2x_2,\dots,q_nx_n)}{t_1}$$
$$\frac{h'(\boldsymbol{x_1})}{t_1}=\frac{h'(q_1x_{1,1},q_2x_{2,1},\dots,q_nx_{n,1})}{t_1}$$

と表せる．また，

$$\frac{h'(\boldsymbol{x_1})}{t_1}-h(\boldsymbol{x_1})=\frac{h'(q_1x_{1,1},q_2x_{2,1},\dots,q_nx_{n,1})}{t_1}-h(x_{1,1},x_{2,1},\dots,x_{n,1})$$

これより，それぞれの項に対する差分は 1 となる。（例えば，h'に項$q_1x_{1,1}$が存在する場合は$\frac{q_1}{t_1}=1$であり$\frac{f'(x)}{t_1}$に$\frac{q_1}{t_1}x_1$の項が含まれる．同様に，項$\frac{q_2q_3}{t_1}x_{2,1}x_{3,1}$が存在する場合は$\frac{q_2q_3}{t_1}=1$であり$\frac{f'(x)}{t_1}$に$\frac{q_2q_3}{t_1}x_2x_3$の項が含まれる．）以上の議論より，以下が言える．

$$\text{検証成功}\ \Leftrightarrow\ \frac{f'(\boldsymbol{x})}{t_1}=f(\boldsymbol{x})$$

ここで，任意回の積和演算を行う場合は，あるサーバは毎回に同じ乱数を扱うという条件が追加される．（例えば，$\delta_{1,j}$を復元したサーバS_jは，その後の演算で$\delta_{1,i}\ (i\neq j)$ を復元しない．）

6. 結論

提案方式では，以下の条件において安全な秘匿計算を実現した．

[安全性要件]

①機密性： honest なユーザが入力した秘密情報が攻撃者に漏洩しない．

②完全性：honest な復元者はディーラが分散した値，またはそれらから計算される値を復元処理で得る．

[安全性設定]

(1)攻撃者は変換用乱数組$[\varepsilon_h]_i,\varepsilon_{h,i}$で求まる$\varepsilon_h$を知らない．

(2)攻撃者の計算能力は有限でハッシュ値h_xからxを計算できない．

(3) 演算を連続させる場合，サーバは毎回同じ乱数を扱う．この場合，積和演算の連続において各サーバは同じ乱数の断片を扱うという前提が加わる．（$\alpha_{1,i}$を復元したサーバは$\alpha_{1,j}$を復元しない．$(i\neq j)$）

今後の方針としては，まず任意回積和演算に対する完全な証明示す事を目標とする．本方式は，演算の最終結果に対してのみ検証を行えば良いため非常に軽量であると考えている．

参考文献

[1] A. Shamir “How to Share a Secret” Communications of the ACM November 1979 Volume 22 Number 11 pp. 612-613

[2] 神宮武志，青井健，ムハンマド　カマル　アフマド　アクマル　アミヌディン，岩村惠市 “秘密分散法を用いた次数変化のない秘匿計算手法” 情報処理学会論文誌 Vol.59 No.3 pp.1038-

Vol.2019-CSEC-86 No.42
Vol.2019-SPT-34 No.42
2019/7/24

1049 (Mar. 2018)

[3] AHMAD AKMAL AMINUDDIN BIN MOHD KAMAL, 岩村惠市 “秘密分散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” 情報処理学会論文誌 59 巻 9 号 pp. 1581-1595 2018/09/15

[4] 鴇田恭平, 岩村惠市 “高速かつ n<2k-1 において秘密情報に 0 を含んでも実行可能な秘密分散による秘匿計算” 電気学会論文誌 C Vol.138 No.12 pp.1634-1645

[5] J. Kurihara, S. Kiyomoto, K. Fukushima, and T.Tanaka “On a fast (k;n)-threshold secret sharing scheme” IEICE Transactions on Fundamentals of Electronics, Communications and Computer Science, Vol.E91-A, No.9, pp.2365-2378 (2008)

[6] J. Kurihara, S. Kiyomoto, K. Fukushima, and T.Tanaka “A new (k;n)-threshold secret sharing scheme and its extension” ISC 2008 Conference (2008)

[7] S. Yakoubov, V. Gadepally, N. Schear, E. Shen, A. Yerukhimovich “A Survey of Cryptographic Approaches to Securing Big-Data Analytics in the Cloud” 2014 IEEE High Performance Extreme Computing Conference

[8] Michael Backes, Aniket Kate, Arpita Patra “Computational Verifiable Secret Sharing Revisited” ASIACRYPT 2011, LNCS 7073, pp. 590-609

[9] I. Damgard,V. Pastro, N. Smart, S. Zakarias “Multiparty Computation from Somewhat Homomorphic Encryption” Advaces in Cryptology – CRYPTO 2012 pp 643-662

[10] I. Damgard, M. Keller, E. Larraria,V. Pastro, P. Scholl, Nigel P. Smart ‘Practical Covertly Secure MPC for Dishonest Majority – Or: Breaking the SPDZ Limits’ Computer Security – ESORICS 2013, pp 1-18, Springer LNCS 8134

[11] M. Keller, E. Orsiniy, P. Schollz ” MASCOT: Faster Malicious Arithmetic Secure Computation with Oblivious Transfer” CCS '16 Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Pages 830-842

[12] R. Cramer, I. Damgård, D. Escudero, P. Scholl, C. Xing, “SPD$\mathbb{Z}_{2^k}$: Efficient MPC mod 2^k for Dishonest Majority” Advances in Cryptology – CRYPTO 2018.