Conditionally Secure Multiparty Computation using Secret Sharing Scheme for $n < 2k - 1$ (short paper)

Ahmad Akmal Aminuddin Mohd Kamal
Tokyo University of Science
Department of Electrical Engineering
Tokyo, Japan
ahmad@sec.ee.kagu.tus.ac.jp

Keiichi Iwamura
Tokyo University of Science
Department of Electrical Engineering
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 $2k - 1$. 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, multiparty computation, MPC, secret sharing scheme, secure computation, product-sum operation, $n < 2k-1$.

I. 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 using secret sharing schemes. 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 [10].

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 [11].

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 previous works in the full version [9].

II. PREVIOUS WORK

A. TUS method

Shingu et al. proposed a secure multiparty computation called TUS method [11]. TUS method utilizes secret sharing scheme to perform multiparty computation and is secure against a passive adversary under the setting $n < 2k - 1$. TUS method achieve the setting of $n < 2k - 1$ even when performing secrecy multiplication using the approach of $scalar\ value \times polynomial$ to prevent an increase in the polynomial degree, provided that the secret input and the random numbers for protocol multiplication do not include 0.

The TUS method has been proven to be information-theoretic secure against Adversary 1, who has access to information from $k - 1$ servers, Adversary 2, who has information on an input in addition to information from $k - 1$ servers, and Adversary 3, who has information on the output in addition to information from $k - 1$ servers. However, all 2-inputs-1-output computation is not secure against Adversary 4, who has information on an input and an output in addition to information from $k - 1$ servers. Therefore, Adversary 4 is not considered in 2-inputs-1-output computation of the TUS method.

B. Problem of the TUS method

The problem with the TUS method is when combination of different computations (secrecy multiplication and addition) is used to compute a product-sum operation of $ab + c$. Since the

protocol for the secrecy product-sum computation is a 3-input-1-output operation, 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.

TUS method of product-sum operation is not secure against Adversary 4 when the adversary 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. However, a product-sum operation using the TUS method remains information-theoretic secure against Adversaries 1–3. We refer to the full version for details.

III. 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. 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 $(j = 0, 1, \dots, k-1)$ 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 $(\mathrm{i} = 0, \dots, n-1)$ 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.

A. Secrecy product-sum computation

Every server holds a set of shares of secret input through the distribution protocol. 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 created from random numbers not known to the adversary . Here, the set of shares is

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

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.

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

Protocol for Product-Sum Operation of $ab + c$

- 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

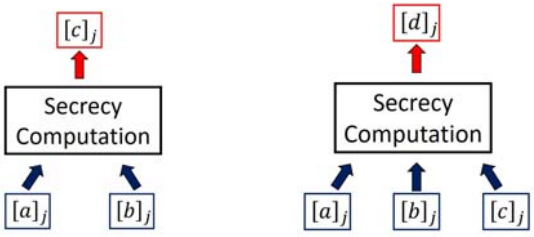
servers S_i by using Shamir's (k, n) threshold secret sharing scheme.

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.

B. Security of single product-sum computation

The TUS proposed method is a 2-input-1-output operation and can be represented by Fig. 1(a), 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, Adversary 2, and Adversary 3.



(a) 2-input-1-output computation (TUS method) (b) 3-input-1-output computation (Proposed method)

Fig. 1 Block diagram of secrecy computation

Since our proposed secrecy computation method is a 3-input-1-ouput computation, it can be represented by Fig. 1(b), 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(a). 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 $(i = 0, \ldots, k - 1)$, and in Steps 1–2, he/she learns about αa, in Steps 4–5 about $\alpha\beta ab, \lambda c$, in Step 8 about $\alpha_j, \beta_j, \lambda_j, \delta_j, \eta_j, \gamma_j$ $(j = 0, 1, \ldots, k - 2)$, 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 simplified into the problem of determining whether he/she can learn about the remaining inputs a, c from the information about $b, \beta, \alpha a, \lambda c, \gamma, \alpha\delta, \lambda\eta, ab + c, \beta_i, \alpha_j, \lambda_j, \delta_j, \eta_j (j = 0, 1, \ldots, k - 2)$.

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,

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

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.

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

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

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

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 simplified into the problem of determining whether the adversary can learn about the remaining input a or output $ab + c$ from the information of $b, \beta, c, \lambda, \alpha a, \gamma/\alpha\delta, \gamma/\eta, \alpha_j, \delta_j, \eta_j, \gamma_j, \gamma(ab + c)_j \ (j = 0, 1, \dots, k-2)$

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 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,

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

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.

C. Security of combination of multiple product-sum computation

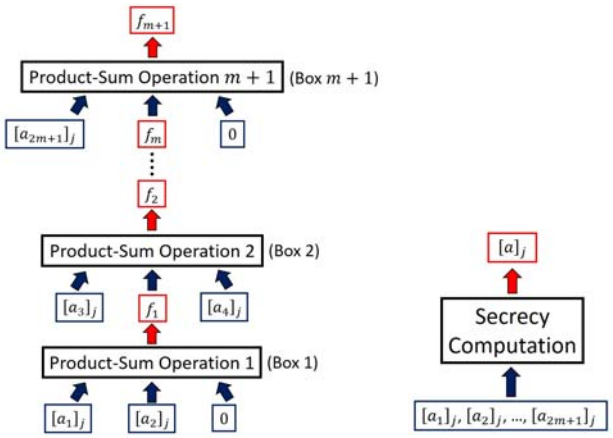
In general, any computation that comprises the four basic operations (addition, subtraction, multiplication, and division) can be decomposed into a combination of multiple product-sum operations. For example, the computation of $a = f(a_1, a_2, \dots, a_{2m}, a_{2m+1}) = a_{2m+1}(a_1a_2 + a_3a_4 + \cdots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

$$\text{Product-sum operation } 1 : f_1 = f(a_1, a_2, 0) = a_1a_2$$
$$\text{Product-sum operation } 2 : f_2 = f(a_3, a_4, f_1) = a_3a_4 + a_1a_2$$
$$\vdots$$
$$\text{Product-sum operation } m : f_m = f(a_{2m-1}, a_{2m}, f_{m-1}) = a_1a_2 + a_3a_4 + \cdots + a_{2m-1}a_{2m}$$
$$\text{Product-sum operation } m+1 : f_{m+1} = f(a_{2m+1}, f_m, 0) = a_{2m+1}(a_1a_2 + a_3a_4 + \cdots + a_{2m-1}a_{2m}) = a$$

By combining the basic product-sum operation boxes shown in Fig. 1(b), the above computation can be represented as in Fig. 2(a). 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. 2(b); however, if we were to decompose it into a basic product-sum operation, we could state that the secrecy computation box in Fig. 2(b) is composed of the operations in Fig. 2(a).

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. 2(b), 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.



(a) Combination of multiple product-sum operation for computation of a (b) General computation of a

Fig. 2 Block diagram for 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.

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 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 α_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. 2(a).

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. 2(a), 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. 2(a). 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.

IV. 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. 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 does not raise a serious problem. In addition, this condition 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][8], where the assumptions 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 [12], 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, 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.

V. 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.

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] 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)

[3] 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)

[4] 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)

[5] 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)

[6] 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)

[7] Gentry C.: A Fully Homomorphic Encryption Scheme, Ph.D Thesis, Stanford University, Stanford, CA, USA, Sept 2009

[8] Hamada K., Kikuchi R.: "Commodity-Based Secure Multi-Party Computation." Computer Security Symposium 2015 (CSEC2015), pp. 995-1002 (2015) (In Japanese)

[9] Mohd Kamal A. A. A., Iwamura K.: "Conditionally Secure Secrecy Computation using Secret Sharing Scheme for $n < 2k - 1$ (full paper)." Cryptology ePrint Archive, Report 2017/718, http://eprint.iacr.org/. (2017)

[10] Shamir A.: "How to Share a Secret." Communications of the ACM, 22, (11), pp. 612-613, (1979)

[11] 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)

[12] 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)