Secure Computation by Secret Sharing using Input Encrypted with Random Number

Keiichi Iwamura and Ahmad Akmal Aminuddin Mohd Kamal
Department of Electrical Engineering, Tokyo University of Science, Tokyo, Japan

Keywords: Secure Computation, Multiparty Computation, Secret Sharing, n<2k-1, Information-theoretical Security, Fast Computation.

Abstract: Typically, unconditionally secure computation using a (k,n) threshold secret sharing is considered impossible when $n < 2k - 1$. Therefore, in our previous work, we first took the approach of finding the conditions required for secure computation under the setting of $n < 2k - 1$ and showed that secure computation using a (k,n) threshold secret sharing can be realized with a semi-honest adversary under the following three preconditions: (1) the result of secure computation does not include 0; (2) random numbers reconstructed by each server are fixed; and (3) each server holds random numbers unknown to the adversary and holds shares of random numbers that make up the random numbers unknown to the adversary. In this paper, we show that by leaving condition (3), secure computation with information-theoretic security against a semi-honest adversary is possible with $k \leq n < 2k - 1$. In addition, we clarify the advantage of using secret information that has been encrypted with a random number as input to secure computation. One of the advantages is the acceleration of the computation time. Namely, we divide the computation process into a preprocessing phase and an online phase and shift the cost of communication to the preprocessing phase. Thus, for computations such as inner product operations, we realize a faster online phase, compared with conventional methods.

1 INTRODUCTION

Recently, with advancements in big data and the Internet of Things (IoT), there has been a 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, to solve this problem, there is a need for a technology that allows their information to be used without infringing their privacy. One of the available technologies that could permit this is called secure computation, wherein a set of parties with private inputs wish to compute a joint function of their inputs, without revealing anything but the output. There are two main approaches for constructing secure computation protocols:

- Secret sharing (Araki et al, 2016; Ben-Or et al, 1988; Cramer et al., 2000; Kamal et al., 2017; Shingu et al., 2016; Tokita et al., 2018)
- Homomorphic encryption (Bendlin et al., 2011; Brakerski et al., 2012; Damgård et al., 2012; Damgård et al., 2013; Smart et al., 2010; van Dijk et al., 2010)

However, homomorphic encryption is known to be expensive in terms of computational cost, and therefore it requires a much longer computation time. Therefore, approaches with lower computational cost are preferable to homomorphic encryption, when considering the utilization of big data and IoT data.

Secret sharing is a method in which a single input is divided into multiple shares, which are then distributed to multiple users. A known example of a secret sharing is the (k,n) threshold secret sharing, where an input s is divided into n number of shares. The original input s can only be reconstructed or retrieved from a threshold k number of shares, but any $k - 1$ or smaller number of shares reveals nothing about the original input. Therefore, when $n > k$, a (k,n) threshold secret sharing can realize resistance toward loss of at most $n - k$ servers.

However, secure computation using a secret sharing can perform secure addition and subtraction easily, but this is not so in the case of multiplication. For example, in the (k,n) threshold secret sharing proposed by Shamir (Shamir, 1979), the degree of a polynomial changes from $k - 1$ to $2k - 2$ for each multiplication of polynomials. To restore the

Iwamura, K. and Kamal, A.
Secure Computation by Secret Sharing using Input Encrypted with Random Number.
DOI: 10.5220/0010552305400547
In *Proceedings of the 18th International Conference on Security and Cryptography (SECRYPT 2021)*, pages 540-547
ISBN: 978-989-758-524-1

multiplication result, the number of shares required increases from k to $2k-1$. Typically, unconditionally secure computation is considered impossible when $n < 2k-1$. Therefore, for most information-theoretically secure computations using a secret sharing, it is assumed that $n \geq 2k-1$.

Conversely, there is little research on secure computation using a secret sharing with $k \leq n < 2k-1$. Methods such as the SPDZ method (Damgård et al., 2012) have been proposed to combine secret sharing with homomorphic encryption to solve this problem. However, this approach only realizes computational security, not information-theoretic security. Our research focuses on realizing secure computation of secret sharing with $k \leq n < 2k-1$; however, we took an approach of first finding the conditions required to realize information-theoretic security against semi-honest adversaries, and then find another method for easing the conditions required. The result is that we proposed a secure computation known as the TUS (**T**okyo **U**niversity of **S**cience) methods (Shingu et al., 2016; Kamal et al., 2017; Tokita et al., 2018) with the following three conditions.

(1) The computation result does not include 0.
(2) Random numbers reconstructed by each server are fixed.
(3) Each server holds random numbers unknown to the adversary, and the shares of random numbers that make up the random numbers unknown to the adversary.

The characteristics of all TUS methods are that the input of secure computation is first encrypted with a random number before being used for secure computation. However, the TUS methods incur high computational costs and cannot realize fast computation.

Therefore, in this study, we introduce a method to improve the TUS methods to provide information-theoretic security against a semi-honest adversary in a particular condition with a faster computation speed. Namely, we divide the computation process into *preprocessing* and *online phases* and shift all computations related to random numbers to the preprocessing phase. Thus, we propose a method in which the communication cost can be totally eliminated from the online phase (known as the TUS 4 method). Next, we reduce the conditions required in the TUS methods and show that the TUS methods can be realized securely with only Condition (3). In addition, we also discuss the merits and demerits of the TUS methods.

Our proposed secure computation model is based on a client/server model where any number of clients can send shares of their inputs to n servers that perform the computation for the clients and return the results to them without learning anything.

The remainder of this paper is organized as follows: in Section 2, we present related works; in Section 3, we explain the TUS 4 method, in Section 4 we explain the discussion on the merits and demerits of encrypting inputs with random numbers and discuss each conditions of the TUS methods. Finally, in Section 5, we perform an experimental evaluation and show that the TUS 4 method can realize an overwhelmingly fast computation speed.

2 RELATED WORKS

2.1 SPDZ Method

Damgård et al. proposed a secure multiparty computation called SPDZ methods (Damgård et al., 2012; Damgård et al., 2013) that utilizes a somewhat homomorphic encryption and is secure against a dishonest majority with $n = k$. SPDZ consists of a preprocessing and an online phase. SPDZ ensures the confidentiality of the inputted secrets by using an additive secret sharing. Moreover, multiplication is based on Beaver's circuit randomization (Beaver, 1991), where shares of random numbers $\langle a\rangle, \langle b\rangle, \langle c\rangle$, called a multiplicative triple, that satisfy $a \cdot b = c$ are used. However, the construction of a multiplication triple requires a fully homomorphic encryption (Gentry, 2010) where the computation cost is high, thus significantly increasing the overall process time. Please refer to Section 2.1 in (Iwamura et al., 2021) for the detailed protocol of SPDZ method.

2.2 Araki et al.'s Method

Typically, in a secure multiparty computation, the cost of communication between servers can affect the overall processing speed more than the actual cost of computation. Therefore, Araki et al. proposed a method for rapid secure computation under the parameters $n = 3, k = 2$, which requires only one communication per multiplication (Araki et al., 2016). Please refer to Section 2.2 in (Iwamura et al., 2021) for the detailed multiplication protocol.

Note that it is usually not considered a problem, even if communication is required in the preprocessing phase. In addition, secure computation of addition in Araki et al.'s method is performed locally, where the shares are added together.

2.3 (k, n) Threshold Secret Sharing

A (k, n) threshold secret sharing satisfies both the conditions stated below:

- Any $k-1$, or less, number of shares will reveal nothing about the input s.
- Any k and above number of shares will allow for the reconstruction of the input s.

The classic methods of the (k, n) threshold secret sharing are Shamir's (k, n) threshold secret sharing *(Shamir's method)* proposed in (Shamir, 1979) and the XOR-based method *(XOR method)* proposed in (Kurihara et al., 2008). In our protocol, Shamir's method was used, and all computations were performed in modulus p. The shares of input s, are represented by $\overline{[s]}_i$.

2.4 The TUS Methods

Shingu et al. proposed a 2-inputs-1-output computation called the TUS 1 method (Shingu et al., 2016), where the input is first encrypted with a random number. The encrypted input is momentarily restored as a scalar value, and multiplication is realized using the *scalar value* × *polynomial* approach to prevent an increase in the polynomial degree. However, the TUS 1 method is not capable of coping with computations that require a combination of addition/subtraction and multiplication/division.

Kamal et al. introduced an improved method called the TUS 2 method, where the computation involving a combination of addition/subtraction and multiplication/division can also be performed securely (Kamal et al., 2017). This method was proven to be secure under the three aforementioned conditions. However, the TUS 2 method incurs significantly more computational cost.

Therefore, Tokita et al. realizes a more efficient method for computation, known as the TUS 3 method (Tokita et al., 2018), where XOR method of secret sharing is used. The TUS 3 method also proposed a way to ease one of the conditions (known as the TUS 3' method), wherein there are no longer limitation for the inputs of computation; however, the three conditions still remain.

Moreover, information in Condition (3) is defined as follow, where $\varepsilon = \prod_{j=0}^{k-1} \varepsilon_j$ is defined as a random number unknown to the adversary.

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

Here, $\overline{[\varepsilon]}_j$ is a share of ε and $[\varepsilon]_j$ is a set of share containing shares of ε and ε_j (used to compute ε).

3 THE TUS 4 METHOD

By dividing the computation process into the ***preprocessing phase*** and ***online phase***, it allows us to shift parts of the computation that require communication to the preprocessing phase. This significantly reduce the cost of communication in the online phase and speed up the entire process.

Here, instead of the simple product-sum operation of $ab + c$, we present a solution for computing the following extended product-sum operation:

$$\sum_{i=1}^{l} \left(a_{1,i} a_{2,i} \dots a_{m_i,i}\right)$$

This allows multiple computations to be performed at once instead of only one computation of $ab + c$ each time. However, a single product-sum operation can also be realized by setting the parameters $l = 2, m_1 = 2, m_2 = 1$.

Typically, because Equations (1) and (2) hold, any computation of $(a_1 a_2 \dots a_m)$ can be computed from $(a_1+1), (a_2+1), \dots, (a_m+1)$.

$$a_1 a_2 = (a_1+1)(a_2+1) - (a_1+1) - (a_2+1) + 1 \quad (1)$$

$$a_1 a_2 \dots a_m = (a_1 \dots a_{m-1})(a_m+1) - (a_1 \dots a_{m-1}) \quad (2)$$

In addition, extending Equation (2) will result in the following.

$$a_1 a_2 \dots a_m = \sum_{i=0}^{mCm-i} (-1)^i \prod_{j'=1}^{m-i} \left(a_{j'} + 1\right)$$

However, j' is an element of the combination of choosing the $m-i$ number from m number of $\left(a_j + 1\right)$. For example, the following holds when $m = 3,4$.

$$\begin{aligned} a_1 a_2 a_3 = {} & (a_1+1)(a_2+1)(a_3+1) \\ & - \{(a_1+1)(a_2+1) \\ & + (a_1+1)(a_3+1) \\ & + (a_2+1)(a_3+1)\} + \{(a_1+1) \\ & + (a_2+1) + (a_3+1)\} - 1 \end{aligned}$$

$$\begin{aligned} a_1 a_2 a_3 a_4 = {} & (a_1+1)(a_2+1)(a_3+1)(a_4+1) \\ & - \{(a_1+1)(a_2+1)(a_4+1) \\ & + (a_1+1)(a_3+1)(a_4+1) \\ & + (a_2+1)(a_3+1)(a_4+1) \\ & + (a_1+1)(a_2+1)(a_3+1)\} \\ & + \{(a_1+1)(a_2+1) \\ & + (a_1+1)(a_3+1) \\ & + (a_1+1)(a_4+1) \\ & + (a_2+1)(a_3+1) \\ & + (a_2+1)(a_4+1) \\ & + (a_3+1)(a_4+1)\} - \{(a_1+1) \\ & + (a_3+1) + (a_2+1) + (a_4+1)\} \\ & + 1 \end{aligned}$$

Therefore, when $l = 2, m_1 = 3, m_2 = 4$, $a_1 a_2 a_3$ will be $a_{1,1} a_{2,1} a_{3,1}$, and $a_1 a_2 a_3 a_4$ will be

$a_{1,2}a_{2,2}a_{3,2}a_{4,2}$, thus allowing the following to be computed.

$$a_{1,1}a_{2,1}a_{3,1} + a_{1,2}a_{2,2}a_{3,2}a_{4,2}$$

In the TUS 4 method, inputs $a_{1,i}, a_{2,i}, \dots . a_{m_i,i}$ must be within the modulus p and be a number under $p-2$. In addition, all random numbers used were uniformly distributed and did not include the value 0. Moreover, all other values belong to $GF(p)$, and all computations are performed with the modulus p. Moreover, we assume that communication between players and servers is secure. In addition, random numbers not known to the adversary shown in Condition (3) are assumed to be $\varepsilon_i^{(h)}(h = 1, \dots 8)$, and shares of all random numbers $\varepsilon_{i,j}^{(h)}$ used to construct $\varepsilon_i^{(h)}$ are prepared and stored in the servers in advance. Moreover, k servers in Steps 4 and 5 shown in the preprocessing phase are chosen in advance from n servers. Below, for ease of understanding, we show our protocol for $m_i = 3$; however, it can also be extended to any m_i.

Preprocessing Phase.

1. Dealer D generates k random numbers $b_{(1,i),0}, b_{(1,i),1}, \dots, b_{(1,i),k-1}$ with respect to inputs $a_{1,i}(i = 1, \dots, l)$, computes $b_{1,i} = \prod_{j=0}^{k-1} b_{(1,i),j}$, and sends $b_{(1,i),j}$ to server S_j.
2. Dealer D performs Step 1. On $a_{2,i}, a_{3,i}$.
3. Dealer D sends $b_{g,i}$ to User $U_{g,i}$ $(g = 1,2,3)$.
4. Server S_j $(j = 0, \dots, k-1)$ collects each $\overline{[\varepsilon_{i,j}^{(h)}]}_u (u = 0, \dots, k-1, h = 1, \dots, 8)$ and reconstructs $\varepsilon_{i,j}^{(h)}$.
5. Server S_j generates d_j, computes the following, and sends it to one of the servers (Here, we assume the server to be server S_0).

$$\frac{d_j}{b_{(1,i),j}b_{(2,i),j}b_{(3,i),j}\varepsilon_{i,j}^{(1)}}, \frac{d_j}{b_{(1,i),j}b_{(2,i),j}\varepsilon_{i,j}^{(2)}}, \frac{d_j}{b_{(1,i),j}b_{(3,i),j}\varepsilon_{i,j}^{(3)}},$$

$$\frac{d_j}{b_{(2,i),j}b_{(3,i),j}\varepsilon_{i,j}^{(4)}}, \frac{d_j}{b_{(1,i),j}\varepsilon_{i,j}^{(5)}}, \frac{d_j}{b_{(2,i),j}\varepsilon_{i,j}^{(6)}}, \frac{d_j}{b_{(3,i),j}\varepsilon_{i,j}^{(7)}}, \frac{d_j}{\varepsilon_{i,j}^{(8)}}$$

6. Server S_0 computes the following and sends to all servers. $(i = 1, \dots, l)$.

$$\frac{d}{b_{1,i}b_{2,i}b_{3,i}\varepsilon_i^{(1)}} = \prod_{j=0}^{k-1} \frac{d_j}{b_{(1,i),j}b_{(2,i),j}b_{(3,i),j}\varepsilon_{i,j}^{(1)}},$$

$$\frac{d}{b_{1,i}b_{2,i}\varepsilon_i^{(2)}} = \prod_{j=0}^{k-1} \frac{d_j}{b_{(1,i),j}b_{(2,i),j}\varepsilon_{i,j}^{(2)}},$$

$$\frac{d}{b_{1,i}b_{3,i}\varepsilon_i^{(3)}} = \prod_{j=0}^{k-1} \frac{d_j}{b_{(1,i),j}b_{(3,i),j}\varepsilon_{i,j}^{(3)}},$$

$$\frac{d}{b_{2,i}b_{3,i}\varepsilon_i^{(4)}} = \prod_{j=0}^{k-1} \frac{d_j}{b_{(2,i),j}b_{(3,i),j}\varepsilon_{i,j}^{(4)}},$$

$$\frac{d}{b_{1,i}\varepsilon_i^{(5)}} = \prod_{j=0}^{k-1} \frac{d_j}{b_{(1,i),j}\varepsilon_{i,j}^{(5)}},$$

$$\frac{d}{b_{2,i}\varepsilon_i^{(6)}} = \prod_{j=0}^{k-1} \frac{d_j}{b_{(2,i),j}\varepsilon_{i,j}^{(6)}},$$

$$\frac{d}{b_{3,i}\varepsilon_i^{(7)}} = \prod_{j=0}^{k-1} \frac{d_j}{b_{(3,i),j}\varepsilon_{i,j}^{(7)}},$$

$$\frac{d}{\varepsilon_i^{(8)}} = \prod_{j=0}^{k-1} \frac{d_j}{\varepsilon_{i,j}^{(8)}}$$

7. All servers S_j compute and hold the following.

$$\overline{\left[\frac{d}{b_{1,\iota}b_{2,\iota}b_{3,\iota}}\right]}_j = \frac{d}{b_{1,i}b_{2,i}b_{3,i}\varepsilon_i^{(1)}} \times \overline{[\varepsilon_\iota^{(1)}]}_j,$$

$$\overline{\left[\frac{d}{b_{1,\iota}b_{2,\iota}}\right]}_j = \frac{d}{b_{1,i}b_{2,i}\varepsilon_i^{(2)}} \times \overline{[\varepsilon_\iota^{(2)}]}_j,$$

$$\overline{\left[\frac{d}{b_{1,\iota}b_{3,\iota}}\right]}_j = \frac{d}{b_{1,i}b_{3,i}\varepsilon_i^{(3)}} \times \overline{[\varepsilon_\iota^{(3)}]}_j,$$

$$\overline{\left[\frac{d}{b_{2,\iota}b_{3,\iota}}\right]}_j = \frac{d}{b_{2,i}b_{3,i}\varepsilon_i^{(4)}} \times \overline{[\varepsilon_\iota^{(4)}]}_j,$$

$$\overline{\left[\frac{d}{b_{1,\iota}}\right]}_j = \frac{d}{b_{1,i}\varepsilon_i^{(5)}} \times \overline{[\varepsilon_\iota^{(5)}]}_j,$$

$$\overline{\left[\frac{d}{b_{2,\iota}}\right]}_j = \frac{d}{b_{2,i}\varepsilon_i^{(6)}} \times \overline{[\varepsilon_\iota^{(6)}]}_j,$$

$$\overline{\left[\frac{d}{b_{3,\iota}}\right]}_j = \frac{d}{b_{3,i}\varepsilon_i^{(7)}} \times \overline{[\varepsilon_\iota^{(7)}]}_j,$$

$$\overline{[d]}_j = \frac{d}{\varepsilon_i^{(8)}} \times \overline{[\varepsilon_\iota^{(8)}]}_j$$

Encryption Phase.

1. User $U_{g,i}$ compute $b_{g,i}(a_{g,i} + 1) = b_{g,i} \times (a_{g,i} + 1)$ in regard to its input $a_{g,i}$ and send to all servers. $(g = 1,2,3)$.

Online Phase.

1. All servers S_j compute the following.

$$\begin{aligned}
&\overline{\left[d\sum\nolimits_{\iota=1}^{l}(a_{1,\iota}a_{2,\iota}a_{3,\iota})\right]_j}\\
&\quad=\sum\nolimits_{i=1}^{l}\Bigg\{b_{1,i}(a_{1,i}+1)\\
&\quad\times b_{2,i}(a_{2,i}+1)\times b_{3,i}(a_{3,i}+1)\\
&\quad\times\overline{\left[\frac{d}{b_{1,\iota}b_{2,\iota}b_{3,\iota}}\right]_j}\\
&\quad- b_{1,i}(a_{1,i}+1)\times b_{2,i}(a_{2,i}+1)\\
&\quad\times\overline{\left[\frac{d}{b_{1,\iota}b_{2,\iota}}\right]_j}\\
&\quad- b_{1,i}(a_{1,i}+1)\times b_{3,i}(a_{3,i}+1)\\
&\quad\times\overline{\left[\frac{d}{b_{1,\iota}b_{3,\iota}}\right]_j}\\
&\quad- b_{2,i}(a_{2,i}+1)\times b_{3,i}(a_{3,i}+1)\\
&\quad\times\overline{\left[\frac{d}{b_{2,\iota}b_{3,\iota}}\right]_j}+b_{1,i}(a_{1,i}+1)\times\overline{\left[\frac{d}{b_{1,\iota}}\right]_j}\\
&\quad+b_{2,i}(a_{2,i}+1)\times\overline{\left[\frac{d}{b_{2,\iota}}\right]_j}\\
&\quad+b_{3,i}(a_{3,i}+1)\times\overline{\left[\frac{d}{b_{3,\iota}}\right]_j}-\overline{[d]_j}\Bigg\}
\end{aligned}$$

Reconstruction Phase.

1. The player collects $\overline{\left[d\sum_{\iota=1}^{l}(a_{1,\iota}a_{2,\iota}a_{3,\iota})\right]_j}$, d_j from k servers S_j, reconstructs $d\sum_{i=1}^{l}(a_{1,i}a_{2,i}a_{3,i}), d$, and computes the following.

$$\frac{d\sum_{i=1}^{l}(a_{1,i}a_{2,i}a_{3,i})}{d}=\sum\nolimits_{i=1}^{l}(a_{1,i}a_{2,i}a_{3,i})$$

3.1 Security of the TUS 4 Method

In the proposed method of computation with $t(=\sum_{i=1}^{l}m_i)$ inputs and one output, if $t-1$ inputs and the output are leaked to the adversary, the remaining input can also be leaked. Similarly, when all t inputs are known to the adversary, the output can also be leaked to the adversary. Therefore, we consider only the following two types of adversaries:

Adversary 1 has information on $t-2$ inputs and one output. *Adversary 1* has information on the inputs (and the random number used to encrypt them) and the information needed to reconstruct the output. In addition, the adversary also has knowledge of information from $k-1$ server and attempts to learn the remaining two inputs.

Adversary 2 has information on the $t-1$ inputs. *Adversary 2* has information on the inputs (and the random numbers used to encrypt them). In addition, the adversary also has knowledge of information from $k-1$ servers and attempts to learn the remaining one input or output of the computation.

Proof of Security of the Preprocessing Phase. Because our proposed method assumes a semi-honest adversary, Dealer D performs Steps 1–3 correctly and privately, and sends it to each server and user. Server S_j $(j=0,\dots,k-1)$ reconstruct $\varepsilon_{i,j}^{(h)}$ (which is used to construct $\varepsilon_i^{(h)}$) in Step 4; however, $\varepsilon_i^{(h)}$ will not leak from $k-1$ servers. In Step 5, server S_j sends its computed values to server S_0; however, Adversaries 1 and 2 cannot decompose each individual random number from this information. Therefore, Adversaries 1 and 2 will not be able to learn $d, b_{1,i}, b_{2,i}, b_{3,i}, \varepsilon_i^{(1)},\dots,\varepsilon_i^{(8)}$. In Step 6, Server S_0 multiplied all values and broadcast the result; however, Adversaries 1 and 2 cannot decompose the information to learn each individual random number. Moreover, because the random number unknown to the adversary shown in Condition (3) is used to compute shares in Step 7, the shares held by each server can be computed securely.

Therefore, for example, the following statement is true. The same is true for the remaining shares. Here, $H(x)$ represents the entropy of x.

$$H\left(\overline{\left[\frac{d}{b_{1,\iota}b_{2,\iota}b_{3,\iota}}\right]_j}\right)=H\left(\overline{\left[\frac{d}{b_{1,\iota}b_{2,\iota}b_{3,\iota}}\right]_j}\,\middle|\,\frac{d}{b_{1,i}b_{2,i}b_{3,i}\varepsilon_i^{(1)}}\right)$$

Proof of Security of the Encryption Phase. Since the input is less than $p-2$, even if one is added to the input, 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 remainder of the inputs.

$$H(a_{g,i})=H(a_{g,i}|b_{g,i}(a_{g,i}+1))$$

Proof of Security of the Online Phase.

Security against Adversary 1

Assume that the adversary has information on all inputs except $a_{1,1}, a_{2,1}$. He/she also has information from $k-1$ servers and the result of the computation. Furthermore, *Adversary 1* has information from Step 5 of the preprocessing phase, and also learns $b_{g,i}(a_{g,i}+1)$ from the online phase. In addition, *Adversary 1* learns $d\sum_{i=1}^{l}(a_{1,i}a_{2,i}a_{3,i}), d$, and $\sum_{i=1}^{l}(a_{1,i}a_{2,i}a_{3,i})$ from the reconstruction phase. Consequently, *Adversary 1* has the following information (however, g', i' excludes 1,1 and 2,1; $\varepsilon_1^{(i,j)}$ means that it relates to the same value; $\varepsilon_*^{(h)}$ is used when $i>1$). By using this information, *Adversary 1* tries to learn about inputs $a_{1,1}, a_{2,1}$.

$$B = \left\{ d, a_{g',i'}, b_{g',i'}, b_{1,1}(a_{1,1}+1), b_{2,1}(a_{2,1}+1), \frac{1}{b_{1,1}b_{2,1}\varepsilon_1^{(1,2)}}, \frac{1}{b_{1,1}\varepsilon_1^{(3,5)}}, \frac{1}{b_{2,1}\varepsilon_1^{(4,6)}}, \varepsilon_1^{(7,8)}, \varepsilon_*^{(h)} \right\}$$

However, because each $\varepsilon_i^{(h)}$ is independent, $b_{1,1}, b_{2,1}$ will not leak. In addition, because $\varepsilon_i^{(h)}$ is not used in the computation, it does not affect the computation. Therefore, *Adversary 1* will not be able to learn $a_{1,1}, a_{2,1}$. Therefore, the following are true:

$$H(a_{1,1}) = H(a_{1,1}|B), \qquad H(a_{2,1}) = H(a_{2,1}|B)$$

The same argument remains valid even for combination of inputs other than $a_{1,1}, a_{2,1}$.

In addition, in the aforementioned protocol, we assumed $m_i = 3$ for ease of understanding. However, even in the case other than $m_i = 3$, values that are not leaked are not included in *B*; therefore, the same is true for any m_i. Therefore, the TUS 4 method is information-theoretical secure against *Adversary 1*.

Security against Adversary 2

Assume that the adversary has information on all inputs except $a_{1,1}$ in addition to information from $k-1$ servers. Furthermore, *Adversary 2* has information from Step 5 of the preprocessing phase, and also learns $b_{g,i}(a_{g,i}+1)$ from the online phase. In addition, *Adversary 2* learns $k-1$ number of $\overline{\left[d\sum_{\iota=1}^{l}(a_{1,\iota}a_{2,\iota}a_{3,\iota})\right]}_{j'}, d_j$ from the reconstruction phase. As a result, *Adversary 2* has the following information (however, g', i' exclude 1,1; $\varepsilon_1^{(i,j)}$ is used to show that it relates to the same value, and $\varepsilon_*^{(h)}$ is used when $i > 1$). However, j' is less than $k-1$. By using this information, *Adversary 2* tries to learn about input $a_{1,1}$ and output $\sum_{i=1}^{l}(a_{1,i}a_{2,i}a_{3,i})$.

$$C = \left\{ a_{g',i'}, b_{g',i'}, b_{1,1}(a_{1,1}+1), \frac{d}{b_{1,1}\varepsilon_1^{(1,2,3,5)}}, \frac{d}{\varepsilon_1^{(4,6,7,8)}}, \varepsilon_*^{(h)}, \overline{\left[d\sum_{\iota=1}^{l}(a_{1,\iota}a_{2,\iota}a_{3,\iota})\right]}_{j'}, d_{j'} \right\}$$

However, since each $\varepsilon_i^{(h)}$ is independant, $b_{1,1}$ and input $a_{1,1}$ will not be leaked. Moreover, *Adversary 2* learns $k-1$ number of $\overline{\left[d\sum_{\iota=1}^{l}(a_{1,\iota}a_{2,\iota}a_{3,\iota})\right]}_{j'}, d_j$. However, reconstruction is impossible with $k-1$ shares. Therefore, *Adversary 2* will not be able to learn the result of the computation.

The same is true even for inputs other than $a_{1,1}$. In addition, even in the case other than $m_i = 3$, values that are not leaked are not included in *C*; therefore, the same can be said for any m_i .Therefore, the following statements are true, and the TUS 4 method is information-theoretical secure against Adversary 2.

$$H(a_{1,1}) = H(a_{1,1}|C)$$

$$H\left(\sum_{i=1}^{l}(a_{1,i}a_{2,i}a_{3,i})\right) = H\left(\sum_{i=1}^{l}(a_{1,i}a_{2,i}a_{3,i})\,|C\right)$$

Proof of Security of the Reconstruction Phase. Even if $\sum_{i=1}^{l}(a_{1,i}a_{2,i}a_{3,i})$ is equal to 0, because nothing is leaked from k or fewer number of shares of $\overline{\left[d\sum_{\iota=1}^{l}(a_{1,\iota}a_{2,\iota}a_{3,\iota})\right]}_{j}, d_j$, *Adversary 2* will not able to learn about the result of the computation.

From the above, because the preprocessing phase only processes information that does not depend on any of the inputs, it can be performed in advance. However, communication is required during the preprocessing phase. Inputs are introduced in the encryption phase, and the result is sent to all the servers. All processes up to this stage are considered the preprocessing of information. The online phase uses all the computed values to perform computations, and no communication is required. Finally, in the reconstruction phase, the player collects $\overline{\left[d\sum_{\iota=1}^{l}(a_{1,\iota}a_{2,\iota}a_{3,\iota})\right]}_{j}, d_j$ from k servers S_j. From this, we learn that communication only occurs in the preprocessing, encryption, and reconstruction phases, and no communication occurs in the online phase.

In addition, when $n > k$, Steps 4 and 5 of the preprocessing phase can be performed by k servers, but Step 7 and onwards must be performed by all n servers. Therefore, the TUS 4 method is also realizable when $n \geq k$. However, when the player who wishes to reconstruct the result wants to collect information from any k servers in the reconstruction phase, each server must distribute the values d_j in the preprocessing phase so that it can be reconstructed from any k servers.

4 DISCUSSION

4.1 Features of the TUS Methods

The TUS methods realize secure computation of secret sharing by using inputs that has been encrypted with random numbers. This is a combination of an encryption with a random number and computation using secret sharing, and the merits of this approach are shown below. Please refer to Section 4 in (Iwamura et al., 2021) for further details on each feature. Here, Features I and II are realized in all the TUS methods, but Feature III is realized in this study. Therefore, we can state that the TUS 4 method realized all the merits below.

Feature I: Input encrypted with random number can be made public. In the TUS methods, the encrypted input can be made public because the input is encrypted with a random number. However, k random numbers that make up the random number need to be concealed.

Feature II: Secure computation with $n < 2k - 1$ is possible. Because the inputs are not distributed, but are encrypted by multiplying with random numbers, direct multiplication of these encrypted inputs will not cause any increase in the polynomial degree. Therefore, we could realize secure computation with minimal server resources (minimum $n = k = 2$).

Feature III: Processes involving random numbers can be computed in advance. Because the processes in the encryption phase can be separated into multiple processes such as multiplication with random numbers, it is possible to realize an additional preprocessing phase, where only processes related to random numbers are performed in advance.

Demerit: The disadvantage of encrypting an input with a random number is that when the input or the result of the computation is equal to 0, information of the input or output will be leaked. Therefore, Condition (1) is required.

4.2 Discussion

Condition (1) is solved using the TUS 4 method, where the reconstruction of the multiplication result is only performed by the player that is allowed to know the result.

Condition (2) is required when $n > k$. If the server or dealer distributes the random number using secret sharing to all n servers, even if $n - k$ servers are broken or lost, a substitute server can reconstruct the random number that was handled by the broken server and continue the computation. Thus, realizing the server loss resistance of secret sharing. However, it is important that the new server must handle the same random number as the server that it is substituting. This can be realized by implementing it in the algorithm (assuming a semi-honest adversary).

Finally, Condition (3) can be solved depending on the application considered. For example, when considering implementation in searchable encryption (Kamal et al., 2017), because the owner of secret information will not be the adversary, Condition (3) can be realized by requesting the owner to generate random numbers that satisfy Condition (3). Moreover, the use of a trusted execution environment such as Intel's Software Guard Extensions (SGX) (Intel Corporation, 2015) can also help with the realization of Condition (3). In future studies, we will consider the most suitable approach for realizing Condition (3).

Therefore, the TUS 4 method only requires Condition (3) to realizes secure computation against a semi-honest adversary.

4.3 Qualitative Comparison

The SPDZ method is limited to the setting $n = k$, and Araki et al.'s method is limited to the setting $n = 3, k = 2$. Only the TUS methods allow for n, k to be set at any value and can realize resistance toward server loss. Araki et al.'s method uses the setting of $n = 3, k = 2$; however, the computation cannot be continued even if one server is lost; therefore, it is not robust against server loss.

However, SPDZ method can accommodate malicious adversaries. Moreover, Araki et al. proposed two protocol versions: a protocol with information-theoretic security and a protocol with computational security.

5 PERFORMANCE EVALUATION

The evaluation is performed by computing $l = 1, 100, 10000$-times of inner-product computation and parameters n, k set at $n = k = 2$ for the TUS 4 method. Inner-product computation is often used in statistical calculations such as distribution and sum of squared deviation, meaning that it can be applied to areas such as searching for gene sequences. In addition, in the computation of the inner product, no communication is required in the online phase of the TUS 4 method. The detailed protocol used is the same as that of the TUS 4 method when $m_i = 2$.

Tables 1 shows the results of the implementation of the online phase using Amazon Web Service with a maximum number of three servers. In particular, we uses the t2.micro instance with 1 vCPU @ 2.5GHz and 1GiB of RAM. Because the preprocessing and encryption phases can be computed in advance, we did not include a comparison of the processing time in our evaluation. From Table 1, the TUS 4 method with no communication in the online phase shows an overwhelming increase in the computation speed.

Table 1: Comparison with conventional methods (for $l = 1, 100, 10000$).

	TUS 4 Method			SPDZ Method			Araki et al.'s Method		
	l=1	l=100	l=10000	l=1	l=100	l=10000	l=1	l=100	l=10000
Computation [ms]	0.03	0.72	88.9	0.04	0.87	86.4	0.03	1.16	105
Connection establishment [ms]	0	0	0	2.06	2.08	2.06	1.99	2.03	2.01
Communication [ms]	0	0	0	1.71	4.54	278	1.36	2.61	73.3
Total time [ms]	0.03	0.72	88.9	3.81	7.49	367	3.38	5.80	181

6 CONCLUSION

In this paper, we provide a method for easing the conditions of the TUS methods that realize information-theoretic security against a semi-honest adversary when $k \leq n < 2k - 1$, and show that it can be realized by using only one condition. In addition, we showed that the acceleration of the computation time is possible by using the property that processes related to random numbers can be separated. In addition, we discussed the properties in detail and showed that our proposed method is also suitable for use in IoT. We also showed that our proposed method is the only method that allows for any n, k to be chosen for $n \geq k$.

In a future study, we will consider the most suitable methods to solve Condition (3) and consider a secure computation against malicious adversaries.

REFERENCES

Araki T., Furukawa J., Lindell Y., Nof A., Ohara K., 2016. High throughput semi-honest secure three-party computation with an honest majority. In CCS 2016, pp. 805-817. ACM, New York, NY, USA.

Beaver D., 1991. Efficient multiparty protocols using circuit randomization. In CRYPTO 1991. LNCS, vol 576, pp. 420-432. Springer, Berlin, Heidelberg.

Ben-Or M., Goldwasser S., Wigderson A., 1988. Completeness theorems for non-cryptographic fault-tolerant distributed computation." In STOC 1988, pp. 1-10. ACM, New York, NY, USA.

Bendlin R., Damgård I., Orlandi C., Zakarias S., 2011. Semi-homomorphic encryption and multiparty computation." In EUROCRYPT 2011. LNCS, vol. 6632, pp. 169-188. Springer, Berlin, Heidelberg.

Brakerski Z., Gentry C., Vaikuntanathan V., 2009. (Leveled) fully homomorphic encryption without bootstrapping. In ITCS 2012, pp. 309-325. ACM, New York, NY, USA.

Cramer R., Damgård I., Maurer U., 2000. General secure multi-party computation from any linear secret-sharing scheme. In EUROCRYPT 2000. LNCS, vol 1807, pp. 316-334. Springer, Berlin, Heidelberg.

Damgård I., Pastro V., Smart N., Zakarias S., 2012. Multiparty computation from somewhat homomorphic encryption. In CRYPTO 2012. LNCS, vol 7417, pp. 643-662. Springer, Berlin, Heidelberg.

Damgård I., Keller M., Larraia E., Pastro V., Scholl P., Smart N.P., 2013. Practical covertly secure MPC for dishonest majority – Or: Breaking the SPDZ Limits. In ESORICS 2013. LNCS, vol. 8134, pp. 1-18. Springer, Berlin, Heidelberg.

Gentry C., 2010. A fully homomorphic encryption scheme." Ph.D Thesis, Stanford University, Stanford, CA, USA.

Intel Corporation. Intel SGX Evaluation SDK, User's Guide for Windows* OS, 2015. https://software.intel.com/sites/products/sgx-sdk-usersguide-windows/Default.htm.

Iwamura K., Kamal A.A.A.M., 2021. Secure computation by secret sharing using input encrypted with random number (full paper). In Cryptology ePrint Archive, Report 2021/548.

Kamal A.A.A.M., Iwamura K., 2017. Conditionally secure multiparty computation using secret sharing scheme for n<2k-1. In PST 2017, pp. 225-230. IEEE, Calgary, AB, Canada.

Kamal A.A.A.M., Iwamura K., Kang H., 2017. Searchable encryption of image based on secret sharing scheme. In APSIPA ASC 2017. IEEE, pp. 1495-1503, Kuala Lumpur, Malaysia.

Kurihara J., Kiyomoto S., Fukushima K., Tanaka T., 2008. A new (k,n)-threshold secret sharing scheme and its extension. In ISC 2008, pp. 455-470, Springer, Berlin, Heidelberg.

Shamir A., 1979. How to share a secret. Communications of the ACM, Vol. 22, Issue 11, pp. 612-613. ACM, New York, NY, USA.

Shingu T., Iwamura K., Kaneda K., 2016. Secrecy computation without changing polynomial degree in Shamir's (k, n) secret sharing scheme. In ICETE 2016, pp.89-94. Lisbon, Portugal.

Smart N.P., Vercauteren F., 2010. Fully homomorphic encryption with relatively small key and ciphertext sizes. In PKC 2010. LNCS, vol. 6056, pp. 420-443. Springer, Berlin, Heidelberg.

Tokita K., Iwamura K., 2018. Fast secure computation based on secret sharing scheme for n<2k-1. In MobiSecServ 2018, pp. 1-5. IEEE, Miami Beach, FL.

van Dijk M., Gentry C., Halevi S., Vaikuntanathan V., 2010. Fully homomorphic encryption over the integers." In EUROCRYPT 2010. LNCS, vol. 6110, pp. 24-43. Springer, Berlin, Heidelberg.