Conditionally Secure Secrecy Computation using Secret Sharing Scheme for $n < 2k - 1$

(full paper)

Ahmad Akmal Aminuddin Mohd Kamal[1] and Keiichi Iwamura[2]

[1]Tokyo University of Science, Tokyo, Japan
ahmad@sec.ee.kagu.tus.ac.jp
[2]Tokyo University of Science, Tokyo, Japan
iwamura@ee.kagu.tus.ac.jp

Abstract. Typically, when secrecy multiplication is performed in multiparty computation using Shamir's (k, n) threshold secret sharing scheme, the result is a polynomial with degree of $2k - 2$ instead of $k - 1$. This causes a problem where, in order to reconstruct a multiplication result, the number of polynomials needed will increase from k to $2k - 1$. Shingu et al. proposed a method to solve the problem that the degree of polynomial increases when secrecy multiplication is performed by using the $(scalar\ value \times polynomial)$ approach instead of the typical $(polynomial \times polynomial)$. However, this method is not secure when a combination operation, such as a product-sum operation, is performed. In this paper, we propose a multiparty computation that uses a secret sharing scheme that is secure against a product-sum operation but does not increase the degree of polynomial of the output. We prove that all combinations of the basic operations (addition, subtraction, multiplication, and division) can be performed securely using this scheme. We also propose three preconditions and finally show that our proposed method is information-theoretic secure against a passive adversary.

Keywords: conditionally secure, secret sharing, secrecy computation, product-sum operation, n < 2k-1.

1 Introduction

In recent years, with the advancement of big data and the IoT ecosystem, there is considerable anticipation that it will be possible to utilize big data to obtain valuable statistical data. However, this utilization could affect the individuals' privacy if their privacy information is leaked. Therefore, a large amount of research is being conducted on utilizing big data while ensuring that sensitive material, such as individuals' privacy information, is protected.

In this paper, with the aim of allowing big data to be utilized while the individuals' privacy information is still protected, we propose a multiparty computation technique that can protect these data while performing computation. Two main techniques have

been proposed for performing computation while protecting information: homomorphic encryption [3, 5, 6, 11, 12, 14, 19] and secret sharing schemes [2, 7, 9, 13, 17, 20]. However, homomorphic encryption is known to be typically computationally very expensive. Therefore, secret sharing schemes that have a relatively low computational cost are preferable to homomorphic encryption when considering utilization in a cloud system.

A secret sharing scheme is a protocol in which a single secret is divided into shares, which are then distributed. An example of a secret sharing scheme is Shamir's (k, n) threshold secret sharing scheme [16]. It divides a secret s into an n number of shares, distributes the shares, and restores the original secret s from a threshold k number of shares. Any $k-1$ or smaller number of shares reveals nothing about the secret. From this, we know that Shamir's (k, n) threshold secret sharing scheme also provides tolerance to a broken server when $n > k$.

Conventional methods of multiparty computation using a secret sharing scheme can perform secrecy addition and subtraction easily. However, this is not so in the case of secrecy multiplication, where the degree of a polynomial changes from $k-1$ to $2k-2$ for each multiplication of polynomials. To restore the multiplication result, the number of shares required increases from k to $2k-1$. To solve this problem, Shingu et al. proposed a multiparty computation method using a secret sharing scheme called the TUS method [17]. In this method, the secret is first encrypted with a random number; when performing secrecy multiplication, the encrypted secret is momentarily restored as a scalar value and multiplication is realized using the $(scalar\ value \times polynomial)$ approach to prevent an increase in the polynomial degree. However, in the TUS method, when computation involving a combination of operations, such as that of $ab + c$, is performed, if the adversary has information about one of the inputs and outputs, he/she can specify the value of the remaining two inputs.

In this paper, we propose a multiparty computation using a secret sharing scheme that is secure even when computation involving a combination of multiple different operations is performed. Typically, unconditionally secure multiparty computation is considered impossible under the setting of $2k - 1 > n$. In contrast, this means that secure multiparty computation using a secret sharing scheme is possible with certain conditions. Therefore, in this study, we also considered the conditions needed to achieve information-theoretic secure multiparty computation using a secret sharing scheme in the setting of $2k - 1 > n$, even when computation involving a combination of different types of operations is performed. If the conditions can be realized, we can state that the proposed multiparty computation method is practical. In addition, we verify the effectiveness of our proposed method by comparing it with SPDZ 2 proposed by Dåmgard et al. [12].

2 Previous work

2.1 TUS method

We explain the protocol proposed in the TUS method [17]. Note that variables a, b, c and random numbers $\alpha_j, \beta_j, \gamma_j$ generated during the protocol distribution and protocol addition process are derived from finite field $GF(p)$, provided that the secret input a, b and the random numbers for protocol multiplication do not include 0. All computations including protocol distribution are performed under finite field $GF(p)$.

Notation:

- $\overline{[a]}_i$: Share of a for player P_i.
- $[a]_i$: Set of shares, such as $\left(\overline{[\alpha a]}_i, \overline{[\alpha_0]}_i, \dots, \overline{[\alpha_{k-1}]}_i\right)$, on share a for server S_i.

Preconditions

(1) Inputs in protocol multiplication do not include value 0.

Distribution Protocol

- Input: a

1. Player A selects k random numbers $\alpha_0, \alpha_1, \dots, \alpha_{k-1}$ from finite field $GF(p)$ and computes the value of $\alpha = \prod_{j=0}^{k-1} \alpha_j$.
2. Player A then computes αa as an encrypted secret and distributes $\alpha a, \alpha_0, \alpha_1, \dots, \alpha_{k-1}$ to n servers using Shamir's (k, n) threshold secret sharing scheme.
3. Each server S_i has $[a]_i = \overline{[\alpha a]}_i, \overline{[\alpha_0]}_i, \dots, \overline{[\alpha_{k-1}]}_i$ as a set of shares about secret a.

Restoration Protocol

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

1. A player who wishes to restore the secret collects k sets of shares $[a]_j$.
2. The player restores the value of $\alpha_0, \alpha_1, \dots, \alpha_{k-1}$ from $\overline{[\alpha a]}_j, \overline{[\alpha_0]}_j, \dots, \overline{[\alpha_{k-1}]}_j$ and computes the value of $\alpha = \prod_{j=0}^{k-1} \alpha_j$.
3. The player obtains information about original secret a using the following computation.

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

Addition and Subtraction Protocol

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

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

1. Servers S_j collect $\overline{[\alpha_j]}_l, \overline{[\beta_j]}_l$ and restore α_j, β_j. Servers S_j then select a random number γ_j, compute γ_j/α_j , γ_j/β_j , and send to server S_0.
2. Server S_0 then computes the value of $\gamma/\alpha = \prod_{j=0}^{k-1} \gamma_j/\alpha_j$, $\gamma/\beta = \prod_{j=0}^{k-1} \gamma_j/\beta_j$ and sends them to all servers S_i.
3. Servers S_i then compute $\overline{[\gamma(a \pm b)]}_i = \frac{\gamma}{\alpha}\overline{[\alpha a]}_i + \frac{\gamma}{\beta}\overline{[\beta b]}_i$.
4. Servers S_j distribute γ_j to all servers S_i by using Shamir's (k, n) threshold secret sharing scheme.
5. Each server S_i now holds $[a \pm b]_i = \overline{[\gamma(a \pm b)]}_i, \overline{[\gamma_0]}_i, \dots, \overline{[\gamma_{k-1}]}_i$ as a set of shares for the result of $a \pm b$.

Multiplication and Division Protocol

- Input:
  $[a]_j = \overline{[\alpha a]}_j, \overline{[\alpha_0]}_j, \dots, \overline{[\alpha_{k-1}]}_j$ $(j = 0, 1, \dots, k-1)$
  $[b]_j = \overline{[\beta b]}_j, \overline{[\beta_0]}_j, \dots, \overline{[\beta_{k-1}]}_j$ $(j = 0, 1, \dots, k-1)$
- Output:
  $[c]_i = [ab]_i = \overline{[\alpha\beta ab]}_i, \overline{[\alpha_0\beta_0]}_i, \dots, \overline{[\alpha_{k-1}\beta_{k-1}]}_i$ $(i = 0, 1, \dots, n-1)$

1. Server S_o collects $\overline{[\alpha a]}_j$ from k servers. Server S_o then restores αa and sends it to all servers S_i.
2. Servers S_i compute $\overline{[\alpha\beta ab]}_i = \alpha a \times \overline{[\beta b]}_i$.
3. Servers S_j collect $\overline{[\alpha_j]}_l, \overline{[\beta_j]}_l$ and restore α_j, β_j. Servers S_j then calculate $\alpha_j\beta_j$.
4. Servers S_j distribute $\alpha_j\beta_j$to all servers S_i by using Shamir's (k, n) threshold secret sharing scheme.
5. Each server S_i now holds $[ab]_i = \overline{[\alpha\beta ab]}_i, \overline{[\alpha_0\beta_0]}_i, \dots, \overline{[\alpha_{k-1}\beta_{k-1}]}_i$ as a set of shares for the result of ab.
6. Protocol division can be achieved by changing the computation in Step 2 to $[\beta b/\alpha a] = [\beta b]_i/\alpha a$ and the computation in Step 3 to β_j/α_j. Thus, servers S_i hold $[c]_i = \overline{[\beta b/\alpha a]}_i, \overline{[\beta_0/\alpha_0]}_i, \dots, \overline{[\beta_{k-1}/\alpha_{k-1}]}_i$ as a set of shares for the result of $c = b/a$.

Typically, in secrecy division computation, when the divisor is 0, we are not able to find a solution for the division. Therefore, cases where the divisor is 0 must be excluded. In our proposed method, we can identify whether the divisor for secrecy multiplication is 0. In Step 1, we can identify that the divisor is 0 when $\alpha a = 0$. When $\alpha a = 0$, computation is halted, thus preventing secrecy division with divisor 0.

The TUS computation method has been proven to be information-theoretic secure against the three different adversaries described in the following, which are all assumed to be passive.

Adversary 1: The adversary has information from $k-1$ servers. According to this information, the adversary attempts to know two inputs and an output of the secrecy computation.

Adversary 2: One of the players who inputted a secret is the adversary. The adversary has knowledge of one of the inputs and the random number used to encrypt the input. In addition, the adversary also has information from $k-1$ servers. According to this information, the adversary attempts to know the remaining one input or output of the secrecy computation.

Adversary 3: The player who reconstructed the output is the adversary. The adversary has knowledge of the k amount of information needed to reconstruct the output. In addition, the adversary has information from $k-1$ servers. According to this information, the adversary attempts to know two inputs of the secrecy computation.

However, the TUS method is not secure against Adversary 4, who has information of one of the inputs and outputs. This is because in a 2-input-1-output computation, when the adversary has information of one of the inputs and outputs, the second input can be leaked, regardless of the security level of the method used. Therefore, in a 2-input-1-output computation, Adversary 4 is not considered.

2.2 Problem of the TUS method

The proposed protocol for a combination of different computations (secrecy multiplication and addition) in the TUS method to compute a product-sum operation of $ab+c$ is shown below. In the protocol, Steps 1–5 show the computation of the multiplication of ab and Steps 6–10 show the computation for $ab+c$. All variables a, b, c and random numbers $\alpha_j, \beta_j, \lambda_j, \gamma_j$ are derived from finite field $GF(p)$ (However, no inputs or random numbers used include value 0).

Protocol for Product-Sum Operation of $ab+c$
(TUS Method)

- Input:
  $[a]_j = \overline{[\alpha a]}_j, \overline{[\alpha_0]}_j, \dots, \overline{[\alpha_{k-1}]}_j \ (j = 0, 1, \dots, k-1)$
  $[b]_j = \overline{[\beta b]}_j, \overline{[\beta_0]}_j, \dots, \overline{[\beta_{k-1}]}_j \ (j = 0, 1, \dots, k-1)$
  $[c]_j = \overline{[\lambda c]}_j, \overline{[\lambda_0]}_j, \dots, \overline{[\lambda_{k-1}]}_j \ (j = 0, 1, \dots, k-1)$
- Output:
  $[ab+c]_i = \overline{[\gamma(ab+c)]}_i, \overline{[\gamma_0]}_i, \dots, \overline{[\gamma_{k-1}]}_i$
  $(i = 0, 1, \dots, n-1)$

1. Server S_o collects $\overline{[\alpha a]}_j$ from k servers. Server S_o then restores αa and sends it to all servers S_i.
2. Servers S_i compute $\overline{[\alpha\beta ab]}_i = \alpha a \times \overline{[\beta b]}_i$.
3. Servers S_j collect $\overline{[\alpha_j]}_l, \overline{[\beta_j]}_l$ and restore α_j, β_j. Servers S_j then compute $\alpha_j\beta_j$.
4. Servers S_j distribute $\alpha_j\beta_j$ to all servers S_i by using Shamir's (k, n) threshold secret sharing scheme.

5. Each server S_i now holds $[ab]_i = \overline{[\alpha\beta ab]_i}, \overline{[\alpha_0\beta_0]_i}, \dots, \overline{[\alpha_{k-1}\beta_{k-1}]_i}$ as a set of shares for the result of ab.
6. Servers S_j collect $\overline{[\alpha_j\beta_j]}_l, \overline{[\lambda_j]}_l$ and restore $\alpha_j\beta_j, \lambda_j$. Servers S_j then select a random number γ_j, compute $\gamma_j/\alpha_j\beta_j, \gamma_j/\lambda_j$, and send them to server S_0.
7. Server S_0 then computes the values of $\gamma/\alpha\beta = \prod_{j=0}^{k-1}\gamma_j/\alpha_j\beta_j, \gamma/\lambda = \prod_{j=0}^{k-1}\gamma_j/\lambda_j$ and sends them to all servers S_i.
8. Servers S_i then compute $\overline{[\gamma(ab+c)]}_i = \frac{\gamma}{\alpha\beta}\overline{[\alpha\beta\delta ab]}_i + \frac{\gamma}{\lambda}\overline{[\lambda\eta c]}_i$.
9. Servers S_j distribute γ_j to all servers S_i by using Shamir's (k, n) threshold secret sharing scheme.
10. Each server S_i now holds $[ab+c]_i = \overline{[\gamma(ab+c)]}_i, \overline{[\gamma_0]}_i, \dots, \overline{[\gamma_{k-1}]}_i$ as a set of shares for the result of $ab + c$.

Since this protocol for the secrecy product-sum operation is a 3-input-1-output operation, we assume that there are three players who input a secret and one player who reconstructs the output. Therefore, we also need to consider the case of Adversary 4, where the adversary is one of the players who inputted a secret and at the same time is the player who reconstructed the output. For example, Adversary 4 is the player who inputted secret b, random number β, and at the same time is the player who reconstructed output $ab + c$, random number γ. In addition, Adversary 4 also has information of $\alpha a, \gamma/\alpha\beta, \gamma/\lambda$ obtained from $k - 1$ servers during the computation process. By using the information of $\beta, \gamma, \gamma/\alpha\beta$, Adversary 4 is able to gain information of random number α used to encrypt secret a. With information of αa and random number α, Adversary 4 is able to decrypt secret a. With information of secret a, b and output $ab + c$, information of secret c is also leaked to the adversary. Thus, we can conclude that, when Adversary 4 has information about secret b, output $ab + c$, and additional information from $k - 1$ servers, the remaining two inputs are eventually leaked. Therefore, the TUS method of product-sum operation is not secure against Adversary 4. However, a product-sum operation using the TUS method remains information-theoretic secure against Adversaries 1–3.

2.3 SPDZ 2 method

Dåmgard et al. proposed a secure multiparty computation called SPDZ 2 [12] that utilizes a somewhat homomorphic encryption (SHE) and is secure against a dishonest majority under the setting $n = k$. In SPDZ 2, the owner of the secret is one of n players involved in the multiparty computation. Moreover, in SPDZ 2, even when $n - 1$ players form a coalition, provided that the owner keeps his/her share of the secret secure, the original secret cannot be reconstructed from $n - 1$ shares.

SPDZ 2 consists of a preprocessing and an online phase. It ensures the confidentiality of inputted secrets by using an additive secret sharing scheme. Through the SPDZ 2 method, secrecy addition is easily achievable. Secrecy multiplication in SPDZ 2 is based on Beaver's circuit randomization. To perform secrecy multiplication, a multiplicative triple is used. Specifically, to perform secrecy multiplication between shares

$\langle x\rangle, \langle y\rangle$, shares of random numbers $\langle a\rangle, \langle b\rangle, \langle c\rangle$, called a multiplicative triple, which satisfy $a \cdot b = c$, are prepared beforehand in the preprocessing phase. For the actual multiplication computation, first the values of $d = open(\langle x\rangle - \langle a\rangle), e = open(\langle y\rangle - \langle b\rangle)$ are reconstructed. Then, by using equation $\langle x \cdot y\rangle = d \cdot e + e \cdot \langle a\rangle + d \cdot \langle b\rangle + \langle c\rangle$, multiplication of xy is performed. However, the problem of SPDZ 2 is that, in the generation process of the multiplicative triple in the preprocessing phase, SHE, which is known to be computationally very expensive, is used. Therefore, although the online phase of SPDZ 2 is very effective, the computational cost of its preprocessing phase is high and it consumes a huge amount of processing time.

3 Proposed method

In this section, we consider a new method to overcome the problem of the TUS method that it is not secure against Adversary 4 when a product-sum operation is performed. As explained in Section II, in the TUS method, when secrecy multiplication and secrecy addition are combined to compute product-sum operation $ab + c$, the remaining input can be leaked when the adversary has knowledge of one of the inputs and outputs of the computation. Therefore, it is very important to prevent the remaining information from being leaked even in the case of Adversary 4. To achieve this, in our proposed protocol we suggest an approach in which random numbers that are not known to the adversary are implemented. Therefore, we need to define a new precondition that sets of shares of random numbers exist that are not known to the adversary. To facilitate this, we assume that a set of shares of secret 1 exists that is derived from random numbers δ_j, η_j $(j = 0, 1, \dots, n-1)$ that are not known to the adversary.

$$[1]^{(1)}{}_i = \left(\overline{[\delta]}_i, \overline{[\delta_0]}_i, \dots, \overline{[\delta_{k-1}]}_i\right)$$

$$[1]^{(2)}{}_i = \left(\overline{[\eta]}_i, \overline{[\eta_0]}_i, \dots, \overline{[\eta_{k-1}]}_i\right)$$

This set of shares is called the set of shares on 1. For example, this set of shares on 1 can be easily prepared by using the protocol below.

Protocol of Set of Shares on 1

1. Generate k random numbers $\delta_0, \delta_1, \dots, \delta_{k-1}$.
2. Calculate random number $\delta = \prod_{j=0}^{k-1} \delta_j$.
3. Distribute random number $\delta, \delta_0, \delta_1, \dots, \delta_{k-1}$ to servers S_i (i $= 0, \dots, n-1$)using Shamir's(k, n)threshold secret sharing scheme.
4. Each server S_i now holds $[1]_i = \overline{[\delta]}_i, \overline{[\delta_0]}_i, \dots, \overline{[\delta_{k-1}]}_i$ as a set of shares for secret 1.

Hence, we define condition (2) as follows.

(2) There are sets of shares on 1 derived from random numbers that are not known to the adversary.

3.1 Secrecy product-sum computation

Every server holds a set of shares of secret input through the distribution protocol shown in Section II. All the computation, including the distribution protocol, is performed in finite field $GF(p)$.

Notation:

- $\overline{[a]}_i$: Share of a for player P_i.
- $[a]_i$: Set of shares, such as $\left(\overline{[\alpha a]}_i, \overline{[\alpha_0]}_i, \dots, \overline{[\alpha_{k-1}]}_i\right)$ on share a for server S_i.

Preconditions:

(1) Inputs in the multiplication protocol never include value 0.

(2) There are sets of shares on 1 derived from random numbers (excluding the value 0) unknown to the adversary. Here, the set of shares is

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

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

Protocol for Product-Sum Operation of $ab + c$
(Proposed Method)

- Input:
  $[a]_j = \overline{[\alpha a]}_j, \overline{[\alpha_0]}_j, \dots, \overline{[\alpha_{k-1}]}_j$ $(j = 0, 1, \dots, k-1)$
  $[b]_j = \overline{[\beta b]}_j, \overline{[\beta_0]}_j, \dots, \overline{[\beta_{k-1}]}_j$ $(j = 0, 1, \dots, k-1)$
  $[c]_j = \overline{[\lambda c]}_j, \overline{[\lambda_0]}_j, \dots, \overline{[\lambda_{k-1}]}_j$ $(j = 0, 1, \dots, k-1)$
- Output:
  $[d]_i = [ab + c]_i = \overline{[\gamma(ab + c)]}_i, \overline{[\gamma_0]}_i, \dots, \overline{[\gamma_{k-1}]}_i$
  $(i = 0, 1, \dots, n-1)$

1. Server S_o collects $\overline{[\alpha a]}_j$ from k servers. Server S_o then restores αa.
2. Server S_o then sends αa to all servers S_i.
3. Servers S_i compute $\overline{[\alpha\beta ab]}_i = \alpha a \times \overline{[\beta b]}_i$.
4. Server S_o collects $\overline{[\alpha\beta ab]}_j, \overline{[\lambda c]}_j$ from k servers and restores $\alpha\beta ab, \lambda c$.
5. Server S_o then sends $\alpha\beta ab, \lambda c$ to all servers S_i.
6. Servers S_i compute $\overline{[\alpha\beta\delta ab]}_i = \alpha\beta ab \times \overline{[\delta]}_i$. ($\overline{[\delta]}_i$is a share on set of shares $[1]^{(1)}{}_i$)
7. Servers S_i compute $\overline{[\lambda\eta c]}_i = \lambda c \times \overline{[\eta]}_i$. ($\overline{[\eta]}_i$is a share on set of shares $[1]^{(2)}{}_i$)
8. Servers S_j collect $\overline{[\alpha_j]}_l, \overline{[\beta_j]}_l, \overline{[\lambda_j]}_l, \overline{[\delta_j]}_l, \overline{[\eta_j]}_l$ and restore $\alpha_j, \beta_j, \lambda_j, \delta_j, \eta_j$. Servers S_j then select a random number γ_j.
9. Servers S_j compute $\gamma_j/\alpha_j\beta_j\delta_j$, $\gamma_j/\lambda_j\eta_j$ and send to server S_0.
10. Server S_0 then computes the value of $\gamma/\alpha\beta\delta = \prod_{j=0}^{k-1} \gamma_j/\alpha_j\beta_j\delta_j$, $\gamma/\lambda\eta = \prod_{j=0}^{k-1} \gamma_j/\lambda_j\eta_j$ and sends them to all servers S_i.
11. Servers S_i then compute $\overline{[\gamma(ab + c)]}_i = \frac{\gamma}{\alpha\beta\delta}\overline{[\alpha\beta\delta ab]}_i + \frac{\gamma}{\lambda\eta}\overline{[\lambda\eta c]}_i$.

12. Servers S_j distribute γ_j (if the operation involves only the secrecy multiplication of ab, $\gamma_j = \alpha_j\beta_j$; if the operation involves secrecy addition, $\gamma_j = \gamma_j$) to all servers S_i by using Shamir's (k, n) threshold secret sharing scheme.
13. Each server S_i now holds $[ab + c]_i = \overline{[\gamma(ab + c)]_i}, \overline{[\gamma_0]_i}, \ldots, \overline{[\gamma_{k-1}]_i}$ as a set of shares for the result of $ab + c$.

Based on the protocol for the product-sum operation shown above, Steps 1–3 are the same as the TUS method of secrecy multiplication and Steps 8–13 are secrecy addition based on TUS. Therefore, Steps 4–7 are new steps that require a set of shares on 1, as stated in precondition (2). As we know, in a product-sum operation of $ab + c$, if $c = 0$, secrecy multiplication of ab can be realized, and if $a = 1$, secrecy addition of $b + c$ can be realized. Therefore, in the product-sum operation protocol mentioned previously, if $c = 0$, secrecy multiplication is achieved, and Steps 4–7 and 9–11 can be skipped. However, in Step 8, only α_j, β_j is reconstructed without the need to generate γ_j. Moreover, if multiplication in Step 3 is replaced with division by αa, $\gamma_j = \alpha_j\beta_j$ in Step 12 is replaced with $\gamma_j = \beta_j/\alpha_j$, and secrecy division is realizable. If $a = 1$(random number α is also equal to 1), secrecy addition is achieved and Steps 1–3 can be skipped. However, because multiplication in Step 3 is skipped, computation after Step 4, which involves $\overline{[\alpha\beta ab]}_j$ and $\alpha\beta ab$, becomes $\overline{[\beta b]}_i$ and βb, $\overline{[\alpha\beta\delta ab]}_i$ in Step 6 becomes $\overline{[\beta\delta b]}_i$, reconstruction of α_j in Step 8 is skipped, and α_j in Step 9 and onwards is set as $\alpha_j = \alpha = 1$. In addition, by changing the symbol of addition to subtraction, secrecy subtraction is also realizable. In conclusion, based on the aforementioned product-sum operation protocol, we can achieve all four basic computations, including secrecy division and subtraction. However, the evaluation of the effectiveness of our proposed method through the introduction of Steps 4–7 is discussed in a later section.

3.2 Security of single product-sum operation

The TUS proposed method is a 2-input-1-output operation and can be represented by Fig. 1, where two inputs $[a]_j, [b]_j$ are inputted in the secrecy computation box, which contains the protocols shown in Section II, to produce output $[c]_j$. The TUS method was proven to be secure against Adversary 1, who has access to information from $k - 1$ servers, Adversary 2, who has information on input $[a]_j$ or $[b]_j$ in addition to information from $k - 1$ servers, and Adversary 3, who has information on the output $[c]_j$ in addition to information from $k - 1$ servers.

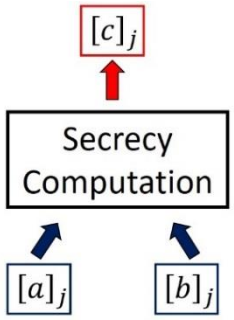

Fig. 1. 2-input-1-output secrecy computation (TUS method)

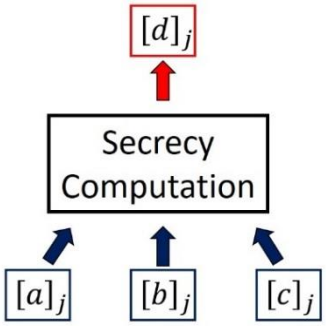

Fig. 2. 3-input-1-output secrecy computation (Proposed method)

Since our proposed secrecy computation method is a 3-input-1-ouput computation, it can be represented by Fig. 2, where three inputs $[a]_j, [b]_j, [c]_j$ are inputted into the secrecy computation box to produce an output $[d]_j$. However, if one of the outputs is made known (for example, $a = 1$ or $c = 0$), we can realize a 2-input-1-output computation (secrecy addition, subtraction, multiplication, and division), as in Fig. 1. Here, we define Adversary 4 and Adversary 5 as follows. The attack is considered a success if the adversary is able to achieve the aim of learning the information that he/she wants to know.

Adversary 4: In the product-sum operation, one of the players who inputted the secret and the player who reconstructed the output constitute the adversary. Adversary 4 has information of one of the inputs (and the random number used to encrypt it) and the information needed to reconstruct the output. In addition, the adversary also has knowledge of information from $k - 1$ servers. According to this information, the adversary attempts to learn the remaining two inputs.

Adversary 5: In the product-sum operation, two of the players who inputted secrets constitute the adversary. Adversary 5 has information of two of the secrets (and the random numbers used to encrypt them). In addition, the adversary also has knowledge of information from $k - 1$ servers. According to this information, the adversary attempts to learn the remaining one input or the output of the computation.

Here, suppose that in the case of Adversary 4, where one of the inputs is treated as a constant and does not contribute to the process of decoding other values, this adversary can be treated in the same manner as Adversary 3. In contrast, in the case of Adversary 5, if one of the known inputs is assumed to be constant and does not contribute to decoding other secrets, Adversary 5 can be treated in the same manner as Adversary 2. Further, Adversary 1 is part of Adversary 4 and Adversary 5, where both Adversaries 4 and 5 have the information obtained by Adversary 1, which is information from $k-1$ servers. Moreover, in a 3-input-1-output computation, regardless of the security level of the method used, if two out of three inputs and the output are leaked to the adversary, the remaining one input can also be leaked. Similarly, when all three of the inputs are known to the adversary, the output can also be leaked to the adversary. Therefore, we do not consider these two types of adversary. We can state that our proposed secrecy computation method is secure if it is secure against Adversaries 4 and 5.

In the following, we evaluate the safety security of our proposed method toward Adversaries 4 and 5.

Evaluation of Security against Adversary 4

Assume that the player who inputted input b is the adversary. He/she also has information from $k-1$ servers. Therefore, in the process of inputting data, Adversary 4 has information about b, β, β_i $(i = 0, \dots, k-1)$, and in Steps 1–2, he/she learns about αa, in Steps 4–5 about $\alpha\beta ab, \lambda c$, in Step 8 about $\alpha_j, \beta_j, \lambda_j, \delta_j, \eta_j, \gamma_j$ $(j = 0, 1, \dots, k-2)$, in Step 10 about $\gamma/\alpha\beta\delta, \gamma/\lambda\eta$, and finally in the reconstruction process, about $\gamma, \gamma_i, ab + c$. Therefore, the evaluation of security against Adversary 4 can be translated to the problem of determining whether he/she can learn about the remaining inputs a, c from the information about $b, \beta, \alpha a, \alpha\beta ab, \lambda c, \gamma/\alpha\beta\delta, \gamma/\lambda\eta, \gamma, \beta_i, \gamma_i (i = 0, 1, \dots, k-1), \alpha_j, \beta_j, \lambda_j, \delta_j, \eta_j, \gamma_j$ $(j = 0, 1, \dots, k-2), ab + c$.

First, in order to simplify the problem, we redefined the parameters above to avoid any duplication of a parameter. As a result, we can transform the problem into determining whether from information $b, \beta, \alpha a, \lambda c, \gamma, \alpha\delta, \lambda\eta, ab + c, \beta_i, \alpha_j, \lambda_j, \delta_j, \eta_j (j = 0, 1, \dots, k-2)$ the adversary can learn about the remaining inputs a, c.

To obtain information about secret a from αa, the adversary must first obtain information of random number α. The information that is related to random number α is $\alpha a, \alpha\delta, \alpha_j, \delta_j (j = 0, 1, \dots, k-2)$ ($b, \beta, c, \lambda, \eta$ is independent of α, a). However, even from this information, random number α and secret a is not leaked. Therefore,

$$H(\alpha) = H\left(\alpha | \alpha_j \ (j = 0, 1, \dots, k-2)\right)$$

$$H(\delta) = H\left(\delta | \delta_j \ (j = 0, 1, \dots, k-2)\right)$$

$$H(a) = H\left(a | \alpha a, \alpha\delta, \alpha_j, \delta_j \ (j = 0, 1, \dots, k-2)\right)$$

In the proposed TUS method, because there was no implementation of a set of shares on 1 that hold information about random number δ, $\alpha\delta$ becomes α and secret a can be leaked. In contrast, in our proposed method, through the implementation of a set of shares on 1 that hold information about random number δ, we were able to prevent the leakage of secret a to the adversary.

In addition, to obtain secret c from λc, the adversary needs first to learn about random number λ, and therefore, the same can also be asserted about secret c.

$$H(\lambda) = H\big(\lambda|\lambda_j\ (j = 0, 1, \dots, k-2)\big)$$

$$H(\eta) = H\big(\eta|\eta_j\ (j = 0, 1, \dots, k-2)\big)$$

$$H(c) = H\big(c|\lambda c, \lambda\eta, \lambda_j, \eta_j\ (j = 0, 1, \dots, k-2)\big)$$

By implementing the set of shares on 1, which hold information about random number η, in contrast to the TUS method, our method is able to prevent random number λ from being leaked, thus allowing our method to prevent secret c from being known to the adversary.

Finally, because Adversary 4 also has information about output $d = ab + c$, he/she has information about $d = ab + c$ and b; however, with no information about either secret a or c, he/she is not able to obtain any information about the remaining inputs. Therefore, it can be stated that

$$H(a) = H\big(a|\mathrm{ab} + c, \beta, b, \alpha a, \alpha\delta, \lambda c, \lambda\eta, \alpha_j, \delta_j, \lambda_j, \eta_j (j = 0, 1, \dots, k-2)\big)$$

$$H(c) = H\big(c|ab + c, \beta, b, \alpha a, \alpha\delta, \lambda c, \lambda\eta, \alpha_j, \delta_j, \lambda_j, \eta_j (j = 0, 1, \dots, k-2)\big)$$

In addition, the evaluation above remains valid even if the adversary is the player who inputted input a or c. Therefore, our proposed method is secure against Adversary 4.

Evaluation of Security against Adversary 5

Assume that the player who inputted input b, c is the adversary. He/she also has information from $k-1$ servers. Therefore, in the process of inputting data, Adversary 4 has information of $b, \beta, \beta_i\ (i = 0, \dots, k-1)$, and in Steps 1–2 learns about αa, in Steps 4–5 about $\alpha\beta ab, \lambda c$, in Step 8 about $\alpha_j, \beta_j, \lambda_j, \delta_j, \eta_j, \gamma_j\ (j = 0, 1, \dots, k-2)$, and finally in Step 10, about $\gamma/\alpha\beta\delta\ , \gamma/\lambda\eta$. Therefore, the evaluation of security against Adversary 5 can be translated into the problem of determining whether the adversary can learn about the remaining input a or output $ab + c$ from the information of $b, \beta, c, \lambda, \alpha a, \alpha\beta ab, \lambda c,\quad \gamma/\alpha\beta\delta\ , \gamma/\lambda\eta\ , \gamma, \beta_i, \lambda_i (i = 0, 1, \dots, k-1), \alpha_j, \beta_j, \lambda_j, \delta_j, \eta_j, \gamma_j (j = 0, 1, \dots, k-2)$.

First, in order to simplify the problem, we redefine the parameters above to avoid any duplication of a parameter. As a result, we can change the problem into determining whether the adversary can learn about the remaining input a or output $ab + c$ from information $b, \beta, c, \lambda, \alpha a, \gamma/\alpha\delta\ , \gamma/\eta\ , \alpha_j, \delta_j, \eta_j, \gamma_j, \gamma(ab + c)_j\ (j = 0, 1, \dots, k-2)$.

To obtain information of secret a from αa, the adversary must first obtain information of random number α. The information that is related to random number α is $\alpha a, \gamma/\alpha\delta\ , \alpha_j, \delta_j, \gamma_j\ (j = 0, 1, \dots, k-2)$. Even from this information, random number α and secret a cannot be leaked. Therefore, the following statements are true.

$$H(\alpha) = H\big(\alpha|\alpha_j\ (j = 0, 1, \dots, k-2)\big)$$

$$H(\gamma) = H\big(\gamma|\gamma_j\ (j = 0, 1, \dots, k-2)\big)$$

$$H(\delta) = H\big(\delta|\delta_j\ (j = 0, 1, \dots, k-2)\big)$$

$$H(a) = H\big(a|\ \gamma/\alpha\delta\ , \alpha_j, \delta_j, \gamma_j\ (j = 0, 1, \dots, k-2)\big)$$

In the TUS method, because there was no implementation of a set of shares on 1 that hold information about random number $\delta, \eta,\ \gamma/\lambda\eta$ becomes γ/λ and $\gamma/\alpha\delta$ becomes γ/α. From information of γ/λ and λ, random number γ is leaked to the adversary. With information on γ/α and γ, the adversary can learn the value of random number α. Information of α and αa allows information of secret a to be leaked. With the leaked information of a and initial information of b, c, the adversary can learn about output $ab + c$. In contrast, our proposed method utilizes a set of shares on 1 that hold information about random number δ, η, which is derived from a random number that is not known to the adversary, and thus, we are able to prevent secret a from being leaked to the adversary.

In addition, the adversary has information about $\gamma(ab + c)_j$, but he/she cannot learn about $\gamma(ab + c)$ from this. Therefore, it can be stated that

$$H\big(\gamma(ab + c)\big) = H\big(\gamma(ab + c)|\gamma(ab + c)_j\ (j = 0, 1, \dots, k-2)\big)$$

Next, we consider whether the output of the secrecy computation and random number γ can be leaked to the adversary. First, the information related to random number γ is $\gamma/\alpha\delta\ , \gamma/\eta\ , \alpha_j, \delta_j, \gamma_j, \eta_j\ (j = 0, 1, \dots, k-2)$. However, even with this information, the adversary cannot learn about random number γ. Therefore,

$$H(\gamma) = H\big(\gamma|\gamma_j\ (j = 0, 1, \dots, k-2)\big)$$

$$H(\alpha) = H\big(\alpha|\alpha_j\ (j = 0, 1, \dots, k-2)\big)$$

$$H(\delta) = H\big(\delta|\delta_j\ (j = 0, 1, \dots, k-2)\big)$$

$$H(\eta) = H\big(\eta|\eta_j\ (j = 0, 1, \dots, k-2)\big)$$

$$H(\gamma) = H\big(\gamma|\ \gamma/\alpha\delta\ , \gamma/\eta\ , \alpha_j, \delta_j, \gamma_j, \eta_j\ (j = 0, 1, \dots, k-2)\big)$$

In the TUS method, because there was no implementation of a set of shares on 1 that hold information about random number $\eta\ ,\ \gamma/\lambda\eta$ becomes γ/λ. With information of $\gamma/\lambda\ , \lambda$, the adversary can learn about random number γ. In contrast, our proposed method utilizes a set of shares on 1 that hold information about random number η, which is not known to the adversary, and therefore, we can prevent the adversary from learning about random number γ.

Finally, Adversary 5 may know about secret b, c, but without any information of output $ab + c$, he/she cannot learn about the remaining input. Therefore,

$$H(a) = H\big(a|b, \beta, c, \lambda, \alpha a, \gamma/\alpha\delta\ , \gamma/\eta\ , \alpha_j, \delta_j, \eta_j, \gamma_j, \gamma(ab + c)_j\big)$$

$$H(ab + c) = H\big(ab + c|b, \beta, c, \lambda, \alpha a, \gamma/\alpha\delta\ , \gamma/\eta\ , \alpha_j, \delta_j, \eta_j, \gamma_j, \gamma(ab + c)_j\big)$$

In addition, the aforementioned evaluation remains valid, even if the adversary is the player who inputted secret a, c or a, b. Therefore, we can state that our proposed method is secure against Adversary 5.

3.3 Security of combination of multiple product-sum operation

In general, any computation that comprises the four basic operations (addition, subtraction, multiplication, and division) can be decomposed into a combination of multiple product-sum operations. For example, the computation of $a = f(a_1, a_2, \dots, a_{2m}, a_{2m+1}) = a_{2m+1}(a_1a_2 + a_3a_4 + \cdots + a_{2m-1}a_{2m})$ can be divided into multiple combinations of product-sum operations:

$$\text{Product-sum operation } 1 : f_1 = f(a_1, a_2, 0) = a_1a_2$$

$$\text{Product-sum operation } 2 : f_2 = f(a_3, a_4, f_1) = a_3a_4 + a_1a_2$$

$$\vdots$$

Product-sum operation m : $f_m = f(a_{2m-1}, a_{2m}, f_{m-1}) = a_1a_2 + a_3a_4 + \cdots + a_{2m-1}a_{2m}$

Product-sum operation $m+1$: $f_{m+1} = f(a_{2m+1}, f_m, 0) = a_{2m+1}(a_1a_2 + a_3a_4 + \cdots + a_{2m-1}a_{2m}) = a$

By combining the basic product-sum operation boxes shown in Fig. 2, the above computation can be represented as in Fig. 3. However, because all the outputs of the boxes, except the last box, are not restored, in every connection between boxes the output of each box is inputted into the next box in its encrypted state. Moreover, each computation for each box is performed by the same set of servers. In contrast, the computation of a can in general be represented as in Fig. 4; however, if we were to decompose it into a basic product-sum operation, we could state that the secrecy computation box in Fig. 4 is composed of the operations in Fig. 3.

Here, we consider the combination of product-sum operations shown in Section III. For example, regardless of the security level of the computation method used in the box shown in Fig. 4, if all the inputs except a_1 are known, the input a_1 is eventually leaked from $a_1 = f^{-1}(a, a_2, \dots, a_{2m}, a_{2m+1})$.

On the other hand, in the computation of a, if two of the inputs (for example, a_1, a_2) are not leaked, we can state that these two inputs cannot be leaked. This is because $a_1 = f^{-1}(a, a_2, \dots, a_{2m}, a_{2m+1})$, and if the value of a_2 is not known, the value of a_1 cannot be specified. The same applies to the opposite situation. Therefore, we define Adversary 6 as follows.

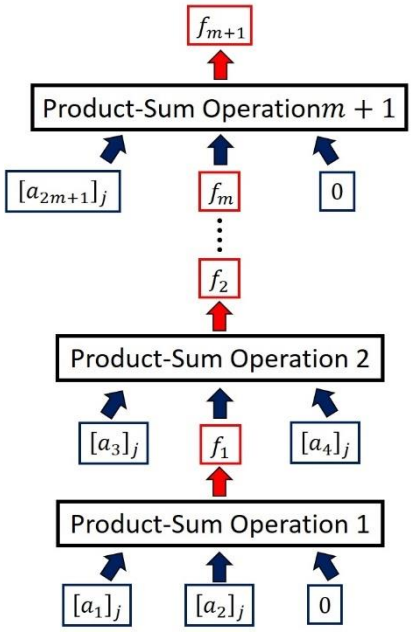

Fig. 3. Computation of a using combination of multiple product-sum operation

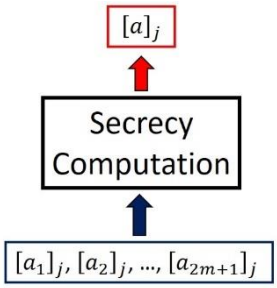

Fig. 4. General computation of a

Adversary 6: In a t-input-1-output computation, $t-1$ and below players are the adversary. Only the remaining two inputs or one input and one output are not known. According to the information that he/she has, the adversary attempts to learn about the remaining two unknowns.

The evaluation of security against Adversary 6 is shown in the following. However, because the consecutive computation of multiple product-sum operations must be executed by the same set of servers, we propose the following precondition (3).

(3) In secrecy computation involving consecutive computation, the position of a share in a set of shares that are handled by each server is fixed.

In Shamir's (k, n) threshold secret sharing scheme, the computation of the product-sum operation is defined and can be represented as in Fig. 2, while the combination of multiple computations is as shown in Fig. 3, where all computation boxes are independent of each other. In other words, a group of servers that performs a secrecy computation does not have to be the same group of servers that conducted the previous computation, and a combination of servers can be freely used to perform a computation. In contrast, in our proposed secrecy computation method, if servers have participated in a computation, they retain fragments of information from that computation. Assuming the case where multiple computations are performed consecutively, if the same server handles random numbers that are different from those in the previous computation, the server has information of two different random numbers. This risks a situation where the adversary is able to obtain more than a k number of information data from $k-1$ servers, since one server may have more than one fragment of information of the random number. Therefore, according to Condition 3, even in the condition where multiple computations are performed repeatedly, the random numbers that are handled by a server are limited to the same random numbers handled in a previous computation. In other words, for example, in secrecy multiplication, the values of random numbers $\alpha_j$and β_j are reconstructed, and the server that distributes the value of random number $\alpha_j\beta_j$ also reconstructs the same random number$\alpha_j\beta_j$ even in the next computation.

However, suppose a situation where $n > k$ and some of the servers broke down, for example, servers S_j that handled random numbers α_j, β_j have broken down and were replaced with servers S_j , which did not previously participate in any of the computation. Here, in a secret sharing scheme, we assume that information from $k-1$ out of n servers is leaked to the adversary. Therefore, out of a set of servers that participated in a secrecy computation, if $k-1$ servers are dishonest servers that leaked information to the adversary, we can state that the new server that is used to replace the broken server is not a dishonest server. Therefore, even with the addition of new condition (3), no problem arises. In addition, since each server has information only of fragments of the secret, the adversary is not able to collect more than k of those fragments. Therefore, the secret is safe from reconstruction by the adversary.

In the following, under the conditions mentioned previously, we explain the evaluation of our proposed method's effectiveness against Adversary 6 when multiple product-sum operations are performed consecutively, as shown in Fig. 3.

Evaluation of Security against Adversary 6

We consider three different situations in the evaluation of our proposed method's effectiveness against Adversary 6.

1. Two out of three inputs in a product-sum operation box are not known, while all the remaining inputs and outputs are known.

   Two inputs in a product-sum operation box are unknown. In other words, one input and one output of a product-sum operation box are known to the adversary. Thus, in this case Adversary 6 is the same as Adversary 4. We proved that our proposed method is secure against Adversary 4. Therefore, we can also state that the remaining two unknown inputs will remain unknown to Adversary 6.

Next, for example, the product-sum operation box mentioned previously is Box 2 shown in Fig. 3, where all the inputs for the remaining boxes and the final output are leaked to the adversary. In this case, because our proposed method is secure against Adversary 4, even if all information except the two unknown inputs in Box 2 are leaked, we can state that these two unknowns will remain unknown to Adversary 6. This does not change , regardless of the position of the product-sum operation box in the computation.

2. Two of the unknown inputs are each inputted in different boxes while the remaining inputs and the output for the last box are known.

   If two of the unknown inputs are each inputted in different product-sum operation boxes, the remaining two out of three inputs of that box can be leaked to the adversary. For example, two unknown inputs are each inputted in Box 1 and Box 2 respectively in Fig. 3. The remaining inputs for Box 1 and Box 2 can be leaked to the adversary. In Box 1, even if two inputs are known to the adversary, without the remaining one input, the adversary cannot learn information of the output, thus adding another unknown input to Box 2. Therefore, in Box 2, because two unknown inputs are inputted, and, based on the previous case, our proposed method is secure against Adversary 4, we can state that the two unknown inputs in Box 2 cannot be leaked to Adversary 6. This does not depend on the manner in which the boxes are combined.

3. The output for the last box and one input are not known while all the remaining inputs are known.

   In a box into which unknown inputs are inputted, the output cannot be leaked. Therefore, inputs related to that output are also protected from leakage. Because of this, one input and output for the last product-sum operation box are also unknown. In this case, we can state that two inputs of the last box are known to the adversary. However, our proposed method is proved to be secure against Adversary 5, where the adversary has information of two inputs. Therefore, we can state that, even if two inputs of the last box are known to the adversary, the remaining one input and output of the box cannot be leaked to Adversary 6.

4 Comparison with previous work and discussion

4.1 Comparison of our method and previous methods

Examples of multiparty computation methods that use secret sharing schemes are SPDZ 2 [12] proposed by Dåmgard et al. and the TUS method [17] proposed by Shingu. Since SHE, which is known to be computationally very expensive, is used in the preprocessing phase of SPDZ 2, it is computationally secure against a dishonest majority. In addition, to show the improvement achieved by our proposed method as compared to the TUS method, we also include this method in our comparison.

SPDZ 2 is limited to the setting $n = k$, where the owner of the secret is one of the players involved in the secrecy computation. Provided that the owner protects his/her

own share, even if $n-1$ players, excluding the owner, form a coalition, the protocol is secure and the secret cannot be leaked, since insufficient shares are collected to restore the secret. In particular, SPDZ 2 supposes an active adversary and is secure against a dishonest majority. Moreover, because SPDZ 2 uses SHE in the preprocessing phase, it is considered computationally secure. However, the implementation of SHE renders the preprocessing phase extremely computationally expensive. Moreover, SPDZ 2 is secure against computation that involves a combination of different types of operation, such as the combination of secrecy addition and multiplication. However, it cannot perform secrecy division directly from shares a and b inputted by the player (it is possible to compute the secrecy division using a method of secrecy multiplication if shares on $1/a$ and b are distributed by the player.).

In contrast, our proposed method and the TUS method are not limited only to situations where $n = k$. Both these methods are capable of performing secrecy computation even under the setting of $k \leq n$, although our proposed method shows effectiveness under the setting $n < 2k-1$, it is also usable in the setting $n \geq 2k-1$. However, they both suppose a passive adversary and include a proposed secrecy computation that is information-theoretic secure against a passive adversary. In addition, the TUS method suffers a problem when different types of operation are combined to perform a more complicated computation, whereas our proposed method is secure even when different types of computation are combined. However, the TUS method requires only one condition, whereas our proposed method requires three conditions. Moreover, our proposed method can perform secrecy division directly from shares of a and b, as shown in Section III. All the comparisons discussed above are summarized in Table 1.

Table 1. Comparison with previous work

	Poposed method	TUS method	SPDZ 2 method
Condition of n, k	$n \geq k$	$n \geq k$	$n = k$
Type of adversary	Passive adversary	Passive adversary	Active adversary
Type of security	Information-theoretic secure	Information-theoretic secure	Computationally secure
Combination of computation	Unlimited	Limited to the same type of computation	Unlimited
Number of conditions needed	3	1	0
Secrecy division	Directly executable	Directly executable	Possible (provided that inverse value of secret is given)

A comparison of the computational and communication cost of our proposed method, TUS, and SPDZ 2 is shown in Table 2. However, the number of communications is evaluated as the number of rounds in proportion to the direction of the communication. We define the parameters used in the comparison as follows.

Definition of Parameters

- d_1 : Size of share from secret sharing scheme
- d_2 : Size of share from SHE
- C_1 : Computational cost of secret sharing scheme
- C_2 : Computational cost of SHE

Parameter d_1 is usually almost the same size as the original secret, whereas d_2 is typically larger than the original secret. Therefore, $d_2 > d_1$. Moreover, C_1 is considerably smaller than C_2. Therefore, $C_1 \ll C_2$. However, in a secret sharing scheme, the computational cost of the distribution and the reconstruction process differs, but since both are considerably smaller than C_2, we consider that the computation cost of both the distribution and reconstruction process of a secret sharing scheme is C_1.Moreover, because our proposed method does not include an authentication process, such as zero knowledge proof and a message authentication code, all processing costs for authentication processes are omitted. In addition, the values in the comparison shown in Table 2 include the cost of preprocessing, distribution, and reconstruction. In Table 2, in computational cost, the proposed and the TUS method do not include the computational cost for SHE C_2, and therefore, we can state that our method is better in terms of this cost than SPDZ 2. In terms of communication cost, the merits and demerits of each method depend on n, k, d_1, d_2. Finally, a comparison of each method's number of rounds, since our proposed method includes the processes of generating, restoring, and distributing random numbers in secret distribution and secrecy computation, shows that the total number of rounds of our proposed method is considerably more than that of TUS and SPDZ 2.

However, since our proposed method does not contain the computational cost of SHE, it is lighter than SPDZ 2 in terms of overall processing cost. Our proposed method also allows different types of operation to be combined, while remaining secure, which is not possible in the TUS method.

Table 2. Comparison of computational cost and communication cost of proposed method and previous work

	Process	Proposed method	TUS method	SPDZ 2 method
Computational Cost	$a \times b$	$2(3k+2) \cdot C_1$	$2(3k+2) \cdot C_1$	$8 \cdot C_1 + 2 \cdot C_2$
	$a + b$	$(10k+7) \cdot C_1$	$3(2k+1) \cdot C_1$	$3 \cdot C_1$
	$ab + c$	$3(4k+3) \cdot C_1$	Not executable	$9 \cdot C_1 + 2 \cdot C_2$
Communication cost	$a \times b$	$\{3(k+n)(k+1) - k\}d_1$	$\{3n(k+1) + k(3k+2)\}d_1$	$(12n-2)d_2$
	$a + b$	$\{5(k+n)(k+1) + n(k+4)\}d_1$	$\{n(3k+4) + 3k(k+1)\}d_1$	$3nd_2$
	$ab + c$	$\{6(k+n)(k+1) + 4n\}d_1$	Not executable	$(13n-2)d_2$
Number of rounds	$a \times b$	6	6	6
	$a + b$	10	6	2
	$ab + c$	10	Not executable	6

4.2 Discussion

We discuss the realizability of our three proposed conditions in the following.

(1). Inputs in secrecy multiplication do not include value 0.

In secrecy multiplication, if the secret inputted is 0, αa that is restored in the protocol of secrecy multiplication is $\alpha a = 0$. From this, the adversary can know that secret a is 0, since a random number does not contain value 0. However, information that does not contain value 0 is abundant, such as medical data. For example, a patient's pulse and blood pressure are usually recorded as positive values, where the value 0 refers to dead patients and is not used in statistics calculation in the medical field. Moreover, information such as blood glucose level and much more is also recorded as positive values other than 0. Therefore, when performing secrecy computation from data collected from patients admitted to or being treated at a hospital, the condition that requires the exclusion of value 0 in inputs does not raise a serious problem. In addition, the condition of exclusion of input 0 applies only to secrecy multiplication. The inclusion of input 0 does not cause a problem when used in secrecy addition, subtraction, and division. Therefore, although our next task is to avoid this condition, a considerable amount of information does not require value 0 and our proposed method is an effective multiparty computation protocol for this information.

(2). There are sets of shares on 1 created from random numbers not known to the adversary.

The simplest method to fulfill this condition is to obtain a set of shares on 1 from a trustable third-party (server) that is not involved in the multiparty computation. The technique of assuming a trustable third party or server was included in methods such as those proposed in [1][15], where the assumptions of a trustable server contributes to realizing a more effective process. Therefore, the establishment of a trustable server is effective in practical use. However, as shown in Section III, a set of shares of secret 1 does not depend on the secret inputs and is easily realizable by producing k random numbers, multiplying all the random numbers, and distributing them using a secret sharing scheme. In addition, in this study, we assumed a passive adversary. Therefore, if we add the process of producing a set of shares on 1 from different random numbers into all servers and execute the "shuffle process" shown in [18], the connection between the server that produced the set of shares and the set of shares is removed. Moreover, in a server set containing multiple servers, if they share no interest between each other, we can achieve a structure close to a trustable third party by utilizing these servers to exchange, mix, and remove while they shuffle the set of shares on 1 between each other. Therefore, although the avoidance of the above problem is our next task, since there are many means by which this condition can be realized, we can state that it is very possible to realize it in practice.

(3) In secrecy computation involving consecutive computation, the position of shares in a set of shares that are handled by each server is fixed.

Because our proposed method assumes a passive adversary, we can realize this condition by setting a regulation for servers that hold shares required for secrecy computation and servers involved in secrecy computation

5 Conclusion

In this study, with three proposed conditions, we realized a secure multiparty computation when $2k - 1 > n$ even when different types of computation are performed consecutively.

(1). The value of a secret and a random number used in secrecy multiplication does not include 0.

(2). There is a set of shares on 1 that is constructed from random numbers that are unknown to the adversary.

(3). In secrecy computation involving consecutive computation, the position of shares in a set of shares that are handled by each server is fixed.

In a future study, we will consider means of fulfilling all the aforementioned conditions.

References

1. Beaver D.: "Commodity-based cryptography." In Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing (STOC '97). ACM. El Paso, Texas, USA, pp. 445-455 (1997)
2. Ben-Or M., Goldwasser S., Wigderson A.: "Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation." In Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing (STOC '88). ACM, New York, NY, USA, pp. 1-10 (1988)
3. Bendlin R., Damgård I., Orlandi C., Zakarias S.: "Semi-homomorphic Encryption and Multiparty Computation." In Paterson K. G. (eds) Advances in Cryptology-EUROCRYPT 2011. LNCS, vol. 6632, pp. 169-188. Springer, Berlin, Heidelberg (2011)
4. Blakley G. R.: "Safeguarding Cryptographic Keys." In Proceedings of the AFIPS 1979 National Computer Conference, vol. 48, pp. 313-317 (1979)
5. Brakerski Z., Gentry C., Vaikuntanathan V.:"Leveled Fully Homomorphic Encryption without Bootstrapping." ITCS 2012, Mitzenmacher M. (ed), pp. 309-325, Cambridge, MA, USA, Jan. (2009)
6. Brakerski Z., Vaikuntanathan V.: "Fully Homomorphic Encryption from Ring-LWE and Security for Key Dependent Messages." In: Rogaway P. (eds) Advances in Cryptology – CRYPTO 2011. CRYPTO 2011. LNCS, vol 6841. Springer, Berlin, Heidelberg (2011)
7. Chaum D., Crépeau C., Damgård I.: "Multiparty Unconditionally Secure Protocols." In Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing (STOC '88). ACM, New York, NY, USA, pp. 11-19 (1988)

8. Cleve R.: “Limits on the Security of Coin Flips When Half the Processors are Faulty.” In 18th Annual ACM Symposium on Theory of Computing (STOC ’86), pp. 364-369. ACM Press (1986)
9. Cramer R., Damgård I., Maurer U.: “General Secure Multi-Party Computation from any Linear Secret-Sharing Scheme.” In Preneel B. (eds) Advances in Cryptology-EUROCRYPT 2000. LNCS, vol 1807, pp. 316-334. Springer, Berlin, Heidelberg (2000)
10. Damgård I., Ishai Y., Krøigaard M.: “Perfectly Secure Multiparty Computation and the Computational Overhead of Cryptography.” In Gilbert H. (eds) Advances in Cryptology-EUROCRYPT 2010. LNCS, vol. 6110, pp. 445-465. Springer, Berlin, Heidelberg (2010)
11. Damgård I., Pastro V., Smart N., Zakarias S.: “Multiparty Computation from Somewhat Homomorphic Encryption.” In Safavi-Naini R., Canetti R., (eds) Advances in Cryptology-CRYPTO 2012. LNCS, vol 7417, pp. 643-662. Springer, Berlin, Heidelberg (2012)
12. Damgård I., Keller M., Larraia E., Pastro V., Scholl P., Smart N.P.: “Practical Covertly Secure MPC for Dishonest Majority – Or: Breaking the SPDZ Limits.” In: Crampton J., Jajodia S., Mayes K. (eds) Computer Security – ESORICS 2013. ESORICS 2013. Lecture Notes in Computer Science, vol. 8134. Springer, Berlin, Heidelberg (2013)
13. Gennaro R., Rabin M. O., Rabin T.: “Simplified VSS and Fast-Track Multiparty Computations with Applications to Threshold Cryptography.” In Proceedings of the Seventeenth Annual ACM Symposium on Principles of Distributed Computing (PODC ’98). ACM, New York, NY, USA, pp. 101-111 (1998)
14. Gentry C.: A Fully Homomorphic Encryption Scheme, Ph.D Thesis, Stanford University, Stanford, CA, USA, Sept 2009
15. Hamada K., Kikuchi R.: “Commodity-Based Secure Multi-Party Computation.” Computer Security Symposium 2015 (CSEC2015), pp. 995-1002 (2015) (In Japanese)
16. Shamir A.: “How to Share a Secret.” Communications of the ACM, 22, (11), pp. 612-613, (1979)
17. Shingu T., Iwamura K., Kaneda K.: “Secrecy Computation without Changing Polynomial Degree in Shamir’s (k, n) Secret Sharing Scheme.” In Proceedings of the 13th International Joint Conference on e-Business and Telecommunications - Volume 1: DCNET, pp. 89-94 (2016)
18. Tsujishita K., Iwamura K.: “Application of Password Protected Secret Sharing Scheme to Searchable Encryption.” 77th Computer Security Group (Special Interest Groups) (CSEC77) (2017) (In Japanese)
19. van Dijk M., Gentry C., Halevi S., Vaikuntanathan V.: “Fully Homomorphic Encryption over the Integers.” EUROCRYPT 2010, Gilbert H. (ed). Lecture Notes Computer Science, vol. 6110, pp. 24-43, Nice, France (2010)
20. Watanabe T., Iwamura K., Kaneda K.: “Secrecy Multiplication Based on a (k, n)-Threshold Secret-Sharing Scheme Using Only k Servers.” In Park J., Stojmenovic I., Jeong H., Yi G. (eds) Computer Science and Its Applications. Lecture Notes in Electrical Engineering, vol. 330, pp. 107-112. Springer, Berlin, Heidelberg (2015)
21. Yao A. C.: “Protocols for Secure Computations.” *23rd Annual Symposium on Foundations of Computer Science (SFCS 1982)*, Chicago, IL, USA, 1982, pp. 160-164 (1982)