n<2k-1 において入力者の active な攻撃に対しても安全な秘密分散を用いた秘匿計算

落合将吾 [1,*]　岩村惠市 [1]

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，1 回乗算を行うと閾値が k から 2k-1 へ増加してしまい，乗算結果の復元を考えると n≥2k-1 が要求される．これに対し，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．この手法は入力者が active な攻撃者になった場合に対しても安全であり，セキュリティの 3 要素である機密性，完全性，可用性をすべて実現する．

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

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

Shogo Ochiai[1,*]　Keiichi Iwamura[1]

Abstract: Secret sharing scheme proposed by Shamir can be used for secure multiparty computation. However, secure multiplication using Shamir's scheme, the threshold increases from k to 2k-1. In other words, for reconstruction of multiplication result, n>=2k-1is required. To solve this problem, Tokita et al. have proposed a secure multiparty computation scheme that has the advantage of being applicable to n<2k-1. However, their scheme is not secure against active adversary. So, we proposed the secure computation scheme that can verify the result of the computation and apply to n<2k-1. Our scheme has security for active adversary and achieves confidentiality, integrity and availability.

Keywords: Secure Computation, Secret Sharing Scheme, MPC, active adversary

1. はじめに

近年，IoT (Internet of Things) の発展などにより，世の中のあらゆる事象をデータとして取得できるようになり，取得した多種多様かつ膨大なデータから新たな価値を創造し，ビジネスや生活にフィードバックするという動きが加速している[1]．一方で，IoT などで収集したデータの中には個人情報などが含まれるため，データの扱いに注意するべきである．つまり，データを保護しつつ利活用できるような手法が必要であり秘匿計算という技術がこれを可能にする．

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．一般に，秘密分散法を用いた手法は計算量が少ないため，著者らは秘密分散法を用いた秘匿計算手法の研究を行っている．秘密分散法は，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[2](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 回行うと閾値がkから$2k-1$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 方式が提案されている (TUS1[3], TUS2[4], TUS3[5]). TUS 方式は，乱数で秘密情報を秘匿化して公開情報とし，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．

ここで，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. 実際にデータの利活用を行う場合を考えると，入力に対して改ざんなどの攻撃が行われた場合その出力は正当な結果とならず，データを利活用する本来の目的を達成できなくなってしまう．よって，データの改ざんなどが行われる可能性を考慮すると，active な攻撃者に対して安

1 東京理科大学
Tokyo University of Science
* Ochiai_shogo@sec.ee.kagu.tus.ac.jp

全な手法が必要となる．秘密分散法においては，配布する分散値を増やすなどしてデータの正当性を検証することができ,そのような手法は検証可能秘密分散 (VSS) と呼ばれているが必ずしも秘匿計算が可能なわけではない．また，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マルチパーティ計算 (MPC) は，一般に秘密分散を用いて実現され，出力の正当性が検証可能な方式も多い．特に，現在著名な MPC 手法の多くは分散値の総和を秘密情報とする加法的秘密分散法を用いている．加法的秘密分散法は,処理は高速であるが$n = k$という制限がある．また，加法的秘密法を拡張した Replicated 秘密分散法があり，これは加法的秘密法の高速さを実現しつつ，$n = k$という制限はない．しかし，その分散方法から$k = 2$という制限が発生する．

以上の問題に対し,本論文では TUS 方式のアプローチを採用することで秘匿乗算における次数変化が起こらず，$n < 2k - 1$で適用できる手法を提案する．さらに,計算結果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 な攻撃者に対するマルチパーティ計算の説明を行う．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 つの性質(定義)を持つ．

(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 法や栗原らの XOR による方式[6][7](以降 XOR 法)，加法的秘密分散法などが提案されている．秘密分散法はセキュリティの 3 要素の 1 つである機密性を実現できる．ここで，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[8]．また，$n > k$に適用できる場合は可用性を実現し，分散値や復元値の正当性を検証可能にすることで完全性も実現できる[9][10]．

2.2 TUS3 方式[5]

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[3]．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[4]．この方式は，秘密情報に 0 は含まないなど 3 つの条件において，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,栗原らの提案した高速な XOR 法に着目し，一部を XOR 法に置き換えることにより高速化を実現した[5]．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，[5]では 2 つの方式が提案されてい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．

[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，任意の四則演算に関して，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 な攻撃者に対するマルチパーティ計算

Active (または malicious) な攻撃者に対して安全なマルチパーティ計算の代表例として Damgard らによる SPDZ

(SPDZ-1[11], SPDZ-2[12]) やその改良方式 (MASCOT[13], [14]) ，また，荒木らによる[15]などが挙げられる．SPDZ およびその改良方式は$n = k$にのみ適用可能な加法的秘密分散を用いており，Beaver により提案された乗算用組[16]$(a, b, c = ab)$に対する分散値を事前処理で生成し，秘匿乗算に利用する．また，検証方法として，情報理論的 MAC (information-theoretic MAC) やコミットメントを活用する．[15]では加法的秘密分散における Replicated 秘密分散を用いており，n個の分散値に対してユーザはある特定の 1 つ以外をすべて保持する．つまり，任意の 2 人から秘密情報の復元が可能であり，[15]では$n = 3, k = 2$で限定された手法が提案されている．この手法も加法的秘密分散に基づくため，秘匿乗算には[11]～[14]と同様に乗算用組$(a, b, c = ab)$を用いる．[11]～[14]における乗算用組の生成は 2012 に提案された準同型暗号を用いる生成方法[11]から改良され続け，2018 に提案された[14]が最速である．乗算用組は秘密情報に依存しないため事前処理として前もって生成しておくことができるが，生成過程で準同型暗号や紛失通信を用いておりそのコストは依然として無視できないといえる．

3. 提案方式

3.1 概要

提案方式では，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，秘匿乗算を行うために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$n \geq k$に適用できるので従来方式に対して可用性を高める．

3.2 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
h：ハッシュ関数
h_x：xに対するハッシュ値
S：分散値を保持するn台のサーバ集合

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

P：秘密情報の入力者または復元者

3.3 前提

(1) 生成される乱数はすべて非 0 とする．

(2) 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．

(3) S_iは事前に変換用乱数組$[\varepsilon_h]_i, [\varepsilon_{h,j}]_i^X$を保持しており，毎回異なるものを使用する．ただし，
$\varepsilon_h = \prod_{i=0}^{k-1} \varepsilon_{h,i}$，$h = 1, \dots$,必要個数，$j = 0,1, \dots, k-1$
であり，特に，$n = k$の場合は$[\varepsilon_h]_i, \varepsilon_{h,i}$を保持する．

(4) 演算の連続において，各サーバが扱う XOR で分散された乱数は固定される．

3.4 分散処理

以下に，$n \geq k$に適用できる分散処理を示す．特に，$n = k$に適用する場合は，XOR 法による乱数の分散は行わず，そのまま保持することで高速化できる．

① ユーザ$A(\in P)$は，乱数$A_{1,i}$を生成して，$A_1 = \prod_{j=0}^{k-1} A_{1,j}$を計算する．さらに$A_{1,i}$を分散に参加するサーバ$S_i$に送信する．

② ユーザAは，自身の秘密情報aを用いて$(a + A_1)$を計算し，全サーバにブロードキャストする．

③ サーバS_iは，乱数$\alpha_{1,i}, a_{1,i}, \alpha_{2,i}$を生成し，ハッシュ値$h_{a_{1,i}}$をブロードキャストする．さらに，$\alpha_{1,i}, a_{1,i}, \alpha_{2,i}$を XOR 法で秘密分散する．

④ サーバS_iは，$\alpha_{1,i}a_{1,i} = \alpha_{1,i} \times a_{1,i}$および$\alpha_{2,i}a_{1,i} = \alpha_{2,i} \times a_{1,i}$を計算してブロードキャストし，全サーバは以下を計算する．

$$\alpha_1 a_1 = \prod_{j=0}^{k-1} (\alpha_{1,j} a_{1,j}), \qquad \alpha_2 a_1 = \prod_{j=0}^{k-1} (\alpha_{2,j} a_{1,j})$$

⑤ サーバS_iは，以下ように乱数比の断片（左側）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(右側)を計算する．

$$\frac{\alpha_{2,i}}{\varepsilon_{1,i}}, \frac{\alpha_{2,i} A_{1,i}}{\varepsilon_{2,i}} \rightarrow \frac{\alpha_2}{\varepsilon_1}, \frac{\alpha_2 A_1}{\varepsilon_2}$$

⑥ サーバS_iは，以下のように分散値を計算し，ブロードキャストする．

$$[\alpha_2(a + a_1)]_i = (a + A_1) \times \frac{\alpha_2}{\varepsilon_1} \times [\varepsilon_1]_i + \alpha_2 a_1 - \frac{\alpha_2 A_1}{\varepsilon_2} \times [\varepsilon_2]_i$$

⑦ 全サーバは，k個数の分散値から$\alpha_2(a + a_1)$を復元する．

⑧ 全サーバは，手順①から手順⑦を別々の乱数を用いて再度行い，$\alpha_3 a_2, \alpha_4(a + a_2)$を計算しブロードキャストする．

⑨ サーバS_iは，以下を保持する．

$$\alpha_1 a_1, \alpha_2(a + a_1), \alpha_3 a_2, \alpha_4(a + a_2), (a + A_1), (a + A_2)$$

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

$$h_{a_{1,j}}, h_{a_{2,j}}, \qquad (j = 0,1, \dots, k-1)$$

ただし，$n = k$の場合は以下を保持する．

$$\alpha_1 a_1, \alpha_2(a + a_1), \alpha_3 a_2, \alpha_4(a + a_2), (a + A_1), (a + A_2)$$

$$\alpha_{1,i}, \alpha_{2,i}, \alpha_{3,i}, \alpha_{4,i}, a_{1,i}, a_{2,i}$$

$$h_{a_{1,j}}, h_{a_{2,j}}, \qquad (j = 0,1, \dots, k-1)$$

3.5 復元処理

[検証準備]

① 復元者$R \in P$は，k台のサーバS_iから$\alpha_2(a + a_1)', \alpha_4(a + a_2)', \left([\alpha_{2,j}]_i^X\right)', \left([\alpha_{4,j}]_i^X\right)'$を収集し，$\alpha'_{2,j}, \alpha'_{4,j}$を復元して α'_2, α'_4 を計算し，$(a_1 - a_2)'$を計算する．$(j = 0,1, \dots, k-1)$

$$(a_1 - a_2)' = \frac{\alpha_2(a + a_1)'}{\alpha_2'} - \frac{\alpha_4(a + a_2)'}{\alpha_4'}$$

② サーバS_iは，復元者Rに $[a_{1,j}]_i^X, [a_{2,j}]_i^X, h_{a_{1,j}}, h_{a_{2,j}}$ を送る．$(j = 0,1, \dots, k-1)$

③ 復元者Rは，$a_{1,i}, a_{2,i}$を復元してハッシュ値$h_{a_{1,i}}, h_{a_{2,i}}$を計算し，受けとったハッシュ値と一致するか検証する．

$$h_{a_{1,i}} = H(a_{1,i}), \quad h_{a_{2,i}} = H(a_{2,i}), \ (i = 0,1, \dots, k-1)$$

④ 復元者は，手順③の検証が問題なければ，以下を計算する．

$$a_1 = \prod_{i=0}^{k-1} a_{1,i}, \ a_2 = \prod_{i=0}^{k-1} a_{2,i}$$

[入力検証]

⑤ サーバS_iは，以下ように乱数比の断片（左側）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（右側）を計算する．

$$\frac{1}{\alpha_{2,i}\varepsilon_{1,i}}, \frac{A_{1,i}}{\varepsilon_{2,i}}, \frac{1}{\alpha_{4,i}\varepsilon_{3,i}}, \frac{A_{1,i}}{\varepsilon_{4,i}} \ \rightarrow \ \frac{1}{\alpha_2\varepsilon_1}, \frac{A_1}{\varepsilon_2}, \frac{1}{\alpha_4\varepsilon_3}, \frac{A_1}{\varepsilon_4}$$

⑥ サーバS_iは，以下を計算し、復元者へ送信する．

$$[a_1']_i = \alpha_2(a + a_1)' \times \frac{1}{\alpha_2\varepsilon_1}[\varepsilon_1]_i + \frac{A_2}{\varepsilon_2}[\varepsilon_2]_i - (a + A_2)$$

$$[a_2']_i = \alpha_4(a + a_2)' \times \frac{1}{\alpha_4\varepsilon_3}[\varepsilon_3]_i + \frac{A_1}{\varepsilon_4}[\varepsilon_4]_i - (a + A_1)$$

⑦ 復元者は，上記分散値からa_1', a_2'を復元し，手順④で求めた値と一致するか検証する．

[出力検証]

⑧ 復元者は，手順①で計算した$(a_1 - a_2)'$と手順④で求めた値から計算される$(a_1 - a_2)$が一致するか検証する．

$$(a_1 - a_2)' - (a_1 - a_2) = 0 \, ?$$

[復元部分]

⑨ 復元者は，すべての検証が問題なければ，手順①で計算された $\alpha_2(a + a_1)'/\alpha_2'$ から手順④で求めた a_1 を引いて，a を復元する．

$$a = \frac{\alpha_2(a + a_1)'}{\alpha_2'} - a_1$$

3.6 秘匿計算(0 が復元されない場合)

本稿では 3 入力a, b, cから 1 出力$d = ab + c$を求める積和演算を考える．3 人のユーザ$(\mathrm{A}, \mathrm{B}, \mathrm{C}) \in P$による 3 入力$a, b, c$に対する分散値を以下のように定義する．ただし，ハッシュ値などの演算に用いない値は省略してある．また，簡単のため$n = k$の場合を示す．

$$\alpha_1 a_1, \alpha_2(a + a_1), \alpha_3 a_2, \alpha_4(a + a_2), \alpha_{1,i}, \alpha_{2,i}, \alpha_{3,i}, \alpha_{4,i}$$
$$\beta_1 b_1, \beta_2(b + b_1), \beta_3 b_2, \beta_4(b + b_2), \beta_{1,i}, \beta_{2,i}, \beta_{3,i}, \beta_{4,i}$$
$$\gamma_1 c_1, \gamma_2(c + c_1), \gamma_3 c_2, \gamma_4(c + c_2), \gamma_{1,i}, \gamma_{2,i}, \gamma_{3,i}, \gamma_{4,i}$$

また，出力に対する分散値を以下のように定義する．

$$\delta_1(c_1 - a_1 b_1), \delta_2\{(ab + c) + (c_1 - a_1 b_1)\}$$
$$\delta_3(c_2 - a_2 b_2), \delta_4\{(ab + c) + (c_2 - a_2 b_2)\}$$
$$\delta_{1,i}, \delta_{2,i}, \delta_{3,i}, \delta_{4,i}$$

ここで，簡単のために以下のような分散値を定義することも可能であるが，計算量を削減するため上記の定義を用いる．安全性に影響は無い．

$$\delta_1(a_1 b_1 + c_1), \delta_2\{(ab + c) + (a_1 b_1 + c_1)\}$$
$$\delta_3(a_2 b_2 + c_2), \delta_4\{(ab + c) + (a_2 b_2 + c_2)\}$$
$$\delta_{1,i}, \delta_{2,i}, \delta_{3,i}, \delta_{4,i}$$

① 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．

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

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，$\delta_1(c_1 - a_1 b_1)$を復元する．

④ 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．

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

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$\delta_2\{(ab + c) + (c_1 - a_1 b_1)\}$を復元する．

⑦ 同様にして，$\delta_3(c_2 - a_2 b_2), \delta_4\{(ab + c) + (c_2 - a_2 b_2)\}$を計算する．

⑧ サーバS_jは，積和演算結果$ab + c$に対して以下を保持する．

$$\delta_1(c_1 - a_1 b_1), \delta_2\{(ab + c) + (c_1 - a_1 b_1)\}$$
$$\delta_3(c_2 - a_2 b_2), \delta_4\{(ab + c) + (c_2 - a_2 b_2)\}$$
$$\delta_{1,i}, \delta_{2,i}, \delta_{3,i}, \delta_{4,i}$$

3.7 秘匿計算(0 が復元される場合)

ここでは，3.6 に示した秘匿積和演算において，手順③で復元した$\delta_1(c_1 - a_1 b_1)$が 0 であった場合，$\delta_1 \neq 0$より，$c_1 - a_1 b_1 = 0$となる．この場合，手順⑥で再び 0 が復元されてしまうと，$ab + c = 0$が漏洩してしまう．手順④を行う前に

必要な処理を示す.

① $\delta_1(c_1 - a_1 b_1) = 0$を復元した場合, サーバ$S_i$は, 乱数$w_{1,i}$を生成し, $\delta_{1,i} w_{1,i}$を計算してブロードキャストする. さらに, w_iのハッシュ値h_{w_i}をブロードキャストする.

$$h_{w_{1,i}} = H(w_{1,i})$$

② 全サーバは, $\delta_1 w_1 = \prod_{j=0}^{k-1} \delta_{1,i} w_{1,i}$を計算する.

③ $\delta_1 w_1$を, $\delta_1(c_1 - a_1 b_1)$に代わる新たな分散値とする.

以上の処理を行った後に, 3.6 の手順④へ進む. ここで, 本処理は乱数$(c_1 - a_1 b_1) = 0$を新たな乱数$w_1 \neq 0$に置き換える処理である. よって, 3.6 の手順⑥では, $\delta_2\{(ab + c) + (c_1 - a_1 b_1)\}$の代わりに$\delta_2\{(ab + c) + (w_1)\}$を復元する必要がある. よって, 3.6 の手順⑥は以下のような計算となる.

$$[\delta_2\{(ab + c) + (w_1)\}]_i$$
$$= [\delta_2\{(ab + c) + (c_1 - a_1 b_1)\}]_i + [\delta_2 w_1]_i$$
$$= [\delta_2\{(ab + c) + (c_1 - a_1 b_1)\}]_i + \delta_1 w_1 \times \left(\frac{\delta_2}{\delta_1 \varepsilon_7}\right) \times [\varepsilon_7]_i$$

以上の追加処理を行う事で, 以下の分散値が生成できる.

$$\delta_1(w_1), \delta_2\{(ab + c) + (w_1)\}$$
$$\delta_{1,i}, \delta_{2,i}$$

4. 安全性

本章では, 提案方式の安全性を示す. 本章中のHはエントロピーとする. 提案方式で実現する安全性要件は, 以下の 2 点である. 可用性に関しては, 提案方式が$n \geq k$に適用できることを根拠にし, ここでの議論は省略する.

[安全性要件]

(1) 機密性 : honest な入力者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.

また, 提案方式では以下の設定において, active な攻撃を行う攻撃者を考える.

[攻撃者設定]

(1) 変換用乱数組$[\varepsilon_h]_i, [\varepsilon_{h,j}]_i^X$で求まる$\varepsilon_h$を知らない.

(2) 計算能力は有限であり, ハッシュ値$h_x = h(x)$からその入力xを計算できない.

(3) t入力 1 出力の演算に関して, 攻撃者$(\in P)$の結託は最大$t - 1$人である.

(4) 最大$k - 1$台のサーバが持つ情報を知れる.

4.1 分散処理に対する機密性

3.4 の分散処理において, honest なユーザAが入力した秘密情報aが攻撃者に漏洩しないことを示す. 手順①から順に確認していく. 手順①で, A_1を知るサーバは無い. 手順②で, A_1を知るサーバは無いので秘密情報aは漏洩しない. 手順③で, 攻撃者設定(2)よりハッシュ値の入力$a_{1,0}, \dots, a_{1,k-1}$およびこれらから求まる$a_1$は漏洩しない.

$$H(a_1, a_{1,0}, \dots, a_{1,k-1}) = H(a_1, a_{1,0}, \dots, a_{1,k-1} | h_{a_{1,0}}, \dots, h_{a_{1,k-1}})$$

手順④で, α_1, a_1, α_2はお互いに秘匿し合うためこの 3 値は漏洩しない.

$$H(\alpha_1, a_1, \alpha_2) = H(\alpha_1, a_1, \alpha_2 | \alpha_1 a_1, \alpha_2 a_1)$$

手順⑤で, 攻撃者設定(1)より, 変換用乱数$\varepsilon_1, \varepsilon_2$は誰も知らないので, 乱数比から$\alpha_2$や$A_1$は漏洩しない.

$$H(\alpha_2, A_1) = H\left(\alpha_2, A_1 | \frac{\alpha_2}{\varepsilon_1}, \frac{\alpha_2 A_1}{\varepsilon_2}\right)$$

手順⑥の計算はローカルで行われるため機密性に影響は無い. 手順⑦では, α_2, a_1は誰も知らないので, aは漏洩しない. 手順⑧は, 独立した別の乱数を用いた処理を行うため, 同様に安全性が示せる. 以上の議論より, 以下が言える.

$$H(a) = H(a | \, all\ opened\ values\ during\ 3.4)$$

以上の議論より, 3.4 の秘密分散は安全性要件(1)の機密性を持つことが言える.

4.2 秘匿計算に対する機密性

4.1 と同様に, 秘匿計算を行う事により honest な入力者が入力した秘密情報が攻撃者に漏洩しないことを示す. まず, 3.6 の秘匿計算において, honest なユーザが入力した秘密情報または復元結果が攻撃者に漏洩しないことを示す. 4.1 と同様に, 手順①から順に確認していく. 手順①は, 3.4 の手順⑤の議論と同様であり, 以下が言える.

$$H(\alpha_1, \beta_1, \gamma_1, \delta_1) = H\left(\alpha_1, \beta_1, \gamma_1, \delta_1 | \frac{\delta_1}{\gamma_1 \varepsilon_1}, \frac{\delta_1}{\alpha_1 \beta_1 \varepsilon_2}\right)$$

よって, 手順①では以下が言える.

$$H(a_1, b_1, c_1) = H\left(a_1, b_1, c_1 | \frac{\delta_1}{\gamma_1 \varepsilon_1}, \frac{\delta_1}{\alpha_1 \beta_1 \varepsilon_2}\right)$$

手順②の計算はローカルで行われるため機密性に影響は無い. 手順③では, δ_1,は誰も知らないので, $(c_1 - a_1 b_1)$は漏洩しない. 手順④は, $H(\delta_1)$の議論と同様あり, 以下が言える.

$$H(\alpha_1, \beta_1, \alpha_2, \beta_2, \gamma_2, \delta_2)$$
$$= H\left(\alpha_1, \beta_1, \alpha_2, \beta_2, \gamma_2, \delta_2 | \frac{\delta_2}{\alpha_2 \beta_2 \varepsilon_3}, \frac{\delta_2}{\alpha_2 \beta_1 \varepsilon_4}, \frac{\delta_2}{\alpha_1 \beta_2 \varepsilon_5}, \frac{\delta_2}{\gamma_2 \varepsilon_6}\right)$$

よって, 手順④まででは以下が言える.

$$H(a_1, b_1, c_1)$$
$$= H\left(a_1, b_1, c_1 | \frac{\delta_2}{\alpha_2 \beta_2 \varepsilon_3}, \frac{\delta_2}{\alpha_2 \beta_1 \varepsilon_4}, \frac{\delta_2}{\alpha_1 \beta_2 \varepsilon_5}, \frac{\delta_2}{\gamma_2 \varepsilon_6}\right)$$

手順⑤の計算はローカルで行われるため機密性に影響は無い. 手順⑥では, δ_2,は誰も知らないので, $\{(ab + c) + (c_1 - a_1 b_1)\}$は漏洩せず, 今までの議論より$\alpha_2, \beta_2, \gamma_2, a_1, b_1, c_1$も漏洩しないため, 秘密情報$a, b, c$は漏洩しない事が言える. 手順⑦は, 独立した別の乱数を用いた処理を行うため, 同様に安全性が示せる. 以上の議論より, 以下が言える.

$$H(a, b, c, ab + c)$$
$$= H(a, b, c, ab + c | \, all\ opened\ values\ during\ 3.6)$$

次に, 3.7 の秘匿計算を考える. 手順①は, 攻撃者設定(2)より, ハッシュ値からその入力は求まらない.

$$H(w_1, w_{1,0}, \dots, w_{1,k-1})$$

$$= H\left(w_1, w_{1,0}, \dots, w_{1,k-1} | h_{w_{1,0}}, \dots, h_{w_{1,k-1}}\right)$$

手順②で，δ_1, w_1はお互いに秘匿し合うため，この 2 値は漏洩しない．

$$H(\delta_1, w_1) = H(\delta_1, w_1 | \delta_1 w_1)$$

また，3.7 の処理を行った場合，3.6 の手順⑥が以下のように変化するが，この処理においても安全性は同様に証明できる．

$$[\delta_2\{(ab + c) + (w_1)\}]_i$$
$$= [\delta_2\{(ab + c) + (c_1 - a_1 b_1)\}]_i + [\delta_2 w_1]_i$$
$$= [\delta_2\{(ab + c) + (c_1 - a_1 b_1)\}]_i + \delta_1 w_1 \times \left(\frac{\delta_2}{\delta_1 \varepsilon_7}\right) \times [\varepsilon_7]_i$$

以上の議論より，以下が言える．

$$H(a, b, c, ab + c)$$
$$= H(a, b, c, ab + c | \textit{ all opened values during } 3.6 \textit{ and } 3.7)$$

つまり，提案方式の秘匿計算は安全性要件(1)の機密性を持つことが言える．

4.3 秘密情報の復元に対する完全性

秘密情報の復元に関して，復元処理 3.5 において，honest な復元者に対し偽の値が復元されないことを示す．今までと同様に，手順①から確認していく．手順①で，復元者が得る復元処理に用いる値は以下の 4 値である．

$$\alpha_2(a + a_1)', \alpha_4(a + a_2)', \alpha'_{2,j}, \alpha'_{4,j}$$

ここで，3.4 手順⑥より，$\alpha_2(a + a_1)', \alpha_4(a + a_2)'$ は以下のように表せる．ただし，$p_1, \dots, p_6$は，3.4 手順⑥の各項における正しい値との比である．

$$\alpha_2(a + a_1)' = \alpha_2(p_1 a + p_2 a_1 + (p_1 - p_3)A_1)$$
$$\alpha_4(a + a_2)' = \alpha_4(p_4 a + p_5 a_2 + (p_4 - p_6)A_2)$$

3.5 の手順②および手順③では，攻撃者設定(2)より不正があれば検出できるので 3.5 手順④では正しいa_1, a_2が計算できる．次に，3.5 手順⑤と⑥をまとめて考える．再び，3.5 手順⑥の各項に対して正しい値との比を$q_1, \dots, q_4$として，a_1を以下のように表記できる．ただし，$(a + A_1), (a + A_2)$は分散時に全サーバが共通の値を受け取っており，この項へ不正はできない．

$$[a'_1]_i = [(p_1 q_1 - 1)a + p_2 q_1 a_1 + (p_1 - p_3)q_1 A_1 + (q_2 - 1)A_2]_i$$
$$[a'_2]_i = [(p_4 q_3 - 1)a + p_5 q_3 a_2 + (p_4 - p_6)q_3 A_2 + (q_4 - 1)A_1]_i$$

以上より，手順⑦の検証は以下となる．

$$(p_1 q_1 - 1)a + (p_2 q_1 - 1)a_1 + (p_1 - p_3)q_1 A_1 + (q_2 - 1)A_2 = 0\,?$$
$$(p_4 q_3 - 1)a + (p_5 q_3 - 1)a_2 + (p_4 - p_6)q_3 A_2 + (q_4 - 1)A_1 = 0\,?$$

ここで，復元する情報aの入力者が攻撃者か否かで，議論を別々に進める．

[1] aの入力者が攻撃者ではない場合

この場合，攻撃者は a, A_1, A_2 を知らない．さらに a_1, a_2 も知らないため，手順⑦の検証を通過するには各項が 0 となる必要がある．この場合，検証を通過するためには以下が要求される．

$$p_1 q_1 = 1, \quad p_2 q_1 = 1, \quad p_1 = p_3, \quad q_2 = 1$$
$$p_4 q_3 = 1, \quad p_5 q_3 = 1, \quad p_4 = p_6, \quad q_4 = 1$$

まとめると，

$$p_1 = p_2 = p_3, \quad p_4 = p_5 = p_6, \quad q_2 = q_4 = 1$$

この時，復元者が 3.5 手順①で受け取った $\alpha_2(a + a_1)', \alpha_4(a + a_2)'$ は以下のように表せる．

$$\alpha_2(a + a_1)' = \alpha_2(p_1 a + p_1 a_1) = \alpha_2 p_1(a + a_1)$$
$$\alpha_4(a + a_2)' = \alpha_4(p_4 a + p_4 a_2) = \alpha_4 p_4(a + a_2)$$

上記の通り，ここでは差分p_1, p_4は検出できないが，a, a_1には同じ差分p_1，a, a_2には同じ差分p_4というように，各項に生じる差分が等しいことが安全性証明で非常に重要となる．また，この時 3.5 手順⑧は以下のような計算となる．ただし正しい値との比をt_1, t_2として $\alpha'_2 = t_1 \alpha_2, \alpha'_4 = t_2 \alpha_4$ と定義する．

$$\frac{\alpha_2 p_1(a + a_1)}{t_1 \alpha_2} - \frac{\alpha_4 p_4(a + a_2)}{t_2 \alpha_4} - (a_1 - a_2)$$
$$= \left(\frac{p_1}{t_1} - \frac{p_4}{t_2}\right)a + \left(\frac{p_1}{t_1} - 1\right)a_1 - \left(\frac{p_4}{t_2} - 1\right)a_2$$
$$= 0\,?$$

ここで，攻撃者は a, a_1, a_2 を知らないことから，上記の式のすべての項を 0 と調整するしかない．よって，3.5 手順⑧の検証を通過するための条件は，以下となる．

$$\frac{p_1}{t_1} = \frac{p_4}{t_2} = 1$$

最後に，手順⑨を考える．復元者が得る復元値は以下のように表せる．

$$\frac{\alpha_2 p_1(a + a_1)}{t_1 \alpha_2} - a_1 = \frac{p_1}{t_1}a + \left(\frac{p_1}{t_1} - 1\right)a_1 = a$$

よって，すべての検証を通過するなら攻撃者ではない honest な入力者が入力した正しい値aが復元者に対して復元されることが言える．

[2] aの入力者が攻撃者である場合

手順⑦の検証を再度示す．

$$(p_1 q_1 - 1)a + (p_2 q_1 - 1)a_1 + (p_1 - p_3)q_1 A_1 + (q_2 - 1)A_2 = 0\,?$$
$$(p_4 q_3 - 1)a + (p_5 q_3 - 1)a_2 + (p_4 - p_6)q_3 A_2 + (q_4 - 1)A_1 = 0\,?$$

今回の場合，攻撃者は a, A_1, A_2 を知るので，a_1, a_2 以外の項を調整し，上記 2 値を 0 とすることが可能である．よって，ここでは不正検出できず，以降の処理は以下の値で行われる．

$$\alpha_2(a + a_1)' = \alpha_2(p_1 a + p_2 a_1 + (p_1 - p_3)A_1)$$
$$\alpha_4(a + a_2)' = \alpha_4(p_4 a + p_5 a_2 + (p_4 - p_6)A_2)$$

すると，手順⑧の検証は，以下となる．ただし，正しい値との比をt_1, t_2として，$\alpha'_2 = t_1 \alpha_2, \alpha'_4 = t_2 \alpha_4$ と定義する．

$$\frac{\alpha_2(p_1a+p_2a_1+(p_1-p_3)A_1)}{t_1\alpha_2}$$

$$-\frac{\alpha_4(p_4a+p_5a_2+(p_4-p_6)A_2)}{t_2\alpha_4}-(a_1-a_2)$$

$$=\left(\frac{p_1}{t_1}-\frac{p_4}{t_2}\right)a+\left(\frac{p_2}{t_1}-1\right)a_1+\frac{(p_1-p_3)}{t_1}A_1-\left(\frac{p_5}{t_2}-1\right)a_2$$

$$-\frac{(p_4-p_6)}{t_2}A_2=0\,?$$

同様に，攻撃者は a,A_1,A_2 を知るので，a_1,a_2 以外の項を調整し，上記 2 値を 0 とすることが可能である．ただし，a_1,a_2 の項は調整できないので，以下の条件が必要である．

$$\frac{p_2}{t_1}=\frac{p_5}{t_2}=1$$

最後に，手順⑨を考える．復元者が得る復元値は以下のように表せる．

$$\frac{p_1}{t_1}a+\left(\frac{p_2}{t_1}-1\right)a_1+\frac{(p_1-p_3)}{t_1}A_1=\frac{p_1}{t_1}a+\frac{(p_1-p_3)}{t_1}A_1$$

ここで，手順⑧の通過条件より $p_2=t_1$ を代入すると，

$$\frac{p_1}{p_2}a+\frac{(p_1-p_3)}{p_2}A_1$$

が復元される．よって，復元値はaではない．しかし，aの入力者は攻撃者であり分散値を正しく生成していないため，a が復元されないことは当然と言える．よって，aの入力者は実際には何を分散したかを考える．生成された分散値は，以下であった．

$$\alpha_2(p_1a+p_2a_1+(p_1-p_3)A_1)$$

ここで，提案方式の復元方法(3.5 手順⑨)は分散値を α_2 で除算した後 a_1 で減算するという処理であるため，以下のように分散値の中身を a_1 の部分とその他の部分に分ける．

$$\alpha_2(p_1a+p_2a_1+(p_1-p_3)A_1)=\alpha_2(A+p_2a_1)$$

不正の有無にかかわらず，分散値はすべて上記のように表せることは自明である．3.5 復元処理を考えると，この分散値とα_2',a_1から復元される値は，以下のように一意に表せる．

$$\frac{A}{p_2}$$

例えば，不正がない場合は$\mathrm{A}=a,p_2=1$であるから，

$$\text{復元値}=\frac{A}{p_2}=\frac{a}{1}=a$$

となり正しい．また，前述の「a の入力者が攻撃者ではない場合」では，$\mathrm{A}=p_1a,p_2=p_1$ であるから，

$$\text{復元値}=\frac{A}{p_2}=\frac{p_1a}{p_1}=a$$

となり正しい．つまり，aの入力者は不正により自ら分散値を操作し，実際にはA/p_2 を分散したと考えることができ，復元値がaではなく A/p_2 であることは完全性に全く影響はない．まとめると，分散値は以下のように表すことができ，この形で表せない場合検証は通らない．

$$\alpha_2(p_1a+p_2a_1+(p_1-p_3)A_1)=\alpha_2\left(\frac{A}{p_2}+a_1\right)=\alpha_2(a'+a_1)$$

※　a'は入力者が真に入力した値．

4.4 積和演算結果の復元に対する完全性

積和演算結果の復元に関して，復元処理 3.5 において，honest な復元者に対し偽の値が復元されないことを示す．ここで，4.3 での議論を活用する．まず，aの入力者が攻撃者でない場合は，分散値は以下のように表せなければ検証は通らない．

$$\alpha_2p_1(a+a_1),\qquad \alpha_4p_4(a+a_2)$$

また，aの入力者が攻撃者である場合は，分散値は以下のように表せなければ検証は通らない．

$$\alpha_2(a'+a_1),\alpha_4(a'+a_2)$$

ここで，4.3 の議論より$\alpha_2(a'+a_1),\alpha_4(a'+a_2)$ は正しい分散値である．しかし，$\alpha_2p_1(a+a_1),\alpha_4p_4(a+a_2)$ は入力者が分散値生成で不正をしていないのにも関わらずサーバの不正によって差分p_1,p_4が発生している．このまま復元する分には問題ないが，秘匿計算において問題が発生する可能性がある．よって，本節の議論で用いる分散値(積和演算の入力)は，入力者が不正をしない場合である $\alpha_2p_1(a+a_1)$, $\alpha_4p_4(a+a_2)$ をベースに考える．ここで，積和演算では a に対する残りの 2 つの分散値 $(\alpha_1a_1)',(\alpha_3a_2)'$ を使用する．よって，a に対する分散値を以下のように再度定義する．

$$p_1\alpha_1a_1,p_2\alpha_2(a+a_1),p_3\alpha_3a_2,p_4\alpha_4(a+a_2)$$

また，積和演算の残りの 2 入力b,cも，同様に定義する。

$$q_1\beta_1b_1,q_2\beta_2(b+b_1),q_3\beta_3b_2,q_4\beta_4(b+b_2)$$

$$r_1\gamma_1c_1,r_2\gamma_2(c+c_1),r_3\gamma_3c_2,r_4\gamma_4(c+c_2)$$

上記の分散値を用いて，3.6 秘匿計算を考える．手順①から手順③で計算する分散値は復元には用いないため，手順④から手順⑥を考える．手順⑤の計算は，以下のように表すことができる．ただし，攻撃者の攻撃によって，乱数比に発生した差分を，正しい値との比$x_1,\dots,x_4$を用いて表す．

$$\begin{aligned}&\delta_2\{(ab+c)+(c_1-a_1b_1)\}'\\&=\delta_2\{x_1p_2q_2(a+a_1)(b+b_1)-x_2p_2q_1(a+a_1)b_1\\&\qquad-x_3p_1q_2a_1(b+b_1)+x_4r_2(c+c_1)\}\\&=\delta_2\{P_1(a+a_1)(b+b_1)-P_2(a+a_1)b_1-P_3a_1(b+b_1)\\&\qquad+P_4(c+c_1)\}\\&=\delta_2\{P_1ab+P_4c+(P_1-P_3)a_1b+(P_1-P_2)ab_1\\&\qquad+(P_1-P_2-P_3)a_1b_1+P_4c_1\}\end{aligned}$$

同様に，

$$\begin{aligned}&\delta_4\{(ab+c)+(c_2-a_2b_2)\}'\\&=\delta_4\{Q_1ab+Q_4c+(Q_1-Q_3)a_2b+(Q_1-Q_2)ab_2\\&\qquad+(Q_1-Q_2-Q_3)a_2b_2+Q_4c_2\}\end{aligned}$$

よって，積和演算結果の復元に関して，3.5 の手順①では，復元者は以下を受け取る．ただし，t_1,t_2は δ_2',δ_4'に対する正しい値との比である．

$$\begin{aligned}&\delta_2\{P_1ab+P_4c+(P_1-P_3)a_1b+(P_1-P_2)ab_1\\&\qquad+(P_1-P_2-P_3)a_1b_1+P_4c_1\}\end{aligned}$$

$$\delta_4\{Q_1ab + Q_4c + (Q_1 - Q_3)a_2b + (Q_1 - Q_2)ab_2 + (Q_1 - Q_2 - Q_3)a_2b_2 + Q_4c_2\}$$
$$t_1\delta_2, \quad t_2\delta_4$$

上記 4 値で，3.5 復元処理を考える．手順①は手順⑧の議論でまとめて行う．まず，手順②から手順④では，不正検出されなければ正しい乱数 $a_1, a_2, b_1, b_2, c_1, c_2$ を復元者は得る．次に，手順⑤から手順⑦であるが，これはすでに考慮して議論しているため，手順⑧を考える．手順⑧の検証は以下となる．

$$\frac{\delta_2\{(ab + c) + (c_1 - a_1b_1)\}'}{t_1\delta_2} - \frac{\delta_4\{(ab + c) + (c_2 - a_2b_2)\}'}{t_2\delta_4} - \{(c_1 - a_1b_1) - (c_2 - a_2b_2)\}$$
$$= \left(\frac{P_1}{t_1} - \frac{Q_1}{t_2}\right)ab + \left(\frac{P_4}{t_1} - \frac{Q_4}{t_2}\right)c + \frac{(P_1 - P_3)}{t_1}a_1b + \frac{(P_1 - P_2)}{t_1}ab_1 - \frac{(Q_1 - Q_3)}{t_2}a_2b - \frac{(Q_1 - Q_2)}{t_2}ab_2 + \left\{\frac{(P_1 - P_2 - P_3)}{t_1} + 1\right\}a_1b_1 + \left(\frac{P_4}{t_1} - 1\right)c_1 - \left\{\frac{(Q_1 - Q_2 - Q_3)}{t_2} + 1\right\}a_2b_2 - \left(\frac{Q_4}{t_2} - 1\right)c_2 = 0\ ?$$

ここで，攻撃者は $a_1, a_2, b_1, b_2, c_1, c_2$ を知らないため，ab, c の項以外はすべて 0 とならなければならない．よって，手順⑧の検証通過条件は，

$$\frac{P_1}{t_1} = \frac{P_2}{t_1} = \frac{P_3}{t_1} = \frac{P_4}{t_1} = \frac{Q_1}{t_2} = \frac{Q_2}{t_2} = \frac{Q_3}{t_2} = \frac{Q_4}{t_2} = 1$$

このとき，手順⑨で復元者が復元する値は，

$$\frac{P_1}{t_1}ab + \frac{P_4}{t_1}c + \frac{(P_1 - P_3)}{t_1}a_1b + \frac{(P_1 - P_2)}{t_1}ab_1 + \left\{\frac{(P_1 - P_2 - P_3)}{t_1} + 1\right\}a_1b_1 + \left(\frac{P_4}{t_1} - 1\right)c_1 = ab + c$$

であり，honest な入力者が入力した正しい値が復元される．次に議論すべきは 3.7 であるが，本節の議論に w_1 を追加するだけである．3.5 の手順④で正しい w_1 を復元者は得ることができ，分散値に発生する差分を持った w_1 を正しい値との比 s_1 を用いて s_1w_1 とあらわすと，本節の検証式に以下を追加するだけである．

$$s_1w_1 - w_1 = (s_1 - 1)w_1 = 0\ ?$$

明らかに $s_1 = 1$ が要求され，同様に完全性が証明できる．

4.5 $n > k$に対する安全性

今までの議論では，簡単のため $n = k$ の設定であった．$n > k$の場合は 3.3 前提(4)を適用するが，安全性の証明は同様に行える．

5. 結論

本稿では，$n < 2k - 1$において，下記の active な攻撃者に対して安全な秘匿計算手法を提案した．

(1) 変換用乱数組$[\varepsilon_h]_i, [\varepsilon_{h,j}]_i^X$で求まる$\varepsilon_h$を知らない．

(2) 計算能力は有限であり，ハッシュ値$h_x = h(x)$からその入力xを計算できない．

(3) t入力 1 出力の演算に関して，攻撃者$(\in P)$の結託は最大$t - 1$人である．

(4) 最大$k - 1$台のサーバが持つ情報を知れる．

今後は，変換用乱数の安全かつ効率的な生成方法の検討や，従来方式との詳細な比較を行う事が挙げられる．

参考文献

[1] 株式会社 日立製作所, https://www.hitachi.co.jp/products/it/bigdata/bigdata_ai/index.html, 閲覧 2019/8/15

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

[3] 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.1038-1049 (Mar. 2018)

[4] 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

[5] 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

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

[7] L. Ham, M. Fuyou, C. Chang, “Verifiable secret sharing based on the Chinese remainder theorem”, SECURITY AND COMMUNICATION NETWORKS, 2014

[8] 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)

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

[10] 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

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

[12] 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

[13] 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

[14] M. Keller, V. Pastro, D. Rotaru, “Overdrive: Making SPDZ Great Again”, EUROCRYPT 2018: Advances in Cryptology – EUROCRYPT 2018 pp 158-189.

[15] T. Araki, A. Barak, J. Furukawa, T. Lichter, Y. Lindell, “Optimized Honest-Majority MPC for Malicious Adversaries — Breaking the 1 Billion-Gate Per Second Barrier”, 38th IEEE Symposium on Security and Privacy, 2017

[16] D. Beaver. Efficient multiparty protocols using circuit randomization. In J. Feigenbaum, editor, CRYPTO, volume 576 of Lecture Notes in Computer Science, pages 420-432. Springer, 1991.