Improvement of Secure Multi-Party Multiplication of (k, n) Threshold Secret Sharing using Only $N = k$ Servers

Ahmad Akmal Aminuddin Mohd Kamal[1] and Keiichi Iwamura[2]
[1]*Graduate School of Engineering, Department of Electrical Engineering, Tokyo University of Science, Tokyo, Japan*
[2]*Faculty of Engineering, Department of Electrical Engineering, Tokyo University of Science, Tokyo, Japan*

Keywords: Secure Multi-Party Computation, MPC, Secure Multiplication, (k, n) Threshold Secret Sharing, Information Theoretic Secure.

Abstract: Secure multi-party computation (MPC) allows a set of n servers to jointly compute an arbitrary function of their inputs, without revealing these inputs to each other. A (k, n) threshold secret sharing is a protocol in which a single secret is divided into n shares and the secret can be recovered from a threshold k shares. Typically, multiplication of (k, n) secret sharing will result in increase of polynomial degree from $k-1$ to $2k-2$, thus increasing the number of shares required from k to $2k-1$. Since each server typically hold only one share, the number of servers required in MPC will also increase from k to $2k-1$. Therefore, a set of n servers can compute multiplication securely if the adversary corrupts at most $k-1 < n/2$ of the servers. In this paper, we differentiate the number of servers N required and parameter n of (k, n) secret sharing scheme, and propose a method of computing $(k-1)$ sharing of multiplication ab by using only $N = k$ servers. By allowing each server to hold two shares, we realize MPC of multiplication with the setting of $N = k, n \geq 2k-1$. We also show that our proposed method is information theoretic secure against a semi-honest adversary.

1 INTRODUCTION

1.1 Background

In recent years, advancement of IoT ecosystem and big data had attracted considerable anticipation that it will be possible to utilize big data to obtain valuable statistical data. Here, big data refer to large quantities of unstructured and semi structured data. Analyzation of these data allows researchers and businesses to extract important and useful information. However, since big data also includes individuals' privacy information, there is a risk that their privacy information can be leaked if it is not used correctly. Therefore, a method that allows for the utilization of big data while protecting sensitive information such as individuals' privacy information is required. One of the methods that can realize this is known as *secure multi-party computation* (Yao, 1982). Secure multi-party computation allows for a set of servers to jointly compute an arbitrary function of their inputs, without revealing these inputs to each other. Typically, there are two main techniques that had been proposed to realize secure multi-party computation: homomorphic encryption (Bendlin et al. 2011; Brakerski et al., 2011; Damgard et al., 2012; Damgard et al., 2013; Gentry, 2009) and secret sharing schemes (Ben-Or et al., 1988; Chaum et al., 1988; Cramer et al., 2000; Gennaro et al., 1998; Shingu et al., 2016; Watanabe et al., 2015). However, homomorphic encryption is known to be typically computationally very expensive in term of computation cost. Therefore, secret sharing schemes that have a relatively low computational cost are preferable to homomorphic encryption when considering utilization in a cloud system.

A secret sharing scheme is a protocol in which a single secret is divided into shares, which are then distributed. An example of a secret sharing scheme is Shamir's (k, n) threshold secret sharing scheme (Shamir, 1979). It divides a secret s into an n number of shares, distributes the shares, and restores the original secret s from a threshold k number of shares. Any $k-1$ or smaller number of shares reveals nothing about the secret.

The classical result of secure multi-party computation using secret sharing scheme state that n servers can compute any function in such a way that any subset of up to $k-1 < n/2$ servers obtains no information about the other servers' inputs, except for

Kamal, A. and Iwamura, K.
Improvement of Secure Multi-Party Multiplication of (k, n) Threshold Secret Sharing Using Only N = k Servers.
DOI: 10.5220/0010206400770088
In *Proceedings of the 7th International Conference on Information Systems Security and Privacy (ICISSP 2021)*, pages 77-88
ISBN: 978-989-758-491-6

what can be derived from the public information (Ben-Or et al., 1988; Hirt, 2001). Conventional methods of secure multi-party computation using Shamir's (k, n) secret sharing scheme perform addition by locally adding the shares together. However, this is not so in the case of multiplication. For example, let secrets a and b be encoded by polynomials $f(x)$ and $g(x)$ of $(k-1)$ degree. Note that the free coefficient of the polynomial $h(x) = f(x)g(x)$ is ab. However, the problems of using $h(x)$ to encode the product of a times b is that the degree of $h(x)$ increase from $k-1$ to $2k-2$. In most conventional methods, this poses no problem of interpolating $h(x)$ from its n shares since it is assumed that parameter $n \geq 2k-1$. Since each server holds only one share for each secret, this means that, for each multiplication performed, the number of servers required will increase from k to $2k-1$.

Shingu et al. proposed a solution for multiplication method called the TUS method (Shingu et al, 2016). In this method, the secret is first encrypted with a random number; when performing multiplication, the encrypted secret is momentarily restored as a scalar value and multiplication is realized using the $(scalar\ value \times polynomial)$ approach to prevent an increase in the polynomial degree. However, in the TUS method, there is a limitation where input with a value of 0 is not allowed in the protocol.

On the other hand, Watanabe et al. proposed a solution by taking an approach of differentiating the relationship between the number of servers required N and parameter n of Shamir's (k, n) secret sharing (Watanabe et al., 2015). In other word, Watanabe et al. had proposed a method using $N \leq k$ servers to realize $n \geq 2k-1$ multiplication. However, this method still did not solve the problem of increase in the polynomial degree. Therefore, although the multiplication was performed using only $N = k$ servers, shares required to restore the multiplication result are $2k-1$ and not k.

1.2 Our Results

In this study, we focus on solving the problem of increase in polynomial degree during **multiplication.** We propose a new method of multiplication that could compute $k-1$ sharing of multiplication ab using only $N = k$ servers. The contributions of this paper can be summarized as follows:

Our Contributions:

- We propose a new protocol that allows for multiplication with the setting of $n \geq 2k-1$ to be performed using only $N = k$ servers, and show that $k-1$ sharing of ab can be computed by using only $N = k$ servers. (we also include protocols for addition and scalar multiplication in Appendices 1 and 2)
- We present a clear security evaluation and show that our proposed method is secure against semi-honest adversaries.
- Finally, we present a clear evaluation of efficiency of our method. In addition, we also present a comparison between the methods in (Watanabe et al., 2015; Shingu et al., 2016).

System Model:

In this paper, we assume a client/server model, where the clients (e.g. the owner of the secret information a, b) send shares of their inputs to n number of servers. The servers then carry out the computation and return the results to them without learning anything about secret information a and b. This model is widely used nowadays and in fact is the business model used in Cybernetica (Sharemind). In addition, we assume a semi-honest adversary, where the adversary follows the protocol specification but may try to learn more than is allowed by the protocol, with at most $k-1$ corrupted servers. We also assume that secure communication exists between the client and the servers.

The organization of this paper is as follows. In Section 2, we present preliminaries. In Section 3, we explain the related works. In Section 4, we present our new protocol for multiplication. In Section 5, we discuss the security of our proposed method of multiplication. In addition, in Chapter 6, we evaluate our proposed method. Finally, in Chapter 7, we show the comparison between our proposed method and conventional methods.

2 PRELIMINARIES

In this section, we introduce some notations and known techniques.

2.1 (k, n) Threshold Secret Sharing Scheme

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

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

– Any k or greater number of shares will allow for the reconstruction of the original secret information s.

The classic methods for the (k,n) threshold secret sharing scheme is Shamir's (k,n) threshold secret sharing scheme, proposed by Shamir (Shamir, 1979) (Shamir's (k,n) method). In our protocol, all computations are performed in finite field $GF(p)$ and shares of secret information s is represented by $\overline{[s]}_i$.

The protocol for the distribution and reconstruction of Shamir's (k,n) method is as follows.

Distribution Protocol:

1. The dealer selects n number of x_i $(i = 0,1,\dots,n-1)$ and assigns them as the server ID.
2. The dealer selects $k-1$ random numbers α_l $(l = 1,2,\dots,k-1)$ and generates a random polynomial $W(x_i)$.

$$\overline{[s]}_i = W(x_i) = s + \alpha_1 x_i + \alpha_2 {x_i}^2 + \dots + \alpha_{k-1} {x_i}^{k-1}$$

3. The dealer then inserts the server ID into x_i, calculates the shares $\overline{[s]}_i$ corresponding to each ID, and distributes them.

Restoration Protocol:

1. The player who wishes to restore the original secret collects k shares $\overline{[s]}_j$ $(j = 0,\dots,k-1)$.
2. The player restores the original secret s by using Lagrange's Interpolation.

$$s = \sum_{i=1}^{n} \prod_{j=1,\, j\neq i}^{n} \frac{\alpha_j}{\alpha_j - \alpha_i} s_i$$

2.2 Multiplication based on Shamir's (k,n) Method

Let a and b be two secrets. Shares of each secret are produced by Shamir's (k,n) method as shown below and are distributed to n servers. Note that $i = 0,1,\dots,n-1$.

$$\overline{[a]}_i = a + \alpha_1 x_i + \alpha_2 {x_i}^2 + \dots + \alpha_{k-1} {x_i}^{k-1}$$
$$\overline{[b]}_i = b + \beta_1 x_i + \beta_2 {x_i}^2 + \dots + \beta_{k-1} {x_i}^{k-1}$$

Each server then computes the multiplication between shares of a and b, and produces $\overline{[ab]}_i$ as shown below.

$$\overline{[ab]}_i = ab + \dots + (\alpha_{k-1}\beta_{k-1}) {x_i}^{2k-2}$$

Although secrets a and b are shared using polynomials of $(k-1)$ degree, the result of multiplication ab is a polynomial of $(2k-2)$ degree. Therefore, the problem with conventional method of multiplication of Shamir's (k,n) method is that the number of shares required to reconstruct ab increase from k to $2k-1$. Thus, the following Theorem 1 was proposed for the passive model (Ben-Or et al., 1988).

Theorem 1. In the passive mode, a set $\mathcal{P} = \{P_1, \dots, P_n\}$ of n servers can compute every specification securely if and only if the adversary corrupts at most $k-1 < n/2$ of the servers.

2.3 Multiplication of Shares using Recombination Vector

As mentioned in Section 2.2, the result of multiplication of two polynomials of degree $(k-1)$ will be a polynomial of degree $(2k-2)$. Note that $n \geq 2k-1$ implies that the n product shares are sufficient for recovering ab. However, any further multiplication will raise the degree, and once the degree passes n, there will be not have enough points for the interpolation. Hence, a $(k-1)$ sharing of ab can be achieved by using *recombination vector* as shown in (Cramer et al. 2015).

To better understand this, let us assume that the parameter $k = 2, n = 2k-1 = 3$, the resulting multiplication will be a quadratic polynomials $y(x_i) = \alpha_0 + \alpha_1 x_i + \alpha_2 {x_i}^2$, where α_0 is the result of the multiplication. Since $n = 3$, the shares for each server ID are as follows.

$$y(1) = \alpha_0 + \alpha_1 + \alpha_2$$
$$y(2) = \alpha_0 + 2\alpha_1 + 4\alpha_2$$
$$y(3) = \alpha_0 + 3\alpha_1 + 9\alpha_2$$

By solving the equations above, we can state that multiplication result α_0 can always be computed from the shares $y(1), y(2)$ and $y(3)$ as $\alpha_0 = 3y(1) - 3y(2) + y(3)$. This formula was found using simple Gaussian elimination, but is also given by the Lagrange interpolation formula, where $r = (3,-3,1)$ is known as the ***recombination vector***.

More precisely, each party first shares its value of multiplication result $\overline{[ab]}_i$ using polynomials of $(k-1)$ degree to all parties. The parties then locally combine their shares by an inner product with the *recombination vector*. By this, each party will hold $(k-1)$ sharing of ab. However, the problem with this method is that it still requires $n > 2k - 1$ number of servers, therefore increasing the total operation cost of the system.

3 RELATED WORKS

3.1 Watanabe Method

Typically, in a (k, n) threshold secret sharing scheme, a server possesses only one share. When multiplication of shares is performed, the number of servers required will also increase from k to $2k - 1$.

Watanabe et al. solved this problem by allowing a server to hold two shares (Watanabe et al., 2015). However, this method does not solve the problem of increase in degree of polynomial from $k - 1$ to $2k - 2$. This means that the number of shares required to reconstruct the result remain at $2k - 1$ instead of k. Therefore, the communication cost between the client and the servers remain the same as all conventional methods. Our method solves this by proposing method of computing $(k - 1)$ sharing of ab using only $N = k$ servers.

Due to the page limit, we only included the distribution and multiplication protocols below. Note that variables a, b and all random numbers generated are derived from finite field $GF(p)$ and all computations are performed under finite field $GF(p)$.

Distribution Protocol:

1. Players A and B each generates $2n$ shares from secrets a and b and distributes $\overline{[a]}_i, \overline{[b]}_i$ $(i = 0, \dots, n-1)$ to n servers S_i.
2. Player A generates a random number r_A and distributes $\overline{[r_A a]}_n, \dots, \overline{[r_A a]}_{2n-1}$ to n servers S_i. Then, distributes shares $\overline{[r_A]}_i$ of r_A to n servers S_i.
3. Player B generates a random number r_B and distributes $\overline{[r_B b]}_n, \dots, \overline{[r_B b]}_{2n-1}$ to n servers S_i. Then, distributes shares $\overline{[r_B]}_i$ of r_B to n servers S_i.

Multiplication Protocol:

1. Each server calculates the following:

$$\overline{[ab]}_i = \overline{[a]}_i \times \overline{[b]}_i \ (i = 0, 1, \dots, n-1)$$

$$\overline{[r_A r_B ab]}_{i+k} = \overline{[r_A a]}_{i+k} \times \overline{[r_B b]}_{i+k}$$

3.2 The TUS Method

Shingu et al. proposed a 2-inputs-1-output multi-party computation named the TUS method (Shingu et al., 2016), where the secret (e.g. a) is first encrypted with a random number (e.g. α). When performing multiplication, the encrypted secret is momentarily restored as a scalar value (e.g. αa) and multiplication is realized using the $(scalar\ value \times polynomial)$ approach to prevent an increase in the polynomial degree. However, the TUS method introduced another problem: when the reconstructed scalar value $\alpha a = 0$, the adversary will learn that secret $a = 0$. Therefore, condition where the secret does not include the value 0 is required.

Due to the page limit, we only included the distribution and multiplication protocols below. Note that variables a, b and all random numbers generated are derived from finite field $GF(p)$, provided that the secret inputs a, b and all random numbers do not include 0.

Distribution Protocol:

1. Players A and B each selects k random numbers α_j, β_j $(j = 0, 1, \dots, k-1)$ and computes the value of $\alpha = \prod_{j=0}^{k-1} \alpha_j$ and $\beta = \prod_{j=0}^{k-1} \beta_j$, respectively.
2. Player A computes $\alpha a = \alpha \times a$ and distributes $\alpha a, \alpha_j$ to n servers using Shamir's (k, n) method.
3. Player B computes $\beta b = \beta \times b$ and distributes $\beta b, \beta_j$ to n servers using Shamir's (k, n) method.
4. Each server S_i $(i = 0, 1, \dots, n-1)$ holds the following as set of shares about secrets a, b:

$$[a]_i = \overline{[\alpha a]}_i, \overline{[\alpha_0]}_i, \dots, \overline{[\alpha_{k-1}]}_i,$$

$$[b]_i = \overline{[\beta b]}_i, \overline{[\beta_0]}_i, \dots, \overline{[\beta_{k-1}]}_i$$

Multiplication Protocol:

1. One of the servers (here, we assume server S_0) collects $\overline{[\alpha a]}_j$ from k servers. Server S_0 then restores αa and sends it to all servers S_i.
2. Servers S_i compute $\overline{[\alpha\beta ab]}_i = \alpha a \times \overline{[\beta b]}_i$.
3. k number of servers S_j each collect shares $\overline{[\alpha_0]}_\ell, \dots, \overline{[\alpha_{k-1}]}_\ell, \overline{[\beta_0]}_\ell, \dots, \overline{[\beta_{k-1}]}_\ell$ $(\ell = 0, 1, \dots, k-1)$ and restore α_j, β_j. Servers S_j then calculate $\alpha_j\beta_j = \alpha_j \times \beta_j$.
4. Servers S_j distribute $\alpha_j\beta_j$ to all servers S_i by using Shamir's (k, n) method.
5. Each server S_i now holds the following as a set of shares for the result of ab.

$$[ab]_i = \overline{[\alpha\beta ab]}_i, \overline{[\alpha_0\beta_0]}_i, \dots, \overline{[\alpha_{k-1}\beta_{k-1}]}_i$$

4 PROPOSED METHOD OF MULTIPLICATION

Our protocol focus on an approach that differentiate between parameter N, which is the number of servers that is actually needed, and parameter n of the (k, n) threshold secret sharing scheme, and realizes multiplication under a setting of $N = k, n \geq 2k - 1$. In addition, to solve the problem of Watanabe method, where the result of multiplication can only be reconstructed by collecting $2k$ shares from k servers, we propose a new method of reducing the polynomial degree of ab from $2k - 2$ to $k - 1$ by using *recombination vector* with only $N = k$ servers.

Below, for ease of understanding, we demonstrate multiplication under the setting of $N = k, n \geq 2k - 1$. In the protocol below, secrets a, b, all random numbers and all computations are performed in finite field $GF(p)$.

Notation:

— $[a]_i$: Share of a for server S_i where the number of shares required for reconstructing a is k

— $[\alpha_1\beta_1 ab]_i^*$: Share of $\alpha_1\beta_1 ab$ for server S_i where the number of shares required for reconstructing $\alpha_1\beta_1 ab$ is $2k - 1$.

Distribution Protocol:

1. Player A generates $2k$ random numbers $\alpha_{1,0}, \dots, \alpha_{1,k-1}, \alpha_{2,0}, \dots, \alpha_{2,k-1}$ and computes the following.

$$\alpha_1 = \prod_{j=0}^{k-1} \alpha_{1,j}$$

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

2. Player A generates $2k$ shares of secret a using Shamir's $(k, 2k)$ method and computes the following.

$$[\alpha_1 a]_0 = \alpha_1 \times [a]_0, \dots, [\alpha_1 a]_{k-1} = \alpha_1 \times [a]_{k-1}$$

$$[\alpha_2 a]_k = \alpha_2 \times [a]_k, \dots, [\alpha_2 a]_{2n-1} = \alpha_2 \times [a]_{2n-1}$$

3. Player A sends $[\alpha_1 a]_i, [\alpha_2 a]_{i+k}, \alpha_{1,i}, \alpha_{2,i}$ to server S_i $(i = 0, 1, \dots, k - 1)$.
4. Player B generates $2k$ random numbers $\beta_{1,0}, \dots, \beta_{1,k-1}, \beta_{2,0}, \dots, \beta_{2,k-1}$ and computes the following.

$$\beta_1 = \prod_{j=0}^{k-1} \beta_{1,j}$$

$$\beta_2 = \prod_{j=0}^{k-1} \beta_{2,j}$$

5. Player B generates $2k$ shares of secret b using Shamir's $(k, 2k)$ method and computes the following.

$$[\beta_1 b]_0 = \beta_1 \times [b]_0, \dots, [\beta_1 b]_{k-1} = \beta_1 \times [b]_{k-1}$$

$$[\beta_2 b]_k = \beta_2 \times [b]_k, \dots, [\beta_2 b]_{2n-1} = \beta_2 \times [b]_{2n-1}$$

6. Player B sends $[\beta_1 b]_i, [\beta_2 b]_{i+k}, \beta_{1,i}, \beta_{2,i}$ to server S_i $(i = 0, 1, \dots, k - 1)$.

Multiplication Protocol:

1. Each server S_i $(i = 0, 1, \dots, k - 1)$ computes the following.

$$[\alpha_1\beta_1 ab]_i^* = [\alpha_1 a]_i \times [\beta_1 b]_i$$

$$[\alpha_2\beta_2 ab]_{i+k}^* = [\alpha_2 a]_{i+k} \times [\beta_2 b]_{i+k}$$

$$\alpha_{1,i}\beta_{1,i} = \alpha_{1,i} \times \beta_{1,i}$$

$$\alpha_{2,i}\beta_{2,i} = \alpha_{2,i} \times \beta_{2,i}$$

2. Each server S_i generates random number γ_i, computes the following and sends to one of the servers (here, we assume server S_0).

$$\frac{\gamma_i}{\alpha_{1,i}\beta_{1,i}}, \quad \frac{\gamma_i}{\alpha_{2,i}\beta_{2,i}}$$

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

$$\frac{\gamma}{\alpha_1\beta_1} = \prod_{i=0}^{k-1} \frac{\gamma_i}{\alpha_{1,i}\beta_{1,i}}$$

$$\frac{\gamma}{\alpha_2\beta_2} = \prod_{i=0}^{k-1} \frac{\gamma_i}{\alpha_{2,i}\beta_{2,i}}$$

4. Each server S_i computes $[\gamma ab]_i^*, [\gamma ab]_{i+k}^*$ as follows, and distribute $[\gamma ab]_i^*, [\gamma ab]_{i+k}^*$ using Shamir's (k, k) method to all servers S_i.

$$[\gamma ab]_i^* = \frac{\gamma}{\alpha_1\beta_1} \times [\alpha_1\beta_1 ab]_i^*$$

$$[\gamma ab]_{i+k}^* = \frac{\gamma}{\alpha_2\beta_2} \times [\alpha_2\beta_2 ab]_{i+k}^*$$

$$[\gamma ab]_i^* \Rightarrow \begin{cases} [\gamma ab]_{i,0} \Rightarrow send\ to\ S_0 \\ \vdots \\ [\gamma ab]_{i,k-1} \Rightarrow send\ to\ S_{k-1} \end{cases}$$

$$[\gamma ab]_{i+k}^* \Rightarrow \begin{cases} [\gamma ab]_{i+k,0} \Rightarrow send\ to\ S_0 \\ \vdots \\ [\gamma ab]_{i+k,k-1} \Rightarrow send\ to\ S_{k-1} \end{cases}$$

5. Each server S_i computes the following (λ_i are the recombination vector).

$$[\gamma ab]_i = \lambda_0 \times [\gamma ab]_{0,i} + \cdots + \lambda_{2k-1} \times [\gamma ab]_{2k-1,i}$$

Reconstruction Protocol:
1. The player collects $[\gamma ab]_i, \gamma_i$ from k servers S_i, reconstructs γab and computes γ as follows.

$$\gamma = \prod_{i=0}^{k-1} \gamma_i$$

2. Finally, the player reconstructs multiplication result ab as follows.

$$ab = \frac{\gamma ab}{\gamma}$$

5 SECURITY OF THE PROPOSED METHOD

In a *2-input-1-output* multiplication process, when the adversary has information of one of the inputs (e.g. input a) and output (e.g. output ab), the second input (e.g. input b) will be leaked. Therefore, we only consider the following adversaries. The attack is considered a success if the adversary can achieve the aim of learning the information that he/she wants to know. Therefore, we can state that our proposed method is secure if it is secure against the following adversaries.

Adversary 1: The adversary has information from $k-1$ servers. According to this information, the adversary attempts to know inputs a, b and output ab.

Adversary 2: One of the players who inputted a secret is the adversary. In addition, the adversary also has information from $k-1$ servers. According to this information, the adversary attempts to know the remaining one input a or b, and output ab.

Adversary 3: The player who reconstructed the output is the adversary. In addition, the adversary has information from $k-1$ servers. According to this information, the adversary attempts to know two inputs a and b.

In the following, we evaluate the security of our proposed method.

Evaluation of Security against Adversary 1:
Here, *Adversary 1* has information from $k-1$ number of servers. In the distribution protocol, *Adversary 1* has the following information D_A from Player A and D_B from Player B.

$$D_A = [\alpha_1 a]_l, [\alpha_2 a]_{l+k}, \alpha_{1,l}, \alpha_{2,l}\ (l = 0, \dots, k-2)$$

$$D_B = [\beta_1 b]_l, [\beta_2 b]_{l+k}, \beta_{1,l}, \beta_{2,l}\ (l = 0, \dots, k-2)$$

However, encrypted secrets $\alpha_1 a, \alpha_2 a, \beta_1 b, \beta_2 b$ are not leaked from $k-1$ shares. Moreover, *Adversary 1* is not able to learn about random numbers $\alpha_1, \alpha_2, \beta_1, \beta_2$ from $k-1$ servers. Therefore, even with this information, secrets a and b are not leaked. Thus, the following are true.

$$H(a) = H(a|D_A)$$

$$H(b) = H(b|D_B)$$

In Step 1 of the multiplication protocol, *Adversary 1* learns about $\alpha_{1,l}\beta_{1,l}, \alpha_{2,l}\beta_{2,l}\ (l = 0, \dots, k-2)$, in Step 2 about $\gamma_l, \gamma_l/\alpha_{1,l}\beta_{1,l}, \gamma_l/\alpha_{2,l}\beta_{2,l}$, in Step 3 about $\gamma/\alpha_1\beta_1, \gamma/\alpha_2\beta_2$, in Step 4 about $[\gamma ab]_0^*, \dots, [\gamma ab]_{k-2}^*, [\gamma ab]_k^*, \dots, [\gamma ab]_{2k-2}^*$ and in Step 5 about $[\gamma ab]_0, \dots, [\gamma ab]_{k-2}$. As a result, we can transform the problem into determining whether the adversary can learn about inputs a, b or output ab from the following information.

$$\alpha_{1,l}, \alpha_{2,l}, \beta_{1,l}, \beta_{2,l}, \gamma_l, \frac{\gamma}{\alpha_1\beta_1}, \frac{\gamma}{\alpha_2\beta_2},$$

$$[\gamma ab]_l^*, [\gamma ab]_{l+k}^*, [\gamma ab]_l\ (l = 0, \dots, k-2)$$

Since $[\gamma ab]_i^*$ is represented by polynomial of $(2k-2)$ degree, $2k-1$ number of shares are required to reconstruct γab. However, *Adversary 1* only has information of $2k-2$ number of shares, therefore, γab is not leaked. The same is true when *Adversary 1* only has information of $k-2$ number of shares $[\gamma ab]_l$, γab is not leaked. Moreover, because *Adversary 1* has no information $\alpha_1, \alpha_2, \beta_1, \beta_2$, random number γ used to encrypt the output ab is not leaked. Therefore, our proposed method is secure against *Adversary 1* and the following are true:

$$H(\gamma) = H\left(\gamma|\alpha_{1,l}, \alpha_{2,l}, \beta_{1,l}, \beta_{2,l}, \gamma_l, \frac{\gamma}{\alpha_1\beta_1}, \frac{\gamma}{\alpha_2\beta_2}\right)$$

$$H(\gamma ab) = H\left(\gamma ab | [\gamma ab]_l^*, [\gamma ab]_{l+k}^*, [\gamma ab]_l \ (l = 0, \dots, k-2)\right)$$

Evaluation of Security against Adversary 2:
Assume that the player who inputted input a is *Adversary 2*. *Adversary 2* also has information from $k-1$ servers. Therefore, in the distribution protocol, *Adversary 2* has information about $a, \alpha_{1,i}, \alpha_{2,i}, \alpha_1, \alpha_2$ $(i = 0, \dots, k-1)$ in addition to information from $k-1$ servers (*Adversary 1*).

Therefore, the evaluation of security against *Adversary 2* can be translated to the problem of determining whether the adversary can learn about the remaining input b and output ab from the following information:

$$a, \alpha_{1,i}, \alpha_{2,i}, \alpha_1, \alpha_2, \beta_{1,l}, \beta_{2,l}, \gamma_l, \frac{\gamma}{\beta_1}, \frac{\gamma}{\beta_2},$$

$$[\beta_1 b]_l, [\beta_2 b]_{l+k}, [\gamma ab]_l^*, [\gamma ab]_{l+k}^*, [\gamma ab]_l \ (l = 0, \dots, k-2)$$

To obtain information about secret b, the adversary must first obtain information of $\beta_1 b, \beta_2 b$ and random numbers β_1, β_2. The information that is related to random numbers β_1, β_2 are $\beta_{1,l}, \beta_{2,l}, \gamma_l, \gamma/\beta_1, \gamma/\beta_2$. However, even from this information, random numbers β_1, β_2 are not leaked. Moreover, encrypted secret $\beta_1 b, \beta_2 b$ will not be leaked from $k-1$ number of shares. Therefore,

$$H(\beta_1 b) = H(\beta_1 b | [\beta_1 b]_0, \dots, [\beta_1 b]_{k-2})$$

$$H(\beta_2 b) = H(\beta_2 b | [\beta_2 b]_0, \dots, [\beta_2 b]_{k-2})$$

$$H(\beta_1) = H\left(\beta_1 \middle| \beta_{1,l}, \gamma_l, \frac{\gamma}{\beta_1}\right)$$

$$H(\beta_2) = H\left(\beta_2 | \beta_{2,l}, \gamma_l, , \frac{\gamma}{\beta_2}\right)$$

Finally, to obtain multiplication result ab, the adversary must first obtain information γab and random number γ. However, from $k-1$ shares $[\gamma ab]_l$ and $2k-2$ shares of $[\gamma ab]_l^*, [\gamma ab]_{l+k}^*$, and random numbers $\gamma_l, \gamma/\beta_1, \gamma/\beta_2$, information of γab and random number γ are not leaked. Therefore,

$$H(\gamma ab) = H\left(\gamma ab | [\gamma ab]_l^*, [\gamma ab]_{l+k}^*, [\gamma ab]_l \ (l = 0, \dots, k-2)\right)$$

$$H(\gamma) = H\left(\gamma | \gamma_l, \frac{\gamma}{\beta_1}, \frac{\gamma}{\beta_2}\right)$$

In addition, the evaluation above remains valid even if the adversary is the player who inputted input b. Therefore, our proposed method is secure against *Adversary 2*.

Evaluation of Security against Adversary 3:
Assume that the player who reconstructed output ab is *Adversary 3*. *Adversary 3* also has information from $k-1$ servers. Therefore, in the reconstruction protocol, *Adversary 3* has information about $[\gamma ab]_i, \gamma_i, \gamma ab, \gamma$ $(i = 0, \dots, k-1)$ in addition to information from $k-1$ servers (*Adversary 1*).

Therefore, the evaluation of security against *Adversary 3* can be translated to the problem of determining whether the adversary can learn about the inputs a, b from the following information:

$$A = \{ab, \gamma_i, \gamma, \alpha_{1,l}, \alpha_{2,l}, \beta_{1,l}, \beta_{2,l}, \alpha_1\beta_1, \alpha_2\beta_2,$$

$$[\alpha_1 a]_l, [\alpha_2 a]_l, [\beta_1 b]_l, [\beta_2 b]_l \ (l = 0, \dots, k-2)\}$$

To obtain information about secret a, b, the adversary must first obtain information of $\alpha_1 a, \alpha_2 a, \beta_1 b, \beta_2 b$ and random numbers $\alpha_1, \alpha_2, \beta_1, \beta_2$. The information that is related to random numbers $\alpha_1, \alpha_2, \beta_1, \beta_2$ are $\alpha_{1,l}, \alpha_{2,l}, \beta_{1,l}, \beta_{2,l}, \alpha_1\beta_1, \alpha_2\beta_2$. However, even from these information, random numbers $\alpha_1, \alpha_2, \beta_1, \beta_2$ are not leaked. Moreover, encrypted secret $\alpha_1 a, \alpha_2 a, \beta_1 b, \beta_2 b$ will not be leaked from $k-1$ number of shares. Finally, even with the multiplication result ab, *Adversary 3* will not be able to learn about each secret a, b. Therefore,

$$H(a) = H(a|A)$$

$$H(b) = H(b|A)$$

Therefore, we can state that our proposed method is also secure against *Adversary 3*.

6 EVALUATION OF OUR PROPOSED METHOD

In this section, we perform evaluation of our proposed method in term of computation and communication costs. Below is the definition of parameters used throughout our analysis. Note that in secret sharing scheme, size of share d_1 is usually almost the same size as the original secret. Moreover, in a secret sharing scheme, the computational cost of the distribution and the reconstruction process differs, but for ease of understanding, we consider that the computation cost of both the distribution and reconstruction process of a secret sharing scheme to be the same. Table 1 shows the communication cost and number of rounds of our proposed method. Table 2 shows the computational cost of our method.

Definition of Parameters:

- d_1: Size of share from secret sharing scheme
- C_1: Computational cost of Shamir's $(k, 2k)$ secret sharing scheme
- C_2: Computational cost of Shamir's (k, k) secret sharing scheme
- M: Computational cost of multiplication
- D: Computational cost of division
- A: Computational cost of addition

Table 1: Communication and number of rounds of the proposed method.

Process	Communication		Rounds
Distribution of a	$4kd_1$		1
Distribution of b	$4kd_1$		
Multiplication of a and b	Step 2	$2kd_1$	3
	Step 3	$2Nd_1$	
	Step 4	$2Nkd_1$	
Reconstruction of ab	$2kd_1$		1

Table 2: Computational cost of the proposed method.

Process	Computation Cost	
Distribution of a, b	Step 1	$2(k-1)M$
	Step 2	$C_1 + (2n-1)M$
	Step 4	$2(k-1)M$
	Step 5	$C_1 + (2n-1)M$
Multiplication of a and b	Step 1	$4kM$
	Step 2	$2k(M+D)$
	Step 3	$2(k-1)M$
	Step 4	$2k(M+C_2)$
	Step 5	$2Nk(M+A)$
Reconstruction of ab	$C_2 + (k-1)M + D$	

7 COMPARISON WITH CONVENTIONAL METHODS

In this section, we perform comparison with conventional methods (Watanabe method (Watanabe et al., 2015) proposed by Watanabe et al. and the TUS method (Shingu et al., 2016) proposed by Shingu et al.) that also realize multiplication of secret sharing schemes using only $N = k$ servers.

First, the TUS method allows for multiplication in the setting of $N = n \geq k$ since multiplication is performed by multiplying scalar value with a share, therefore, allowing the result of multiplication to be restored by only k shares instead of the conventional $2k-1$ shares. However, the TUS method requires one precondition where the input does not include the value 0 to securely perform multiplication. In contrast, our method allows for any values (including 0) to be used since the encrypted secret is not reconstructed in the protocol.

Next, Watanabe method also allows for multiplication in the setting of $N \geq k$ and $n \geq 2k-1$; however, the number of shares required to reconstruct the result remain at $2k-1$ instead of k. On the other hand, our method allows for multiplication in the setting of $n \geq 2k-1$ and number of servers N to remain at k. Moreover, our protocol produces $k-1$ sharing of ab, therefore, we only need to collect k instead of $2k-1$ shares for reconstruction. All the comparisons discussed above are summarized in Table 3.

Table 3. Comparison with conventional methods (for multiplication).

	Proposed method	Watanabe method	TUS method
Parameter of n, k	$n \geq 2k-1$	$n \geq 2k-1$	$n \geq k$
Number of servers N	$N = k$	$N \geq k$	$N \geq k$
Number of shares for reconstruction	k	$2k-1$	k
Number of Precondition	0	0	1

Next, in Table 4, we show comparison with conventional methods. However, since the computation cost of secret sharing scheme C_1, C_2 are typically larger than local computation cost of M, D and A, we omit the cost of M, D and A when either C_1 or C_2 is present in the computation cost.

Table 4 shows that the computation cost for distribution of a, b and reconstruction of ab of our method are lower than both Watanabe and TUS methods. Next, since our proposed method includes the process of redistributing of local shares to all servers, we learnt that the computation cost of multiplication of our proposed method is larger than Watanabe method. However, we were able to reduce the computation cost for the reconstruction, and therefore, reducing the computation cost needed by the client.

Table 4: Comparison with conventional methods.

	Process	Proposed method	Watanabe method	TUS method
Computation	Distribution of a, b	$2C_1$	$2(C_1 + C_2)$	$2(k+1)C_2$
	Multiplication of ab	$2kC_2$	$2nM$	$(3k+1)C_2$
	Reconstruction of ab	C_2	$C_1 + 2C_2$	$(k+1)C_2$
Communication	Distribution of a, b	$8kd_1$	$8nd_1$	$2nd_1(k+1)$
	Multiplication of ab	$(2k + 2N + 2Nk)d_1$	0	$(k + n + 2k^2 + nk)d_1$
	Reconstruction of ab	$2kd_1$	$4kd_1$	$(k^2 + k)d_1$
Rounds	Distribution of a, b	1	1	1
	Multiplication of ab	3	0	4
	Reconstruction of ab	1	1	1

In terms of communication cost, the merits and demerits of each method depend on d_1, n, k. However, when comparing with Watanabe method, since our proposed method produce a polynomial of $(k-1)$ degree instead of polynomial of $(2k-2)$ degree, we were able to reduce the communication cost for reconstruction of ab by half. Finally, a comparison of each method's number of rounds, since our proposed method includes the process of redistributing and computation of random numbers, Table 4 shows that the number of rounds of our proposed method is considerably more than Watanabe method but lower than the TUS method.

8 CONCLUSIONS

In this paper, we proposed an improved method of multiplication of shares by using only $N = k$ number of servers. Furthermore, by implementing the use of *recombination vector,* we proposed a method of computing $k-1$ sharing of multiplication ab by using only k servers instead of the previous $2k-1$ servers. Through this proposed method, we realized secure multi-party computation of multiplication using Shamir's (k, n) method in the setting of $n \geq 2k-1, N = k$.

In a future study, we will focus on including the function for verification of shares in addition to allowing for different combination of computation (such as product-sum operation) to be performed simultaneously.

REFERENCES

Ben-Or M., Goldwasser S., Wigderson A., 1988. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In Proceedings of the 20th Annual ACM Symposium on Theory of Computing, pp. 1-10. ACM, New York, NY, USA.

Bendlin R., Damgård I., Orlandi C., Zakarias S., 2011. Semi-homomorphic encryption and multiparty computation. In Paterson K. G. (eds) Advances in Cryptology-EUROCRYPT 2011. LNCS, vol. 6632, pp. 169-188. Springer, Berlin, Heidelberg.

Brakerski Z., Vaikuntanathan V., 2011. Fully homomorphic encryption from ring-LWE and security for key dependent messages. In Rogaway P. (eds) Advances in Cryptology – CRYPTO 2011. LNCS, vol 6841, pp. 505-524. Springer, Berlin, Heidelberg.

Chaum D., Crépeau C., Damgård I., 1988. Multiparty unconditionally secure protocols. In Proceedings of the 20th Annual ACM Symposium on Theory of Computing. pp. 11-19. ACM, New York, NY, USA.

Cramer R., Damgård I., Maurer U., 2000. General secure multiparty computation from any linear secret sharing scheme. In Preneel B. (eds) Advances in Cryptology-EUROCRYPT 2000. LNCS, vol. 1807, pp. 316-334. Springer, Berlin, Heidelberg.

Cramer R., Damgård I., Nielsen J., 2015. Secure multiparty computation and secret sharing. Cambridge University Press, 1st edition.

Damgård I., Pastro V., Smart N., Zakarias S., 2012. Multiparty computation from somewhat homomorphic encryption. In Safavi-Naini R., Canetti R., (eds) Advances in Cryptology-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 Crampton J., Jajodia S., Mayes K. (eds) Computer Security – 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 Proceedings of the 17th Annual ACM Symposium on Principles of Distributed Computing. pp. 101-111. ACM, New York, NY, USA.

Gentry C., 2009. A fully homomorphic encryption scheme, Ph.D Thesis, Stanford University, Stanford, CA, USA.

Hirt M., 2001. Multiparty computation: efficient protocols, general adversaries, and voting. PhD Thesis, ETH Zurich. Reprint as ETH Series in Information Security and Cryptography vol. 3.

Shamir A., 1979. How to share a secret. Communications of the ACM, 22, (11), pp. 612-613.

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

Shingu T., Iwamura K., Kaneda K., 2016. Secrecy computation without changing polynomial degree in Shamir's (k, n) secret sharing scheme. In Proceedings of the 13th International Joint Conference on e-Business and Telecommunications, vol.1, pp. 89-94. SCITEPRESS.

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

Yao A. C., 1982. Protocols for Secure Computations. In 23rd Annual Symposium on Foundations of Computer Science. pp. 160-164. Chicago, IL, USA.

APPENDIX 1: ADDITION $(a + b)$

Protocol for computing addition of $a + b$ using our proposed method of distribution is as follows:

1. Each server $S_i\ (i = 0, 1, \dots, n-1)$ generates random number γ_i, computes the following and sends to one of the servers (here, we assume server S_0).

$$\frac{\gamma_i}{\alpha_{1,i}}, \quad \frac{\gamma_i}{\beta_{1,i}}$$

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

$$\frac{\gamma}{\alpha_1} = \prod_{i=0}^{k-1} \frac{\gamma_i}{\alpha_{1,i}}, \qquad \frac{\gamma}{\beta_1} = \prod_{i=0}^{k-1} \frac{\gamma_i}{\beta_{1,i}}$$

3. Each server S_i computes $[\gamma(a+b)]_i$as follows.

$$[\gamma(a+b)]_i = \frac{\gamma}{\alpha_1} \times [\alpha_1 a]_i + \frac{\gamma}{\beta_1} \times [\beta_1 b]_i$$

Security.
Here, due to the page limit, we had omitted the security proof against Adversaries 2 and 3. Below, we show the security against *Adversary 1*, where the adversary has information from $k-1$ servers. In the distribution protocol, *Adversary 1* has the following information D_A from Player A and D_B from Player B.

$$D_A = [\alpha_1 a]_l, [\alpha_2 a]_{l+k}, \alpha_{1,l}, \alpha_{2,l}\ (l = 0, \dots, k-2)$$

$$D_B = [\beta_1 b]_l, [\beta_2 b]_{l+k}, \beta_{1,l}, \beta_{2,l}\ (l = 0, \dots, k-2)$$

As shown in Section 5, *Adversary 1* will not able to learns neither a nor b from the information above. Moreover, in the protocol for addition, the adversary learns about the following.

$$\frac{\gamma}{\alpha_1}, \frac{\gamma}{\beta_1}, \gamma_l, \alpha_{1,l}, \beta_{1,l}, [\gamma(a+b)]_l\ (l = 0, \dots, k-2)$$

To learn the output $a + b$, *Adversary 1* has to obtain information $\gamma(a+b)$ and random number γ. However, from $k-1$ shares $[\gamma(a+b)]_l$ and random numbers γ_l, information of $\gamma(a+b)$ and γ are not leaked. Therefore, we can state that the addition protocol is secure against *Adversary 1*.

APPENDIX 2: SCALAR MULTIPLICATION (ca)

Protocol for computing scalar multiplication between constant c and secret a is as follows:

1. Let $c \in GF(p), c \neq 0$ be some constant. Each server $S_i\ (i = 0, 1, \dots, n-1)$ computes the following locally.

$$[\alpha_1(ca)]_i = c \times [\alpha_1 a]_i$$

Security. In the protocol for scalar multiplication between constant c and secret a, all computations are performed locally without any communication between players. Therefore, the security will depend on the distribution of secret a (which was proven to be secure in Section 5). Moreover, the adversary will not be able to learn the result of ca if no more than

$k-1$ shares of $[\alpha_1(ca)]_i$ are leaked. Therefore, we can state that our protocol for scalar multiplication of ca is also secure against semi-honest adversary (we omitted the detailed proof due to the page limit).

APPENDIX 3: EXAMPLE OF COMPUTATION

Below, for ease of understanding, we demonstrate the computation of multiplication between secrets $a = 3$ and $b = 2$ of Players A and B, respectively, under the setting of $N = k = 2, n \geq 3$. Since $k = 2$, multiplication of shares of a and b will produce a $(2k-2) = 2$ degree polynomial. As shown in Section 2.3, the process of reducing the degree of polynomial from $(2k-2) = 2$ to $(k-1) = 1$ can be achieved by using the *recombination vector* $r = (3, -3, 1)$. In the example shown below, secrets a, b, all random numbers and all computations are performed with $p = 97$.

Distribution Protocol:

1. Player A generates $2k = 4$ random numbers $\alpha_{1,0}, \alpha_{1,1}, \alpha_{2,0}, \alpha_{2,1}$ and computes the following.

$$\alpha_{1,0} = 2, \qquad \alpha_{1,1} = 4$$

$$\alpha_{2,0} = 3, \qquad \alpha_{2,1} = 6$$

$$\alpha_1 = \alpha_{1,0} \times \alpha_{1,1} = 2 \times 4 = 8\ (mod\ 97)$$

$$\alpha_2 = \alpha_{2,0} \times \alpha_{2,1} = 3 \times 6 = 18\ (mod\ 97)$$

2. Player A generates $2k = 4$ shares of secret $a = 3$ using Shamir's $(2, 4)$ method and computes the following. Here, let $[a]_i = 3 + x$.

$$[\alpha_1 a]_0 = 8 \times 4 = 32\ (mod\ 97)$$

$$[\alpha_1 a]_1 = 8 \times 5 = 40\ (mod\ 97)$$

$$[\alpha_2 a]_2 = 18 \times 6 = 11\ (mod\ 97)$$

$$[\alpha_2 a]_3 = 18 \times 7 = 29\ (mod\ 97)$$

3. Player A sends $[\alpha_1 a]_0, [\alpha_2 a]_2, \alpha_{1,0}, \alpha_{2,0}$ to server S_0 and $[\alpha_1 a]_1, [\alpha_2 a]_3, \alpha_{1,1}, \alpha_{2,1}$ to server S_1 .
4. Player B generates $2k = 4$ random numbers $\beta_{1,0}, \beta_{1,1}, \beta_{2,0}, \beta_{2,1}$ and computes the following.

$$\beta_{1,0} = 1, \qquad \beta_{1,1} = 6$$

$$\beta_{2,0} = 8, \qquad \beta_{2,1} = 2$$

$$\beta_1 = \beta_{1,0} \times \beta_{1,1} = 1 \times 6 = 6\ (mod\ 97)$$

$$\beta_2 = \beta_{2,0} \times \beta_{2,1} = 8 \times 2 = 16\ (mod\ 97)$$

5. Player B generates $2k = 4$ shares of secret $b = 2$ using Shamir's $(2, 4)$ method and computes the following. Here, let $[b]_i = 2 + 3x$.

$$[\beta_1 b]_0 = 6 \times 5 = 30\ (mod\ 97)$$

$$[\beta_1 b]_1 = 6 \times 8 = 48\ (mod\ 97)$$

$$[\beta_2 b]_2 = 16 \times 11 = 79\ (mod\ 97)$$

$$[\beta_2 b]_3 = 16 \times 14 = 30\ (mod\ 97)$$

6. Player B sends $[\beta_1 b]_0, [\beta_2 b]_2, \beta_{1,0}, \beta_{2,0}$ to server S_0 and $[\beta_1 b]_1, [\beta_2 b]_3, \beta_{1,1}, \beta_{2,1}$ to server S_1.
7. Finally, each server $S_i\ (i = 0, 1)$ holds the following.

– Server S_0 holds the following:

$$[\alpha_1 a]_0 = 32, [\alpha_2 a]_2 = 11, \alpha_{1,0} = 2, \alpha_{2,0} = 3$$

$$[\beta_1 b]_0 = 30, [\beta_2 b]_2 = 79, \beta_{1,0} = 1, \beta_{2,0} = 8$$

– Server S_1 holds the following:

$$[\alpha_1 a]_1 = 40, [\alpha_2 a]_3 = 29, \alpha_{1,1} = 4, \alpha_{2,1} = 6$$

$$[\beta_1 b]_1 = 48, [\beta_2 b]_3 = 30, \beta_{1,1} = 6, \beta_{2,1} = 2$$

Multiplication Protocol:

1. Each server $S_i\ (i = 0, 1)$ computes the following.

– Server S_0 computes the following:

$$[\alpha_1\beta_1 ab]_0^* = [\alpha_1 a]_0 \times [\beta_1 b]_0 = 32 \times 30 = 87\ (mod\ 97)$$

$$[\alpha_2\beta_2 ab]_2^* = [\alpha_2 a]_2 \times [\beta_2 b]_2 = 11 \times 79 = 93\ (mod\ 97)$$

$$\alpha_{1,0}\beta_{1,0} = \alpha_{1,0} \times \beta_{1,0} = 2 \times 1 = 2\ (mod\ 97)$$

$$\alpha_{2,0}\beta_{2,0} = \alpha_{2,0} \times \beta_{2,0} = 3 \times 8 = 24\ (mod\ 97)$$

– Server S_1 computes the following:

$$[\alpha_1\beta_1 ab]_1^* = [\alpha_1 a]_1 \times [\beta_1 b]_1 = 40 \times 48 = 77\ (mod\ 97)$$

$$[\alpha_2\beta_2 ab]_3^* = [\alpha_2 a]_3 \times [\beta_2 b]_3 = 29 \times 30 = 94\ (mod\ 97)$$

$$\alpha_{1,1}\beta_{1,1} = \alpha_{1,1} \times \beta_{1,1} = 4 \times 6 = 24\ (mod\ 97)$$

$$\alpha_{2,1}\beta_{2,1} = \alpha_{2,1} \times \beta_{2,1} = 6 \times 2 = 12\ (mod\ 97)$$

2. Each server S_i $(i = 0, 1)$ generates random number γ_i, computes the following and sends to one of the servers (here, we assume server S_0).

— Server S_0 generates $\gamma_0 = 4$, computes the following and sends to server S_0.

$$\frac{\gamma_0}{\alpha_{1,0}\beta_{1,0}} = \frac{4}{2} = 2\ (mod\ 97)$$

$$\frac{\gamma_0}{\alpha_{2,0}\beta_{2,0}} = \frac{4}{24} = 81\ (mod\ 97)$$

— Server S_1 generates $\gamma_1 = 2$, computes the following and sends to server S_0.

$$\frac{\gamma_1}{\alpha_{1,1}\beta_{1,1}} = \frac{2}{24} = 89\ (mod\ 97),$$

$$\frac{\gamma_1}{\alpha_{2,1}\beta_{2,1}} = \frac{2}{12} = 81\ (mod\ 97)$$

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

$$\frac{\gamma}{\alpha_1\beta_1} = \frac{\gamma_0}{\alpha_{1,0}\beta_{1,0}} \times \frac{\gamma_1}{\alpha_{1,1}\beta_{1,1}} = 2 \times 89 = 81\ (mod\ 97)$$

$$\frac{\gamma}{\alpha_2\beta_2} = \frac{\gamma_0}{\alpha_{2,0}\beta_{2,0}} \times \frac{\gamma_1}{\alpha_{2,1}\beta_{2,1}} = 81 \times 81 = 62\ (mod\ 97)$$

4. Each server S_i $(i = 0, 1)$ computes $[\gamma ab]_i^*, [\gamma ab]_{i+k}^*$ as follows, and distribute $[\gamma ab]_i^*, [\gamma ab]_{i+k}^*$ using Shamir's $(2, 2)$ method to all servers S_i.

— Server S_0 computes the following:

$$[\gamma ab]_0^* = \frac{\gamma}{\alpha_1\beta_1} \times [\alpha_1\beta_1 ab]_0^* = 81 \times 87 = 63\ (mod\ 97)$$

$$[\gamma ab]_2^* = \frac{\gamma}{\alpha_2\beta_2} \times [\alpha_2\beta_2 ab]_2^* = 62 \times 93 = 43\ (mod\ 97)$$

$$let\ the\ polynomial\ be\ [\gamma ab]_0^* = 63 + x$$

$$\begin{cases} [\gamma ab]_{0,0} = 64 \Rightarrow send\ to\ S_0 \\ [\gamma ab]_{0,1} = 65 \Rightarrow send\ to\ S_1 \end{cases}$$

$$let\ the\ polynomial\ be\ [\gamma ab]_2^* = 43 + 2x$$

$$\begin{cases} [\gamma ab]_{2,0} = 45 \Rightarrow send\ to\ S_0 \\ [\gamma ab]_{2,1} = 47 \Rightarrow send\ to\ S_1 \end{cases}$$

— Server S_1 computes the following:

$$[\gamma ab]_1^* = \frac{\gamma}{\alpha_1\beta_1} \times [\alpha_1\beta_1 ab]_1^* = 81 \times 77 = 29\ (mod\ 97)$$

$$[\gamma ab]_3^* = \frac{\gamma}{\alpha_2\beta_2} \times [\alpha_2\beta_2 ab]_3^* = 62 \times 94 = 8\ (mod\ 97)$$

$$let\ the\ polynomial\ be\ [\gamma ab]_1^* = 29 + x$$

$$\begin{cases} [\gamma ab]_{1,0} = 30 \Rightarrow send\ to\ S_0 \\ [\gamma ab]_{1,1} = 31 \Rightarrow send\ to\ S_1 \end{cases}$$

$$let\ the\ polynomial\ be\ [\gamma ab]_3^* = 8 + 3x$$

$$\begin{cases} [\gamma ab]_{3,0} = 11 \Rightarrow send\ to\ S_0 \\ [\gamma ab]_{3,1} = 14 \Rightarrow send\ to\ S_1 \end{cases}$$

5. Each server S_i $(i = 0, 1)$ computes the following using the *recombination vector* $r = (3, -3, 1, 0)$.

— Server S_0 computes the following:

$$[\gamma ab]_0 = 3 \times [\gamma ab]_{0,0} + (-3) \times [\gamma ab]_{1,0} + 1 \times [\gamma ab]_{2,0} + 0 \times [\gamma ab]_{3,0}$$

$$= 3 \times 64 - 3 \times 30 + 1 \times 45 + 0 \times 11 = 50\ (mod\ 97)$$

— Server S_1 computes the following:

$$[\gamma ab]_1 = 3 \times [\gamma ab]_{0,1} + (-3) \times [\gamma ab]_{1,1} + 1 \times [\gamma ab]_{2,1} + 0 \times [\gamma ab]_{3,1}$$

$$= 3 \times 65 - 3 \times 31 + 1 \times 47 + 0 \times 14 = 52\ (mod\ 97)$$

Reconstruction Protocol:

1. The player collects $[\gamma ab]_0 = 50, [\gamma ab]_1 = 52, \gamma_0 = 4, \gamma_1 = 2$ from $N = k = 2$ servers S_i $(i = 0, 1)$, reconstructs γab using Shamir's (2, 2) method and computes γ as follows.

$$\gamma ab = 48$$

$$\gamma = \gamma_0 \times \gamma_1 = 4 \times 2 = 8\ (mod\ 97)$$

2. Finally, the player reconstructs multiplication result ab as follows.

$$ab = \frac{\gamma ab}{\gamma} = \frac{48}{8} = 6\ (mod\ 97)$$