IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 7 October 2022, accepted 6 November 2022, date of publication 14 November 2022, date of current version 21 November 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3222312

RESEARCH ARTICLE

TTP-Aided Secure Computation Using (k, n) Threshold Secret Sharing With a Single Computing Server

KEIICHI IWAMURA[1], (Member, IEEE), AND AHMAD AKMAL AMINUDDIN MOHD KAMAL[2], (Member, IEEE)

[1]Department of Electrical Engineering, Tokyo University of Science, Tokyo 125-8585, Japan
[2]Department of Information and Computer Technology, Tokyo University of Science, Tokyo 125-8585, Japan

Corresponding author: Ahmad Akmal Aminuddin Mohd Kamal (ahmad.amin@rs.tus.ac.jp)

ABSTRACT Secure computation can be divided into those using homomorphic encryption and secret sharing. The advantage of the former method is that the computation can be performed on a single computing server and the computation process can be made public if the encryption key used is securely managed. However, the latter is computationally light and capable of high-speed processing, but it requires multiple independently managed computing servers, and processes of k or more servers cannot be disclosed. When considering a business model for implementing secure computation that uses secret sharing, private information can be leaked if computing servers are managed by the same organization. Therefore, we need a complex business model in which multiple companies without conflicts of interest manage each computing server independently and make a profit. However, this approach is difficult to implement in practice. In this study, we proposed a secure computation using secret sharing. Moreover, by effectively using TTP, we demonstrated that it is possible to realize secure computation with a single computing server. In addition, we demonstrated that, if the security keys used are securely managed, the entire computational process can be made public. In other words, we realize a method that addresses all the drawbacks of the aforementioned methods, and it is possible to realize faster and more secure computations than conventional methods using secret sharing.

INDEX TERMS Secure computation, secret sharing, multiparty computation, trusted third party, fast processing.

I. INTRODUCTION

Society 5.0 achieves a high degree of convergence between cyberspace (virtual space) and physical space (real space). In Society 5.0, a large amount of data collected by countless Internet of things (IoT) devices in physical space is automatically sent to the clouds in cyberspace to be analyzed and fed back into the physical space as new quality data to solve various problems. However, because individuals' personal and confidential information is often included in the collected data, there are several issues in realizing both privacy protection and data utilization.

The associate editor coordinating the review of this manuscript and approving it for publication was Amjad Mehmood.

Therefore, several studies have been conducted on the concept of secure computation, which can perform various computations while keeping the inputs private. Secure computation methods can be divided into those using homomorphic encryption [1], [2], [3], [4], [5], [6], [7], [8] and secret sharing [9], [10], [11], [12], [13], [14], [15].

Secure computation using homomorphic encryption can be performed on a single computing server. If the encryption keys used are securely managed, the entire computation process can be made public, making the practical implementation of this method extremely easy and simple. However, this method incurs extremely high computational costs.

However, secure computation using secret sharing is computationally light and can be processed at high speeds.

However, it requires multiple computing servers (generally, the number of computing servers n required must be at least three). Moreover, the computational processes for k or more servers cannot be disclosed.

When considering a business that offers secure computation using secret sharing as a service, confidential information distributed by secret sharing is leaked if the computing servers are managed by the same organization. Therefore, a business model is required in which multiple companies without conflicts of interest manage each computing server independently and still profit from it, which is difficult to implement practically.

In addition, there are no known secure computation methods using secret sharing that achieve information-theoretic security for $n < 2k - 1$ when performing multiplication, and communication between computing servers is required for each multiplication performed. Because communication often requires more time than the actual secure computation itself, a high-speed communication network between all computing servers is necessary to perform high-speed processing (including multiplication) in secure computation.

Damgård et al. proposed an SPDZ method that combines somewhat homomorphic encryption (SHE) with secure computation based on secret sharing to realize computational security against a dishonest majority when $k \leq n < 2k - 1$ [6], [7], [8]. The SPDZ method realizes secure computations with $n = k$ servers. However, the minimum number of servers required is $n = k = 2$, which involves communication between two independent servers for each multiplication performed. Moreover, this method performs multiplication using a multiplication triple constructed using the SHE, which significantly increases the overall process cost and time [16].

Meanwhile, our research group at Tokyo University of Science (TUS) proposed multiple variations of the secure computation method (called *TUS methods*) that is information-theoretic secure against semi-honest adversaries when $n \geq k$ instead of $n \geq 2k - 1$, by assuming a trusted third party (TTP) [17], [18], [19], [20], [21]. The characteristic of TUS methods is that the encryption of the secret input is realized by multiplying the secret input with a random number, and consecutive secure computations are performed using secret sharing. Therefore, the overall computational cost is significantly lower than that of the SPDZ method that implements the SHE. However, the greatest disadvantage of TUS methods is that they require three conditions to realize secure computations (described later in Section II).

However, other research on secure computation using secret sharing used TTP or reliable servers [22], [23], [24]. However, the TTP implemented in these methods is used as a supporting entity to reduce the amount of computation and to reinforce security and is not used to eliminate communication from the secure computation process or reduce the number of computing servers required.

Our Contributions:

This paper leverages the TTP and proposes a secure computation method using (k, n) threshold secret sharing. The preliminary version of this appears in the poster presentation in [25]. (k, n) threshold secret sharing is a method where a secret input s is converted into n different values (known as *shares*) and sent among n computing servers [26]. However, the proposed method differentiates the parameters used for shares and computing servers and proposes a secure computation that can be executed even when $N < k$, where n is the number of shares and N is the number of computing servers.

The contributions of this study are as follows:

1) We demonstrate that (k, n) threshold secret sharing and secure computation based on it can be realized with a minimum of only one computing server. Therefore, our proposed method can be hosted by a single organization instead of multiple (generally, three or more) independent organizations, as in most conventional methods.
2) We show that, if the security keys (random numbers) used are managed securely, the entire computational process can be made public, eliminating the need for independent server management. This is typically possible in secure computations using homomorphic encryption. However, in our proposed method, we also realize the advantage of homomorphic encryption without the significant computational cost incurred by homomorphic encryption.
3) We demonstrate that secure computation using (k, n) threshold secret sharing is possible without communication. When secure computations are performed within a single computing server, no communication is required between multiple servers. However, communication is still required between the TTP and single computing servers. Therefore, we also discuss replacing the role of TTP with a trusted execution environment (TEE) commonly found in the recent generation of computer processing units (CPU). Replacing the TTP with a TEE in the CPU removes the required communication and enables high-speed computation.
4) Finally, we show that it is possible to build an effective business model for secure computation using (k, n) threshold secret sharing.

II. RELATED WORKS

This section explains the basic methodology required to realize secure computation using (k, n) threshold secret sharing and describes conventional methods of secure computation based on secret sharing.

A. (k, n) THRESHOLD SECRET SHARING

Secret sharing is a method to convert a secret input s into n different values (known as shares) and distribute them to n different computing servers. A secret sharing that satisfies both conditions stated below is known as (k, n) threshold secret sharing. However, $n \geq k > 1$.

1) Any $k-1$, or fewer shares will reveal nothing about the secret input s.
2) Any k or more shares will allow for the reconstruction of the secret input s.

Therefore, if the computing servers that store the shares are managed by the same organization, k (or more) shares are collected, and the secret input is leaked.

Examples of (k, n) threshold secret sharing include methods that use polynomials for the distribution of secret input proposed by Shamir [26] (hereinafter called Shamir's (k, n) method), a method that involves the XOR operation proposed by Kurihara et al. [27] (hereinafter called the XOR method), and additive secret sharing. Unless otherwise stated, Shamir's (k, n) method is used, and the prime number used is p. In addition, the share of the secret input s held by each server S_i $(i = 0, \ldots, k-1)$ is denoted by $[s]_i$.

B. SPDZ METHOD BY DAMGÅRD ET AL

Damgård et al. proposed a secure computation method known as SPDZ (pronounced *"speedz"*) method that enables secure multiparty computation even against a dishonest majority in the setting of $n = k$ [6], [7], [8]. In the SPDZ method, the secret input belongs to one of the n players, and even if all players $(n-1)$ other than the owner collude, provided that the owner keeps their share of the secret secure, the secret input of the owner will not be leaked.

The SPDZ method consists of pre-processing and an online phase. It ensures the confidentiality of secret inputs using additive secret sharing. Addition can easily be achieved using the SPDZ method. However, multiplication in the SPDZ method is based on Beaver circuit randomization [28] (or Beaver multiplication). To perform multiplication, shares of random numbers $[a]$, $[b]$, $[c]$, called a *multiplicative triple* that satisfies $ab = c$ are used.

In the SPDZ method, for example, the process in which the secret input x is reconstructed from its shares $[x]$, is denoted as $x = open([x])$; however, because this process is performed twice for each multiplication, two communications will be required for each multiplication.

The protocol for multiplying xy proposed in the SPDZ method is described below. However, constructing a multiplication triple requires SHE, which entails a prohibitive computation cost and thus significantly increases the overall process time. Moreover, the overall security achieved using the SPDZ method is related to computational security.

Pre-processing

1) Generates multiplication triple $[a]$, $[b]$, $[c]$.

Distribution

1) Compute shares $[x]$ and $[y]$ for secret inputs x and y using additive secret sharing.

Secure computation

1) Each server reconstructs $d = open([x] - [a])$ and $e = open([y] - [b])$, and computes the following:

$$[xy] = de + e[a] + d[b] + [c]$$

C. TTP-AIDED SECURE COMPUTATION

Although there are few studies on secure computation using secret sharing that assumes a TTP, there are a few examples [22], [23], [24] as described below.

When analyzing images acquired by mobile sensors with a convolutional neural network (CNN), to reduce the communication delay from the mobile sensor to the CNN, two edge servers are placed on the mobile sensor side, and CNN feature extraction is jointly performed using secret sharing. This study used a TTP to compute the multiplication triple required in the SPDZ method and sent it to the two edge servers [23]. Therefore, this method reduces the computations by computing the multiplication triple of the SPDZ method with TTP instead of SHE. However, similar to the SPDZ method, this method still requires communication between edge servers for each multiplication and independent management of the two edge servers.

There are also methods that assume two independent TTPs and maintain security even if the information in one TTP is leaked. However, because the two TTPs are independent, the amount of communication between the servers and TTPs is excessive; therefore, we must improve communication efficiency.

There are also examples where a server originally in the system momentarily takes on a trusted server role (known as partial TTP). For example, partial TTP is a server that assumes no collusion with input users who participate in secure computation and improves the security of secure computation [22].

In contrast, the following TUS methods and our proposed method include the collusion of computing servers and players as one of the possible attacks to be considered during the security evaluation. In addition, we do not require any assumptions, such as the role of partial TTP.

D. TUS METHODS

TUS methods were used to represent the secure computation methods proposed by the research group at the Tokyo University of Science (TUS).

The first variation of the TUS method was proposed by Shingu et al. [17] to perform a two-inputs-one-output secure computation (called the TUS 1 method), in which the secret input of a client is first encrypted with a random number. When performing secure multiplication, the encrypted secret is momentarily restored as a scalar value, and multiplication is performed using the scalar value $\times$ polynomial approach to prevent an increase in the polynomial degree.

However, the TUS 1 method introduces another problem: when computation involving a combination of operations, such as $ab + c$, is performed, if the adversary has information about one of the inputs and the output, they can specify the value of the remaining two inputs. Therefore, a condition in which computation involving a combination of addition and multiplication is not performed is needed, in addition to the existing condition, in which the input of the secure

computation does not include the value 0. Therefore, the TUS 1 method can enable highly effective specific computation, such as the computation of Rivest–Shamir–Adleman (RSA) encryption. However, they cannot handle computations that require a combination of addition and multiplication.

To solve this problem, Kamal et al. [18] introduced an improved TUS 2 method, in which the computation involving a combination of addition and multiplication can also be performed securely. This method has been proven secure under the following three conditions:

(1) The values of the inputs and outputs of the computation do not include 0.
(2) Random numbers restored by each server are fixed.
(3) Each server holds a share $[\varepsilon]$ of a random number ε and shares $[\varepsilon_0], \ldots, [\varepsilon_{k-1}]$ of k random numbers $\varepsilon_0, \ldots, \varepsilon_{k-1}$ unknown to the adversary. Here, $\varepsilon = \prod_{i=0}^{k-1} \varepsilon_i$.

In addition, this method is secure against computations that involve 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 a significantly higher computational cost than the conventional method for $n \geq 2k-1$; therefore, it is not the most efficient method.

Therefore, Tokita et al. [19] proposed an improved version known as the TUS 3 method, which incorporates the XOR method to achieve more efficient secure computation and eases one of the conditions, wherein the limit on the inputs of secure computation is removed; however, the three conditions remain. Moreover, because the TUS 3 method included communication in secure computation, the improvement in processing speed was limited to a certain extent.

Subsequently, Iwamura et al. [20] proposed the TUS 4 method, which divides the computation into pre-processing and secure computation and concentrates the processes that require communication in the pre-processing phase. Moreover, the communication required in the TUS 4 method is independent of the number of multiplications performed. The TUS 4 method performs the following extended product-sum operation: In addition, the TUS 4 method solves conditions (1) and (2), leaving only condition (3).

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

In addition, Ochiai et al. [21] proposed the TUS 5 method, which is secure against malicious adversaries.

III. PROPOSED METHOD

This section describes our proposed method that can realize secure computation with a single computing server using (k, n) threshold secret sharing and is secure against a semi-honest adversary. Contribution (1) can be realized using the following algorithm:

A. PROPOSED PROTOCOLS

As with the TUS 4 method [20], our proposed method also supports the following extended product-sum operation:

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

However, for simplicity, the protocol described below assumes $m_i = l = 2$ to compute the following operation. However, for ease of understanding, let $a_{1,1} = a_1$, $a_{2,1} = a_2$, $a_{1,2} = a_3$, $a_{2,2} = a_4$.

$$a_{1,1}a_{2,1} + a_{1,2}a_{2,2} = a_1a_2 + a_3a_4$$

In the following algorithm, inputs a_1, a_2, a_3, a_4 and the output (computation result a) are chosen from modulo p and are numbers less than or equal to $p-2$ in modulo p. In addition, random numbers throughout the proposed method were selected from uniformly distributed random numbers, and zero was not used. All values used in the protocol are in $GF(p)$, and all operations, including secret sharing, are performed within modulo p. We assume that p to be a sufficiently large prime number to improve security (described later in Section III.B).

We also assume that the communication between the players, computing server, and TTP is secure. For simplicity, in the following, the number of computing servers is assumed to be $N = 1$ (referred to as server S), and shares n and threshold k are assumed to be $n = k > 2$. However, to consider when $n > k$, the parameters n and k are used separately below. Additionally, when $n = k$, additive secret sharing can be used.

Therefore, in the protocols described below, we consider a system model with one TTP, one computing server S, and g players U_i $(i = 1, \ldots, g)$ (hereinafter $g = 4$) that provides secret inputs a_1 (Player U_1), a_2 (Player U_2), a_3 (Player U_3), a_4 (Player U_4), and one player who restores the computation result $a = a_1a_2 + a_3a_4$.

Protocol 1.1: Processes by TTP

1) TTP generates $n-1$ random numbers τ_j $(j = 1, \ldots, n-1)$. Here, $\tau_0 = 1$.
2) TTP generates random numbers d, b_i $(i = 1, \ldots, g)$ and computes the following auxiliary random numbers:

$$\frac{d}{b_1b_2}, \frac{d}{b_1}, \frac{d}{b_2}, \frac{d}{b_3b_4}, \frac{d}{b_3}, \frac{d}{b_4}$$

3) TTP distributes d and the computed auxiliary random numbers using Shamir's (k, n) method and computes the following shares:

$$\tau_j \left[\frac{d}{b_1b_2}\right]_j = \tau_j \times \left[\frac{d}{b_1b_2}\right]_j,$$
$$\tau_j \left[\frac{d}{b_1}\right]_j = \tau_j \times \left[\frac{d}{b_1}\right]_j,$$
$$\tau_j \left[\frac{d}{b_2}\right]_j = \tau_j \times \left[\frac{d}{b_2}\right]_j,$$

$$\tau_j \left[\frac{d}{b_3 b_4}\right]_j = \tau_j \times \left[\frac{d}{b_3 b_4}\right]_j,$$
$$\tau_j \left[\frac{d}{b_3}\right]_j = \tau_j \times \left[\frac{d}{b_3}\right]_j,$$
$$\tau_j \left[\frac{d}{b_4}\right]_j = \tau_j \times \left[\frac{d}{b_4}\right]_j,$$
$$\tau_j [d]_j = \tau_j \times [d]_j$$

4) TTP sends b_i to Player U_i ($i = 1, \ldots, g$), random number d used for the last computation to the player who reconstructs the computation result, and shares of auxiliary random numbers above to server S.

Protocol 1.2: Encryption of secret inputs

1) Each player U_i computes the following for their secret input a_i and sends it to S:

$$b_i (a_i + 1) = b_i \times (a_i + 1)$$

Protocol 1.3: Secure computation

1) Server S computes the following for $j = 0, \ldots, n-1$.

$$\begin{aligned}\tau_j d\,[(a_1 a_2 + a_3 a_4) + 1]_j &= \left\{ b_1(a_1+1) \times b_2(a_2+1) \times \tau_j \left[\frac{d}{b_1 b_2}\right]_j - b_1(a_1+1) \right. \\ &\quad \left. \times \tau_j \left[\frac{d}{b_1}\right]_j - b_2(a_2+1) \times \tau_j \left[\frac{d}{b_2}\right]_j + 3 \times \tau_j [d]_j \right\} \\ &\quad + \left\{ b_3(a_3+1) \times b_4(a_4+1) \times \tau_j \left[\frac{d}{b_3 b_4}\right]_j \right. \\ &\quad \left. - b_3(a_3+1) \times \tau_j \left[\frac{d}{b_3}\right]_j - b_4(a_4+1) \times \tau_j \left[\frac{d}{b_4}\right]_j \right\}\end{aligned}$$

2) TTP distributes 0 using Shamir's (k, n) method and computes the following:

$$d[0]_j = d \times [0]_j \; (j = 0, \ldots, n-1)$$

3) TTP obtains $\tau_m d[a+1]_m$ from server S, divides it by τ_m, and computes $d[a+1]_m + d[0]_m$. However, $m = 1, \ldots, k-1$ and $a = a_1 a_2 + a_3 a_4$.

4) For computation with repetition, TTP sends the following to server S:

$$d[a+1]_m + d[0]_m,\ d[0]_0$$

5) Server S adds $d[0]_0$ to $d[a+1]_0$, reconstructs $d(a+1)$ as new input, and performs consecutive computations using the new input.

Protocol 1.4: Reconstruction process

1) In the last computation, server S and TTP send the following to the player:

$$d[a+1]_0,\ d[a+1]_m + d[0]_m,\ d[0]_0$$

2) The player adds $d[0]_0$ to $d[a+1]_0$, reconstructs $d(a+1)$, divides it by d, and subtracts it by 1 to obtain the computation result $(a_1 a_2 + a_3 a_4)$.

B. SECURITY OF THE PROPOSED METHOD

For g-input-1-output secure computation, regardless of the secure computation method used, if $g-1$ input and one output are known, the remaining input will be leaked. Moreover, if g inputs are known, the output will be leaked.

Therefore, we assume only the following semi-honest adversaries. Alternatively, we assume all conceivable and effective collusion in our proposed method, unlike the method described in Section II.C, where assumptions, such as the existence of a partial TTP, are made to avoid certain collusions, such as the collusion between the players and the computing servers.

If adversaries 1 and 2 are not given the information that each attacker requires, the proposed method is a secure computation that achieves a high level of security against all possible semi-honest adversaries. However, to simplify the evaluation, we first considered Adversary 0.

Adversary 0: The adversary knows the information of computing server S and attempts to learn the inputs and/or outputs of the computation.

Adversary 1: A player who knows $g-1$ inputs become an attacker. Adversary 1 knows $g-1$ secret inputs and the random numbers used to encrypt them. Furthermore, Adversary 1 knows the information from server S, and based on this, they attempt to learn the remaining secret input and/or the computation result.

Adversary 2: A player who knows $g-2$ inputs and output becomes an attacker. Adversary 2 knows $g-2$ inputs, random numbers used to encrypt them, and information required to reconstruct the computation result. Furthermore, Adversary 2 knows the information from server S, and based on this, they attempt to learn the remaining two secret inputs individually.

1) SECURITY AGAINST ADVERSARY 0

The adversary knows the shares of the auxiliary random numbers computed in Protocol 1.1 (processes by TTP). Additionally, Adversary 0 learns $b_i(a_i+1)$ from Protocol 1.2 (encryption of secret inputs). Moreover, from the computation result in Protocol 1.3 (secure computation), Adversary 0 learns $\tau_j d[a+1]_j$. If the computation is repeated (i.e., the computation result is used for the consecutive computation), Adversary 0 also learns $d[a+1]_j + d[0]_j$ and inputs $d(a+1)$ for the consecutive computation. However, let $j = 0, \ldots, k-1$, and $i = 1, \ldots, 4$.

If the overlapping information in the information shown above is removed and organized, Adversary 0 knows information A:

$$\begin{aligned}A = \left\{ \tau_j \left[\frac{d}{b_1 b_2}\right]_j, \tau_j \left[\frac{d}{b_1}\right]_j, \tau_j \left[\frac{d}{b_2}\right]_j, \tau_j \left[\frac{d}{b_3 b_4}\right]_j, \right. \\ \tau_j \left[\frac{d}{b_3}\right]_j, \tau_j \left[\frac{d}{b_4}\right]_j, \tau_j [d]_j, b_i (a_i + 1), \tau_j d\,[a+1]_j, \\ \left. d\,[a+1]_j + d\,[0]_j, d\,[0]_0, d\,(a+1) \right\}\end{aligned}$$

Regarding the information presented in A, the following attack can be considered (attack A): For simplicity, we let $n = 3$.

1) The attacker assumes random τ_1 and τ_2 and reconstructs the auxiliary random numbers computed in protocol 1.1 (processes by TTP).
2) The attacker computes the following from the reconstructed auxiliary random numbers: However, $(\ )''$ shows a reconstructed value in which the correctness of the value has not been verified, indicating that the value may or may not be the correct value as originally computed by the TTP:
$$\left(\frac{d}{b_1 b_2}\right)'' (d)'' = \left(\frac{d}{b_1}\right)'' \left(\frac{d}{b_2}\right)''$$
3) The attacker verifies whether the above equation holds, and if not, it changes τ_j and returns to Step (1). If the above holds, τ_j is included as one of the candidates; then, τ_j is changed to different values and returns to step (1).

Here, if the prime p used is sufficiently large (for example, a 128-bit prime number), attack A requires extensive computations to be realized.

However, if the attacker has infinite computational power, the candidates for τ_j can be narrowed. If the correct τ_j is computed, d is known; if d is used with d/b_1, b_1 is known and a_1 is leaked. The same applies to inputs other than a_1.

Regarding the computation result, if τ_j, d are known, $\tau_j d[a+1]_j$ can be solved to learn a.

When describing the protocol for our proposed method in Section III.A, $m_i = l = 2$ is used for simplicity. However, values that do not leak for any other m_i, l are not included in A; therefore, the same security is achieved even for any m_i, l.

Therefore, the proposed method does not realize information-theoretic security against Adversary 0; instead, it realizes computational security corresponding to the size of the prime p.

2) SECURITY AGAINST ADVERSARY 1

Consider the situation wherein Adversary 1 knows the values related to all inputs other than input a_1. Here, the difference from Adversary 0 is that, in addition to the information obtained by Adversary 0, Adversary 1 also knows all the inputs (other than a_1) and the random number used to encrypt them in Protocol 1.2 (encryption of secret inputs). Therefore, Adversary 1 knows information B. However, let $j = 0, \ldots, k-1$ and $i = 2, \ldots, 4$.

$$B = \left\{ \tau_j \left[\frac{d}{b_1 b_2}\right]_j, \tau_j \left[\frac{d}{b_1}\right]_j, \tau_j \left[\frac{d}{b_2}\right]_j, \tau_j \left[\frac{d}{b_3 b_4}\right]_j, \right.$$
$$\tau_j \left[\frac{d}{b_3}\right]_j, \tau_j \left[\frac{d}{b_4}\right]_j, \tau_j [d]_j, b_1 (a_1+1), a_i, b_i,$$
$$\left. \tau_j d [a+1]_j, d [a+1]_j + d [0]_j, d [0]_0, d (a+1) \right\}$$

The same attack as that in A is possible regarding the information shown in B. However, because Adversary 1 knows values related to all inputs other than input a_1, in step (2) of attack A, for example, the following is computed:

$$b_2 \left(\frac{d}{b_2}\right)'' = (d)''$$

Although this is less computationally complex than attack A, it still requires a significant computational cost corresponding to the size of p. If τ_1, τ_2 can be narrowed down (or computed) from the computation above, a_1 will be known, as in attack A, and the computation result will also be leaked.

This remains true even if the input a_1 unknown to Adversary 1 is changed to another input.

In Section III.A, $m_i = l = 2$ was used for simplicity. However, values that do not leak for any m_i, l are not included in B, and therefore the same security level is achieved.

Therefore, the proposed method does not realize information-theoretic security against Adversary 1; instead, it realizes computational security, which corresponds to the size of prime p.

3) SECURITY AGAINST ADVERSARY 2

Consider the situation wherein Adversary 2 knows all inputs other than a_1 and a_2, and information collected during the reconstruction of the computation result. The difference from Adversary 0 is that, in addition to the information of Adversary 0, Adversary 2 also knows all the inputs (other than a_1, a_2) and random numbers related to them in Protocol 1.2 (encryption of secret inputs). Moreover, Adversary 2 knows the value of d and the computation result $a = a_1 a_2 + a_3 a_4$ in Protocol 1.4 (reconstruction process). Therefore, Adversary 2 knows information C. However, let $i = 3, 4$.

$$C = \left\{ \tau_j \left[\frac{d}{b_1 b_2}\right]_j, \tau_j \left[\frac{d}{b_1}\right]_j, \tau_j \left[\frac{d}{b_2}\right]_j, \tau_j \left[\frac{d}{b_3 b_4}\right]_j, \tau_j [d]_j, \right.$$
$$\tau_j \left[\frac{d}{b_3}\right]_j, \tau_j \left[\frac{d}{b_4}\right]_j, b_1 (a_1+1), b_2 (a_2+1), a_i, b_i,$$
$$\left. \tau_j d [a+1]_j, d [a+1]_j + d [0]_j, d [0]_0, d (a+1), d, a \right\}$$

Adversary 2 knows the information on d used in the previous computation. Therefore, if the computation is not repeated, the adversary can attempt to restore d from $\tau_j [d]_j$ in step (2) of attack A and narrow down τ_j if they match. The computation shown in the previous section (security against Adversary 1) for step (2) of attack A is effective when the computation is repeated. However, both are computationally intensive (depending on the size of p).

By contrast, Adversary 2 knows $d[a+1]_j + d[0]_j$, $\tau_j d[a+1]_j$ when reconstructing the computation result. However, because the share of value 0 is added to $d[a+1]_j$, the values of τ_j cannot be determined simply by the ratio of the above shares. In addition, only one share of 0 is given to the player who reconstructs the result (in this case, Adversary 2), and because $k > 2$, the remaining shares of 0 cannot

be analyzed and computed. In addition, τ_j does not leak from d and resulting a. In addition, a_1a_2 can be determined from $a = a_1a_2 + a_3a_4$, but a_1 and a_2 cannot be separated and therefore are not leaked individually.

In Section III.A, $m_i = l = 2$ was used for simplicity. However, values that do not leak for any m_i, l are not included in C, and therefore the same security level is achieved.

Therefore, the proposed method does not realize information-theoretic security against Adversary 2; instead, it realizes computational security, which corresponds to the size of prime p.

From the arguments above, if random numbers τ_j are assumed to play the role of a security key, we can observe that contribution (2) is realized.

IV. DISCUSSION AND CONSIDERATION

A. OPTIMIZATION AND GENERALIZATION

In Section III, we described our proposed method by setting $N = 1$ and $n = k$. However, the proposed method can be extended in several ways.

The simplest method is to set $n = k = 3$ to minimize the processing performed by one server. However, a larger k increases the number of combinations of τ_j and therefore also increases the security of the proposed method. In addition, when $N > 2$, if $k = n = N$, the computations in Protocol 1.3 (secure computation) can be performed by one server for each j, and the process can be accelerated through parallel computation. However, even if multiple servers are used in parallel, there is no communication between the servers.

In addition, to simplify the explanation in our proposed algorithm, τ_j in $\tau_j[\varepsilon_h]_j$ is changed according to j. However, even if up to $k-1$ τ_j are the same, it still cannot be restored (however, if the first $k-1$ τ_j is set to $\tau_0 = 1$, because $k-1$ shares of value 0 must be sent to the player in protocol 1.4 (reconstruction process) and shares of 0 may be analyzed (because the adversary knows $k-1$ shares and input value 0), let τ_j from the first to $k-2$ be 1).

Therefore, if τ_j up to $j = 0, \ldots, k-3$ is $\tau_0 = 1$, and τ_j for $j = k-2, k-1$ are the same random number τ_1, the TTP only needs to divide the reconstruction result of $j = k-2, k-1$ by τ_1 during the reconstruction process. In addition, if $n > k$, if τ_j after that is updated every $k-1$, it can be handled with less τ_j for any n.

Next, we discuss security. Information-theoretic security means that if a certain piece of information is not known, the solution cannot be identified, even with infinite computational power. For example, in the case of (k, n) threshold secret sharing with $k = 2$, even if an attacker knows one of the shares but does not know the other remaining share, the correct solution cannot be identified because there is a solution for all combinations, even if the adversary assumes all possible values for the other share, thereby realizing information-theoretic security.

The same applies to the proposed method when only one secret input exists. For example, if $\tau_j[\varepsilon_1]_j$ is solved with all values in τ_j, information-theoretic security is achieved because there is a solution for all of ε_1. However, in conventional secret sharing, when there are two secret inputs, because there are all possible solutions for each input, there are a total of p^2 solutions when prime p is assumed. However, when considering $\tau_j[\varepsilon_1]_j$ and $\tau_j[\varepsilon_2]_j$ in the proposed method, there are solutions for ε_1 and ε_2 for all combinations of τ_j.

However, for example, when $\varepsilon_1 = \varepsilon_2$, p candidate solutions remain in the conventional method; however, the combination of τ_j that realizes $\varepsilon_1 = \varepsilon_2$ in the proposed method is considerably narrowed (except for the correct answer). This is also true when multiple ε_i values satisfy a specific relational expression.

If $\tau_j[\varepsilon_1]_j$, $\mu_j[\varepsilon_2]_j$ are used instead, the above problem is solved. However, in the proposed method, the shares included in the same equation must be encrypted by the same τ_j, so possible candidates might be narrowed down.

However, the above discussion assumes that the reconstruction is performed for all combinations of τ_j. Therefore, the security of the proposed method is computationally secure depending on the size of p when $N = 1$ or when the servers are managed by the same organization.

However, in the proposed method, if $N > 1$ and multiple servers can be managed independently in the same way as normal secret sharing, not all shares can be gathered. Therefore, even if attack A is performed, unknown shares cannot be identified and restored. Here, the proposed scheme can achieve information-theoretical security.

Thus, the proposed method is a secure computation method that can achieve information-theoretical security by independently managing servers when $N > 1$, and computational security is guaranteed even when servers are not managed independently or when $N = 1$.

B. ACCELERATING COMPUTATION USING TEE

In the proposed method, communication between the TTP and server S occurs when the computation is repeated, or the computation result is reconstructed.

However, a TEE is known as an alternative to realizing TTP. A TEE is a technology that enhances security by providing an isolated execution environment in a processor (CPU). An example of a TEEs is Intel's Software Guard Extension (Intel SGX) [29], [30], [31]. This extension function creates a cryptographically protected area on the CPU and executes a program therein. However, because Intel SGX uses public-key cryptography in communication, its security is generally computationally secure. However, the proposed method does not achieve information-theoretic security when only one computing server exists.

Therefore, for a single computing server, by setting server S to be a server equipped with Intel SGX, and the TTP used during the computation for repetition of computation and/or the reconstruction of computation result to be Intel SGX in that server, subsequent processes after protocol 1.2 (encryption of secret inputs) can be completed within a single server, enabling extremely high-speed secure computation. Intel SGX is available in Intel Core i7 and later and can be

widely used. The following describes the changes/additions made to the protocol shown in Section III when assuming using a TEE, in this case, Intel SGX.

Additional Step in Protocol 1.1: Processes by TTP:

5) The TTP secret shares 0, calculates the following, and sends $d[0]_m$ and τ_j to Intel SGX, and $d[0]_0$ to server S. However, let $m = 1, \ldots, k-1$.

$$d[0]_j = d \times [0]_j \ (j = 0, \ldots, n-1)$$

Changes in Protocol 1.3: Secure computation: Step (2) is removed, and the subsequent TTP computations are processed using Intel SGX. However, $d[0]_0$ in Step (4) of Protocol 1.3 (secure computation) and Step (1) of Protocol 1.4 (reconstruction phase) were not sent.

Intel SGX is known for its CrossTalk attack against random-number generation [32]. However, with the above changes, random number generation is performed by TTP, and Intel SGX performs only addition; hence, there is no problem. An attack called Cache Out [33] also exists wherein the private key is inferred from the public key, but this can be addressed by updating the BIOS version. Therefore, Intel SGX can be used to realize high-speed, computationally secure computation on a single computing server.

Hence, it is clear that contribution (3) can also be realized by implementing the TEE.

C. TRUSTED THIRD PARTY (TTP)

Generally, a TTP is assumed to be trustworthy and reliable, such that in the setting of multiparty (or secure) computation, it will not collude with the adversary or attack the actual computing server(s) to learn the secret inputs/output. However, information that TTP might obtain should be considered throughout the computation process.

In our proposed method, discussed in Section III, because the TTP does not handle any secret input(s) in Protocol 1.1 (processes by TTP), the secret input will not be leaked to the TTP. In addition, $\tau_m d[a+1]_m$ is known by the TTP in Protocol 1.3 (secure computation); however, because k shares are not gathered, the computation result is not leaked to the TTP.

In addition, by implementing a TEE, as discussed in Section IV.B, the TPP can delete all generated random numbers after transmitting all the computed information to the TEE, thereby eliminating the need for a secure key management model in the TTP. The TTP can execute steps (1)–(3) in protocol 1.1 (processes by TTP) locally, encrypt using a key shared in advance with the player, and erase all the random numbers used or generated. These steps can be performed before communication with the server or players is established, thereby allowing Protocol 1.1 (processes by TTP) to be secured independently.

Moreover, the TEE receives the security key and holds it until step (3) of Protocol 1.3 (secure computation). However, suppose that the program executed by server S and the TEE are made public (but the values used in computations are kept in their encrypted state), the TEE will not be able to perform any cheating, such as sending the security key to an adversary, because all processes can be monitored. Moreover, because the security key only acts as a session key, in this case, it can be deleted from the TEE after step (3) of Protocol 1.3 (secure computation).

However, when a TTP is assumed, there is always a question of ''wouldn't it be better and more efficient if all computations are performed by the TTP?''. Indeed, if all computations are entrusted to a single trusted TTP, secure computation is possible without problems. However, here, the TTP will have to know all secret inputs and the computation result, and the processing load of the TTP will increase. Furthermore, although the TTP is assumed to be secure, in the worst case, where the TTP is successfully attacked, all information stored by the TTP may be exposed.

Conversely, in our proposed method, the TTP does not know any secret inputs or computation results. Moreover, as explained previously, by making Protocol 1.1 (processes by TTP) independent, even if the TTP is attacked, no information is leaked because it has deleted all random numbers generated.

Moreover, the roles of the TTP in our proposed method are to generate random numbers in Protocol 1.1 (processes by TTP) and to provide computation assistance when computation with repetition is needed and when reconstructing the result; however, these processes are computationally light and do not increase the load of the TTP.

In addition, the proposed method described in Section III is explained using the operation of $a_1a_2 + a_3a_4$ as an example. However, in Protocol 1.2 (encryption of secure inputs), by setting $a_2 = a_3 = 1$, adding $a_1 + a_4$ is possible, by setting $a_4 = 0$, multiplication a_1a_2 is possible, and by setting $a_2 = a_4 = 0$, no actual computation will be performed.

Alternatively, TTP can be made to perform a fixed form of processing in protocol 1.1 (processes by TTP) by selecting an appropriate m_i, l in advance. Then, by adjusting the secret input to be either 0 or 1 in Protocol 1.2, various operations can be performed without informing the TTP of the actual type of operation needed during the secure computation.

D. CONSIDERATION FROM A BUSINESS PERSPECTIVE

With the introduction of new privacy and security laws, such as the General Data Protection Regulation (GDPR) by the European Union (EU) [34], the demand for secure computational services is increasing rapidly.

Recently, there have been businesses based on secure computation using homomorphic encryption, such as Partisia [35]. Secure computation services using secret sharing, such as Sharemind [36], have also been realized, but the independent management of servers is essential in realizing secure secret sharing. However, it is difficult to imagine a business model in which multiple companies with no conflict of interest manage each server independently and still profit from it.

Therefore, in most business models that use secret sharing, an organization often manages all computing servers and

provides a somewhat secure computation service. However, here, even if the customer entrusts their confidential information by secret-sharing it with the servers, it is still possible for the confidential information to be leaked if a single organization manages all the servers.

Moreover, even if the security is guaranteed by a binding agreement, such as signing a nondisclosure agreement (NDA) between the organization and the customers, it would be faster to receive the information directly than to secret shares to multiple servers owned by that single organization, which is more costly. In addition, even if the organization claims that the servers are managed independently by its subsidiaries or affiliated companies, they can still be accessed by the same parent company. In contrast, a single organization can manage the server in the proposed method.

In cryptography, a public key infrastructure (PKI) is an arrangement that binds public keys in public key encryption with the respective identities of entities, such as people or devices. The binding is commonly established through the registration and issuance of certificates by an independent, trusted party called the certificate authority (CA) [37].

Alternatively, in the security industry, a trusted third party called CA is used in the PKI to store, sign, and issue certificates validating a public key. This indicates that TTP is also considered viable as a business. When implementing this business model in our proposed method, the TTP is not involved in the actual secure computation; instead, it can profit simply by generating random numbers for users who want to participate in the secure computation.

In addition, our method is computationally light because it performs secure computations based on secret sharing and does not require communication when combined with a TEE. In addition, it is safe to publicize the entire computation process. Therefore, it is possible to execute relatively high-speed secure computation even without a high-performance server or high-speed communication network like other methods, and it can be implemented in any computing environment. Alternatively, the user can perform secure computation on their locally available PC or server or can be entrusted to an organization that provides a high-performance server as a service.

Therefore, in the proposed method, if the TTP can be realized and managed independently such as the CA, it will be simple and easy to develop various business opportunities, and it will be possible to further spread the use of secure computation and make secure computation services accessible to more users.

Therefore, contribution (4) is also realized.

V. IMPLEMENTATION ANALYSIS

The proposed method was implemented using C++. However, instead of assuming only a TTP, we implement our method using $N = 1$ computing server equipped with Intel SGX and one TTP to realize secure computation, as described in Section IV.B.

Tables 1–4 show the times taken (in microseconds) to realize secure computation on one machine with an Intel Xeon processor and parameters $n = k = 3$. However, for simplicity, in the implementation shown below, a single machine plays the computing server S, TTP, TEE, clients who input l-inputs, and players who reconstruct the result of the computation. The details of the environment used in the implementation are as follows:

- CPU: Intel Xeon E-2378G CPU @ 2.80GHz
- Memory: 16.00GB
- OS: Ubuntu Desktop 20.04.5LTS
- Code Language: C++

Tables 1 and 2 show the implementation results of the secure computation phase of our proposed method to realize inner-product computation of length l. The inner-product computation is often used for statistical calculations, such as variance and sum of squared deviation, and a wide range of applications, such as in a secure search of gene sequences and secure matrix calculation. In addition, because the computation is not repeated in the inner-product computation, communication is not required; therefore, steps (4) and (5) are not required. The specific algorithm for this corresponds to the case in which $m_i = 2$ in the proposed method. However, because the processes for pre-processing and encryption of secret inputs can be performed in advance, these operations are not subject to implementation.

In Table 1, we list the time taken for each step in Protocol 1.3 (secure computation). From Table 1, we learn that Step (1) requires the most amount of time; however, because no communication is required during this step, it can be completed fairly quickly, with the result showing that less than 75 msec is required to compute an inner product of two $l = 4,500$ elements in Step (1). Moreover, steps (2) and (3) are relatively light computationally and require considerably less time than step (1).

However, in the implementation below, we assume $n = k = 3$, indicating that the time shown in Step (1) is the total time taken to compute the three shares. We also include the detailed time taken to compute each share in Table 2. Therefore, if $N = 3$ computing servers are used instead, and the computation of Step (1) is performed in parallel (as explained in Section IV.A), we can achieve a faster computing time, albeit at the cost of communication required to collect the final shares during reconstruction.

Table 3 shows the time required to perform a secure computation $l = 100$ for each $m_i = 3, 4, 5$, and 6. From the result, we learn that the time needed to perform secure computation doubled when m_i increased; however, our proposed method still achieved fast computing speed because no communications were required (other than with the TTP). Table 4 shows the time taken to compute each share in Step (1).

In our future study, we will also include a detailed implementation of conventional methods and perform a detailed comparison of the computation time of each method. The implementation described in this section was performed

TABLE 1. Time in microseconds to realize secure computation of an inner-product ($m_i = 2$) of two l-inputs.

Parameter l	Time taken in μs		
	Step (1)	Step (2)	Step (3)
$l = 100$	1,505	30	17
$l = 500$	7,842	56	18
$l = 1,000$	16,061	56	18
$l = 1,500$	24,551	68	20
$l = 2,000$	32,368	58	18
$l = 2,500$	40,532	69	19
$l = 3,000$	48,628	56	18
$l = 3,500$	57,450	60	18
$l = 4,000$	65,961	58	18
$l = 4,500$	74,006	60	19

TABLE 2. Time in microseconds to compute each share in Step (1) of protocol 1.3 (secure computation).

Parameter l	Time taken in μs		
	Share 1	Share 2	Share 3
$l = 100$	503	493	495
$l = 500$	2,668	2,584	2,573
$l = 1,000$	5,465	5,339	5,238
$l = 1,500$	8,255	8,274	7,864
$l = 2,000$	10,944	10,738	10,657
$l = 2,500$	13,668	13,503	13,339
$l = 3,000$	16,448	16,181	15,975
$l = 3,500$	19,411	19,037	18,965
$l = 4,000$	21,907	21,606	21,410
$l = 4,500$	25,213	24,432	24,313

TABLE 3. Time in microseconds to realize secure computation of $l = 100$ for each $m_i = 3, 4, 5, 6$.

Parameter l	Time taken in μs		
	Step (1)	Step (2)	Step (3)
$m_i = 3, l = 100$	3,261	46	17
$m_i = 4, l = 100$	7,064	56	17
$m_i = 5, l = 100$	15,597	60	18
$m_i = 6, l = 100$	32,314	58	18

TABLE 4. Time in microseconds to compute each share in Step (1) when $n = k = 3$.

Parameter l	Time taken in μs		
	Share 1	Share 2	Share 3
$m_i = 3, l = 100$	1,095	1,079	1,072
$m_i = 4, l = 100$	2,379	2,334	2,336
$m_i = 5, l = 100$	5,388	5,126	5,057
$m_i = 6, l = 100$	10,874	10,833	10,580

without parallelization. However, in secure computation based on secret sharing with multiple inputs, parallel computation is essential to improve the computation speed. Therefore, in our future study, we will also consider how to translate our implementation into parallel computation to achieve a faster computing speed.

VI. CONCLUSION

This study used TTP to show that secure computation using (k, n) threshold secret sharing can be realized with a single computing server. Moreover, we showed that, in our proposed method, information-theoretic security could be realized under the condition that the number of computing servers $N > 1$, whereas computational security is guaranteed even when $N = 1$. We also discussed using TEEs to replace TTP and showed that TEEs, such as Intel SGX, can be implemented to reduce communication and enable high-speed processing.

In a future study, we will consider the implementation of parallel computation and perform a detailed comparison with other conventional methods. We will also consider using TEEs other than Intel SGX in the future.

ACKNOWLEDGMENT

The authors would like to thank Kota Shirai, Daichi Kuroi, Prof. Masaki Inamura, and the people of GMO Cybersecurity by Ierae Inc., for their cooperation in the implementation.

REFERENCES

[1] P. N. Smart and F. Vercauteren, "Fully homomorphic encryption with relatively small key and ciphertext sizes," in *Public Key Cryptography—PKC* (Lecture Notes in Computer Science), vol. 6056, P. Q. Nguyen and D. Pointcheval, Eds. Berlin, Germany: Springer, Berlin, Heidelberg, 2010, pp. 420–443.

[2] M. V. Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan, "Fully homomorphic encryption over the integers," in *Advances in Cryptology—EUROCRYPT* (Lecture Notes in Computer Science), vol. 6110, H. Gilbert, Ed. Berlin, Germany: Springer, 2010, pp. 24–43.

[3] R. Bendlin, I. Damgård, C. Orlandi, and S. Zakarias, "Semi-homomorphic encryption and multiparty computation," in *Advances in Cryptology—EUROCRYPT* (Lecture Notes in Computer Science), vol. 6632, K. G. Paterson, Ed. Berlin, Germany: Springer, 2011, pp. 169–188.

[4] Z. Brakerski and V. Vaikuntanathan, "Fully homomorphic encryption from ring-LWE and security for key dependent messages," in *Advances in Cryptology—CRYPTO* (Lecture Notes in Computer Science), vol. 6841, P. Rogaway, Ed. Berlin, Germany: Springer, 2011, pp. 505–524.

[5] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(Leveled) fully homomorphic encryption without bootstrapping," in *Proc. 3rd Innov. Theor. Comput. Sci. Conf. (ITCS)*. New York, NY, USA: Association for Computing Machinery, 2012, pp. 309–325.

[6] I. Damgård, V. Pastro, N. Smart, and S. Zakarias, "Multiparty computation from somewhat homomorphic encryption," in *Advances in Cryptology—CRYPTO* (Lecture Notes in Computer Science), vol. 7417, R. Safavi-Naini and R. Canetti, Eds. Berlin, Germany: Springer, 2012, pp. 643–662.

[7] I. Damgård, M. Keller, E. Larraia, V. Pastro, P. Scholl, and P. Nigel Smart, "Practical covertly secure MPC for dishonest majority—Or: Breaking the SPDZ limits," in *Computer Security—ESORICS* (Lecture Notes in Computer Science), vol. 8134, J. Crampton, S. Jajodia, and K. Mayes, Eds. Berlin, Germany: Springer, 2013, pp. 1–18.

[8] M. Keller, "MP-SPDZ: A versatile framework for multi-party computation," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*. New York, NY, USA: Association for Computing Machinery, 2020, pp. 1575–1590.

[9] M. Ben-Or, S. Goldwasser, and A. Wigderson, "Completeness theorems for non-cryptographic fault-tolerant distributed computation," in *Proc. 20th Annu. ACM Symp. Theory Comput. (STOC)*. New York, NY, USA: Association for Computing Machinery, 1988, pp. 1–10.

[10] D. Chaum, C. Crépeau, and I. Damgard, "Multiparty unconditionally secure protocols," in *Proc. 20th Annu. ACM Symp. Theory Comput. (STOC)*. New York, NY, USA: Association for Computing Machinery, 1988, pp. 11–19.

[11] R. Gennaro, O. Michael Rabin, and T. Rabin, "Simplified VSS and fast-track multiparty computations with applications to threshold cryptography," in *Proc. 17th Annu. ACM Symp. Princ. Distrib. Comput. (PODC)*. New York, NY, USA: Association for Computing Machinery, 1998, pp. 101–111.

[12] R. Cramer, I. Damgård, and U. Maurer, "General secure multi-party computation from any linear secret-sharing scheme," in *Advances in Cryptology—EUROCRYPT* (Lecture Notes in Computer Science), vol. 1807, B. Preneel, Ed. Berlin, Germany: Springer, 2000, pp. 316–334.

[13] T. Araki, J. Furukawa, Y. Lindell, A. Nof, and K. Ohara, "High-throughput semi-honest secure three-party computation with an honest majority," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*. New York, NY, USA: Association for Computing Machinery, 2016, pp. 805–817.

[14] A. A. A. M. Kamal and K. Iwamura, "(Server-aided) two-party multiplication of encrypted shares using (k, n) threshold secret sharing with N≥k servers," *IEEE Access*, vol. 9, pp. 113117–113129, 2021.

[15] A. A. A. M. Kamal and K. Iwamura, "Searchable encryption using secret sharing scheme that realizes direct search of encrypted documents and disjunctive search of multiple keywords," *J. Inf. Secur. Appl.*, vol. 59, Jun. 2021, Art. no. 102824.

[16] C. Gentry, "A fully homomorphic encryption scheme," Ph.D. dissertation, Dept. Comput. Sci., Stanford Univ., Stanford, CA, USA, 2009.

[17] T. Shingu, A. Ken, A. A. A. M. Kamal, and K. Iwamura, "Secure computation without changing polynomial degree in (k, n) secret sharing scheme, (in Japanese)," *J. Inf. Process.*, vol. 59, no. 3, pp. 1038–1049, 2018.

[18] A. A. A. Mohd Kamal and K. Iwamura, "Conditionally secure multiparty computation using secret sharing scheme for n<2k-1 (short paper)," in *Proc. 15th Annu. Conf. Privacy, Secur. Trust (PST)*, Aug. 2017, pp. 2225–2255.

[19] K. Tokita and K. Iwamura, "Fast secure computation method based on a secret sharing scheme even for n<2k-1 which can deal with the secret zero," *IEEJ Trans. Electron., Inf. Syst.*, vol. 138, no. 12, pp. 1634–1645, Dec. 2018.

[20] K. Iwamura and A. Kamal, "Secure computation by secret sharing using input encrypted with random number," in *Proc. 18th Int. Conf. Secur. Cryptography*. Setúbal, Portugal: SciTePress, May 2021, pp. 540–547.

[21] S. Ochiai and K. Iwamura, "New approach to dishonest-majority secure multiparty computation for malicious adversaries when n<2k-1," in *Proc. 8th Int. Symp. Comput. Netw. Workshops (CANDARW)*, Nov. 2020, pp. 355–361.

[22] H. Morita, N. Attrapadung, T. Teruya, S. Ohata, K. Nuida, and G. Hanaoka, "Constant-round client-aided secure comparison protocol," in *Computer Security. ESORICS* (Lecture Notes in Computer Science), vol. 11099, J. Lopez, J. Zhou, and M. Soriano, Eds. Cham, Switzerland: Springer, 2018, pp. 395–415.

[23] K. Huang, X. Liu, S. Fu, D. Guo, and M. Xu, "A lightweight privacy-preserving CNN feature extraction framework for mobile sensing," *IEEE Trans. Dependable Secure Comput.*, vol. 18, no. 3, pp. 1441–1455, Jun. 2021.

[24] Y. Liu, Z. Ma, X. Liu, S. Ma, and K. Ren, "Privacy-preserving object detection for medical images with faster R-CNN," *IEEE Trans. Inf. Forensics Security*, vol. 17, pp. 69–84, 2022.

[25] K. Iwamura, A. A. A. Mohd Kamal, and M. Inamura, "TTP-aided secure computation using secret sharing with only one computing server," in *Proc. ACM Asia Conf. Comput. Commun. Secur.* New York, NY, USA: Association for Computing Machinery, May 2022, pp. 1243–1245.

[26] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, pp. 612–613, Nov. 1979.

[27] J. Kurihara, S. Kiyomoto, K. Fukushima, and T. Tanaka, "A new (k, n)-threshold secret sharing scheme and its extension," in *Information Security. ISC* (Lecture Notes in Computer Science), vol. 5222, T. C. Wu, C. L. Lei, V. Rijmen, and D. T. Lee, Eds. Berlin, Germany: Springer, 2008, pp. 455–470.

[28] D. Beaver, "Efficient multiparty protocols using circuit randomization," in *Advances in Cryptology—CRYPTO* (Lecture Notes in Computer Science), vol. 576, J. Feigenbaum, Ed. Berlin, Germany: Springer, 1992, pp. 420–432.

[29] Intel. *Intel/Linux-SGX: Intel SGX for Linux**. GitHub. Accessed: Oct. 4, 2022. [Online]. Available: https://github.com/intel/linux-sgx

[30] Intel. *Intel® Software Guard Extensions SDK Installation Guide*. Accessed: Oct. 4, 2022. [Online]. Available: https://www.intel.com/content/www/us/en/developer/articles/guide/sgx-sdk-installation-guide.html

[31] Intel. (2022). *Intel® Software Guard Extensions*. Accessed: Oct. 4, 2022. [Online]. Available: https://www.intel.com/content/www/us/en/developer/tools/software-guard-extensions/overview.html

[32] H. Ragab, A. Milburn, K. Razavi, H. Bos, and C. Giuffrida, "CrossTalk: Speculative data leaks across cores are real," in *Proc. IEEE Symp. Secur. Privacy (SP)*, 2021, pp. 1852–1867.

[33] S. V. Schaik, M. Minkin, A. Kwong, D. Genkin, and Y. Yarom, "CacheOut: Leaking data on Intel CPUs via cache evictions," in *Proc. IEEE Symp. Secur. Privacy (SP)*, Apr. 2021, pp. 339–354.

[34] B. Wolford. *What is GDPR, the EU's New Data Protection Law?* Accessed: Oct. 4, 2022. [Online]. Available: https://gdpr.eu/what-is-gdpr/

[35] Partisia. (Feb. 3, 2021). *MPC Protocols*. Accessed: Oct. 4, 2022. [Online]. Available: https://partisia.com/smc-protocols/

[36] Sharemind. *Privacy Enhancing Technology for Data-Driven Business*. Accessed: Oct. 4, 2022. [Online]. Available: https://sharemind.cyber.ee/

[37] R. Housley, "Public key infrastructure (PKI)," in *The Internet Encyclopedia*, H. Bidgoli Ed. Hoboken, NJ, USA: Wiley, 2004.

KEIICHI IWAMURA (Member, IEEE) received the B.S. and M.S. degrees in information engineering from Kyushu University, Japan, in 1980 and 1982, respectively, and the Ph.D. degree from The University of Tokyo.

From 1982 to 2006, he belongs to Canon Inc. He is currently a Professor with the Tokyo University of Science. His research interests include coding theory, information security, and digital watermarking. He is a fellow of the Information Processing Society of Japan and the Chairperson of the Technical Committee of Information Hiding and its Criteria for Evaluation and Technical Committee of Enriched Multimedia, Institute of Electronics, and Information and Communication Engineers, Japan.

AHMAD AKMAL AMINUDDIN MOHD KAMAL (Member, IEEE) was born in Penang, Malaysia, in 1994. He received the B.S. and M.S. degrees in electrical engineering and the Ph.D. degree in engineering from the Tokyo University of Science, Japan, in 2017, 2019, and 2022, respectively.

He is currently an Assistant Professor with the Tokyo University of Science. His research interests include information security and multiparty computation using secret sharing and its application into searchable encryption.