Secure Computation by Secret Sharing Using Input Encrypted with Random Number (Full Paper)

Keiichi Iwamura[1] and Ahmad Akmal Aminuddin Mohd Kamal[1]
[1]*Department of Electrical Engineering, Tokyo University of Science, Tokyo, Japan.*
iwamura@ee.kagu.tus.ac.jp, ahmad@sec.ee.kagu.tus.ac.jp

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 scheme 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 secret sharing scheme 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; Chaum et al., 1988; Cramer et al., 2000; Gennaro et al., 1998; Ikarashi et al., 2010; Kamal and Iwamura, 2017; Shingu et al., 2016; Tokita et al., 2018)
- Homomorphic encryption (Bendlin et al., 2011; Brakerski et al., 2012; Brakerski et al., 2011; 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.

The secret sharing scheme is a method in the field of cryptography for data encryption, in which a single secret/input is divided into multiple shares, which are then distributed to multiple users. A known example of a secret sharing scheme is the (k, n) threshold secret sharing scheme. In this scheme, a secret s is divided into n number of shares. The original secret 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 secret. Therefore, when $n > k$, a (k, n) threshold secret sharing scheme can realize resistance toward loss of at most $n - k$ servers.

However, secure computation using a secret sharing can perform secrecy addition and subtraction easily, but this is not so in the case of secrecy 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 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 scheme, 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; Damgård et al., 2013) 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 3 (**T**okyo **U**niversity of **S**cience) method (Tokita et al., 2018) that realizes secure computation under the following three conditions.

- Condition 1: The computation result does not include 0.
- Condition 2: Random numbers reconstructed by each server are fixed.
- Condition 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.

In the TUS 3 method, a product-sum operation was proposed, and it was also proved that the combination of this product-sum operation is also secure. Thus, any computation is possible, even when $k \leq n < 2k-1$. In addition, the characteristics of all TUS methods, including the TUS 3 method, are that the input of secure computation is first encrypted with a random number before being used for secure computation, and the aforementioned computation when $k \leq n < 2k-1$ makes use of this particular property. However, TUS methods incur high computational costs and cannot realize fast computation.

Therefore, in this study, by using the property that secret information is encrypted with a random number, we propose a secure computation 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, realizing a faster online phase (known as the TUS 4 method).

Next, we reduce the conditions in the TUS methods. Depending on the application, there are cases where the three aforementioned conditions can be realized easily; however, there are also cases where it cannot be handled easily. Therefore, we reduce the condition required and show that by leaving Condition (3), the TUS methods can be realized securely. In addition, we also discuss the merits and demerits of the properties of the TUS methods.

System Model

In this study, 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. This model is widely used currently and is the business model used in Sharemind.

The remainder of this paper is organized as follows: in Chapter 2, we present related works; in Chapter 3, we explain the TUS 4 method, in Chapter 4 we the discussion on the merits and demerits of encrypting secret information with random numbers and discuss each condition of the TUS methods. Finally, in Chapter 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 under the setting $n = k$. In SPDZ, the owner of the secret is one of the n players involved in multiparty computation. Moreover, in SPDZ, 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 consists of ***preprocessing*** and an ***online phase***. This ensures the confidentiality of the inputted secrets by using an additive secret-sharing scheme. Through the SPDZ method, secrecy addition can be easily achieved. Secrecy multiplication in SPDZ is based on Beaver's circuit randomization (Beaver, 1991). To perform secrecy multiplication, 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.

In SPDZ, for example, the secret information of x is reconstructed from its shares $\langle x\rangle$, denoted as $x = open(\langle x\rangle)$. The protocol for multiplication of $x \cdot y$ proposed by SPDZ is shown below. 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.

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

Typically, in a secure secrecy 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 secrecy computation under the parameters $n = 3, k = 2$, which requires only one communication per multiplication (Araki et al., 2016). The detailed protocol is described below. Note that it is usually not considered a problem, even if communication is required in the *preprocessing phase*. In addition, secrecy computation of addition is performed locally, where the shares are added together.

Preprocessing Phase:

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 secret v_1 to player 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 player P_i. Note that $b_i = y_{i-1} - v_2, y_1 + y_2 + y_3 = 0$.

Multiplication

1. Player P_i computes $r_i = (a_i b_i - x_i y_i + \beta_i)/3$ and sends to player P_{i+1}.
2. Player P_i computes $z_i = r_{i-1} - r_i$, $c_i = -2r_{i-1} - r_i$, and holds (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 multiplying $v_1 v_2$ can be computed using the equation shown below. Note that $c_i = -2r_{i-1} - r_i = z_{i-1} - v_1 v_2$.

$$z_j - c_i = v_1 v_2$$

2.3 (k, n) threshold secret sharing

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 (Shamir, 1979) (Shamir's method) and the XOR-based method for sharing and reconstruction of secret information proposed by Kurihara et al. (Kurihara et al., 2008) (XOR method). In our protocol, unless stated otherwise, Shamir's method was used, and all computations were performed in modulus p. In addition, the shares of the secret information s, are represented by $\overline{[s]}_i$.

2.4 The TUS Methods

First, Shingu et al. proposed a 2-inputs-1-output computation called the TUS 1 method (Shingu et al., 2016), 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* × *polynomial* approach to prevent an increase in the polynomial degree. However, the TUS1 method introduces 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, the 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 TUS 1 method can realize a very effective specific computation, such as computation of Rivest–Shamir–Adleman (RSA) encryption. However, it is not capable of coping with computations that require a combination of addition/subtraction and multiplication/division.

Next, 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 and Iwamura, 2017). This method was proven to be secure under the three aforementioned conditions. However, the first condition in this method is extended to the following: the value of the inputs and output of the computation does not include 0. In addition, it was shown that this method is secure against computation that involves a combination of product-sum operations. Therefore, this method can realize any arithmetic computation under the setting $k \leq n < 2k - 1$. However, the TUS 2 method incurs significantly more computational cost compared with the conventional method in $n \geq 2k - 1$; therefore, it is not the most efficient method.

Therefore, Tokita et al. proposed an improved version of the TUS 2 method, known as the TUS 3 method, where XOR method (Kurihara et al., 2008) is introduced and realizes a more efficient method for secrecy computation (Tokita et al., 2018). Out of the three aforementioned conditions, the TUS 3 method proposed a way to ease one of the conditions (known as the TUS3' method), wherein there are no longer limitation for the inputs of computation; however, the three conditions still remain.

Note that all TUS methods share a common point wherein the secret information is first encrypted with a random number and is then used in the secrecy computation using secret sharing. Moreover, Condition (3) where each server S_j holds random numbers unknown to the adversary, and the shares of random numbers that make up the random numbers unknown to the adversary is defined as follow.

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

Here, $\varepsilon = \prod_{j=0}^{k-1} \varepsilon_j$ is defined as a random number unknown to the adversary.

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* in which 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 of communication in the *online phase* and speed up the entire process.

Below, we explain the protocol for the TUS 4 method.

3.1 The Protocol

Here, instead of the simple product-sum operation of $ab + c$, we present a solution for computing the extended product-sum operation of $\sum_{i=1}^{l}\left(a_{1,i}a_{2,i} \dots a_{m_i,i}\right)$. 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.

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

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 $m = 3,4$.

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

$$
\begin{aligned}
a_1a_2a_3a_4 = (a_1+1)(a_2+1)(a_3+1)(a_4+1)\\
-\{(a_1+1)(a_2+1)(a_4+1)\\
+(a_1+1)(a_3+1)(a_4+1)\\
+(a_2+1)(a_3+1)(a_4+1)\\
+(a_1+1)(a_2+1)(a_3+1)\}\\
+\{(a_1+1)(a_2+1)\\
+(a_1+1)(a_3+1)\\
+(a_1+1)(a_4+1)\\
+(a_2+1)(a_3+1)\\
+(a_2+1)(a_4+1)\\
+(a_3+1)(a_4+1)\}-\{(a_1+1)\\
+(a_2+1)+(a_3+1)+(a_4+1)\}\\
+1
\end{aligned}
$$

Therefore, when $l=2, m_1=3, m_2=4$, $a_1a_2a_3$ will be $a_{1,1}a_{2,1}a_{3,1}$, and $a_1a_2a_3a_4$ 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 protocol of the TUS 4 method, inputs $a_{1,i}, a_{2,i}, \ldots . a_{m_i,i}$ 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. In addition, 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 $\varepsilon_i^{(h)}(h=1,\ldots 8)$, 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 number of servers in Steps 4 and 5 shown in the *preprocessing phase* are chosen in advance from n number of servers. Below, for ease of understanding, we show our protocol for $m_i=3$; however, it is clear that it can also be extended to any m_i.

Preprocessing Phase

1. Dealer D generates k random numbers $b_{(1,i),0}, b_{(1,i),1}, \ldots, b_{(1,i),k-1}$ with respect to secrets $a_{1,i}(i=1,\ldots,l)$, 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 $(j=0,\ldots,k-1)$ collects each $\overline{[\varepsilon_{\iota,j}^{(h)}]}_u (u=0,\ldots,k-1, h=1,\ldots,8)$ 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. $(i=1,\ldots,l)$.

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

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

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

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

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

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

$$\frac{d}{b_{3,i}\varepsilon_i^{(7)}}=\prod_{j=0}^{k-1}\frac{d_j}{b_{(3,i),j}\varepsilon_{i,j}^{(7)}}, \quad \frac{d}{\varepsilon_i^{(8)}}=\prod_{j=0}^{k-1}\frac{d_j}{\varepsilon_{i,j}^{(8)}}$$

7. All servers S_j compute and hold the following.

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

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

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

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

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

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

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

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

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.

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

Reconstruction Phase

1. The player who wishes to reconstruct the result collects $\overline{\left[d\sum_{\iota=1}^{l}(a_{1,\iota}a_{2,\iota}a_{3,\iota})\right]_j}$ and d_j from k number of servers S_j, reconstructs $d\sum_{i=1}^{l}(a_{1,i}a_{2,i}a_{3,i})$, d, and computes the result of $\sum_{i=1}^{l}(a_{1,i}a_{2,i}a_{3,i})$ as follows:

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

3.2 Security Analysis of the TUS 4 method

In the proposed method of extended product-sum operation with $t(=\sum_{i=1}^{l} m_i)$ inputs and one output, regardless of the security level of the method used, 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: We can state that our proposed TUS 4 method is secure if it is secure against Adversaries 1 and 2 defined below.

Adversary 1 :

The adversary has information on $t-2$ inputs and one output of the extended product-sum operation. *Adversary 1* has information on the entered 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 $k-1$ servers. According to this information, the adversary attempts to learn the remaining two inputs.

Adversary 2:

The adversary has information on the $t-1$ inputs of the extended product-sum operation. *Adversary 2* has information on the input 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 output of the computation.

The security proof for the TUS 4 method is shown below.

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 random numbers $\varepsilon_{i,j}^{(h)}$ (which is used to construct random numbers $\varepsilon_i^{(h)}$) in Step 4; however, random numbers $\varepsilon_i^{(h)}$ will not leak from $k-1$ servers. In addition, 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 addition, 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. $H(x)$ 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

Because the secret information is smaller than $p-2$, even if one 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 remainder of the secret information.

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

Proof of security of the Online Phase

<u>*Security against Adversary 1*</u>

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, in addition to 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*. By using this information, *Adversary 1* tries to learn about secret inputs $a_{1,1}, a_{2,1}$. However, if fewer than k number of shares and random numbers are collected, information regarding the random numbers and secret information will not be leaked.

To simplify the problem, we redefined the parameters above to avoid any parameter duplication. 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 $i>1$).

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

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:

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

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, we can state that the TUS 4 method is information-theoretical secure against *Adversary 1*.

<u>*Security against Adversary 2*</u>

Assume that the adversary has information on all inputs except $a_{1,1}$ and 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*. By using this information, *Adversary 2* tries to learn about secret input $a_{1,1}$ and output $\sum_{i=1}^{l}(a_{1,i}a_{2,i}a_{3,i})$. However, if fewer than k number of shares and random numbers are collected, information regarding the random numbers and secret information will not be leaked.

To simplify the problem, we redefined the parameters above to avoid any duplication of a parameter. 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 $k-1$.

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

In addition, *Adversary 2* learns $k-1$ number of $\overline{\left[d\sum_{\iota=1}^{l}(a_{1,\iota}a_{2,\iota}a_{3,\iota})\right]}_j, d_j$; however, because reconstruction is impossible with $k-1$ shares, *Adversary 2* will not be able to learn the result of the computation.

The same is true even for inputs other than $b_{1,1}$. 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-theoretically 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 on $\overline{\left[d\sum_{\iota=1}^{l}(a_{1,\iota}a_{2,\iota}a_{3,\iota})\right]}_j, d_j$, we can say that *Adversary 2* is 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 secret information, it can be performed in advance before inputting the secret information. However, communication is required during the *preprocessing phase*. Secret information is 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 between servers is required. Finally, in the *reconstruction phase*, the player who wants to receive the final computation result 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 and obtains the computation result. 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 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 the player can reconstruct them from any k servers.

4. DISCUSSION AND CONSIDERATION

4.1 Features of Our Proposed Method

The TUS methods realize secure computation of secret sharing by using secret information 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 discussed below. However, Features I and II are realized in all the TUS methods, but Feature III is the feature realized in this study. Therefore, we can state that the TUS 4 method realized all the merits discussed below.

Feature I : Secret information encrypted with random number can be made public

In the TUS methods, the encrypted secret can be made public because the secret information is encrypted with a random number. However, k random numbers that make up the random number need to be concealed. For example, by making the IoT device to hold $b_{g,i}$ of the TUS 4 method and compute $b_{g,i}(a_{g,i}+1)$ on the collected information $a_{g,i}$, if n servers that received it know each $b_{(1,i),j}$ (and shares that satisfy Condition (3)), secure computation can be performed by using the output of the IoT device as it is. In contrast, in conventional methods where secret information is distributed directly, the IoT device will require a separate encryption method to conceal the secret information, and servers that receive it will have to momentarily reconstruct the encrypted information and distribute it using a secret sharing scheme. In this case, there is a risk of information leakage, and an additional process of reconstruction will be required.

Feature II : Secure computation with $n < 2k-1$ is possible

Because the secret information is not distributed, but is encrypted by multiplying with random numbers, direct multiplication of these encrypted secrets will not cause any increase in the polynomial degree. However, if it is left as it is, it will cause a difference in random number later, making addition impossible. Therefore, when adding this secret information, a process to unify the random number is needed, and by integrating this process, a combination of addition and multiplication of the encrypted input will be possible. Using this approach, we could realize secure computation of secret sharing with minimal server resources (minimum $n = k = 2$).

Feature III: Processes involving random numbers can be computed in advance.

Secret information is encrypted with a random number; however, 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. However, we need to predefine in advance the computation that needs to be performed in the preprocessing phase. Typically, it is difficult to consider a case where a predefined type of computation is not set in advance. Usually, the type of computation required is set in advance, and then the user that is needed for that computation is assembled (or gathered). At that time, each user can perform the required preprocessing process that involves a random number when confirming the type and flow of the process or computation to be performed. Thus, a faster secure computation using the user's input can be realized.

Demerit:

The disadvantage of encrypting secret information with a random number is that when the secret information or the result of the computation is equal to 0, information of the secret or output will be leaked. To prevent this, Condition (1) is required. In contrast, when secret information is distributed directly, even if the value of the secret information is 0, it can be concealed as is.

4.2 Discussion About the Preconditions

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

Condition (2) is typically required in the following situation. In the preprocessing phase of the TUS 4 method, k servers each hold random numbers that had been specified beforehand. However, if one of the servers is broken and can no longer be used, the random number that had been specified to that server will be lost; therefore, secure computation will no longer be possible. Therefore, when $n > k$, if the server or the 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 is handled by the broken server and continue the computation. Thus, when $n > k$, the server loss resistance of the secret sharing scheme can be maintained. However, even if the honest server is replaced with a dishonest server, because it is assumed that only $k - 1$ out of n servers are dishonest, it will not pose any problem. Therefore, when substituting the broken server with a new server, it is important to handle the same random number as the server that it is substituting. However, when considering a semi-honest adversary, this condition can be realized by implementing it in the algorithm. However, extra precautions are required when a malicious adversary is assumed.

Finally, Condition (3) can be solved depending on the application considered. For example, when considering implementation in searchable encryption (Kamal et al., 2017; Kamal et al., 2019), 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). In addition, when considering implementation into outsourcing computation (Iwamura et al., 2020), because the client who requested the service will not be the adversary, the client can generate random numbers that satisfy Condition (3). However, the realization of Condition (3) can be difficult for applications other than the one mentioned above. Typically, in this case, we assume the use of a trusted third party (TTP), and the TTP will generate random numbers to satisfy Condition (3). However, the installation of a TTP is a significant problem. However, the use of a trusted execution environment (TEE), which has attracted significant attention recently, can also help with the realization of Condition (3). TEE is a technology that increases the security of the execution environment by creating an isolated execution environment in the processor. A representative method was proposed by Intel, known as the Software Guard Extensions (SGX) (Intel Corporation, 2015). Intel SGX is available in almost all CPUs after Intel Core i7 and can be used easily. However, Intel SGX is not the only option for realizing Condition (3). Therefore, in future studies, we will consider the most suitable method for this.

From the above, in the TUS 4 method of secure computation against a semi-honest adversary, only Condition (3) is remaining.

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 parameters n, k to be set at any value and are able to realize resistance toward server-loss. Araki et al.'s method uses the setting of $n = 3, k = 2$; however, the computation cannot be performed 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. We present a qualitative comparison between our proposed methods and conventional methods in Table 1.

5. PERFORMANCE EVALUATION AND EXPERIMENTAL RESULTS

The evaluation is performed by computing the l-times of inner-product computation with $l = 1, 100, 10000$, and parameters n, k set at $n = k = 2$ for the TUS 4 method. The results are then compared with the implementation results of the SPDZ and Araki et al. methods. Currently, the method by Araki et al. shows the fastest computation time.

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 algorithm used was the same as that of the TUS 4 method when $m_i = 2$.

Tables 2–4 show the results of the implementation of the *online phase* using Amazon Web Service with a maximum number of three servers. Because the *preprocessing* and *encryption phases* can be computed in advance, we did not include a comparison of the processing time in our evaluation. Because the SPDZ method requires two, whereas the method proposed in Araki et al. requires one data point to be sent for each multiplication, a total of $2l$ and l data are required to be sent in the SPDZ method and Araki et al.'s method, respectively. However, because a significant amount of time is required to establish a connection, all the data are sent at once after the connection has been established. In addition, the size of one dataset is 127 bits.

From Tables 2–4, the TUS 4 method with no communication in the *online phase* shows an overwhelming increase in the computation speed. In addition, even if all the required data are sent at once after the connection has been established, the communication time increases when $l = 10000$ for both the SPDZ and Araki et al.'s methods.

Table 1. Qualitative comparison of our proposed method with conventional methods (I.T.: Information-theoretic Security; Comp.: Computational Security)

	n and k	Adversary	Security	Server-loss Resistance
TUS Methods	$n \geq k$	Semi-honest	I.T.	Yes
SPDZ Method	$n = k$	Malicious	Comp.	No
Araki et al's Method	$n = 3, k = 2$	Semi-honest	I.T. or Comp.	No

Table 2. Comparison with conventional methods (for $l = 1$)

	TUS 4 Method	SPDZ Method	Araki et al.'s Method
Computation time [s]	3.80×10^{-5}	3.41×10^{-5}	1.92×10^{-5}
Communication establishment [s]	0	0.100482542	0.099988509
Communication time [s]	0	0.100984535	3.80×10^{-5}
Total time [s]	3.80×10^{-5}	0.201693428	0.100248701

Table 3. Comparison with conventional methods (for $l = 100$)

	TUS 4 Method	SPDZ Method	Araki et al.'s Method
Computation time [s]	2.41×10^{-3}	3.36×10^{-4}	2.40×10^{-4}
Communication establishment	0	0.10020276	0.100412364
Communication time [s]	0	0.10052742	1.70×10^{-4}
Total time [s]	2.14×10^{-3}	0.201361759	0.101024941

Table 4. Comparison with conventional methods (for $l = 10000$)

	TUS 4 Method	SPDZ 2 Method	Araki et al.'s Method
Computation time [s]	2.43×10^{-1}	2.91×10^{-2}	2.11×10^{-2}
Communication Establishment [s]	0	0.102026318	0.100967475
Communication time [s]	0	1.018629671	8.34×10^{-1}
Total time [s]	2.43×10^{-1}	1.158342564	0.964274707

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 computational acceleration is possible using the property that processes related to random numbers can be separated. In addition, we discuss the properties in detail and show 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 with security 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.

Brakerski Z., Vaikuntanathan V., 2011. Fully homomorphic encryption from ring-LWE and security for key dependent messages. In CRYPTO 2011. LNCS, vol 6841, pp. 505-524. Springer, Berlin, Heidelberg.

Chaum D., Crépeau C., Damgård I., 1988. Multiparty unconditionally secure protocols." In STOC 1988, pp. 11-19. 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.

Gennaro R., Rabin M. O., Rabin T., 1998. Simplified VSS and fast-track multiparty computations with applications to threshold cryptography." In PODC 1998, pp. 101-111. ACM, New York, NY, USA.

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

Ikarashi D., Chida K., Takahashi K., 2010. Information-theoretic Security Analysis of the Efficient 3-Party Secure Function Evaluation. In SIG Technical Reports, vol. 2010-CSEC-50, no. 46, pp. 1-8. (*In Japanese*)

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, Yamane M., 2020. Secure Outsourcing Computation for Matrix Multiplication based on Secret Sharing Scheme using Only One Server. In Journal of Information Processing, vol. 61, no. 5, pp. 1073-1079. (*In Japanese*)

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.

Mohd Kamal A.A.A, 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.

Mohd Kamal A.A.A, 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.

Mohd Kamal A.A.A., Iwamura K., 2019. Searchable Encryption Using Secret-Sharing Scheme for Multiple Keyword Search Using Conjunctive and Disjunctive Searching, In CyberSciTech 2019, pp. 149-156. Fukuoka, Japan.

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., 2015. Secure Multiplication Using Ramp Secret Sharing Scheme. In CSS 2015, vol. 2015, no. 3, pp.987-994. (*In Japanese*)

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.

Sharemind, Cybernetica. https://sharemind.cyber.ee.

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.

Watanabe T., Iwamura K., Kaneda K., 2015. Secrecy multiplication based on a (k, n) -threshold secret-sharing scheme using only k servers." In Computer Science and Its Applications. LNEE, vol. 330, pp. 107-112. Springer, Berlin, Heidelberg.