情報処理学会論文誌 Vol.62 No.9 1457–1475 (Sep. 2021)

$n < 2k - 1$ において malicious な攻撃者に対しても安全な秘密分散を用いた秘匿計算とその拡張

落合 将吾[1,a)] 岩村 恵市[1]

受付日 2020年11月27日, 採録日 2021年6月7日

概要：一般に，(k, n) 閾値秘密分散法を用いた秘匿計算では，$n < 2k - 1$ において無条件に安全な秘匿計算は不可能とされている．そのため，著者らのグループでは $n < 2k - 1$ において安全に秘匿計算が実行できる条件を探索するというアプローチをとり，その 1 つの研究結果として semi-honest な攻撃者に対して，(1) 秘匿計算結果および乱数に 0 を含まない，(2) 攻撃者が知らない乱数を用いた 1 に対するシェア集合がある，(3) 演算の連続において各サーバが扱うシェア集合内のシェアの位置は固定される，という 3 つの条件の下で安全な秘匿計算手法を示した．しかし，$n < 2k - 1$ に適用できる代表的な秘匿計算方式である SPDZ が malicious な攻撃者に対して安全であるため，出力の正当性を検証できない著者らのグループで提案されている手法は完全性の観点で劣る．そのため，本稿では著者らのグループの方式と同様にして $n < 2k - 1$ への適用を実現し，かつ malicious な攻撃者に対しても安全な方式を提案する．

キーワード：秘密分散，秘匿計算，マルチパーティ計算，malicious adversary，$n < 2k - 1$

Maliciously Secure MPC for $n < 2k - 1$ Based on Secret Sharing Scheme and Its Extension

Shogo Ochiai[1,a)] Keiichi Iwamura[1]

Received: November 27, 2020, Accepted: June 7, 2021

Abstract: In general, the (k, n)-threshold secret sharing scheme does not allow unconditionally secure multiparty computation (MPC) for $n < 2k - 1$. Therefore, our research team has been focusing on conditions under which the secure MPC can be performed securely for $n < 2k - 1$. One of their research findings shows that their secure MPC protocol for $n < 2k - 1$ is allowed under these three conditions against semi-honest adversaries: (1) the result of secure computation and any random values do not contain 0; (2) there is a set of distributed values for 1 using random numbers unknown to the adversaries; (3) the position of the values in the set of distributed values handled by each server is fixed during the sequence of computation. However, their protocol is not secure against malicious adversaries while SPDZ, which is a representative secure MPC protocol, is secure against malicious adversaries. Therefore, we propose a secure MPC protocol against malicious adversaries based on the protocol of our research team.

Keywords: secret sharing scheme, secure computation, MPC, malicious adversary, $n < 2k - 1$

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，データの取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，秘匿計算がこれを可能にする．

秘匿計算を実現するアプローチとして，準同型暗号用いる方式と，秘密分散法を用いる方式がよく知られている．

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

完全準同型暗号 [18] は暗号文のまま加算と乗算が可能であるが計算量が多く，演算に多大な時間がかかるという問題がある．そのため，著者らは処理が軽い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，秘密情報に基づき n 個の値（シェアと呼ぶ）を生成してそれらを n 台のサーバに1個ずつ配布する方式であり，以下の2つの性質を持つ．

(1) n 個のシェアのうち k 個以上のシェアから秘密情報を復元できる．
(2) k 個未満のシェアから秘密情報に関する情報はいっさい得られない．

ただし，$n < k$ では秘密情報を復元できず，$k = 1$ では秘密情報をそのままサーバへ送信することを意味するため，$n \geq k \geq 2$ が要求される．Shamir 法ではシェア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，著者らのグループである神宮らは TUS1 方式 [3] を，Aminuddin らは TUS2 方式 [4] を，鴇田らは TUS3 方式 [5] を提案している．これらの方式（以降 TUS 方式）は，秘密情報に乱数を掛けることで秘匿して公開情報とし，スカラー量として乗算するというアプローチが共通であり，これにより次数変化のない秘匿乗算を実現している．ただし，TUS 方式は semi-honest な攻撃者を想定しているため，サーバを corrupt（乗っ取り，管理下に置くこと）してプロトコルから逸脱させる malicious な攻撃者によるデータ改ざんなどの攻撃に対して安全性を持たない．実際にデータの利活用を行う場合を考えると，サーバに保存されたデータに対して改ざんなどの攻撃が行われた場合，その改ざんされたデータを入力とした秘匿計算は正当な結果とならず，無意味な計算となってしまう．よって，データの改ざんなどが行われる可能性を考慮した malicious な攻撃者に対して安全な手法が注目されている．malicious な攻撃者に対して安全な手法に関して，秘密分散法においてシェア以外に検証用の値を追加配布することで，各参加者が受け取ったシェアおよび復元値を検証する方法が一般的であり，検証可能秘密分散（VSS）と呼ばれている．また，複数人のプレイヤ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，その目的から出力の正当性が検証可能な方式も多い．現在代表的な MPC 手法の多くはシェア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 $n = k$ という制限はないが，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，本稿では 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，$n < 2k-1$ を含む任意の $n \geq k$ に対して適用でき，計算結果の正当性を検証可能にすることで malicious な攻撃者に対しても安全な方式を提案する．以下に，本稿の具体的貢献を示す．

- **高い安全性を持つ秘匿計算手法の提案**

本稿の提案方式は，攻撃者の割合，n，k の制限，攻撃者の計算能力，攻撃者の振る舞い，という4つの観点において下記 (1) から (4) に示すような高い安全性を実現している．これら4つの観点は，順に攻撃者が過半数か否か，n，k の制限が少ないか多いか，攻撃者の計算能力は無限か有限か，攻撃者はプロトコルから逸脱するか否か，のように安全性レベルを2種類に分けることができる．従来方式ではこの4つの観点のうち最低1つが安全性レベルの低い設定になっており，著者らの知る限り本稿の提案方式はこれらすべてを安全性レベルの高い設定で実現する唯一の手法である．

(1) Dishonest majority を想定．
(2) $n < 2k-1$ においても実行可能．
(3) 秘密情報の機密性を情報理論的安全に確保．
(4) Malicious な攻撃者を想定．

- **TUS 方式に適した検証方法の提案**

以下の2つの手法を組み合わせて TUS 方式に適した検証方法を提案する．

– **2通りでの秘匿計算による計算結果の検証**：TUS3 方式では秘密情報に1を加えた値を一体として扱ったが，提案方式では異なる2つの乱数を加えて2通りのシェアを作り，2通りの秘匿計算を行う．乱数を加えたままの2通りの復元値の差分を取れば，秘密情報を秘匿したまま乱数の差分を得ることができる．この差分の正当性は後述のコミットメントスキームによる検証で得られる正当な乱数を用いて検証できる．
– **コミットメントスキームによる秘密情報の検証**：秘密情報と一体になった乱数を秘匿計算で個別に取り出して復元し，コミットメントスキームによりその正当性を検証する．これによって，検証した乱数と一体になった秘密情報を間接的に検証できる．

- **全入力検証による演算中の全秘匿乗算結果の検証処理の削減**

Malicious な攻撃者に対して安全な秘匿計算手法には，計算結果の正当性を検証する処理が必要である．一般に，検証が計算結果のみに対してでは不十分であり，[9], [10], [13], [17] のように多くの方式では計算結果に加えて全秘匿乗算結果の検証を行う．提案方式では，分散処理で秘密情報と乱数を対応させ，計算結果に加えて全入力（秘密情報）の検証

を行う．前述のとおり秘密情報の検証にコミットメントを採用することで，攻撃者が秘匿計算中などに秘密情報と乱数の対応付けを壊した場合に不正検出できる．よって，提案方式では秘匿乗算など演算中の不正を検証する処理が不必要となるため，全入力と計算結果に対しての検証により不正が検出できる．

- **柔軟な安全性レベルの変更**

5章で議論するが，提案方式の安全性において，想定する攻撃者の計算能力は秘密情報に対応させる乱数の扱いにのみ依存する．よって，3章で示す提案方式におけるコミットメントスキームは，たとえばシンプルなハッシュ関数に置き換えることにより安全性レベルを下げる代わりにより効率化することが可能である．また，秘匿計算に直接関与しない第三者を想定できる場合，機密性だけでなく完全性に対しても計算能力が無限な攻撃者を想定できる．以上のように，提案方式では秘匿計算を実行する状況や環境に応じて，コストや安全性を選択できる．

以下，本稿の構成を示す．まず，2章では従来方式や3章で利用するコミットメントスキームの説明などを行う．3章で提案方式を示して4章でその安全性を証明し，5章では 安全性に関する拡張方法を議論する．そして，6章で他方式との比較を行い，7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，n 人の参加者が各々1ずつ秘密情報に関する値（シェア）を保持し，設定した閾値 k により以下の2つの性質（定義）を持つ．

(1) k 個未満のシェアからは，秘密情報に関する情報はいっさい得られない．
(2) 任意の k 個以上のシェアからは，秘密情報が一意に得られる．

代表例として，Shamir の方式 [2] や Kurihara らの XOR による方式 [7], [8]（以降 XOR 法），加法的秘密分散法などが提案されている．秘密分散法は情報セキュリティの3要素の1つである機密性の確保を主の目的としており，n, k が柔軟に設定可能なら可用性を確保でき，VSS のように完全性を確保する方式も存在する．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) 閾値秘密分散法の方が，はるかに計算量が少なく高効率である．

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，1回の秘匿乗算を行うと乗算結果に関する閾値が k から $2k-1$ へ増加してしまう．いい換えれば，$n < 2k-1$ の設定では秘匿乗算結果を復元できないということであり，これは Shamir 法のシェアが多項式によって生成されていることが原因となっている．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て次数変化のない秘匿乗算を実現した [3]．しかし，この方式は安全性の観点から加減算と乗算の組合せが不可能であるという制約があった．この問題を Aminuddin らは攻撃者が知らない “1” のシェア集合という Trusted Third Party（TTP）のような存在を仮定して解決した [4]．この “1” のシェア集合は，1は乗算しても結果は変わらず，かつ TUS 方式では分散する値には乱数を乗じるため，1に乗ずる乱数とそれを構成する乱数のシェアからなる集合である．TTP は “1” のシェア集合を生成して各サーバに送り，秘匿計算時に各項に乗算することによって，各項はこの乱数により秘匿されている状態となり，情報漏洩を防いでいる．“1” のシェア集合に用いられる乱数は TTP のみが知る．この方式は，秘匿計算結果および乱数に0を含まないなど3つの条件において，情報理論的安全性を持つ．しかし，Shamir 法におけるシェアの復元が多いため，処理速度に問題があった．そこで鴇田らは，Kurihara 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．TUS 方式において秘密情報 a を秘匿する乱数は1個（下記 α）であり，以下に TUS3 方式の分散と復元に関する処理を示す．TUS3 方式で生成される乱数は0を含まない有限体 F_p 上の値であり，すべての演算は p を法として行われる．ただし，秘密情報は0を含んで $p-1$ より小さな数であり，ブロードキャストを含め，すべての通信は安全に行われる．

[記号定義]

$\overline{[a]}_i$：値 a に対するサーバ S_i が保持するシェア．

$[a]_i$：値 a に関するサーバ S_i が保持するシェア集合．

[分散処理]

① 入力者は，k 個の乱数 $\alpha_0, \alpha_1, \ldots, \alpha_{k-1}$ 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 $a \in \{0, 1, \ldots, p-2\}$ に1を加算し，$(a+1)$ に乱数 α 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．

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

[復元処理]

① 復元者は，k 台のサーバ S_j を選択し，それらが持つシェア集合 $[a]_j$ を収集する．

② 復元者は，収集したシェア集合を復元し，$\alpha(a+1)$, $\alpha_0, \ldots, \alpha_{k-1}$ を得る．

③ 復元者は，以下のように秘密情報 a 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，秘匿積和演算を基本演算とするが，その組合せによる任意の t 入力 1 出力の四則演算に関して，$t-1$ 個以下の入出力を知る semi-honest な攻撃者が，自らが知る情報と $k-1$ 台のサーバから得る情報から残り 2 つの未知な入出力を知ろうとしても，その意図は実現されない．すなわちその攻撃者に対して情報理論的安全であることが示されている．

2.3 従来の malicious な攻撃者に対する MPC

MPC は複数人のプレイヤが持つプライベートな値を入力として関数 f を計算し，各プレイヤはその関数 f の出力のみを知るプロトコルである．TUS 方式は semi-honest な攻撃者に対して安全な方式であるが，malicious な攻撃者に対して安全な MPC も活発に研究されており，その代表例として Damgård らによる SPDZ（SPDZ-1 [9]，SPDZ-2 [10]）やその改良方式（MASCOT [11], [12]），また，Araki らによる文献 [13]，Chida らの文献 [17] などがあげられる．SPDZ およびその改良方式（以降，SPDZ 系列）は $n = k$ にのみ適用可能であり，攻撃者が過半数である dishonest majority を想定している．また，加法的秘密分散を用いており，Beaver により提案された multiplication triple $(x, y, z = xy)$ [14] に対するシェア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）やコミットメントスキームを活用する．文献 [13] では Replicated 秘密分散を用いており，これは n 個のシェア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，文献 [13] では $n = 3$，$k = 2$ で限定された手法が提案されている．Replicated 秘密分散は加法的秘密分散に基づくため，SPDZ 系列と同様に秘匿乗算は multiplication triple $(x, y, z = xy)$ を用いて実現している．また，この方式は $k = 2$ より攻撃者が 1 人に制限されるため，$n = 3$ であるから honest majority を想定している．また，文献 [17] も honest majority を想定しており．大きな有限体を想定することで偶然検証が成功する確率を無視できる程度に小さくして安全性を実現しており，このアプローチは提案方式でも採用する．代表的な MPC 手法の中で，$n < 2k - 1$ に適用でき，dishonest majority を想定した方式は著者らの知る限り SPDZ 系列のみである．しかし，SPDZ 系列は $n = k$ 限定で計算量的安全性を実現するため，これらの点などにおいて提案方式の優位性を示すことで明確に差別化する．ここで，一般に情報理論的安全な方式は，計算量的安全な方式よりコストが多くかかるため性能（計算量・通信量・記憶容量など）では基本的に劣り，TUS3 方式と同様に本稿での提案方式も SPDZ 系列よりコストが大きくなる．詳細に関しては，6 章で示す．

2.4 コミットメントスキーム [6]

コミットメントスキームは，その性質から malicious な攻撃者に対して安全な VSS や MPC に用いられ，情報をコミットする committer と情報を検証する verifier の 2 者間のプロトコルで，下記 2 フェイズで構成される．表記は文献 [6] を引用した．ここでの g はコミットメントを扱うための識別子のような役割を持つ．

(1) コミットフェイズ

メッセージ m に対し，Committer は $[C, (m, g)] = Commit(m)$ を実行し，C をコミットメントとして公開する（コミットメントをオープンするための値 g は場合によって出力される）．

(2) オープンフェイズ

Committer は，(m, g) を検証者に明らかにすることで，コミットメント C をオープンする．検証者は，コミットメント C とメッセージ m が一貫した値か検証できる（たとえば，$m = Open(C, m, g)$? を検証する）．

コミットメント C はメッセージ m と結び付けられるが（束縛性），メッセージ m は明らかにしない（秘匿性）．これらはコミットメントスキームにおいて重要な性質であり，両者を同時に無条件（完全または統計的）に実現することはできないことが知られている．つまり，コミットメントスキームは下記 2 種類に大別できる．

(1) 無条件（完全または統計的）な束縛性かつ計算量的な秘匿性を持つコミットメントスキーム
(2) 無条件（完全または統計的）な秘匿性かつ計算量的な束縛性を持つコミットメントスキーム

一般的には，秘匿性を重要視するシナリオが多いことから，(2) のタイプが多く利用されており，文献 [15] が有名である．また，本稿の提案方式でも (2) のタイプを採用する．(2) のタイプの実現方法は様々であり，ハッシュ関数を用いて構成することもできる [16]．たとえば，ハッシュ関数への入力に十分大きな乱数をビット連結させることで入力をハッシュ関数の出力より十分大きくすれば，特定のハッシュ値に対する入力が複数存在し，無限回の施行でもコミットされた値は特定できなくなる．本稿では採用するコミットメントスキームに関しては必要な性質のみを明示し，具体的な中身の議論は省略する．ただし，公平な比較のため，ラウンド数の計算などには比較対象である SPDZ2 で用いられているランダムオラクルを用いたコミットメントスキームを参考にする．

3. 提案方式

3.1 概要

本章では，(k, n) 閾値秘密分散法を用いて $n \geq k$ に適用

できる秘匿計算手法を示す．また，提案方式では TUS 方式と同様にスカラー倍では次数変化が起こらないことに着目し，スカラー量として秘密情報や計算結果を保持する．具体的には，秘密情報に対して乱数を加算したあと，乱数を乗算して秘匿して全サーバがそのまま保持する．また，秘匿乗算を行うために，秘密情報に加算した乱数を別の乱数で乗算して秘匿し，これも全サーバがそのまま保持する．さらに，これら 2 つの値をすべて異なる乱数を用いてもう 1 組生成する．つまり，これら 4 つの値に関して，スカラー量として全サーバが共通して保持する．一方，提案方式におけるほぼすべての乱数は k 個の乱数の積から構成され，これらは (1) $n = k$ なら生成したサーバがそのまま保持し，(2) $n > k$ なら生成したサーバが XOR 法で分散する．復元処理中の検証処理の際には，秘密情報を秘匿するために加算した乱数（1 つの秘密情報に対して 2 つ存在する）の正当な値を用いる．これらの乱数は復元処理まで攻撃者に知られてはならず，正当な値を用いるには攻撃者の意図する値への改ざんを防ぐ必要がある．この問題は，提案方式では 2.4 節で議論した (2) のタイプのコミットメントスキームを利用することで対応する．検証処理に関して，不正が発覚した時点で処理を終了する secure with abort を採用する．これは SPDZ 系列や文献 [17] など代表的な MPC で採用されており，不用意に処理を進めることにより発生しうる意図しない情報漏洩を防ぐ．また，提案方式では安全な通信路の存在を仮定して盗聴などは起こらないとし，通信路に関する安全性は省略する．本方式は TUS 方式と同様に $n \geq k$ に適用できるため，n，k が限定されている SPDZ 系列や文献 [13] などより強いデータの欠損耐性を持ち，演算中におけるサーバの故障などに対応できる．また，提案方式では秘密分散法を利用して秘匿計算を実現することから，$n \geq k \geq 2$ である．すべ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 で行われるが，煩雑になるため各式における mod p の表記は省略する．本プロトコルのエンティティは，入力者，復元者，n 台のサーバとするが，入力者や復元者を 1 台のサーバと 1 対 1 対応させ，サーバが入力を行ったり，復元値を得たりするという基本的な MPC と同様な設定と考えても本質的に違いはない．また，以降，特別に記載がない限り本稿では $l = 0, \ldots, k-1$ とする．

3.2 表記

n：秘匿計算に参加するサーバ数．
k：採用する秘密分散の閾値．
p：法とする十分大きな（たとえば 128 bit 以上）素数．
S_i：i 番目のサーバ（$i = 0, 1, \ldots, n-1$）．
$[x]_i$：x に対する，サーバ S_i が保持する線形性のある秘密分散法によるシェア（6 章の比較では Shamir 法とする）．
$[x]_i^{XOR}$：x に対する，サーバ S_i が保持する XOR 法によるシェア．
ε_h，$\varepsilon_{h,l}$：変換用乱数（$h = 1, 2, \ldots,$ 必要個数）．対応するシェアが信頼できる第三者により事前に n 台のサーバに秘密分散され，各サーバがローカルに保持する．
a，b，c：各入力者の秘密情報．入力者のみがローカルに保持する．
d：計算結果．本稿では積和演算結果 $d = ab + c$ を意味する．
$A_{x,j}$，$\alpha_{y,j}$：秘密情報 a に対して，サーバ S_j が生成してローカルに保持する乱数（$x = 1, 2, y = 0, \ldots, 5$）．
$\beta_{y,j}$：秘密情報 b に対して，サーバ S_j が生成してローカルに保持する乱数（$y = 0, \ldots, 5$）．
$\gamma_{y,j}$：秘密情報 c に対して，サーバ S_j が生成してローカルに保持する乱数（$y = 0, \ldots, 5$）．
$\delta_{z,j}$：計算結果 d に対して，サーバ S_j が生成してローカルに保持する乱数（$z = 0, 2, 3, 5$）．
$w_{1,j}$：3.8 節補正処理において，サーバ S_j が生成してローカルに保持する乱数．秘密情報と対応付けられる乱数 $\alpha_{1,j}$，$\beta_{1,j}$，$\gamma_{1,j}$ を更新する目的で計算に用いられる．
d_w：計算結果 d に対応付けられる乱数（$w = 1, 4$）．$d = ab + c$ の場合，$d_w = (\gamma_w - \alpha_w \beta_w)$ となる．

また，線形性のある秘密分散法によるシェアに関して，シェアどうしの加減算 $[x \pm y]_i = [x]_i \pm [y]_i$，定数 y による加減算 $[x \pm y]_i = [x]_i \pm y$，定数 y による乗算 $[xy]_i = y \times [x]_i$ が可能である．

3.3 前提条件

提案方式では，TUS3 方式と同様な以下の 3 つの前提条件を設定する．

(1) 乱数に 0 は用いられない．
(2) サーバ S_i は事前に十分な数の変換用乱数組 $([\varepsilon_{h,0}]_i^{XOR}, \ldots, [\varepsilon_{h,k-1}]_i^{XOR}, [\varepsilon_h]_i)$ を保持しており，各秘匿計算で毎回異なるものを使用する．ただし，$\varepsilon_h = \prod_{j=0}^{k-1} \varepsilon_{h,j}$ である．この必要個数は実行する計算や入力数に依存し，計算式中の項の数に依存する．たとえば，3.4 節手順④で 8 個利用し 3.5 節では利用していないことから，1 つの秘密情報を分散するためには 8 個必要となる．また，3.7 節手順①で 12 個利用していることから，1 回の積和演算では 12 個必要となる．
(3) $n > k$ における秘匿計算結果を，次の秘匿計算の入力とする場合，各サーバが扱う乱数の断片の位置は固定される．たとえば，上記変換用乱数に関して，$\varepsilon_{h,0}$ を復元したサーバは，演算終了まで $\varepsilon_{h,l}$（$l \neq 0$）を復元しない．

3.4 分散処理のための事前処理

本節では，分散処理で用いる値の中で，秘密情報に依存

せず事前に生成可能な値の生成方法を示す．ここで生成した値は任意の入力に対して利用でき，分散処理での計算量や通信量などの削減を目的とする．以下の 1 回の分散処理に必要な値を生成する手順を示す．以降，$S_0,\ldots,S_{k-1}$ を直接演算に関与する k 台のサーバとし，S_j と表記した場合は $j=0,1,\ldots,k-1$ の k 台が各々処理を実行し．S_i と表記した場合は $i=0,1,\ldots,n-1$ の全 n 台が各々処理を実行することを意味する．

① サーバ S_j は乱数 $A_{1,j},A_{2,j},\alpha_{0,j},\ldots,\alpha_{5,j}$ を生成し，$\alpha_{1,j}$，$\alpha_{4,j}$ をコミットする．また，$A_{1,j},A_{2,j},\alpha_{0,j},\ldots,\alpha_{5,j}$ を n 台のサーバに XOR 法で分散する．ただし，$n=k$ なら XOR 法での分散は不要である．

② サーバ S_j は $\alpha_{0,j}\alpha_{1,j},\alpha_{2,j}\alpha_{1,j},\alpha_{3,j}\alpha_{4,j},\alpha_{5,j}\alpha_{4,j}$ を計算し，n 台のサーバに送信する．

③ 全サーバは以下を計算する．

$$\alpha_0\alpha_1=\prod_{j=0}^{k-1}\alpha_{0,j}\alpha_{1,j},\quad \alpha_2\alpha_1=\prod_{j=0}^{k-1}\alpha_{2,j}\alpha_{1,j},$$
$$\alpha_3\alpha_4=\prod_{j=0}^{k-1}\alpha_{3,j}\alpha_{4,j},\quad \alpha_5\alpha_4=\prod_{j=0}^{k-1}\alpha_{5,j}\alpha_{4,j}$$

④ サーバ S_j は $\varepsilon_{1,j},\varepsilon_{2,j},\ldots,\varepsilon_{8,j}$ に対するシェアを収集し，復元する（サーバ S_j は $\varepsilon_{1,m},\varepsilon_{2,m},\ldots,\varepsilon_{8,m},m\neq j$ を復元しない）．

⑤ サーバ S_j は以下を計算し，n 台のサーバに送信する．

$$\frac{\alpha_{2,j}}{\varepsilon_{1,j}},\frac{\alpha_{2,j}A_{1,j}}{\varepsilon_{2,j}},\frac{\alpha_{5,j}}{\varepsilon_{3,j}},\frac{\alpha_{5,j}A_{2,j}}{\varepsilon_{4,j}},\frac{1}{\alpha_{2,j}\varepsilon_{5,j}},\frac{A_{2,j}}{\varepsilon_{6,j}},$$
$$\frac{1}{\alpha_{5,j}\varepsilon_{7,j}},\frac{A_{1,j}}{\varepsilon_{8,j}}$$

⑥ 全サーバは手順③と同様に，手順⑤で受け取った各 k 個の値の積を計算し，以下を得る．

$$\frac{\alpha_2}{\varepsilon_1},\frac{\alpha_2A_1}{\varepsilon_2},\frac{\alpha_5}{\varepsilon_3},\frac{\alpha_5A_2}{\varepsilon_4},\frac{1}{\alpha_2\varepsilon_5},\frac{A_2}{\varepsilon_6},\frac{1}{\alpha_5\varepsilon_7},\frac{A_1}{\varepsilon_8}$$

⑦ 各サーバ S_i は，手順⑥で得た各値に対して $[\alpha_2]_i=\frac{\alpha_2}{\varepsilon_1}\times[\varepsilon_1]_i$ のように，分母の変換用乱数を打ち消すように対応する変換用乱数のシェアを掛けることにより以下を計算する．

$$[\alpha_2]_i,[\alpha_2A_1]_i,[\alpha_5]_i,[\alpha_5A_2]_i,$$
$$\left[\frac{1}{\alpha_2}\right]_i,[A_2]_i,\left[\frac{1}{\alpha_5}\right]_i,[A_1]_i$$

⑧ 各サーバ S_i は，1 回の分散処理用に以下を保持する．

［$n=k$ の場合］

$$\alpha_0\alpha_1,\alpha_2\alpha_1,\alpha_3\alpha_4,\alpha_5\alpha_4,A_{1,i},A_{2,i}$$
$$\alpha_{0,i},\alpha_{1,i},\alpha_{2,i},\alpha_{3,i},\alpha_{4,i},\alpha_{5,i}$$
$$[\alpha_2]_i,[\alpha_2A_1]_i,[\alpha_5]_i,[\alpha_5A_2]_i,$$
$$\left[\frac{1}{\alpha_2}\right]_i,[A_2]_i,\left[\frac{1}{\alpha_5}\right]_i,[A_1]_i$$

［$n>k$ の場合］

$$\alpha_0\alpha_1,\alpha_2\alpha_1,\alpha_3\alpha_4,\alpha_5\alpha_4,[A_{1,l}]_i^{XOR},[A_{2,l}]_i^{XOR}$$
$$[\alpha_{0,l}]_i^{XOR},[\alpha_{1,l}]_i^{XOR},[\alpha_{2,l}]_i^{XOR},[\alpha_{3,l}]_i^{XOR},$$
$$[\alpha_{4,l}]_i^{XOR},[\alpha_{5,l}]_i^{XOR}$$
$$[\alpha_2]_i,[\alpha_2A_1]_i,[\alpha_5]_i,[\alpha_5A_2]_i,$$
$$\left[\frac{1}{\alpha_2}\right]_i,[A_2]_i,\left[\frac{1}{\alpha_5}\right]_i,[A_1]_i$$

3.5 分散処理

本節では，1 人の入力者が保持する 1 つの秘密情報 a を n 台のサーバに分散する方法を示す．また，以降は $n>k$ としてプロトコルを示す．$n=k$ と $n>k$ の場合との違いは，演算に参加する k 台のサーバが，自身が生成した乱数を XOR 法で分散するか否かであり，$n=k$ の場合は XOR 法による分散はせず，自身が生成した乱数をそのまま保持するのみで十分であるため，コストの観点からメリットはないが $n>k$ の場合のプロトコルは $n=k$ に対しても適用できる．

① 演算に関与する k 台のサーバ S_j は $[A_{1,l}]_j^{XOR}$，$[A_{2,l}]_j^{XOR}$ を入力者に送信する．

② 入力者は $A_{1,l}$，$A_{2,l}$ を復元する．

③ 入力者は以下を計算し，n 台のサーバに送信する．

$$a+A_1=a+\prod_{j=0}^{k-1}A_{1,j},\quad a+A_2=a+\prod_{j=0}^{k-1}A_{2,j}$$

④ サーバ S_j は以下を計算し，n 台のサーバへ送信する．

$$\begin{aligned}&[\alpha_2(a+\alpha_1)]_j\\&=[\alpha_2(a+A_1)+\alpha_2\alpha_1-\alpha_2A_1]_j\\&=[\alpha_2(a+A_1)]_j+\alpha_2\alpha_1-[\alpha_2A_1]_j\\&=(a+A_1)\times[\alpha_2]_j+\alpha_2\alpha_1-[\alpha_2A_1]_j\\&[\alpha_5(a+\alpha_4)]_j\\&=[\alpha_5(a+A_2)+\alpha_5\alpha_4-\alpha_5A_2]_j\\&=[\alpha_5(a+A_2)]_j+\alpha_5\alpha_4-[\alpha_5A_2]_j\\&=(a+A_2)\times[\alpha_5]_j+\alpha_5\alpha_4-[\alpha_5A_2]_j\end{aligned}$$

⑤ 全サーバは，手順④で受け取った各々 k 個のシェアより，$\alpha_2(a+\alpha_1)$，$\alpha_5(a+\alpha_4)$ を復元する．

⑥ 各サーバ S_i は，以下を計算する．

$$\begin{aligned}&[\alpha_1]_i\\&=[(a+\alpha_1)+A_2-(a+A_2)]_i\\&=\left[\frac{1}{\alpha_2}\times\alpha_2(a+\alpha_1)+A_2-(a+A_2)\right]_i\\&=\alpha_2(a+\alpha_1)\times\left[\frac{1}{\alpha_2}\right]_i+[A_2]_i-(a+A_2)\end{aligned}$$

$$
\begin{aligned}
&[\alpha_4]_i \\
&= [(a+\alpha_4) + A_1 - (a+A_1)]_i \\
&= \left[\frac{1}{\alpha_5} \times \alpha_5(a+\alpha_4) + A_1 - (a+A_1)\right]_i \\
&= \alpha_5(a+\alpha_4) \times \left[\frac{1}{\alpha_5}\right]_i + [A_1]_i - (a+A_1)
\end{aligned}
$$

⑦ 各サーバ S_i は，秘密情報 a に対するシェアとして，以下を保持する．

$$
\begin{aligned}
&\alpha_0\alpha_1, \alpha_2(a+\alpha_1), \alpha_3\alpha_4, \alpha_5(a+\alpha_4), [\alpha_1]_i, [\alpha_4]_i \\
&[\alpha_{0,l}]_i^{XOR}, [\alpha_{1,l}]_i^{XOR}, [\alpha_{2,l}]_i^{XOR}, [\alpha_{3,l}]_i^{XOR}, \\
&[\alpha_{4,l}]_i^{XOR}, [\alpha_{5,l}]_i^{XOR}
\end{aligned}
$$

3.6 復元処理

本節では，秘密情報や計算結果の復元方法を示す．処理中では復元結果の正当性を検証し，不正が発覚した時点で処理を強制終了する．ここでは 3.5 節での値を用いて a の復元方法を示すが，秘匿計算の結果を復元する場合も同様な手順で復元が可能である．

① 復元に参加する k 台のサーバ S_j は，以下を復元者に送信する．計算結果を復元する場合は，計算に用いた秘密情報に対応するコミットされた乱数のシェアをすべて送信する．ただし，全サーバが共通に保持している値 $\alpha_2(a+\alpha_1)$，$\alpha_5(a+\alpha_4)$ は，最低 1 台のサーバが送信すればよい．

$$
\begin{aligned}
&\alpha_2(a+\alpha_1), [\alpha_{2,0}]_j^{XOR}, \ldots, [\alpha_{2,k-1}]_j^{XOR}, [\alpha_1]_j \\
&\alpha_5(a+\alpha_4), [\alpha_{5,0}]_j^{XOR}, \ldots, [\alpha_{5,k-1}]_j^{XOR}, [\alpha_4]_j
\end{aligned}
$$

② 復元者はシェアから α_1，α_4 を復元し，$\alpha_{1,j}$，$\alpha_{4,j}$ に対するコミットメントをオープンする．オープンに必要な情報は省略してあるが，手順①でまとめて送信可能である．そして，オープンした $\alpha_{1,j}$，$\alpha_{4,j}$ をそれぞれ k 個掛け合わせて α_1，α_4 を計算し，これら 2 種類の手順で得られる α_1，α_4 が一致するか検証する．

③ 復元者はシェアから $\alpha_{2,j}$，$\alpha_{5,j}$ を復元し，以下を計算する．そして，ここで計算した値と，手順②で得た α_1，α_4 から計算される $(\alpha_1 - \alpha_4)$ を比較して一致するか検証する．

$$
\frac{\alpha_2(a+\alpha_1)}{\prod_{j=0}^{k-1}\alpha_{2,j}} - \frac{\alpha_5(a+\alpha_4)}{\prod_{j=0}^{k-1}\alpha_{5,j}}
$$

④ すべての検証が問題なければ，復元者は以下の計算で復元結果を得る．

$$
a = \frac{\alpha_2(a+\alpha_1)}{\prod_{j=0}^{k-1}\alpha_{2,j}} - \alpha_1 = \frac{\alpha_5(a+\alpha_4)}{\prod_{j=0}^{k-1}\alpha_{5,j}} - \alpha_4
$$

3.7 秘匿積和演算

本節では，演算中において秘密分散法によるシェアから復元される値がすべて 0 ではない場合に関しての秘匿計算手法を示す．演算中に 0 が復元される場合に必要な処理に関しては，3.8 節で議論する．

また，本節では 3 人の入力者がそれぞれ 1 つずつ保持する秘密情報 a，b，c の合計 3 入力から 1 出力 $d = ab + c$ を求める積和演算を考える．ただし，本節で示す処理と同様にして和 $a+b$ および積 $a \times b$ も計算可能である．3 人の入力者による 3 入力 a，b，c に対する，サーバ S_i が保持するシェアを 3.5 節と同様に以下のように定義する．ただし，演算に用いない値は省略してある．また，煩雑な表記を避けるため変換用乱数は再び ε_1 から順に用いるが，3.3 節の (2) のとおり，他の処理で扱ったものとは別のものである．

$$
\begin{aligned}
&\alpha_0\alpha_1, \alpha_2(a+\alpha_1), \alpha_3\alpha_4, \alpha_5(a+\alpha_4), \\
&[\alpha_{0,l}]_i^{XOR}, [\alpha_{2,l}]_i^{XOR}, [\alpha_{3,l}]_i^{XOR}, [\alpha_{5,l}]_i^{XOR} \\
&\beta_0\beta_1, \beta_2(b+\beta_1), \beta_3\beta_4, \beta_5(b+\beta_4), \\
&[\beta_{0,l}]_i^{XOR}, [\beta_{2,l}]_i^{XOR}, [\beta_{3,l}]_i^{XOR}, [\beta_{5,l}]_i^{XOR} \\
&\gamma_0\gamma_1, \gamma_2(c+\gamma_1), \gamma_3\gamma_4, \gamma_5(c+\gamma_4), \\
&[\gamma_{0,l}]_i^{XOR}, [\gamma_{2,l}]_i^{XOR}, [\gamma_{3,l}]_i^{XOR}, [\gamma_{5,l}]_i^{XOR}
\end{aligned}
$$

また，出力 $ab+c$ に対するサーバ S_i が保持するシェアを以下のように定義する．ただし，$d_x = \gamma_x - \alpha_x\beta_x$，$x = 1, 4$ である．

$$
\begin{aligned}
&\delta_0 d_1, \delta_2(d+d_1), \delta_3 d_4, \delta_5(d+d_4), \\
&[\delta_{0,l}]_i^{XOR}, [\delta_{2,l}]_i^{XOR}, [\delta_{3,l}]_i^{XOR}, [\delta_{5,l}]_i^{XOR}
\end{aligned}
$$

① 演算に参加する k 台のサーバ S_j は，$\varepsilon_{1,j}, \varepsilon_{2,j}, \ldots, \varepsilon_{12,j}$ に対するシェアを収集し，復元する．

② サーバ S_j は乱数 $\delta_{0,j}, \delta_{2,j}, \delta_{3,j}, \delta_{5,j}$ を生成して，n 台のサーバに XOR 法で分散する．

③ サーバ S_j は以下を計算し，n 台のサーバに送信する．

$$
\begin{aligned}
&\frac{\delta_{0,j}}{\gamma_{0,j}\varepsilon_{1,j}}, \frac{\delta_{0,j}}{\alpha_{0,j}\beta_{0,j}\varepsilon_{2,j}}, \frac{\delta_{2,j}}{\alpha_{2,j}\beta_{2,j}\varepsilon_{3,j}}, \\
&\frac{\delta_{2,j}}{\alpha_{2,j}\beta_{0,j}\varepsilon_{4,j}}, \frac{\delta_{2,j}}{\alpha_{0,j}\beta_{2,j}\varepsilon_{5,j}}, \frac{\delta_{2,j}}{\gamma_{2,j}\varepsilon_{6,j}} \\
&\frac{\delta_{3,j}}{\gamma_{3,j}\varepsilon_{7,j}}, \frac{\delta_{3,j}}{\alpha_{3,j}\beta_{3,j}\varepsilon_{8,j}}, \frac{\delta_{5,j}}{\alpha_{5,j}\beta_{5,j}\varepsilon_{9,j}}, \\
&\frac{\delta_{5,j}}{\alpha_{5,j}\beta_{3,j}\varepsilon_{10,j}}, \frac{\delta_{5,j}}{\alpha_{3,j}\beta_{5,j}\varepsilon_{11,j}}, \frac{\delta_{5,j}}{\gamma_{5,j}\varepsilon_{12,j}}
\end{aligned}
$$

④ 全サーバは，手順③で受け取った各 k 個の値の積を計算し，以下を得る．

$$
\begin{aligned}
&\frac{\delta_0}{\gamma_0\varepsilon_1}, \frac{\delta_0}{\alpha_0\beta_0\varepsilon_2}, \frac{\delta_2}{\alpha_2\beta_2\varepsilon_3}, \frac{\delta_2}{\alpha_2\beta_0\varepsilon_4}, \frac{\delta_2}{\alpha_0\beta_2\varepsilon_5}, \frac{\delta_2}{\gamma_2\varepsilon_6} \\
&\frac{\delta_3}{\gamma_3\varepsilon_7}, \frac{\delta_3}{\alpha_3\beta_3\varepsilon_8}, \frac{\delta_5}{\alpha_5\beta_5\varepsilon_9}, \frac{\delta_5}{\alpha_5\beta_3\varepsilon_{10}}, \frac{\delta_5}{\alpha_3\beta_5\varepsilon_{11}}, \frac{\delta_5}{\gamma_5\varepsilon_{12}}
\end{aligned}
$$

⑤ サーバ S_j は以下を計算し，$[\delta_0 d_1]_j$，$[\delta_3 d_4]_j$ を n 台の

サーバに送信する．

$$[\delta_0 d_1]_j = [\delta_0(\gamma_1 - \alpha_1\beta_1)]_j$$
$$= \gamma_0\gamma_1 \times \left(\frac{\delta_0}{\gamma_0\varepsilon_1}\right) \times [\varepsilon_1]_j$$
$$- \alpha_0\alpha_1 \times \beta_0\beta_1 \times \left(\frac{\delta_0}{\alpha_0\beta_0\varepsilon_2}\right) \times [\varepsilon_2]_j$$
$$[\delta_3 d_4]_j = [\delta_3(\gamma_4 - \alpha_4\beta_4)]_j$$
$$= \gamma_3\gamma_4 \times \left(\frac{\delta_3}{\gamma_3\varepsilon_7}\right) \times [\varepsilon_7]_j$$
$$- \alpha_3\alpha_4 \times \beta_3\beta_4 \times \left(\frac{\delta_3}{\alpha_3\beta_3\varepsilon_8}\right) \times [\varepsilon_8]_j$$

⑥ 全サーバは，$\delta_0 d_1$，$\delta_3 d_4$ を復元する．

⑦ サーバ S_j は以下を計算し，$[\delta_2(d+d_1)]_j$，$[\delta_5(d+d_4)]_j$ を n 台のサーバに送信する．

$$[\delta_2(d+d_1)]_j = [\delta_2\{(ab+c)+(\gamma_1-\alpha_1\beta_1)\}]_j$$
$$= \alpha_2(a+\alpha_1)\times\beta_2(b+\beta_1)\times\left(\frac{\delta_2}{\alpha_2\beta_2\varepsilon_3}\right)\times[\varepsilon_3]_j$$
$$-\alpha_2(a+\alpha_1)\times\beta_0\beta_1\times\left(\frac{\delta_2}{\alpha_2\beta_0\varepsilon_4}\right)\times[\varepsilon_4]_j$$
$$-\alpha_0\alpha_1\times\beta_2(b+\beta_1)\times\left(\frac{\delta_2}{\alpha_0\beta_2\varepsilon_5}\right)\times[\varepsilon_5]_j$$
$$+\gamma_2(c+\gamma_1)\times\left(\frac{\delta_2}{\gamma_2\varepsilon_6}\right)\times[\varepsilon_6]_j$$
$$[\delta_5(d+d_4)]_j = [\delta_5\{(ab+c)+(\gamma_4-\alpha_4\beta_4)\}]_j$$
$$= \alpha_5(a+\alpha_4)\times\beta_5(b+\beta_4)\times\left(\frac{\delta_5}{\alpha_5\beta_5\varepsilon_9}\right)\times[\varepsilon_9]_j$$
$$-\alpha_5(a+\alpha_4)\times\beta_3\beta_4\times\left(\frac{\delta_5}{\alpha_5\beta_3\varepsilon_{10}}\right)\times[\varepsilon_{10}]_j$$
$$-\alpha_3\alpha_4\times\beta_5(b+\beta_4)\times\left(\frac{\delta_5}{\alpha_3\beta_5\varepsilon_{11}}\right)\times[\varepsilon_{11}]_j$$
$$+\gamma_5(c+\gamma_4)\times\left(\frac{\delta_5}{\gamma_5\varepsilon_{12}}\right)\times[\varepsilon_{12}]_j$$

⑧ 全サーバは，$\delta_2(d+d_1)$，$\delta_5(d+d_4)$ を復元する．

⑨ 各サーバ S_i は，計算結果 $d=ab+c$ に対するシェアとして，以下を保持する．

$$\delta_0 d_1, \delta_2(d+d_1), \delta_3 d_4, \delta_5(d+d_4),$$
$$[\delta_{0,l}]_i^{XOR}, [\delta_{2,l}]_i^{XOR}, [\delta_{3,l}]_i^{XOR}, [\delta_{5,l}]_i^{XOR}$$

3.8 補正処理

本節では，演算中において秘密分散法によるシェアから 0 が復元された場合に関して必要な処理を示す．たとえば，3.7 節手順⑥で $\delta_0 d_1 = 0$ となった場合を考える．δ_0 は 0 でない k 個の乱数 $\delta_{0,0},\ldots,\delta_{0,k-1}$ の積として定義され，提案方式は素数を法とする有限体上での演算を仮定しているため，$\delta_0 \neq 0$ がいえる．よって，$d_1 = (\gamma_1 - \alpha_1\beta_1) = 0$ が漏洩したことになる．α_1，β_1，γ_1 は復元処理の検証で用いられ，復元処理より前に漏洩してはならない．$(\gamma_1 - \alpha_1\beta_1) = 0$ のみでは個々の乱数は漏洩しないが，たとえば後の演算で $(c-a)$ を実行する場合に $(\gamma_1 - \alpha_1) = 0$ が漏洩する可能性があり，これらを組み合わせると $\beta_1 = 1$ が漏洩する．よって，このような問題を防ぐため，以下の対策を講じる．

(1) 0 が復元された場合，その情報を全サーバが保持する．

(2) シェアの計算前に，演算に参加するサーバはその時点で漏洩している情報を参照する．次に計算しようとしているシェアからの復元結果が 0 となった場合に個々の乱数や秘密情報，計算結果が漏洩するなら，計算式を変える．

計算式を変えるとは，具体的には新たな乱数 w_1 を加算することにより d_1 を更新し，$d_1' = d_1 + w_1$ とする．このようにすることで，もし $d_1' = d_1 + w_1 = 0$ となっても，w_1 はこの手順で新しく生成された乱数であり，それまでに漏洩したいかなる情報と組み合わせても，この w_1 は打ち消せない．また，$d_1' = 0$ となった場合は，$d_1 = d_1' - w_1 \neq 0$ のようにして本来の d_1 を比較的簡単に計算できる．ただし，$d_1' \neq 0$ の場合に攻撃者の独断で $d_1 = d_1' - w_1$ を計算できないようにしなければ本処理は無意味なため，この点に注意してプロトコルを構築する．上記 (2) に記載の計算式を変えるか否かを判断するタイミングは，最も遅くてシェアの計算の直前である 3.7 節の手順⑤および手順⑦である．判断するタイミングを早めることで，不要な計算を削減できる．まず，3.7 節の手順⑤で $[\delta_0 d_1]_i$ の計算式を変える必要がある場合に関しての処理を示す．また，3.7 節と変更がない処理に関しては省略する．

[3.7 節の手順⑤で $[\delta_0 d_1]_i$ に対して補正処理が実行される場合]

各サーバは，3.7 節手順④までの処理が済んでいるとする．

① サーバ S_j は $\varepsilon_{13,j}$ に対するシェアを収集し，復元する．

② サーバ S_j は乱数 $w_{1,j}$ を生成し，n 台のサーバへ XOR 法で分散する．また，$w_{1,j}$ をコミットする．

③ サーバ S_j は $\delta_{0,j}w_{1,j}/\varepsilon_{13,j}$ を計算し，n 台のサーバに送信する．

④ 全サーバは，手順③で受け取った各 k 個の値の積を計算し，$\delta_0 w_1/\varepsilon_{13}$ を得る．

⑤ サーバ S_j は以下を計算し，$[\delta_0 d_1']_j$ を n 台のサーバに送信する．

$$[\delta_0 d_1']_j = [\delta_0(w_1 + \gamma_1 - \alpha_1\beta_1)]_j$$
$$= \frac{\delta_0 w_1}{\varepsilon_{13}} \times [\varepsilon_{13}]_i + \gamma_0\gamma_1 \times \left(\frac{\delta_0}{\gamma_0\varepsilon_1}\right) \times [\varepsilon_1]_j$$
$$- \alpha_0\alpha_1 \times \beta_0\beta_1 \times \left(\frac{\delta_0}{\alpha_0\beta_0\varepsilon_2}\right) \times [\varepsilon_2]_j$$
$$+ \frac{\delta_0 w_1}{\varepsilon_{13}} \times [\varepsilon_{13}]_i$$

⑥ 全サーバは，$\delta_0 d_1'$ を復元する．復元結果が 0 の場合は，以下を計算して n 台のサーバに送信し，全

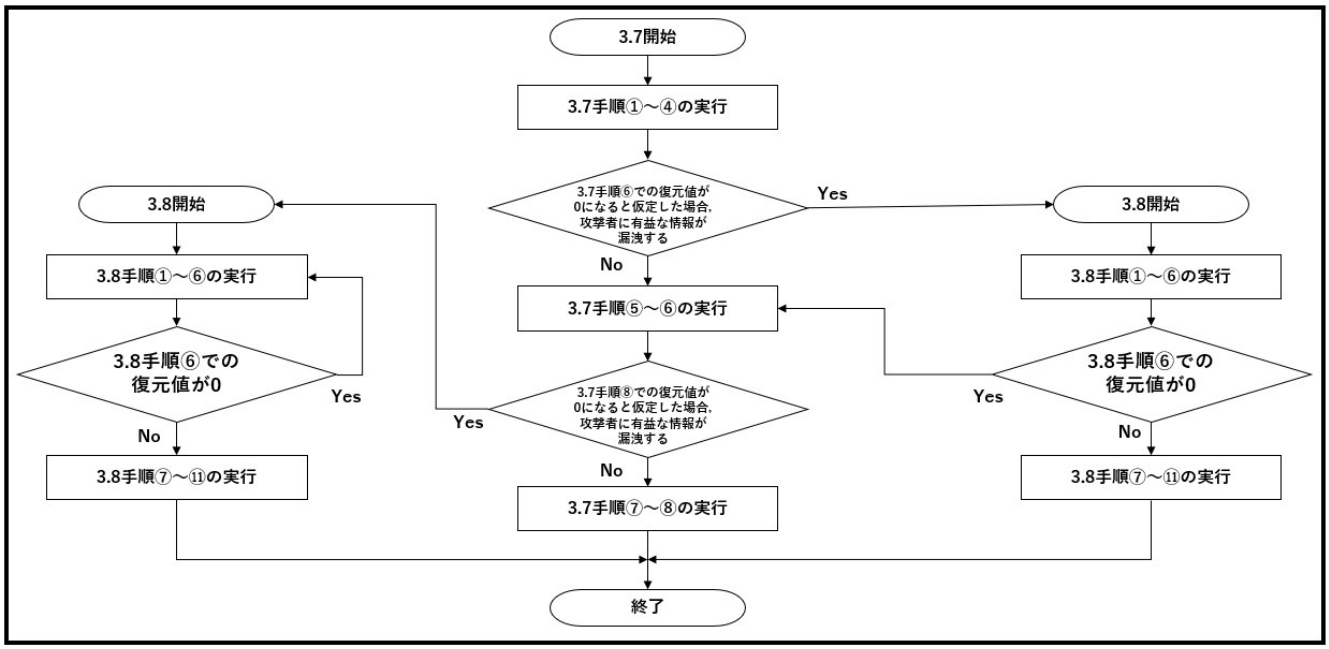


図 1 積和演算のフローチャート
Fig. 1 Flowchart of product-sum protocol.

サーバは，$\delta_0 d_1 \neq 0$ を復元して保持する．ここでの $\delta_0 w_1/\varepsilon_{13} \times [\varepsilon_{13}]_i$ は，手順⑤と同じでよい．

$$[\delta_0 d_1] = \delta_0 d_1' - \frac{\delta_0 w_1}{\varepsilon_{13}} \times [\varepsilon_{13}]_i$$

⑦ $\delta_0 d_1 \neq 0$ を保持した場合は 3.7 節と同じであるため，3.7 節手順⑦から行う．$\delta_0 d_1' \neq 0$ を保持した場合，サーバ S_j は $\varepsilon_{14,j}$ に対するシェアを収集し，復元する．

⑧ サーバ S_j は $\delta_{2,j} w_{1,j}/\varepsilon_{14,j}$ を計算し，n 台のサーバに送信する．

⑨ 全サーバは，手順⑧で受け取った各 k 個の値の積を計算し，$\delta_2 w_1/\varepsilon_{14}$ を得る．

⑩ サーバ S_j は以下を計算し，$[\delta_2(d+d_1')]_j$ を n 台のサーバに送信する．

$$\begin{aligned}[\delta_2(d+d_1')]_j &= [\delta_2\{(ab+c)+(w_1+\gamma_1-\alpha_1\beta_1)\}]_j \\ &= \frac{\delta_2 w_1}{\varepsilon_{14}} \times [\varepsilon_{14}]_j + \alpha_2(a+\alpha_1) \times \beta_2(b+\beta_1) \\ &\quad \times \left(\frac{\delta_2}{\alpha_2\beta_2\varepsilon_3}\right) \times [\varepsilon_3]_j \\ &\quad - \alpha_2(a+\alpha_1) \times \beta_0\beta_1 \times \left(\frac{\delta_2}{\alpha_2\beta_0\varepsilon_4}\right) \times [\varepsilon_4]_j \\ &\quad - \alpha_0\alpha_1 \times \beta_2(b+\beta_1) \times \left(\frac{\delta_2}{\alpha_0\beta_2\varepsilon_5}\right) \times [\varepsilon_5]_j \\ &\quad + \gamma_2(c+\gamma_1) \times \left(\frac{\delta_2}{\gamma_2\varepsilon_6}\right) \times [\varepsilon_6]_j\end{aligned}$$

⑪ 全サーバは，$\delta_2\,(d+d_1')$ を復元し，保持する．

⑫ サーバ S_i は，以下を保持する（3.7 節との変更分および追加分のみ示す）．

$$\delta_0 d_1', \delta_2(d+d_1'), [w_{1,l}]_i^{XOR}$$

これ以降は，復元処理におけるラウンド数削減のために必要な処理を示す．この処理は復元処理まで後回しにしても安全性に影響はないが，これ以降任意のタイミングで他の処理と並列で実行できるため，早めに計算しておくことが望ましい．上記補正処理で生成された w_1 は α_1，β_1，γ_1 と同様にコミットされ，復元処理中の検証でオープンされる．そのため，$[\alpha_1]_i$ などと同様に各サーバ S_i は $[w_1]_i$ も保持する必要がある．以下に $[w_1]_i$ の計算方法を示す．

① サーバ S_j は $\varepsilon_{15,j}$ に対するシェアを収集し，復元する．

② サーバ S_j は $w_{1,j}/\varepsilon_{15,j}$ を計算し，n 台のサーバに送信する．

③ 全サーバは，手順②で受け取った各 k 個の値の積を計算し，w_1/ε_{15} を得る．

④ サーバ S_i は以下を計算し，保持する．

$$[w_1]_i = \frac{w_1}{\varepsilon_{15}} \times [\varepsilon_{15}]_i$$

次に，3.7 節の手順⑤で $[\delta_0 d_1]_i$ に対して補正処理を実行する必要がなく，3.7 節の手順⑦で変える必要が発生した場合に関しての処理を考える．この場合は，補正処理の手順①から再び行うのみである．ただし，この場合 $\delta_0 d_1 \neq 0$ を保持しても無意味なため，$\delta_0 d_1' \neq 0$ となるまで乱数 w_1 を変えながら計算を繰り返す．

以上のように，3.8 節を考慮すると提案方式の秘匿積和演算は分岐が多く複雑になってしまうため，**図 1** に積和演算の開始から終了までをフローチャート形式で示す．

4. 安全性

本章では，提案方式の安全性を示す．また，シャノンエ

ントロピーを H とする．前述のとおり，コミットメントスキームは 2.4 節で議論した (2) のタイプである，無条件な秘匿性かつ計算量的な束縛性を持つタイプを用いた場合に関して安全性を議論する．この秘匿性に関して，以降は無条件と表記するがそれが完全か統計的かは採用するコミットメントスキームに依存する．

4.1 提案方式における安全性要件

一般に，情報セキュリティには機密性，完全性，可用性という 3 要素がある．本章において，それぞれを以下のように設定する．

[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：偶然すべての検証が通過する場合を除き，honest な復元者は入力者が実際に分散した値，またはそれらから計算される値を復元処理で得る．
(3) 可用性：$n \geq k$ の設定で利用できる．

4.2 攻撃者設定

本章で設定する攻撃者を以下に示す．

(1) 変換用乱数組 $[\varepsilon_h]_i$，$[\varepsilon_{h,j}]_i^{XOR}$ から定義される ε_h を直接知らない（サーバを乗っ取るなどによって不正に知ろうとする）．
(2) t 入力 1 出力の演算に関して，入力者および復元者の攻撃者との結託は最大 $t-1$ 人である（t 人が結託する場合，残りの 1 値が漏洩することは自明である）．
(3) 最大 $k-1$ 台のサーバを corrupt でき，それらが持つ情報を知れる．ただし，元々攻撃者の管理下にあるサーバは corrupt されていると考える．
(4) 攻撃者は corrupt したサーバをプロトコルから逸脱させることが可能である．
(5) 機密性に対しては計算能力が無限な攻撃者を仮定し，完全性に関しては計算能力が有限な攻撃者を仮定する．

4.3 機密性

提案方式では 1 つの秘密情報に対して同様な値を 2 つペアで生成するが，それらは独立な乱数を用いて生成されるため，基本的に一方のみに着目して安全性を議論する．また，以降は攻撃者が corrupt する $k-1$ 台のサーバを $S_0, \ldots, S_{k-2}$ と定め，honest なサーバ S_{k-1} がローカルに保持する情報が攻撃者に対して直接漏洩することはないとする．機密性の議論においては，攻撃者の計算能力が無限であると仮定し，攻撃者が corrupt する $S_0, \ldots, S_{k-2}$ がローカルに保持する値や演算中に全サーバへ送信される値などを組み合わせて，honest なサーバ S_{k-1} がローカルに保持する情報や honest な入力者の秘密情報を知ろうとする．4.1 節で示したように，honest な入力者の秘密情報が攻撃者に漏洩しないことを証明する．また，以下の定理 1 より，機密性の証明においては，基本的に $n = k$ のプロトコルに対して議論を行う．

・定理 1：機密性において，$n = k$ のプロトコルと $n \geq k$ のプロトコルの安全性に関して，$n \geq k$ の場合のプロトコルで採用する XOR 法が情報理論的安全性を有するなら，両者の安全性は同等である．

・定理 1 の証明：$n \geq k$ のプロトコルは，$n = k$ のプロトコルにおいて k 台のサーバが各々生成してローカルに保持する乱数を n 台のサーバに XOR 法で分散することで実現し，他の変更点はいっさいない．つまり，演算で発生する相違点は，演算に参加する各サーバが扱う乱数が，そのまま保持されているかシェアとして保持されているかのみである．よって，この XOR 法が情報理論的安全ならば，XOR 法で分散および復元することによる乱数の漏洩はなく，$n \geq k$ のプロトコルの安全性が $n = k$ のプロトコルより劣ることはないため，これらは同等の安全性を有する．以上の議論より，定理 1 が証明された．

次に，提案方式では，3.4 節手順②や 3.7 節手順③などの各サーバが保持する乱数を掛け合わせた値を n 台のサーバへ送信する処理や，3.4 節手順③や 3.7 節手順④などのそれらを k 個掛け合わせる処理が多く存在する．これらに対して，定理 2 がいえる．

・定理 2：演算中に全サーバが受け取る 2 つ以上の乱数の積および商で構成された値を組み合わせても，個々の乱数を得ることはできない．さらに，これらのような値を構成する乱数の一部は入力者や復元者が攻撃者と結託した場合に漏洩するが，変換用乱数を打ち消せず残りの乱数は漏洩しない．また，入力者や復元者との結託によって漏洩する乱数は，直接的な機密性への影響はない．

・定理 2 の証明：まず，変換用乱数を含まない 3.4 節手順②および 3.4 節手順③で全サーバが得る値を考える．これらの手順で全サーバが得る値は，$\alpha_{0,j}\alpha_{1,j}$，$\alpha_{2,j}\alpha_{1,j}$，$\alpha_{3,j}\alpha_{4,j}$，$\alpha_{5,j}\alpha_{4,j}$，$\alpha_0\alpha_1$，$\alpha_2\alpha_1$，$\alpha_3\alpha_4$，$\alpha_5\alpha_4$ である．また，$\alpha_{1,j}$，$\alpha_{4,j}$ はコミットされるが，コミットメントスキームの無条件な秘匿性により，攻撃者がコミットメントスキームの秘匿性を破ることはできないため，honest なサーバ S_{k-1} がコミットする $\alpha_{1,k-1}$，$\alpha_{4,k-1}$ はコミットメントから漏洩しない．また，攻撃者は honest なサーバがローカルに保持する $k-1$ 番目の乱数どうしの積 $\alpha_{0,k-1}\alpha_{1,k-1}$，$\alpha_{2,k-1}\alpha_{1,k-1}$ を知るが，$\alpha_{0,k-1}$，$\alpha_{1,k-1}$，$\alpha_{2,k-1}$ のうち最低 1 値を知らなければ他の 2 値は得られない．これは，$\alpha_{3,k-1}$，$\alpha_{4,k-1}$，$\alpha_{5,k-1}$ に対しても同様である．よって，攻撃者は変換用乱数を含まない乱数の積を，個々の乱数に分解することはできない．次に，変換用乱数を含む 3.4 節手順⑤や 3.4 節手順⑥などで全サーバが得る値を考える．この処理は 3.7 節秘匿積和演算などでも新しいシェアの計算の前に実行される．これらの処理に関しては，4.2 節の (1) により定理 2

が証明できる．4.2 節の (1) により攻撃者は honest なサーバが保持する $k-1$ 番目の変換用乱数を知らない．よって，前述の変換用乱数を含まない乱数の積に関する議論と同様に，攻撃者は $\alpha_{2,j}/\varepsilon_{1,j}$，$\alpha_{2,j}A_{1,j}/\varepsilon_{2,j}$ などの乱数どうしの積を構成する個々の乱数を 1 つも知らないため，これらを個々の乱数に分解することはできない．また，たとえば a の入力者が攻撃者と結託した場合，攻撃者は A_1，$A_{1,j}$ を知る．このとき，$\alpha_2 A_1/\varepsilon_2$ に関して A_1 が攻撃者に漏洩しているが，変換用乱数 $\varepsilon_{2,j}$ を打ち消せないため α_2 は漏洩しない．そして，A_1 の漏洩は a の入力者が攻撃者と結託した場合を想定しており，A_1 は a に関する計算にのみ用いられるため，honest な入力者が保持する秘密情報を知る有益な情報とはならない．また，たとえば 3.7 節の d の復元者が攻撃者と結託した場合，3.6 節の手順①で δ_2，$\delta_{2,j}$ などが攻撃者に漏洩する．このとき，3.7 節手順③の $\delta_2/\alpha_2\beta_2\varepsilon_3$ に関して δ_2 が攻撃者に漏洩しているが，変換用乱数 ε_3 を打ち消せないため $\alpha_2\beta_2$ は漏洩しない．そして，δ_2 の漏洩は $d=ab+c$ の復元者が攻撃者と結託した場合を想定しており，δ_2 は秘密情報の秘匿には用いられないため，honest な入力者が保持する秘密情報を知る有益な情報とはならない．以上の議論より，定理 2 が証明された．

4.3.1 3.4 節の分散処理のための事前処理に対する機密性

本処理では秘密情報が存在しないため，honest なサーバのみがローカルに保持するいかなる乱数も漏洩しないことを示す．手順①では通信が発生する処理は $n=k$ の場合コミット処理のみであり，コミットメントスキームの無条件な秘匿性によりいっさいの情報漏洩はない．手順②から手順⑥は定理 2 よりいかなる乱数の漏洩もない．手順⑦はローカルの計算である．よって，本処理では honest なサーバのみがローカルに保持する乱数は漏洩しない．

4.3.2 3.5 節の分散処理に対する機密性

この場合，入力者の秘密情報 a に対する安全性を証明するため，入力者は honest と考える．手順①および②で入力者のみが $A_{1,j}$，$A_{2,j}$ を得るため，攻撃者は $A_{1,k-1}$，$A_{2,k-1}$ を得られず，A_1，A_2 も得られない．よって，手順③で知る値から A_1，A_2 を取り除けないため，ここでは a を求めることはできない．手順④から⑥はローカルに保持する値を用いた新たなシェアの計算とその復元であり，いかなる乱数も漏洩していないことから，復元される $\alpha_2(a+\alpha_1)$ などから a を求めることはできない．よって，本処理では入力者の秘密情報は攻撃者に漏洩せず，エントロピー H を用いて以下がいえる．

$$H(a)=H\,(a \mid \text{攻撃者が 3.4 節，3.5 節で知るすべての値})$$

4.3.3 3.6 節の復元処理に関して，秘密情報の復元対する機密性

この場合，入力者も復元者も honest と考える．手順①で，復元処理に必要なすべての値が復元者に対して送信される．後の処理は復元者のローカルな処理であるため，攻撃者は関与できない．よって，本処理では入力者の秘密情報は攻撃者に漏洩せず，エントロピー H を用いて以下がいえる．

$$H(a)=H\,(a \mid \text{攻撃者が 3.4 節，3.5 節，3.6 節で知るすべての値})$$

4.3.4 3.7 節の秘匿積和演算に対する機密性

本処理では，最大 2 人の入力者が攻撃者と結託し，残りの 1 人の入力者の秘密情報を知ろうとする．まず a の入力者が攻撃者と結託した場合を考える．このとき，攻撃者は a，A_1，A_2，$A_{1,k-1}$，$A_{2,k-1}$ を新たに知る．まず a を知ることに関して，攻撃者が知る直接 a に関する情報は，$(a+A_1)$，$(a+A_2)$，$\alpha_2(a+\alpha_1)$，$\alpha_5(a+\alpha_4)$ のみであり，A_1，A_2 を除く各乱数は 1 つも漏洩しない．次に A_1，A_2 を知ることに関して，3.4 節手順⑤で $\alpha_{2,k-1}A_{1,k-1}/\varepsilon_{2,k-1}$ などを知るが，定理 2 により攻撃者は $\alpha_{2,k-1}$ などを知ることができない．$A_{2,k-1}/\varepsilon_{6,k-1}$ から $\varepsilon_{6,k-1}$ を知ることなど，特定の変換用乱数は知ることが可能であるが，3.3 節の (2) より同じ変換用乱数は再度用いられないため，これは安全性に影響はない．また，定理 2 より手順①から手順④で honest なサーバのみがローカルに保持するいかなる乱数も漏洩しないことがいえ，つまり新たに生成された δ_0，δ_2，δ_3，δ_5 も漏洩しない．よって，手順⑥と⑧で復元される値から δ_0，δ_2，δ_3，δ_5 を取り除くことはできず，本処理では入力者の秘密情報は攻撃者に漏洩しない．これは，2 人の入力者が攻撃者と結託した場合でも同様にいえ，エントロピー H を用いて以下がいえる．ただし，x は honest な入力者の入力とする．

$$H(x)=H\,(x \mid \text{攻撃者が 3.4 節，3.5 節，3.7 節で知るすべての値})$$

4.3.5 3.8 節の補正処理に対する機密性

3.8 節は 3.7 節の手順⑥と⑧において 0 が復元された場合に実行される．対策 (1) および (2) より，0 が復元されると情報漏洩が発生する場合は計算式が変更され，万が一の情報漏洩が回避される．よって，3.7 節の実行で 0 が復元されること自体は問題にならず，3.8 節に対して安全性の議論を行えばよい．手順①と手順②は XOR 法の処理とコミット処理であり，すでに安全性は議論済みである．手順③と手順④は定理 2 により個々の乱数はいっさい漏洩しない．手順⑤はローカルな計算である．手順⑥に関して，まず $\delta_0 d_1'=0$ が復元された場合を考える．このとき，$\delta_0\neq 0$ より $d_1'=0$ が漏洩するが，本処理で新たに生成された乱数 w_1 は攻撃者に漏洩していないため，d_1' から w_1 を取り除けず新たに有益な情報は得られない．また，この場合は秘匿計算により $\delta_0 d_1=\delta_0 d_1'-\delta_0 w_1$ を計算する．$\delta_0 w_1\neq 0$

より必ず $\delta_0 d_1 \neq 0$ であるため，補正処理はここで中断され 3.7 節に示すとおりに処理を続行する．よって，3.7 節と同様に安全性を証明できる．次に $\delta_0 d'_1 \neq 0$ が復元された場合を考える．この場合は $\delta_2(d+d_1)$ の代わりに $\delta_2(d+d'_1)$ を計算する必要があり，補正処理が続行される．手順⑦から⑨は定理 2 より個々の乱数はいっさい漏洩しない．手順⑩はローカルな計算である．手順⑪では 0 が復元されなければいっさい問題はない．0 が復元された場合，$(d+d'_1)=0$ が漏洩する．しかし，$(d+d'_1)$ に含まれる w_1 はこの補正処理で初めて生成され，この手順より前に w_1 は攻撃者に漏洩していないため，$(d+d'_1)$ から w_1 を取り除けず新たに有益な情報は得られない．次に示されるラウンド数削減のための処理は，定理 2 により個々の乱数はいっさい漏洩しないといえる．よって，エントロピー H を用いて以下がいえる．

$$H(x) = H\,(x \mid \text{攻撃者が 3.4 節，3.5 節，3.7 節，3.8 節で知るすべての値})$$

ただし，3.8 節の (1) が起こる確率は $1/p$ であるため，十分大きな p を選択すれば 3.8 節の実行はめったに起こらないといえる．

4.3.6 3.6 節の復元処理に関して，計算結果の復元に対する機密性

この場合，入力者と復元者の合計 $t+1$ 人の XOR うち，最低 2 人は honest と考える．まず 1 入力者と復元者が honest である場合に関して，手順①では，復元処理に必要なすべての値が復元者に対して送信される．後の処理は復元者のローカルな処理であるため，攻撃者は関与できない．よって，復元者が honest なら，復元処理において入力者の秘密情報は攻撃者に漏洩しない．次に 2 入力者が honest である場合，復元者攻撃者と結託しうるため，手順①で受け取ったすべての値と復元結果を攻撃者と共有できる．ここで，t 入力の場合に復元者が手順①で得る値を以下のように定義する．ただし，f は t 入力 $a_1,\ldots,a_t$ からの計算結果，$f_{2,j}$，$f_{5,j}$，f_2，f_5 は計算結果に対する乱数，$v_{1,1},\ldots,v_{1,t}$ は f_1 を構成する乱数，$v_{4,1},\ldots,v_{4,t}$ は f_4 を構成する乱数とし，定理 1 より $n=k$ の場合を示す．

$$f_2(f+f_1), f_{2,j}, ([v_{1,1}]_j,\ldots,[v_{1,t}]_j)$$
$$f_5(f+f_4), f_{5,j}, ([v_{4,1}]_j,\ldots,[v_{4,t}]_j)$$

以上のように，計算結果を復元する場合は，f_1，f_4 を構成するすべての乱数に対するシェアを収集する必要があり，その他の値は秘密情報の復元の場合と同等なものを同じ数だけ収集するのみである（たとえば，$f_2(f+f_1)$ は $\alpha_2(a+\alpha_1)$ に対応し，$f_{2,j}$ は $\alpha_{2,j}$ に対応する）．また，f_2，f_5 は計算結果を乗算で秘匿する乱数であり，定理 2 より f_2，f_5 が漏洩することによる別の乱数の漏洩はない．$v_{1,1},\ldots,v_{1,t},v_{4,1},\ldots,v_{4,t}$ が漏洩することに関して，以下のようにこれらの乱数は入力の秘匿に利用され，以下の値は全サーバが保持している．

$$\{v_{0,1}v_{1,1}, v_{2,1}(a_1+v_{1,1}), v_{3,1}v_{4,1}, v_{5,1}(a_1+v_{4,1})\}$$
$$\cdots$$
$$\{v_{0,t}v_{1,t}, v_{2,t}(a_t+v_{1,t}), v_{3,t}v_{4,t}, v_{5,t}(a_t+v_{4,t})\}$$

しかし，$v_{1,1},\ldots,v_{1,t},v_{4,1},\ldots,v_{4,t}$ のみでなく $v_{2,1},\ldots,v_{2,t},v_{5,1},\ldots,v_{5,t}$ を知らなければいかなる入力も得られず，$v_{2,1},\ldots,v_{2,t},v_{5,1},\ldots,v_{5,t}$ が漏洩しないことは定理 2 よりいえる．よって，入力者と復元者の合計 $t+1$ 人のうち，どのような組合せの結託においても，honest な入力者の秘密情報は攻撃者に漏洩せず，エントロピー H を用いて以下がいえる．

$$H(x) = H\,(x \mid \text{攻撃者が 3.4 節，3.5 節，3.6 節，3.7 節，3.8 節で知るすべての値})$$

4.4 完全性

以降，提案方式の完全性に関して議論を行う．また，以下の定理 3 より，完全性の証明においても，$n=k$ のプロトコルに対してのみ議論を行う．

・定理 3：完全性において，$n=k$ のプロトコルと $n \geq k$ のプロトコルの安全性に関して，$n \geq k$ の場合のプロトコルで採用する XOR 法が情報理論的安全性を有するなら，両者の安全性は同等である．

・定理 3 の証明：まず，XOR 法が情報理論的安全性を有するなら，XOR 法による情報漏洩は考えなくてよい．次に，$n \geq k$ のプロトコルで XOR 法により分散される乱数に対して，どのような攻撃が可能か考える．攻撃者が可能な攻撃は 2 つのみであり，1 つ目は乗っ取ったサーバが XOR 法により生成したシェアを偽の値に改ざんして送信することであり，2 つ目は XOR 法によるシェアを他のサーバへ送信する際に偽の値に改ざんして送信することである．よって，たとえば 3.4 節の手順①で生成される $\alpha_{0,0}$ に関して，サーバ S_i が受けとるその XOR 法によるシェア $[\alpha_{0,0}]_i^{XOR}$ に対して攻撃者が上記 2 つの攻撃を行った場合，演算において $\alpha_{0,0}$ を扱うサーバ（S_0 とする）は XOR 法によって偽の値 $\alpha'_{0,0} \neq \alpha_{0,0}$ を復元し，演算に使用することになる．たとえば，$n=k$ のプロトコルにおいて S_0 が攻撃者に乗っ取られているなら，S_0 が生成して保持している $\alpha_{0,0}$ を $\alpha'_{0,0}$ に改ざんして演算に使用することが可能であり，$n \geq k$ のプロトコルの場合と違いはない．しかし，$n=k$ のプロトコルにおいて S_{k-1} が honest なら S_{k-1} は自身が生成して保持している正しい $\alpha_{0,k-1}$ を演算に使用するが，$n \geq k$ のプロトコルの場合は honest な S_{k-1} は XOR 法によって偽の値 $\alpha'_{0,k-1}$ を復元する可能性があるため，これが完全性における違いとなる．しかし，これの違いは問題とならな

い．XOR 法により分散および復元される乱数は，3.7 節の手順③のようにローカルで掛け合わされて（Δ_j とする）全サーバへ送信され，その後さらに k 個を掛け合わせて各計算に利用される（$\Delta = \prod_{j=0}^{k-1} \Delta_j$ とする）．よって，完全性において重要なことはシェアの計算に利用される Δ が改ざんされた値であるかどうかであり，k 個の Δ_j のうち 1 つでも改ざんされていれば，Δ も改ざんされていることになる．つまり，k 個の Δ_j が 1 つ以上 k 個以下のうち何個改ざんされていても完全性の議論に変化はない．よって，honest な S_0 が偽の値を演算に使用することは $n \geq k$ のプロトコルでのみ発生するが，$n = k$ のプロトコルと $n \geq k$ のプロトコルに対する完全性の証明は同様に行える．以上の議論より，定理 3 が証明された．

4.4.1 秘密情報の復元に対する完全性

完全性に関しては，攻撃者の計算能力は有限であると仮定する．よって，コミットメントスキームの計算量的束縛性を破れないとする．3.6 節復元処理において，honest な復元者に対して偽の値が復元されないことを示す．ここでは秘密情報 a の復元を考える．3.6 節手順①および②で，復元者は以下の値を得る．ここで，「′」付きの値は正当性が信頼できない値とする．α_1，α_4 はコミットメントスキームを用いて計算した値であり，α'_1，α'_4 は受け取ったシェアから復元した値である．

$$\alpha_2(a+\alpha_1)', \alpha_5(a+\alpha_4)', \alpha'_2, \alpha'_5, \alpha_1, \alpha_4, \alpha'_1, \alpha'_4$$

ここで，3.5 節手順④の計算式より，$\alpha_2(a+a_1)'$，$\alpha_5(a+a_2)'$ は以下のように表せる．ただし，$p_1, \ldots, p_6$ は，3.4 節手順④の各項における正しい値との比であり，3.4 節において攻撃者がブロードキャストする乱数比の断片を定数倍することにより，攻撃者が任意に調整できる．

$$\alpha_2(a+\alpha_1)' = \alpha_2(p_1 a + p_2\alpha_1 + (p_1 - p_3)A_1)$$
$$\alpha_5(a+\alpha_4)' = \alpha_5(p_4 a + p_5\alpha_4 + (p_4 - p_6)A_2)$$

次に，攻撃者は 3.4 節において，前述の $p_1, \ldots, p_6$ と同様にして，3.5 節手順⑥で計算される α'_1，α'_4 を以下のように調整できる．ただし，3.5 節手順⑥の各項に対して正しい値との比を q_1，q_2，q_3，q_4 とする．また，$(a + A_1)$，$(a + A_2)$ は全サーバが事前に共通の値を受け取っており，この項への不正は無意味である．

$$\alpha'_1 = (p_1q_1 - 1)a + p_2q_1\alpha_1 + (p_1 - p_3)q_1A_1 + (q_2 - 1)A_2$$
$$\alpha'_4 = (p_4q_3 - 1)a + p_5q_3\alpha_4 + (p_4 - p_6)q_3A_2 + (q_4 - 1)A_1$$

以上より，3.6 節手順②では，以下が成り立つかを検証する．

$$\alpha'_1 - \alpha_1 = (p_1q_1 - 1)a + (p_2q_1 - 1)\alpha_1 + (p_1 - p_3)q_1A_1 + (q_2 - 1)A_2 = 0$$

$$\alpha'_4 - \alpha_4 = (p_4q_3 - 1)a + (p_5q_3 - 1)\alpha_4 + (p_4 - p_6)q_3A_2 + (q_4 - 1)A_1 = 0$$

これ以降の処理に関して，4.4.1.1 で秘密情報の入力者が攻撃者ではない場合，4.4.1.2 で秘密情報の入力者が攻撃者である場合に関して議論する．

4.4.1.1 入力者が攻撃者と結託しない場合

a の入力者が攻撃者ではない場合は，攻撃者は a，α_1，α_4，A_1，A_2 を知らないため，3.6 節手順②の検証式を a，α_1，α_4，A_1，A_2 に対する恒等式と見ることができる．つまり，攻撃者が意図的に不正な値で検証を通過させるには，すべての項を 0 に調整するほかない．この場合，検証を通過するために以下の条件が要求される．

$$p_1q_1 = 1, \quad p_2q_1 = 1, \quad p_1 = p_3, \quad q_2 = 1$$
$$p_4q_3 = 1, \quad p_5q_3 = 1, \quad p_4 = p_6, \quad q_4 = 1$$

これらを整理すると，以下のようになる．

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

この場合，$\alpha_2(a+\alpha_1)'$ および $\alpha_5(a+\alpha_4)'$ は以下のように整理できる．

$$\alpha_2(a+\alpha_1)' = \alpha_2p_1(a+\alpha_1)$$
$$\alpha_5(a+\alpha_4)' = \alpha_5p_4(a+\alpha_4)$$

よって，3.6 節手順②の時点では，攻撃者の不正分である p_1，p_4 は検出できない．この条件で，3.6 節手順③の検証を考えると，この手順では以下の式が成り立つかを検証することを意味する．ただし，正しい値との比 t_1，t_2 を用いて $\alpha'_2 = t_1\alpha_2$，$\alpha'_5 = t_2\alpha_5$ とする．この t_1，t_2 も，攻撃者が乗っ取るサーバ S_0 が $\alpha_{1,0}$ ではなく $t_1\alpha_{1,0}$ を送信するなどして，攻撃者が任意に調整できる．

$$\left\{\frac{\alpha_2p_1(a+\alpha_1)}{t_1\alpha_2} - \frac{\alpha_5p_4(a+\alpha_4)}{t_2\alpha_5}\right\} - (\alpha_1 - \alpha_4) = 0$$

上の式を整理すると，以下のようになる．

$$\left(\frac{p_1}{t_1} - \frac{p_4}{t_2}\right)a + \left(\frac{p_1}{t_1} - 1\right)\alpha_1 - \left(\frac{p_4}{t_2} - 1\right)\alpha_4 = 0$$

ここで，前述のとおり攻撃者は a，α_1，α_4 を知らないため，上記式を攻撃者が意図的に成立させるためには，各項が 0 となるように調整するしかない．この場合以下が要求される．

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

よって，3.6 節手順③の時点でも，攻撃者の不正分である p_1，p_4，t_1，t_2 は検出できない．最後に，3.5 節手順⑧で復元される値は，以下のように表される．

$$\frac{\alpha_2 p_1(a+\alpha_1)}{t_1\alpha_2}-\alpha_1=\frac{p_1}{t_1}a+\left(\frac{p_1}{t_1}-1\right)\alpha_1=a$$

上式のように，復元結果は正しい値となる．つまり，入力者が攻撃者ではない場合，すべての検証が成功するなら，偶然検証が成功する場合を除き，正しい復元値が honest な復元者に対して復元されるといえる．また，本議論より定理 4 がいえる．

定理 4：秘密情報 a の復元に関して，a に対応する $\alpha_2(a+\alpha_1)$，$\alpha_5(a+\alpha_4)$ のような [乱数] × ([秘密情報] + [コミットされた乱数]) という値を構成する [秘密情報] と [コミットされた乱数] に発生する差分が等しくなければ，偶然の場合を除いて 3.6 節手順②の検証は通過できない．

4.4.1.2 入力者が攻撃者と結託する場合の完全性

この場合，攻撃者は a，A_1，A_2 を知る．つまり，3.6 節手順②で要求される条件は α_1，α_4 の項が 0 となる以下のみである．

$$p_2q_1=1,\quad p_5q_3=1$$

この場合，3.6 節手順②は以下のように表され，p_1，p_3，p_4，p_6，q_1，q_2，q_3，q_4 を調整することにより，攻撃者は不正な値で検証を通過させることが可能である．

$$\alpha_1'-\alpha_1=(p_1q_1-1)a+(p_1-p_3)q_1A_1+(q_2-1)A_2=0$$
$$\alpha_4'-\alpha_4=(p_4q_3-1)a+(p_4-p_6)q_3A_2+(q_4-1)A_1=0$$

この時点で，$p_1,\ldots,p_6$ に関する条件はないため，引き続き以下が成り立つ．

$$\alpha_2(a+\alpha_1)'=\alpha_2(p_1a+p_2\alpha_1+(p_1-p_3)A_1)$$
$$\alpha_5(a+\alpha_4)'=\alpha_5(p_4a+p_5\alpha_4+(p_4-p_6)A_2)$$

この条件で，3.6 節手順③の検証を考えると，

$$\left\{\frac{\alpha_2(p_1a+p_2\alpha_1+(p_1-p_3)A_1)}{t_1\alpha_2}-\frac{\alpha_5(p_4a+p_5\alpha_4+(p_4-p_6)A_2)}{t_2\alpha_5}\right\}-(\alpha_1-\alpha_4)=0$$

上の等式を整理すると，

$$\left(\frac{p_1}{t_1}-\frac{p_4}{t_2}\right)a+\left(\frac{p_2}{t_1}-1\right)\alpha_1-\left(\frac{p_5}{t_2}-1\right)\alpha_4+\frac{(p_1-p_3)A_1}{t_1}-\frac{(p_4-p_6)A_2}{t_2}=0$$

前述のとおり各差分を調整する時点で攻撃者は α_1，α_4 を知らないため，上記式を意図的に成立させるためには，少なくとも α_1，α_4 の項が 0 となる必要があり，この場合以下が要求される．

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

ここで，上記条件より，差分 t_1，t_2 をいかなる値に調整しても，$p_2=t_1$，$p_5=t_2$ でなければ検証は通らない．よって，差分 t_1，t_2 の調整は無意味なことが分かる．この条件で復元値を計算すると，以下のようになる．

$$\begin{aligned}&\frac{\alpha_2(p_1a+p_2\alpha_1+(p_1-p_3)A_1)}{t_1\alpha_2}-\alpha_1\\&=\frac{p_1}{t_1}a+\left(\frac{p_2}{t_1}-1\right)\alpha_1+\frac{(p_1-p_3)A_1}{t_1}\\&=\frac{p_1}{t_1}a+\frac{(p_1-p_3)A_1}{t_1}\\&=\frac{p_1}{p_2}a+\frac{(p_1-p_3)A_1}{p_2}\end{aligned}$$

よって，入力者が攻撃者である場合，復元者は a を復元できない．しかし，入力者が攻撃者と結託しているため，入力者が 3.4 節および 3.5 節の処理中に自ら入力値を改ざんしたといえ，そもそも a は分散されていないと考えられる．よって，a が復元されないことは当然であり，a の入力者が実際には何を入力したかを考える必要がある．a の入力者が実際に入力した値とは，分散処理終了時点で，honest なサーバが保持するシェアから定義される値であり，これが復元されるなら問題ないといえる．よって，何が復元されるかを確認する．まず，各値の表記を以下のように簡略化する．ただし，$a'=p_1a+(p_1-p_3)A_1$，$a''=p_4a+(p_4-p_6)A_2$ である．

$$\alpha_2(p_1a+p_2\alpha_1+(p_1-p_3)A_1)=\alpha_2(a'+p_2\alpha_1)$$
$$\alpha_5(p_4a+p_5\alpha_4+(p_4-p_6)A_2)=\alpha_5\left(a''+p_5\alpha_4\right)$$

この場合，3.6 節手順②の検証は成功するとし，3.6 節手順③は以下となる．

$$\left\{\frac{\alpha_2(a'+p_2\alpha_1)}{t_1\alpha_2}-\frac{\alpha_5(a''+p_5\alpha_4)}{t_2\alpha_5}\right\}-(\alpha_1-\alpha_4)=0$$

整理すると，

$$\frac{1}{t_1}a'-\frac{1}{t_2}a''+\left(\frac{p_2}{t_1}-1\right)\alpha_1-\left(\frac{p_5}{t_2}-1\right)\alpha_4=0$$

よって，α_1，α_1 を知らない攻撃者がこの検証を意図的に成功させるためには，最低でも以下が要求される．

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

この場合，復元値は

$$\frac{\alpha_2(a'+p_2\alpha_1)}{t_1\alpha_2}-\alpha_1=\frac{a'}{t_1}=\frac{a'}{p_2}$$

つまり，$\alpha_2(a+\alpha_1)'$ を $\alpha_2(a'+p_2\alpha_1)$ と表記した場合，実際に分散された値は a'/p_2 と表せる．ここで，a の入力者が攻撃者ではない場合，$\alpha_2p_1(a+\alpha_1)=\alpha_2(p_1a+p_1\alpha_1)=\alpha_2(a'+p_1\alpha_1)$ と表記でき，$a'/p_1=(p_1a)/p_1=a$ となるから，この a'/p_2 という表記は入力者が攻撃者と結託す

るか否かを問わず，一般化された表記といえる．ここで，$\alpha_2(p_1a+p_2\alpha_1+(p_1-p_3)A_1)$ に対して $a'=p_1a+(p_1-p_3)A_1$ であり，

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

となる．つまり，honest な復元者は，a の入力者が実際に分散した値を復元処理で得られていることが分かる．よって，偶然不正な値で検証が通過する場合を除き，秘密情報の復元において，提案方式は 4.1 節で定義した完全性を有する．また，これまでの議論により，定理 5 がいえる．

定理 5：秘密情報 a の復元に関して，3.6 節手順①で復元者が得る α_2，α_5 への改ざん攻撃は，$\alpha_2(a+\alpha_1)$，$\alpha_5(a+\alpha_4)$ のような [乱数] × ([秘密情報] + [コミットされた乱数]) という値に関して，[秘密情報] と [コミットされた乱数] に同じ差分を発生させるのみであるため，定理 4 とあわせてこれは無意味な攻撃である．

4.4.2 秘匿計算結果の復元に対する完全性

秘匿計算結果に対する完全性の議論を行う．提案方式では，どのような計算を行っても秘密情報に対する値である [乱数] × ([秘密情報] + [コミットされた乱数]) と同型である [乱数] × ([計算結果] + [コミットされた乱数]) という値が計算される．さらに，この [計算結果] と [コミットされた乱数] はそれぞれ対応する項を持つ．すなわち，たとえば [計算結果] に c の項が含まれるなら [コミットされた乱数] には γ_1 の項が含まれ，[計算結果] に ab の項が含まれるなら [コミットされた乱数] には $\alpha_1\beta_1$ の項が含まれる．ただし，3.7 節に示されるように正負も等しいとは限らないがこれは安全性において問題とはならず，計算量を増やせば正負を等しくすることも可能である．よって，秘匿計算結果の復元に対しては [計算結果] と [コミットされた乱数] を対応する項どうしで独立に議論し，それぞれに発生する差分が等しくなることがいえれば，4.4.1 項での秘密情報の復元に対する議論と同様に完全性が証明できる．ただし，計算結果に対応する値は 2 通り生成されるため，c は γ_4 にも対応し，ab は $\alpha_4\beta_4$ にも対応するが，2 通りの値は独立であるため本議論では一方にのみ着目する．

まず，定理 4 より，3.6 節手順②の検証がすべて通過した場合，秘匿計算に用いられたすべての秘密情報に関しては [秘密情報] と [コミットされた乱数] に発生する差分は等しいといえる．以上より，[計算結果] に含まれる秘密情報単体の項（たとえば，[計算結果] $=ab+c+abc+a$ のように示される場合，2 項目の c と 4 項目の a が該当する）に対して 4.4.1 項と同様に完全性を証明できる．

次に，[計算結果] に含まれる秘密情報どうしの積の項（たとえば，[計算結果] $=ab+c+abc+a$ のように示される場合，1 項目の ab と 3 項目の abc が該当する）に対して議論を行う．これらの項は秘匿乗算の実行によって発生する．たとえば 3.7 節手順⑦の $ab+c$ における積 ab の計算に関する部分のみを取り出すと，以下の計算となる．

$$\begin{aligned}
&[\delta_2\{(ab)-(\alpha_1\beta_1)\}]_j\\
&=\alpha_2(a+\alpha_1)\times\beta_2(b+\beta_1)\times\left(\frac{\delta_2}{\alpha_2\beta_2\varepsilon_3}\right)\times[\varepsilon_3]_j\\
&\quad-\alpha_2(a+\alpha_1)\times\beta_0\beta_1\times\left(\frac{\delta_2}{\alpha_2\beta_0\varepsilon_4}\right)\times[\varepsilon_4]_j\\
&\quad-\alpha_0\alpha_1\times\beta_2(b+\beta_1)\times\left(\frac{\delta_2}{\alpha_0\beta_2\varepsilon_5}\right)\times[\varepsilon_5]_j
\end{aligned}$$

ここで，4.4.1 項の議論と同様に攻撃者が各項の乱数比 $\left(\frac{\delta_2}{\alpha_2\beta_2\varepsilon_3}\right)$ などを調整した場合，このシェアから復元される値は以下となる．ただし，この計算の実行時点では $\alpha_2(a+\alpha_1)$ や $\alpha_0\alpha_1$ にも不正が含まれる可能性はあるが，定理 4 より x を不正分として $\alpha_2(a+\alpha_1)'=x\alpha_2(a+\alpha_1)$ のように表すことができなければ 3.6 節手順②検証で検出できるため，各項に発生する差分をまとめて以下のように r_1，r_2，r_3 で表すことができる．

$$\begin{aligned}
&\delta_2\{(ab)-(\alpha_1\beta_1)\}'\\
&=r_1\delta_2(a+\alpha_1)(b+\beta_1)\\
&\quad-r_2\delta_2(a+\alpha_1)\beta_1-r_3\delta_2\alpha_1(b+\beta_1)\\
&=r_1\delta_2(ab+a\beta_1+b\alpha_1+\alpha_1\beta_1)\\
&\quad-r_2\delta_2(a\beta_1+\alpha_1\beta_1)-r_3\delta_2(b\alpha_1+\alpha_1\beta_1)\\
&=\delta_2\{r_1ab+(r_1-r_2)a\beta_1+(r_1-r_3)b\alpha_1\\
&\qquad+(r_1-r_2-r_3)\alpha_1\beta_1\}
\end{aligned}$$

つまり，ab に発生している差分 r_1 と対応する乱数の項 $\alpha_1\beta_1$ に発生している差分 $(r_1-r_2-r_3)$ が等しいことがいえれば，[計算結果] に含まれる秘密情報どうしの積の項に対しても，4.4.1 項と同様に完全性が証明できる．3.6 節手順③の検証では，上記式を δ_2 で除算し，3.6 節手順②で検証した乱数で減算する．つまり，上記式の $(r_1-r_2)a\beta_1$，$(r_1-r_3)b\alpha_1$ は 0 とならなければ 3.6 節手順③の検証で検出できる．ただし，$(r_1-r_2)a\beta_1+(r_1-r_3)b\alpha_1=0$ のようにこれらが打ち消し合う可能性もあるが，今までの議論と同様に攻撃者は α_1，β_1 を知らないことから，その確率は十分無視できる低確率 $1/p$ である．よって，検証が通過する場合 $(r_1-r_2)a\beta_1=(r_1-r_3)b\alpha_1=0$ が得られ，このとき $r_1=r_2=r_3$ となる．つまり，ab に発生している差分 r_1 と対応する乱数の項 $\alpha_1\beta_1$ に発生している差分 $(r_1-r_2-r_3)=r_1$ が等しいことが証明された．つまり，秘匿乗算に関して，攻撃者は少なくとも計算式のすべての項に生じさせる差分を等しい値に調整しなければ，攻撃者の意図する値で検証を通過させることはできない．よって，[計算結果] に含まれる秘密情報どうしの積の項に対しても，4.4.1 項と同様に完全性が証明できるといえる．

また，定理 5 は計算結果の復元に対しても同様なことが成り立つといえる．さらに，3.8 節は 3.7 節の計算式に 1 項

追加するのみであり，ここで追加された乱数 w_1 に対応する秘密情報はない．また，検証にはコミットメントスキームによる正当な w_1 が使用される．よって，これら 2 通りの w_1 が打ち消し合わない場合は検証を通過できず，検証を通過する場合は復元値に w_1 の項が残らないため，完全性に影響は生じない．

以上の議論より，定理 4 と同様に，秘匿計算の結果を復元する場合においても，[計算結果] と [コミットされた乱数] を秘密情報に関して分解したとき，対応する項どうしに発生する差分は等しくなることから，偶然の場合を除いて不正は必ず検出される．

また，処理中の各検証が偶然通過する確率は $1/p$ であり，十分大きな素数 p を法とした大きな有限体を仮定している提案方式では，$1/p$ は十分小さいといえる．さらに，正しく復元を行うためには，3.6 節で行われるすべての検証を通過する必要があり，3.6 節中の検証回数を η とすると不正な値が復元される確率は $1/p^{\eta}$ となり，$\eta > 0$ よりこれは negligible といえる．

以上の議論より，提案方式は 4.1 節で定義した完全性を実現する．

5. UTP モデルへの拡張と安全性レベルの変更

5.1 変換用乱数の生成および配布

3.3 節前提 (2) のように，提案方式は従来の TUS 方式と同様に，変換用乱数組は事前にサーバに格納されているとした．これは，いわゆる信頼できる第三者（Trusted Third Party，以降 TTP）を仮定し，TTP が変換用乱数を生成およびサーバへ配布していること同等である．TUS 方式において，この TTP に対する要件をまとめると以下の 3 つである．

[TUS 方式に要求された第三者（TTP）の役割]

(1) 攻撃者と結託しない（変換用乱数を攻撃者へ提供しない）．

(2) サーバを corrupt しない（サーバが知る情報を入手しない）．

(3) プロトコルから逸脱しない（正しい変換用乱数を生成，配布する）．

上記 (3) に関して，提案方式では malicious な攻撃者を想定しているため変換用乱数に間違いがあれば検出できる．よって，プロトコルから逸脱すること許容した第三者を Untrusted Third Party（UTP）と定義し，提案方式において，この UTP に対する要件を以下に示す．

[提案方式に要求される第三者（UTP）の役割]

(1) 攻撃者と結託しない（変換用乱数を攻撃者へ提供しない）．

(2) サーバを corrupt しない（サーバが知る情報を入手しない）．

この UTP は，以下のようにして 3.3 節前提 (2) を実行できる．

[UTP による変換用乱数の生成および配布]

① UTP は，k 個の乱数 $\varepsilon_{0,j}$ を生成し，その積 $\varepsilon_0 = \prod_{j=0}^{k-1} \varepsilon_{0,j}$ を計算する．

② UTP は，ε_0 を n 台のサーバにシェアが線形性を持つ方式で秘密分散する．

③ $n = k$ の場合，UTP は，$\varepsilon_{0,j}$ を S_j に送信する．

④ $n > k$ の場合，UTP は，k 個の $\varepsilon_{0,j}$ を n 台のサーバにそれぞれ XOR 法で秘密分散する．

UTP は，上記の手順②から④でサーバへ送信する値を任意の値に調整できるため，配布された値の正当性は保証できない．しかし，ここでの不正は 3.6 節復元処理で検出可能であるため，追加の検証処理は不要である．なぜなら，変換用乱数へ差分が発生することと，3.4 節手順④などで攻撃者が偽の変換用乱数を送信することは同等であり，不正な値であれば復元処理で検出できることは証明済みである（4.4 節の完全性の議論と同様に，4.4 節における p_1，q_1，t_1 などと同等の差分が発生するのみである）．よって，本プロトコル追加による，安全性の低下はない．

5.2 計算能力が無限な攻撃者に対する完全性の実現

提案方式においては，採用した 2.4 節における (2) のタイプのコミットメントスキームが安全性証明の要となっている．この部分は他のプロトコルへ簡単に置き換えることができる．たとえば，この部分を 2.4 節における (1) のタイプのコミットメントスキームやハッシュ関数に置き換えると，機密性および完全性は両者ともに計算能力が有限な攻撃者に対してのみ安全となる．同様に，この部分に UTP を用いて，計算能力が無限な攻撃者に対して完全性を実現する方法を示す．ただし，ここでの UTP は，5.1 節で定義したものと同様である．方法は非常にシンプルであり，3.4 節手順③や 3.7 節手順①において，サーバは乱数をコミットする代わりに UTP へ送信する．そして，UTP は 3.5 節手順②で，復元者から要求された乱数を送信する．この場合の安全性であるが，[提案方式に要求される第三者の役割] の (1) より，攻撃者は変換用乱数を得られない．また，[提案方式に要求される第三者の役割] の (2) より，UTP が自ら演算に関与し，サーバを corrupt したり通信路を盗聴したりすることはない．よって，UTP は秘密情報に加算される乱数と変換用乱数のみを知るため，これらのみでは秘密情報を求めることはできない．よって，安全性の低下はないといえる．また，5.1 節の処理と 5.2 節の処理における UTP は同一でなくてよい．

5.3 安全性レベルの変更

4 章では，計算能力が無限な攻撃者に対して機密性を証明し，計算能力が有限な攻撃者に対して完全性を証明した．さらに，5.2 節では 3 章におけるコミットメントスキームを

表 1 乱数の扱いに採用する方法とその安全性
Table 1 The way of handling random values and its security.

	ハッシュ関数	コミットメントスキーム	UTP
機密性	計算量的	情報理論的	情報理論的
完全性	計算量的	計算量的	情報理論的

UTP に置き換えることにより，計算能力が無限な攻撃者に対して完全性を有する方式への拡張方法を示した．以上のように，提案方式の安全性レベルは 3 章においてコミットされる乱数を誰がどのように扱うかで決定され，安全性は低下するが単純にコミットメントスキームをハッシュ関数に置き換えることでも提案方式は成立する．**表 1** に，乱数の扱いに採用する方法とその安全性を示す．

6. 他方式との比較

本章では，$n < 2k-1$ に適用できる dishonest majority を想定する SPDZ 系列（SPDZ2）および従来方式 TUS3 方式（提案方式 1）と本稿 3 章で示した手法の比較を**表 2** に示す．表 2 の値は事前処理に関する部分はすべて無視して示してある．データサイズは，1 台のサーバが保持するシェアを基に算出した．つまり，提案方式における変換用乱数や TUS3 方式における 1 に対するシェア集合，SPDZ2 における multiplication triple など，直接入力と関係ない値は含まない．計算量は，特別に記載がない限り 1 台のサーバに対する値である．通信量は，通信路に流れる全データ量とした．また，ラウンド数という観点で通信回数を比較する．

【記号定義】

- I：入力（秘密情報）の数
- t：SPDZ2 において，演算中に復元された値の数
- A：1 回の加減算に関する計算量．
- M：1 回の乗算に関する計算量．
- C_d^S：Shamir 法の分散に関する計算量．
- C_r^S：Shamir 法の復元に関する計算量．
- C_d^X：XOR 法の分散に関する計算量．
- C_r^X：XOR 法の復元に関する計算量．
- C_c：コミットメントに関する計算量．
- C'：コミットメントに関する通信量．
- D：秘密情報のデータサイズ．
- D_C：コミットメントに関するデータサイズ．

各計算量の大小関係について示す．コミットメントスキームはハッシュ関数などの暗号技術によって実現され，秘密分散法よりも計算量が多い．さらに，Shamir 法において，分散における計算量よりも復元における計算量の方が大きくなる．よって，$C_d^S < C_r^S \ll C_c$ である．XOR 法は分散および復元を XOR 演算のみで行える軽量な秘密分散

表 2 他方式との比較表
Table 2 Comparison with other methods.

		提案方式	TUS3 方式	SPDZ2
振舞い		**malicious**	**Semi-honest**	**malicious**
機密性		**Information-theoretic**	**Information-theoretic**	**Computational**
完全性		**Computational**	**nothing**	**Computational**
可用性		$n \geq k$	$n \geq k$	$n = k$
1 入力に対するデータサイズ		$6(k+1)D + 2kD_c$	$(k+1)D$	$2D$
計算量	分散	$2kC_r^X$(入力者) $2C_r^S$ (サーバ)	kC_d^X(入力者)	$3A$ (入力者) $2A$ (サーバ)
	復元	$2kC_r^X + 2C_r^S + 2kIC_c$ (復元者)	kC_r^X(復元者)	$(4n+4t-6)A + (4t+2)M + 8C_c$
	和	$8C_r^X + 4C_d^X + 4C_r^S$	$4C_r^X + C_d^X + C_r^S$	$2A$
	積	$8C_r^X + 4C_d^X + 4C_r^S$	$2C_r^X + C_d^X$	$2(k-1)A$
	積和	$12C_r^X + 4C_d^X + 4C_r^S$	$5C_r^X + C_d^X + C_r^S$	$2(k-1)A$
通信量	分散	$2\{k^2 + (k+1)n\}D$	$(k+1)nD$	$2nD$
	復元	$2\{(k^2+1) + kI\}D + 2kIC'$	$(k^2+1)D$	$8C'$
	和	$(8k^2 + 16kn)D$	$(2k^2 + nk)D$	0
	積	$(8k^2 + 16kn)D$	$(4k^2 + 3k + nk + 3n)D$	$4nD$
	積和	$(12k^2 + 20kn)D$	$(5k^2 + 3k + nk + 3n)D$	$4nD$
ラウンド数	分散	3	1	1
	復元	1	1	8
	和	4	5	0
	積	4	2	1
	積和	4	5	1

法であり，文献 [7] のシミュレーション結果より，復元よりも分散における処理時間の方が大きくなるといえるため，$C_r^X < C_d^X < C_d^S < C_r^S \ll C_c$ となる．ここで，$A < M$ は自明であるが，M と C_r^X は単純に比較できない．提案方式や TUS3 方式の計算量においては A，M は支配的な要素ではないため表中では省略するが，各方式における計算量の大小関係および本稿での主張に影響はない．

提案方式は情報理論的安全である TUS3 方式を malicious な攻撃者に対しても安全となるように改良することが最大のモチベーションであり，表 2 より提案方式の機密性および可用性は SPDZ2 に勝り，完全性は SPDZ2 と同等なレベルを実現した．(n, k) を柔軟に設定できるという性質は honest majority を仮定している Araki らの高速な MPC [13] でも実現されず，安全性レベルに関しては提案方式が非常に優れているといえる．

次に，提案方式の性能を検討する．提案方式および TUS3 方式では可用性を実現するために各乱数を XOR 法で秘密分散する必要があるため，両者の 1 入力に対するデータサイズは SPDZ2 より劣る．また，提案方式は秘密情報の検証のために多くのシェアを保持する必要があるため，検証のない TUS3 方式と比較して少なくとも 6 倍のデータサイズが要求される．補足として，表 2 中の 3 方式の中で，SPDZ2 のみが事前処理にて Somewhat 準同型暗号による暗号化，秘匿計算，複合の各処理が行われる．表 2 では事前処理に関する表を省いたが，処理中にこれを格納できる分の容量は要求される．ここで，表 2 で事前処理に対する評価を省いた理由は，事前処理は任意のタイミング（サーバの CPU や通信路に余裕のあるタイミングなど）で行うことができ，秘匿計算時には直接影響しないことが理由である．しかし，データサイズに関して，提案方式の変換用乱数や TUS3 方式の 1 のシェア集合，SPDZ2 における multiplication triple は，計算に応じて十分な数を事前に生成し，演算に利用するまで保持し続ける必要があるため，積和演算で使用して破棄するまでサーバの容量を圧迫する．よって，データサイズのみ事前処理で生成された値を考慮した比較を別途行う．**表 3** に，3 入力による 1 回の積和演算を行うために必要なデータサイズを示す．

表 3 に関して，3.7 節手順①より，提案方式における積和演算には 12 個組の変換用乱数が必要となる．TUS3 方式においては，1 回の積和演算で 2 組の 1 に対するシェア集合が必要となる．SPDZ2 においては，1 組の multiplication triple を秘匿乗算で利用する．これらに加え，積和演算に必要な 3 入力に対するデータサイズを合計した値を表 3 に示した．表 3 より，提案方式におけるデータサイズが表 2 と同様に最も大きくなっている．SPDZ2 が最も小さい理由としては，$n = k$ 限定であることがあげられる．TUS3 方式や提案方式では，積和演算の各項を秘匿するために変換用乱数や 1 に対するシェアを利用するが，提案方式では秘密情報に乱数を加算していることから積和演算時の項が多く，結果として必要な変換用乱数の数も多くなっている．よって，事前処理で生成された値を考慮してもデータサイズの評価は変わらず，提案方式は多くのメモリ容量が要求される．

表 3 事前処理を考慮したデータサイズの比較
Table 3 Comparison of data size considering preprocessing phase.

	提案方式	TUS3	SPDZ2
1 回の積和演算を行う場合に必要なデータサイズ	$\mathbf{30(k+1)D + 6kD_c}$	$\mathbf{5(k+1)D}$	$\mathbf{12D}$

通信量に関して，データサイズの増加にともない通信量も増加し，さらに提案方式および TUS3 方式は加算にも通信が発生するため，全体として SPDZ2 より通信量が多くなる．

計算量に関して，$n = k$ 限定で採用できる非常に軽量な加法的秘密分散法を採用した SPDZ2 は全体的に計算量が小さくなる．しかし，提案方式や TUS3 方式においても，$n = k$ の設定の場合は Shamir 法および XOR 法の両者は加法的秘密分散法に置き換えることができ，計算量を抑えることができる．さらに，提案方式において 3.4 節のような事前処理を 3.7 節秘匿計算に対しても行うことで，計算量および通信量を削減できる．具体的な方法は，2 値の加算を想定した事前処理，3 値の乗算を想定した事前処理というように，演算のタイプに応じて事前処理を行うことになる．ほとんどの演算は積和演算（適切に計算式を変更することで加算および乗算も可能）の組合せに分解することが可能であるため，たとえば 3.7 節のような 3 値による積和演算を仮定した事前処理を十分な回数行っておくことで，実際の演算での計算量と通信量を大幅に削減できる．

以上のように提案方式は，全体として SPDZ2 より性能が良いとは言い難い．これは，提案方式の主な貢献が TUS3 方式に検証処理を追加するという点であるため，基本的に性能は SPDZ2 が高く，次点で TUS3 方式，そして提案方式の順番となる．性能の向上は今後の課題である．

次にラウンド数に関して，計算中にシェアの計算および復元が必要な提案方式と TUS3 方式はラウンド数が多くなる．しかし，復元処理のラウンド数に関して，検証処理がない TUS3 方式が最も小さいことは自明であるが，SPDZ2 が 8 ラウンドかかる一方で提案方式は 1 ラウンドで復元が行える．SPDZ2 の検証処理では，まず演算中に復元された値すべてに対して 4 ラウンドの MACCheck プロトコルによ

る検証が行われ，その後計算結果に対して再度 MACCheck プロトコルが行われるため，合計 8 ラウンドとなっている．一方で，提案方式における検証対象の値はすべての入力と出力であり，入力の検証に必要な値は 3.3 節および 3.4 節で生成され，出力の検証に必要な値は出力の秘匿計算で生成される．よって，提案方式における復元処理は，各サーバが復元者に対して必要な値を送信するだけで，あとは復元者のローカルな計算となり 1 ラウンドでの復元を実現している．

7. 結論

本稿では，3.3 節に示した 3 つの条件下において，攻撃者が知らない変換用乱数を用いて，機密性を情報理論的安全に実現し，偶然の場合を除き正しい値が復元されるという完全性を実現する検証可能秘密分散および検証可能秘匿計算を提案した．

6 章で議論したように，提案方式は高い安全性と引き換えに性能が全体的に劣っている．よって，今後の第 1 の課題は性能の向上である．TUS3 における XOR 法の採用と同様に高効率なプロトコルへの置き換えや，3.4 節を例にして可能な限りの処理を事前処理で行うなどが第 1 の方針としてあげられる．

参考文献

[1] 株式会社日立製作所：ビッグデータ × AI：ビッグデータ × AI（人工知能）：日立，入手先 ⟨https://www.hitachi.co.jp/products/it/bigdata/bigdata_ai/index.html⟩（参照 2020-11-11）．

[2] Shamir, A.: How to Share a Secret, *Comm. ACM*, Vol.22, No.11, pp.612–613 (1979).

[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，岩村恵市：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 (2018).

[4] ムハンマド カマル アフマド アクマル アミヌディン，岩村恵市：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，Vol.59, No.9, pp.1581–1595 (2018).

[5] 鴇田恭平，岩村恵市：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 (2018).

[6] Backes, M., Kat, A. and Patra, A.: Computational Verifiable Secret Sharing Revisite, *ASIACRYPT 2011*, Lee D. and Wang X. (Eds.), LNC, Vol.7073, p.590–609, Springer (2011).

[7] Kurihara, J., Kiyomoto, S., Fukushima, K. and Tanaka, T.: On a fast (k,n)-threshold secret sharing scheme, *IEICE Trans. Fundamentals of Electronics, Communications and Computer Science*, Vol.E91-A, No.9, pp.2365–2378 (2008).

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

[9] Damgård, I., Pastr, V., Smart, N. and Zakaria, S.: Multiparty Computation from Somewhat Homomorphic Encryptio, *CRYPTO 2012*, pp.643–662 (2012).

[10] Damgård, I., Kelle, M., Larrari, E., et al.: Practical Covertly Secure MPC for Dishonest Majority–Or: Breaking the SPDZ Limits, *ESORICS 2013*, LNCS 8134, pp.1–18, Springer (2013).

[11] Keller, M., Orsiniy, E. and Schollz, P.: MASCOT: Faster Malicious Arithmetic Secure Computation with Oblivious Transfer:, *Proc. 2016 ACM SIGSAC Conference on Computer and Communications Security*, pp.830–842 (2016).

[12] Keller, M., Pastro, V. and Rotaru, D.: Overdrive: Making SPDZ Great Again, *EUROCRYPT 2018*, pp.158–189 (2018).

[13] Araki, T., Barak, A., Furukawa, J., et al.: Optimized Honest-Majority MPC for Malicious Adversaries–Breaking the 1 Billion-Gate Per Second Barrier, *2017 IEEE Symposium on Security and Privacy*, pp.843–862 (2017).

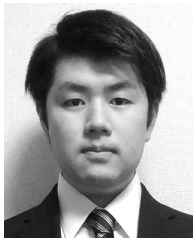
[14] Beaver, D.: Efficient multiparty protocols using circuit randomization, *CRYPTO 1991*, Feigenbaum, J. (Ed.), LNCS, Vol.576, pp.420–432, Springer (1991).

[15] Pedersen, T.P.: Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing, *CRYPTO 1991*, Feigenbaum J. (Ed.), LNCS, Vol.576, pp.129–140, Springer (1992).

[16] Haitner, I. and Reingold, O.: Statistically-hiding commitment from any one-way function, *STOC'07*, pp.1–10, ACM (2007).

[17] Chida, K., Genkin, D., Hamada, K., et al.: Fast Large-Scale Honest-Majority MPC for Malicious Adversaries, *CRYPTO 2018*, Shacham, H. and Boldyreva, A. (Eds.), LNCS, Vol.10993, pp.34–64, Springer (2018).

[18] Brakerski, Z., Gentry, C. and Vaikuntanathan, V.: (Leveled) Fully Homomorphic Encryption without Bootstrapping, *ACM TOCT*, O'Donnell, R. (Ed.), Vol.6, No.3, pp.309–325, ACM (2009).



落合 将吾

1996 年生．2019 年東京理科大学工学部電気工学科卒業．2021 年同大学大学院工学研究科修士課程修了．同年東日本電信電話（株）入社．

岩村 恵市　（正会員）

1958 年生．1980 年九州大学工学部情報工学科卒業．1982 年同大学大学院情報工学研究科修士課程修了．同年キャノン（株）入社．1994 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．