Fast Secrecy Computation with Multiplication Under the Setting of k≤N<2k-1 using Secret Sharing Scheme

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

[1] Tokyo University of Science, 6-3-1 Niijuku, Katsushika-ku, Tokyo 125-8585 JAPAN
iwamura@ee.kagu.tus.ac.jp

[2] Tokyo University of Science, 6-3-1 Niijuku, Katsushika-ku, Tokyo 125-8585 JAPAN
ahmad@sec.ee.kagu.tus.ac.jp

Abstract. In this paper, we describe two new protocols for secure secrecy computation with information theoretical security against the semi-honest adversary and the dishonest majority. Typically, unconditionally secure secrecy computation using the secret sharing scheme is considered impossible under the setting of $\boldsymbol{n < 2k - 1}$. Therefore, in our previous work, we first took the approach of finding conditions required for secure secrecy computation under the setting of $\boldsymbol{n < 2k - 1}$ and realized a new technique of conditionally secure secrecy computation. We showed that secrecy computation using a secret sharing scheme can be realized with a semi-honest adversary with the following three preconditions: (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; and (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 this paper, we differentiate the relationship between the parameter $\boldsymbol{n}$ of $(\boldsymbol{k}, \boldsymbol{n})$-threshold secret sharing scheme and $\boldsymbol{N}$ of the number of servers/parties, and realize secrecy computation of multiplication under the setting of $\boldsymbol{k \leq N < 2k - 1}$. In addition, we improve the processing speed of our protocol by dividing the computation process into a *Preprocessing Phase* and a *Computation Phase* and shifting the cost for communication to the *Preprocessing Phase*. This technique allows for *information* that does not depend on any of the private values, to be generated in advance and significantly reduce the cost of communication in the *Computation Phase*. For example, for secrecy computation without repetition, the cost for communication can be totally removed in the *Computation Phase*. As a result, we realize the method for secrecy computation that is faster compared to conventional methods. In addition, our protocols provided solutions for the aforementioned three preconditions and realize secure secrecy computation without any limitation in terms of usability.

Keywords: secrecy computation, multiparty computation, secret sharing, n<2k-1, information-theoretically secure, fast computation

1 Introduction

1.1 Background

In recent years, with the improvement of big data and the internet of things (IoT), there is high anticipation regarding technology that could make use of an individual's information. However, there is still concern among individuals about the privacy, security, and confidentiality of their information. Therefore, as a measure to solve this problem, there is a need for technology that allows for their information to be used without revealing anything about their privacy. One of the available technologies that could permit this is called secrecy computation or multiparty computation. In the setting of secure secrecy computation, a set of parties with private inputs wish to compute a joint function of their inputs, without revealing anything but the output. Protocols for secure secrecy computation guarantee privacy (meaning that the protocol reveals nothing but the output), correctness (meaning that the correct function is computed), and more. These security guarantees are provided in the presence of two types of adversary models: semi-honest (where the adversary follows the protocol specification but may try to learn more than allowed from the protocol) and malicious (where the adversary can run any arbitrary polynomial-time attack strategy). In the information theoretic model, security is obtained even in the presence of computationally unbounded adversaries. In contrast, in the computational model, security is obtained in the presence of polynomial-time adversaries and relies on cryptographic hardness assumptions.

There are three main approaches for constructing secure secrecy computation protocols:

- Secret sharing approach [1, 5, 9, 11, 12, 15, 17, 20, 22]
- Homomorphic encryption approach [4, 7, 8, 13, 14, 21]
- Garbled circuit approach [3, 24].

However, homomorphic encryption is known to be costly in term of computational cost. Therefore, approaches with lower computational cost are preferable to homomorphic encryption when considering utilization in a cloud system. On one hand, the garbled circuit approach yields protocols with a constant number of rounds, therefore, outperforming secret sharing based protocols, which have a number of rounds that is linear in the depth of the circuit being computed in high-latency networks. On the other hand, protocols based on secret sharing typically have low bandwidth, and given that bandwidth is often a bottleneck, it follows that protocols with low communication have the potential to achieve much higher throughput. Note that the secret sharing approach typically relies on simple operations making them fast. However, as far as we know, no protocols achieve low communication while realizing information theoretical security under the setting of $n < 2k - 1$. From now on, $n < 2k - 1$ means $k \leq n < 2k - 1$.

1.2 Our Results

Our approach to achieving an information theoretical secure secrecy computation was using the secret sharing scheme. The secret sharing scheme is a method in the field of cryptography for data encryption, in which a single data is divided into multiple shares, which are then distributed to multiple users. A known example of a secret sharing scheme is Shamir's (k, n) threshold secret sharing scheme [18]. In this scheme, a secret s is divided into n number of shares. The original secret s could only be reconstructed or retrieved from a threshold k number of shares. That is, any $k - 1$ or smaller number of shares reveals nothing about the original secret. However, there are a few major challenges. Secure 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 $2k - 1$. This problem was solved by Shingu et al. through the TUS1 method [20], where 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, the TUS1 method introduced another problem: 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. This problem was solved by Mohd Kamal et al. through the TUS2 method [17], by introducing the concept of conditionally secure secrecy computation. Typically, unconditionally secure secrecy computation is considered impossible under the setting of $n < 2k - 1$. In contrast, this means that secure secrecy computation using a secret sharing scheme is possible with certain conditions. Therefore, Mohd Kamal et al. proposed three conditions as shown below, and realized a new method to solve the problem of the TUS1 method. Note that both methods share a common point where the secret information is first encrypted with a random number and is then used in the actual secrecy computation.

- Condition 1: The value of a secret and a random number used in secrecy multiplication does not include 0.
- Condition 2: There is a set of shares on 1 that is constructed from random numbers that are unknown to the adversary.
- Condition 3: In secrecy computation involving consecutive computation, the position of shares in a set of shares that are handled by each server is fixed.

However, introduction of the aforementioned conditions limits the usability of the TUS2 method. In addition, the TUS2 method requires a large computational cost, thus making it slow.

Therefore, in this paper, we propose protocols to solve these three conditions and realize secure secrecy computation without any limitation in terms of usability. Furthermore, our protocol significantly reduces the communication cost compared to conventional methods and realizes a faster secrecy computation.

New protocols

Below, we describe 2 new secrecy computation protocols.

New Protocol 1:

In the conventional method of secrecy computation using a secret sharing scheme, to restore the multiplication result, the number of shares required increases from k to $2k-1$. Therefore, the number of servers/parties (parameter n) needs to be more than $2k-1$. Due to this, unconditionally secure secrecy computation is considered impossible under the setting of $n < 2k-1$. In our new protocol 1, we differentiate the relationship between the parameter n required in the (k,n)-threshold secret sharing scheme and parameter N, which is defined as the number of servers/parties participating in the computation, realizing a secrecy computation using a secret sharing scheme with the parameters of $k \leq N < 2k-1, n \geq 2k-1$. In other words, the number of servers/parties remained at $k \leq N < 2k-1$, but secrecy computation is realized under $n \geq 2k-1$. By using this technique, we were able to produce the set of shares on 1, proposed in Condition 2 of the TUS2 method. Therefore, our protocol realizes a secure secrecy computation of multiplication using a secret sharing scheme while maintaining the number of servers/parties at less than $2k-1$. From now on, $N < 2k-1$ also means $k \leq N < 2k-1$.

New Protocol 2:

In order to prevent the degree of a polynomial changing from $k-1$ to $2k-2$ for each multiplication between polynomials, secrecy computation proposed thus far required extra cost for both communication and computation. In our new protocol 2, we divide the computation process into a ***Preprocessing Phase*** and a ***Computation Phase***. The new techniques used here allow us to shift parts of the communication cost to the ***Preprocessing Phase***, where raw material that does not depend on any of the private values can be generated at convenience. As we shall see later, this can be used to significantly reduce the communication cost in the ***Computation Phase***, and significantly speed up the ***Computation Phase***. For example, when performing computation with repetition, communication between the servers/parties are required to reconstruct the result before being used as input for the consecutive multiplication. In contrast, when performing computation without repetition, because no reconstruction of result is required, the ***Computation Phase*** could be computed with zero communication cost.

Security

We show that our protocols are secure in the presence of semi-honest adversaries with at most $k-1$ corrupted servers, under the standard information entropy-based definitions. The basis of our protocol is information theoretic. In addition to the above, our proposed model of secrecy computation is based on a *client/server model* where any number of clients send shares of their inputs to n servers that carry out the computation for the clients and return the results to them, without learning anything. This model is widely used nowadays and is the business model used in Sharemind [19]. Our protocols are designed to be flexible with a dishonest majority, any number of n parties and, at most, $k-1$ corrupted parties. This is unlike Araki et al.'s method [1] that works for only a fixed number of 3 parties, with at most one corrupted party.

The construction of this paper is as follows: in Chapter 2, we present related works, in Chapter 3 we explain the basic building blocks for our protocol, in Chapter 4 we present our first new protocol and solution for Conditions 2 of the TUS2 method, and in Chapter 5, we present our second new protocol and solution for Condition 1 of the TUS2 method. In addition, in Chapter 6, we discuss the solution for Condition 3 and the features of our protocol. Finally, in Chapter 7, we compare our protocol with conventional works and show that our protocol can achieve a faster computation speed, especially when performing computation without repetition.

2 Related Works

Most of the work on concretely efficient secure secrecy computation has focused on the dishonest majority case. For example, the SPDZ 2 protocol [13] introduced a computationally secure multiparty computation with dishonest majority through implementation of homomorphic encryption. In the setting of an honest majority, to the best of our knowledge, the method by Araki et al. [1] is the only highly efficient protocol with security for semi-honest adversaries. We will present a detailed comparison of our protocol in Section 7.

2.1 SPDZ 2 Method by Damgård et al. [13]

Damgård et al. proposed a secure multiparty computation called SPDZ 2 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 [2]. To perform secrecy multiplication, shares of random numbers $\langle a\rangle, \langle b\rangle, \langle c\rangle$, called a multiplicative triple, which satisfy $a \cdot b = c$, are used.

In SPDZ 2, for example, the process where the secret information of x is reconstructed from its shares $\langle x\rangle$ is denoted as $x = open(\langle x\rangle)$. The protocol for multiplication of $x \cdot y$ proposed by SPDZ 2 is shown below. However, the construction of a multiplication triple requires fully homomorphic encryption (FHE) [14], where the computation cost is expensive, thus increasing the overall process time significantly.

1. Prepare the multiplication triple $\langle a\rangle, \langle b\rangle, \langle c\rangle$ (***Offline Phase***).
2. Compute shares $\langle x\rangle, \langle y\rangle$ on secret x, y (***Distribution Phase***).
3. Each server reconstructs $d = open(\langle x\rangle - \langle a\rangle), e = open(\langle y\rangle - \langle b\rangle)$ and computes $\langle x \cdot y\rangle = d \cdot e + e \cdot \langle a\rangle + d \cdot \langle b\rangle + \langle c\rangle$ (***Online Phase***).

2.2 Araki et al.'s Method [1]

Typically, in a secure secrecy computation, the cost for communication between servers could affect the overall processing speed more than the actual cost of computation. Therefore, Araki et al. proposed a very fast method of secrecy computation under the parameters $n = 3, k = 2$, that require only 1 times of communication per multiplication. The detail protocol is shown below. Note that it is usually not considered a problem, even if communications are required in the ***Preprocessing Phase***. In addition, secrecy computation of addition is performed locally, where the shares are added together.

Preprocessing Phase: Generation of correlated randomness

1. Players P_1, P_2, P_3 generate and hold $\beta_1, \beta_2, \beta_3 \in Z_{2^n}$, where $\beta_1 + \beta_2 + \beta_3 = 0$.

Computation Phase:

Distribution

1. Dealer D chooses a random number $x_1, x_2, x_3 \in Z_{2^n}$, where $x_1 + x_2 + x_3 = 0$.
2. Dealer D sends a share (x_i, a_i) of the secret v_1 to players P_i. a_i is computed as $a_i = x_{i-1} - v_1$ $(i = 1,2,3)$.
3. Dealer D performs the same process on secret v_2, producing share (y_i, b_i) for players P_i. Note that $b_i = y_{i-1} - v_2, y_1 + y_2 + y_3 = 0$.

Multiplication

1. Players P_i compute $r_i = (a_i b_i - x_i y_i + \beta_i)/3$ and send to players P_{i+1}.
2. Players P_i compute $z_i = r_{i-1} - r_i$, $c_i = -2r_{i-1} - r_i$ and hold (z_i, c_i) as a share on the result of multiplication of $v_1 v_2$.

Reconstruction

1. From information z_i, c_i, z_j, c_j of Player P_i and Player P_j, the result of multiplication $v_1 v_2$ can be computed using the equation shown below. Note that $c_i = -2r_{i-1} - r_i = z_{i-1} - v_1 v_2$.

$$z_j - c_i = v_1 v_2$$

3 Building Blocks and Sub-Protocols

3.1 (k, n) threshold secret-sharing scheme

A secret sharing scheme that satisfies both the conditions stated below is known as the (k, n) threshold secret-sharing scheme.

- Any $k - 1$, or less, number of shares will reveal nothing about the original secret information s.

- Any k and above number of shares will allow for the reconstruction of the original secret information s.

The classic methods of the (k, n) threshold secret sharing scheme are Shamir's (k, n) threshold secret sharing scheme [18](Shamir's method) and the XOR-based method for sharing and reconstruction of secret information proposed by Kurihara et al. [16] (XOR method). In our protocol, unless stated otherwise, Shamir's method is used, and all computations are performed in modulus p. In addition, shares of secret information, s, are represented by $\overline{[s]}_i$

3.2 Ben-Or's Method [5]

Multiplication of the k-degree polynomial $f(x), g(x)$ will result in $h(x) = f(x) \times g(x)$ with a degree of $2k$. Therefore, at least $2k + 1$ number of shares are required for reconstructing $h(x)$. Here, Ben-Or proposed a method of reducing the degree of the polynomial of $h(x)$ from $2k$ to k, where each coefficient of $h(x)$ is randomized.

However, this method does not change the limitation where the required number of shares n for reconstruction must be $n \geq 2k + 1$. In this method, by using matrix A shown below, $R = W \cdot A$ is computed in regard to a vector of shares $W = (W_0, W_1, \dots, W_{2k-1})$ with a degree of $2k$, producing a vector of shares $R = (R_0, R_1, \dots, R_{2k-1})$ with a degree of k.

$$A = \begin{bmatrix} a_{0,0} & \cdots & a_{0,k-1} \\ \vdots & \ddots & \vdots \\ a_{2k-1,0} & \cdots & a_{2k-1,2k-1} \end{bmatrix} \quad (1)$$

3.3 The TUS Methods

First, Shingu et al. proposed a 2-inputs-1-output computation named the TUS1 method [20], where 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, the TUS1 method introduced another problem: 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. Therefore, condition where computation involving a combination of addition/subtraction and multiplication/division is not performed is needed, in addition to the existing condition where the input of the secret does not include the value 0. Therefore, the TUS1 method could realize a very effective specific computation such as a computation of RSA encryption. However, it is not capable of coping with computation that requires a combination of addition/subtraction and multiplication/division.

Next, Mohd Kamal et al. introduced an improved method called the TUS 2 method, where computation that involves a combination of addition/subtraction and multiplication/division can also be performed securely [17]. This method was proven to be secure under the aforementioned three conditions. In addition, it was shown that this method is secure against computation that involves a combination of product-sum operation. Therefore, this method could realize any arithmetic computation under the setting of $n < 2k - 1$. However, the TUS 2 method requires much more computational cost compared to the conventional method in $n \geq 2k - 1$, so it is not the most efficient method.

Because of this, Iwamura et al. proposed an improved version of the TUS 2 method, known as the TUS 3 method, where XOR based secret sharing [16] is introduced and realizes a more efficient method of secrecy computation [15]. Out of the aforementioned three conditions, the TUS 3 method proposed a way to ease one of the conditions (known as the TUS3' method); however, the three conditions still remain. Note that all TUS methods share a common point where the secret information is first encrypted with a random number, and is then used in the secrecy computation using a secret sharing scheme.

4 Proposed Method 1

4.1 Secrecy Multiplication for $N < 2k - 1$ and Solution for Condition 2

In the TUS method, it is proposed that in Condition 2 there is a set of shares on 1 that is constructed from random numbers that are unknown to the adversary, as defined below.

$$[\varepsilon]_i = \left(\overline{[\varepsilon]_i}, \overline{[\varepsilon_0]_i}, \dots, \overline{[\varepsilon_{k-1}]_i}\right) \quad (2)$$

Here, $\varepsilon = \prod_{j=0}^{k-1} \varepsilon_j$ is defined as a random number unknown to the adversary. However, in order to produce a set of shares on 1, shown above, multiplication $\varepsilon = \prod_{j=0}^{k-1} \varepsilon_j$ of random numbers ε_j $(j = 0, 1, \dots, k - 1)$ is required. However, it is known that unconditionally secure multiparty computation is considered impossible under the setting of $n < 2k - 1$. Therefore, our protocol introduces the approach of differentiating the relationship between parameter N, which is the number of servers/players that is actually needed, and parameter n of secret sharing scheme, and realizing computation under the parameter setting of $N < 2k - 1, n \geq 2k - 1$. Because of this, we could extend the work of Ben-Or where $N = n \geq 2k - 1$ is assumed, into a protocol that can be realized even when $N < 2k - 1, n \geq 2k - 1$.

Below, where the number of participants is u, we show the method of multiplication and construction of a set of shares on 1 under the setting of $N < 2k - 1$. However, for ease of understanding, we explain the algorithm using $u = 3, N = k$. In the case where $u > 3$, Player $P_i (i = 3, \dots, u - 1)$ performs the same process of Step 3 as Player P_2, and broadcasts each encrypted secret information of λ_i which is equivalent to $\gamma\lambda_2$. From Step 4 onwards, all encrypted information by Player P_i is treated the same as $\gamma\lambda_2$, and random numbers used to encrypt the secret information, which was distributed to every server, are also treated the same way as random numbers γ_i. However, we assumed that λ_i does not include 0. In addition, in the case of $u = 2$, secrecy multiplication by players P_0, P_1 is realized.

In this section, for ease of understanding, $\overline{[a]}_i^{(k)}$ is defined as shares of a where the number of shares required for reconstructing a is k. Below, $a_{i,j}$ are elements of matrix A of Ben-Or's method, and 0_i, $0'_i$ are used in order to differentiate between the shares of 0 that had been distributed by different polynomials. In the protocol below, random numbers are chosen from uniformly distributed random numbers and do not include the value of 0. In addition, all values are chosen from finite field $GF(p)$, and all computations including the secret sharing are performed in modulus p. We also presume there exists a secure communication between the players and servers.

For ease of understanding, we include the computation for $N = k = 3$ from Step 4 to Step 8 in Appendix 1.

***Protocol 4.1: Secrecy multiplication for N* $< 2k - 1$**

1. Player P_0 generates random numbers $\alpha_0, \dots, \alpha_{k-1}$, computes $\alpha = \prod_{j=0}^{k-1} \alpha_j$ and distributes secret information λ_0 using $(k, 2k)$ Shamir's method. Then, computes $\overline{[\alpha\lambda_0]}_i^{(k)} = \alpha \times \overline{[\lambda_0]}_i^{(k)}, \overline{[\alpha\lambda_0]}_{i+k}^{(k)} = \alpha \times \overline{[\lambda_0]}_i^{(k)} (i = 0, \dots, k-1)$ and sends $\overline{[\alpha\lambda_0]}_i^{(k)}, \overline{[\alpha\lambda_0]}_{i+k}^{(k)}, \alpha_i$ to servers S_i.
2. Player P_1 generates random numbers $\beta_0, \dots, \beta_{k-1}$, computes $\beta = \prod_{j=0}^{k-1} \beta_j$ and distributes secret information λ_1 using $(k, 2k)$ Shamir's method. Then, computes $\overline{[\beta\lambda_1]}_i^{(k)} = \beta \times \overline{[\lambda_1]}_i^{(k)}, \overline{[\beta\lambda_1]}_{i+k}^{(k)} = \beta \times \overline{[\lambda_1]}_{i+k}^{(k)} (i = 0, \dots, k-1)$ and sends $\overline{[\beta\lambda_1]}_i^{(k)}, \overline{[\beta\lambda_1]}_{i+k}^{(k)}, \beta_i$ to servers S_i.
3. Player P_2 generates random numbers $\gamma_0, \dots, \gamma_{k-1}$, computes $\gamma = \prod_{j=0}^{k-1} \gamma_j$ and $\gamma\lambda_2$ using secret information λ_2, sends $\gamma\lambda_2$ to all servers and γ_i to servers S_i.
4. Servers S_i compute $\alpha_i\beta_i\gamma_i$, distribute 0_i using $(2k, 2k)$ Shamir's method, compute the following and send $\overline{[\alpha_{-\iota}\beta_{-\iota}\gamma_{-\iota}\lambda_0\lambda_1\lambda_2]'}_i^{(2k)}, \overline{[\alpha_{-\iota}\beta_{-\iota}\gamma_{-\iota}\lambda_0\lambda_1\lambda_2]'}_{i+k}^{(2k)}$ and shares of 0_i (except for $\overline{[0_\iota]}_i^{(2k)}, \overline{[0_\iota]}_{i+k}^{(2k)}$) to server $S_{i+1(modk)}$. However, α that does not include α_i is denoted as α_{-i}.

$$\overline{[\alpha\beta\gamma\lambda_0\lambda_1\gamma\lambda_2]}_i^{(2k-1)} = \overline{[\alpha\lambda_0]}_i^{(k)} \times \overline{[\beta\lambda_1]}_i^{(k)} \times \gamma\lambda_2$$

$$\overline{[\alpha\beta\gamma\lambda_0\lambda_1\gamma\lambda_2]}_{i+k}^{(2k-1)} = \overline{[\alpha\lambda_0]}_{i+k}^{(k)} \times \overline{[\beta\lambda_1]}_{i+k}^{(k)} \times \gamma\lambda_2$$

$$\overline{[\alpha_{-\iota}\beta_{-\iota}\gamma_{-\iota}\lambda_0\lambda_1\lambda_2]'}_i^{(2k)} = \frac{\overline{[\alpha\beta\gamma\lambda_0\lambda_1\gamma\lambda_2]}_i^{(2k-1)}}{\alpha_i\beta_i\gamma_i} + \overline{[0_0]}_i^{(2k)}$$

$$\overline{[\alpha_{-\iota}\beta_{-\iota}\gamma_{-\iota}\lambda_0\lambda_1\lambda_2]'}_{i+k}^{(2k)} = \frac{\overline{[\alpha\beta\gamma\lambda_0\lambda_1\gamma\lambda_2]}_{i+k}^{(2k-1)}}{\alpha_i\beta_i\gamma_i} + \overline{[0_0]}_{i+k}^{(2k)}$$

5. Servers $S_i (i = 0, \dots, k-1)$ distribute $0'_i$ using $(2k, 2k)$ Shamir's method, perform the following process on round r $(r = 1, \dots, k-1)$ and send the result to servers $S_{i+1(modk)}$. However, distribution of $0'_i$ is different for each round and j corresponds to the received $\overline{[0_{\iota-1}]}_j^{(2k)}$. Note that the subscript of the shares is in $mod2k$. This process is repeated until $\alpha\beta\gamma$ is cleared.

$$\overline{[\alpha_{-(\iota-r,\dots,\iota)}\beta_{-(\iota-r,\dots,\iota)}\gamma_{-(\iota-r,\dots,\iota)}\lambda_0\lambda_1\lambda_2]}'^{(2k)}_{i-r} = \frac{\overline{[\alpha_{-(\iota-r,\dots,\iota-1)}\beta_{-(\iota-r,\dots,\iota-1)}\gamma_{-(\iota-r,\dots,\iota-1)}\lambda_0\lambda_1\lambda_2]}'^{(2k)}_{i-r}}{\alpha_i\beta_i\gamma_i} + \overline{[0'_\iota]}^{(2k)}_{i-r}$$

$$\overline{[\alpha_{-(\iota-r,\dots,\iota)}\beta_{-(\iota-r,\dots,\iota)}\gamma_{-(\iota-r,\dots,\iota)}\lambda_0\lambda_1\lambda_2]}'^{(2k)}_{i-r+k} = \frac{\overline{[\alpha_{-(\iota-r,\dots,\iota-1)}\beta_{-(\iota-r,\dots,\iota-1)}\gamma_{-(\iota-r,\dots,\iota-1)}\lambda_0\lambda_1\lambda_2]}'^{(2k)}_{i-r+k}}{\alpha_i\beta_i\gamma_i} + \overline{[0'_\iota]}^{(2k)}_{i-r}$$

$$\overline{[0_{\iota-r\dots\iota}]}^{(2k)}_j = \frac{\overline{[0_{\iota-r\dots\iota-1}]}^{(2k)}_j}{\alpha_i\beta_i\gamma_i} + \overline{[0'_\iota]}^{(2k)}_j$$

6. After the $(k-1)$-th round, servers S_i collect $\overline{[0_{\jmath\dots\jmath+k-1}]}^{(2k)}_{i+1}, \overline{[0_{\jmath\dots\jmath+k-1}]}^{(2k)}_{i+1+k}(j = 0,\dots,k-1)$ that correspond to its own $\overline{[\lambda_0\lambda_1\lambda_2]}'^{(2k)}_{i+1}, \overline{[\lambda_0\lambda_1\lambda_2]}'^{(2k)}_{i+1+k}$, add them together and produce $\overline{[\lambda_0\lambda_1\lambda_2]}"^{(2k)}_{i+1}, \overline{[\lambda_0\lambda_1\lambda_2]}"^{(2k)}_{i+1+k}$.
7. Servers S_i distribute $0"_i$ using (k, N) Shamir's method, compute the following for $j = 0,\dots,N-1$, and send $R_{i,j}$ to servers S_j.

$$R_{i,j} = a_{j,i+1} \times \overline{[\lambda_0\lambda_1\lambda_2]}"^{(2k)}_{i+1} + a_{j,i+1+k} \times \overline{[\lambda_0\lambda_1\lambda_2]}"^{(2k)}_{i+1+k} + \overline{[0"_\iota]}^{(k)}_j$$

8. Servers S_i compute $\overline{[\lambda]}^{(k)}_i = \sum_{j=0}^{N-1} R_{j,i}$, and obtain $\overline{[\lambda]}^{(k)}_i$. However, note that $\lambda = \lambda_0\lambda_1\lambda_2$.

When performing reconstruction of the multiplication result, the player only needs to collect k numbers of $\overline{[\lambda]}^{(k)}_i$ and reconstruct the result of λ. In addition, if k number of players distribute $\lambda_0,\dots,\lambda_{k-1}$ as random numbers $\varepsilon_0,\dots,\varepsilon_{k-1}$ in Steps 1 and 2, using the (k, N) XOR method, servers S_i can obtain set of shares on 1 as shown above.

In addition, in Step 7 of the above protocol, if $N \geq k$ and computation is performed until $R_{i,N-1}$ is computed and sent to N number of servers, the above protocol can also accommodate the case of $N \geq k$.

4.2 Security Analysis on Secrecy Multiplication for $k \leq N < 2k-1$

We assumed that there were u number of players. In this case, $\lambda_0,\dots,\lambda_{u-1}$ is secret information, and $\prod_{j=0}^{u-1}\lambda_j$ is the result of secrecy multiplication. Here, we assumed that the adversary had information from $k-1$ number of servers. First, we defined $u-2$ number of players and the player who reconstructed the result as *Adversary 1* and the adversary attempting to learn about the secret information of the remaining players. Next, we defined $u-1$ number of players as *Adversary 2* and the adversary attempting to learn about the secret information of the remaining players and the result of the secrecy computation.

Proof of Security against Adversary 1

We suppose that Players $P_0, \ldots, P_{u-3}$ and the player who reconstructed the result are *Adversary 1*. In addition, *Adversary 1* has information from servers $S_0, \ldots, S_{k-2}$. In this case, if *Adversary 1* managed to learn about secret information $\lambda_{u-2}, \lambda_{u-1}$ of the remaining players P_{u-2}, P_{u-1}, the attack would be considered as successful.

From Step 3, *Adversary 1* learns from players P_{u-2}, P_{u-1} about encrypted information $\omega\lambda_{u-2}, \zeta\lambda_{u-1}$ and random numbers $\omega_0, \ldots, \omega_{k-2}, \zeta_0, \ldots, \zeta_{k-2}$ which were used to construct random numbers ω, ζ. Therefore, *Adversary 1* also learns about random numbers $\omega_{k-1}\lambda_{u-2}$ and $\zeta_{k-1}\lambda_{u-1}$. However, because secret information $\lambda_{u-2}, \lambda_{u-1}$ does not consist of value 0, and *Adversary 1* has no information on $\omega_{k-1}, \zeta_{k-1}$, the adversary will not be able to identify information of $\lambda_{u-2}, \lambda_{u-1}$. In addition, in Step 4, *Adversary 1* learns about $\overline{[0_{k-1}]}_i^{(2k)}, \overline{[0_{k-1}]}_{i+k}^{(2k)} (i = 0, \ldots, k-2)$ from server S_{k-1}. Because the constant for these are the value 0, it can be solved if $2k-1$ number of shares are collected; however, because *Adversary 1* only knows $2k-2$ number of shares, it cannot be solved. From Step 5 and onwards, because shares of value 0 is added to all information that is being sent, even if the original value is known it cannot be solved. In addition, because *Adversary 1* has k numbers of $\overline{[\lambda]}_i^{(k)}$ in the reconstruction process, from the computation result of λ and information of $\lambda_0, \ldots, \lambda_{u-3}$ from $u-2$ number of players, the adversary will be able to learn about $\lambda_{u-2}\lambda_{u-1}$. However, it cannot be decomposed into $\lambda_{u-2}, \lambda_{u-1}$. Therefore, *Adversary 1* cannot learn about each individual information of $\lambda_{u-2}, \lambda_{u-1}$. The same can be said even if *Adversary 1* learns about other combinations of $k-1$ number of servers. In addition, it is also obvious that the same can be said even if the combination of players is changed. Therefore, we can state that the algorithm shown above is secure.

Proof of Security against Adversary 2

Suppose that the players $P_0, \ldots, P_{u-2}$ make up *Adversary 2* and the adversary also has information from servers $S_0, \ldots, S_{k-2}$. In this case, the attack will be considered successful if *Adversary 2* learns about input λ_{u-1} or the result of computation $\prod_{j=0}^{u-1} \lambda_j$. From Step 3, *Adversary 2* learns from player P_{u-1} about encrypted information $\zeta\lambda_{u-1}$ and random numbers $\zeta_0, \ldots, \zeta_{k-2}$ used to construct random number ζ. Therefore, *Adversary 2* also learns about $\zeta_{k-1}\lambda_{u-1}$. However, because secret information λ_{u-1} does not consist of value 0, and *Adversary 2* has no information of ζ_{k-1}, the adversary will not be able to learn about λ_{u-1}. In addition, From Step 4, *Adversary 2* has information about $\overline{[0_{k-1}]}_i^{(2k)}, \overline{[0_{k-1}]}_{i+k}^{(2k)} (i = 0, \ldots, k-2)$ from server S_{k-1}. Because the constant of these is the value 0, it can be solved by collecting $2k-1$ number of shares. However, because the adversary only has information about $2k-2$ number of shares, it cannot be solved. From Step 5 onwards, because shares of value 0 are added to all information that is being sent, even if the original value is known, it cannot be solved. Therefore, *Adversary 2* cannot learn about secret information λ_{u-1} of player P_{u-1} and the result of computation λ. The same can be said even if *Adversary 2* learns about information from other combinations of servers. The same is true even for other combinations of players. Therefore, we can state that the algorithm above is secure and the statements below are true.

$$\begin{aligned} H(\lambda_i) = H(\lambda_i | & information\ of\ u \\ & - 2\ number\ of\ players\ that\ have\ no\ information\ on\ \lambda_i \\ & + information\ of\ player\ who\ reconstructed\ the\ result \\ & + information\ from\ k-1\ number\ of\ servers) \end{aligned}$$

$$\begin{aligned} H(\lambda_i) = H(\lambda_i | & information\ of\ all\ players\ that\ have\ no\ information\ on\ \lambda_i \\ & +\ information\ from\ k-1\ number\ of\ servers) \end{aligned}$$

$$\begin{aligned} H(\lambda_0\lambda_1\lambda_2) = H(\lambda_0\lambda_1\lambda_2 | & information\ of\ u-1\ number\ of\ players \\ & + information\ from\ k-1\ number\ of\ servers) \end{aligned}$$

In regard to the set of shares on secret 1, k number of players are required to distribute $\varepsilon_0, \dots, \varepsilon_{k-1}$ using (k, N) XOR instead of $\lambda_0, \dots, \lambda_{k-1}$ in Steps 1, 2. This is considered to be independent of the secrecy multiplication, thus it is clear that even by adding this process it will still be secure.

4.3 Verification of the Result of Secrecy Multiplication

If any of the players or servers perform different processes other than as shown in Section 4.1, result $\overline{[\lambda]}_i$ will be computed but it will not be a correct result of multiplication. Here, we need to think about the verification of the result of secrecy multiplication generated by k number of servers from Section 4.1. In order to do that, we first assumed that each server held another different result of secrecy multiplication, namely $(\overline{[\mu_1]}_i, \mu_{1,i})$. Here, $(\overline{[\mu_1]}_i, \mu_{1,i})$ is also generated from the protocol shown in Section 4.1.

From above, the verification of $(\overline{[\lambda]}_i, \lambda_i)$ produced in Section 4.1 can be performed as follows:

Protocol 4.3: Verification of the Result of Secrecy Multiplication

1. Servers S_i or Player P_i compute $\lambda_i / \mu_{1,i}$ and broadcast it.
2. Servers S_i compute the following and broadcast $\overline{[0']}_i$.

$$\overline{[\lambda']}_i = \left(\frac{\lambda_0}{\mu_{1,0}}\right) \times \dots \times \left(\frac{\lambda_{k-1}}{\mu_{1,k-1}}\right) \times \overline{[\mu_1]}_i \quad (3)$$

$$\overline{[0']}_i = \overline{[\lambda']}_i - \overline{[\lambda]}_i \quad (4)$$

3. Servers S_i reconstruct $[0']_i$ and verify whether it it is equal to 0. If the reconstructed result is equal to 0, the result of secrecy multiplication $(\overline{[\lambda]}_i, \lambda_i)$ is correct; if the reconstructed result is not equal to 0, the result of secrecy multiplication has been corrupted

<u>Case 1</u>: We suppose a situation where one of the results of secrecy computation $([\mu_1]_i, \mu_{1,i})$, $([\lambda]_i, \lambda_i)$ is incorrect, without adversary.

In this case, either μ_1 or λ will not become $\mu_1 = \prod_{i=0}^{k-1} \mu_{1,i}$ or $\lambda = \prod_{i=0}^{k-1} \lambda_i$, respectively. Therefore, we suppose $\mu_1 = \rho_1(\prod_{i=0}^{k-1} \mu_{1,i})$ or $\lambda = \rho_0(\prod_{i=0}^{k-1} \lambda_i)$ $(\rho_0, \rho_1 \neq 1)$.

However, because one of these is correct, for example, if μ_1 is assumed to be $\mu_1 = \rho_1(\prod_{i=0}^{k-1} \mu_{1,i})$, from equation (3), $\overline{[\lambda']}_i$ will become $\overline{[\lambda']}_i = \rho_1(\prod_{i=0}^{k-1} \lambda_i) \times \overline{[1]}_i$. Thus, it will not be a matched with $\overline{[\lambda]}_i$ and the reconstruction result of $[0']_i$ will not equal to 0, resulting in multiplication deemed as incorrect. However, in this case $([\lambda]_i, \lambda_i)$ is assumed to be correct. We can also think about the opposite situation of judging the incorrect result by giving priority to safety.

Case 2: We suppose a situation where both results of the secrecy computation $([\mu_1]_i, \mu_{1,i})$ and $([\lambda]_i, \lambda_i)$ are incorrect, without adversary.

If we assumed that $\mu_1 = \rho_1(\prod_{i=0}^{k-1} \mu_{1,i})$, $\lambda = \rho_0(\prod_{i=0}^{k-1} \lambda_i)$, equation (3) will become $\overline{[\lambda']}_i = \rho_1(\prod_{i=0}^{k-1} \lambda_i) \times \overline{[1]}_i$, and equation (4) will become $(\rho_0 - \rho_1) \times (\prod_{i=0}^{k-1} \lambda_i) \times \overline{[1]}_i$. Therefore, only when both ρ_0 and ρ_1 coincidently match, reconstruction of $[0']_i$ will produce a value of 0, causing an error in the verification process. If the verification process is repeated by changing $([\mu_1]_i, \mu_{1,i})$ to $([\mu_h]_i, \mu_{h,i})(h = 2,3 \ldots)$, the percentage for errors to occur in verification of $([\lambda]_i, \lambda_i)$ can be arbitrarily reduced.

Case 3: We suppose a situation where the adversary makes use of servers that it had taken over and tried to output false result of multiplication and at the same time tried to establish equations (3) and (4).

From the proof of security against *Adversary 2* in Section 4.2, even if players and servers are dishonest, no one can learn about the result of the secrecy multiplication. In other words, because the adversary has no information on $\overline{[\lambda_0\lambda_1\lambda_2]}''^{(2k)}_{i+1}$, $\overline{[\lambda_0\lambda_1\lambda_2]}''^{(2k)}_{i+1+k}$ held by an honest server, the adversary cannot learn about the result of the secrecy multiplication. Therefore, because the adversary has no information on λ and μ_1, the adversary will not be able to learn about ρ_0, ρ_1, which are required to establish equations (3) and (4). The detail of the proof is shown in Appendix 2.

Case 4: We suppose a situation where the adversary performs protocol in Section 4.1 correctly and obtains the correct result of multiplication. However, the adversary tries to establish equations (3) and (4) by outputting false values.

For example, the adversary makes use of two servers S_i, S_j that it had taken over to output false values of $(\lambda'_i/\mu'_{1,i}) \neq (\lambda_i/\mu_{1,i})$ and $(\lambda'_j/\mu'_{1,j}) \neq (\lambda_j/\mu_{1,j})$. However, if the values of $\lambda'_i/\mu'_{1,i}, \lambda'_j/\mu'_{1,j}$ satisfies $(\lambda'_i/\mu'_{1,i})(\lambda'_j/\mu'_{1,j}) = (\lambda_i/\mu_{1,i})(\lambda_j/\mu_{1,j})$, equations (3) and (4) will be established. In addition, we also need to consider a different corruption where the adversary is assumed to has shares of $\lambda_i, \mu_{1,i}$ where $\lambda = \prod_{i=0}^{k-1} \lambda_i$, $\mu_1 = \prod_{i=0}^{k-1} \mu_{1,i}$. In regard to that, as shown in Case 3, the adversary cannot learn about the shares of λ where $\lambda = \prod_{i=0}^{k-1} \lambda_i$ if dishonest activities are performed in the protocol shown in Section 4.1. Therefore, equations (3) and (4) cannot be established. We can conclude that if equations (3) and (4) are established, the result of secrecy multiplication is correct. It is also possible that false shares are used as inputs for the next secrecy computation. However, the problem whether correct shares of multiplication are used for the consecutive secrecy computation will depend on the verification process of that particular secrecy computation.

From above, in regard to multiple $([\mu_h]_i, \mu_{h,i})$, if equations (3) and (4) are established, we can conclude that the protocol in Section 4.1 had been performed correctly.

4.4 Implementation into generation of Multiplication Triple

In the ***offline phase*** (preprocessing phase) of the SPDZ method, "Multiplication Triple" is generated by using homomorphic encryption. Even when using the SPDZ 2 method, if the number of servers/parties is at $N = k$ no substantial problem will occur, even without the use of homomorphic encryption. Therefore, we will show the method of generating multiplication triple using our proposed method. Below, we show the algorithm for u number of players. By this, SPDZ 2 can also be improved with faster processing speed and realization of information-theoretical security.

Protocol 4.4: Generation of multiplication triple for $N = k$

1. u number of players P_i each generates random numbers $\lambda_{0,i}$ and $\lambda_{1,i}$, and computes $\lambda_i = \lambda_{0,i}\lambda_{1,i}$.
2. In regard to λ_i, u number of players perform secrecy multiplication as shown in Section 4.1 and obtain $\overline{[\lambda]}_i^{(k)}$. Note that $\lambda = \lambda_1 \dots \lambda_u$.
3. In regard to $\lambda_{0,i}$, u number of players perform secrecy multiplication as shown in Section 4.1 and obtain $\overline{[\lambda_0]}_i^{(k)}$. Note that $\lambda_0 = \lambda_{0,1} \dots \lambda_{0,u}$.
4. In regard to $\lambda_{1,i}, u$ number of players perform secrecy multiplication as shown in Section 4.1 and obtain $\overline{[\lambda_1]}_i^{(k)}$. Note that $\lambda_1 = \lambda_{1,1} \dots \lambda_{1,u}$

Suppose that when $\lambda_0 = a, \lambda_1 = b, \lambda = ab$, "Multiplication Triple" can be obtained. In addition, it is obvious that the security for this algorithm can be proven the same way as shown in Sections 4.2. and 4.3.

4.5 Generation of Set of Shares on 1 When the Same Participant Participates in the Secrecy Computation of the TUS methods

The method of generating a set of shares on 1 shown in Section 4.1 is valid when it is performed independently from the actual ***Computation Phase*** using TUS methods. However, there is a problem if the participant of the secrecy computation is the same entity as the player who provided an input for generating the set of shares on 1. For example, assume that $N = k = 2$ and Player A, who knows about random numbers ε_0, participates in the secrecy computation. If the player could learn about the information of the server that reconstructed the random number ε_1, Player A would be able to learn about the random number ε used in the set of shares on 1. Therefore, Condition 3 is required. In order to omit Condition 3, Player P_i in Section 4.1 must be replaced with servers S_i. In this case, the adversary will be able to learn information from at most $k - 1$ number of servers; however, this is the same as *Adversary 2* shown in 4.2, where $u - 1 = k - 1$ number of players (who provide inputs) are attacked. Because our proposed method is secure against this, it is not a problem. Therefore, in the case where the player who provided input in Section 4.1 is the participant for the secrecy computation, if servers S_i generate random numbers λ_i and store them to be used in the ***Computation Phase***, even if $k - 1$ numbers of ε_i are leaked, random number ε, which is the result of multiplication, is not known.

In addition, because random number λ_i, which is the input for Multiplication Triple and the set of shares on 1, does not need to include the value of 0, leakage of the result will not occur.

5 Proposed Method 2

By dividing the computation process into a ***Preprocessing Phase*** and a ***Computation Phase***, it allows us to shift parts of the computation that require communication to the ***Preprocessing Phase***, where information that does not depend on any of the private values can be generated in advance. This can be used to significantly reduce the cost for communication in the ***Computation Phase*** and speed up the whole process. When computation is performed without repetition, the ***Computation Phase*** can be performed without any communication cost. However, when computation is performed with repetition, communication between servers is required during reconstruction. Here, computation with repetition means that the result of computation is reconstructed temporarily and is then used as an input for the consecutive secrecy computation of multiplication. In this case, communication between servers is required only for the reconstruction process. On the other hand, in the case where repetition is not required, the cost for communication can be totally eliminated from the ***Computation Phase***, realizing a very effective method of secure computation. Below, we show two types of protocols that can accommodate computation without repetition (TUS 5' Method) and computation with repetition (TUS 5 Method).

5.1 TUS 5' Method: Protocol for Computation without Repetition

In computation without repetition, even if the result of computation is equal to 0, because the result is not reconstructed midway we only need to respond when the input is zero (which means that we can use the same approach as the TUS3' method proposed by Iwamura et al. [15]). However, the input needs to be within modulus p and be a number under $p - 2$. For ease of understanding, we show our protocol for computation of the product-sum operation of $ab + c$ in the setting of $N = k$. However, extension of our protocol into computation such as $\sum_{i=1}^{m}(a_i b_i \dots z_i)$ is also possible. In addition, we assume that communication between the dealer, players and servers is secure.

Preprocessing Phase

1. Servers S_i $(i = 0, \dots, k - 1)$ that participate in the computation produce 4 random numbers $\varepsilon_{j,i} (j = 1, \dots 4)$, and by using the method shown in Chapter 4, multiply them together and compute 4 shares on $\overline{[\varepsilon_j]}_i$, and store $\overline{[\varepsilon_j]}_i$ and $\varepsilon_{j,i}$.
2. Dealer D generates k number of random numbers $\alpha_{0,0}, \alpha_{0,1}, \dots, \alpha_{0,k-1}$, computes $\alpha_0 = \prod_{j=0}^{k-1} \alpha_{0,j}$ and sends $\alpha_{0,j}$ to Servers S_j.

3. Dealer D performs the same process as Step 2 on random numbers $\beta_{0,j}$ and $\gamma_{0,j}$.
4. k number of Servers S_j $(j = 0, \dots, k-1)$ generate random numbers $\delta_{0,j}$, compute the following, and send to one of the servers (Here, we assume the server to be Server S_0).

$$\frac{\delta_{0,j}}{\alpha_{0,j}\beta_{0,j}\varepsilon_{1,j}}, \frac{\delta_{0,j}}{\alpha_{0,j}\varepsilon_{2,j}}, \frac{\delta_{0,j}}{\beta_{0,j}\varepsilon_{3,j}}, \frac{\delta_{0,j}}{\gamma_{0,j}\varepsilon_{4,j}}$$

5. By using the received information, Server S_0 computes the following and sends $\delta_0/\alpha_0\beta_0\varepsilon_1$, $\delta_0/\alpha_0\varepsilon_2$, $\delta_0/\beta_0\varepsilon_3$, $\delta_0/\gamma_0\varepsilon_4$ to all servers.

$$\frac{\delta_0}{\alpha_0\beta_0\varepsilon_1} = \prod_{j=0}^{k-1} \frac{\delta_{0,j}}{\alpha_{0,j}\beta_{0,j}\varepsilon_{1,j}}$$

$$\frac{\delta_0}{\alpha_0\varepsilon_2} = \prod_{j=0}^{k-1} \frac{\delta_{0,j}}{\alpha_{0,j}\varepsilon_{2,j}}$$

$$\frac{\delta_0}{\beta_0\varepsilon_3} = \prod_{j=0}^{k-1} \frac{\delta_{0,j}}{\beta_{0,j}\varepsilon_{3,j}}$$

$$\frac{\delta_0}{\gamma_0\varepsilon_4} = \prod_{j=0}^{k-1} \frac{\delta_{0,j}}{\gamma_{0,j}\varepsilon_{4,j}}$$

6. All servers S_i compute and hold the following information:

$$\overline{\left[\frac{\delta_0}{\alpha_0\beta_0}\right]_i} = \frac{\delta_0}{\alpha_0\beta_0\varepsilon_1} \times \overline{[\varepsilon_1]_i}$$

$$\overline{\left[\frac{\delta_0}{\alpha_0}\right]_i} = \frac{\delta_o}{\alpha_0\varepsilon_2} \times \overline{[\varepsilon_2]_i}$$

$$\overline{\left[\frac{\delta_0}{\beta_0}\right]_i} = \frac{\delta_0}{\beta_0\varepsilon_3} \times \overline{[\varepsilon_3]_i}$$

$$\overline{\left[\frac{\delta_0}{\gamma_0}\right]_i} = \frac{\delta_0}{\gamma_0\varepsilon_4} \times \overline{[\varepsilon_4]_i}$$

7. Dealer D sends α_0 to Player A, β_0 to Player B, and γ_0 to Player C.

Encryption Phase

1. Player A computes $\alpha_0(a+1) = \alpha_0 \times (a+1)$ for secret information a and sends to all servers.

2. Player B computes $\beta_0(b+1) = \beta_0 \times (b+1)$ for secret information b and sends to all servers.
3. Player C computes $\gamma_0(c+1) = \gamma_0 \times (c+1)$ for secret information c and sends to all servers.

Computation Phase (Product-Sum Operation)

1. All servers S_i compute $\overline{[\delta_0(ab+c)]_i}$ as shown below

$$\overline{[\delta_0(ab+c)]_i} = \alpha_0(a+1) \times \beta_0(b+1) \times \overline{\left[\frac{\delta_0}{\alpha_0\beta_0}\right]_i} - \alpha_0(a+1) \times \overline{\left[\frac{\delta_0}{\alpha_0}\right]_i} - \beta_0(b+1) \times \overline{\left[\frac{\delta_0}{\beta_0}\right]_i} + \gamma_0(c+1) \times \overline{\left[\frac{\delta_0}{\gamma_0}\right]_i}$$

Reconstruction Phase

1. Player who wishes to reconstruct the result collects $\overline{[\delta_0(ab+c)]_j}$ and $\delta_{0,j}$ from k number of servers, reconstructs $\delta_0(ab+c), \delta_0$, and computes the result of $ab+c$ as follows:

$$\frac{\delta_0(ab+c)}{\delta_0} = ab+c$$

5.2 Security Analysis and Discussion of the TUS 5' Method

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 consider these two types of adversaries. We can state that our proposed secrecy computation method is secure if it is secure against Adversaries 1 and 2 defined below. However, if each adversary can learn information that he/she wanted to learn, the attack will be considered to be successful.

Adversary 1 :

In the product-sum operation, one of the players who inputted the secret and the player who reconstructed the output constitute the adversary. *Adversary 1* has information on 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 2:

In the product-sum operation, two of the players who inputted secrets constitute the adversary. *Adversary 2* has information on 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.

The security proof for the TUS5' method is shown below.

Proof of security of the Preprocessing Phase

First, in Step 1, as proven in Section 4, the process of generating a set of shares on 1 is secure. In addition, because our proposed method assumed a semi-honest adversary, we assumed that the Dealer performed Step 2, Step 3, and Step 7 securely and correctly, and sent to each server. In Step 4. k number of servers S_j $(j = 0, \dots, k-1)$ compute $\delta_{0,j}/\alpha_{0,j}\beta_{0,j}\varepsilon_{1,j}$, $\delta_{0,j}/\alpha_{0,j}\varepsilon_{2,j}$, $\delta_{0,j}/\beta_{0,j}\varepsilon_{3,j}$, $\delta_{0,j}/\gamma_{0,j}\varepsilon_{4,j}$ and send to server S_0. However, *Adversary 1* and *Adversary 2* cannot decompose each random number from the above information. Therefore, *Adversary 1* and *Adversary 2* cannot learn about random numbers $\delta_0, \alpha_0, \beta_0, \gamma_0, \varepsilon_1, \dots, \varepsilon_4$. In Step 5, Server S_0 computes and broadcasts $\delta_0/\alpha_0\beta_0\varepsilon_1$, $\delta_0/\alpha_0\varepsilon_2$, $\delta_0/\beta_0\varepsilon_3$, $\delta_0/\gamma_0\varepsilon_4$; however, *Adversary 1* and *Adversary 2* cannot decompose this information into random numbers. Shares in Step 6 are generated by using shares on 1. Because *Adversary 1* and *Adversary 2* have no information on random numbers, or random number ε_j in shares on 1, we can say that the shares for each server can be generated securely.

Therefore, the following statement is true and the evaluation above remains valid for the rest of the shares.

$$H\left(\overline{\left[\frac{\delta_0}{\alpha_0\beta_0}\right]_i}\right) = H\left(\overline{\left[\frac{\delta_0}{\alpha_0\beta_0}\right]_i} \middle| \frac{\delta_0}{\alpha_0\beta_0\varepsilon_1}\right)$$

Proof of security of the Encryption Phase

Because the secret information is smaller than $p-2$, even if 1 is added to the secret information it will not become 0. In addition, a random number generated by the dealer is secure. Therefore, the following statement is true, and remains true for the rest of the secret information b and c.

$$H(a) = H\big(a \big| \alpha_0(a+1)\big)$$

Proof of security of the Computation Phase

Security against Adversary 1

Assume that the player who inputted secret information b and random numbers β_0 is the adversary. He/she also has information from $k-1$ servers in addition to the result of computation. In Step 6 of the ***Preprocessing Phase***, *Adversary 1* has information about $\delta_0/\alpha_0\beta_0\varepsilon_1$, $\delta_0/\alpha_0\varepsilon_2$, $\delta_0/\beta_0\varepsilon_3$, $\delta_0/\gamma_0\varepsilon_4$. In the ***Distribution Phase***, *Adversary 1* has information about $\alpha_0(a+1), \gamma_0(c+1)$. In the ***Reconstruction Phase***, *Adversary 1* has information about $\delta_0(ab+c), \delta_0$ and $ab+c$. By using this information, we assume that *Adversary 1* tries to learn about secret information a, c. However, if less than k number of shares about random numbers are collected, information regarding the random numbers and secret information will not be leaked.

First, in order to simplify the problem, we redefined the parameter above to avoid any duplication of a parameter. As a result, *Adversary 1* has information about $B = \{b, \beta_0, \alpha_0(a+1), \gamma_0(c+1), 1/\alpha_0\varepsilon_1, 1/\alpha_0\varepsilon_2, 1/\varepsilon_3, 1/\gamma_0\varepsilon_4, \delta_0, ab+c\}$.

One of the methods to learn about secret information a, is to first learn about random number α_0. However, *Adversary 1* has no information on random numbers $\varepsilon_1 \sim \varepsilon_4$, therefore, he/she is not able to learn about α_0 from B. Similarly, because *Adversary 1* has no information on γ_0, he/she is not able to learn about c. Even if *Adversary 1* tries to learn about a, c from $ab+c$, he/she cannot decompose secret a, c from $ab+c$. Therefore, the following statements are true.

$$H(a) = H(a|B)$$

$$H(c) = H(c|B)$$

In addition, the evaluation above remains valid even if the adversary is the player who inputted secret a or c. Therefore, the TUS5' method is information-theoretically secure against *Adversary 1*.

Security against Adversary 2

Assume that the player who inputted secrets b, c and random numbers β_0, γ_0 is the adversary. He/she also has information from $k-1$ servers. Therefore, in the ***Preprocessing Phase***, *Adversary 2* has information about $\delta_0/\alpha_0\beta_0\varepsilon_1$, $\delta_0/\alpha_0\varepsilon_2$, $\delta_0/\beta_0\varepsilon_3$, $\delta_0/\gamma_0\varepsilon_4$. In the ***Distribution Phase***, *Adversary 2* has information about $\alpha_0(a+1)$. By using this information, the adversary tries to learn about the remaining input a or output $ab+c$. However, if less than k number of shares about random numbers are collected, information regarding the random numbers and secret information will not be leaked.

In order to simplify the problem, we redefined the parameters above to avoid any their duplication. As a result, we know that *Adversary 2* has information about $BC = \{b, c, \beta_0, \gamma_0, \alpha_0(a+1), \delta_0/\alpha_0\varepsilon_1, \delta_0/\alpha_0\varepsilon_2, \delta_0/\varepsilon_3, \delta_0/\varepsilon_4, \delta_0(ab+c)\}$.

One of the methods to learn about secret information a, is to first learn about random number α_0. However, *Adversary 2* has no information on random numbers $\varepsilon_1 \sim \varepsilon_4$, therefore, he/she is not able to learn about α_0 from BC. In addition, even if *Adversary 2* tries to learn about output $ab+c$ from $\delta_0(ab+c)$, because *Adversary 2* has no information on δ_0, the output will not be leaked. The same can also be said for other combinations. Therefore, the following statements are true.

$$H(a) = H(a|BC)$$

$$H(ab+c) = H(ab+c|BC)$$

In addition, if $b=1$, or $c=0$, it is clear that our protocol is also secure against secrecy computation of addition and multiplication (because $b=1$ or $c=0$ is treated as one of the pieces of information that is known to the adversary and the adversary tries to learn about the remaining one input or output) .

Proof of security of the Reconstruction Phase

Even if $ab + c$ is equal to 0, because nothing is leaked from k or less number of shares on $\overline{[\delta_0(ab + c)]}_i$ and $\delta_{0,j}$, we can say that *Adversary 2* is not able to learn about the result of the computation.

Because the ***Preprocessing Phase*** only processes information that does not depend on any of the secret information, it can be performed in advance before inputting the secret information. Secret information is introduced in the ***Encryption Phase*** and the result is sent to all servers. Both these phases are equal to the process of distribution/setting of the secret sharing scheme. Therefore, the actual computation is performed by the ***Computation Phase*** using values set in advance, so no communication between servers is needed. Finally, in the ***Reconstruction Phase***, a player collects $\overline{[\delta_0(ab + c)]}_i$ and $\delta_{0,j}$ from k number of servers S_j and obtains $ab + c$. From this, we learn that communication only occurs in the ***Preprocessing Phase***, ***Encryption Phase*** and ***Reconstruction Phase***.

5.3 TUS 5 Method: Protocol for Computation with Repetition

Because no repetition of computation is assumed in the TUS5' method, reconstruction is only performed by the player who wishes to know about the results of computation. However, when computation with repetition is assumed, encrypted results are reconstructed temporarily by one of the servers and are then used as input for the consecutive computation. Here, if the reconstructed result of that particular computation is equal to 0, the information that the result of that particular computation is equal to 0 will be leaked. Even if a value of 1 is added to the result before being reconstructed, if the reconstructed result is still equal to 0, the information that the resulting value is equal to -1 in modulus p will be leaked. Therefore, the protocol shown below will introduce the measure to accommodate secrecy computation with repetition.

Preprocessing Phase

1. Servers S_i $(i = 0, \dots, k - 1)$ that participate in the computation generate random numbers $\varepsilon_{j,i}(j = 1, \dots required\ number)$ and by using the method in Section 4, multiply them together and compute enough number of shares $\overline{[\varepsilon_j]}_i$ on 1, and store $\overline{[\varepsilon_j]}_i$ and $\varepsilon_{j,i}$.
2. Dealer D generates 2 sets of k numbers of random numbers $\alpha_{0,0}, \alpha_{0,1}, \dots, \alpha_{0,k-1}$ and $\alpha_{2,0}, \alpha_{2,1}, \dots, \alpha_{2,k-1}$, computes the following and sends $\alpha_{0,j}, \alpha_{2,j}$ to Servers S_j.

$$\alpha_0 = \prod_{j=0}^{k-1} \alpha_{0,j}$$

$$\alpha_2 = \prod_{j=0}^{k-1} \alpha_{2,j}$$

3. Dealer D performs the same process as Step 2 on random numbers $\beta_{0,j}, \beta_{2,j}$ and $\gamma_{0,j}, \gamma_{2,j}$.

4. Servers S_j $(j = 0, \dots, k-1)$ compute the following and send to Server S_0.

$$\frac{\delta_{0,j}}{\alpha_{0,j}\beta_{0,j}\varepsilon_{1,j}}, \frac{\delta_{0,j}}{\alpha_{0,j}\beta_{2,j}\varepsilon_{2,j}}, \frac{\delta_{0,j}}{\alpha_{2,j}\beta_{0,j}\varepsilon_{3,j}}$$

$$\frac{\delta_{0,j}}{\gamma_{0,j}\varepsilon_{4,j}}, \frac{\delta_{2,j}}{\alpha_{2,j}\beta_{2,j}\varepsilon_{5,j}}, \frac{\delta_{2,j}}{\gamma_{2,j}\varepsilon_{6,j}}$$

5. Server S_0 computes $\delta_0/\alpha_0\beta_0\varepsilon_1$, $\delta_0/\alpha_0\beta_2\varepsilon_2$, $\delta_0/\alpha_2\beta_0\varepsilon_3$, $\delta_0/\gamma_0\varepsilon_4$, $\delta_2/\alpha_2\beta_2\varepsilon_5$, $\delta_2/\gamma_2\varepsilon_6$ as shown below, and sends to all servers.

$$\frac{\delta_0}{\alpha_0\beta_0\varepsilon_1} = \prod_{j=0}^{k-1} \frac{\delta_{0,j}}{\alpha_{0,j}\beta_{0,j}\varepsilon_{1,j}}$$

$$\frac{\delta_0}{\alpha_0\beta_2\varepsilon_2} = \prod_{j=0}^{k-1} \frac{\delta_{0,j}}{\alpha_{0,j}\beta_{2,j}\varepsilon_{2,j}}$$

$$\frac{\delta_0}{\alpha_2\beta_0\varepsilon_3} = \prod_{j=0}^{k-1} \frac{\delta_{0,j}}{\alpha_{2,j}\beta_{0,j}\varepsilon_{3,j}}$$

$$\frac{\delta_0}{\gamma_0\varepsilon_4} = \prod_{j=0}^{k-1} \frac{\delta_{0,j}}{\gamma_{0,j}\varepsilon_{4,j}}$$

$$\frac{\delta_2}{\alpha_2\beta_2\varepsilon_5} = \prod_{j=0}^{k-1} \frac{\delta_{2,j}}{\alpha_{2,j}\beta_{2,j}\varepsilon_{5,j}}$$

$$\frac{\delta_2}{\gamma_2\varepsilon_6} = \prod_{j=0}^{k-1} \frac{\delta_{2,j}}{\gamma_{2,j}\varepsilon_{6,j}}$$

6. All servers S_j compute and hold the following.

$$\overline{\left[\frac{\delta_0}{\alpha_0\beta_0}\right]_i} = \frac{\delta_0}{\alpha_0\beta_0\varepsilon_1} \times \overline{[\varepsilon_1]_i}$$

$$\overline{\left[\frac{\delta_0}{\alpha_0\beta_2}\right]_i} = \frac{\delta_0}{\alpha_0\beta_2\varepsilon_2} \times \overline{[\varepsilon_2]_i}$$

$$\overline{\left[\frac{\delta_0}{\alpha_2\beta_0}\right]_i} = \frac{\delta_0}{\alpha_2\beta_0\varepsilon_3} \times \overline{[\varepsilon_3]_i}$$

$$\overline{\left[\frac{\delta_0}{\gamma_0}\right]_i} = \frac{\delta_0}{\gamma_0\varepsilon_4} \times \overline{[\varepsilon_4]_i}$$

$$\overline{\left[\frac{\delta_2}{\alpha_2\beta_2}\right]_i} = \frac{\delta_2}{\alpha_2\beta_2\varepsilon_5} \times \overline{[\varepsilon_5]_i}$$

$$\overline{\left[\frac{\delta_2}{\gamma_2}\right]_i} = \frac{\delta_2}{\gamma_2\varepsilon_6} \times \overline{[\varepsilon_6]_i}$$

7. Dealer D sends α_0, α_2 to Player A, β_0, β_2 to Player B, and γ_0, γ_2 to Player C.

Encryption Phase

1. Player A generates random number α_1, such that $a + \alpha_1 \neq 0$ for secret information a, computes $\alpha_0(a + \alpha_1), \alpha_2\alpha_1$, and sends to all servers.
2. Player B generates random number β_1, such that $b + \beta_1 \neq 0$ for secret information b, computes $\beta_0(b + \beta_1), \beta_2\beta_1$, and sends to all servers.
3. Player C generates random number γ_1, such that $c + \gamma_1 \neq 0$ for secret information c, computes $\gamma_0(c + \gamma_1), \gamma_2\gamma_1$, and sends to all servers.

Computation Phase

1. All servers S_i compute $\overline{[\delta_0(a' + \alpha'_1)]_i}, \overline{[\delta_2\alpha'_1]_i}$ as follows. Note that $a' = ab + c, \alpha'_1 = \gamma_1 - \alpha_1\beta_1$.

$$\begin{aligned}\overline{[\delta_0(a' + \alpha'_1)]_i} = {} & \alpha_0(a + \alpha_1) \times \beta_0(b + \beta_1) \times \overline{\left[\frac{\delta_0}{\alpha_0\beta_0}\right]_i} \\ & - \alpha_0(a + \alpha_1) \times \beta_2\beta_1 \times \overline{\left[\frac{\delta_0}{\alpha_0\beta_2}\right]_i} - \beta_0(b + \beta_1) \times \alpha_2\alpha_1 \times \overline{\left[\frac{\delta_0}{\alpha_2\beta_0}\right]_i} \\ & + \gamma_0(c + \gamma_1) \times \overline{\left[\frac{\delta_0}{\gamma_0}\right]_i}\end{aligned}$$

$$\overline{[\delta_2\alpha'_1]_i} = \gamma_2\gamma_1 \times \overline{\left[\frac{\delta_2}{\gamma_2}\right]_i} - \alpha_2\alpha_1 \times \beta_2\beta_1 \times \overline{\left[\frac{\delta_2}{\alpha_2\beta_2}\right]_i}$$

2. Server S_0 collects $[\delta_2\alpha'_1]_j$ from k number of servers S_j, and reconstructs $\delta_2\alpha'_1$.
3. If $\delta_2\alpha'_1 = 0$ or $\delta_0(a' + \alpha'_1) = 0$, skip to ***Correction Phase***, and obtain $\delta_2\alpha''_1, \delta_0(a' + \alpha''_1)$, such that $\delta_2\alpha''_1 \neq 0, \delta_0(a' + \alpha''_1) \neq 0$.

Correction Phase

1. Servers S_j generate random numbers ρ_j, λ_j, and send the product of $\rho_j\lambda_j$ to Server S_0.
2. Server S_0 computes the multiplication of $\rho\lambda = \Pi\rho_j\lambda_j$, and broadcasts the result to all servers.
3. Servers S_j compute the following and send to Server S_0. However, a set of shares on 1 used in the ***Correction Phase***, such as $\varepsilon_7, \varepsilon_8$ are also generated through Step 1 of the ***Preprocessing Phase***.

$$\frac{\delta_{2,j}}{\rho_j\varepsilon_{7,j}}, \frac{\delta_{0,j}}{\rho_j\varepsilon_{8,j}}$$

4. Server S_0 computes the following and sends to all servers.

$$\frac{\delta_2}{\rho\varepsilon_7} = \prod_{j=0}^{k-1} \frac{\delta_{2,j}}{\rho_j\varepsilon_{7,j}}$$

$$\frac{\delta_0}{\rho\varepsilon_8} = \prod_{j=0}^{k-1} \frac{\delta_{0,j}}{\rho_j\varepsilon_{8,j}}$$

5. All servers S_i compute the following.

$$\overline{[\delta_2\lambda]}_i = \rho\lambda \times \frac{\delta_2}{\rho\varepsilon_7} \times \overline{[\varepsilon_7]}_i$$

$$\overline{[\delta_0\lambda]}_i = \rho\lambda \times \frac{\delta_0}{\rho\varepsilon_8} \times \overline{[\varepsilon_8]}_i$$

6. All servers S_i compute the following. Note that $\alpha"_1 = \alpha'_1 + \lambda$.

$$\overline{[\delta_2\alpha"_1]}_i = \overline{[\delta_2\alpha'_1]}_i + \overline{[\delta_2\lambda]}_i$$

$$\overline{[\delta_0(a' + \alpha"_1)]}_i = \overline{[\delta_0(a' + \alpha'_1)]}_i + \overline{[\delta_0\lambda]}_i$$

7. Server S_0 first collects $\overline{[\delta_2\alpha"_1]}_i$ from k numbers of Servers S_j and reconstructs $\delta_2\alpha"_1$. If $\delta_2\alpha"_1 = 0$, Step 1 is repeated with different random numbers. If $\delta_2\alpha"_1$ is not equal to 0, Server S_0 collects $\overline{[\delta_0(a' + \alpha"_1)]}_j$ from k numbers of Servers S_j and reconstructs $\delta_0(a' + \alpha"_1)$. If $\delta_0(a' + \alpha"_1) = 0$, returns to Step 1, and repeats the process by changing the random numbers. This process is repeated until both $\delta_0(a' + \alpha"_1), \delta_2\alpha"_1$ are not equal to 0.

Reconstruction Phase

1. The player collects $\delta_{0,j}, \delta_{2,j}$ from k numbers of servers S_j who participated in the last computation, and by using $\delta_0(a' + \alpha"_1), \delta_2\alpha"_1$ and the reconstructed δ_0, δ_2, computes result a' by using the equation below.

$$\frac{\delta_0(a' + \alpha"_1)}{\delta_0} - \frac{\delta_2\alpha"_1}{\delta_2} = a'$$

5.4 Security Analysis of the TUS5 Method

The number of random numbers produced in the ***Preprocessing Phase*** increased compared to the ***Preprocessing Phase*** of the TUS5' method; however, the security remains the same as in the TUS5' method. In the ***Encryption Phase***, because random number

α_1 is added to produce $a + \alpha_1 \neq 0$, even if secret information $a = 0$, $\alpha_0(a + \alpha_1) \neq 0$, secret information a will not be leaked. In the ***Computation Phase***, the number of random numbers known to *Adversary 1* and *Adversary 2* increases. However, the security of the ***Computation Phase*** in the TUS5 method can still be proven to be the same as in the TUS5' method. In addition, the security of the ***Reconstruction Phase*** is also the same as in the TUS5' method. However, because of the limit on the number of pages, we have omitted the detailed proof. Below, we show the security analysis for the newly added ***Correction Phase***.

Proof of security of the Correction Phase

In the ***Correction Phase***, if either one of the reconstructed results is equal to 0, a new random number λ is added for α'_1 to be renewed as α''_1. Only $\delta_0(a' + \alpha''_1), \delta_2\alpha''_1$ that are not equal to 0 become the new $\alpha_0(a + \alpha_1), \alpha_2\alpha_1$. This allows for the repetition of computation of secrecy product-sum computation. Note that, in the ***Correction Phase***, $\delta_2\alpha''_1$ and $\delta_0(a' + \alpha''_1)$ are not reconstructed together. If either one of the reconstructed results $\delta_2\alpha''_1 = 0$, $\delta_0(a' + \alpha''_1) = 0$, information about either $\alpha''_1 = 0$ or $(a' + \alpha''_1) = 0$ will be leaked. However, because α''_1 consists of only random numbers, information regarding the secret information will not be leaked. In addition, if $(a' + \alpha''_1) = 0$, information that $a' = -\alpha''_1$ will be leaked. However, because α''_1 is not equal to 0 and is encrypted by $\delta_2\alpha''_1$, information regarding secret information will not be leaked. Therefore, Adversaries 1 and 2 cannot learn about the secret information.

In addition, even if the ***Correction Phase*** is repeated, because different random numbers (except for δ_2, δ_0) are used each time, $\delta_2, \delta_0, \lambda$ will not be leaked from $\overline{[\delta_2\lambda]_i}, \overline{[\delta_0\lambda]_i}$. Therefore, Adversaries 1 and 2 cannot learn about the secret information and random numbers used.

6 Discussion and Consideration

6.1 Merits of Secrecy Computation for $N < 2k - 1$

We proposed secrecy computation using a secret sharing scheme under the setting of $n < 2k - 1$. There are two merits of this approach.

The first merit is the parameters of n, k can be optimized without having to increase the number of servers. Typically, when parameter k is decided based on the attack tolerance of the system and when considering the loss resistance of servers, n is set such that $n \geq k$. However, when performing secrecy multiplication, because $n \geq 2k - 1$, the number of servers needed will increase, increasing overall cost. In a conventional method, there are examples where in order to minimize the increase of cost, parameters n and k are set at 3 and 2, respectively. In such a case, even if one of the servers breaks down, the secrecy computation will no longer be possible, thus losing the loss resistance of servers.

The second merit is that because parameters n and k can be set such that $n = k$, if the owner of the secret information participates in the computation and he/she manages his share securely, even if all players except for the owner collude, the information regarding the secret information will not be leaked. In contrast, in the case when $n \geq$

$2k-1$, even if the owner manages the share securely, if more than k players except for the owner colluded, all information regarding the secret information will be leaked.

We can strictly say that our proposal realizes $N < 2k-1$, but it can also realize the above merits.

6.2 Security Comparison with SPDZ 2 Method

In the SPDZ 2 method [13], the owner of the secret information is one of the players and it is assumed that even if all players except for the owner ($n-1$ number of players) is attacked, the secret information will not be leaked. However, in some cases such as when the reconstructed result is shared with all players, the above statement will not be true. For example, in the computation of f shown below, suppose that a_1 is the secret information of the owner.

$$a = f(a_1, a_2, \dots, a_n)$$

Here, if the result of computation a is known to $(n-1)$ number of players, by using the computation shown below, secret information a_1 of the owner will be leaked. However, g is a function of f where a_1 had been transformed to $a_1 =$ and f is a function that realizes arithmetic computation.

$$a_1 = g(a, a_2, \dots, a_n)$$

Therefore, the security definition of the SPDZ 2 method only ensures security during the secrecy computation phase, where secret information of the owner will not be leaked even if $n-1$ players (excluding the owner) colluded.

The same can be said for the TUS method, where when performing a secrecy computation it can also be resistant to attack by $(n-1)$ number of players (excluding the owner). For example, in the ***Computation Phase*** of the product-sum operation, shown in Section 5.1, the reconstructed result is not known. Therefore, from the security analysis against *Adversary 2*, we learn that out of t number of players, even if $t-1$ number of players colluded, the rest of the secret information will not be leaked.

6.3 About Condition 3

Through the ***Preprocessing Phase*** in the TUS5 and TUS5' methods, random numbers used are generated beforehand and are sent directly to all servers participating in the secrecy computation. However, if one of the servers is broken and cannot be used, secrecy computation can no longer be continued because random numbers that are handled by that server will be lost. Therefore, random numbers used in the secrecy computation are not sent directly to all participating servers, but instead are distributed by using the XOR method. By this, even if one of the servers is no longer functional, if the substitute server could reconstruct the random numbers used by that particular server, the process of secrecy computation can be continued. In this case, Condition 3 is needed. In other words, the newly added server and the previous server are deemed as the same server and it must only handle the same random numbers as the previous server. However, by doing this, in the case of $n > k$, the server loss-resistant characteristic of the secret sharing scheme could also be maintained. In addition, even if the old server is honest and the new server is not honest, we assumed only up to $k-1$ servers were not honest out of the original n number of servers. Therefore, it will not cause any problem.

7 Performance Evaluation

7.1 Function used for Evaluating Performance

There are many cases where Advanced Encryption Standard (AES) is used for evaluating the performance of a secrecy computation method. However, regarding AES, where the process is considerably light, it is hard to imagine a situation where the process of encryption is outsourced. In a case such as Public Key Encryption where the process is considerably heavy, it is beneficial to outsource the process of encryption. However, this is not equivalent to secrecy computation by multiple players, but instead simplifies the process of outsourcing the encryption process by one player. Therefore, we make use of the inner-product sum operation by multiple players as the function for evaluating the performance of multiparty computation, as shown below. Operations such as an inner-product sum are often used in statistical calculation such as distribution, sum of squared deviation etc., meaning that it can be applied to areas such as searching of gene sequence and much more. Therefore, the range of possible application of our protocols is very wide.

The detailed algorithm is shown below for an inner-product sum computation between the secret information $(a_1, \dots, a_m)$ of Player A and the secret information $(b_1, \dots, b_m)$ of Player B. The security for this algorithm can be proven the same way as in the TUS5' method.

Preprocessing Phase

1. Servers S_i $(i = 0, \dots, k-1)$ that participate in the computation generate random numbers $\varepsilon_{j,i} (j = 1, \dots \mathit{desired\ number})$, and by using the method shown in Chapter 4, multiply them together and compute the desired number of shares $\overline{[\varepsilon_j]}_i$ on 1. Servers S_i each hold information $\overline{[\varepsilon_j]}_i$ and $\varepsilon_{j,i}$.
2. Player A decides random numbers $(\alpha_{01}, \dots, \alpha_{0m})$ correspond to secret information $(a_1, \dots, a_m)$, and sends random numbers $\alpha_{0l,j}$, such that $\alpha_{0l} = \prod_{j=0}^{k-1} \alpha_{0l,j}$ $(l = 1, \dots, m)$ to servers S_j.
3. Player B decides random numbers $(\beta_{01}, \dots, \beta_{0m})$ correspond to secret information $(b_1, \dots, b_m)$, and sends random numbers $\beta_{0l,j}$, such that $\beta_{0l} = \prod_{j=0}^{k-1} \beta_{0l,j}$ $(l = 1, \dots, m)$ to servers S_j.
4. Servers S_j $(j = 0, \dots, k-1)$ generate random numbers $\delta_{0,j}$, compute the following and send to Server S_0.

$$\frac{\delta_{0,j}}{\alpha_{0l,j}\beta_{0l,j}\varepsilon_{1l,j}}, \frac{\delta_{0,j}}{\alpha_{0l,j}\varepsilon_{2l,j}}, \frac{\delta_{0,j}}{\beta_{0l,j}\varepsilon_{3l,j}}, \frac{\delta_{0,j}}{\varepsilon_{4l,j}} \ (l = 1, \dots, m)$$

5. Server S_0 computes the following using the received information and sends to all servers.

$$\frac{\delta_0}{\alpha_{0l}\beta_{0l}\varepsilon_{1l}} = \prod_{j=0}^{k-1} \frac{\delta_{0,j}}{\alpha_{0l,j}\beta_{0l,j}\varepsilon_{1l,j}}$$

$$\frac{\delta_0}{\alpha_{0l}\varepsilon_{2l}} = \prod_{j=0}^{k-1} \frac{\delta_{0,j}}{\alpha_{0l,j}\varepsilon_{2l,j}}$$

$$\frac{\delta_0}{\beta_{0l}\varepsilon_{3l}} = \prod_{j=0}^{k-1} \frac{\delta_{0,j}}{\beta_{0l,j}\varepsilon_{3l,j}}$$

$$\frac{\delta_0}{\varepsilon_{4l}} = \prod_{j=0}^{k-1} \frac{\delta_{0,j}}{\varepsilon_{4l,j}}$$

6. Servers S_i compute the following:

$$\overline{\left[\frac{\delta_0}{\alpha_{0l}\beta_{0l}}\right]_i} = \frac{\delta_0}{\alpha_{0l}\beta_{0l}\varepsilon_{1l}} \times \overline{\left[\varepsilon_{1l,j}\right]_i}$$

$$\overline{\left[\frac{\delta_0}{\alpha_{0l}}\right]_i} = \frac{\delta_0}{\alpha_{0l}\varepsilon_{2l}} \times \overline{\left[\varepsilon_{2l,j}\right]_i}$$

$$\overline{\left[\frac{\delta_0}{\beta_{0l}}\right]_i} = \frac{\delta_0}{\beta_{0l}\varepsilon_{3l}} \times \overline{\left[\varepsilon_{3l,j}\right]_i}$$

$$\overline{[\delta_0]_i} = \frac{\delta_0}{\varepsilon_{4l}} \times \overline{\left[\varepsilon_{4l,j}\right]_i}$$

Encryption Phase

1. In regard to secret information a_l, Player A computes $\alpha_{0l}(a_l + 1)(l = 1, \dots, m)$ and broadcasts it.
2. In regard to secret information b_l, Player B computes $\beta_{0l}(b_l + 1)(l = 1, \dots, m)$ and broadcasts it.

Computation Phase

1. All servers S_i compute the following:

$$\overline{\left[\sum_{l=1}^{m} \delta_0(a_l b_l)\right]_i} = \sum_{l=1}^{m} \{\alpha_{0l}(a_l + 1) \times \beta_{0l}(b_l + 1) \times \overline{\left[\frac{\delta_0}{\alpha_{0l}\beta_{0l}}\right]_i} - \alpha_{0l}(a_l + 1) \times \overline{\left[\frac{\delta_0}{\alpha_{0l}}\right]_i}$$
$$- \beta_{0l}(b_l + 1) \times \overline{\left[\frac{\delta_0}{\beta_{0l}}\right]_i} + \overline{[\delta_0]_i}\}$$

Reconstruction Phase

1. The player collects $\overline{[\sum_{l=1}^{m} \delta_0(a_l b_l)]_j}, \delta_{0,j}$ from k number of servers, reconstructs $\sum_{l=1}^{m} \delta_0(a_l b_l)\,, \delta_0$ and computes the following :

$$\sum_{l=1}^{m} (a_l b_l) = \sum_{l=1}^{m} \frac{\delta_0(a_l b_l)}{\delta_0}$$

The above computation of the inner-product sum operation does not require any communication. In addition, without the use of a junction process such as "if", and if all the computations for the second process onwards had been decided and computed in advance in the ***Preprocessing Phase***, along with all the combinations of random numbers required, we only need to perform communication when the process of reconstruction is required for computation with repetition. For example, suppose that after the computation of the inner-product sum operation, the result is going to be used for other secrecy computations. By performing the above algorithm using the TUS5 method, one of the servers first reconstruct $\overline{[\sum_{l=1}^{m} \delta_0(a_l b_l - \alpha_{1l}\beta_{1l})]_i}$ and use it as the encrypted input $\sum_{l=1}^{m} \delta_0(a_l b_l - \alpha_{1l}\beta_{1l}) = \delta_0 \sum_{l=1}^{m}(a_l b_l - \alpha_{1l}\beta_{1l})$ for the consecutive computation. By this, the secrecy computation can be continued (a more detail explanation will be discussed later). In addition, if the result of the computation is not equal to 0, even if there are m number of multiplications of $a_l b_l$, the reconstruction process required in the ***Computation Phase*** is only 1. If the result of the computation is equal to 0, the number of reconstructions required for the ***Correction Phase*** will increase; however, this will only happen several times at most.

7.2 Qualitative Comparison

SPDZ 2 method [13] is limited to the setting $n = k$, and Araki et al.'s method [1] is limited to the setting $n = 3, k = 2$. Therefore, comparison by using the same condition is not possible. Only TUS methods allow for parameters n, k to be set at any value and are able to accommodate various use cases. However, SPDZ 2 can accommodate a malicious adversary. For equal comparison, all costs for authentication processes, such as zero knowledge proof and message authentication, are omitted in the next section. Araki et al. proposed two versions of protocols: a protocol with information-theoretical security and a protocol with computational security. However, in the next section, we only perform comparison with the information-theoretical secure protocol. Below, we show the qualitative comparison between our proposed methods and conventional methods.

Table 1. Qualitative comparison of our proposed method with conventional methods

	Parameter n and k	Type of Adversary	Security
TUS Methods	$n \geq k$	Semi-honest	Information Theoretical
SPDZ 2 Method	$n = k$	Malicious	Computational
Araki et al's Method	$n = 3, k = 2$	Semi-honest	Information Theoretical or Computational

7.3 Quantitative Comparison

By using the computation of m number of inner-product sum operations, in Table 2 we show the communication and computation costs for the SPDZ 2, TUS5', and TUS5 methods when $n = k = 2$. In the case of $n = k$, because the TUS method could make use of additive secret sharing, computation of shares and reconstruction of secret information can be done by one process of addition.

In Table 3 we show the computation and communication costs of the Araki, TUS5', and TUS5 methods when $n = 3, k = 2$. Araki et al's method does not possess the loss resistance of the server; however, the TUS method still holds the loss resistance of the server.

In addition, in the inner-product sum computation, no repetition of computation is required. We assume a situation where the reconstructed result in the TUS5 method might be equal to 0, thus we include the process of performing r-times of ***Correction Phase***. We define the parameters used in the comparison as follows.

Definition of Parameters

- d_1 : Size of share from Shamir's secret sharing or XOR-based secret sharing
- d_2 : Size of share from SHE
- C_1 : Computational cost of Shamir's secret sharing
- C_2 : Computational cost for generating a set of *Multiplication Triple* using SHE
- C_3 : Computational cost of XOR-based secret sharing
- A : Computational cost of a single addition/subtraction
- M : Computational cost of a single multiplication/division

Parameter d_1 is usually almost the same size as the secret information, therefore, the size of the secret information is assumed as d_1. In contrast, d_2 is typically larger than the original secret. Therefore, $d_2 > d_1$. Moreover, C_1, C_3 are considerably smaller than C_2. Therefore, $C_1, C_3 \ll C_2$. In a secret sharing scheme, the computational cost of the distribution and the reconstruction process differs, but because both are computations on a polynomial (reconstruction is performed using the Lagrange Interpolation method), we consider the computation costs for computing one share and for reconstruction is C_1. A, M are computational costs for a single computation of addition/subtraction or multiplication/division, and $C_1, C_2, C_3 \gg A, M$. Process costs regarding the generation of random numbers and secret information is not included. Communication to all servers is assumed as a one-time communication of broadcast, and u represents the number of players.

In Tables 2 and 3 the process of generating a share on 1, as shown in Section 4, is labelled as ***Preprocessing0***, and the rest of the preprocessing process is labelled as ***Preprocessing1***. The ***Correction Phase*** is also separated and is evaluated as ***Correction***.

The secrecy computation of the TUS5 method is assumed to be as follows:

$$\begin{aligned}\overline{\left[\sum_{l=1}^{m}\delta_0(a_l b_l-\alpha_{1l}\beta_{1l})\right]}_i &= \sum_{l=1}^{m}\{\alpha_{0l}(a_l+\alpha_{1l})\times\beta_{0l}(b_l+\beta_{1l})\times\overline{\left[\frac{\delta_0}{\alpha_{0l}\beta_{0l}}\right]}_i\\ &-\alpha_{0l}(a_l+\alpha_{1l})\times\beta_{2l}\beta_{1l}\times\overline{\left[\frac{\delta_0}{\alpha_{0l}\beta_{2l}}\right]}_i\\ &-\beta_{0l}(b_l+\beta_{1l})\times\alpha_{2l}\alpha_{1l}\overline{\left[\frac{\delta_0}{\alpha_{2l}\beta_{0l}}\right]}_i\}\end{aligned}$$

$$\overline{\left[\sum_{l=1}^{m}\delta_2(-\alpha_{1l}\beta_{1l})\right]}_i=-\sum_{l=1}^{m}\{\alpha_{2l}\alpha_{1l}\times\beta_{2l}\beta_{1l}\times\overline{\left[\frac{\delta_2}{\alpha_{2l}\beta_{2l}}\right]}_i\}$$

Therefore, even in the TUS5 method, the number of shares on 1 required in order to perform secrecy computation is $4m$. However, considering r-times of the ***Correction Phase***, we assumed that it needed $(4m+2r)$ number of shares on 1.

If all preprocessing processes and the encryption process are also considered as preprocessing processes, from Table 2, the communication cost for the SPDZ 2 method includes the computation C_2, which requires expensive computation using Somewhat Homomorphic Encryption (SHE). Therefore, the computation cost for the SPDZ 2 method is the most expensive. In addition, communication costs depend a great deal on the relationship between d_2, d_1, but the TUS methods typically perform more small communication. However, the processes up to this point are performed by the ***Preprocessing Phase***. In the secrecy computation of the inner-product sum operation, in order to see the detailed breakdown of the operation, we show the computation and communication cost for one server and it is then multiplied by N for the total cost. In the SPDZ 2 method, the computation and communication costs per server are $m(3M+5A)$ and $m(2d_1)$, respectively. In contrast, in the TUS5' method, only the computation cost of $m(4M+3A)$ is required per server and no communication cost is required. Therefore, in the case where the communication process requires a lot of time, we can state that the TUS5' method is faster than the SPDZ 2 method.

Furthermore, when comparing with Araki et al's method, there are instances where the cost of both computation and communication of the TUS method are larger during the ***Preprocessing*** and ***Encryption Phase***. However, in the ***Computation Phase*** of inner product, Araki et al's method requires $m(3M+5A)$ of computation cost and $m(d_1)$ communication cost per server. In contrast, the TUS5' method requires only $m(4M+3A)$ computation cost without any communication cost. Therefore, in the case where communication takes some time, we can also state that the TUS5' method is faster than Araki et al's method. In the evaluation of the TUS method, we know it requires $(r+1)$-times of communication for the reconstruction process, but it is important to note that it does not depend on the parameter m. In contrast, the SPDZ 2 method and Araki et al.'s method each require $m(2d_1), m(d_1)$ of communication. Therefore, when the value of m is large, even if we include the process of correction, we can state that the TUS5 method will be faster.

8 Conclusion

In this paper, we supposed the number of servers/parties to be N, and realized a secure secrecy computation under the setting of $N < 2k - 1$. In addition, we also proposed a method that allows for the communication cost to be eliminated from the ***Computation Phase*** when computation without repetition is assumed, thus realizing a faster processing speed. Furthermore, our protocols realized secure secrecy computation without any limitation on the usability of the protocol by solving all three conditions proposed in the TUS methods. However, in the case of $n > k$, by keeping Condition 3, we maintained server loss-resistant of the system. Therefore, we can say that our protocols are the only protocols that allow for secure computation without any limitation for $n \geqq k$.

In a future study, we will perform a detailed experimental analysis on the processing speed of our protocol and consider an improved method of secrecy computation that is also secure against a malicious adversary.

Table 2. Comparison with conventional methods (for $n = k = 2$)

		SPDZ 2 Method	TUS 5' Method	TUS5 Method
Computation	Preprocessing0	mC_2	$4m(30\mathrm{M} + 40\mathrm{A})$	$(4m + 2r)(30\mathrm{M} + 40\mathrm{A})$
	Preprocessing1	0	$m(33M)$	$m(42M)$
	Encryption	$2mA$	$3m(M + A)$	$3m(2M + A)$
	Reconstruction	A	$2M + A$	$4M + \mathrm{A}$
	Inner product	$mN(3M + 5A)$	$mN(4M + 3A)$	$mN(6M + 2A) + 2A$
	Correction	0	0	$r(21M + 4A)$
Communication	Preprocessing0	$m(ud_2 + 3d_1)$	$4m(20d_1)$	$(4 + 2r)m(20d_1)$
	Preprocessing1	0	$m(17d_1)$	$m(26d_1)$
	Encryption	$4md_1$	$m(3d_1)$	$m(6d_1)$
	Reconstruction	$2d_1$	$4d_1$	$6d_1$
	Inner product	$mN(2d_1)$	0	$2d_1$
	Correction	0	0	$r8d_1$

Table 3. Comparison with conventional methods (for $n = 3, k = 2$)

		Araki et al.'s Method	TUS5' Method	TUS5 Method
Computa-tion	Preprocessing0	mNA	$4m(18C_1 + 34\mathrm{M} + 27\mathrm{A})$	$(4m + 2r)(18C_1 + 34\mathrm{M} + 27\mathrm{A})$
	Preprocessing1	0	$m(37M)$	$\mathrm{m}(46\mathrm{M})$
	Encryption	$m7A$	$3m(M + A)$	$3m(2M + A)$
	Reconstruction	A	$2M + \mathrm{A}$	$4M + \mathrm{A}$
	Inner product	$mN(3M + 5A)$	$mN(4M + 3A)$	$mN(6M + 2A) + 2A$
	Correction	0	0	$r(21C_1 + 25M + 6A)$
Commu-nication	Preprocessing0	$mN(d_1)$	$4m(22d_1)$	$(4m + 2r)(28d_1)$
	Preprocessing1	0	$m(17d_1)$	$m(26d_1)$
	Encryption	$12md_1$	$m(3d_1)$	$m(6d_1)$
	Reconstruction	$4d_1$	$4d_1$	$6d_1$
	Inner product	$mN(d_1)$	0	$2d_1$
	Correction	0	0	$r8d_1$

Appendix 1

4. Server S_i computes the following and sends all except $\overline{[0_\iota]}_i^{(2k)}, \overline{[0_\iota]}_{i+k}^{(2k)}$ to servers $S_{i+1(modk)}$.

- Server S_0computes the following :

– $\overline{[\alpha_1\alpha_2\beta_1\beta_2\gamma_1\gamma_2\lambda_0\lambda_1\lambda_2]'}_0^{(2k)} = \frac{\overline{[\alpha\beta\gamma\lambda_0\lambda_1\gamma\lambda_2]}_0^{(2k-1)}}{\alpha_0\beta_0\gamma_0} + \overline{[0_0]}_0^{(2k)}$

– $\overline{[\alpha_1\alpha_2\beta_1\beta_2\gamma_1\gamma_2\lambda_0\lambda_1\lambda_2]'}_3^{(2k)} = \frac{\overline{[\alpha\beta\gamma\lambda_0\lambda_1\gamma\lambda_2]}_3^{(2k-1)}}{\alpha_0\beta_0\gamma_0} + \overline{[0_0]}_3^{(2k)}$

- Server S_1 computes the following :

– $\overline{[\alpha_0\alpha_2\beta_0\beta_2\gamma_0\gamma_2\lambda_0\lambda_1\lambda_2]'}_1^{(2k)} = \frac{\overline{[\alpha\beta\gamma\lambda_0\lambda_1\gamma\lambda_2]}_1^{(2k-1)}}{\alpha_1\beta_1\gamma_1} + \overline{[0_1]}_1^{(2k)}$

– $\overline{[\alpha_0\alpha_2\beta_0\beta_2\gamma_0\gamma_2\lambda_0\lambda_1\lambda_2]'}_4^{(2k)} = \frac{\overline{[\alpha\beta\gamma\lambda_0\lambda_1\gamma\lambda_2]}_4^{(2k-1)}}{\alpha_1\beta_1\gamma_1} + \overline{[0_1]}_4^{(2k)}$

- Server S_2 computes the following :

– $\overline{[\alpha_0\alpha_1\beta_0\beta_1\gamma_0\gamma_1\lambda_0\lambda_1\lambda_2]'}_2^{(2k)} = \frac{\overline{[\alpha\beta\gamma\lambda_0\lambda_1\gamma\lambda_2]}_2^{(2k-1)}}{\alpha_2\beta_2\gamma_2} + \overline{[0_2]}_2^{(2k)}$

- $\overline{[\alpha_0\alpha_1\beta_0\beta_1\gamma_0\gamma_1\lambda_0\lambda_1\lambda_2]'}_5^{(2k)} = \frac{\overline{[\alpha\beta\gamma\lambda_0\lambda_1\gamma\lambda_2]}_5^{(2k-1)}}{\alpha_2\beta_2\gamma_2} + \overline{[0_2]}_5^{(2k)}$

5-1. Here we show the process by servers S_i on the first round. Servers S_i perform the following on the received $\overline{[\alpha_{-\iota}\beta_{-\iota}\gamma_{-\iota}\lambda_0\lambda_1\lambda_2]'}_i^{(2k)}$, $\overline{[\alpha_{-\iota}\beta_{-\iota}\gamma_{-\iota}\lambda_0\lambda_1\lambda_2]'}_{i+k}^{(2k)}$, and send the result of computation to servers $S_{i+1(modk)}$.

- Server S_0 computes the following :

- $\overline{[\alpha_1\beta_1\gamma_1\lambda_0\lambda_1\lambda_2]'}_2^{(2k)} = \frac{\overline{[\alpha_0\alpha_1\beta_0\beta_1\gamma_0\gamma_1\lambda_0\lambda_1\lambda_2]'}_2^{(2k)}}{\alpha_0\beta_0\gamma_0} + \overline{[0'_0]}_2^{(2k)} =$
  $\overline{[\alpha_1\beta_1\gamma_1\lambda_1\lambda_0\lambda_2]}_2^{(2k-1)} + \overline{[0_{20}]}_2^{(2k)}$
- $\overline{[\alpha_1\beta_1\gamma_1\lambda_0\lambda_1\lambda_2]'}_5^{(2k)} = \frac{\overline{[\alpha_0\alpha_1\beta_0\beta_1\gamma_0\gamma_1\lambda_0\lambda_1\lambda_2]'}_5^{(2k)}}{\alpha_0\beta_0\gamma_0} + \overline{[0'_0]}_5^{(2k)} =$
  $\overline{[\alpha_1\beta_1\gamma_1\lambda_1\lambda_0\lambda_2]}_5^{(2k-1)} + \overline{[0_{20}]}_2^{(2k)}$
- $\overline{[0_{20}]}_0^{(2k)} = \frac{\overline{[0_2]}_0^{(2k)}}{\alpha_0\beta_0\gamma_0} + \overline{[0'_0]}_0^{(2k)}$
- $\overline{[0_{20}]}_1^{(2k)} = \frac{\overline{[0_2]}_1^{(2k)}}{\alpha_0\beta_0\gamma_0} + \overline{[0'_0]}_1^{(2k)}$
- $\overline{[0_{20}]}_3^{(2k)} = \frac{\overline{[0_2]}_3^{(2k)}}{\alpha_0\beta_0\gamma_0} + \overline{[0'_0]}_3^{(2k)}$
- $\overline{[0_{20}]}_4^{(2k)} = \frac{\overline{[0_2]}_4^{(2k)}}{\alpha_0\beta_0\gamma_0} + \overline{[0'_0]}_4^{(2k)}$

- Server S_1 computes the following :

- $\overline{[\alpha_2\beta_2\gamma_2\lambda_0\lambda_1\lambda_2]'}_0^{(2k)} = \frac{\overline{[\alpha_1\alpha_2\beta_1\beta_2\gamma_1\gamma_2\lambda_0\lambda_1\lambda_2]'}_0^{(2k)}}{\alpha_1\beta_1\gamma_1} + \overline{[0'_1]}_0^{(2k)} =$
  $\overline{[\alpha_2\beta_2\gamma_2\lambda_0\lambda_1\lambda_2]}_0^{(2k-1)} + \overline{[0_{01}]}_0^{(2k)}$
- $\overline{[\alpha_2\beta_2\gamma_2\lambda_0\lambda_1\lambda_2]'}_3^{(2k)} = \frac{\overline{[\alpha_1\alpha_2\beta_1\beta_2\gamma_1\gamma_2\lambda_0\lambda_1\lambda_2]'}_3^{(2k)}}{\alpha_1\beta_1\gamma_1} + \overline{[0'_1]}_3^{(2k)} =$
  $\overline{[\alpha_2\beta_2\gamma_2\lambda_0\lambda_1\lambda_2]}_3^{(2k-1)} + \overline{[0_{01}]}_3^{(2k)}$
- $\overline{[0_{01}]}_1^{(2k)} = \frac{\overline{[0_0]}_1^{(2k)}}{\alpha_1\beta_1\gamma_1} + \overline{[0'_1]}_1^{(2k)}$
- $\overline{[0_{01}]}_2^{(2k)} = \frac{\overline{[0_0]}_2^{(2k)}}{\alpha_1\beta_1\gamma_1} + \overline{[0'_1]}_2^{(2k)}$
- $\overline{[0_{01}]}_4^{(2k)} = \frac{\overline{[0_0]}_4^{(2k)}}{\alpha_1\beta_1\gamma_1} + \overline{[0'_1]}_4^{(2k)}$
- $\overline{[0_{01}]}_5^{(2k)} = \frac{\overline{[0_0]}_5^{(2k)}}{\alpha_1\beta_1\gamma_1} + \overline{[0'_1]}_5^{(2k)}$

- Server S_2 computes the following :

- $\overline{[\alpha_0\beta_0\gamma_0\lambda_0\lambda_1\lambda_2]'}_1^{(2k)} = \frac{\overline{[\alpha_0\alpha_2\beta_0\beta_2\gamma_0\gamma_2\lambda_0\lambda_1\lambda_2]'}_1^{(2k)}}{\alpha_2\beta_2\gamma_2} + \overline{[0'_2]}_1^{(2k)} =$
  $\overline{[\alpha_2\beta_2\gamma_2\lambda_0\lambda_1\lambda_2]}_1^{(2k-1)} + \overline{[0_{12}]}_1^{(2k)}$

- $\overline{[\alpha_0\beta_0\gamma_0\lambda_0\lambda_1\lambda_2]'}_4^{(2k)} = \frac{\overline{[\alpha_0\alpha_2\beta_0\beta_2\gamma_0\gamma_2\lambda_0\lambda_1\lambda_2]'}_4^{(2k)}}{\alpha_2\beta_2\gamma_2} + \overline{[0'_2]}_4^{(2k)} =$
  $\overline{[\alpha_0\beta_0\gamma_0\lambda_0\lambda_1\lambda_2]}_4^{(2k-1)} + \overline{[0_{12}]}_4^{(2k)}$
- $\overline{[0_{12}]}_0^{(2k)} = \frac{\overline{[0_1]}_0^{(2k)}}{\alpha_2\beta_2\gamma_2} + \overline{[0'_2]}_0^{(2k)}$
- $\overline{[0_{12}]}_2^{(2k)} = \frac{\overline{[0_1]}_2^{(2k)}}{\alpha_2\beta_2\gamma_2} + \overline{[0'_2]}_2^{(2k)}$
- $\overline{[0_{12}]}_3^{(2k)} = \frac{\overline{[0_1]}_3^{(2k)}}{\alpha_2\beta_2\gamma_2} + \overline{[0'_2]}_3^{(2k)}$
- $\overline{[0_{12}]}_5^{(2k)} = \frac{\overline{[0_1]}_5^{(2k)}}{\alpha_2\beta_2\gamma_2} + \overline{[0'_2]}_5^{(2k)}$

5-2. Here, we show the process for the second round. In order to differentiate with the first round, we denote the value of $0'_i$ used as $0''_i$.

- Server S_0 computes the following :

- $\overline{[\lambda_1\lambda_0\lambda_2]'}_1^{(2k)} = \frac{\overline{[\alpha_0\beta_0\gamma_0\lambda_1\lambda_0\lambda_2]'}_1^{(2k)}}{\alpha_0\beta_0\gamma_0} + \overline{[0''_0]}_1^{(2k)} = \overline{[\lambda_1\lambda_0\lambda_2]}_1^{(2k-1)} + \overline{[0_{120}]}_1^{(2k)}$
- $\overline{[\lambda_1\lambda_0\lambda_2]'}_4^{(2k)} = \frac{\overline{[\alpha_0\beta_0\gamma_0\lambda_1\lambda_0\lambda_2]'}_4^{(2k)}}{\alpha_0\beta_0\gamma_0} + \overline{[0''_0]}_4^{(2k)} = \overline{[\lambda_1\lambda_0\lambda_2]}_4^{(2k-1)} + \overline{[0_{120}]}_4^{(2k)}$
- $\overline{[0_{120}]}_0^{(2k)} = \frac{\overline{[0_{12}]}_0^{(2k)}}{\alpha_0\beta_0\gamma_0} + \overline{[0''_0]}_0^{(2k)}$
- $\overline{[0_{120}]}_2^{(2k)} = \frac{\overline{[0_{12}]}_2^{(2k)}}{\alpha_0\beta_0\gamma_0} + \overline{[0''_0]}_2^{(2k)}$
- $\overline{[0_{120}]}_3^{(2k)} = \frac{\overline{[0_{12}]}_3^{(2k)}}{\alpha_0\beta_0\gamma_0} + \overline{[0''_0]}_3^{(2k)}$
- $\overline{[0_{120}]}_5^{(2k)} = \frac{\overline{[0_{12}]}_5^{(2k)}}{\alpha_0\beta_0\gamma_0} + \overline{[0''_0]}_5^{(2k)}$

- Server S_1 computes the following :

- $\overline{[\lambda_1\lambda_0\lambda_2]'}_2^{(2k)} = \frac{\overline{[\alpha_1\beta_1\gamma_1\lambda_1\lambda_0\lambda_2]'}_2^{(2k)}}{\alpha_1\beta_1\gamma_1} + \overline{[0''_1]}_2^{(2k)} = \overline{[\lambda_1\lambda_0\lambda_2]}_2^{(2k-1)} + \overline{[0_{201}]}_2^{(2k)}$
- $\overline{[\lambda_1\lambda_0\lambda_2]'}_5^{(2k)} = \frac{\overline{[\alpha_1\beta_1\gamma_1\lambda_1\lambda_0\lambda_2]'}_5^{(2k)}}{\alpha_1\beta_1\gamma_1} + \overline{[0''_1]}_5^{(2k)} = \overline{[\lambda_1\lambda_0\lambda_2]}_5^{(2k-1)} + \overline{[0_{201}]}_5^{(2k)}$
- $\overline{[0_{201}]}_0^{(2k)} = \frac{\overline{[0_{20}]}_0^{(2k)}}{\alpha_1\beta_1\gamma_1} + \overline{[0''_1]}_0^{(2k)}$
- $\overline{[0_{201}]}_1^{(2k)} = \frac{\overline{[0_{20}]}_1^{(2k)}}{\alpha_1\beta_1\gamma_1} + \overline{[0''_1]}_1^{(2k)}$
- $\overline{[0_{201}]}_3^{(2k)} = \frac{\overline{[0_{20}]}_3^{(2k)}}{\alpha_1\beta_1\gamma_1} + \overline{[0''_1]}_3^{(2k)}$
- $\overline{[0_{201}]}_4^{(2k)} = \frac{\overline{[0_{20}]}_4^{(2k)}}{\alpha_1\beta_1\gamma_1} + \overline{[0''_1]}_4^{(2k)}$

- Server S_2 computes the following :

- $\overline{[\lambda_1\lambda_0\lambda_2]'}_0^{(2k)} = \frac{\overline{[\alpha_2\beta_2\gamma_2\lambda_1\lambda_0\lambda_2]'}_0^{(2k)}}{\alpha_2\beta_2\gamma_2} + \overline{[0''_2]}_0^{(2k)} = \overline{[\lambda_1\lambda_0\lambda_2]}_0^{(2k-1)} + \overline{[0_{012}]}_0^{(2k)}$

$- \overline{[\lambda_1\lambda_0\lambda_2]'}_3^{(2k)} = \frac{\overline{[\alpha_2\beta_2\gamma_2\lambda_1\lambda_0\lambda_2]'}_3^{(2k)}}{\alpha_2\beta_2\gamma_2} + \overline{[0"_2]}_3^{(2k)} = \overline{[\lambda_1\lambda_0\lambda_2]}_3^{(2k-1)} + \overline{[0_{012}]}_3^{(2k)}$

$- \overline{[0_{012}]}_1^{(2k)} = \frac{\overline{[0_{01}]}_1^{(2k)}}{\alpha_2\beta_2\gamma_2} + \overline{[0'_2]}_1^{(2k)}$

$- \overline{[0_{012}]}_2^{(2k)} = \frac{\overline{[0_{01}]}_2^{(2k)}}{\alpha_2\beta_2\gamma_2} + \overline{[0'_2]}_2^{(2k)}$

$- \overline{[0_{012}]}_4^{(2k)} = \frac{\overline{[0_{01}]}_4^{(2k)}}{\alpha_2\beta_2\gamma_2} + \overline{[0'_2]}_4^{(2k)}$

$- \overline{[0_{012}]}_5^{(2k)} = \frac{\overline{[0_{01}]}_5^{(2k)}}{\alpha_2\beta_2\gamma_2} + \overline{[0'_2]}_5^{(2k)}$

6. Server $\boldsymbol{S_i}$ collects that $\overline{[\mathbf{0}_{J\ldots J+k-1}]}_{i+1}^{(2k)}$, $\overline{[\mathbf{0}_{J\ldots J+k-1}]}_{i+1+k}^{(2k)}$ corresponds to itself, adds them together and computes $\overline{[\boldsymbol{\lambda_1\lambda_0\lambda_2}]"}_{i+1}^{(2k)}$, $\overline{[\boldsymbol{\lambda_1\lambda_0\lambda_2}]"}_{j+1+k}^{(2k)}$.

- Server $\boldsymbol{S_0}$ computes the following :

$- \overline{[\lambda_1\lambda_0\lambda_2]"}_1^{(2k)} = \overline{[\lambda_1\lambda_0\lambda_2]}_1^{(2k-1)} + \overline{[0_{012}]}_1^{(2k)} + \overline{[0_{120}]}_1^{(2k)} + \overline{[0_{201}]}_1^{(2k)}$
$- \overline{[\lambda_1\lambda_0\lambda_2]"}_4^{(2k)} = \overline{[\lambda_1\lambda_0\lambda_2]}_4^{(2k-1)} + \overline{[0_{012}]}_4^{(2k)} + \overline{[0_{120}]}_4^{(2k)} + \overline{[0_{201}]}_4^{(2k)}$

- Server $\boldsymbol{S_1}$ computes the following :

$- \overline{[\lambda_1\lambda_0\lambda_2]"}_2^{(2k)} = \overline{[\lambda_1\lambda_0\lambda_2]}_2^{(2k-1)} + \overline{[0_{012}]}_2^{(2k)} + \overline{[0_{120}]}_2^{(2k)} + \overline{[0_{201}]}_2^{(2k)}$
$- \overline{[\lambda_1\lambda_0\lambda_2]"}_5^{(2k)} = \overline{[\lambda_1\lambda_0\lambda_2]}_5^{(2k-1)} + \overline{[0_{012}]}_5^{(2k)} + \overline{[0_{120}]}_5^{(2k)} + \overline{[0_{201}]}_5^{(2k)}$

- Server $\boldsymbol{S_2}$ computes the following :

$- \overline{[\lambda_1\lambda_0\lambda_2]"}_0^{(2k)} = \overline{[\lambda_1\lambda_0\lambda_2]}_0^{(2k-1)} + \overline{[0_{012}]}_0^{(2k)} + \overline{[0_{120}]}_0^{(2k)} + \overline{[0_{201}]}_0^{(2k)}$
$- \overline{[\lambda_1\lambda_0\lambda_2]"}_3^{(2k)} = \overline{[\lambda_1\lambda_0\lambda_2]}_3^{(2k-1)} + \overline{[0_{012}]}_3^{(2k)} + \overline{[0_{120}]}_3^{(2k)} + \overline{[0_{201}]}_3^{(2k)}$

7. Servers S_i distribute $0"_i$ using (k, N) Shamir's method, compute the following for $j = 0, \ldots, 2$, and send $R_{i,j}$ to servers S_j.

- Server S_0 computes the following and sends $R_{0,1}, R_{0,2}$ to S_1 and S_2, respectively.

$- R_{0,0} = a_{0,1} \times \overline{[\lambda_0\lambda_1\lambda_2]"}_1^{(2k)} + a_{0,4} \times \overline{[\lambda_0\lambda_1\lambda_2]"}_4^{(2k)} + \overline{[0"_0]}_0^{(k)}$
$- R_{0,1} = a_{1,1} \times \overline{[\lambda_0\lambda_1\lambda_2]"}_1^{(2k)} + a_{1,4} \times \overline{[\lambda_0\lambda_1\lambda_2]"}_4^{(2k)} + \overline{[0"_0]}_1^{(k)}$
$- R_{0,2} = a_{2,1} \times \overline{[\lambda_0\lambda_1\lambda_2]"}_1^{(2k)} + a_{2,4} \times \overline{[\lambda_0\lambda_1\lambda_2]"}_4^{(2k)} + \overline{[0"_0]}_2^{(k)}$

- Server S_1 computes the following and sends $R_{1,0}, R_{1,2}$ to S_0 and S_2, respectively.

$- R_{1,0} = a_{0,2} \times \overline{[\lambda_0\lambda_1\lambda_2]"}_2^{(2k)} + a_{0,5} \times \overline{[\lambda_0\lambda_1\lambda_2]"}_5^{(2k)} + \overline{[0"_1]}_0^{(k)}$
$- R_{1,1} = a_{1,2} \times \overline{[\lambda_0\lambda_1\lambda_2]"}_2^{(2k)} + a_{1,5} \times \overline{[\lambda_0\lambda_1\lambda_2]"}_5^{(2k)} + \overline{[0"_1]}_1^{(k)}$
$- R_{1,2} = a_{2,2} \times \overline{[\lambda_0\lambda_1\lambda_2]"}_2^{(2k)} + a_{2,5} \times \overline{[\lambda_0\lambda_1\lambda_2]"}_5^{(2k)} + \overline{[0"_1]}_2^{(k)}$

- Server S_2 computes the following and sends $R_{2,0}, R_{2,1}$ to S_0 and S_1, respectively.

− $R_{2,0} = a_{0,0} \times \overline{[\lambda_0\lambda_1\lambda_2]}"^{(2k)}_0 + a_{0,3} \times \overline{[\lambda_0\lambda_1\lambda_2]}"^{(2k)}_3 + \overline{[0"_2]}^{(k)}_0$
− $R_{2,1} = a_{1,0} \times \overline{[\lambda_0\lambda_1\lambda_2]}"^{(2k)}_0 + a_{1,3} \times \overline{[\lambda_0\lambda_1\lambda_2]}"^{(2k)}_3 + \overline{[0"_2]}^{(k)}_1$
− $R_{2,2} = a_{2,0} \times \overline{[\lambda_0\lambda_1\lambda_2]}"^{(2k)}_0 + a_{2,3} \times \overline{[\lambda_0\lambda_1\lambda_2]}"^{(2k)}_3 + \overline{[0"_2]}^{(k)}_2$

8. Servers S_i compute $\overline{[\lambda]}^{(k)}_i = \sum_{j=0}^{N-1} R_{j,i}$, and obtain $\overline{[\lambda]}^{(k)}_i$. However, note that $\lambda = \lambda_0\lambda_1\lambda_2$.

- Server S_0 computes the following:

− $\overline{[\lambda]}^{(k)}_0 = R_{0,0} + R_{1,0} + R_{2,0}$

- Server S_1 computes the following:

− $\overline{[\lambda]}^{(k)}_1 = R_{0,1} + R_{1,1} + R_{2,1}$

- Server S_2 computes the following:

− $\overline{[\lambda]}^{(k)}_2 = R_{0,2} + R_{1,2} + R_{2,2}$

Appendix 2

The adversary manipulated servers that it had taken over to perform different processes than as shown in Section 4.1, causing each server to obtain incorrect shares of $\overline{[\lambda"]}_i$. However, $\lambda" = \lambda + \delta_0$. The δ_0 can be known from information such as the difference from the correct process performed by the dishonest servers. For example, in step 7 of Section 4.1, when different $R'_{i,j}$ than $R_{i,j}$ is sent to server S_j, δ_0 can be solved if the difference is assumed to be the share of each server. Shares of each server shown in Section 4.1 are constructed by adding all the values in Steps 7 and 8. However, from the proof of security against *Adversary 2*, we can state that the adversary will not be able to learn about λ and $\lambda"$.Therefore, as shown below, it is impossible to obtain result of $\lambda"$ where $\lambda" = \lambda\delta_0$ if dishonest processes are performed in the protocol shown in Section 4.1. For example, for ease of understanding, suppose that $\overline{[\mu_1]}_i$ are correct and the information $\lambda_i/\mu_{1,i}$ given by servers that had been manipulated by the adversary is $\rho_0\lambda_i/\mu_{1,i}$. By this, the reconstructed value of shares $\overline{[\lambda']}_i$ obtained from equation (3) will be $\rho_0\lambda$. Therefore, if the equation shown below is established, equations (3) and (4) will also be established.

$$\lambda + \delta = \rho_0\lambda \quad (5)$$

In order to establish equation (5), the adversary needs to learn about $\rho_0 = 1 + \delta/\lambda$. However, from the proof of security against *Adversary 2*, because the adversary cannot learn about λ, the adversary will also not be able to learn about ρ_0.Therefore, the adversary will not be able to manipulate the verification from equations (3) and (4). In addition, the same can be said even if $\mu_1" = \mu_1 + \delta_1$ and $\lambda_i/\mu_{1,i} = \rho_0\lambda_i/\rho_1\mu_{1,i}$.

References

1. Araki T., Furukawa J., Lindell Y., Nof A., Ohara K.: High throughput semi-honest secure three-party computation with an honest majority. Proceedings of CCS 2016. pp. 805-817. ACM, Vienna, Austria (2016)
2. Beaver D.: Efficient Multiparty Protocols using Circuit Randomization. In: Feigenbaum J. (eds) Advances in Cryptology — CRYPTO 1991. LNCS, vol 576, pp. 420-432. Springer, Berlin, Heidelberg (1991)
3. Beaver D., Micali S., Rogaway P.: The round complexity of secure protocols. Proceedings of the 22nd STOC. pp. 503-513. ACM, Baltimore, Maryland, USA (1990)
4. 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)
5. Ben-Or M., Goldwasser S., Wigderson A.: Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation. Proceedings of the 20th STOC. pp. 1-10. ACM, New York, NY, USA (1988)
6. Blakley G. R.: Safeguarding Cryptographic Keys. Proceedings of the AFIPS 1979 National Computer Conference, Vol. 48, pp. 313-317 (1979)
7. Brakerski Z., Gentry C., Vaikuntanathan V.: (Leveled) Fully Homomorphic Encryption without Bootstrapping. Proceedings of the 3rd ITCS. pp. 309-325. ACM, Cambridge, MA, USA (2009)
8. 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)
9. Chaum D., Crépeau C., Damgård I.: Multiparty Unconditionally Secure Protocols. Proceedings of the 20th STOC. pp. 11-19. ACM, Chicago, Illinois, USA (1988)
10. Cleve R.: Limits on the Security of Coin Flips When Half the Processors are Faulty. Proceedings of the 18th STOC. pp. 364-369. ACM, Berkeley, California, USA (1986)
11. 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)
12. 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)
13. 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. LNCS, vol. 8134. Springer, Berlin, Heidelberg (2013)
14. Gentry C.: A Fully Homomorphic Encryption Scheme, Ph.D Thesis, Stanford University, Stanford, CA, USA (2009)
15. Iwamura K., Tokita K.: Fast Secure Computation based on Secret Sharing Scheme for n<2k-1. Proceedings of the 4th MobiSecServ. pp. 1-5. IEEE, Miami Beach, FL (2018)
16. Kurihara J., Kiyomoto S., Fukushima K., Tanaka T.: A new (k,n)-threshold secret sharing scheme and its extension. In Proceedings of ISC 2008. pp. 455-470, Springer, Berlin, Heidelberg (2008)
17. Mohd Kamal A. A. A, Iwamura K.: Conditionally Secure Multiparty Computation using Secret Sharing Scheme for n<2k-1. Proceedings of the 15th PST. pp. 225-230. IEEE, Calgary, AB, Canada (2017)

18. Shamir A.: How to Share a Secret. Communications of the ACM, Vol. 22, Issue 11, pp. 612-613. ACM, New York, NY, USA (1979)
19. Sharemind, Cybernetica. https://sharemind.cyber.ee.
20. Shingu T., Iwamura K., Kaneda K.: Secrecy Computation without Changing Polynomial Degree in Shamir's (k, n) Secret Sharing Scheme. Proceedings of the 13th ICETE. Vol. 1, pp. 89-94. DCNET, Lisbon, Portugal (2016)
21. van Dijk M., Gentry C., Halevi S., Vaikuntanathan V.: Fully Homomorphic Encryption over the Integers. In: Gilbert H. (eds) Advances in Cryptology – EUROCRYPT 2010. LNCS, Vol. 6110, pp. 24-43. Springer, Berlin, Heidelberg (2010)
22. 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. LNEE, Vol. 330, pp. 107-112. Springer, Berlin, Heidelberg (2015)
23. Yao A. C.: Protocols for Secure Computations. Proceedings of the *23rd SFCS. p*p. 160-164. IEEE Computer Society, Chicago, IL, USA (1982)
24. Yao A. C: How to Generate and Exchange Secrets. Proceedings of the 27th SFCS. pp. 162-167. IEEE Computer Society, Washington, DC, USA (1986)